ویرایش ۲۳ - سال ۹۸

کتاب الکترونیکی

هوش مصنوعی

به نام خدا

مترجم: سهراب جلوه‌گر

مهندس کامپیوتر – گرایش نرم‌افزار رایانه

با استفاده از منبع‌های انگلیسی در اینترنت

ویرایش ۲۳ - سال ۹۸

کتاب الکترونیکی

هوش مصنوعی

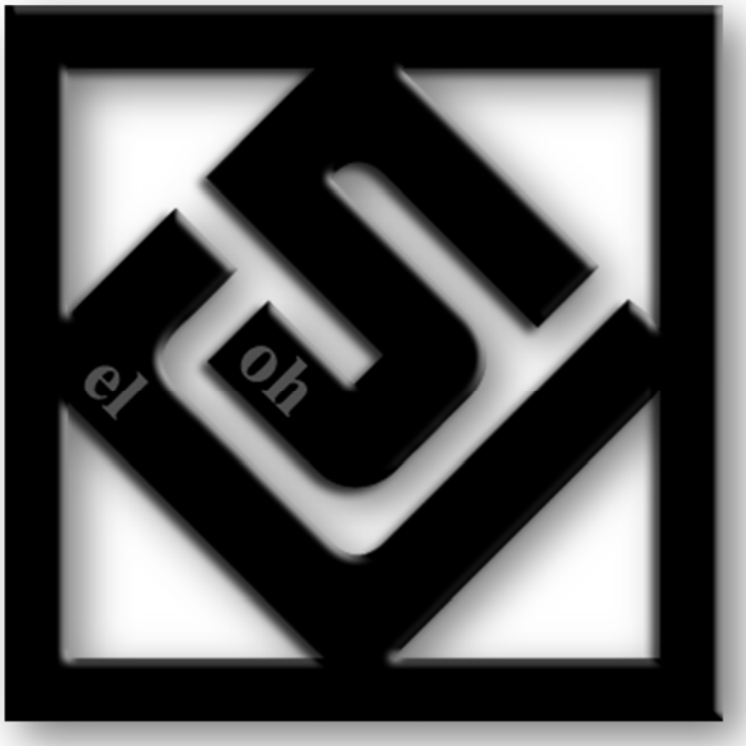
مترجم: سهراب جلوه‌گر

مهندس کامپیوتر – گرایش نرم‌افزار رایانه

با استفاده از منبع‌های انگلیسی در اینترنت


ویرایش ۲۳ - سال ۹۸
کتاب الکترونیکی
هوش مصنوعی
مترجم: سهراب جلوه‌گر
مهندس کامپیوتر – گرایش نرم‌افزار رایانه
با استفاده از منبع‌های انگلیسی در اینترنت
هوش مصنوعی (رایگان)
مترجم: سهراب جلوه‌گر
23th Edition - Year 2020
eBook
Artificial Intelligence
(In Persian) (Free)
Translated By:
Sohrab Jelvehgar
, Computer Software Engineer
Using:
English Sources On The Internet
sohjel


SohJel.blogfa.com
SohJel@yahoo.com


el
oh

نحوه‌ی آگاهی‌رسانی و ارتباط

برای آگاهی یافتن از پخش(انتشار) ویرایش‌های جدیدتر این کتاب الکترونیکی؛ بیان دیدگاه‌ها، پیشنهادها، انتقادهای سازنده‌ی خود و همچنین بیان موردهای نادرست احتمالی و غلط‌های نوشتاری می‌توانید برحسب مورد از وبلاگ‌ها و پست الکترونیکی‌های زیر استفاده نمایید:

الف) وبلاگ‌ها:

http://sohjel.blogfa.com

[وبلاگ اصلی و توصیّه شده][1]

http://sohjel.persianblog.ir

۱ - کلمه‌ی *sohjel* کوتاه شده‌ی نام اینجانب سهراب جلوه‌گر(*Soh*rab *Jel*vehgar) می‌باشد.

http://sohjel.loxblog.ir

ب) پست الکترونیکی‌ها:

sohjel@yahoo.com

[پست الکترونیکی اصلی و توصیّه شده]

sohjel@live.com

sohjelveh@gmail.com

(دو پست الکترونیکی قبلی را به ندرت برّرسی می‌کنم.)[۱]

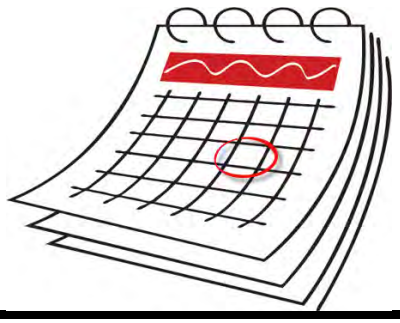
بااحترام، سهراب جلوه‌گر، کارشناس نرم‌افزار رایانه

ایران – شیراز – زمستان ۱۳۹۳ – زمستان ۱۳۹۸؛

۱ – معرّفی چند وبلاگ و چند پست الکترونیکی به دلیل این است که اگر به هر دلیلی مورد یا موردهایی برای همه یا اکثریّت قابل استفاده نبود(ند)، از مورد(های) دیگر بتوانید استفاده نمایید.

ویرایش «بیست و سوّم» (ویرایش سال ۱۳۹۸)

«ارائه‌ی اوّل» - ۹۸/۱۱/۹



برخی از قانون های استفاده از این کتاب الکترونیکی

* استفاده از مطلب های این کتاب الکترونیکی، در کتاب ها، مجلّه ها، پژوهش ها، پروژه های دانشجویی و غیره، با ذکر دقیق منبع، مجاز است.

برخی نکته‌ها و راهنمایی‌ها در مورد این کتاب الکترونیکی

این کتاب برای گروه‌های زیر قابل استفاده است:

الف) علاقه‌مندان به هوش مصنوعی

ب) دانشجویان دوره‌ی کارشناسی، که درس هوش مصنوعی را می‌گذرانند، ***به عنوان کتاب کمک درسی؛*** **مخصوصاً فصل‌هایی که عنوان(نام) آنها در قسمت «فهرست فصل‌ها» با علامت ستاره(*) مشخّص شده است.**

ج) داوطلبان آزمون‌های کارشناسی ارشد **مخصوصاً**

«سازمان سنجش آموزش کشور( )»

، که درس هوش مصنوعی جزو مواد امتحانی آزمون آنها است، ***به عنوان کتاب کمک درسی؛*** **مخصوصاً فصل‌هایی که عنوان(نام) آنها در قسمت «فهرست فصل‌ها» با علامت ستاره(*) مشخّص شده است.**

در این کتاب الکترونیکی عبارت‌هایی شبیه «Babylon[۱] > Britannica.com» به معنی این است که برای این مطلب از واژه‌نامه[۲]ی «Britannica.com» نرم‌افزار بَبیلُون استفاده شده است.

برخی از پیوند(لینک)های اینترنتی درون متن این کتاب، که به عنوان منبع مطلب آمده‌اند، ممکن است تغییر کرده باشند یا حذف شده باشند یا در موقع تبدیل کتاب به فایل پی. دی. اف. عوض شده باشند و در نتیجه باز نشوند.

جای برخی از پاورقی‌ها درست در همان صفحه نمی‌باشد؛ مثلاً پاورقی‌ای که در صفحه‌ی شماره‌ی x باید قرار بگیرد، ممکن است در صفحه‌ی x+1 قرار گرفته باشد.

در قسمت‌هایی از کتاب ممکن است از مترادف(هم‌معنی)نگاری، به عنوان مثال، به صورت زیر هم استفاده شده باشد:
«فقط *دریافت(گرفتن)* کلمات متوالی، در یک کامپیوتر کافی نمی‌باشد.»؛ در این عبارت دو کلمه‌ی «دریافت» و «گرفتن» مترادف(هم‌معنی) هستند؛
عبارت بالا را می‌توانید به یکی از این چهار صورت بخوانید: «فقط *دریافت یا گرفتن* کلمات متوالی، در یک کامپیوتر کافی نمی‌باشد.»؛ یا «فقط *دریافت و گرفتن* کلمات متوالی، در یک کامپیوتر کافی نمی‌باشد.»؛ یا «فقط *دریافت* کلمات متوالی، در یک کامپیوتر کافی نمی‌باشد.»؛ یا «فقط *گرفتن* کلمات متوالی، در یک کامپیوتر کافی نمی‌باشد.».

بهتر است مطالب کتاب را از ابتدا به انتها بخوانید.

برخی از مواردی که به‌وسیله‌ی اینجانب به متن کتاب اضافه شده‌اند، درون علامت قلّاب([ ]) قرار گرفته‌اند؛ مثلاً در متن زیر «[در این مورد باید این نکته را بگوییم که]» به وسیله‌ی من به متن کتاب اضافه شده است:
«برخی از پژوهشگران گفته‌اند که می‌خواهند افکار انسان را با استفاده از هوش مصنوعی، در کامپیوتر پیاده‌سازی نمایند؛ [در این مورد باید این نکته را بگوییم که] فکر انسان دارای موردهای زیادی می‌باشد و هر فردی به‌طورکامل نمی‌تواند همه‌ی آنها را شبیه‌سازی نماید.»

همیشه از آخرین نسخه‌ی موجود این کتاب الکترونیکی استفاده نمائید!

اوّل، مطلب‌های فصل را به دقّت خوانده و سپس به سراغ قسمت «یادآوری یا تکمیل» بروید!

---

۱ - بَبیلُون؛ نام نرم‌افزار «لغتنامه و مترجم» معروفی است؛ پایگاه اینترنتی این نرم‌افزار، http://www.babylon.com است و آرم این برنامه به صورت زیر است:

۲ - glossary

ممکن است برخی از تصویرها به صورت افقی، چرخانده شده باشند!؛ به عنوان مثال:


تصویر کتاب


تصویر اصلی

در این کتاب سعی شده است که مطالبی که در آزمون‌های سراسری کارشناسی ارشد

«سازمان سنجش آموزش کشور( )»

هم به طور مستقیم یا با اندکی تفاوت مورد سؤال قرار گرفته‌اند و در این کتاب وجود دارند، با علامت «☑» مشخّص شوند، هر چند که ممکن است تمام این مطالب مشخّص نشده باشند!. همچنین اینگونه مطالب که در آزمون‌های

«دانشگاه آزاد اسلامی( )»

مورد سؤال قرار گرفته‌اند، با علامت « » مشخّص شده‌اند.

فهرست فصل‌ها

فهرست برخی از منبع‌ها و مأخذها

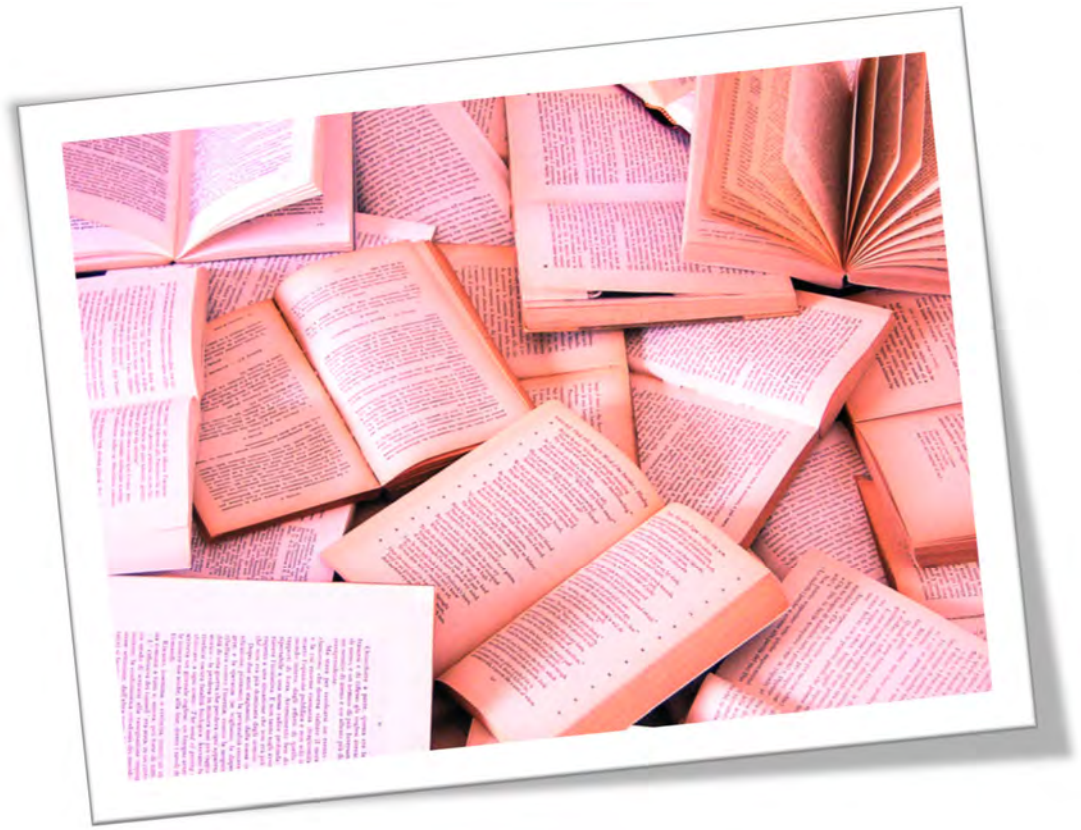
فصل اوّل

۱ آشنایی با هوش مصنوعی[۲]

۱ - تصویر، استاد، دکتر، جانْ مِکْکارتی(John McCarthy)، ۲۰۱۱ - ۱۹۲۷ میلادی، دانشمند بِنام(مشهور) علوم کامپیوتری و علم شناخت از کشور آمریکا، ملقّب به پدر هوش مصنوعی و پدر زبان برنامه‌نویسی لیسپ(Lisp) را در یک کنفرانس در سال ۲۰۰۶ میلادی نشان می‌دهد. (http://en.wikipedia.org/wiki/John_McCarthy_(computer_scientist)؛ ویکی‌پدیا، دایِرَتُ الْمَعارِف مشهور رایگان اینترنتی است. آرم این پایگاه اینترنتی را در زیر می‌بینید:

WIKIPEDIA
The Free Encyclopedia

۲ - Artificial Intelligence(*AI*)؛ از نام‌های دیگر برای هوش مصنوعی می‌توان «هوش ساختگی» و «هوش ماشینی» را نام برد.



فهرست برخی از عنوان‌های نوشته‌ها(رئوس مطالب)

فرزند ماشینی

توضیحی در مورد چند بازی

آیا برخی از افراد نمی‌گویند که «هوش مصنوعی ایده‌ی بدی است.»؟

هوش مصنوعی ضعیف

هوش مصنوعی قوی

آیا تئوری‌های «قابلیّت محاسبه» و «پیچیدگی محاسباتی» روش‌هایی کلیدی برای هوش مصنوعی نمی‌باشند؟

مسأله‌های NP-کامل

زندگی مصنوعی

برخی از شاخه‌های هوش مصنوعی

هوش مصنوعی منطقی

الگوشناسی(شناخت الگو)

استنتاج

برنامه‌ریزی

شناخت‌شناسی

هستی‌شناسی

ابتکارها(اکتشافات)

برخی از کاربردهای هوش مصنوعی

تئوری بازی‌ها

سخن‌شناسی

فهم زبان طبیعی

تصویر مَجازی، در کامپیوتر

سیستم‌های خِبره

ارتباط میان هوش مصنوعی و فلسفه

پیش‌نیازهای هوش مصنوعی

برخی از سازمان‌ها و مؤسّساتی که در زمینه‌ی هوش مصنوعی فعّالیّت می‌کنند

معرّفی چند کتاب خوب به زبان انگلیسی در زمینه‌ی هوش مصنوعی

توجّه:

بخشی از مطلب‌های این فصل ترجمه‌ی مطلب‌های استاد، جان مِک‌کارتی در مورد هوش مصنوعی، که در آدرس اینترنتی http://www-formal.stanford.edu/jmc/whatisai آمده‌اند، می‌باشد.

توجّه:

برخی از مطلب‌هایی که در این فصل بیان شده‌اند، در فصل‌های بعدی به صورت بیش‌تر مورد برّرسی قرار گرفته‌اند.

موردهای اساسی

هوش مصنوعی چیست؟

تعریف نخست: شاخه‌ای از علم که به شبیه‌سازی و پیاده‌سازی هوش انسان(بشر) در کامپیوتر می‌پردازد.[۱]

تعریف دوّم: ساخت ماشین‌هایی که کارهایی را انجام می‌دهند که آن کارها معمولاً با استفاده از هوش انسان انجام می‌شوند؛ مثل ترجمه‌ی یک زبان به زبان دیگر.[۲]

تعریف سوّم: دانش و مهندسی[۳] ساخت ماشین‌های هوشمند و مخصوصاً برنامه‌های کامپیوتری هوشمند می‌باشد.

هوش مصنوعی لازم نیست که خودش را به روش‌هایی که به صورت زیستی[۱]، قابل مشاهده‌اند، محدود نماید.

---

۱ - Babylon> English - English

۲ - Babylon> Concise Oxford English Dictionary

۳ - engineering؛ کاری که یک مهندس(engineer) انجام می‌دهد، مثل: طرّاحی، برنامه‌ریزی و غیره. (Babylon> Babylon English - English)

چرا هوش مصنوعی را مطالعه می‌کنیم؟

کنجکاوی؛ ساخت سیستم‌های هوشمند؛ انجام بعضی از کارها مثل بازی شطرنج که به نظر می‌رسد برای انجام آنها می‌توانیم از هوش مصنوعی کمک بگیریم؛ انجام کارهایی مثل خنثا کردن مین جنگی[۲]، یا تمیز کردن استخر شنا، که برای انسان، خطرناک یا کسل کننده هستند، از دلیل‌هایی هستند که ما هوش مصنوعی را مطالعه می‌نماییم.

هوش چیست؟

تعریف– هوش، توانایی به‌دست‌آوردن و به‌کار گرفتن دانش و مهارت‌ها می‌باشد.[۳]

انواع و درجه‌های هوش، در افراد، حیوان‌ها و برخی از ماشین‌ها گوناگون است.

---

۱ - biologically

۲ - mine: در زمینه‌ی نظامی معمولاً یک ابزار ثابت قابل انفجار است که برای نابودکردن افراد، کشتی‌ها یا وسیله‌های دشمن مورد استفاده قرار می‌گیرد. (Babylon> Britannica.com) در شکل زیر تصویری از یک نمونه مین ضدّ نفر –که در زمین کاشته شده است– را می‌بینید؛ با برخورد به شاخک مانندهایی که روی مین قرار دارند، مین منفجر خواهد شد:

۳ - Babylon> Concise Oxford English Dictionary

مقایسه‌ی هوش انسانی و هوش کامپیوتری


آرتور آر. جِنسِن

آرْتورْ آرْ. جِنْسِنْ[۱]، «یک فرضیّه‌ی ابتکاری که همه‌ی انسان‌های عادّی[۲]، دارای طرزکار(مکانیزم)های عقلانی شبیه هستند را پیشنهاد می‌کند»؛ نظر جنسن در مورد هوش انسانی درست است، ولی در موقعیّت فعلی، این نظر در مورد هوش مصنوعی، معکوس می‌باشد؛ برنامه‌های کامپیوتری، دارای سرعت و حافظه‌ی زیادی می‌باشند، امّا توانایی عقلانی آنها به توانایی عقلانی طرّاحان برنامه‌ها بستگی دارد. به احتمال قوی، سازماندهی طرزکارهای عقلانی، برای هوش مصنوعی می‌تواند با سازماندهی طرزکارهای عقلانی، برای افراد، متفاوت باشد. هرگاه افراد برخی از کارها را بهتر از کامپیوترها انجام می‌دهند و یا کامپیوترها تعداد زیادی محاسبه را برای انجام کار، به‌خوبی انسان‌ها انجام می‌دهند، این نشان می‌دهد که طرّاحان برنامه فهم طرزکارهای عقلانی لازم برای انجام کار را به طور مناسب نداشته‌اند.

پژوهش در زمینه‌ی هوش مصنوعی از چه زمانی شروع شد؟


آلِن تورینگ

بعد از جنگ جهانی دوّم[۳] تعدادی از افراد به صورت مستقل کار بر روی ماشین‌های هوشمند را شروع کردند؛ ریّاضیدان انگلیسی، آلِن تورینگ[۴] شاید اوّلین آنها باشد؛ وی یک سخنرانی را در مورد هوش مصنوعی در سال ۱۹۴۷ میلادی ارائه نمود؛ همچنین وی شاید اوّلین فردی باشد که گفت: «برنامه‌نویسی کامپیوترها، برای هوش مصنوعی، نسبت به ساخت ماشین‌ها بهتر می‌باشد.»؛ تا اواخر دهه‌ی ۱۹۵۰ میلادی تعداد زیادی پژوهشگر در زمینه‌ی هوش مصنوعی وجود داشت و بیش‌ترِ آنها اساس کار خود را برمبنای برنامه‌نویسی کامپیوترها گذاشته بودند.

---

۱ - Arthur R. Jensen، ۲۰۱۲ - ۱۹۲۳ میلادی، یک پژوهشگر خِبره(ماهر) در هوش انسانی بود.
(http://en.wikipedia.org/wiki/Arthur_R._Jensen)

۲ - normal

۳ - World War II یا World War 2 یا Second World War؛ ۱۹۴۵ – ۱۹۳۹ میلادی

۴ - Alan Turing، ۱۹۵۴ – ۱۹۱۲ میلادی، ریّاضیدان، منطقدان و دانشمند علوم کامپیوتری از کشور انگلیس بود.
(http://en.wikipedia.org/wiki/Alan_Turing)

آیا هدف هوش مصنوعی پیاده‌سازی افکار انسان در کامپیوتر است؟

برخی از پژوهشگران گفته‌اند که می‌خواهند افکار انسان را با استفاده از هوش مصنوعی، در کامپیوتر پیاده‌سازی نمایند؛ [در این مورد باید این نکته را بگوییم که] فکر انسان، دارای موردهای زیادی می‌باشد و هر فردی به‌طورکامل نمی‌تواند همه‌ی آنها را شبیه‌سازی نماید.

آیا هدف هوش مصنوعی در سطح هوش انسانی است؟

بله. نهایت تلاش این است که برنامه‌های کامپیوتری‌ای که می‌توانند مسأله‌ها را حل کنند و به اهداف دسترسی پیدا کنند را به‌خوبی انسان‌ها بسازند.

هوش مصنوعی تا رسیدن به سطح هوش انسانی چقدر فاصله دارد؟؛ چه هنگام این موضوع روی خواهد داد؟

تعداد اندکی از افراد فکر می‌کنند که سطح هوش انسانی با استفاده از زبان‌هایی که اکنون برای بیان دانش استفاده می‌شوند و با نوشتن برنامه‌های طولانی، که هم اکنون در حال نوشتن و سرهم‌بندی پایگاه‌های دانش[1] واقعیّت‌ها هستند، قابل دسترسی باشد.

بیش‌ترِ پژوهشگران هوش مصنوعی تصوّر می‌کنند که ایده‌های بنیادین جدید، مورد نیازند، و بنابراین، *قابل پیش‌بینی نیست که سطح هوش انسانی، قابل دسترسی باشد.*

آیا کامپیوترها ماشین مناسبی برای پیاده‌سازی هوش مصنوعی هستند؟

کامپیوترها برای شبیه‌سازی هر نوع از ماشین‌ها می‌توانند برنامه‌ریزی شوند.

---

۱ - Knowledge Base(*KB*)؛ **یادآوری**- پایگاه دانش، مجموعه‌ای از دانش است که با استفاده از زبانی قراردادی، برای بیان دانش، بیان شده است. (Babylon> FOLDOC)

آیا کامپیوترها برای هوشمند شدن، به اندازه‌ی کافی، سریع هستند؟

برخی از افراد فکر می‌کنند که کامپیوترهای به مراتب سریع‌تر و همچنین ایده‌های جدید لازم هستند. نظر شخصی استاد، جان مِک‌کارتی این بود که: «کامپیوترهای سی سال پیش هم در صورتی که بدانیم آنها را چگونه برنامه‌ریزی نماییم، به اندازه‌ی کافی سریع می‌باشند؛ البتّه به طور مجزّا از بلند پروازی‌های پژوهشگران هوش مصنوعی، کامپیوترها روند سریع‌تر شدن را دنبال می‌کنند.»

فرزند ماشینی[۱]

تعریف– فرزند ماشینی، برنامه‌ای کامپیوتری است که مانند یک بچّه‌ی هیجده ماهه[۲] می‌تواند با آموزش و با یادگیری، از راه تجربه، بهتر شود؛

شروع بحث آن از دهه‌ی ۱۹۴۰ میلادی بوده است؛ امّا برنامه‌های هوش مصنوعی هنوز به آن اندازه نرسیده‌اند که قادر باشند به اندازه‌ای که یک بچّه از تجربه‌ی محیط یاد می‌گیرد، یاد بگیرند.

توضیحی در مورد چند بازی[۳]

بازی شطرنج[۴]

بازی‌ای دو نفره است که در آن هر نفر شانزده مهره را با استفاده از قوانینی ثابت، در یک صفحه‌ی شطرنجی[۵] حرکت می‌دهد و تلاش می‌کند که مهره‌ی شاه[۶] حریف را کیش‌ومات[۷] نماید[۱]. برنامه‌های بازی شطرنج در حال حاضر در سطح

---

۱ – child machine

۲ – http://www.a-i.com/show_tree.asp?id=19&level=2&root=12 (این پایگاه اینترنتی یک پایگاه تحقیقاتی در زمینه‌ی هوش مصنوعی است.)

۳ – برای آگاهی‌های بیش‌تر به فصل «تئوری بازی‌ها»ی همین کتاب الکترونیکی مراجعه کنید.

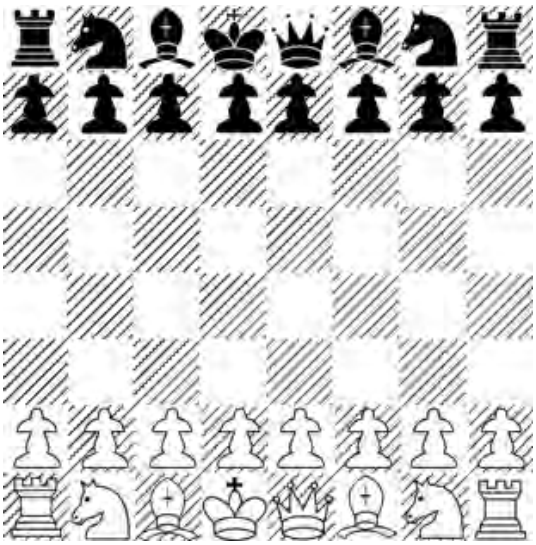
۴ – chess

۵ – checkerboard

۶ – King:

۷ – checkmate؛ در بازی شطرنج، شکست دادن(defeat) حریف با قراردادن مهره‌ی شاه حریف در یک وضعیّت گریزناپذیر(inescapable) است. (Babylon> Babylon English)

«استاد بزرگ»[۲] اجرا می‌شوند؛ امّا آنها این کار را با استفاده از طرزکارهای محدود شده در مقایسه با انسان و با انجام تعداد زیادی محاسبات، برای فهم راه انجام می‌دهند؛ ما این طرزکارها را بهتر می‌فهمیم و می‌توانیم برنامه‌های شطرنج را در سطح انسانی بسازیم، طوری که محاسبه‌های کم‌تری را نسبت به برنامه‌های فعلی انجام دهند. نمونه‌ای از صفحه‌ی این بازی را به همراه مهره‌های آن در شکل زیر می‌توانید ببینید:



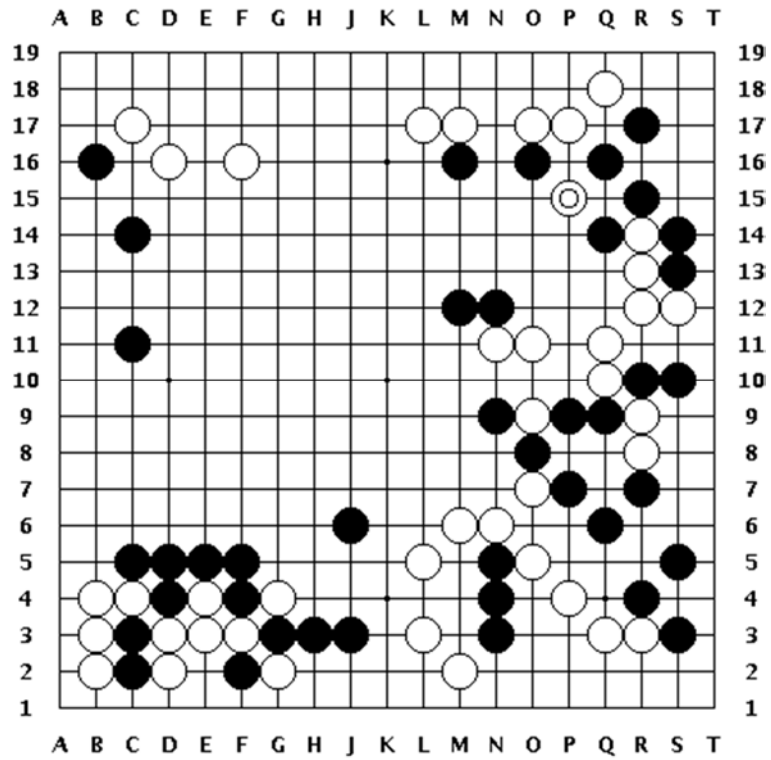
بازی Go

بازی دو نفره‌ی چینی و ژاپنی Go، بازی‌ای روی مقوّایی معمولاً دارای نوزده خطّ افقی و نوزده خطّ عمودی است که در آن معمولاً یکی از بازی‌کنندگان دارای مهره‌های سفید رنگ و یکی دیگر از بازی‌کنندگان دارای مهره‌های سیاه رنگ می‌باشد. در زیر تصویرهایی از صفحه‌ی این بازی را به همراه مهره‌های آن مشاهده می‌نمایید:

---

۱ - Babylon> Merriam-Webster

۲ - grandmaster: فردی است که در بازی شطرنج به طور غیرمعمول دارای مهارت باشد. (Babylon> Babylon English)

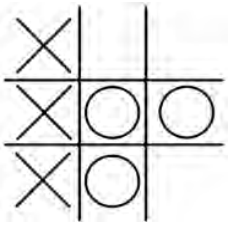


بازی تیک–تاک–تو[1]

بازی تیک–تاک–تو دارای دو بازیگر می‌باشد. برای هر دو بازیگر هدف این است که اوّلین فردی باشند که سه شیء همانند را در یک ردیف، ستون، یا قطر قرار می‌دهند. صفحه‌ی این بازی دارای یک شبکه‌ی سه در سه می‌باشد؛ بنابراین، دارای نه

۱ - tic tac toe یا Xها و Oها

خانه می‌باشد.[۱] در زیر تصویری از این بازی را می‌بینید؛ در این شکل چون فرد استفاده کننده از × سه‌تای آنها را در یک ستون قرار داده است، [بازی را] برده است:



۱ - http://www.seeingwithsound.com/tictactoe.htm

آیا برخی از افراد نمی‌گویند که «هوش مصنوعی ایده‌ی بدی است.»؟


جان سِرل


هیوبِرت دِرایفِس


جُزِف وایزِنباوم

فیلسوفی به نام جانْ سِرْلْ[۱] می‌گوید که: «ایده‌ی ماشینی غیرزنده(غیرزیستی، غیربیولوژیکی)[۲] که می‌تواند هوشمند شود، ایده‌ای نقض‌کننده می‌باشد.»؛ وی *«استدلال اتاق چینی»*[۳] را پیشنهاد می‌کند[۴]؛ فیلسوفی به نام هیوبِرْتْ دِرایْفِسْ[۵]، می‌گوید که: «هوش مصنوعی، غیرممکن می‌باشد.»؛ دانشمند علوم‌کامپیوتری‌ای به نام جُزِفْ وایْزِنْباوْمْ[۶]، می‌گوید که: «ایده‌ی هوش مصنوعی، ضدّانسانی و غیراخلاقی می‌باشد.»؛ برخی دیگر از افراد می‌گویند: «چونکه هوش مصنوعی تاکنون به اهداف خود نرسیده است، پس غیرممکن می‌باشد.»؛ سایر افراد از اینکه می‌بینند کمپانی‌های سرمایه‌گذاری‌کننده، ورشکست می‌شوند، ناامید می‌شوند.

---

۱ – John Searle، متولّد ۱۹۳۲ میلادی، فیلسوفی معروف از کشور آمریکا است.
(http://en.wikipedia.org/wiki/John_Searle)

۲ – non-biological

۳ – Chinese room argument؛ این مطلب در فصل «هوش مصنوعی» همین کتاب الکترونیکی توضیح داده شده است.

۴ – http://www-formal.stanford.edu/jmc/chinese.html

۵ – Hubert Dreyfus، متولّد ۱۹۲۹ میلادی، فیلسوفی آمریکایی است.
(http://en.wikipedia.org/wiki/Hubert_Dreyfus)

۶ – Joseph Weizenbaum، ۲۰۰۸ – ۱۹۲۳ میلادی، نویسنده و استاد بازنشسته‌ی(emeritus) آلمانی – آمریکایی علوم کامپیوتر در دانشگاه *MIT*(Massachusetts Institute of Technology؛ دانشکده‌ای فنّی و معروف است که در شهر کَمبریج(Cambridge)، در ایالت ماساچُست کشور ایالات متّحده‌ی آمریکا قرار دارد. (Babylon English Babylon>) بود.
(http://en.wikipedia.org/wiki/Joseph_Weizenbaum)

آشنایی با برخی از واژه‌ها

هوش مصنوعی ضعیف[۱]

تعریف- اوّلین هدف هوش مصنوعی(هوش مصنوعی ضعیف) ساختن چیزها(موجودیّت‌ها)ی هوشمند می‌باشد.

هوش مصنوعی ضعیف، برخلاف هوش مصنوعی قوی، قصد رسیدن به سطح هوش انسانی یا فراتر رفتن از سطح هوش انسانی را ندارد.[۲] به هوش مصنوعی ضعیف، هوش مصنوعی عملی(کاربردی)[۳] یا هوش مصنوعی محدود[۴] هم گفته می‌شود.[۵]

هوش مصنوعی قوی[۶]

تعریف نخست: *هوش مصنوعی‌ای است که به سطح هوش انسانی می‌رسد و یا از سطح هوش انسانی هم فراتر می‌رود.* هوش مصنوعی قوی می‌تواند درک نماید یا خودآگاه[۷] باشد؛ امّا ممکن است پردازش‌های فکری شبیه انسان را داشته باشد یا نداشته باشد.[۸]

تعریف دوّم: فهمیدن چیزها(موجودیّت‌ها)ی هوشمند و شاید حتّا *فهمیدن و مهندسی هوش انسان می‌باشد.*[۹]

تعریف سوّم: دیدی است که می‌گوید مغز انسان یک ابزار محاسباتی می‌باشد و کامپیوترها به‌طورکلّی قادرند که *فکر* کنند.[۱۰]

---

۱ - weak artificial intelligence

۲ - http://en.wikipedia.org/wiki/Weak_AI

۳ - applied artificial intelligence

۴ - narrow artificial intelligence

۵ - http://en.wikipedia.org/wiki/Strong_AI

۶ - strong artificial intelligence

۷ - self-aware

۸ - http://en.wikipedia.org/wiki/Strong_AI

۹ - استاد(پروفسور)، گرینْوالْدْ(Professor Greenwald)، استاد دانشگاه بِرُوْن(Brown) کشور ایالات متّحده‌ی آمریکا

۱۰ - www.ucd.ie/philosop/documents/2.%20definitions%20of%20some%20key%20terms.htm؛ این پایگاه اینترنتی متعلّق به دانشگاه شهر دوبلین کشور ایرلند است.


تعریف چهارم: به عنوان تعریفی دیگر، هوش مصنوعی قوی یک شکل فرضی از هوش مصنوعی است که می‌تواند به درستی استدلال نماید و مسأله‌ها را حل کند.

آیا تئوری‌های «قابلیّت محاسبه»[۱] و «پیچیدگی محاسباتی»[۲] روش‌هایی کلیدی برای هوش مصنوعی نمی‌باشند؟

(توجّه کنید که افراد عامّی و مبتدیّان علم کامپیوتر نمی‌توانند در این موردها اظهار نظر کنند، اینها کاملاً شاخه‌های منطقی ریّاضی و علم کامپیوتر می‌باشند و جواب این موردها باید تاحدودی تکنیکی باشد.)

---

۱ - computability theory؛ **یادآوری**- بحثی در علم نظری کامپیوتر می‌باشد و می‌گوید مسائل می‌توانند به وسیله‌ی هر کامپیوتری حل شوند. (Babylon> FOLDOC)

۲ - computational complexity؛ **یادآوری**- هزینه‌ی حلّ یک مسأله در محاسبات علمی گسترده است؛ که با استفاده از تعداد عملیّات لازم به علاوه‌ی مقدار حافظه‌ی استفاده شده برای مسأله و ترتیبی که مسأله حل می‌شود، اندازه‌گیری می‌شود. (Babylon> Britannica Concise Encyclopedia)


راجِر پِنرُز


اِستیون کوک


کِرت گادُل

نه؛ این تئوری‌ها مورد استفاده قرار می‌گیرند ولی مسأله‌های اساسی هوش مصنوعی را جوابگو نمی‌باشند. در دهه‌ی ۱۹۳۰ میلادی، منطقدانان ریّاضیّات و مخصوصاً کِرْتْ گادُلْ[۱] و آلن تورینگ ثابت کردند که الگوریتم‌هایی برای ضمانت اینکه همه‌ی مسأله‌ها، در دامنه‌های مهمّ ریّاضی، بتوانند حل شوند، وجود ندارند؛ یک معادله‌ی چندجمله‌ای دارای چند متغیّر، یک مثال می‌باشد؛ راجِرْ پِنْرُزْ[۲]، این مطلب را ادّعا می‌کند که انسان‌ها همیشه همه‌ی مسأله‌ها را در دامنه‌های مهمّ ریّاضی حل کرده‌اند، ولی کامپیوترها به طور ذاتی قادر به انجام کارهایی که افراد می‌توانند انجام دهند، نمی‌باشند. به هر حال افراد حلّ مسأله‌های دلخواه را در این موردها نمی‌توانند ضمانت کنند. در دهه‌ی ۱۹۶۰ میلادی، دانشمندان علوم کامپیوتر و مخصوصاً اِسْتیوِنْ کوکْ[۳] و ریچارْدْ کارْپْ[۴] تئوری مسأله‌های با دامنه‌ی *NP–کامل*[۵] را به وجود آوردند؛ انسان‌ها اغلب، مسأله‌های NP–کامل را در زمان‌هایی به مراتب کوتاه‌تر از آنچه که به وسیله‌ی الگوریتم‌های کلّی(عمومی) انجام می‌شوند، حل می‌کنند، امّا در حالت کلّی نمی‌توانند این مسأله‌ها را به سرعت حل کنند. اثبات اینکه برنامه‌ی انتخاب شده کوتاه‌ترین یا نزدیک به کوتاه‌ترین است یک مسأله‌ی غیر قابل حل می‌باشد.

---

۱ – Kurt Godel، ۱۹۷۸ – ۱۹۰۶ میلادی، ریّاضیدان و منطقدان آمریکایی متولّد کشور اُتریش بود.
(http://en.wikipedia.org/wiki/Kurt_G%C3%B6del)

۲ – Roger Penrose، متولّد ۱۹۳۱ میلادی، ریّاضیدان و فیزیکدانی از کشور انگلیس است.
(http://en.wikipedia.org/wiki/Roger_Penrose)

۳ – Stephen Cook، متولّد ۱۹۳۹ میلادی، دانشمند علوم کامپیوتری و ریّاضیدان معروف آمریکایی – کانادایی می‌باشد.
(http://en.wikipedia.org/wiki/Stephen_Cook)

۴ – Richard Karp، متولّد ۱۹۳۵ میلادی، دانشمند علوم کامپیوتری از کشور آمریکا است.
(http://en.wikipedia.org/wiki/Richard_Karp)

۵ – *NP*-complete(*NPC* = *N*ondeterministic *P*olynomial time *C*omplete)، این مطلب در ادامه‌ی همین فصل یادآوری شده است.

ریچارد کارپ

مسأله‌های NP–کامل

یادآوری- *یک نوع از مسأله‌های محاسباتی هستند که هیچ الگوریتم مناسبی برای حلّ آنها پیدا نشده است؛* مسأله‌ها در دامنه‌های NP-کامل، قابل حل می‌باشند، امّا به نظر می‌رسد که *زمان* برای حلّ این مسأله‌ها، با افزایش اندازه، به صورت نمایی(توان‌دار، به عنوان مثال، $۲^{۲۵}$) افزایش یابد. مسأله‌ی فروشنده‌ی دوره‌گرد[۱] مثالی از این نوع مسأله‌ها می‌باشد.[۲]

---

۱ - traveling salesman problem؛ **یادآوری**- یک مسأله‌ی بهینه‌سازی در تئوری گراف‌ها است که در آن، گره‌ها(شهرها)ی یک گراف به وسیله‌ی خط‌هایی مستقیم(یال یا لَبه) به هم وصل شده است و طول هر خط به فاصله‌ی دو شهر از هم بستگی دارد؛ به عبارت دیگر، هر چقدر طول خط بیش‌تر باشد، فاصله‌ی دو شهر بیش‌تر خواهد بود. حال مسأله این است که مسیری مناسب را طوری پیدا کنیم که از هر شهر یکبار بگذرد و در ضمن کوتاه‌ترین مسیر هم باشد. (Babylon> Britannica.com)

۲ - Babylon> Britannica.com

زندگی مصنوعی[۱]

تعریف– مطالعه‌ی سیستم‌های ترکیبی که به طریقی مانند سیستم‌های زنده‌ی طبیعی رفتار می‌کنند، می‌باشد.[۲]

زندگی مصنوعی دارای دورنماهای بسیار خوبی در زیست‌شناسی و علم کامپیوتر می‌باشد. این شاخه از علم در مورد کِشت مصنوعی؛ رفتارهای شبیه زندگی؛ نظیر: رویِش، سازگاری، تولید مِثل، اجتماعی شدن، یادگیری و حتّا مرگ تحقیق می‌نماید و محدود به چیزهای ابتدایی نمی‌باشد؛ دانشمندان زندگی مصنوعی می‌توانند از ترکیبی از اجزا و برنامه‌های کامپیوتری که در زمان صرفه‌جویی می‌کنند و دیگر فنّآوری(تکنولوژی)[۳]های شگفت انگیز، مانند آنهایی که برای تولید رفتارها جستجو می‌نمایند، استفاده نمایند.[۴]

---

۱ – ***AI***(بخوانید: *ایْ اِلْ*) یا ***AL***ife یا Artificial ***L***ife

۲ – Babylon> FOLDOC

۳ – technology: به کار بردن دانش برای هدف‌های عملی زندگی انسان یا برای تغییر و دستکاری محیط انسان می‌باشد. فنّآوری، شامل استفاده از مواد، ابزارها، روش(تکنیک)ها و منبع‌های انرژی برای راحت‌تر کردن زندگی یا خوشایندتر کردن آن و کار با سودمندی بیش‌تر است. (Babylon> Britannica Concise Encyclopedia)

۴ – http://www.aaai.org/AITopics/html/alife.html

برخی از شاخه‌های هوش مصنوعی

در زیر لیستی از شاخه‌های هوش مصنوعی آمده است، امّا به یقین برخی از شاخه‌ها در لیست زیر وجود ندارند، زیرا هنوز آنها ناشناخته‌اند.

هوش مصنوعی منطقی[۱]

تعریف– اینکه یک برنامه در حالت کلّی، واقعیّت‌های وضعیّت معیّنی که در آن باید عمل کند را می‌داند و اهدافی که به وسیله‌ی عبارات، گاهی با زبان منطقی ریّاضی ارائه می‌شوند را می‌داند.

جستجو[۲]

برنامه‌های هوش مصنوعی معمولاً تعداد زیادی از حالت‌ها را برّرسی می‌کنند؛ به عنوان مثال، حرکت‌های درون یک بازی شطرنج.

الگوشناسی(شناخت الگو)[۳]

تعریف نخست: در علم کامپیوتر، تشخیص داده‌های ورودی؛ مثل سخن، تصویرها و رشته‌های متنی، با شناخت و تشریح ویژگی‌ها و تشخیص ارتباط‌های میان آنها است.[۴]

تعریف دوّم: توانایی یک کامپیوتر برای پیدا کردن و جداکردن شکل‌های درون یک تصویر.[۵]

تعریف سوّم: شاخه‌ای از هوش مصنوعی است که به طبقه‌بندی یا توصیف مشاهده‌ها می‌پردازد.[۶]

به عنوان مثال، در شکل زیر صورت شخص با استفاده از نرم‌افزاری خاص تشخیص داده شده است[۱]:

---

۱ - logical AI

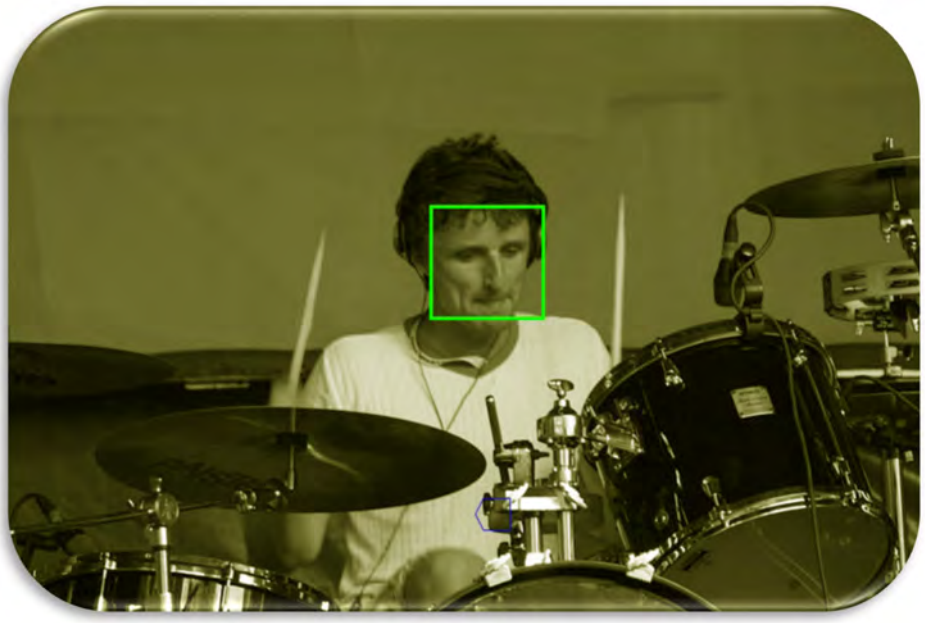
۲ - search

۳ - pattern recognition

۴ - Babylon> Britannica Concise Encyclopedia

۵ - Babylon> Babylon English-English

۶ - Babylon> FOLDOC



نمایش(ارائه، بازنمائی)[۲]

واقعیّت‌های یک محیط باید به طریقی نمایش داده شوند. در این مورد معمولاً از زبان‌های منطقی ریّاضی استفاده می‌شود.

استنتاج[۳]

تعریف– از برخی از واقعیّت‌ها دیگر واقعیّت‌ها می‌تواند به وجود آید.

استنتاج منطقی ریّاضی برای برخی از اهداف، کافی می‌باشد. متدهای جدید استنتاج غیر یکنواخت[۴] از دهه‌ی ۱۹۷۰ میلادی به منطق اضافه شده‌اند. ساده‌ترین نوع استدلال غیر یکنواخت، استدلال پیش‌ْفرض است که در آن، یک نتیجه‌گیری به صورت پیش‌فرض استنتاج(نتیجه‌گیری) می‌شود؛ امّا نتیجه‌گیری می‌تواند در صورتی که مدرک(دلیل) عوض شود، عوض بشود؛ برای مثال، زمانی که ما از یک پرنده حرف می‌زنیم، نتیجه می‌گیریم که می‌تواند پرواز کند؛ امّا این استنتاج زمانی که در مورد پنگوئن حرف می‌زنیم، می‌تواند عوض شود:

---

۱ – http://en.wikipedia.org/wiki/Pattern_recognition

۲ – representation

۳ – inference

۴ – non-monotonic inference

برنامه‌ریزی[۱]

برنامه‌ریزی تلاش می‌کند که عملیّات را برای رسیدن به هدف‌ها مرتّب نماید. کاربردهای برنامه‌ریزی شامل تدارکات[۲]، زمانبندی ساخت[۳] و برنامه‌ریزی ساخت مراحلی برای تولید محصول مطلوب می‌باشد. با یک برنامه‌ریزی بهتر می‌توان مقادیر زیادی در هزینه صرفه‌جویی نمود.

شناخت‌شناسی[۴]

تعریف نخست: یک شاخه از فلسفه[۵] است که در مورد منبع[۶]، طبیعت[۱]، روش‌ها و محدودیّت‌های دانش انسان بحث می‌کند.[۲]

---

۱ - planning

۲ - logistics

۳ - manufacturing scheduling

۴ - epistemology

۵ - philosophy؛ **یادآوری**- یک عقیده یا سیستم عقیده‌ها که به طور معتبر به وسیله‌ی برخی گروه‌ها یا دسته‌ها پذیرفته شده است. (Babylon> WordNet 2.0)

۶ - origin

تعریف دوّم: شناخت‌شناسی، مطالعه در مورد این است که ما چه می‌دانیم و چگونه می‌دانیم.[۳]

تعریف سوّم: مطالعه‌ای بر روی انواع دانش که برای حلّ مسأله‌ها در محیط(جهان) لازم می‌شوند، می‌باشد.

هستی‌شناسی[۴]

تعریف نخست: شاخه‌ای از مِتافیزیک[۵] است که در مورد طبیعت موجودات صحبت می‌کند.[۶]

تعریف دوّم: هستی‌شناسی، مطالعه‌ی انواع چیزهایی است که موجودند؛

در هوش مصنوعی، برنامه‌ها و عبارات، به انواع مختلفی از اشیا می‌پردازند و ما اینکه این انواع چیستند و ویژگی‌های اساسی آنها چیستند را مطالعه می‌کنیم. تأکید بر هستی‌شناسی از دهه‌ی ۱۹۹۰ میلادی شروع شد.

ابتکارها(اکتشافات)[۷]

برای افزایش احتمال حلّ برخی از مسأله‌ها به کار می‌روند[۸]؛ و روش‌هایی مبتنی بر آزمایش برای حلّ مسأله هستند. روش‌های مکاشفه‌ای برای بالا بردن سرعت پردازه‌ی پیداکردن یک راه حلّ به اندازه‌ی کافی خوب، در زمانی که جستجوی کامل نشدنی است، به کار می‌روند.[۹]

این واژه به صورت گوناگون در هوش مصنوعی به کار می‌رود. *توابع مکاشفه‌ای*[۱۰] در برخی از روش‌های جستجو برای اندازه‌گیری فاصله‌ی یک گره[۱۱]ی موجود در یک درخت جستجو[۱۲] تا هدف استفاده می‌شوند. *مستندات اکتشاف*[۱۳]، دو گره را در یک درخت جستجو، برای دیدن اینکه کدام بهتر از دیگری است، مقایسه می‌کند.

---

۱ - nature

۲ - Babylon> Learning, Performance and Training Definitions

۳ - Babylon> Learning, Performance and Training Definitions

۴ - ontology

۵ - metaphysics؛ **تعریف**- مطالعه‌ی فلسفی موجود زنده و باهوش (http://en.wikipedia.org/wiki/Metaphysics)

۶ - Babylon> Concise Oxford English Dictionary

۷ - heuristics

۸ - Babylon> WordNet 2.0

۹ - http://en.wikipedia.org/wiki/Heuristic

۱۰ - heuristic functions

۱۱ - node، این مورد در فصل «حلّ مسأله و جستجو»ی همین کتاب یادآوری شده است.

۱۲ - search tree، این مورد در فصل «حلّ مسأله و جستجو»ی همین کتاب توضیح داده شده است.

۱۳ - heuristic predicates

برخی از کاربردهای هوش مصنوعی

برخی از کاربردهای هوش مصنوعی در اینجا آمده است:

تئوری بازی‌ها[۱]

بازی‌ها موردهایی خوب برای تحقیق می‌باشند؛ زیرا بازی‌ها کوچک و جامع هستند و بنابراین، به آسانی برنامه‌ریزی می‌شوند. بازی‌ها مدل‌های خوبی از وضعیّت‌های رقابتی می‌توانند باشند، بنابراین، روش‌های طرّاحی شده برای تئوری بازی‌ها شاید بتوانند در مسأله‌های عملی هم به کار گرفته شوند.

سخن‌شناسی[۲]

تعریف– سخن‌شناسی، توانایی سیستم‌های کامپیوتری برای پذیرش سخن به صورت ورودی و کار بر روی آن یا تبدیل آن به صورت نوشتاری است.[۳]

در دهه‌ی ۱۹۹۰ میلادی، سخن‌شناسی کامپیوتری به سطحی کاربردی برای اهدافی محدود رسید؛ [به عنوان مثال،] خطوط هوایی کشور ایالات متّحده‌ی آمریکا صفحه‌کلیدی درختی را که برای اطّلاعات پرواز استفاده می‌شد، با سیستمی که از سخن‌شناسی استفاده می‌کرد، برای شماره‌های پرواز و نام شهرها جایگزین کردند؛ این روش، کاملاً مناسب بود.

در شکل زیر هم تصویری از برنامه‌ی محافظ صفحه‌ی نمایش[۴] کامپیوتر شرکت توشیبا[۵] که در آن، شِکلَک به سؤال‌های کاربر، مثل «Where are you?» پاسخ می‌دهد[۶] را می‌بینید:

---

۱ – game playing

۲ – speech recognition

۳ – Babylon> Britannica Concise Encyclopedia

۴ – screensaver

۵ – Toshiba

۶ – http://en.wikipedia.org/wiki/Speech_recognition

فهم زبان طبیعی[۱]

فهم زبان طبیعی یکی از سخت‌ترین مسأله‌ها در هوش مصنوعی است.[۲] فقط دریافت کلمات متوالی، در یک کامپیوتر کافی نمی‌باشد؛ فقط تجزیّه‌ی جملات هم کافی نمی‌باشد؛ *کامپیوتر باید بفهمد که متن در چه موردی می‌باشد؛* و این مورد به زودی برای دامنه‌های خیلی محدود، امکان‌پذیر می‌باشد.

تصویر مَجازی، در کامپیوتر[۳]

تعریف– موردی در روبوتیک[۴] است که در آن برنامه‌ها تلاش می‌کنند اشیائی که به صورت تصاویر دیجیتالی به وسیله‌ی دوربین‌های ویدیویی دریافت کرده‌اند را تشخیص دهند و در نتیجه روبوت‌ها بتوانند ببینند.

جهان از اشیائی سه بعدی تشکیل شده است؛ امّا ورودی‌ها برای چشم انسان و دوربین‌های تلویزیونی کامپیوتر، دو بعدی می‌باشند. برخی از برنامه‌ها فقط در حالت دو بعدی می‌توانند کار کنند؛ امّا تصویر مجازی کامپیوتری کامل، اطّلاعات سه بعدی ناتمام، که فقط مجموعه‌ای از دیدهای دو بعدی نیستند را لازم دارد. در حال حاضر فقط راه‌های محدودی برای نمایش اطّلاعات سه بعدی، به صورت مستقیم وجود دارد و این راه‌ها از قرار معلوم به‌خوبی راه‌هایی که انسان‌ها استفاده می‌کنند، نمی‌باشند.

---

۱ – understanding natural language؛ **توضیح**– زبان طبیعی(natural language، زبان انسان‌ها است که با گذشت سال‌ها گسترش یافته است. (Babylon> Babylon English))، متضاد «کد کامپیوتری(computer code)» یا «زبان مصنوعی(artificial language)» می‌باشد. (Babylon> Concise Oxford English Dictionary)

۲ – Babylon> FOLDOC

۳ – computer vision

۴ – robotic، این مطلب در فصل «آشنایی با روبوتیک» همین کتاب الکترونیکی توضیح داده شده است.

سیستم‌های خِبره[۱]

تعریف- سیستم‌های خبره تلاش می‌کنند که دانش یک انسان خبره(ماهر) را بگیرند و آن را در یک سیستم کامپیوتری پیاده‌سازی نمایند.

از سیستم‌های خبره انتظار می‌رود که بتوانند کارهایی که به یک فرد خبره نیاز دارند را انجام دهند، مثل: پزشکی، زمین‌شناسی، و مشورت در سرمایه گذاری. برخی از سیستم‌های خبره، از موفّق‌ترین کاربردهای هوش مصنوعی بوده‌اند، چونکه این برنامه‌ها باید در دنیای واقعی کار کنند و با برخی از مشکلات مهمّ موجود در هوش مصنوعی، مثل: کمی اطّلاعات ورودی مناسب؛ و استدلال، بر پایه‌ی احتمال؛ مواجه شده‌اند؛ یکی از اوّلین سیستم‌های خبره، در سال ۱۹۷۴ میلادی، به نام MYCIN بود که عفونت(آلودگی)[۲]های باکتریایی[۳] موجود در خون را تشخیص می‌داد و معالجات آن را پیشنهاد می‌کرد. این دستگاه این کار را بهتر از دانشجویان پزشکی یا دکترها انجام می‌داد. هستی‌شناسی آن، شامل باکتری، نشان(علامت)ها و معالجات بود.
از زمانی که خبرگان با مهندسان همکاری کردند، [مهندسان] چیزهایی را در مورد بیماران، دکترها، سلامتی، بهبود و... دانستند و واضح است که دانش مهندسان، تحت تأثیر آنچه که خبرگان به آنها می‌گفتند، در یک چارچوب کاری معیّن قرار داشت؛ در وضعیّت فعلی هوش مصنوعی این مطلب باید درست باشد.

ارتباط میان هوش مصنوعی و فلسفه

هوش مصنوعی دارای ارتباط‌های زیادی با فلسفه می‌باشد، مخصوصاً با فلسفه‌ی تحلیلی[۴] مدرن ارتباط زیادی دارد؛ [هوش مصنوعی و فلسفه] هر دو فکر و هوش عادّی[۵] را مطالعه می‌کنند.

---

۱ - expert systems

۲ - infection

۳ - bacterial؛ باکتری(bacteria)، موجود زنده‌ای ذرّه‌بینی(میکروسکوپی) و تک سلّولی است که برخی از آن‌ها سبب(باعث) بیماری می‌شوند و برخی از آن‌ها مفیدند. (Babylon> Babylon English و Babylon> Environmental Engineering (English ver.)

۴ - analytic philosophy؛ **تعریف**- عقیده‌ای فلسفی که بر تحلیل منطقی فکرها و مطالعه‌ی زبانی که با آن بیان می‌شوند، تأکید می‌کند. (Babylon> Britannica Concise Encyclopedia)

۵ - یا عقل سلیم؛ یا قضاوت صحیح(common sense = normal intelligence)

پیش‌نیازهای هوش مصنوعی

باید ریّاضیّات و مخصوصاً منطق ریّاضیّات[۱] را مطالعه کنیم. برای نزدیکی زیستی به هوش مصنوعی، روان‌شناسی[۲] و فیزیولوژی[۳] سیستم‌های عصبی را مطالعه کنیم. تعدادی زبان برنامه‌نویسی و در کم‌ترین حالت، زبان‌های برنامه‌نویسی سی[۴]، لیسپ[۵] و پرولوگ[۶] را باید بلد باشیم. همچنین فکر خوبی است که یکی از زبان‌های پایه‌ای ماشین[؛ مثل زبان برنامه نویسی اَسِمْبْلی[۷]] را یاد بگیریم. کارها بیش‌تر با زبان‌هایی که مد(متداول) هستند، انجام می‌شوند؛ در اواخر دهه‌ی ۱۹۹۰ میلادی، این زبان‌ها شامل C++[۸] و جاوا[۹] بود.

---

۱ - mathematical logic

۲ - psychology(روان‌شناسی)، مطالعه‌ی رفتار انسان، مطالعه‌ی پردازه‌های روحی(mental) و احساسی(emotional) است (Babylon> Babylon English) و به عنوان تعریفی دیگر، علمی است که پردازه‌های روحی و رفتار را در انسان‌ها و حیوان‌ها مطالعه می‌کند. (Babylon> Britannica.com)

۳ - physiology؛ **تعریف**- مطالعه‌ی عملکردها و فعّالیّت بدن (Babylon> Babylon English)

۴ - C؛ یک زبان برنامه‌نویسی سطح بالا(high-level language)ست (Babylon> Babylon English)؛ یعنی، دستورات آن به زبان انگلیسی نزدیک است.

۵ - *Lisp*؛ **توضیح**- از کلمه‌ی «*LIS*t *P*rocessing»(پردازش لیست) به وجود آمده است؛ زبانی برای پردازش لیست‌ها و پردازش متن می‌باشد و به طور گسترده‌ای در برنامه‌نویسی هوش مصنوعی از آن استفاده می‌شود. (تعدادی از لغت‌نامه‌های نرم‌افزار بیلون(Babylon))

۶ - Prolog؛ **توضیح**- نام یک زبان برنامه‌نویسی است که در هوش مصنوعی از آن استفاده می‌شود؛ برای آگاهی‌های بیش‌تر به فصل «آشنایی با زبان برنامه‌نویسی پرولوگ» همین کتاب مراجعه نمایید.

۷ - Assembly Language؛ نام یک زبان برنامه‌نویسی سطح پایین(low-level language) است؛ یعنی، دستورات آن به زبان ماشین(machine language؛ زبان برنامه‌نویسی‌ای است که به صورت کد دودویی(باینری، binary) نوشته می‌شود و می‌تواند به وسیله‌ی کامپیوتر بدون اینکه ترجمه شود، اجرا شود. (Babylon> Babylon English)) نزدیک است.

۸ - زبان برنامه‌نویسی C است که توانایی‌های برنامه‌نویسی شیءگرا(*O*bject *O*riented *P*rogramming(*OOP*)) به آن اضافه شده است. (Babylon> Britannica Concise Encyclopedia)

۹ - Java؛ یک زبان برنامه‌نویسی همه منظوره‌ی کامپیوتری است که می‌تواند روی هر کامپیوتری و در هر مرورگر(browser؛ مثل اینترنت اکسپلورر(Internet Explorer) و فایر فاکس(Firefox)) وبی اجرا شود. (Babylon> Raynet Business & Marketing Glossary) آرم این زبان به صورت زیر است:

برخی از سازمان‌ها و مؤسّساتی که در زمینه‌ی هوش مصنوعی فعّالیّت می‌کنند:

انجمن آمریکایی هوش مصنوعی[۱]؛ کمیته‌ی هماهنگ کننده‌ی اروپا برای هوش مصنوعی[۲]؛ و جامعه‌ی هوش مصنوعی و شبیه‌سازی رفتار[۳]؛ جوامع علمی علاقه‌مند به هوش مصنوعی می‌باشند.

شرکت ماشین‌آلات محاسبه کننده[۴] دارای یک گروه جالب ویژه به نام SIGART[۵] در زمینه‌ی هوش مصنوعی می‌باشد.

کنفرانس بین المللی هوش مصنوعی[۶]، کنفرانس بین المللی اصلی است. انجمن آمریکایی هوش مصنوعی(AAAI)، کنفرانسی ملّی را در کشور آمریکا اجرا می‌کند.

روزنامه‌های تعاملات الکترونیک با هوش مصنوعی[۷]؛ پژوهش‌های هوش مصنوعی[۸]؛ و تعاملات IEEE[۹] با تحلیل الگو و ماشین‌های هوشمند[۱۰]؛ سه روزنامه‌ی مهمّ پخش کننده‌ی مقالات هوش مصنوعی می‌باشند. در حال حاضر چیز دیگری که مناسب درج در این پاراگراف باشد را پیدا نکرده‌ایم.

---

۱ - The *A*merican *A*ssociation for *A*rtificial *I*ntelligence(*AAAI*)، با آدرس اینترنتی
http://www.aaai.org

۲ - *E*uropean *C*oordinating *C*ommittee for *A*rtificial *I*ntelligence(*ECCAI*)، با آدرس اینترنتی
http://eccai.org

۳ - Society for *A*rtificial *I*ntelligence and *S*imulation of *B*ehavior(*AISB*)، با آدرس اینترنتی
http://www.cogs.susx.ac.uk/aisb

۴ - *A*ssociation *for C*omputing *M*achinery(*ACM*)

۵ - با آدرس اینترنتی http://www.acm.org/sigart

۶ - The *I*nternational *J*oint *C*onference on *AI*(*IJCAI*)، با آدرس اینترنتی http://www.ijcai.org

۷ - Electronic Transactions on Artificial Intelligence، با آدرس اینترنتی
http://www.ida.liu.se/ext/etai

۸ - Journal of Artificial Intelligence Research، با آدرس اینترنتی http://www.jair.org

۹ - *IEEE* (بخوانید، *آیْ تریپِلْ ئیْ*)؛ کوتاه شده‌ی «*I*nstitute of *E*lectrical and *E*lectronics *E*ngineers»، به معنی «انجمن مهندسان برق و الکترونیک» می‌باشد؛ این سازمان تعداد زیادی استاندارد پذیرفته شده در زمینه‌ی مخابرات(communication) دارد. (Babylon> A Glossary of Internet & PC Terminology و Babylon> Babylon English)

۱۰ - IEEE Transactions on Pattern Analysis and Machine Intelligence، با آدرس اینترنتی
http://computer.org/tpami

معرّفی چند کتاب خوب به زبان انگلیسی در زمینه‌ی هوش مصنوعی


اِستوارت جِی. راسِل


پیتِر نورویگ

کتاب *هوش مصنوعی اِستوارْتْ جِیْ. راسِلْ[1] و پیتِرْ نُورْوِیگْ[2]*؛ از این کتاب استفاده‌ی زیادی می‌شود. تصویری از جلدهای ویرایش‌های سوّم، دوّم و اوّل این کتاب در زیر نشان داده شده است:

---

۱ – Stuart J. Russell، متولّد ۱۹۶۲ میلادی، دارای ملّیّت انگلیسی، یکی از دانشمندان علوم کامپیوتری است که به خاطر فعّالیّت‌هایی در زمینه‌ی هوش مصنوعی معروف است؛ وی به همراه پیتر نورویگ یکی از نویسندگان کتاب

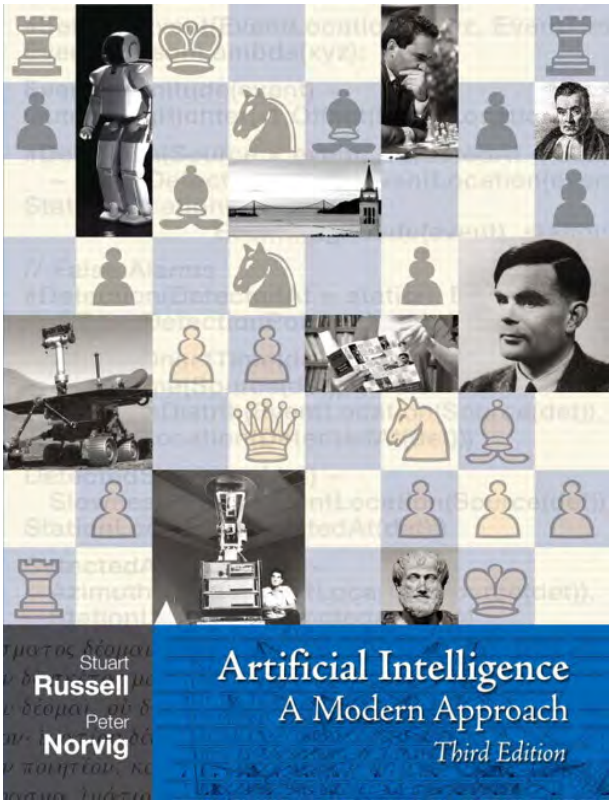
«هوش مصنوعی: نگرش پیشرفته(Artificial Intelligence: A Modern Approach)»

می‌باشد. (http://en.wikipedia.org/wiki/Stuart_J._Russell)

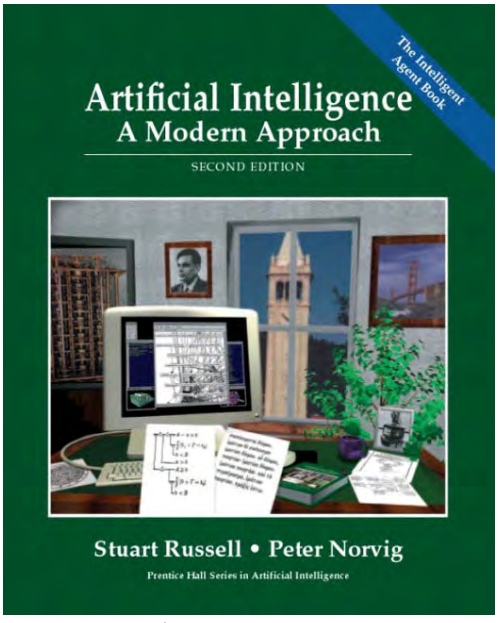
۲ – Peter Norvig، متولّد ۱۹۵۶ میلادی، دانشمند علوم کامپیوتری از کشور آمریکا است و به همراه استوارت راسل یکی دیگر از نویسندگان کتاب

«هوش مصنوعی: نگرش پیشرفته(Artificial Intelligence: A Modern Approach)»

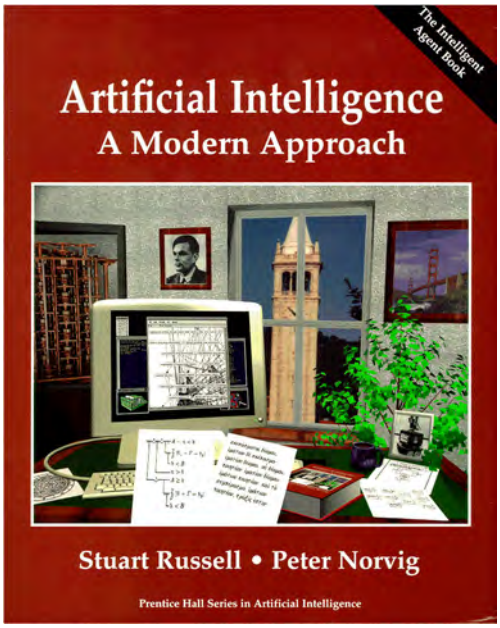
می‌باشد. (http://en.wikipedia.org/wiki/Peter_Norvig و http://norvig.com/bio.html، پایگاه اینترنتی اخیر، حاوی مقاله‌ها، گزارش‌ها، نرم‌افزارها و موارد دیگری است که به وسیله‌ی پیتر نورویگ گردآوری شده است.)



شکل بالا– جلد ویرایش سوّم کتاب

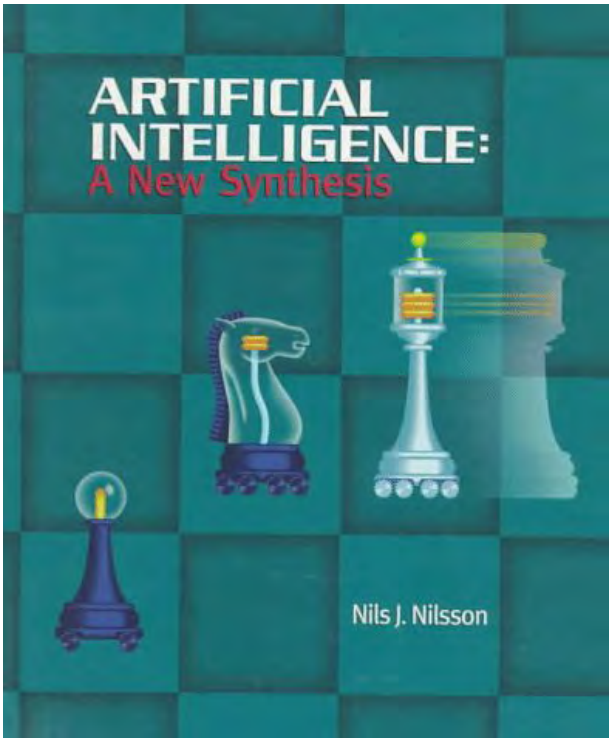


شکل بالا– جلد ویرایش دوّم کتاب



شکل بالا– جلد ویرایش اوّل کتاب

کتاب «هوش مصنوعی: تفسیر جدید»[۱] نوشته‌ی نیلْز نیلْسُونْ[۲] شاید برای خواندن آسان‌تر باشد؛ تصویری از جلد این کتاب را در زیر می‌بینید:



برخی از افراد کتاب «هوش محاسباتی»[۳] دِیویدْ پولْ[۴]، آلِنْ مَکْوُرْثْ[۱] و رَنْدیْ گُبُلْ[۲] را ترجیح می‌دهند. تصویری از جلد این کتاب را در زیر می‌بینید:

---

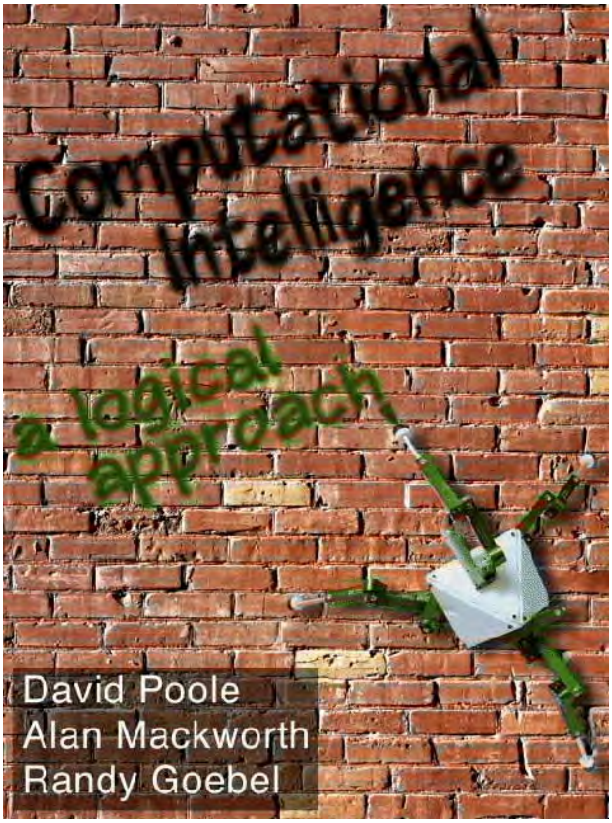
۱ – Artificial Intelligence: A New Synthesis

۲ – Nils John Nilsson؛ پژوهشگری بزرگ در زمینه‌ی هوش مصنوعی و استاد بازنشسته‌ی علوم کامپیوتر دانشگاه اِستَنفُورد کشور آمریکا است. تصویری از او را در زیر می‌بینید:

(http://ai.stanford.edu/~nilsson و http://en.wikipedia.org/wiki/Nils_Nilsson_(researcher)

۳ – Computational Intelligence

۴ – David Poole؛ استاد دانشکده‌ی علوم کامپیوتر دانشگاه بِریتیشْ کُلُمبیا(British Columbia)ی شهر وَنْکُوِر(Vancouver) کشور کانادا است. در زیر تصویری از او را می‌بینید:



(http://people.cs.ubc.ca/~poole)

۱ – Alan Mackworth؛ استاد دانشکده‌ی علوم کامپیوتر دانشگاه بریتیش کلمبیا(British Columbia)ی شهر ونکور(Vancouver) کشور کانادا است. تصویری از او را در زیر می‌بینید:

(http://people.cs.ubc.ca/~mack)

۲ – Randy Goebel؛ استاد دانشگاه آلبِرْتای شهر اِدْمُنْتُن ایالت آلبرتای کشور کانادا است. تصویری از او را در زیر می‌بینید:

(http://webdocs.cs.ualberta.ca/~goebel)

چکیده(خلاصه)ی مطلب‌های فصل اوّل

هوش مصنوعی، دانش و مهندسی ساخت ماشین‌های هوشمند و مخصوصاً برنامه‌های کامپیوتری هوشمند می‌باشد.

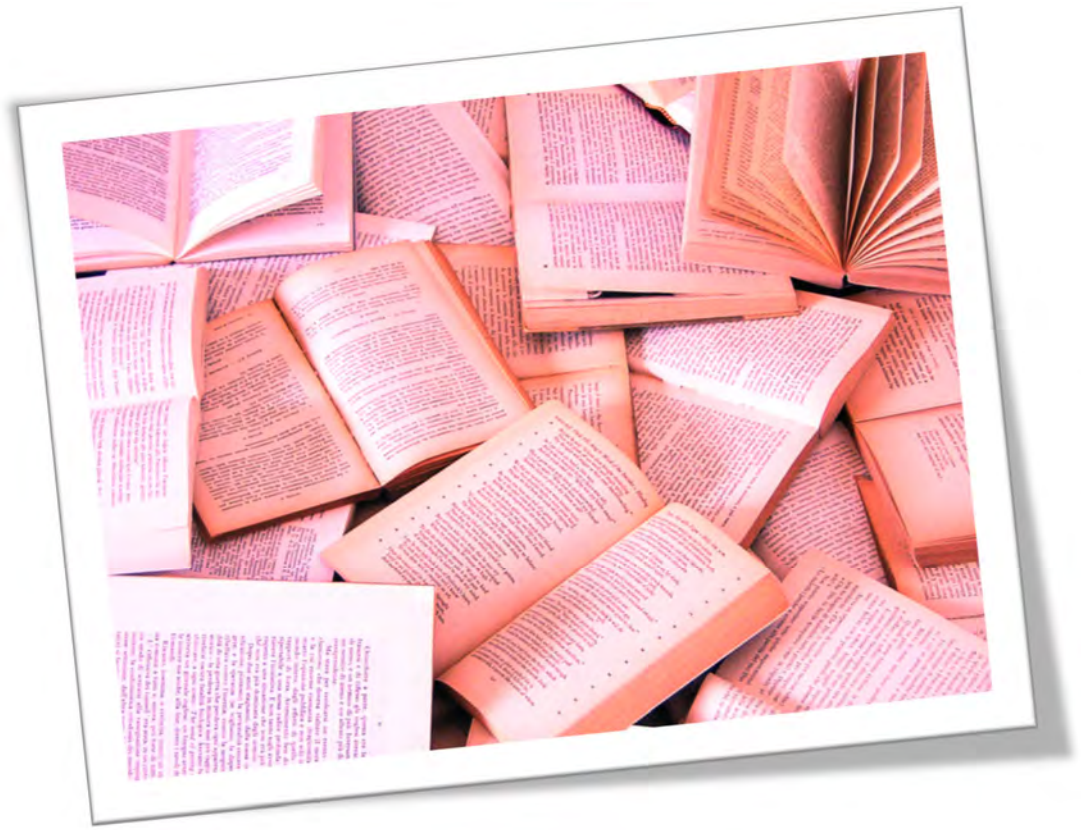
هوش مصنوعی ضعیف، برخلاف هوش مصنوعی قوی، قصد رسیدن به سطح هوش انسانی یا فراتر رفتن از سطح هوش انسانی را ندارد.

فصل دوّم

۱ هوش مصنوعی

---

۱ – پنج نفر از کسانی که در پروژه‌ی تحقیقاتی‌ای که در تابستان سال ۱۹۵۶ میلادی در شهر دارْتْمُوثْ(Dartmouth) در مورد هوش مصنوعی برگزار شد، شرکت داشتند؛ در سال ۲۰۰۶ میلادی به مناسبت پنجاهمین سالگرد هوش مصنوعی دوباره با هم دیدار کردند. در این تصویر به ترتیب از راست به چپ؛ دانشمندان زیر را می‌بینید:
رِیْ سُلُمُنُف(Ray Solomonoff)، ۲۰۰۹ – ۱۹۲۶ میلادی، دانشمندی آمریکایی که والدینش از روسیّه به آمریکا مهاجرت کرده بودند (http://en.wikipedia.org/wiki/Ray_Solomonoff)؛ اُلیورْ سِلْفْریجْ(Oliver Selfridge)، ۲۰۰۸ – ۱۹۲۶ میلادی (http://en.wikipedia.org/wiki/Oliver_Selfridge)؛ مارْوِینْ مینْسْکی(Marvin Minsky)، متولّد ۱۹۲۷ میلادی (http://en.wikipedia.org/wiki/Marvin_Minsky)؛ جان مِک‌کارتی(John McCarthy) و تِرِنْچارْدْ مُرْ(Trenchard More) (http://en.wikipedia.org/wiki/Trenchard_More)
(http://www.dartmouth.edu/~vox/0607/0724/ai50.html)



فهرست برخی از عنوان‌های نوشته‌ها

هوش مصنوعی چیست؟

در اینجا چهار طبقه‌بندی زیر را ارائه می‌کنیم:

۱- *سیستم‌هایی که شبیه انسان‌ها عمل می‌کنند.*
۲- *سیستم‌هایی که همانند انسان‌ها فکر می‌کنند.*
۳- *سیستم‌هایی که معقولانه[۱] فکر می‌کنند.*
۴- *سیستم‌هایی که معقولانه عمل می‌کنند.*

دوّمین و سوّمین مورد می‌توانند در طبقه‌بندی هوش مصنوعی قوی قرار گیرند و موردهای اوّل و چهارم، بیش‌تر در طبقه‌بندی هوش مصنوعی ضعیف قرار می‌گیرند.

در زیر توضیحاتی در مورد طبقه‌بندی‌هایی که ذکر شد، آمده است:

۱ - rationally

عمل کردن شبیه انسان: آزمایش تورینگ[۱]

شکل بالا– آلن تورینگ

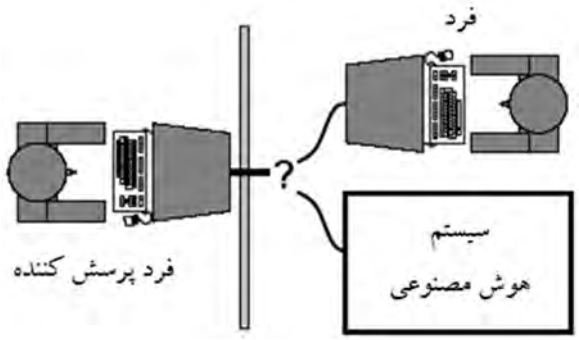
آلن تورینگ در سال ۱۹۵۰ میلادی در مقاله‌ی «ماشین‌آلات محاسبه‌گر و هوش»[۲] در مورد شرایطی برای ماشین هوشمند بحث کرده است. *او گفته است که: «اگر ماشین بتواند کاملاً وانمود کند که مانند انسان می‌باشد، آنگاه شما می‌توانید مطمئن باشید که هوشمند است.»*؛ این آزمایش توانست برخی از افراد و نه همه را مجاب نماید.

آزمایش تورینگ به این صورت است که در یک طرف، *فرد پرسش کننده‌ای* قرار دارد که می‌تواند با ماشین کار کند و در طرف مقابل، یک *فرد* و یک *سیستم هوش مصنوعی* قرار دارند؛ حال اگر *سیستم هوش مصنوعی* بتواند طوری با *فرد پرسش کننده* ارتباط برقرار کند که *فرد پرسش کننده* نتواند بفهمد که با *فرد* در ارتباط است یا با *سیستم هوش مصنوعی*، آنگاه این *سیستم هوش مصنوعی*، هوشمند است.

---

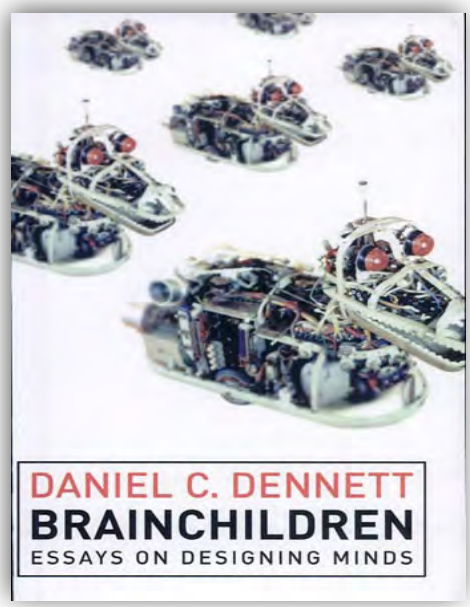
۱ - Turing test

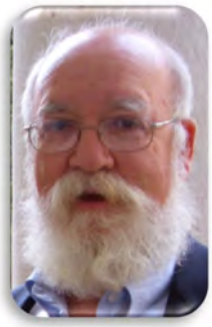
۲ - Computing machinery and intelligence



شکل بالا- آزمایش تورینگ

کتاب «به وجود آوردن فکرها»[۱]ی دانیلْ دِنت[۲] دارای یک بحث عالی در مورد آزمایش تورینگ و جوانب مختلف آزمایش تورینگ می‌باشد. در زیر تصویری از جلد این کتاب را می‌بینید:





دانیل دِنت

برخی از افراد به‌سادگی تصوّر می‌کنند که یک برنامه‌ی نسبتاً گنگ، هوشمند می‌باشد.[۳] اجزای اصلی پیشنهاد شده برای هوش مصنوعی عبارتند از: دانش[۴]، استدلال[۵]، فهم زبان[۶] و یادگیری[۱].

---

۱ - Brainchildren؛ توجّه کنید که کلمه‌ی Brainchild، به معنی «به وجود آوردن فکر»، «اختراع»، «ابتکار»، «ایده‌ی اصلی» می‌باشد.

۲ - Daniel Clement Dennett، متولّد ۱۹۴۲ میلادی، فیلسوفی آمریکایی است.
(http://en.wikipedia.org/wiki/Daniel_Clement_Dennett)

۳- http://www-formal.stanford.edu/jmc/whatisai/node1.html

۴ - knowledge

۵ - reasoning

۶ - language understanding

هنوز هیچ برنامه‌ای از آزمایش تورینگ سربلند بیرون نیامده است؛ یک برنامه در صورتی آزمایش تورینگ را با موفّقیّت می‌گذراند که قادر باشد زبان طبیعی را بفهمد؛ چیزهایی که می‌داند(دانش) را بتواند بیان(ارائه) کند؛ دانش داشته باشد و بتواند استدلال کند.

فکرکردن مانند انسان: علم شناخت[۲]

علم شناخت برای به وجود آوردن تئوری‌هایی که چگونگی عملکرد مغز انسان را توضیح می‌دهند، تلاش می‌کند؛ در این راه از مدل‌های کامپیوتری هوش مصنوعی و روش‌های تجربی روان‌شناسی استفاده می‌نماید. لازم است توجّه کنیم که بیش‌ترِ روش‌های هوش مصنوعی، به طور مستقیم، براساس مدل‌های شناختی نمی‌باشند؛ اغلب دشوار است که روش‌های هوش مصنوعی به صورت برنامه‌های کامپیوتری ترجمه شوند.

در دهه‌ی ۱۹۶۰میلادی، انقلاب شناخت به وجود آمد. علم شناخت دارای اشتراک با هوش مصنوعی می‌باشد. دانشمندان علم شناخت، طبیعت هوش را با یک دید روانی مطالعه می‌کنند و بیش‌تر، مدل‌های کامپیوتری‌ای را که به توضیح رخدادهای درون مغز ما، در مدّت حلّ مسأله، به یادآوردن، درک و دیگر فعّالیّت‌های روان‌شناسی، کمک می‌کنند را می‌سازند. یک اشتراک اساسی میان هوش مصنوعی و علم شناخت، برای روانشناسی، این بوده است که در مدل پردازش اطّلاعات فکری انسان، تشبیه *«مغز، به صورت کامپیوتر»*، به صورت لفظ به لفظ، کاملاً انجام می‌شود.[۳]

---

۱ - learning

۲ - cognitive science: **تعریف**- مطالعه‌ی فکر، یادگیری و سازمان ذهن است. (Babylon> Concise Oxford English Dictionary)

۳ - http://www.aaai.org/AITopics/html/cogsci.html


هِرِبِرت سایمِن

هِرْبِرْتْ سایْمِنْ[۱] در مورد علم شناخت می‌گوید: «هوش مصنوعی می‌تواند دو هدف داشته باشد؛ یکی، استفاده از توانایی کامپیوترها برای تکمیل فکر انسان؛ همان طوری که ما از موتورها یا اسب برای تکمیل توانایی انسان استفاده می‌کنیم؛ علم روبوتیک و سیستم‌های خبره، شاخه‌های اصلی این هدف اوّل می‌باشند. هدف دیگر، استفاده از هوش مصنوعی کامپیوتری برای فهمیدن چگونگی فکر انسان است؛ اگر شما در موقع آزمایش برنامه‌ها به کاری که برنامه‌ها می‌توانند انجام دهند، توجّه نکنید و به این توجّه کنید که *آن کار را چگونه انجام می‌دهند*، آنگاه آن شمایید که واقعاً علم شناخت را انجام می‌دهید؛ شما در حال استفاده از هوش مصنوعی برای فهمیدن عملکرد انسان هستید.»[۲]

علم شناخت و علم عصب‌شناسی[۳] در حال حاضر از هوش مصنوعی مجزّا هستند. هر دو دارای این ویژگی مشترک با هوش مصنوعی هستند: *تئوری‌های در دسترس هیچ چیزی را شبیه هوش انسان تولید نمی‌کنند*؛ بنابراین، هر سه زمینه در یک جهت اساسی مشترک هستند!

فکر منطقی(معقولانه): قوانین فکر– در این مورد قواعد اصولی[۴] بیش از تشریحی حاکم می‌باشند. مشکلات این روش، یکی این است که همه‌ی رفتارهای هوش با برّرسی‌های منطقی انجام نمی‌شوند؛ و دیگری اینکه میان حلّ یک مسأله در کل و حلّ آن در عمل تحت محدودیّت منبع‌های گوناگون، نظیر زمان، محاسبه و دقّت تفاوت وجود دارد.

عمل کردن به صورت منطقی(معقولانه)

رفتار منطقی[۵]: کار درست را انجام می‌دهد.

تعریف– *کار درست*، کاری است که بیش‌ترین دستیابی به هدف را داراست.

بدون فکر هم می‌توانیم کار درست را انجام دهیم؛ امّا فکر کردن باید در انجام کار منطقی(معقولانه) وجود داشته باشد. مزایای عمل کردن به صورت منطقی، یکی این است که کلّی‌تر است و دیگری این است که هدفش به‌خوبی تعریف شده است. اَرَسطو[۶] می‌گوید: هر هنر، هر تحقیق و غیره، هر عمل و عکس العمل، برای انجام صحیح احتیاج به فکر دارد.

---

۱ – Herbert Alexander Simon، ۱۹۱۶– ۲۰۰۱ میلادی، استاد دانشگاه، جامعه‌شناس و روان‌شناسی از کشور آمریکا بود. (http://en.wikipedia.org/wiki/Herbert_Simon)

۲ – http://www.aaai.org/aitopics/pmwiki/pmwiki.php/AITopics/CognitiveScience

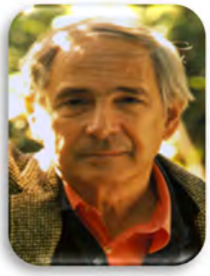
۳ – cognitive neuroscience؛ **تعریف**– شاخه‌ای از علم اعصاب است که اصول زیستی اتّفاقات روانی را مطالعه می‌کند. (Babylon> WordNet 2.0)

۴ – normative

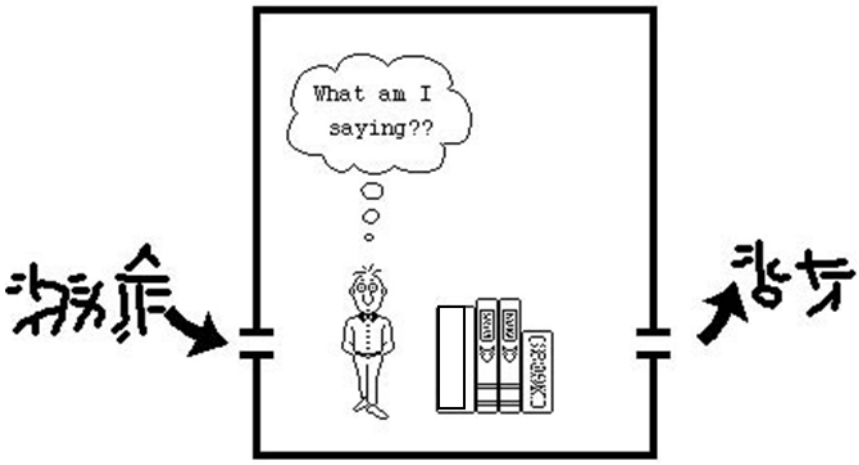
۵ – rational behavior

۶ – Aristotle، ۳۲۲ – ۳۸۴ قبل از میلاد حضرت مسیح (درود بر او باد)، فیلسوف و دانشمندی یونانی بوده است. (Babylon> Britannica Concise Encyclopedia) تصویری از او:

استدلال اتاق چینی


جان سِرل

استدلالی طرّاحی شده به وسیله‌ی جانْ سِرل برای کم ارزش کردن ادّعاهای مطرح شده به وسیله‌ی هوش مصنوعی قوی[۱] می‌باشد.[۲] از این استدلال همچنین برای نشان دادن اینکه مغز، یک کامپیوتر نمی‌باشد و آزمایش تورینگ ناقص است، استفاده شده است.[۳] این استدلال به این صورت می‌باشد: آقای سِرل اصلاً زبان چینی نمی‌فهمد و در یک اتاق با دریچه‌های ورودی و خروجی قرار دارد. قانون‌های استفاده از کاراکترهای چینی را در اختیار دارد و این کاراکترها کاملاً برای او ناآشنا هستند. متن‌ها و پرسش‌هایی به زبان چینی از ورودی به او می‌رسند و او با توجّه به قانون‌هایی که در اختیار دارد، یک جواب را به وجود می‌آورد و آن را از طریق خروجی ارسال می‌نماید. [پس از برّرسی، معلوم می‌شود که] جواب‌هایی که سرل داده، خیلی خوب است و فردی نمی‌تواند به او بگوید که یک سخنگوی زبان چینی نیست. *بنابراین، استدلال اتاق چینی در آزمایش تورینگ قبول می‌شود؛ به عبارت دیگر، شبیه یک فرد هوشمند عمل می‌کند. سرل بدون هیچ فهمی از زبان چینی، در آزمایش تورینگ قبول می‌شود. بنابراین، قبول شدن در آزمایش تورینگ، به فهمیدن احتیاج ندارد [و این در تناقض با آزمایش تورینگ است].*



۱- Strong AI

۲- http://en.wikipedia.org/wiki/Chinese_room_argument

۳- Babylon> Dictionary of Philosophy of Mind

تاریخچه‌ی هوش مصنوعی


وارِن مِک‌کالا


والتِر پیتز

در سال ۱۹۴۳ میلادی، وارِنْ مِک‌کالا[۱] و والْتِرْ پیتْزْ[۲] مدل مداری بولّین مغز را طرّاحی کردند؛ که این کار، پایه‌ای برای شبکه‌های عصبی[۳] شد.

در سال ۱۹۵۰ میلادی، مقاله‌ی آلن تورینگ به نام «ماشین‌آلات محاسبه‌گر و هوش» پخش شد و در آن، آلن تورینگ آنچه که امروزه آزمایش تورینگ نامیده می‌شود را تشریح کرد؛ از آزمایش تورینگ به عنوان یک روش آزمایش رفتار هوشمند استفاده می‌شود.

سال‌های ۱۹۵۲ تا ۱۹۶۹ میلادی؛ نظر و دست به دست کردن

---

۱ - Warren McCulloch، ۱۹۶۹ – ۱۸۹۸ میلادی، نوروفیزیولوژیست(neurophysiologist: فلسفه‌ی سیستم عصب است. (Babylon> Concise Oxford English Dictionary)) و سایبرنتیست(cybernetician: علم ارتباط‌ها و سیستم‌های کنترل خودکار، در ماشین‌ها و موجودهای زنده می‌باشد. (Babylon> Concise Oxford English Dictionary))ی از کشور آمریکا بود.
(http://en.wikipedia.org/wiki/Warren_Sturgis_McCulloch)

۲ - Walter Pitts ، ۱۹۶۹ – ۱۹۲۳ میلادی، منطقدانی از کشور آمریکا بود که در زمینه‌ی روانشناسی شناخت(cognitive psychology: شاخه‌ای از فلسفه است که علاقه‌مند به مطالعه‌ی ادراک یا شناخت انسان و مخصوصاً نتیجه‌ها یا اثرهای یادگیری و رفتار است. (Babylon> Britannica Concise Encyclopedia)) کار کرد.
(http://en.wikipedia.org/wiki/Walter_Pitts)

۳ - neural networks، این مطلب در فصل «شبکه‌های عصبی» همین کتاب توضیح داده شده است.


آرتور ساموئِل


آلِن نوئِل

سال‌های دهه‌ی ۱۹۵۰ میلادی، برنامه‌های هوش مصنوعی اوّلیّه، شامل برنامه‌ی بازی چِکِرْز[۱] ساموئِل[۲] (۱۹۵۹)، نظریّه‌ی منطقی نوئِل[۳] و هِربِرت سایمِن به وجود می‌آیند.

شکل بالا– آرتور ساموئل[۴]

سال ۱۹۵۶ میلادی، در نشست دارْتْمُوثْ[۵]، «هوش مصنوعی» پذیرفته شد.

---

۱ – checkers(جَنگِ نادِر یا دام یا Draughts)؛ برای آگاهی‌های بیش‌تر در زمینه‌ی این بازی، به فصل «تئوری بازی‌ها»ی همین کتاب الکترونیکی مراجعه نمایید.

۲ – Arthur Samuel، ۱۹۹۰ – ۱۹۰۱ میلادی، یک آمریکایی و پیشگام(پیشکسوت یا pioneer)ی در زمینه‌ی بازی کامپیوتری و هوش مصنوعی بود. (http://en.wikipedia.org/wiki/Arthur_Samuel)

۳ – Allen Newell آمریکایی، ۱۹۹۲ – ۱۹۲۷ میلادی، پژوهشگری در زمینه‌ی علوم کامپیوتری و روانشناسی شناخت بود. (http://en.wikipedia.org/wiki/Allen_Newell)

۴ – http://infolab.stanford.edu/pub/voy/museum/pictures/AIlab/3507ArtSamuelTTY.JPG؛ این پایگاه اینترنتی، متعلّق به دانشگاه استنفورد کشور آمریکاست.

۵ – Dartmouth meeting


جانْ آلِنْ رابینسُونْ

در سال ۱۹۶۵ میلادی، الگوریتم کامل رابینْسُونْ[۱] برای استدلا ل منطقی ارائه شد.

در سال‌های۱۹۶۶ تا ۱۹۷۴ میلادی، هوش مصنوعی، پیچیدگی محاسباتی پیدا می‌کند و پژوهش در مورد شبکه‌ی عصبی تقریباً ناپدید می‌شود.

در سال‌های ۱۹۶۹ تا ۱۹۷۹ میلادی، توسعه‌ی اوّلیّه‌ی سیستم‌های برمبنای دانش[۲].

در سال‌های ۱۹۸۰ تا ۱۹۸۸ میلادی، سیستم‌های خبره‌ی صنعتی، توسعه‌ی عظیمی پیدا می‌کنند.

در سال‌های ۱۹۹۳ تا ۱۹۹۸ میلادی، سیستم‌های خبره‌ی صنعتی ورشکست می‌کنند؛ این دوره را «زمستان هوش مصنوعی» می‌گویند.

در سال‌های ۱۹۸۵ تا ۱۹۹۵ میلادی، شبکه‌های عصبی شهرت می‌یابند.

سال ۱۹۸۸ میلادی، تجدید حیات احتمال، پیشرفت در منطق فازی، الگوریتم‌های ژنتیکی و غیره.

سال ۱۹۹۵ میلادی، نظریّه‌ی *عامل‌ها* محبوبیّت پیدا می‌کند.

سال ۲۰۰۳ میلادی، به سطح انسانی هوش مصنوعی پرداخته می‌شود.

---

۱ – John Alan Robinson، متولّد ۱۹۳۰ میلادی، فیلسوف، ریّاضیدان، دانشمند علوم کامپیوتری و استاد دانشگاه می‌باشد. وی آمریکایی است. (http://en.wikipedia.org/wiki/J._Alan_Robinson)

۲ – knowledge-*b*ased systems(*KBS*)؛ **تعریف**– همان طور که در گذشته هم گفتیم، پایگاه دانش(*K*nowledge Base(*KB*)، مجموعه‌ای از دانش است که با استفاده از زبانی قراردادی، برای بیان دانش، بیان شده است. یک پایگاه دانش، قسمتی از یک سیستم بر مبنای دانش را به وجود می‌آورد. سیستم بر مبنای دانش، برنامه‌ای برای گسترش و/یا پرس وجوی یک پایگاه دانش است. (Babylon> FOLDOC)

چگونگی(وضعیّت) فن(جدیدترین فنّـآوری)[۱]

در حال حاضر کدام یک از موردهای زیر به وسیله‌ی کامپیوتر می‌تواند انجام شود؟

اجرای یک بازی تنیس. (می‌شود)

رانندگی با ایمنی در یک جادّه‌ی مارپیچی کوهستانی. (می‌شود)

خرید نیازهای هفتگی از فروشگاه‌های وب. (می‌شود)

اجرای یک بازی پُل[۲]. (می‌شود)

ترجمه کردن بلافاصله‌ی زبان گفتاری انگلیسی به زبان گفتاری سوئدی. (می‌شود)

مکالمه‌ی موفّق با دیگر افراد برای مدّت یک ساعت. (نمی‌شود)

انجام دادن یک عمل جرّاحی پیچیده. (نمی‌شود)

---

۱ - state of the art

۲ - bridge؛ **توضیح**- بازی‌ای کارتی (با ورق یا پاسور) و چهار نفره می‌باشد. مانند سایر بازی‌های کارتی، یکی از ویژگی‌های اوّلیّه‌ی این بازی این است که بازی‌ای با اطّلاعات ناقص می‌باشد که در آن، بازی کننده‌های مختلف، دارای اطّلاعات مختلفی در مورد وضعیّت واقعی بازی می‌باشند؛ این، یکی از ویژگی‌های این بازی می‌باشد که باعث می‌شود ماشین‌ها بازی کننده‌های شایسته‌ای برای بازی پل نباشند.
(http://www.cirl.uoregon.edu/research/bridge.html؛ این پایگاه اینترنتی، متعلّق به آزمایشگاه تحقیقاتی هوش محاسباتی(Computational Intelligence Research Laboratory(CIRL)) دانشگاه ایالت اُرِگُن(Oregon) کشور آمریکا است.) در زیر تصویری از کارت‌های این بازی را مشاهده می‌کنید:

چکیده‌ی مطلب‌های فصل دوّم

آلن تورینگ در مقاله‌ی «ماشین‌آلات محاسبه‌گر و هوش» گفته است که: «اگر ماشین بتواند کاملاً وانمود کند که مانند انسان می‌باشد، آنگاه شما می‌توانید مطمئن باشید که هوشمند است.»

از استدلال اتاق چینی برای نشان دادن اینکه آزمایش تورینگ ناقص است هم استفاده شده است.

یادآوری یا تکمیل مطلب‌های فصل دوّم

سؤال– آزمایش تورینگ را به وسیله‌ی شکل و کلمات توضیح دهید.

جواب– به متن درس بروید.[۱]

درست یا غلط– آزمایش تورینگ، آزمایشی طرّاحی شده برای نشان دادن این است که آیا به یک سیستم می‌توان گفت هوشمند است یا نه؟.

جواب– **«درست» است.**[۲]

درست یا غلط– یک سیستم برای موفّق شدن، به طور قابل اعتماد، در آزمایش تورینگ باید شبیه انسان فکر کند.

جواب– **«غلط» است؛** فقط باید شبیه انسان عمل کند.[۳]

تست– یک کامپیوتر برای موفّق شدن در آزمایش تورینگ استاندارد به کدام توانایی زیر نیاز دارد؟

۱) چشم

۲) دست

۳) استدلال

۴) همه‌ی موارد

---

۱ – آزمون درس «هوش مصنوعی» استاد، رُجِرْ جُنسِنْ(Roger Jonsson)، دانشکده‌ی نوآوری(ابداع)، طرّاحی و مهندسی دانشگاه مالاردالِن(Malardalen) کشور سوئد، ۱۰ ژانویّه‌ی سال ۲۰۰۵ میلادی

۲ – مترجم

۳ – آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۲۰۰۲ میلادی

۵) هیچکدام

جواب- گزینه‌ی «۳» درست است.[۱]

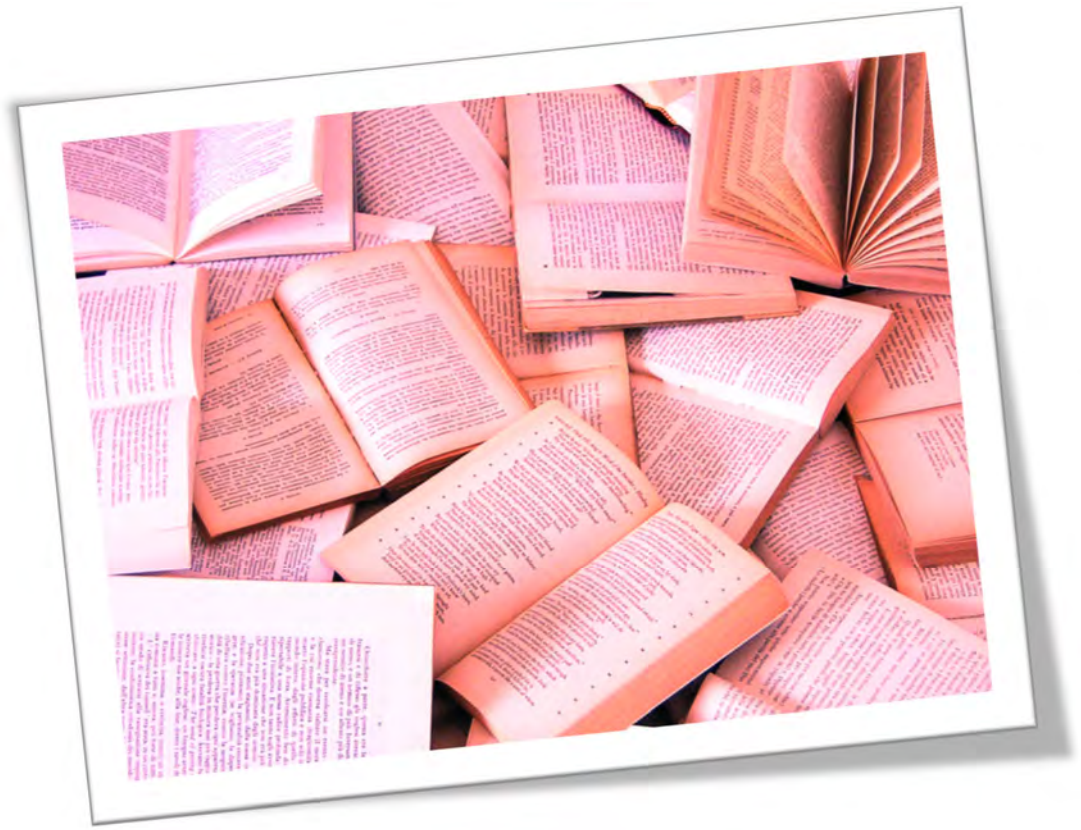
۱ - آزمون میان‌ترم درس «هوش مصنوعی» استاد، «روت شولتز(Ruth Schulz)»، دانشکده‌ی مهندسی برق و فنّـآوری اطلاعات دانشگاه کوئینزلند کشور استرالیا، سال ۲۰۱۲ میلادی

فصل سوّم

عامل‌های هوشمند[۲]

۱ - انسان‌ها و روبوت‌ها نمونه‌هایی از عامل‌ها هستند.

۲ - Intelligent agents



فهرست برخی از عنوان‌های نوشته‌ها

دوره‌ای و ترتیبی بودن

ثابت(ایستا) و پویا(دینامیک) بودن

گسسته و پیوسته بودن

تک عاملی یا چندعاملی بودن

انواع عامل‌ها

عامل‌های بازتابی ساده

عامل‌های بازتابی با حالت(وضعیّت) یا عامل‌هایی که وضعیّت جهان را حفظ می‌نمایند یا عامل‌های بازتابی دارای وضعیّت‌های درونی یا عامل‌های بازتابی براساس مدل

عامل‌های هدف‌گرا

عامل‌های سودمند

عامل‌های یادگیرنده

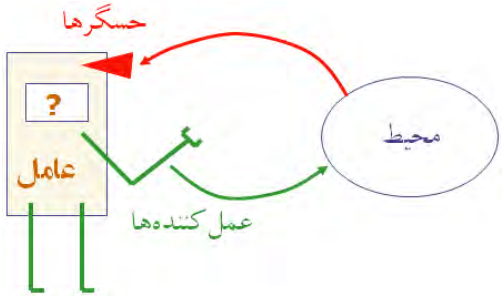
عامل‌های نرم‌افزاری

عامل‌های متحرّک

عامل‌های اطّلاعاتی

عامل‌های هوشمند

تعریف: یک عامل هوشمند، [وضعیّت] محیط[۱] خود را با استفاده از حسگرها[۲] دریافت(حس) می‌کند[۳]؛ روی محیط خود با استفاده از عمل‌کننده‌ها[۴] به صورت معقولانه[۵] عمل می‌نماید.



تعریف ادراک‌ها(دریافت‌ها یا درک‌ها): اطّلاعات فرستاده شده به حسگرهای یک عامل می‌باشند؛ به عنوان مثال، نور، صدا، امواج الکترومغناطیسی و علائم(سیگنال‌ها).

تعریف حسگرها: روش‌های یک عامل برای جمع‌آوری اطّلاعات در مورد محیطش می‌باشند؛ مثل: چشم‌ها، گوش‌ها، سلّول‌های فتوالکتریک[۶] و....

---

۱ - environment

۲ - sensors

۳ - perceives

۴ - actuators

۵ - rationally

۶ - photoelectric cell، یک تبدیل‌کننده‌ی انرژی است که برای پیداکردن و اندازه‌گیری نور و دیگر تابش‌ها استفاده می‌شود. (Babylon> WordNet 2.0)

تعریف عمل‌کننده‌ها: عامل به وسیله‌ی عمل‌کننده‌ها بر روی محیط عمل می‌نماید؛ تایرها، بازوها، نورها و... مثال‌هـایی از عمل‌کننده‌ها هستند.

تعریف عملکرد(عمل)[۱]: کار یا عملی است که بر روی محیط انجام می‌شود؛ مثل: حرکت و غلطیدن.

عامل‌ها به وسیله‌ی کاربران یا برنامه‌های دیگر، برنامه‌ریزی می‌شوند و این برنامه‌ها عامل را برای انجـام کـارش راهنمـایی می‌کند. عامل‌ها به دانش اوّلیّه نیاز دارند و باید بتوانند یاد بگیرند و برای کارهای به مراتب پیچیده‌تر، دارای انعطاف باشند.

تعریف– *یادگیری*، به معنی بهبوددادن عملکرد یک عامل است.
یادگیری ممکن است به معنی کاهش نامعلومی یا اِعمال ملاحظاتی به حساب یا گزارش باشد.

عامل‌ها دوست دارند کاری که انجام می‌دهند دارای پیش‌نیاز نباشد. مشخّص کردن اینکه در هر موردی بایـد چـه کـاری انجام دهند، مشکل است. در مورد عامل‌ها بهتر است فرض را بر این بگذاریم که آنها هوشمند هستند و سپس در مورد آنها صحبت کنیم.

مثال‌هایی از عامل‌ها

عامل‌ها شامل انسان‌ها، روبوت‌ها، ترموستات[۲]هـا(تنظیم کننـده‌های حـرارت) و... می‌باشـند؛ مؤلّفان(نویسـندگان)، دارای عامل‌ها هستند؛ ورزشکاران حرفـه‌ای، دارای عامل‌هـا هسـتند؛ سـتاره‌های سـینما، دارای عامل‌هـا هسـتند؛ و شـما نیـز دارای عامل‌هـا می‌باشید؛ زیرا

تعریف– عامل، فردی یا چیزی دارای تخصّص می‌باشد که عهده‌دار انجام کاری برای شما می‌شود.
برنامه‌های کامپیوتری‌ای که به شما کمک می‌کنند که دانسته‌های محاسباتی خود را افـزایش دهیـد هـم جـزء «عامل‌هـا» می‌باشـند. پژوهش‌های دانشمندان هوش مصنوعی در حال افزایش توانایی، استقلال و... عامل‌ها می‌باشد.[۳]

برای عامل انسانی؛
حسگرها؛ مثل: چشم‌ها، گوش‌ها، پوست، حسّ چشایی[۴] و....
عمل‌کننده‌ها؛ مثل: دست‌ها، انگشتان، پاها، دهان[۱] و....

۱- action
۲- thermostat
۳- http://www.aaai.org/AITopics/html/agents.html
۴- taste buds

برای روبوت؛

حسگرها؛ مثل: دوربین[۲]، دوربین مادون قرمز[۳]، ضربه‌گیر(سپَر)[۴] و....

عمل‌کننده‌ها؛ مثل: گیرنده‌ها(ابزارهایی که چیزی را می‌گیرند)[۵]، تایرها، نورها، بلندگوها و....

برای عامل نرم‌افزاری یا سافتبوت[۶]؛

حسگرها به صورت توابع هستند و ورودی توابع، اطّلاعات آماده شده به صورت رشته‌های بیتی یا نشان(سَمبُل)[۷]های کُد شده هستند. توابع که عمل‌کننده‌ها نیز هستند، خروجی‌ها را اجرا می‌کنند؛ پس در این گونه از عامل‌ها، توابع، هم به صورت حسگر هستند و اطّلاعات را به صورت رشته‌های بیتی یا نشان‌ها(سمبل‌ها)ی کد شده می‌گیرند و هم به صورت عمل‌کننده هستند و بر روی محیط عمل می‌کنند.

خودمختاری[۸] در عامل‌ها

عامل‌ها وظایف خود را تا حدّ زیادی به صورت مستقل انجام می‌دهند، ولی این خودمختاری اغلب به صورت کامل نمی‌باشد. مبنای سیستم‌های خودمختار، تجربه و دانش آنها می‌باشد. مشکلی که در مورد خودمختاری عامل‌ها وجود دارد، این است که عامل خودمختار، چه هنگام باید درخواست کمک کند؟.

ساختار عامل‌های هوشمند

نکته:

(کنکور آزاد مهندسی کامپیوتر سال ۸۱) *عامل از معماری(ساختار) و برنامه تشکیل شده است.*

۱- mouth
۲- camera
۳- infrared
۴- bumper
۵- grippers
۶- softbot
۷- symbol
۸- autonomy

تعریف- معماری، *ابزاری است که به وسیله‌ی آن می‌توان برنامه‌ی عامل را اجرا کرد*؛ مثل: کامپیوتر همه منظوره، ابزار تخصّصی، روبوت و....

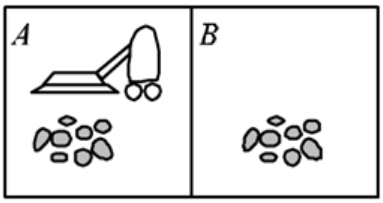
برنامه‌ی عامل‌ها

می‌توانیم رفتار عامل‌مان را با یک تابع F توصیف نماییم. داریم:

عملکرد = (تاریخچه‌ی ادراکی[۱] یا دنباله(رشته)ی ادراکی[۲]، ادراک فعلی) F

به کارگیری این تابع، کار برنامه‌ی عامل می‌باشد.

مثال- دنیای جاروبرقی[۳]



بیایید با یک مثال خیلی ساده شروع کنیم؛ فرض کنید که دو فضای A و B وجود دارند که هر فضا یا می‌تواند تمیز[۴] و یا کثیف[۵] باشد و این، *محیط* عامل است. *حسگرها* عبارتند از: حسگر کثیفی و تعیین محل. *عمل‌کننده‌ها* عبارتند از: جاروبرقی و چرخ‌ها. ادراک‌ها هم، تمیز و کثیف بودن محیط می‌باشد. عملکردها عبارتند از: حرکت به چپ، راست، مَکِش[۶] و هیچ کار. *در این مثال ساده می‌توانیم همه‌ی رشته‌های ادراکی ممکن و عملکردهای وابسته را لیست نماییم و این، یک عامل برمبنای جدول[۷] نام دارد:*

رشته‌ی ادراکی	عملکرد
[A,Clean]	Right
[A,Dirty]	Suck
[B,Clean]	Left
[B,Dirty]	Suck
[A,Clean],[A,Clean]	Right
[A,Clean],[A,Dirty]	Suck
⋮	⋮

۱- percept history
۲- percept sequence
۳- Vacuum Cleaner
۴- clean
۵- dirty
۶- suck
۷- table-based

عامل‌های عقلانی(معقول)[۱]

تعریف- به طور کلّی، عقلانیّت[۲] به معنی «انجام کار(چیز) درست می‌باشد»؛

در این مورد دقّت بیش‌تری لازم می‌باشد؛ «کار(چیز) درست» چیست؟. ما به یک تعریف از موفّقیّت هم نیازمندیم.

یادآوری- *کار درست، کاری است که بیش‌ترین دستیابی به هدف را داراست.*

تعریف- *یک عملکرد عقلانی(عاقلانه یا از روی عقل)، عملکردی است که معیارکارآیی(ارزیابی عملکرد)[۳] یک عامل، ادراک‌ها و عملکردهای آن را بیشینه می‌نماید.*

تعریف معیارکارآیی- یک معیار(اندازه‌ی) ذهنی برای تعیین چگونگی موفّقیّت یک عامل است؛ مثلاً معیارکارآیی می‌تواند سرعت، مصرف انرژی، دقّت، پول و غیره باشد.

مثال- در مورد عامل جاروبرقی؛ معیارکارآیی به این صورت است: فضا یا تعداد کاشی‌هایی که در یک زمان معیّن تمیز می‌شوند؛ انرژی؛ پارازیت(عامل مُخِل)[۴]؛ گم شدن قطعات مفید؛ خراب شدن اثاثیه‌ها؛ و خراش کَف.

تعریف عامل عقلانی- *یک عامل عقلانی هر کدام از اعمالی که مقدار معیارکارآیی ارائه شده به وسیله‌ی رشته‌ی ادراکی مورد نظر را به بیش‌ترین مقدار افزایش می‌دهد، انتخاب می‌نماید.*

عامل عقلانی «به همه چیز آگاهی(همه چیزدانی)»[۵] ندارد؛ دریافت ادراکی ممکن است همه‌ی اطّلاعات را به وجود نیاورد. عامل عقلانی، دارای حس‌های بیش‌تر از حس‌های معمولی[۶] هم نمی‌باشد. عملکرد به وجود آمده ممکن است به صورتی که انتظار داشتیم، نباشد؛ در این صورت عامل عقلانی موفّقیّت‌آمیز نمی‌باشد. عامل عقلانی، دارای ابتکار[۷]، یادگیری و خودمختاری می‌باشد.

آقایان، راسل و نورویگ در مورد عامل‌های عقلانی می‌گویند: *«عامل‌های عقلانی برای هر رشته‌ی ادراکی؛ باید عملکردی که انتظار می‌رود، معیارکارآیی، ملاک ارائه شده به وسیله‌ی رشته‌ی ادراکی و هر چه در دانش عامل وجود دارد را*

۱- rational agent
۲- rationality
۳- Performance measure
۴- noise؛ در اینجا به عنوان مثال: سروصدای جارو برقی
۵- omniscient
۶- clairvoyant
۷- exploration

بیشینه می‌کند، انتخاب نمایند.»[۱] ما نیازی نداریم که عامل قادر باشد آینده را پیش‌بینی نماید یا رویدادهای بد را پیش‌گویی نماید. جمع‌آوری اطّلاعات هم ممکن است یک عملکرد عقلانی باشد؛ به عنوان مثال، عبور بدون توجّه از خیابان، نامعقولانه می‌باشد. عامل‌های عقلانی باید دارای توانایی یادگیری باشند (به جز در موردهای خیلی ساده و محیط‌هایی که به‌خوبی قابل فهم می‌باشند).

PEAS[۲](محیط کار(وظیفه، عملیّاتی)[۳]؛ معیارکارآیی(ارزیابی عملکرد)، محیط، عمل‌کننده‌ها و حسگرها)

برای طرّاحی یک عامل عقلانی باید «محیط عملیّاتی» را مشخّص کنیم؛ مثلاً درطرّاحی یک تاکسی خودکار این موردها را باید مشخّص نماییم: معیارکارآیی؟؟| محیط؟؟| عمل‌کننده‌ها؟؟| حسگرها؟؟.

مثال- در طرّاحی یک تاکسی خودکار؛ محیط عملیّاتی به صورت زیر است:

معیارکارآیی؟؟؛ ایمنی[۴]، مقصد[۵]، مزایا[۶]، رعایت قانون[۷]، آسایش و راحتی[۸] و..... می‌باشند.

محیط؟؟؛ خیابان‌ها(آزاد راه‌ها)ی کشور ایالات متّحده‌ی آمریکا، ترافیک[۹]، عابران پیاده[۱۰]، هوا(وضعیّت جوّی)[۱۱] و..... می‌باشند.

عمل‌کننده‌ها؟؟؛ فرمان[۱۲]، گاز[۱۳]، ترمز[۱۴]، بوق[۱۵]، بلندگو یا نمایش دهنده[۱۶] و..... می‌باشند.

---

۱ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل دوّم، عامل‌های هوشمند(Intelligent Agents)، صفحه‌ی ۳۷

۲ - در کلمه‌ی «*PEAS*»، «*P*» برای کلمه‌ی «Performance measure»، «*E*» برای کلمه‌ی «*E*nvironment»، «*A*» برای کلمه‌ی «*A*ctuators» و «*S*» برای کلمه‌ی «Sensors» استفاده شده است. این کلمه را می‌توانید «پِعَسْ» بخوانید.

۳ - task environment

۴ - safety

۵ - destination

۶ - profit

۷ - legality

۸ - comfort

۹ - traffic

۱۰ - pedestrians

۱۱ - weather

۱۲ - steering

۱۳ - accelerator

۱۴ - brake

۱۵ - horn

۱۶ - speaker/display

حسگرها؟؟؛ ویدئو[۱]، شتابْ سنج[۲]، درجه‌ها[۳]، حسگرهای موتور[۴]، صفحه‌کلید[۵]، GPS[۶] و..... می‌باشند.

مثال- در عامل‌های خرید[۷] اینترنتی؛

معیارکارآیی؟؟؛ قیمت[۸]، کیفیّت[۹]، تناسب[۱۰] و کارآیی[۱۱] می‌باشند.

محیط؟؟؛ سایت‌های فعلی و آینده‌ی www، فروشنده‌ها[۱۲] و حمل‌کننده‌ها[۱۳] می‌باشند.

عمل‌کننده‌ها؟؟؛ نمایش دادن به کاربر[۱۴]، دنبال کردن آدرس اینترنتی(URL) و پرکردن فرم می‌باشند.

حسگرها؟؟؛ صفحات HTML[۱۵]، که به صورت‌های متنی، گرافیکی و غیره می‌باشند.

محیط‌ها

یک معیار برای وجود یک عامل، وجود آن در یک محیط می‌باشد؛ لازم نیست که یک عامل، به صورت فیزیکی، در محیط باشد؛ ممکن است محیط، به صورت یک محیط نرم‌افزاری باشد.

۱- video
۲- accelerometer
۳- gauges
۴- engine sensors
۵- keyboard
۶- *GPS*، کوتاه شده‌ی Global *P*ositioning *S*ystem است و یک سیستم مکان‌یابی ماهواره‌ای می‌باشد.
(Babylon> Concise Oxford English Dictionary)
۷- shopping
۸- price
۹- quality
۱۰- appropriateness
۱۱- efficiency
۱۲- vendors
۱۳- shippers
۱۴- display to user
۱۵- *HTML*، کوتاه شده‌ی «*H*yper*T*ext *M*arkup *L*anguage» است و زبانی پایه‌ای برای به وجود آوردن صفحه‌های وب می‌باشد.
(http://en.wikipedia.org/wiki/HTML)

ویژگی‌های(خصوصیّات) محیط‌ها

عبارتنـد از: قابـل مشـاهده بـودن[۱]؛ قطعـی[۲] /(یـا) اتّفاقی(تصـادفی)[۳] بـودن؛ دوره‌ای(اپیزودیـک)[۴] و ترتیبـی[۵] بـودن؛ ثابت(استاتیک)[۶] و پویا(دینامیک)[۷] بودن؛ گسسته[۸] و پیوسته[۹] بودن؛ تک عاملی[۱۰] و چند عاملی[۱۱] بودن.

قابل مشاهده بودن

تعریف– در صورتی که حسگرهای عامل همیشه اطّلاعات کاملی در مورد اجزای مربوط محیط بدهنـد، محیط، کاملاً قابل مشاهده(قابل دسترس[۱۲]) است؛ به عنوان مثال، محیط بازی شطرنج، کاملاً قابل مشاهده می‌باشد؛ محیط خریـد اینترنتی و تاکسی، به صورت جزئی قابل مشاهده می‌باشند.

قطعی /(یا) اتّفاقی(تصادفی) بودن

می‌توانیم فکر کنیم که جهان دارای انتقال میان وضعیّت‌ها می‌باشد.

تعریف– *در صورتی که وضعیّت بعدی کاملاً مشخّص باشد، جهان به صورت قطعی می‌باشد؛*
به عنوان مثال، بازی شطرنج، بازی‌ای قطعی می‌باشد؛ دنیای جاروبرقی، قطعی است؛ فروشگاه اینترنتی هم قطعی است.

تعریف– *در محیط‌های اتّفاقی، نتیجه‌هایی غیر قابل پیش‌بینی وجود دارند؛* بـه عنـوان مثـال، رانـندگی یـک خـودرو، اتّفـاقی می‌باشد.

---

۱ – observability
۲ – deterministic
۳ – stochastic
۴ – episodic
۵ – sequential
۶ – static
۷ – dynamic
۸ – discrete
۹ – continuous
۱۰ – single agent
۱۱ – multi-agent
۱۲ – accessible

نکته:

ممکن است که جهان، قطعی باشد، امّا با توجّه به پیچیدگی‌اش به نظر اتّفاقی برسد.

دوره‌ای و ترتیبی بودن

تعریف– در حالت دوره‌ای، هر عملکرد به صورت مستقل می‌باشد و عامل، ادراک می‌کند، تصمیم می‌گیرد و عمل می‌نماید و *دوباره[از نو] شروع می‌نماید* و تصمیم‌گیری بعدی، وابسته به وضعیّت‌های قبلی نمی‌باشد؛ به عنوان مثال، عامل فیلترکننده‌ی اِسْپَم[۱]، دوره‌ای می‌باشد.

تعریف– *در صورتی که عامل باید یک سری از عملیّات را برای رسیدن به یک عمل یا رسیدن به یک هدف انجام دهد، محیط، ترتیبی می‌باشد.* در محیط ترتیبی باید به تصمیم‌گیری‌های آینده توجّه شود؛ به عنوان مثال، رانندگی یک خودرو، ترتیبی می‌باشد.

ثابت(ایستا) و پویا(دینامیک) بودن

تعریف– *یک محیط ثابت، مادامی که عامل در حال تصمیم‌گیری در مورد یک عملکرد می‌باشد، ثابت باقی می‌ماند و عامل برای رسیدن به یک تصمیم، تحت محدودیّت یا فشار زمانی نمی‌باشد؛* به عنوان مثال، محیط فیلترکردن اسپم، ثابت می‌باشد؛ بازی شطرنج هم ثابت می‌باشد.

تعریف– *در یک محیط پویا، در زمانی که عامل در حال تصمیم‌گیری در مورد این می‌باشد که چه کاری را انجام بدهد، محیط تغییر پیدا می‌کند؛* عامل در این مورد باید «به اندازه‌ی کافی با سرعت» عمل نماید؛ به طور مثال، (**مشابه کنکور آزاد مهندسی کامپیوتر سال ۹۲**– رانندگی یک خودرو، پویا می‌باشد.)

تعریف– در محیط *نیمه‌پویا*[۲]، محیط تغییر نمی‌کند، امّا معیار کارآیی در طول زمان تغییر می‌کند؛ به عنوان مثال، انجام یک آزمایش، در زمان محدود؛ و بازی شطرنج با ساعت، مثال‌هایی از محیط‌های نیمه‌پویا می‌باشند.

---

۱ – spam، پیام‌های نامربوط یا نامناسبی می‌باشند که در اینترنت برای تعداد زیادی از کاربران یا گروه‌های خبری فرستاده می‌شوند. (Babylon> Concise Oxford English Dictionary)

۲ – semi-dynamic

گسسته و پیوسته بودن

تعریف– می‌توانیم در مورد گسسته و پیوسته بودن ادراک‌ها، عملکردهای عامل یا وضعیّت‌های ممکن محیط صحبت کنیم؛ در صورتی که مقادیر هر کدام از موردهای گفته شده، به صورت مجموعه‌ای گسسته باشد، آنگاه محیط، گسسته می‌باشد. محیط گسسته، شبیه محیط محدود نمی‌باشد.
مثلاً یک محیط فیلترکننده‌ی اسپم، گسسته می‌باشد، حتّا در میان تعداد نامحدود پیام‌های الکترونیکی؛ محیط خرید اینترنتی هم گسسته است.

تعریف– *یک محیط پیوسته، دارای متغّیرهای تغییر کننده به صورت پیوسته است؛* به طور مثال، زاویّه‌های فرمان، در یک محیط رانندگی خودرو؛ *در اینجا نکته این است که آیا یک تغییر قابل تشخیص، میان دو مقدار وجود دارد یا نه؟.*

تک عاملی یا چندعاملی بودن

تعریف– *در محیط تک عاملی، عامل‌های دیگر بر روی عامل ما اثر نمی‌گذارند؛* به طور مثال، محیط فیلترکردن اسپم، به صورت تک عاملی می‌باشد.

تعریف– *در محیط چند عاملی، عملکردها، اهداف و روش‌های عامل‌های دیگر باید به حساب آیند؛* مثل: محیط بازی شطرنج؛ خرید اینترنتی؛ و تاکسی.

نکته:

گرچه یک جهان ممکن است دارای عامل‌های دیگری باشد، امّا ممکن است به خاطر پیچیدگی استنتاج، با آن به صورت تک عاملی و اتّفاقی رفتار نماییم؛ به عنوان مثال، یک عامل کنترل کننده‌ی علائم ترافیکی.

نوع محیط تا حدّ زیادی طرح عامل را تعیین می‌کند. *بیش‌ترِ* وضعیّت‌های دنیای واقعی، اندکی قابل مشاهده، تصادفی، ترتیبی، پویا، پیوسته و چند عاملی می‌باشد.

برنامه‌های محیط

تعریف– برنامه‌های محیط، شبیه‌سازی کننده‌های محیط برای آزمایش‌هایی که به وسیله‌ی عامل‌ها انجام می‌شود، می‌باشند؛ یک ادراک را به یک عامل ارائه می‌نمایند، یک عملکرد را دریافت می‌نمایند و محیط را به‌روز می‌نمایند. خیلی از وقت‌ها هم طرزکارهایی را برای معیارکارآیی عامل‌ها ارائه می‌نمایند.

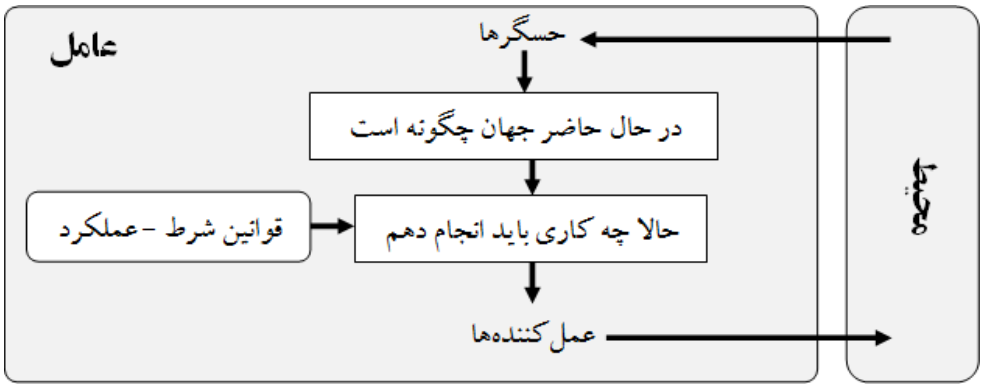
انواع عامل‌ها

۱- *عامل‌های بازتابی ساده*[۱]
۲- *عامل‌های بازتابی با حالت(وضعّیت)* یا *عامل‌هایی که وضعّیت جهان را حفظ می‌نمایند*[۲] یـا *عامل‌هـای بازتـابی دارای وضعّیت‌های درونی*[۳] یا *عامل‌های بازتابی براساس مدل*[۴]
۳- *عامل‌های براساس هدف یا هدف‌گرا*[۵]
۴- *عامل‌های سودمند*[۶]
۵- عامل‌های یادگیرنده[۷]

۱- عامل‌های بازتابی ساده

تعریف- تاریخچه‌ی ادراک‌ها را در نظر نمی‌گیرند(حافظه ندارند)؛ ادراک‌ها را به عملیّات نگاشت می‌نماینـد؛ و تابع آنها براساس قانون‌های شرط- عملکرد[۸] است.

پیاده‌سازی این نوع عامل‌ها آسان می‌باشد؛ ولی دارای کاربرد اندکی می‌باشند و جدول جستجوی شـرط-عملکـرد آنهـا هـم خیلـی بزرگ می‌باشد.

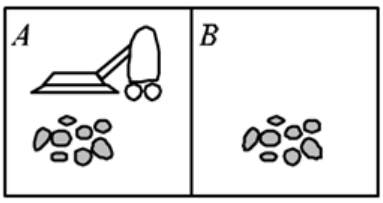


۱ - simple reflex agents
۲ - reflex agents with state
۳ - reflex agents with internal states
۴ - model-based reflex agents
۵ - goal-based agents
۶ - utility-based agents
۷ - learning agents
۸ - condition-action rules؛ یعنی، اگر شرط برقرار باشد، عمل انجام می‌شود.

مثال- در دنیای جاروبرقی،

تابع Reflex[1]-Vacuum-Agent([location[2],status[3]])، یک عملکرد را برمی‌گرداند

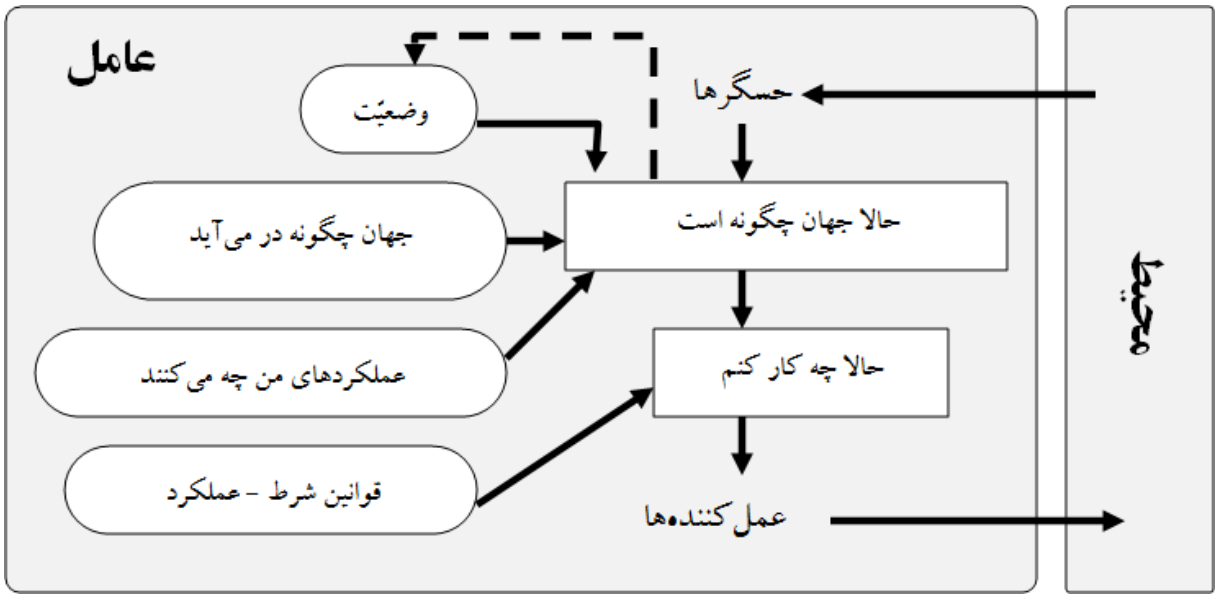
در صورتی که(اگر) وضعیّت، برابر با کثیف باشد، مکش را برمی‌گرداند و در غیر این صورت در صورتی که محل[4]، برابر با A باشد، راست را برمی‌گرداند و در صورتی که محل، برابر با B بود، چپ را برمی‌گرداند



تابع این عامل فقط در محیط‌های کاملاً قابل مشاهده موفّق است و اغلب، افتادن در حلقه‌های تکرار بینهایت(∞)، در محیط‌های قابل مشاهده به صورت جزئی، اجتناب ناپذیر است؛ در این صورت اگر عامل بتواند عملکردهای تصادفی را انجام دهد، ممکن است از حلقه‌ی تکرار بینهایت بگریزد.

۲- عامل‌هایی که وضعیّت جهان را حفظ می‌کنند

در جهان‌هایی که به صورت جزئی، قابل مشاهده هستند، اطّلاعات عامل به تنهایی کافی نمی‌باشند، لازم است که جریان تغییرات جهان را نگهداری نماییم و تکامل(تغییر)ها به طور مستقل از عامل یا سبب شده به وسیله‌ی عملکرد عامل می‌باشند.



---

۱ - در لغت یعنی: «بازتابی، عکس العملی، واکنشی»

۲ - در لغت یعنی: «جا، محل»

۳ - در لغت یعنی: «وضعیّت»

۴ - location

مثال:

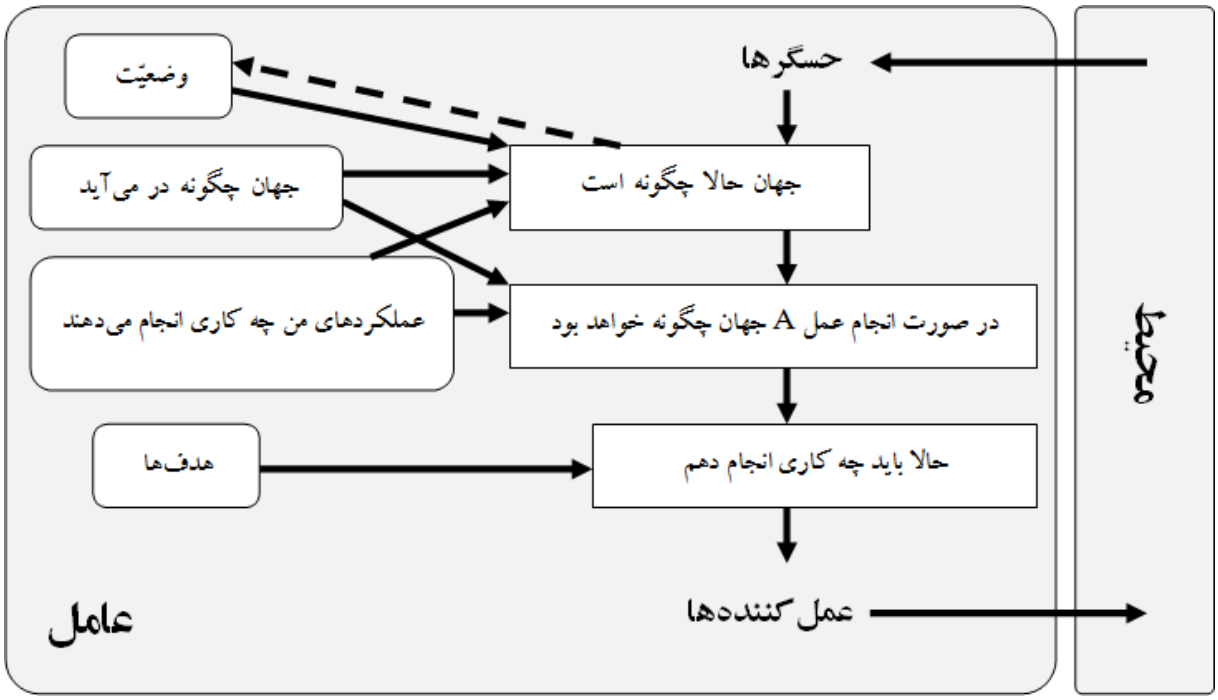
تابع Reflex-Vacuum-Agent([location,status]) یک عملکرد را برمی‌گرداند

متغیّرهای static[۱]: last_A, last_B, numbers دارای مقدار اوّلیّه‌ی ∞ می‌باشند

در صورتی که وضعیّت = کثیف باشد...

۳- عامل‌های هدف‌گرا

در این گونه عامل‌ها، وضعیّت و عملکردها نمی‌گویند که کجا برویم و به *اهداف* برای ساختن رشته‌ای از عملکردها(برنامه‌ریزی) نیازمندیم.



مطالبی در مورد عامل‌های هدف‌گرا: دانستن وضعیّت فعلی محیط همیشه کافی نمی‌باشد؛ عملکرد درست ممکن است همچنین وابسته به اینکه عامل در تلاش برای رسیدن به چه چیزی است، باشد. برای عامل‌های هدف‌گرا، عملکردهایی را که برای رسیدن به هدف‌ها کمک می‌کنند را انتخاب نمایید؛ جستجو و برنامه‌ریزی برای رسیدن به هدف‌ها مورد استفاده قرار می‌گیرند. استفاده از استدلال هدف‌گرا برای محیط‌های ترتیبی، خیلی مفید می‌باشد. مثال‌ها عبارتند از: بازی شطرنج، رانندگی

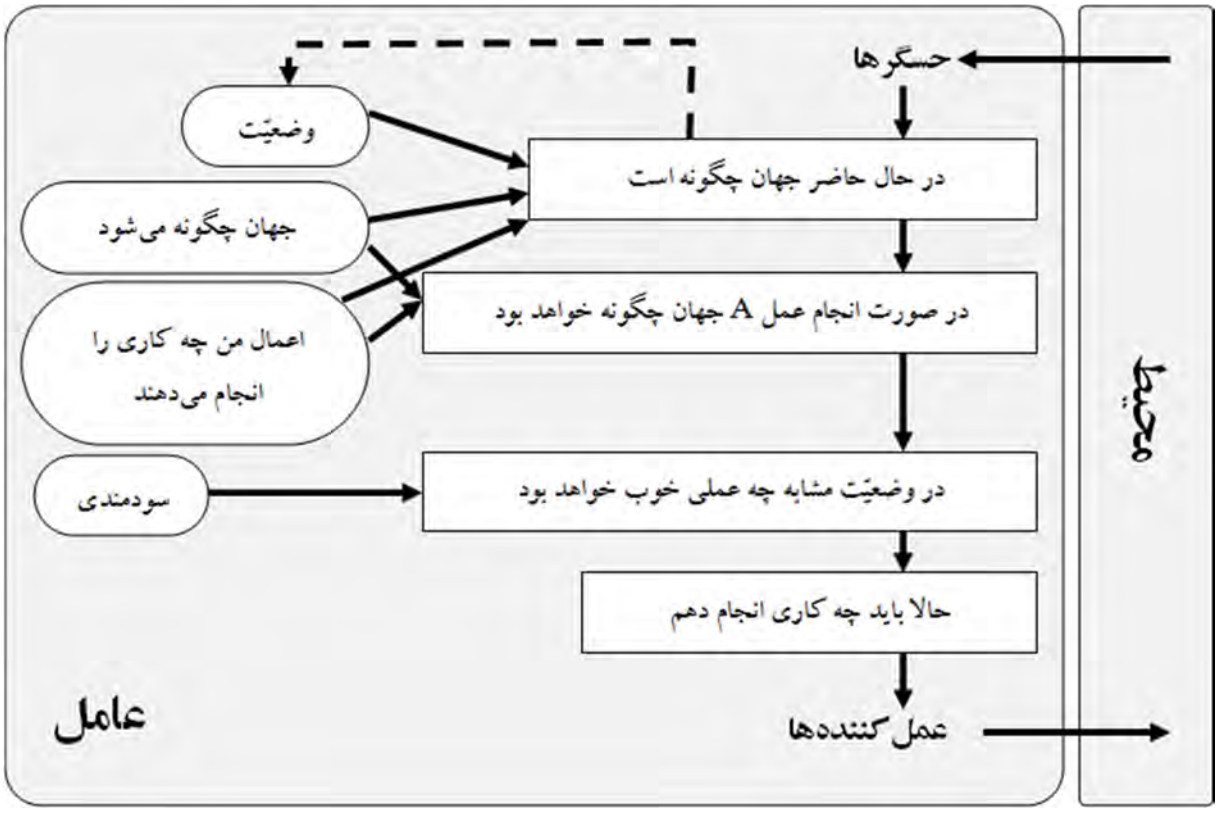
---

۱ - static variables: **یادآوری**- متغیّرهایی هستند که مقدار قبلی خود را در طول اجرای برنامه حفظ می‌کنند؛ در این مثال هرگاه که تابع Reflex-Vacuum-Agent باز هم فراخوانی(صدا زده) شود، *در ابتدای فراخوانی*، مقدارهای متغیّرهای last_A, last_B, numbers همان مقدارهایی هستند که در گذشته پس از خاتمه‌یافتن فراخوانی قبلی داشته‌اند.

تاکسی و خلبانی سفینه‌ی فضایی. عملکرد درست برای یک رشته‌ی ادراکی ارائه شده، وابسته به دانش عامل، وضعیّت جاری آن و چیزی که در حال حاضر در تلاش برای رسیدن به آن است، می‌باشد.

۴- عامل‌های سودمند

دانستن هدف‌ها ممکن است در محیط‌های با پیچیدگی بالا کافی نباشند؛ ممکن است چند رشته از عملکردها برای دسترسی به برخی اهداف لازم باشند؛ ممکن است نیازمند انتخاب میان عملکردها و رشته‌ی عملکردها باشیم؛ *سودمندی*، به هر وضعیّت، عددی حقیقی را نسبت می‌دهد، درجه‌ی برآورده‌سازی را بیان می‌کند و سبک و سنگینی میان موردهای متناقض را مشخّص می‌نماید؛ سودمندی می‌تواند تخمینی از هزینه، زمان، یا مقدار وابسته به نتایج مختلف باشد. سودمندی‌ها در محیط‌هایی که به صورت جزئی، قابل مشاهده‌اند؛ مثل محیط عامل رانندگی تاکسی؛ یا محیط‌های اتّفاقی(تصادفی)؛ مثل محیط قُمارْبازی[۱]، خیلی مفید می‌باشند. سودمندی‌ها برخی وقت‌ها یک چیز بحث‌انگیز در هوش مصنوعی می‌باشند. در مورد نتیجه‌های عملکرد فرض می‌کنیم نتایج می‌توانند به صورت خطّی با یک مقیاس یکسان مرتّب شده باشند؛ در این مورد طرّاح باید نتایج را به طور صریح و به صورت کمّی و کیفی ارزیابی نماید.



۱- gambling

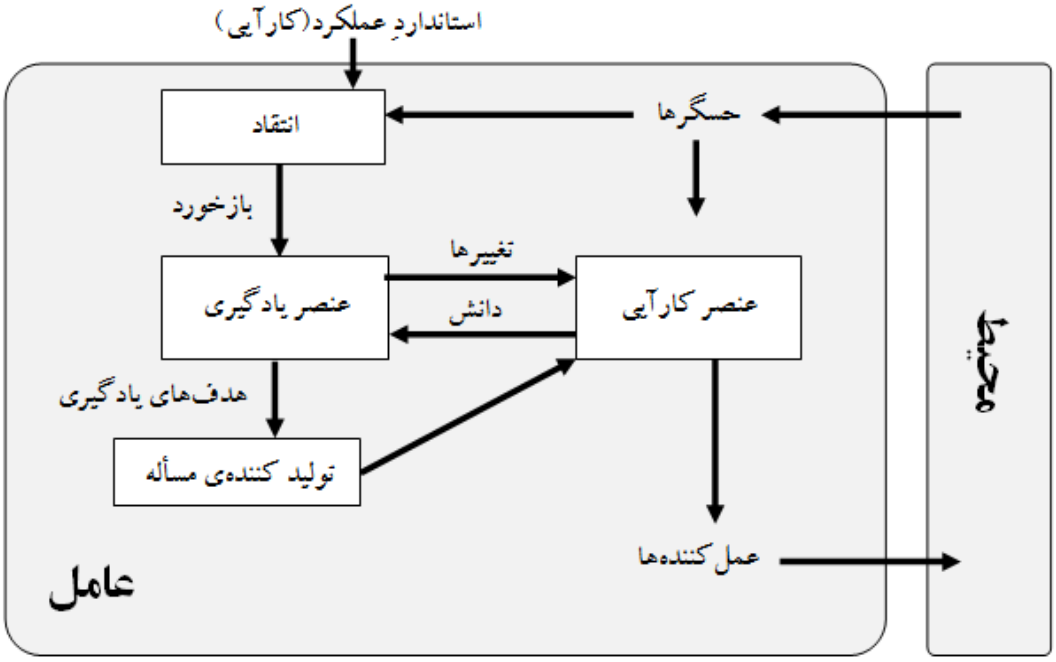
۵- عامل‌های یادگیرنده

تعریف- یک عامل یادگیرنده، عاملی است که عملکرد خود را با توجّه به مجموعه‌ای از عملکردها در طول زمان بهبود می‌بخشد.

با استفاده از یادگیری، عامل‌ها می‌توانند در محیط‌های ناشناخته کار کنند و بهتر از وقتی باشند که فقط از دانش اوّلیه استفاده می‌کنند. مهم‌ترین فرق، میان «عنصر یادگیری»[۱]، که بهبودها را به وجود می‌آورد و «عنصر کارآیی(عملکرد)»[۲]، که عملکردهای بیرونی را انتخاب می‌کند، می‌باشد.

عنصر یادگیری، درستی عنصر کارآیی را ارزیابی می‌نماید. انتقاد[۳]، جمع‌آوری بازخورد[۴] در مورد چگونگی عمل عامل می‌باشد. تولید کننده‌ی مسأله[۵]، عملکرد(آزمایش)های پیشنهادی(اکتشافی) را در نظر می‌گیرد.

اغلب یک عامل ممکن است نیاز به به‌روزرسانی برنامه‌ی عامل خود داشته باشد؛ برنامه‌نویسان ممکن است فهم کاملی از محیط نداشته باشند؛ محیط ممکن است در طول زمان تغییر نماید، در این مورد، برنامه‌نویسی با دست ممکن است خسته کننده باشد؛ یادگیری، در محیط‌های پیچیده ضروری می‌باشد.



۱- learning element
۲- performance element
۳- critic
۴- feedback
۵- problem generator

عامل‌های نرم‌افزاری

به صورت سافتبوت‌ها هم شناخته می‌شوند؛ در محیط‌های مصنوعی که براساس کامپیوترها و شبکه‌ها می‌باشند، وجود دارند؛ می‌توانند خیلی پیچیده با نیازمندی‌های زیاد به عامل باشند. وب جهانی[۱]، مثالی از عامل‌های نرم‌افزاری می‌باشد. در این عامل‌ها، محیط‌های طبیعی و مصنوعی ممکن است با هم ادغام شده باشند.

عامل‌های متحرّک[۲]

در عامل‌های متحرّک، برنامه‌ها می‌توانند از یک ماشین به ماشین دیگر بروند و در یک محیط اجرایی مستقل از سخت‌افزار یا پایگاه[۳] اجرا شوند؛ این عامل‌ها به محیط اجرایی عامل نیازمندند. کاربرد آنها در بازیابی اطّلاعات توزیع شده و مسیریابی شبکه‌ی مخابراتی می‌باشد.

عامل‌های اطّلاعاتی[۴]

رشد انفجاری اطّلاعات را مدیریّت می‌نمایند؛ اطّلاعات چند منبع توزیع شده را به کار می‌گیرند یا مقایسه(یکی)[۵] می‌نمایند. عامل‌های اطّلاعاتی می‌توانند ثابت یا متحرّک باشند؛ به عنوان مثال، اطّلاعات درون وب، متحرّک هستند، ولی یک سند واقعی (مثلاً یک سند کاغذی)، ثابت است.

۱- World Wide Web(WWW)
۲- mobile agents
۳- independent-platform
۴- information agents
۵- collate

چکیده‌ی مطلب‌های فصل سوّم

عامل‌ها روی *محیط‌ها* با استفاده از *عمل‌کننده‌ها* و *حسگرها* اثر می‌گذارند.

تابع عامل چگونکی عملکرد عامل را در همه‌ی شرایط تشریح می‌کند.

دردسرسازترین محیط‌ها، محیط‌های غیرقابل دسترس، غیرقطعی، پویا و پیوسته هستند.

یک عامل عقلانی بی‌عیب، کارآیی مورد انتظار را بیشینه می‌کند.

برنامه‌های عامل، برخی از توابع عامل را اجرا می‌کنند.

توضیحات PEAS محیط‌های کاری را تعریف می‌کند.

محیط‌ها در چند بعد طبقه‌بندی می‌شوند: قابل مشاهده؟ – قطعی؟ - دوره‌ای؟ – پویا؟ – گسسته؟ - تک عاملی؟.

چند معماری موجود عامل‌های پایه عبارتند از: عامل‌های بازتابی ساده – عامل‌هایی که وضعیّت جهان را حفظ می‌نمایند – عامل‌های هدف‌گرا - عامل‌های سودمند

عامل‌های بازتابی؛ یعنی، عامل‌های بازتابی ساده و عامل‌هایی که وضعیّت جهان را حفظ می‌نمایند، بلافاصله به ادراک‌ها پاسخ می‌دهند.

عامل‌های هدف‌گرا برای رسیدن به هدف یا هدف‌هایشان تلاش می‌کنند.

عامل‌های سودمند، تابع سودمندی خود را بیشینه(ماکزیمم) می‌کنند.

تمام عامل‌ها می‌توانند کارآیی خود را با استفاده از یادگیری افزایش دهند.

یادآوری یا تکمیل مطلب‌های فصل سوّم

تعریف– ادراک(دریافت، حس) را تعریف کنید.

جواب– ورودی‌های ادراکی عامل در هر لحظه‌ی داده شده است.[۱]

تعریف– دنباله(رشته)ی ادراکی را تعریف کنید.

جواب– تاریخچه‌ی کامل هر چیزی که عامل دریافت(ادراک، حس) کرده است.[۲]

تعریف– *«اصل قاب»*[۳] را تعریف کنید.

جواب– اصلی بیان کننده‌ی اینکه خصوصیّات معیّنی از وضعیّت، به عنوان نتیجه‌ای از یک عملکرد، بدون تغییر باقی می‌مانند.[۴]

تعریف– ☑ **کنکور سراسری مهندسی کامپیوتر سال ۸۹**– «مشکل قاب»[۵] چیست؟

جواب– چگونه نمایش دادن حقایق یا ارتباط‌هایی است که در برخی از اعمال، بدون تغییر باقی می‌مانند.[۱]

---

۱ – کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل دوّم، عامل‌های هوشمند(Intelligent Agents)، صفحه‌ی ۳۴

۲ – کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل دوّم، عامل‌های هوشمند(Intelligent Agents)، صفحه‌ی ۳۴

۳ – frame axiom

۴ – آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفُرنیا، شهر بِرکْلی(Berkeley) کشور آمریکا، پاییز سال ۱۹۹۱ میلادی

۵ – frame problem

تعریف- «جهانِ اِجبارپذیر»[۲] را تعریف کنید.

جواب- جهانی است که می‌تواند مجبور شود تا به یک وضعیّت شناخته شده برود، حتّا اگر آن وضعیّت قابل حس نباشد.[۳]

تعریف- «عامل خودمختار»[۴] را تعریف کنید.

جواب- یک عامل در صورتی خودمختار است که رفتارش به وسیله‌ی ادراک‌ها و تجربه‌اش تعیین شود و توانایی یادگیری و تطبیق داشته باشد، بدون اینکه فقط وابسته به دانشی که در درونش قرار داده شده است، باشد.[۵]

تعریف- محیط پویا را تعریف کنید.

جواب- در این گونه، در موقع انجام عملکردهای عامل، محیط تغییر می‌کند.[۶]

درست یا غلط- تغییرات، در یک محیط پویا(دینامیک)، الزاماً به خاطر عملکردهای عامل رُخ نمی‌دهند.

جواب- «درست» است.[۷]

تعریف- محیط کاملاً قابل مشاهده را تعریف کنید.

جواب- در این گونه، که در مقابل محیط قابل مشاهده به صورت جزئی قرار دارد، حسگرهای عامل، وضعیّت کامل محیط را در هر نقطه از زمان به عامل می‌دهند.[۸]

درست یا غلط- در یک محیط قابل مشاهده به صورت جزئی، اگر تعداد حسگرهای یک عامل را دوبرابر کنیم، به محیط کاملاً قابل مشاهده تبدیل می‌گردد.

---

۱ - مطلب‌های درس «روبوتیک هوشمند(Intelligent Robotics)» استاد، «مارِک ای. پِرکُوْسْکی(Marek A. Perkowski)»، استاد مهندسی برق دانشگاه ایالت پُرتْلَند(Portland) کشور آمریکا

۲ - coercible world

۳ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۱۹۹۳ میلادی

۴ - autonomous agent

۵ - آزمون درس «مبانی هوش مصنوعی» استاد، دکتر، «تارِکْ حِلْمی(Tarek Helmy)»، دانشکده‌ی علوم کامپیوتر و اطّلاعات دانشگاه پادشاه فَهَد کشور عربستان سعودی

۶ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۱۹۹۳ میلادی

۷ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبِرتی(Antonio Ruberti) دانشگاه ساپینزا(Sapienza)ی کشور ایتالیا، ۷ ژوئن سال ۲۰۱۲ میلادی

۸ - آزمون درس «مبانی هوش مصنوعی» استاد، دکتر، «تارک حلمی»، دانشکده‌ی علوم کامپیوتر و اطّلاعات دانشگاه پادشاه فهد کشور عربستان سعودی

جواب- «غلط» است؛ مشاهده‌پذیری، ربطی به تعداد حسگرها ندارد.[۱]

درست یا غلط- یک محیط کاملاً قابل مشاهده الزاماً قطعی است.

جواب- «غلط» است؛ این دو خصوصیّت باهم نامرتبط هستند.[۲]

درست یا غلط- یک محیط قابل مشاهده به صورت جزئی هیچوقت قطعی نیست.

جواب- «غلط» است؛ می‌تواند قطعی باشد، این وابسته به نوع(طبیعت یا ماهیّت) عملکردهایی است که عامل ممکن است انجام دهد.[۳]

تعریف- محیط دوره‌ای(اپیزودیک) را تعریف کنید.

جواب- در این گونه، که در مقابل محیط ترتیبی قرار دارد، عملکرد عامل در زمانی که عامل ادراک می‌کند، به دوره‌های اتمیک تقسیم می‌شود و سپس عامل یک عمل تک را انجام می‌دهد و انتخاب عمل، در هر دوره، فقط به همان دوره وابسته است.[۴]

سؤال- ☑کنکور سراسری مهندسی کامپیوتر سال ۸۷- خصوصیّات محیط بازی تَخته نَرد(backgammon)[۵] چیست؟

جواب- قابل مشاهده به طور کامل، چندعاملی، تصادفی، ترتیبی، ایستا و گسسته است.[۶]

درست یا غلط- عامل‌های بازتابی ساده به خوبی از عهده‌ی محیط‌های غیرقابل دسترس برمی‌آیند.

جواب- «غلط» است.[۷]

---

۱ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۲۱ فوریّه‌ی سال ۲۰۱۱ میلادی

۲ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۱۴ جولای سال ۲۰۱۱ میلادی

۳ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۱۸ ژانویّه‌ی سال ۲۰۱۲ میلادی

۴ - آزمون درس «مبانی هوش مصنوعی» استاد، دکتر، «تارک حلمی»، دانشکده‌ی علوم کامپیوتر و اطّلاعات دانشگاه پادشاه فهد کشور عربستان سعودی

۵ - برای آگاهی‌های بیش‌تر درباره‌ی این بازی، به فصل «تئوری بازی‌ها»ی همین کتاب الکترونیکی مراجعه کنید.

۶ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل دوّم، عامل‌های هوشمند(Intelligent Agents)، صفحه‌ی ۴۵

۷ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۱۹۹۶ میلادی

درست یا غلط- یک عامل بازتابی ساده از ادراک(دریافت) استفاده نمی‌کند.

جواب- «غلط» است؛ ادراک برای تعیین عملکرد بعدی ضروری است.

درست یا غلط- ☑مشابه کنکور سراسری فنّـآوری اطّلاعات سال ۸۷- یک عامل براساس مدل در پرداختن به مشاهده‌پذیری جزئی کمک می‌کند.

جواب- «درست» است.[۱]

مطلب- ☑کنکور سراسری مهندسی کامپیوتر سال‌های ۸۵ و ۸۶- در محیط‌های قابل مشاهده نیازی به نگهداری وضعیّت درونی نیست.[۲]

درست یا غلط- وضعیّت درونی هیچ کاربردی برای عاملی که در یک محیط قابل دسترس زندگی می‌کند، ندارد.

جواب- «غلط» است؛ وضعیّت درونی می‌تواند برای نگهداری نتایج محاسبات یا نگهداری نتایج یادگیری، مفید باشد (به عنوان مثال، یک پایگاه دانش).[۳]

تعریف- عامل عقلانی را تعریف کنید.

جواب- عاملی است که برای بیشینه(ماکزیمم) کردن معیار کارآیی مورد انتظارش عمل می‌کند.[۴]

درست یا غلط- عاملی که فقط اطّلاعات جزئی را در مورد وضعیّت حس می‌کند، نمی‌تواند کاملاً عقلانی باشد.

جواب- «غلط» است؛ عقلانیّت و همه چیزدانی دو مفهوم متفاوت هستند.[۵]

تعریف- عامل همه چیز دان(همه چیز آگاه)[۱] را تعریف کنید.

---

۱ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۱۱ ژانویّه‌ی سال ۲۰۱۰ میلادی

۲- مطالب درس «هوش مصنوعی» «جِفْ کولونه(Jeff Clune)»، اُستادیار(Assistant Professor) دانشگاه ایالت ویومینگ(Wyoming) کشور آمریکا، پاییز سال ۲۰۱۳ میلادی

۳ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، پاییز سال ۱۹۹۹ میلادی

۴ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اِچ. لاتِرِپْ(Richard H. Lathrop)»، دانشکده‌ی شهر اِروایِن دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۵ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۲۰۰۲ میلادی و تکلیف‌های خانگی(*HW* = *H*ome*W*ork) درس «آشنایی با هوش مصنوعی» استاد، «مَکْسْ وِلینگ(Max Welling)»، دانشکده‌ی شهر اروایِن دانشگاه ایالت کالیفورنیای کشور ایالات متّحده‌ی آمریکا، پاییز سال ۲۰۱۰ میلادی

جواب– این گونه عامل، نتیجه‌ی واقعی عملکردهایش را می‌داند و می‌تواند براین اساس عمل کند؛ امّا همه چیز دانی(همه چیز آگاهی) در حقیقت غیر ممکن است.[۲]

درست یا غلط– اگر یک محیط، کاملاً قابل مشاهده باشد، یک عامل بازتابی ساده می‌تواند به هر محرّکی پاسخ دهد.

جواب– «غلط» است؛ جواب عامل با جدول محرّک–پاسخ عامل، که در درون عامل پیاده‌سازی شده است، مطابق است.[۳]

درست یا غلط– یک عامل عقلانی بر همه‌ی عامل‌های غیرعقلانی دارای برتری کارآیی است؛ زیرا نتیجه‌ی واقعی عملکردهایش را می‌داند.

جواب– «غلط» است؛ در جهان‌های غیرقابل دسترس یا تصادفی، یک عامل عقلانی نمی‌تواند نتیجه‌های عملکردهایش را بداند؛ به علاوه اگر خوش‌شانس نباشد، بر یک عامل غیرعقلانی نمی‌تواند برتری کارآیی داشته باشد.[۴]

درست یا غلط– محیط‌های عملیّاتی‌ای(PEAS) وجود دارند که در آنها برخی «عامل‌های بازتابی خالص»[۵] می‌توانند به طور معقولانه رفتار نمایند.

جواب– «درست» است؛ در کل، هر محیط کاملاً قابل مشاهده می‌تواند به وسیله‌ی عامل بازتابی خالص به کار گرفته شود.[۶]

درست یا غلط– محیط‌های عملیّاتی‌ای(PEAS) وجود دارند که در آنها تمام عامل‌های بازتابی خالص به طور نامعقولانه رفتار می‌نمایند.

جواب– «درست» است.[۷]

درست یا غلط– محیط‌های عملیّاتی‌ای(PEAS) وجود دارند که در آنها هر عاملی عقلانی است.

---

۱ – omniscient agent

۲ – کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل دوّم، عامل‌های هوشمند(Intelligent Agents)، صفحه‌ی ۳۸

۳ – آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۲۰ ژوئن سال ۲۰۱۱ میلادی

۴ – آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، پاییز سال ۱۹۹۹ میلادی

۵ – pure reflex agents

۶ – آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۲۰۰۴ میلادی و شبیه تکلیف‌های خانگی درس «آشنایی با هوش مصنوعی» استاد، «مکس ولینگ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور ایالات متّحده‌ی آمریکا، پاییز سال ۲۰۱۰ میلادی

۷ – آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۲۰۰۴ میلادی

جواب- «درست» است؛ محیط عملیّاتی‌ای را در نظر بگیرید که در آن همه‌ی عملکردها (شامل «هیچ عملکرد»)، پاداش یکسان و برابری بگیرند.[۱]

سؤال- توضیح PEAS محیط کار «بازی فوتبال» را بنویسید و خصوصیّات محیط را هم مشخّص کنید.

جواب-

معیار کارآیی: بردن یا باختن | **محیط:** زمین فوتبال | **عمل کننده‌ها:** پاها، سر، قسمت بالای بدن | **حسگرها:** چشم‌ها، گوش‌ها

خصوصیّات محیط: قابل مشاهده به صورت جزئی، چندعاملی، تصادفی، ترتیبی، پویا، پیوسته[۲]

درست یا غلط- تصوّر کنید که یک عامل عملکردش را به طور یکنواخت، به طور تصادفی از مجموعه‌ای از عملکردهای ممکن انتخاب می‌کند. [در این صورت،] محیط عملیّاتی قطعی‌ای وجود دارد که در آن، این عامل، عقلانی است.

جواب- «درست» است؛ دوباره حالتی را در نظر بگیرید که در آن «همه‌ی عملکردها، همیشه پاداش یکسانی می‌گیرند».[۳]

درست یا غلط- عملی که به وسیله‌ی یک عامل عقلانی انجام می‌شود، همیشه تابعی قطعی از ادراک‌های جاری عامل خواهد بود.

جواب- «غلط» است؛ اوّل، تابع‌های غیرقطعی، در تعداد زیادی از محیط‌ها، نظیر محیط‌های چندعاملی مفید هستند. دوّم، عامل اغلب باید به تاریخچه‌ی ادراکی و دیگر منبع‌های اطّلاعات توجّه نماید.[۴]

درست یا غلط- ممکن است که یک عامل، در دو محیط کار متمایز، کاملاً عقلانی باشد.

جواب- «درست» است.[۵]

درست یا غلط- هر عاملی در یک محیط غیرقابل مشاهده، عقلانی است.

جواب- «غلط» است.[۱]

---

۱ - تکلیف‌های خانگی درس «آشنایی با هوش مصنوعی» استاد، «مکس ولینگ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور ایالات متّحده‌ی آمریکا، پاییز سال ۲۰۱۰ میلادی

۲ - تکلیف‌های خانگی درس «آشنایی با هوش مصنوعی» استاد، «مکس ولینگ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور ایالات متّحده‌ی آمریکا، پاییز سال ۲۰۱۰ میلادی

۳ - تکلیف‌های خانگی درس «آشنایی با هوش مصنوعی» استاد، «مکس ولینگ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور ایالات متّحده‌ی آمریکا، پاییز سال ۲۰۱۰ میلادی

۴ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «دَنْ کِلِیْنْ(Dan Klein)»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۲۰۰۶ میلادی

۵ - تکلیف‌های خانگی درس «آشنایی با هوش مصنوعی» استاد، «مکس ولینگ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور ایالات متّحده‌ی آمریکا، پاییز سال ۲۰۱۰ میلادی

تعریف- «تابع عامل» چیست؟

جواب- هر دنباله(رشته)ی داده شده را به یک عمل نگاشت می‌کند.[۲]

درست یا غلط- ورودی برنامه‌ی یک عامل مانند ورودی تابع آن عامل است.

جواب- «**غلط**» **است؛** برنامه‌ی عامل، ادراک جاری را می‌گیرد، در حالیکه تابع عامل، تاریخچه‌ی ادراکی را می‌گیرد.[۳]

درست یا غلط- هر تابع عامل به وسیله‌ی ترکیب برنامه یا ماشین قابل پیاده‌سازی است.

جواب- «**غلط**» **است؛** توابعی ممکن است وجود داشته باشند که قابل محاسبه باشند، امّا در هیچ مقدار زمانی عملی و بر روی هیچ کامپیوتر ممکنی قابل پیاده‌سازی نباشند.[۴]

تعریف- انتزاع(تجرید، چکیدگی)[۵] را تعریف کنید.

جواب- فرآیند برداشتن جزئیّات از ارائه(بازنمائی)[۶] است.[۷]

---

۱ - تکلیف‌های خانگی درس «آشنایی با هوش مصنوعی» استاد، «مکس ولینگ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور ایالات متّحده‌ی آمریکا، پاییز سال ۲۰۱۰ میلادی

۲ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۳ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۲۰۰۴ میلادی و تکلیف‌های خانگی درس «آشنایی با هوش مصنوعی» استاد، «مکس ولینگ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور ایالات متّحده‌ی آمریکا، پاییز سال ۲۰۱۰ میلادی

۴ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۲۰۰۴ میلادی و تکلیف‌های خانگی درس «آشنایی با هوش مصنوعی» استاد، «مکس ولینگ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور ایالات متّحده‌ی آمریکا، پاییز سال ۲۰۱۰ میلادی

۵ - abstraction

۶ - representation

۷ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

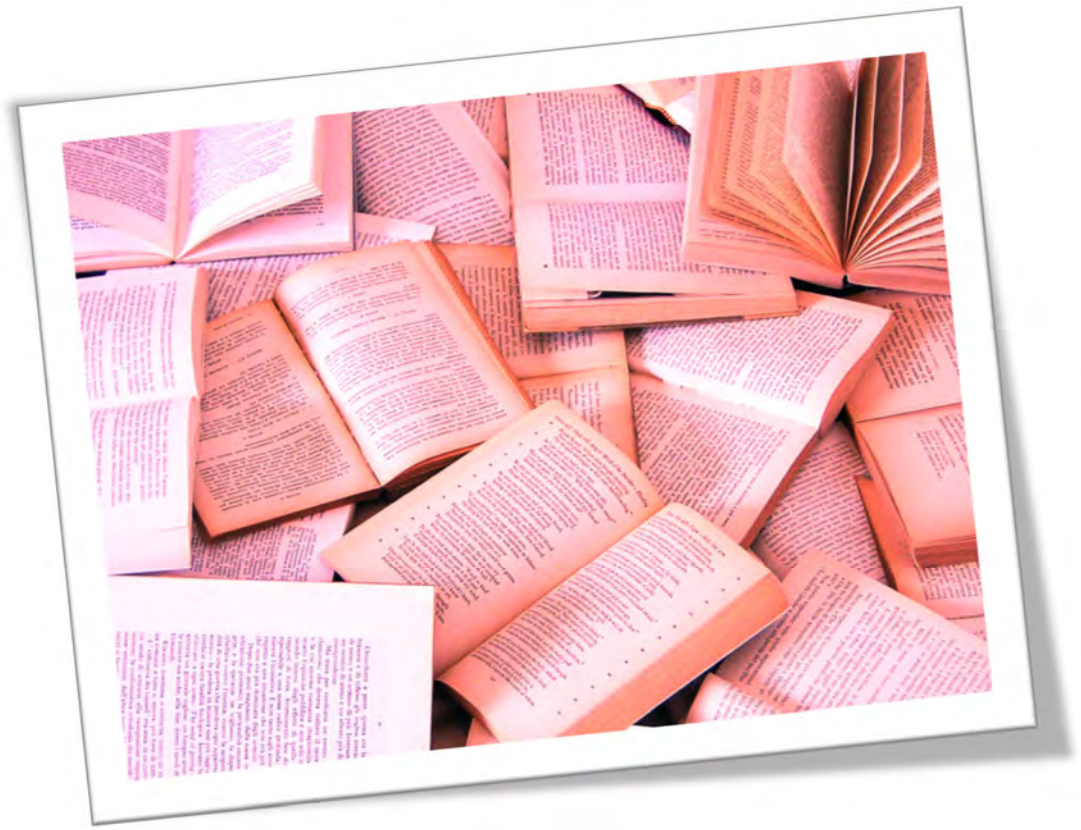
فصل چهارم

۱ حلّ مسأله و جستجو ۲

---

۱ - شکل، مربوط به داستان کلاغ تشنه‌ای است که در هوایی گرم به دنبال آب می‌گشت. کلاغ در گوشه‌ای کوزه‌ای محتوی آب پیدا کرد؛ ولی به دلیل اینکه سطح آب درون کوزه پایین بود، هر چقدر که تلاش کرد تا سر و گردن خود را به درون کوزه برده و از آب درون کوزه بنوشد، نتوانست؛ کلاغ همچنین نتوانست با استفاده از قدرت خود، کوزه را روی زمین بخواباند و آب خارج شده از آن را بنوشد؛ به دلیل خستگی زیاد، دیگر دنبال آب هم نمی‌توانست بگردد؛ کلاغ سرانجام بعد از اینکه راه حل‌های گوناگون را چنانچه گفته شد، برّرسی (جستجو) کرد و جواب نگرفت، مسأله‌ای که برایش به وجود آمده بود را با استفاده از این روش(راه حل) حل کرد: آنقدر خرده‌سنگ(سنگ‌ریزه) به درون کوزه‌ی آب ریخت تا سطح آب درون کوزه بالا آمد و توانست از آب درون کوزه بنوشد.

۲ - problem solving and search



فهرست برخی از عنوان‌های نوشته‌ها

حلّ مسأله به صورت جستجو

انواع مسأله

مسأله‌های تک‌حالته

مسأله‌های چندحالته

مسأله‌های احتمالی

مسأله‌های اکتشافی

مسأله‌های خوب تعریف شده

عملکردها(عملگرها، عملیّات)

تابع جانشین

فضای حالت

مسیر

آزمایش(آزمون) هدف

الگوریتم عامل ساده‌ی حلّ مسأله

پیاده‌سازی الگوریتم‌های جستجو در حالت کلّی

فرمول‌بندی مسأله‌ی تک‌حالته

درخت جستجو

یک الگوریتم جستجوی درختی ساده

الگوریتم جستجوی درختی(به بیان دیگر و کامل‌تر نسبت به الگوریتم قبلی)

استراتژی(روش)های جستجو

جستجوی ناآگاهانه(کور)

روش‌های جستجوی ناآگاهانه

جستجوی اوّلْ سطح

جستجوی با هزینه‌ی یکسان

جستجوی اوّلْ عمق

جستجوی با عمق محدود شده

جستجوی عمیق شونده‌ی تکراری

جریان معکوس(برگشت به عقب)

جستجوی دوطرفه

مقایسه‌ی روش‌های جستجو

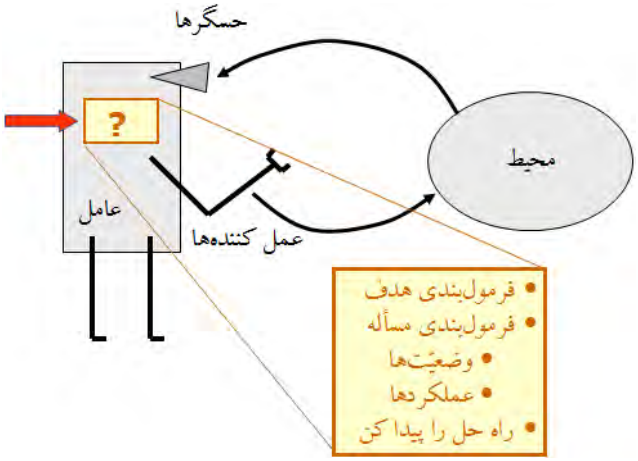
حالت(وضعیّت)های تکرار شده

الگوریتم جستجوی گراف

عامل‌های حل کننده‌ی مسأله

یادآوری: یک عامل هدف‌گرا می‌تواند برّرسی کند که چه کاری را می‌تواند انجام دهد و عملکردهایی که به نتیجه می‌رسند را انتخاب نماید؛ برنامه‌ی عامل، از درک‌ها و هدف، به صورت ورودی استفاده می‌کند.

در اینجا یک نوع عامل هدف‌گرا به نام عامل حل کننده‌ی مسأله[۱] را برّرسی می‌کنیم.



شکل بالا– عامل حل کننده‌ی مسأله

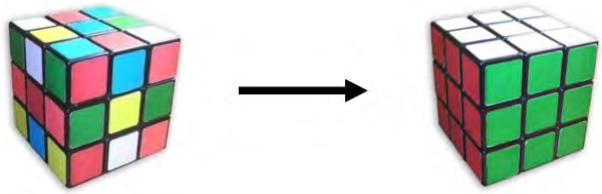
یک عامل حل کننده‌ی مسأله، برای پیداکردن یک رشته از عملکردها که به هدف می‌رسند، تلاش می‌کند؛ به عنوان مثال، چه رشته از حرکت‌ها یک بازی مکعّب رابیک[۲] را حل می‌کنند؟؛ چگونه من از دانشگاه فلوریدای جنوبی[۱] با خودرو به سمت شهر

---

۱ - problem-solving agent

۲ - Rubik's cube، یک پازل به شکل یک مکعّب پلاستیکی که با مربّع‌هایی با چند رنگ پوشانده شده است، می‌باشد؛ هر بازیگر تلاش می‌کند مربّع‌ها را تغییر دهد تا این که همه‌ی مربّع‌های هر وجه، دارای یک رنگ بشوند (Babylon> Concise Oxford English Dictionary). در زیر تصویری از یک وضعیّت شروع بازی مکعّب رابیک(شکل سمت چپ) و وضعیّت پایانی این بازی(شکل سمت راست) را مشاهده می‌نمایید:

لیورمور[۲] رانندگی نمایم؟؛ چگونه می‌توانم اجزای روی یک تَراشه[۳] را بچینم؟؛ کدام رشته از عملیّات، یک روبوت را در یک فضا حرکت می‌دهد؟.



این بازی به وسیله‌ی ارنُو رُوبیک(Erno Rubik)، متولّد ۱۹۴۴ میلادی، طرّاح، مجسّمه‌ساز و استاد معماری از کشور مَجارِستان به وجود آمده است.

(http://en.wikipedia.org/wiki/Ern%C5%91_Rubik)

تصویری از وی:

۱ - *USF* = *U*niversity of *S*outh *F*lorida؛ فلوریدا نام ایالتی در جنوب کشور ایالات متّحده‌ی آمریکا است. (<Babylon Babylon English)

۲ - Livermore، نام شهری در غرب ایالت کالیفورنیای کشور ایالات متّحده‌ی آمریکا می‌باشد.

۳ - chip یا *I*ntegrated *C*ircuit(*IC*)، در علم الکترونیک، یک مدار الکترونیکی کوچک شده است که روی سطح یک لایه از جنس نیم‌رسانا(semiconductor) قرار گرفته است.

(http://en.wikipedia.org/wiki/Integrated_circuit)

چهار گام(مرحله‌ی) کلّی در حلّ مسأله

عبارتند از:

۱- **فرمول‌بندی(فرمولاسیون، فرمول‌سازی)[۱] هدف مسأله:** *وضعّیت‌های موفّق جهان کدامند؟*.

برای فرمول‌بندی هدف مسأله این موردها را در نظر می‌گیریم: وضعّیت هدف چیست؟؛ ویژگی‌های مهمّ وضعّیت هدف چه می‌باشند؟؛ چگونه عامل می‌فهمد که به هدف رسیده است؟، در این مورد برّرسی می‌کند که آیا چند وضعّیت پایانی ممکن وجود دارد؟ و آیا آنها با هم برابرند یا برخی بهترند؟.

۲- **فرمول‌بندی مسأله:** *چه وضعّیت‌ها و عملکردهایی برای رسیدن به هدف باید مورد توجّه قرار گیرند؟*.

برای فرمول‌بندی مسأله این موردها را در نظر می‌گیریم: وضعّیت‌های ممکن وابسته به دنیا برای حلّ مسأله کدامند؟؛ چه اطّلاعاتی برای عامل در دسترس می‌باشند؟؛ رفتن عامل از یک وضعّیت به وضعّیت دیگر چگونه می‌تواند باشد؟.

۳- **جستجو:**

تعریف: پردازه‌ی به صورت ترتیبی پرداختن(رسیدگی کردن) به عملکردها، برای پیداکردن رشته‌ای از عملیّات که ما را از شروع به هدف می‌رسانند، جستجو نام دارد. یک الگوریتم جستجو یک رشته از عملیّات که برای عامل اجرا می‌شوند را برمی‌گرداند.

تعریف جستجوی برون خطّی[۲]: تمام راه حل را در موقع شروع حتّا قبل از آنکه شما شروع به اجرای راه حل کنید، توسعه(گسترش) می‌دهد.

در این فصل تمرکز ما بر روی روش‌های برون خطّی، برای مسأله‌های جستجوی کلاسیک است (توجّه کنید که قطعی و کاملاً قابل مشاهده هستند).

برخی از مسأله‌های جستجوی کلاسیک

مسأله‌های سرگرمی: برای مطالعه به صورت مثال یا برای مقایسه‌ی الگوریتم‌ها مفید می‌باشند؛ مثل: پازلِ ۸–تایی[۳]، دنیای جاروبرقی، مکعّب رابیک و N–وزیر[۱].

---

۱ - formulation

۲ - offline

۳ - 8–puzzle؛ **توضیح**- صفحه‌ای با نه خانه‌ی ۳×۳ و دارای ۸ کاشی(tile یا خانه) است که به صورت نامرتّب قرار گرفته‌اند و ما باید با حرکت دادن کاشی‌ها، کاشی‌ها را در جای درست قرار داده و مرتّب نماییم. در شکل زیر، به عنوان نمونه، دو وضعیت اوّلیّه و هدف(نهائی یا پایانی) نشان داده شده است:

مسأله‌های دنیای واقعی: معمولاً کار زیادی می‌برند، امّا جواب معمولاً جالب می‌باشد؛ مثل: مسیریابی، مسأله‌ی مسافرت شخص(فروشنده‌ی) دوره‌گرد[۲]، طرح VLSI[۳]، و جستجو در اینترنت.

تعریف جستجوی برخط(لحظه‌ای یا آنی)[۴]: دانش را در زمان اجرا جمع‌آوری می‌کند؛ برای مسأله‌های اکتشافی، که در آینده به آنها خواهیم پرداخت، لازم است و برای مسأله‌های غیرقطعی و قابل مشاهده به صورت جزئی می‌تواند مفید باشد.

8	2	
3	4	7
5	1	6

شکل بالا– وضعیّت اوّلیّه

1	2	3
4	5	6
7	8	

شکل بالا– وضعیّت هدف(نهائی)

پازل‌های n– تایی دیگر، مثلاً پازل ۱۵–تایی و پازل ۲۴–تایی هم به صورتی مشابه می‌باشند.

۱ – queen:

صورت مسأله‌ی N–وزیر(N-queens) به این صورت است: n وزیر را در یک صفحه‌ی شطرنج n×n بدون اینکه هیچ یک از دو وزیر در سطر یا ستون همانند قرار گیرند یا به طور مورّب همانند قرار گیرند، قرار دهید.

۲ – *T*raveling *S*alesperson *P*roblem(*TSP*)؛ همان طور که در گذشته هم بیان کردیم، در این مسأله، n شهر به وسیله‌ی جادّه به هم متّصل می‌باشند و مسأله، پیدا کردن یک مسیر که از تمام شهرها عبور کند و تا حدّ امکان کوتاه‌ترین مسیر باشد، است.

۳ – *VLSI*، کوتاه شده‌ی *V*ery *L*arge *S*cale *I*ntegration می‌باشد و تعداد زیادی ترانزیستور(transistor؛ وسیله‌ای به حالت جامد برای تقویّت، کنترل و تولید سیگنال‌های الکتریکی است. (Babylon> Britannica.com)؛ به عنوان تعریفی دیگر، ترانزیستور، وسیله‌ای از جنس نیم‌رسانا است که برای تقویّت و تغییر مسیر سیگنال‌ها استفاده می‌شود. (http://en.wikipedia.org/wiki/Transistor)) است که روی یک *صفحه‌ی* کوچک قرار گرفته‌اند.

۴ – online

جستجو معمولاً به صورت «برون خطّی» انجام می‌شود. در این مورد نکته‌ای که باید در نظر بگیریم، این است که فرض بر این است که محیط به صورت استاتیک(ثابت) می‌باشد، همچنین فرض شده که محیط گسسته می‌باشد، معمولاً محیط به صورت قطعی می‌باشد.

۴- اجرا: **تعریف**- ارائه‌ی راه حلّ انجام عملکردها

جستجو

چرا روش‌های جستجو را بررّسی می‌نماییم؟

روش‌های جستجو روش‌هایی مهم برای حلّ بسیاری از مسأله‌ها می‌باشند و الگوریتم‌های جستجو پایه‌ای برای بهینه‌سازی[۱] و برنامه‌ریزی می‌باشند.

حلّ مسأله به صورت جستجو

بسیاری از مسأله‌ها می‌توانند به صورت رسیدن به یک حالت هدف[۲] از یک نقطه‌ی شروع دیده شوند. *فضای حالت*[۳]، *مسأله و راه حل‌های ممکن آن را به یک صورت رسمی‌تری تعریف می‌نماید؛* معمولاً مسأله‌ها باید به صورت تجزیّه شده(مجزّا)[۴] باشند. عملکردهای عامل، وضعیّت را تغییر می‌دهد و فضای حالت را برای یک راه حل، جستجو(ملاقات) می‌نماید. اطّلاعات، در مورد مسأله‌ی معیّن یا با دامنه‌ی کلّی(عمومی) می‌توانند برای بهبود جستجو استفاده شوند. این اطّلاعات عبارتند از: تجربه از موردهای قبلی مسأله؛ روش‌های بیان شده به صورت اکتشافات(ابتکارها)؛ نوع‌های ساده‌تر مسأله و محدودیّت‌های وضعیّت‌های مشخّص(معیّن) مسأله.

---

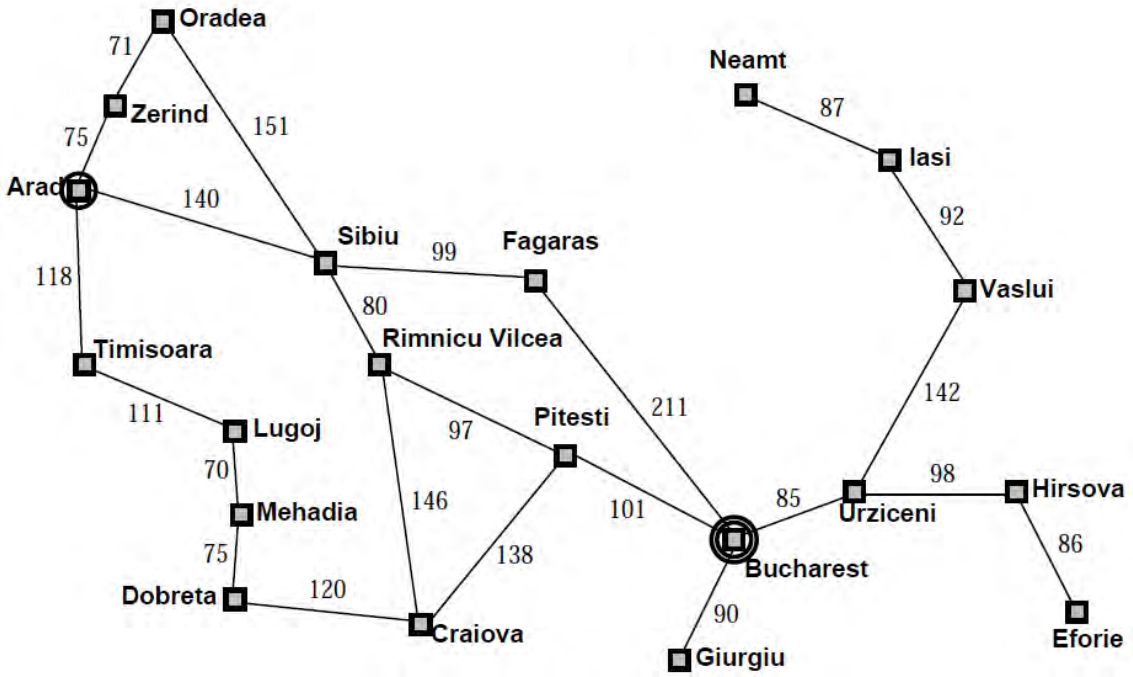
۱ - optimization: در ریّاضیّات، علم کامپیوتر و علم اقتصاد، ارایه‌ی بهترین راه حل از میان راه حل‌های موجود می‌باشد.
(http://en.wikipedia.org/wiki/Optimization_(mathematics))

۲ - goal state

۳ - state space

۴ - abstracted

در مورد وضعیّت به این نکته توجّه کنید که ما معمولاً در مورد وضعیّتی که یک عامل در آن می‌باشد، صحبت می‌کنیم؛ به عنوان مثال، در دنیای جاروبرقی، وضعیّت (۰و۰)، تمیز می‌باشد؛ در مسأله‌ی کشور رومانی[۱]، در t=0 در شهر بُخارِسْت[۲] هستیم:



شکل بالا– یک نقشه‌ی راه ساده شده از قسمتی از کشور رومانی

---

۱ – این مسأله در ادامه‌ی همین فصل توضیح داده شده است؛ صورت این مسأله به این صورت است که در یک تعطیلی در کشور رومانی؛ در حال حاضر در شهر آراد(Arad) هستیم و هواپیما فردا شهر آراد را به مقصد شهر بخارست ترک می‌کند و ما می‌خواهیم در شهر بخارست باشیم.

۲ – Bucharest

Sibiu	سیبیو	Mehadia	مِهادیا	Fagaras	فاگاراس (فَگَراژْ)	Arad	آراد
Timisoara	تیمیسوارا	Neamt	نِمْتْ	Giurgiu	جیورجیو	Bucharest	بخارست
Urziceni	اِرزیچِنی	Oradea	اُرادیا	Hirsova	هِرْسُوْا	Craiova	کِرایُوْا
Vaslui	واسْلُوْی	Pitesti	پیتِسْتی	Iasi	یاسی	Dobreta	دابْرِیْدا
Zerind	زِریند	Rimnicu Vilcea (Rimnicu Vilc)	ریمنیکو ویلسیا (ویلْشا) یا ریمنیکو ویلک	Lugoj	لوگُوْجْ	Eforie	اِفُری

جدول بالا– نام انگلیسی و فارسی شهرهایی که در نقشه‌ی بالا آمده‌اند.

و در بازی مکعّب رابیک، وضعیّت، نظم فعلی مکعّب(چگونگی قرار گرفتن مربّع‌های روی مکعّب، در حال حاضر) می‌باشد.

انواع مسأله

تعریف مسأله‌های تک‌حالته[۱]: در صورتی که مسأله قطعی و کاملاً قابل مشاهده باشد، در این صورت مسأله تک‌حالته خواهد بود؛

در این حالت عامل دقیقاً می‌داند در کدام حالت قرار خواهد گرفت و راه حل، به صورت ترتیبی است؛ در این حالت، جهان در دسترس می‌باشد و نتیجه‌ی(اثر) عملکردها را می‌دانیم و عامل وضعیّتی که بعد از یک رشته از عملیّات در آن خواهد بود را می‌داند.

تعریف مسأله‌های چندحالته[۲]: در صورتی که مسأله غیرقابل مشاهده باشد، در این صورت مسأله تطبیقی(ترکیبی[۳]) خواهد بود؛

در این گونه، عامل ممکن است تصمیمی در مورد کجا قرار گرفتن نداشته باشد؛ و راه حل، در صورت وجود، ترکیبی می‌باشد. در مسأله‌های چندحالته جهان فقط تاحدودی در دسترس می‌باشد و عامل به چند وضعیّت ممکن توجّه دارد.

---

۱ – single-state problems

۲ – multiple-state problems

۳ – conformant

تعریف مسأله‌های احتمالی[۱]: در صورتی که مسأله احتمالی(غیرقطعی[۲]) و یا اندکی قابل مشاهده[۳] باشد، در نتیجه مسأله‌ی احتمال می‌باشد.

در مسأله‌های احتمالی در طول اجرا به مشاهده و حس کردن نیازمندیم؛ همچنین به درخت عملیّات هم نیاز داریم.

تعریف مسأله‌های اکتشافی[۴]: در صورتی که مسأله دارای فضای حالت ناشناخته باشد، در نتیجه مسأله اکتشافی می‌باشد.

در مسأله‌های اکتشافی، عامل نتایج عملکردش را نمی‌داند و آزمایش‌های عامل برای کشف وضعیّت‌های جهان و اثرات عملکردها می‌باشد.

مسأله‌های خوب تعریف شده

تعریف- در مسأله‌های خوب تعریف شده، وضعیّت اوّلیّه، نقطه‌ی شروعی است که عامل از آن شروع می‌کند؛ مثلاً در مورد مسأله‌ی کشور رومانی، وضعیّت اوّلیّه، شهر آراد می‌باشد.

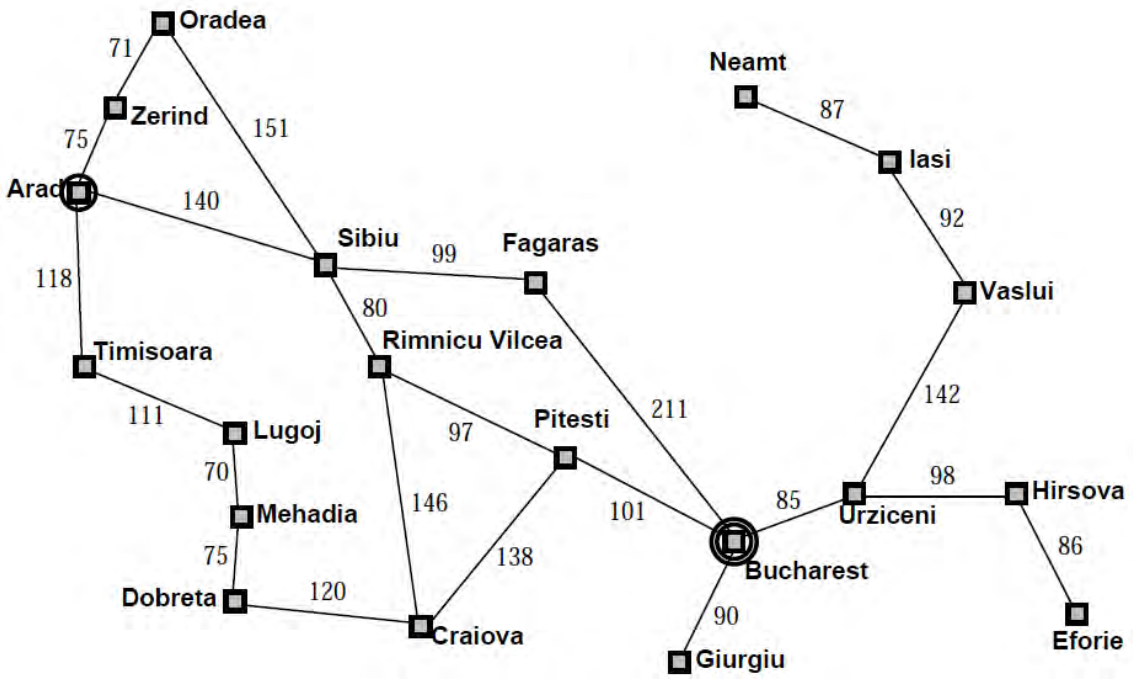
تعریف- عملکردها(عملگرها، عملیّات)، می‌گویند که عامل چه عملکردهایی را می‌تواند داشته باشد؟؛ مثلاً در مورد جاروبرقی، [عملکردها،] چپ، راست، بالا، پایین، مکش و هیچ کار می‌تواند باشد.

عملکردها وابسته به وضعیّت‌ها، توانایی عامل، و ویژگی‌های محیط می‌باشند. انتخاب وضعیّت‌ها و عملگرهای نامناسب می‌تواند یک مسأله‌ی قابل حل را به یک مسأله‌ی غیرقابل حل تبدیل نماید.

تعریف- تابع جانشین[۵]، برای یک وضعیّت داده شده، *یک مجموعه از جفت‌های «عملکرد –وضعیّت جدید» را برمی‌گرداند* و به ما می‌گوید که برای یک وضعیّت داده شده چه کارهایی را می‌توانیم انجام دهیم و در چه جاهایی[آن کارها] مقدّم(دارای اولویّت) هستند.

در یک جهان قطعی، هر عملکرد با یک وضعیّت تک(منفرد) جفت شده است؛ مثلاً در مسأله‌ی کشور رومانی:

۱- contingency problems
۲- nondeterministic
۳- partially observable
۴- exploration problems
۵- successor function



S(Arad)={<Arad→Zerind, Zerind> , …}

در جهان‌های اتّفاقی(تصادفی)، یک عملکرد شاید با تعدادی از وضعیّت‌ها جفت شده باشد.

تعریف- فضای حالت، مجموعه‌ای از همه‌ی وضعیّت‌هایی که از وضعیّت اوّلیّه، به وسیله‌ی رشته‌ای از عملکردها قابل دسترسی می‌باشند، است و

تعریف- مسیر، رشته‌ای از عملکردها که راهنمایی کننده از یک وضعیّت به وضعیّت دیگر است، می‌باشد.

تعریف- آزمایش(آزمون) هدف[۱]، تشخیص می‌دهد که آیا یک وضعیّت ارائه شده، یک وضعیّت هدف می‌باشد یا نه؟؛ مثلاً در مسأله‌ی کشور رومانی، در شهر بخارست بودن، وضعیّت هدف می‌باشد. و

تعریف- راه حل، مسیری از وضعیّت اوّلیّه به یک وضعیّت هدف می‌باشد.

۱- goal test

هزینه‌ی مسیر[۱]

تعریف– *هزینه‌ی مسیر، هزینه‌ی عامل را برای اجرای عملکردها(عملّیات) در یک مسیر تعیین می‌نماید* و مجموع هزینه‌های عملکردهای تکی، در یک مسیر می‌باشد؛

تعریف– به مجموع هزینه، برای هر عملکرد، هزینه‌ی گام(مرحله)[۲] گفته می‌شود.

فرض می‌کنیم که هزینه‌ی گام‌ها [مقدارهایی] غیرمنفی(بزرگ‌تر یا مساوی با صفر(≥۰)) هستند.

فضای حالت

تعریف– ترکیب حالت‌های مسأله و توابع جانشین(راه‌های رسیدن به وضعیّت‌ها)، منجر به مفهوم فضای حالت می‌شود و گِرافِ[۳]ـی است که همه‌ی وضعیّت‌های ممکن جهان و انتقال‌های میان آنها(وضعیّت‌های ممکن جهان) را ارائه می‌نماید.

اغلب در مورد اندازه‌ی این فضاها به صورت یک معیار سختی مسأله صحبت می‌کنیم؛ به عنوان مثال، اگر سرعت ۱۰۰۰۰۰۰۰(ده میلیون) وضعیّت در ثانیه را داشته باشیم: در پازل ۸–تایی، فضای حالت برابر است با: $\frac{9!}{2}$[۴] = ۱۸۱۴۴۰ حالت، که فضای حالتی ساده است و با سرعت بیان شده، ۰٫۱۸(هیجده صدم) ثانیه طول می‌کشد؛ در مورد پازل ۱۵–تایی، فضای حالت برابر است با: $۱۰^{۱۲}$ × ۰٫۶۵(عدد اعشاری، شصت و پنج صدم است) حالت، که تقریباً ساده است و ۶(شش) روز طول می‌کشد؛ در مورد پازل ۲۴–تایی: $۱۰^{۲۵}$ × ۰٫۵(عدد اعشاری نیم است) وضعیّت داریم، که فضای حالتی پیچیده است و ۱۲ میلیارد[۵] سال طول می‌کشد.

عملیّات حلّ مسأله

تعریف– *هزینه‌ی جستجو، زمان و حافظه‌ی لازم برای محاسبه‌ی یک راه حل است* و داریم:

---

۱ – path cost

۲ – step cost

۳ – graph

۴ – توجّه کنید که !۹ وضعیّت داریم، امّا می‌توان ثابت کرد فقط نیمی از این وضعیّت‌ها؛ یعنی، !۹/۲ از هر وضعیّت داده شده قابل دسترس هستند!؛ شبیه این مطلب در مورد پازل‌های ۱۵–تایی و ۲۴–تایی هم برقرار است!.

۵ – billion، که برابر است با: ۱۰۰۰۰۰۰۰۰(یک میلیارد)

مجموع هزینه = هزینه‌ی مسیر + هزینه‌ی جستجو

الگوریتم عامل ساده‌ی حلّ مسأله

شکل محدود شده برای عامل کلّی دارای الگوریتم زیر می‌باشد:

الگوریتم[۱]

تابع Simple[۲]-Problem-Solving[۳]-Agent(percept)، [یک دریافت(ادراک) را می گیرد] و یک عملکرد را برمی گرداند

متغیّرهای static:

seq، که یک رشته از عملکردها است و در ابتدا دارای مقدار تُهی(خالی یا هیچ)[۴] می‌باشد

state، توصیف وضعیّت فعلی جهان

goal، یک هدف که به صورت اوّلیّه دارای مقدار NULL[۱] می‌باشد

---

۱ – در این کتاب به افتخار دانشمند بزرگ کشورمان، «محمّدِ بْنِ موسا خوارَزمی(Muhammad ibn Musa al-Khwarizmi)»، تقریباً ۸۵۰ – ۷۸۰ میلادی که کلمه‌ی «الگوریتم» از نام او گرفته شده است، از تصویر وی به عنوان نمادی برای «الگوریتم»ها استفاده شده است؛ در ضمن، خوارزمی به خاطر نوشته‌هایی که در جبر دارد، معروف است.

تصویر استفاده شده، ویرایش شده‌ی تصویر زیر است:

تصویر بالا تمبری است که در کشور شُوْرَوی سابق(Soviet، روسیّه‌ی کنونی) در تاریخ روز ششم ماه سپتامبر سال ۱۹۸۳ میلادی که تقریباً برابر با ۱۲۰۰ اُمین سالگرد تولّد خوارزمی بوده است، پخش شده است. منبع:

http://en.wikipedia.org/wiki/Mu%E1%B8%A5ammad_ibn_M%C5%ABs%C4%81_al-Khw%C4%81rizm%C4%AB

۲ – در لغت به معنی «ساده، آسان» می‌باشد.

۳ – در اینجا؛ یعنی، «حلّ، حل کننده»

۴ – empty یا NULL

problem، یک فرمول‌بندی مسأله

state←Update[۲]-State(state,percept)

در صورتی که seq خالی است، کارهای زیر را انجام بده

goal←Formulate-Goal(state)

problem ← Formulate-Problem(state,goal)

seq ← Search(problem)

(پایان شرط)

action ← Recommendation[۳](seq,state)

seq ← Remainder[۴](seq,state)

action را برگردان

پایان الگوریتم

پیاده‌سازی الگوریتم‌های جستجو در حالت کلّی

تعریف: *لیستی از گره‌ها که تولید شده‌اند، امّا هنوز توسعه داده نشده‌اند را حاشیه(fringe)[۵] می‌نامیم.*

نکته:

توسعه‌دادن یک گره، به معنی به وجود آوردن همه‌ی فرزندان[۶] آن گره است.

---

۱ – در لغت به معنی «تُهی، خالی، هیچ» می‌باشد.

۲ – در لغت به معنی «به هنگام کردن، به‌روز کردن» می‌باشد.

۳ – در لغت به معنی «پیشنهاد» است.

۴ – در لغت به معنی «باقیمانده، بقیّه» است.

۵ – به حاشیه(fringe)، frontier(مَرز) هم گفته می‌شود؛ تعدادی از نویسندگان به آن open list(لیست باز) هم می‌گویند. (کتاب هوش مصنوعی آفایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌ی ۷۵)

۶ – children

الگوریتم

تابع General-Search(problem,Queuing-Fn[۱])، یک راه حل یا عدم موفّقیّت[۲] را برمی‌گرداند

fringe ← make-queue[۳](make-node(initial-state[problem]))

در حلقه کارهای زیر را انجام بده

اگر fringe خالی می‌باشد، عدم موفّقیّت را برگردان

node ← Remove-Front(fringe)

در صورتی که Goal-Test[problem] به کار گرفته شده برای state(node) موفّق شد، node را برگردان

fringe ← Queuing-Fn(fringe,Expand(node,Operators[problem]))

پایان حلقه

پایان الگوریتم

در الگوریتم بالا، Queuing-Fn(queue,elements[۴]) یک تابع صف‌بندی است که مجموعه‌ای از عناصر را به صف[۵] وارد می‌نماید و ترتیب توسعه‌ی عناصر را تشخیص می‌دهد. گوناگونی‌های تابع صف‌بندی، نوع‌های مختلف الگوریتم جستجو را به وجود می‌آورد.

---

۱ – در لغت به معنی «صف‌بندی» می‌باشد.

۲ – failure

۳ – در لغت به معنی «صف» می‌باشد و در علم کامپیوتر، به عنوان **یادآوری**، ساختمان داده‌ای است که در آن، عنصرها به همان ترتیبی که وارد شده‌اند، خارج می‌شوند؛ شبیه صف‌های انسان‌ها:

۴ – در لغت به معنی «عناصر» است.

۵ – queue؛ همان طور که در گذشته هم یادآوری کردیم، ساختمان داده‌ای است که درآن، عنصرها به همان ترتیبی که وارد می‌شوند، به همان ترتیب هم خارج می‌شوند؛ یعنی، عنصرهایی که زودتر وارد شده باشند، زودتر هم خارج می‌شوند(این مطلب بیانگر مفهوم First-In-First-Out = FIFO می‌باشد).

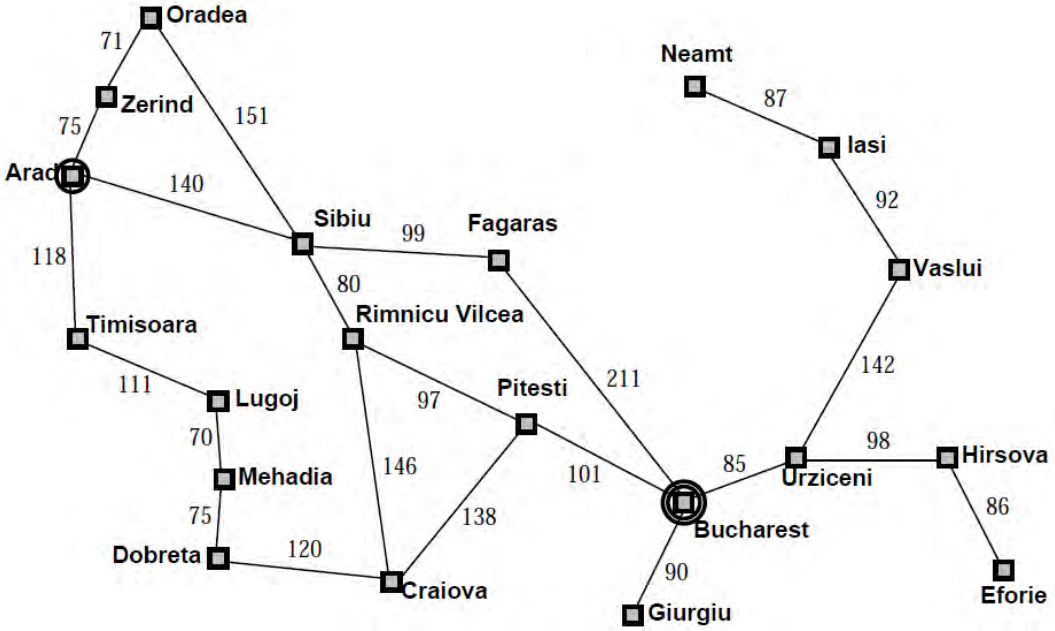
یادآوری: جستجو برای راه حل‌ها

برای پیدا کردن راه حل‌ها، که از رشته‌ای مُجاز از عملیّات تعریف شده به وسیله‌ی توابع جانشین(عملگرها) تشکیل شده‌اند، فضای جستجو را برّرسی می‌کنیم و از وضعیّت اوّلیّه به وضعیّت نهائی(هدف) حرکت می‌کنیم.

مثال- مسأله‌ی کشور رومانی[۱]- در یک تعطیلی در کشور رومانی؛ در حال حاضر در شهر آراد هستیم و هواپیما فردا شهر آراد را به مقصد شهر بُخارِست ترک می‌کند و ما می‌خواهیم در شهر بخارست باشیم.

شکل بالا- نقشه‌ای از کشور رومانی؛ در نقشه‌ی بالا، شهر آراد(مبدأ) و شهر بخارست(مقصد) به صورت روشن نشان داده شده‌اند.

۱- Romania

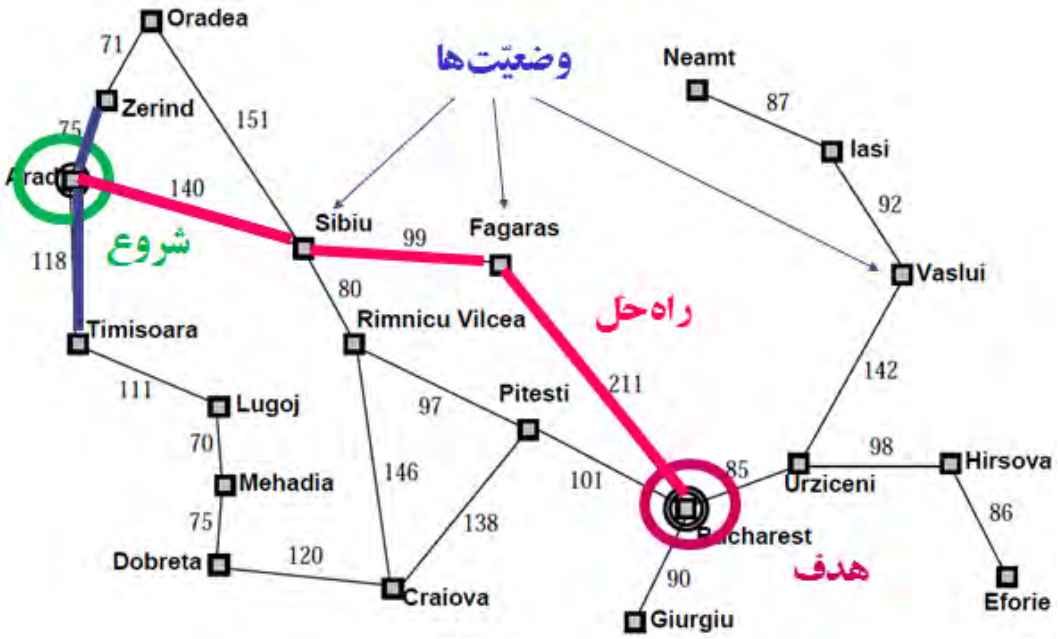


شکل بالا– *در شکل بالا توجّه کنید که عددها، فاصله‌ی خطّی مستقیم شهرها با هم برحسب مایل[1] هستند؛ به عنوان مثال، فاصله‌ی خطّی مستقیم شهر آراد تا شهر سیبیو، طبق شکل بالا برابر با ۱۴۰ مایل است.*

فرمول‌بندی مسأله: حالت(وضعیّت)ها، شهرهای مختلف هستند و عملکردها، حرکت بین دو شهر است.

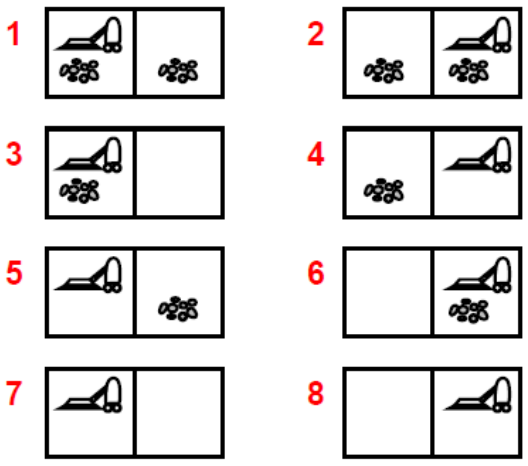
پیداکردن راه حل: ترتیب شهرها می‌تواند به صورت Arad→ Sibiu→ Fagaras→ Bucharest باشد.

۱ – mile: واحدی برای فاصله و تقریباً برابر با ۱۶۰۹ متر است. (Babylon> Babylon English)



شکل بالا– *در شکل بالا توجّه کنید که فقط یک راه حل و برخی از وضعیّت‌ها، به عنوان نمونه، با متن راهنما (متن‌های به زبان فارسی) نشان داده شده‌اند.*

مثال– دنیای جاروبرقی– وضعیّت‌های عامل جاروبرقی، تمام ترکیب‌های دریافت‌ها(درک‌ها) هستند و محیط، کاملاً قابل مشاهده می‌باشد. وضعیّت‌های ۷ و ۸ که در شکل زیر می‌بینید، وضعیّت‌های هدف می‌باشند، چون کثیف نمی‌باشند. فرض می‌کنیم برای هر عمل، هزینه برابر ۱ باشد.



اگر به عنوان مثال، در شماره‌ی پنج باشیم، راه حل، [Right,Suck] است.

فرمول‌بندی مسأله‌ی تک‌حالته

یک مسأله، در اینجا به عنوان مثال، مسأله‌ی کشور رومانی، به وسیله‌ی چهار عنصر زیر تعریف می‌شود:

حالت اوّلیّه: در شهر Arad هستیم.

تابع جانشین: S(x) = مجموعه‌ای از جفت حالات «وضعیّت –عملکرد»، به عنوان مثال،

S(Arad)={<Arad→Zerind, Zerind>,…}

آزمایش هدف: می‌تواند آشکار(صریح)[۱] باشد؛ مثلاً در مسأله‌ی کشور رومانی می‌تواند به صورت

x= «در بخارست»

باشد؛ یا ناآشکار(غیرصریح)[۲] باشد؛ مثلاً در بازی شطرنج می‌تواند به صورت checkmate[۳](x) باشد.

هزینه‌ی مسیر: به عنوان مثال می‌تواند مجموع فاصله‌ها؛ تعداد عملیّات انجام شده و ... باشد.

اگر c(x,a,y)، هزینه‌ی گام باشد، طبق پیش‌فرض باید بزرگ‌تر یا مساوی صفر باشد(c(x,a,y)>=0).

یک راه حل، ترکیبی از عملکردها(عملیّات) است. سِیْرِ آن از حالت اوّلیّه به یک حالت هدف است.

نکته:

راه حلّ بهینه، دارای پایین‌ترین هزینه‌ی مسیر است.

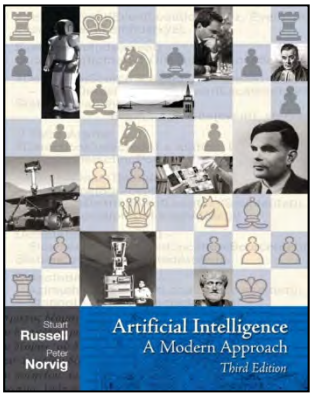
---

۱ - explicit

۲ - implicit

۳ - checkmate، در بازی شطرنج، به معنی «کیش و مات» یا «مات کردن» می‌باشد.

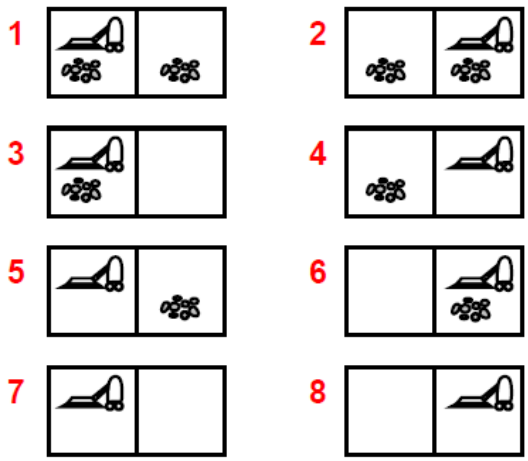
توجّه:



در ویرایش سوّم کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، به جای «تابع جانشین(successor function)»، از «عملکردها(actions)» و «مدل انتقال(transition model)» استفاده شده است. دو مثال زیر براین اساس نوشته شده‌اند.

مثال- دنیای جاروبرقی

وضعیّت(حالت)ها؟؟ دو محلّ با کثیفی یا بدون کثیفی، که می‌شود ۸ = ۲×۲^{۲} وضعیّت؛ که این هشت(۸) وضعیّت را در شکل زیر می‌بینید:



نکته‌ی مهم:

☑ **مورد استفاده در کنکور سراسری مهندسی کامپیوتر سال ۸۳ و مورد استفاده در کنکور آزاد مهندسی کامپیوتر سال ۹۲**- یک محیط، با n محل، دارای $n.2^n$ وضعیّت است.

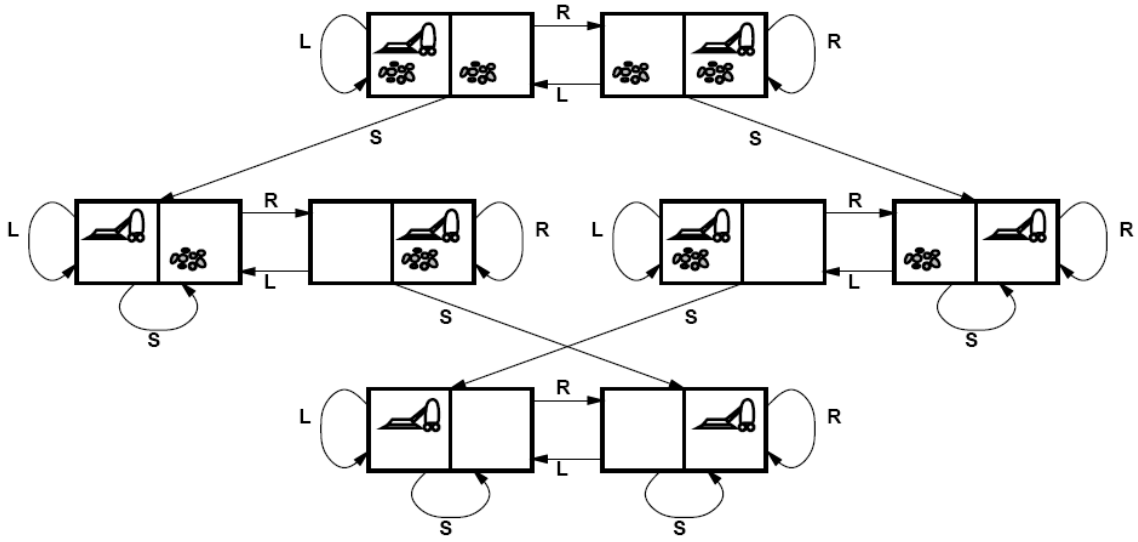
وضعیّت اوّلیّه: هر وضعیّتی می‌تواند به عنوان وضعیّت اوّلیّه در نظر گرفته شود.

عملکردها؟؟ چپ(Left)، راست(Right) و مَکِش(Suck).

مدل انتقال؟؟ عملکردها دارای اثرات متناظر هستند، به جز این موردها که دارای هیچ اثری نمی‌باشند: حرکت به چپ، در سمت چپ‌ترین خانه؛ حرکت به راست، در سمت راست‌ترین خانه؛ و مکش، در خانه‌ای که تمیز است.

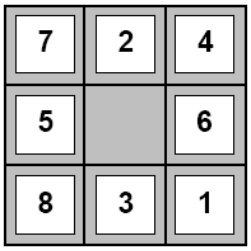
آزمون هدف؟؟ برّرسی می‌کند که آیا فضاها تمیز هستند یا کثیف؟.

هزینه‌ی مسیر؟؟ هر گام، دارای هزینه‌ی ۱ است؛ بنابراین، هزینه‌ی مسیر، تعداد گام‌ها در هر مسیر است.

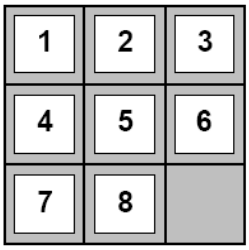


شکل بالا- فضای حالت برای دنیای جاروبرقی[۱]

مثال- پازل هشت‌تایی



شکل بالا- *وضعّیت شروع*



شکل بالا- *وضعّیت هدف*

حالت(وضعیّت)ها؟؟ جاهای کاشی‌ها

وضعیّت اوّلیّه: هر وضعیّتی می‌تواند به عنوان وضعیّت اوّلیّه در نظر گرفته شود. چنانچه قبلاً هم اشاره کردیم، توجّه کنید که هر هدف داده شده می‌تواند از دقیقاً نیمی از وضعیّت‌های اوّلیّه‌ی ممکن، مورد دستیابی قرار گیرد!.

عملکردها؟؟ حرکت کردن به کاشی خالی‌ای که در سمت چپ، راست، بالا یا پایین قرار گرفته است.

مدل انتقال؟؟ با داشتن یک وضعیّت و عملکرد، وضعیّت نتیجه شده را برمی‌گرداند.

آزمایش هدف؟؟ برّرسی می‌کند که آیا به وضعیّت هدف داده شده رسیده‌ایم یا نه؟.

---

۱ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌ی ۷۰

هزینه‌ی مسیر؟؟ هر گام دارای هزینه‌ی ۱ است؛ بنابراین، هزینه‌ی مسیر، تعداد گام‌ها در هر مسیر است.[۱]

نکته:

راه حلّ معمولی پازل n–تایی، NP-hard[۲] می‌باشد.

دو نوع اساسی فرمول‌بندی وجود دارد: یکی ***فرمول‌بندی افزایشی (سازنده)***[۳][۴]، که دارای عملگرهایی است که از یک وضعیّت تُهی شروع می‌کنند و وضعیّت را کامل می‌کنند؛ برای مسأله‌ی ۸–وزیر این، به این معنی است که در هر عملکرد، یک وزیر به حالت اضافه می‌شود. و دیگری ***فرمول‌بندی حالت کامل (بهبود تکرارشونده، بهبود تکراری)***[۵][۶]، که با همه‌ی ۸–وزیر روی صفحه شروع می‌کند و آنها را حرکت می‌دهد.[۷]

یادآوری– درخت‌ها

یک درخت، یک گراف بدون حلقه‌ی[۸] مستقیم[۹] می‌باشد. گراف مستقیم، مجموعه‌ای از گره[۱۰]های متّصل شده به وسیله‌ی لَبه‌ها(یال‌ها)[۱۱] می‌باشد. مسیر، رشته‌ای از لبه‌ها، که امکان دارد تکرار هم شده باشند، است. در درخت حدّاکثر یک مسیر متّصل کننده‌ی هر جفت از گره‌ها وجود دارد(بدون حلقه(دُوْر) است).

درخت‌ها به دلیل اینکه رفتار الگوریتمی خوبی دارند، به صورت گسترده‌ای در علم کامپیوتر استفاده می‌شوند و دارای قدرت زیاد، برای تعریف الگوریتم‌های بازگشتی[۱] می‌باشند. در زیر تصویری از یک درخت را مشاهده می‌نمایید:

---

۱ – کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌ی ۷۱

۲ – **یادآوری**– مسأله‌های NP-hard، مسأله‌هایی هستند که در یک زمان چندجمله‌ای نمی‌توانند حل شوند.

۳ - incremental formulation

۴ - constructive

۵ - complete-state formulation

۶ - iterative improvement

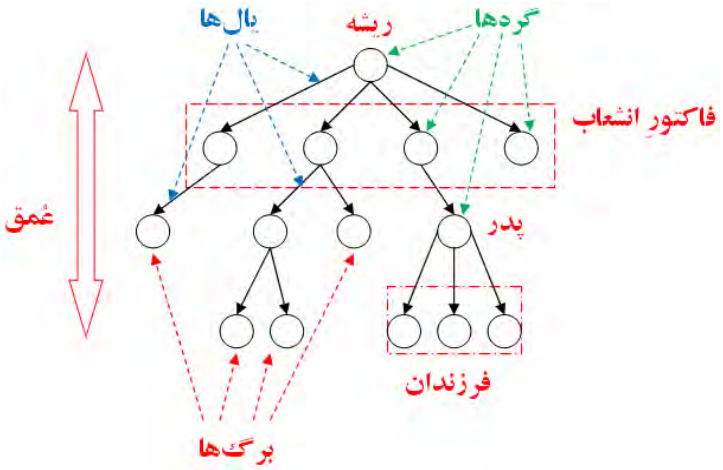
۷ – کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌ی ۷۲

۸ – acyclic

۹ – direct

۱۰ – node

۱۱ – edges



توجّه نمایید که در شکل بالا، برخی از یال‌ها، برخی از گره‌ها، برخی از فرزندان، برخی از برگ[۲]ها، فقط یک فاکتور انشعاب[۱] و فقط یک پدر(والد)[۲] داری متن راهنما هستند؛ مثلاً اینکه فقط سه گره‌ی فرزند نشان داده شده است، به این معنی نیست که در درخت بالا گره فرزند دیگری وجود ندارد!.

---

۱ - recursive algorithms؛ **یادآوری**- الگوریتم‌هایی هستند که در موقع اجرا خودشان را فراخوانی(call) می‌کنند؛ یک مثال رایج در این مورد، تابع فاکتوریل(factorial function) می‌باشد که به صورت

$$n! = n.(n-1).(n-2).(n-3) \dots 3.2.1$$

برای عددهای صحیح مثبت و $0! = 1$ برای عدد صفر می‌باشد. در زبان برنامه‌نویسی C می‌توانیم تابعی را به صورت بازگشتی برای محاسبه‌ی فاکتوریل به صورت زیر بنویسیم:

```
int factorial(int n) {
  if (n == 0)
    return 1;
  else
    return (n * factorial(n-1));
}
```

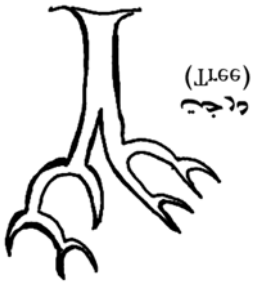
حال برای امتحان این تابع، فرض کنیم تابع را با $n = 3$ فراخوانی نماییم؛ داریم:

```
factorial(3)
3 * factorial(2)
3 * 2 * factorial(1)
3 * 2 * 1 * factorial(0)
3 * 2 * 1 * 1
=> 6
```

در نتیجه همان طور که می‌بینیم، حاصل factorial(3) برابر با عدد ۶ می‌شود، که این نتیجه درست است. (http://www.cse.wustl.edu/~kjg/cse131/Notes/Recursion/recursion.html، این پایگاه اینترنتی، متعلّق به دانشگاه واشینگتُن، در شهر سَنْتْ لوئیس(st. louis) کشور ایالات متّحده‌ی آمریکا است.)

۲ - leaf

همان طور که می‌بینید، شکل درخت، شبیه درخت‌های موجود در طبیعت است که وارونه(واژگون) شده باشند:



توجّه کنید که گره‌ها حدّاکثر دارای یک پدر هستند و ریشه[۳]، گره‌ای متمایز است که دارای پدری نمی‌باشد؛ در ضمن، عمق[۴] هم که برای خاتمه‌ی الگوریتم مهم است، می‌تواند نامحدود باشد.

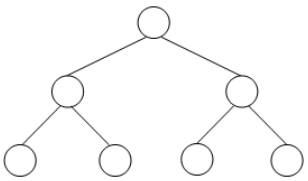
درخت جستجو[۵]

تعریف- درخت‌های جستجو، ساختمان داده[۶] ای برای جستجوی فضای حالت هستند.

نکته:

پس با توجّه به تعریف قبل، درخت جستجو با استفاده از فضای حالت به وجود می‌آید.

۱- branching factor: **یادآوری**- در محاسبه(computing) و ساختمان داده‌های درختی(tree data structures) و تئوری بازی‌ها(game theory)، فاکتور انشعاب، تعداد فرزندان(بچّه‌های) هر گره(node) می‌باشد. به عنوان مثال، درخت شکل زیر دارای فاکتور انشعاب ۲(دو) است:



(http://en.wikipedia.org/wiki/Branching_factor)

۲- parent

۳- root

۴- depth

۵- search tree

۶- data structure؛ روشی است که در آن داده‌ها برای جستجو و بازیابی کارا(مؤثّر یا کارآمد) نگهداری می‌شوند. ساده‌ترین ساختمان داده، آرایه‌ی(Array) یک بعدی(خطّی) است. (Babylon> Britannica Concise Encyclopedia)

در درخت جستجو، گره‌ی ریشه، وضعیّت اوّلیه است و گره‌های فرزند، وضعیّت‌هایی هستند که می‌توانند از پدر(والد) ملاقات شوند. همان طور که گفتیم، عمق درخت می‌تواند نامحدود(بینهایت) باشد؛ مثلاً موقعی که وضعیّت‌های تکراری داشته باشیم.

تعریف- درخت جستجوی جزئی: بخشی از درخت است که تاکنون توسعه داده شده است.

یک درخت جستجو را با شروع از وضعیّت اوّلیّه و به کارگیری مکرّر تابع جانشین(مولّد) تولید می‌نماییم؛

ایده‌ی اصلی این است که از یک وضعیّت توجّه کنید که چه چیزهایی می‌تواند انجام شود؛ سپس توجّه کنید که از هریک از وضعیّت‌های آنها چه چیزهایی می‌توانند انجام بگیرند. در اینجا سؤالاتی به وجود می‌آید که برخی از آنها عبارتند از: «آیا پیداکردن راه حلّ بهینه ضمانت شده است؟»، «تا کِی جستجو انجام خواهد شد؟» و «به چه مقدار از فضا نیاز خواهیم داشت؟».

یک الگوریتم جستجوی درختی ساده

الگوریتم:

تابع TREE-SEARCH(problem , strategy[1])، یک راه حل یا عدم موفّقیّت را برمی‌گرداند

درخت جستجو را با استفاده از حالت اوّلیّه‌ی مسأله مقداردهی اوّلیّه نمایید(در ابتدا درخت جستجو از حالت اوّلیّه‌ی مسأله استفاده می‌کند)

در حلقه کارهای زیر را انجام بده

اگر داوطلبی(کاندیدی) برای توسعه نبود، عدم موفّقیّت را برگردان

یک گره‌ی برگ را با توجّه به روش(استراتژی) انتخاب کن

در صورتی که گره، دارای یک وضعیّت هدف می‌باشد، راه حلّ متناظر را برگردان

در غیراین صورت گره را توسعه بده و گره‌های نتیجه شده را به درخت جستجو اضافه کن

پایان حلقه

---

۱ - در لغت به معنی «روش، استراتژی» می‌باشد.

پایان الگوریتم

یادآوری: توسعه‌دادن یک گره، به معنی به وجود آوردن همه‌ی فرزندان آن گره است.

نکته:

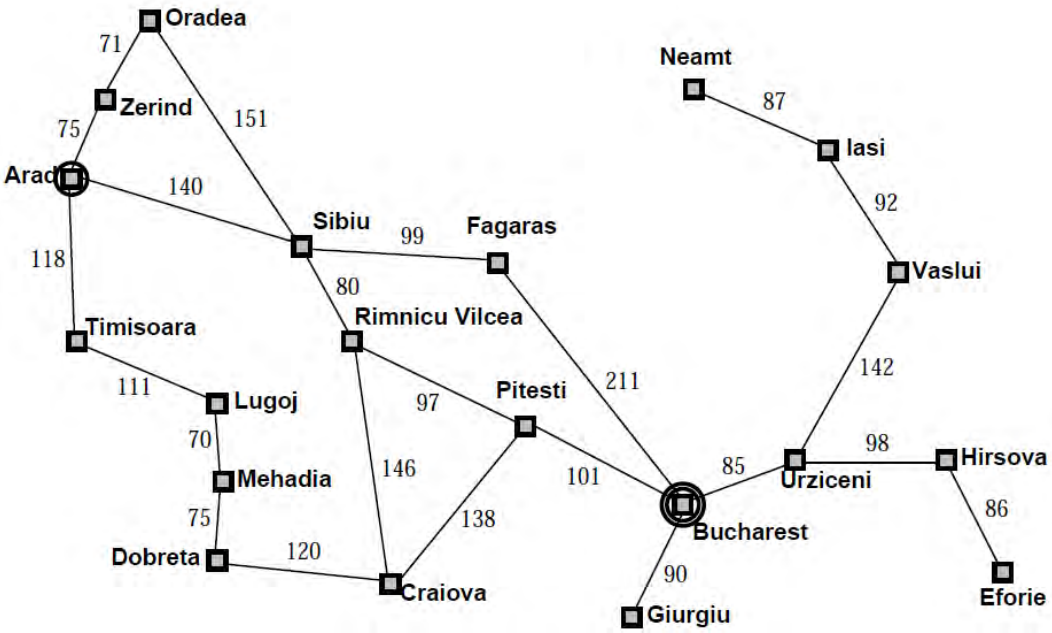
یک گره‌ی x، یک ساختمان داده است و جزء اصلی یک درخت جستجو می‌باشد و شامل وضعیّت s، پدر(والد)، فرزندان (در صورتی که s توسعه داده شده باشد)، عمق و هزینه‌ی مسیر $(g(x))$ است.

نکته‌ی مهم:

پس بنابر نکته‌ی قبلی، یک وضعیّت، جزئی از یک گره است و خود وضعیّت‌ها؛ والدین، فرزندان، عمق، یا هزینه‌ی مسیر ندارند.

روش(استراتژی): روش جستجو، براساس ترتیبی که براساس آن گره‌ها توسعه داده می‌شوند، مشخّص می‌شود.

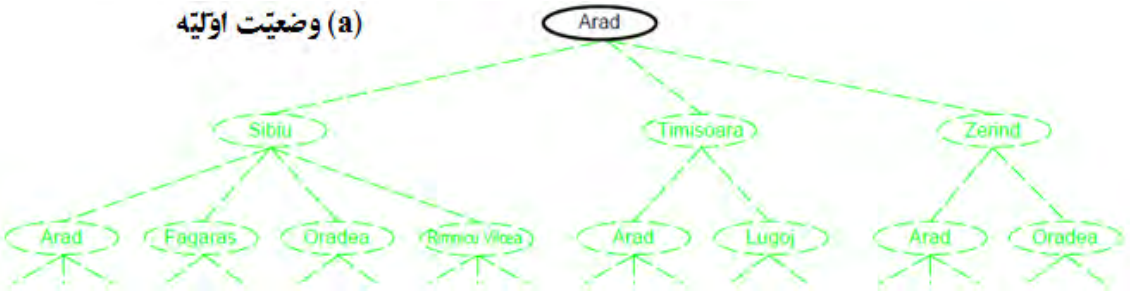
مثال جستجوی درختی (مسأله‌ی رومانی)



در درخت جستجو؛ یک وضعیّت می‌تواند از چند مسیر قابل دسترسی باشد و ریشه، یک گره جستجو، متناظر با وضعیّت اوّلیّه، که در اینجا شهر آراد است، می‌باشد.

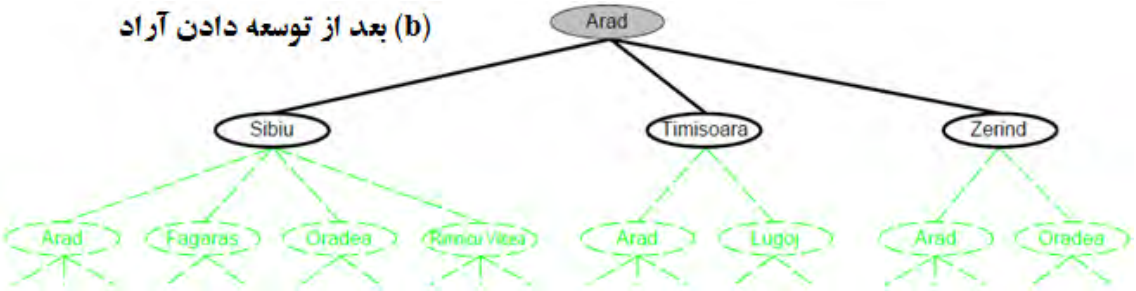
در زیر درخت‌های جستجوی جزئی، برای پیدا کردن یک مسیر از آراد به بخارست را می‌بینید. در این شکل، گره‌های توسعه داده شده، دارای رنگ زمینه هستند؛ گره‌هایی که تولید شده‌اند، امّا هنوز توسعه داده نشده‌اند، خطّ آنها پُر رنگ شده است؛ گره‌هایی که هنوز تولید نشده‌اند، به صورت خط‌چین نشان داده شده‌اند.

درخت جستجو، در ابتدا-



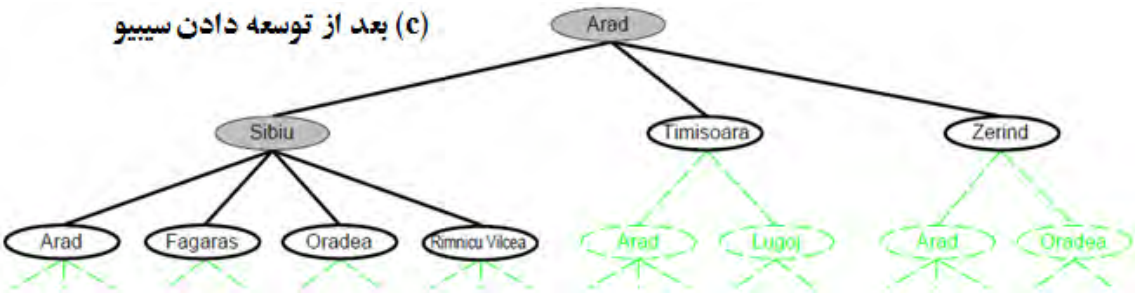
شکل بالا- در شکل بالا آراد تولید شده، امّا هنوز توسعه داده نشده است. سایر گره‌ها هنوز تولید نشده‌اند.

گام(مرحله‌ی) نخست- درخت جستجو، پس از توسعه دادن شهر آراد:



شکل بالا- در شکل بالا آراد توسعه داده شده است.

گام بعدی- درخت جستجو، پس از توسعه دادن شهر سیبیو:



نکته:

درخت جستجو ممکن است به دلیل به وجود آمدن وضعیّت‌های تکراری، نامحدود باشد، حتّا اگر فضای حالت، کوچک باشد.

در موقع پیاده‌سازی یک جستجوی درختی، از یک تابع، مثل تابع EXPAND[۱] برای ساختن فرزندان هر گره می‌توانیم استفاده می‌کنیم و از Successor-Fn می‌توان برای ساختن حالت‌های مورد نظر در مسأله استفاده کرد.

الگوریتم جستجوی درختی(به بیان دیگر و کامل‌تر نسبت به الگوریتم قبلی)

الگوریتم

تابع Tree-Search(problem,fringe)، یک راه حل و یا عدم موفّقیّت را برمی‌گرداند

fringe ← Insert[۲](Make[۳]-Node(Initial[۴]-State[problem]),fringe)

در حلقه کارهای زیر را انجام بده

در صورتی که fringe، خالی است، عدم موفّقیّت را برگردان

node ← Remove[۵]-Front[۶](fringe)

در صورتی که Goal-Test(problem,State(node))، Solution(node)(راه حلّ متناظر با گره) را برگردان

fringe ← InsertAll(Expand(node,problem),fringe)

پایان حلقه

تابع Expand(node,problem)، مجموعه‌ای از گره‌ها را برمی‌گرداند

یک مجموعه‌ی خالی(تهی) ← successors

---

۱ – در لغت به معنی «توسعه دادن، بسط دادن، گسترش دادن» می‌باشد.

۲ – در لغت به معنی «درج کردن» می‌باشد و در اینجا حالت امری دارد؛ یعنی: «درج کن»

۳ – در لغت به معنی «ساختن، به وجود آوردن» می‌باشد و در اینجا حالت امری دارد؛ یعنی: «بساز»

۴ – در لغت به معنی «اوّلیّه» می‌باشد.

۵ – در لغت به معنی «برداشتن، حذف کردن» می‌باشد و در اینجا حالت امری دارد؛ یعنی: «بَردار، حذف کن»

۶ – در لغت به معنی «جلو، پیش» می‌باشد.

برای هر action(عملکرد) و result(نتیجه) در Successor-Fn[1](problem,State[node])، کارهای زیر را انجام بده

یک گره‌ی(Node) جدید ← s

State[s] ← result

Parent-Node[s] ← node

Action[s] ← action

Path-Cost[s] ← Path-Cost[node]+Step-Cost(node,action,s)

Depth[s] ← Depth[node]+1

S را به successors اضافه کن

پایان حلقه

Successors را برگردان

پایان الگوریتم

استراتژی(روش)های جستجو

همان طور که گفتیم، یک استراتژی براساس ترتیبی که در آن گره‌ها توسعه داده می‌شوند، تعریف می‌شود. استراتژی‌ها به وسیله‌ی موردهای زیر ارزیابی می‌شوند:

تمامیّت(کامل بودن)[2]: آیا همیشه اگر راه حل موجود باشد، آن را پیدا می‌کند؟

پیچیدگی زمانی[3]: در بدترین حالت[4] یا در حالت میانگین[1] چقدر زمان طول می‌کشد تا یک راه حل پیدا شود؟؛ [پیچیدگی زمانی] معمولاً براساس تعداد گره‌های به وجود آمده /(یا) توسعه داده شده اندازه‌گیری می‌شود.

---

۱ - **یادآوری**- همان طور که در گذشته هم گفتیم، از این تابع می‌توان برای ساختن حالت‌ها یا وضعیّت‌ها استفاده کرد.

۲ - completeness

۳ - time complexity؛ **یادآوری**- توجّه کنید که پیچیدگی زمانی معمولاً با حرف «اُی بزرگ انگلیسی(O)» نشان داده می‌شود؛ به عنوان مثال، ممکن است پیچیدگی زمانی یک الگوریتم، برابر با $O(n^3)$ باشد.
(http://en.wikipedia.org/wiki/Time_complexity)

۴ - worst case

پیچیدگی فضا(فضایی)[۲]: چقدر فضا(جا) به وسیله‌ی الگوریتم مورد استفاده قرار می‌گیرد؟؛ معمولاً براساس بیشینه‌ی تعداد گره‌های موجود در حافظه، در هنگام جستجو، اندازه‌گیری می‌شود.

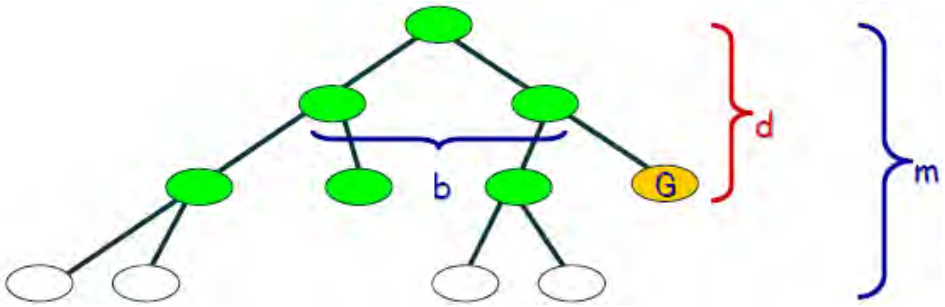
بهینگی[۳]: آیا همیشه کم هزینه‌ترین راه حل را پیدا می‌کند؟.

پیچیدگی زمانی و فضایی به وسیله‌ی پارامترهای زیر ارزیابی می‌شوند:

b: بیشینه‌ی(ماکزیمم) فاکتور انشعاب درخت جستجو، که فرض می‌کنیم محدود باشد.

d: عمق کم هزینه‌ترین راه حل، اگر راه حلّی وجود نداشته باشد، بینهایت خواهد بود.

m: ماکزیمم عمق فضای حالت، که ممکن است بینهایت باشد.



شکل بالا– تصویری برای درک بهتر پارامترهای b، d و m.

یادآوری– پیچیدگی زمانی

در مورد پیچیدگی الگوریتم‌ها نگران هستیم، زیرا یک مسأله ممکن است در حالت کلّی قابل حل باشد، امّا در عمل(حالت عملی) خیلی طولانی باشد. سختی ما در این مورد این است که الگوریتم‌های شناخته شده، دارای پیچیدگی نمایی هستند و تعداد عملیّات به صورت نمایی رشد می‌کند.

مسأله‌های با زمان چندجمله‌ای[۴](P): یک مسأله دارای زمان چندجمله‌ای است، اگر الگوریتمی برای آن وجود داشته باشد که دارای زمان چندجمله‌ای باشد. دشواری ما اینجاست که شاید هیچ الگوریتم چندجمله‌ای که بتواند مسأله را حل

۱- average case
۲- space complexity
۳- optimality
۴- polynomial-time

کند، وجود نداشته باشد و در صورت وجود هم ممکن است نمایش آن آسان نباشد و معمولاً قابل کاهش به صورت مسأله‌های شناخته شده هم نمی‌باشد.

مسأله‌های دارای پیچیدگی زمانی نمایی: می‌توانیم الگوریتم‌هایی را پیدا کنیم که مسأله‌های دارای پیچیدگی نمایی را در یک زمان که به صورت چندجمله‌ای با اندازه‌ی ورودی رشد می‌کنند، حل کنند؛ برای مثال، در مرتّب کردن n عدد، به صورت صعودی(از کوچک به بزرگ)، الگوریتم‌های ضعیف، این کار را با پیچیدگی n^2 انجام می‌دهند، الگوریتم‌های بهتر، این کار را در $n\times\log(n)$، که بهتر از n^2 است، انجام می‌دهند. مسأله در اینجا این است که [در برخی از موردها] نمی‌توانیم وضعیّت مرتبه‌ی(توان) چندجمله‌ای را مشخّص کنیم، شاید خیلی بزرگ باشد!. برای برخی از مسأله‌ها هیچ الگوریتم چندجمله‌ای نداریم، که مسأله‌های دارای زمان چندجمله‌ای غیرقطعی[1] جزء این دسته هستند؛ برای مثال، مسأله‌ی مسافرت فروشنده‌ی دوره‌گرد این گونه است.

چرا پیچیدگی نمایی «مشکل(سخت یا دشوار)» است؟: پیچیدگی نمایی به آن معنی است که تعداد عملیّات لازم، برای محاسبه‌ی راه حلّ دقیق مسأله، به صورت نمایی با اندازه‌ی مسأله رشد می‌کند.

جستجوی ناآگاهانه(کور)[2]

تعریف- ساده‌ترین الگوریتم‌های جستجو آنهایی هستند که هیچ اطّلاعات اضافی، بجز آنهایی که در شرح مسأله وجود دارد را استفاده نمی‌نمایند، این روش‌ها را روش‌های جستجوی ناآگاهانه می‌نامیم.

مطلب مهم:

در صورتی که دارای اطّلاعات اضافی در مورد چگونگی یک وضعیّت غیر هدف باشیم، در این صورت می‌توانیم جستجوی *مکاشفه‌ای*[3] را انجام دهیم.

۱- *n*ondeterministic-*p*olynomial-time(*NP*)

۲- uninformed(blind) search

۳- این روش(جستجوی مکاشفه‌ای) در فصل «جستجوی مکاشفه‌ای» همین کتاب الکترونیکی برّرسی شده است.

روش‌های جستجوی ناآگاهانه

مطلب مهم:

شامل جستجوی اوّلْ سطح(جستجوی ردیفی)[۱]؛ جستجوی با هزینه‌ی یکسان[۲]؛ جستجوی اوّلْ عمق[۳]؛ جستجوی با عمق محدود شده[۴][ی اوّل عمق]؛ و جستجوی عمیق شونده‌ی تکراری[۵][اوّل عمق] می‌باشند.

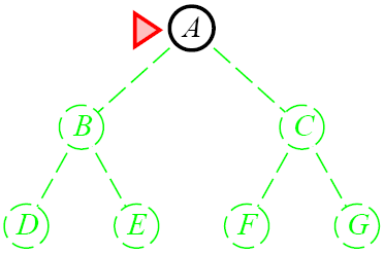
جستجوی اوّلْ سطح

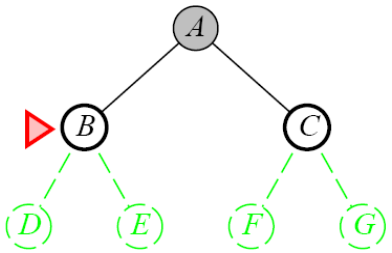
مطلب مهم:

با توسعه دادن یک گره کار می‌کند، سپس هر یک از فرزندانش را توسعه می‌دهد، سپس هریک از فرزندان آنها را توسعه می‌دهد و.... *در این روش همه‌ی گره‌های در عمق n قبل از یک گره در عمق $n+1$ ملاقات می‌شوند.*

به عبارت دیگر، جستجوی اوّل سطح، کم‌عمق‌ترین گره توسعه داده نشده را توسعه می‌دهد. *می‌توانیم جستجوی اوّل سطح را با استفاده از یک* ***صف*** *پیاده‌سازی نماییم.* در **پیاده‌سازی**، fringe، یک صف می‌باشد و جانشینان جدید به انتها می‌روند.

مثال(شکل‌ها را از چپ به راست(←) دنبال نمایید):





۱- Breadth-first
۲- Uniform-cost
۳- Depth-first
۴- Depth-limited
۵- Iterative deepening

برخی نکات ریز

[در روش جستجوی اوّل سطح، مثلاً در مورد مسأله‌ی کشور رومانی،] چگونه از ملاقات مجدّد شهر آراد جلوگیری نماییم؟: *با استفاده از closed[1]-list، یک لیست از وضعّیت‌های توسعه داده شده را نگهداری نماییم.* توجّه کنید که راه حل، یک مسیر می‌باشد، نه یک شهر.

چگونه از وارد کردن دوباره‌ی شهر اُرادیا اجتناب نماییم؟: *با استفاده از لیست open-list، یک لیست از وضعّیت‌هایی که تولید شده‌اند، امّا توسعه داده نشده‌اند را نگهداری می‌نماییم.*

ویژگی‌های جستجوی اوّل سطح

کامل؟؟(آیا جستجوی اوّل سطح راه حل را پیدا می‌کند؟) بله (در صورتی که b محدود باشد).

توضیح:

تصوّر نمایید که راه حل در عمق n می‌باشد. از آنجایی که همه‌ی گره‌های در عمق n، یا در بالای n، قبل از هر گره‌ای در عمق n+1 ملاقات می‌شوند، راه حل پیدا خواهد شد.

زمان؟؟ جستجوی اوّل سطح به زمان اجرای $O(b^{d+1})$ نیاز دارد. جستجوی اوّل سطح

$$1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$$

گره را ملاقات خواهد کرد (به d و چگونگی آن در فرمول توجّه نمایید).[2]

---

۱ – در لغت به معنی «مسدود، بسته» است. به closed list، explored set هم گفته می‌شود. (کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌ی ۷۷)

۲ – **توجّه**– در این فرمول فرض شده است که الگوریتم روش جستجوی اوّل سطح، آزمون هدف را در زمانی که گره‌ها برای توسعه انتخاب می‌شوند، انجام داده است؛ اگر الگوریتم روش جستجوی اوّل سطح، آزمون هدف را در زمان تولید گره‌ها انجام دهد، این فرمول برابر خواهد بود با:

$$1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$$

(کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌ی ۸۲)

☑**کنکور سراسری فنّـآوری اطّلاعات سال‌های ۸۸ و ۸۵: فضـا؟؟** $O(b^{d+1})$ (همـه‌ی گره‌هـا را در حافظـه نگهداری می کند.)

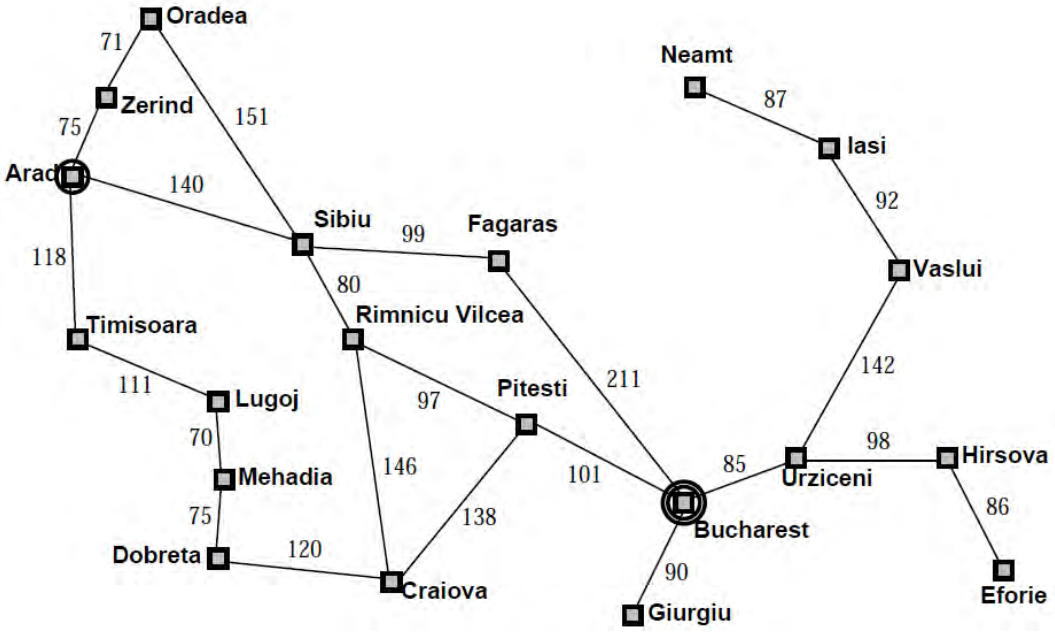
توضیح:

جستجوی اوّل سطح باید همه‌ی درخت جستجو را در حافظه نگهداری نماید (زیرا می‌خواهیم رشته‌ی عملکردها را برای رسیدن به هدف بدانیم).

بهینگی؟؟(اگر چند راه حل وجود داشته باشد، آیا جستجوی اوّل سطح، بهترین راه حل را پیدا می کند؟) بله.

توضیح:

جستجوی اوّل سطح، کم‌عمق‌ترین راه حل را در درخت جستجو پیدا می‌نماید. در صورتی که هزینه‌ی مراحل(گام‌ها)، یکسان باشند، این روش بهینه خواهد بود، در غیر این صورت لزوماً بهینه نخواهد بود.



آراد ← سیبیو ← فَگراز ← بخارست، در ابتدا پیدا خواهند شد (فاصله = ۱۴۰+۹۹+۲۱۱ = ۴۵۰)

آراد ← سیبیو ← ریمنیکو ویلسیا ← پیتِستی ← بخارست، کوتاه‌تر می‌باشد. (فاصله = ۱۴۰+۸۰+۹۷+۱۰۱ = ۴۱۸)

توجّه:

فضا [در روش جستجوی اوّل سطح] مسأله‌ای بزرگ است؛ چونکه گره‌ها باید در حافظه نگهداری شوند؛ فضای زیادی اشغال می کند. در کل، فضای مورد نیاز جستجوی اوّل سطح، مسأله‌ای بزرگ‌تر از زمان مورد نیاز می‌باشد. با فرض b=10، سرعت ۱۰۰۰۰(ده هزار) گره در ثانیه و فضای ۱۰۰۰(هزار) بایت برای هر گره، جدول زیر را خواهیم داشت:

عمق	گره	زمان	حافظه
۲	۱۱۰۰	۰٫۱۱ ثانیه	۱ مگابایت[۱]
۴	۱۱۱۱۰۰	۱۱ ثانیه	۱۰۶ مگابایت
۶	$۱۰^{۷}$	۱۹ دقیقه	۱۰ گیگابایت[۲]
۸	$۱۰^{۹}$	۳۱ ساعت	۱ ترابایت[۳]
۱۰	$۱۰^{۱۱}$	۱۲۹ روز	۱۰۱ ترابایت
۱۲	$۱۰^{۱۳}$	۳۵ سال	۱۰ پتابایت[۴]
۱۴	$۱۰^{۱۵}$	۳۵۲۳ سال	۱ اگزابایت[۵]

جستجوی با هزینه‌ی یکسان

مطلب مهم:

به یاد بیاورید که جستجوی اوّل سطح در زمانی که هزینه‌ی مراحل غیر یکسان باشد، غیربهینه می‌باشد. می‌توانیم این اشکال را رفع نماییم؛ *برای این کار به جای توسعه‌ی گره‌ها به صورت سطحی، جستجوی با هزینه‌ی یکسان، گره‌ی n را که دارای پایین‌ترین هزینه‌ی مسیر $g(n)$ است را توسعه می‌دهد. این کار با نگهداری fringe (حاشیه یا frontier (مرز)) به صورت یک صف اولویّت[۶] مرتّب شده براساس g انجام می‌شود.* این روش کوتاه‌ترین مسیر را پیدا خواهد کرد. در صورتی که هزینه‌ها یکسان باشند، این روش با جستجوی اوّل سطح برابر می‌باشد. در پیاده‌سازی، ریشه، صفی با ارزیابی مسیر است، که کم‌هزینه‌ترین، در اوّل صف قرار دارد.

مثال- در زیر مثالی از جستجوی با هزینه‌ی یکسان، از ویرایش سوّم کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ آورده شده است:

---

۱ - *M*ega*b*yte(*MB*): یک میلیون بایت = ۲ به توان ۲۰ بایت = ۱۰۲۴ کیلوبایت. (Babylon> Computer Terms Glossary)
یک کیلوبایت = ۲ به توان ۱۰ بایت = ۱۰۲۴ بایت.

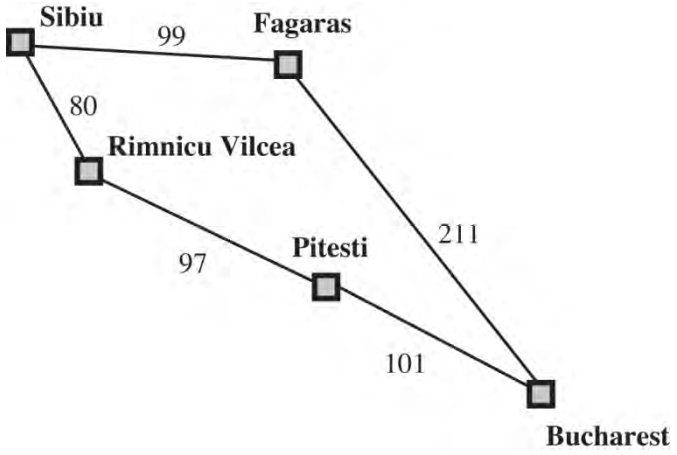
۲ - *G*iga*b*yte(*GB*): یک بیلیون بایت است = ۲ به توان ۳۰ بایت = ۱۰۲۴ مگابایت. (Babylon> FOLDOC)

۳ - *T*era*b*yte(*TB*): یک تریلیون بایت = ۱۰ به توان ۱۲ بایت = ۲ به توان ۴۰ بایت = ۱۰۲۴ گیگابایت. (Babylon> Persian Computer Encyclopedia)

۴ - Petabyte: ۱۰۲۴ ترابایت = ۱۰ به توان ۱۵ بایت = ۲ به توان ۵۰ بایت. (Babylon> Persian Computer Encyclopedia)

۵ - *E*xa*b*yte(*EB*): یک میلیارد گیگابایت، واحد بسیار بزرگی در اندازه‌گیری سایز داده‌ها است؛ ۲ به توان ۶۰ بایت = ۱۰۲۴ پتابایت = ۱۰ به توان ۱۸ بایت. (Babylon> Persian Computer Encyclopedia)

۶ - priority queue؛ **یادآوری**- ساختمان داده‌ای است که دارای سه عملیّات می‌باشد: درج یک عنصر جدید، برگرداندن عنصر دارای بالاترین اولویّت و حذف عنصر دارای بالاترین اولویّت. (Babylon> FOLDOC)



شکل بالا– بخشی از فضای حالت مسأله‌ی کشور رومانی که برای توضیح روش جستجوی با هزینه‌ی یکسان انتخاب شده است.

در اینجا مسأله این است که از Sibiu به Bucharest برویم. جانشینانِ Sibiu، Rimnicu Vilcea و Fagaras، به ترتیب، با هزینه‌های ۸۰ و ۹۹ هستند. کم هزینه‌ترین گره؛ یعنی، Rimnicu Vilcea توسعه داده می‌شود. سپس Pitesti، با هزینه‌ی ۱۷۷=۸۰+۹۷ اضافه می‌شود. کم هزینه‌ترین گره، حالا Fagaras است، بنابراین، توسعه داده می‌شود، اضافه کردن Bucharest با هزینه‌ی ۳۱۰=۲۱۱+۹۹. حالا یک گره‌ی هدف تولید شده است، امّا جستجوی با هزینه‌ی یکسان به کار خود ادامه می‌دهد، انتخاب Pitesti برای توسعه، و اضافه کردن یک مسیر دوّم به Bucharest، با هزینه‌ی ۲۷۸=۱۰۱+۹۷+۸۰. حال الگوریتم برّرسی می‌کند که آیا این مسیر جدید، بهتر از مسیر قبلی است یا نه؟؛ همین گونه است، بنابراین، مسیر قبلی حذف می‌شود. بخارست، حالا با هزینه‌ی g برابر با ۲۷۸ برای توسعه انتخاب می‌شود و راه حل برگردانده می‌شود.[۱]

ویژگی‌های جستجوی با هزینه‌ی یکسان

کامل؟؟ بله، اگر $\varepsilon \geq \text{cost}$(هزینه‌ی گام) ($\varepsilon$، ثابتی با مقدار مثبت و کوچک است).

پیچیدگی زمانی؟؟ اگر C^*، هزینه‌ی راه حلّ بهینه(مطلوب) باشد و هزینه‌ی هر عمل، دست‌کم برابر با ε باشد، در بدترین حالت برابر با $O(b^{\frac{c^*}{\varepsilon}})$ می‌باشد.[۲]

فضا؟؟ مثل پیچیدگی زمانی است.

بهینگی؟؟ بله (در صورتی که کامل باشد.)–*گره‌ها را به ترتیب افزایش هزینه‌ی مسیر توسعه می‌دهد.*

---

۱ – کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌های ۸۴ و ۸۵

۲ – در ویرایش سوّم کتاب هوش مصنوعی آقایان، استوارات راسل و پیتر نورویگ، این فرمول به صورت $(b^{1+\lfloor\frac{C^*}{\varepsilon}\rfloor})$ می‌باشد. (کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌ی ۸۵)

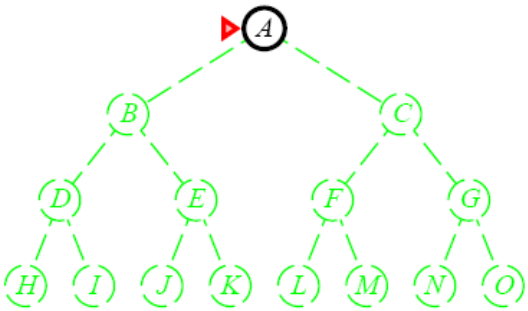
جستجوی اوّلْ عمق

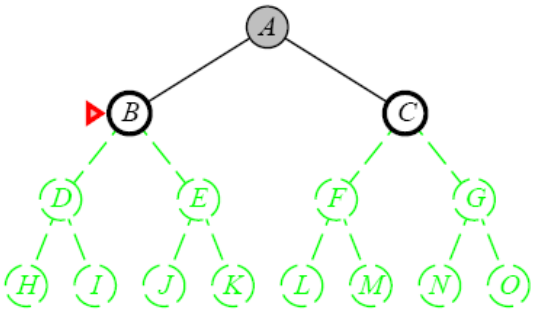
مطلب مهم:

دارای روش متفاوتی نسبت به جستجوی اوّل سطح می‌باشد و همیشه عمیق‌ترین گره را توسعه می‌دهد. ابتدا یک فرزند را توسعه می‌دهد، سپس سمت چپ‌ترین فرزند آن را توسعه می‌دهد و به همین ترتیب. با استفاده از یک ***پُشته***[1] *هم می‌توانیم روش جستجوی اوّل عمق را پیاده‌سازی نماییم.*

پس جستجوی اوّل عمق، عمیق‌ترین گره‌ی توسعه داده نشده را توسعه می‌دهد.

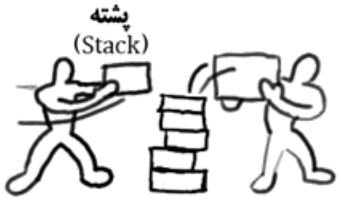
مثال- شکل‌ها را از چپ به راست(←) دنبال نمایید. فرض کنید که تنها گره‌ی هدف، گره‌ی M است. توجّه کنید که در سطح سوّم، که گره‌های H, I, J, K, L, M, N, O قرار دارند، هیچ گره‌ی جانشین(successor)ی نداریم.



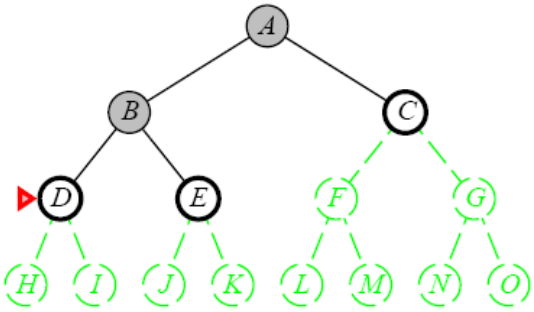
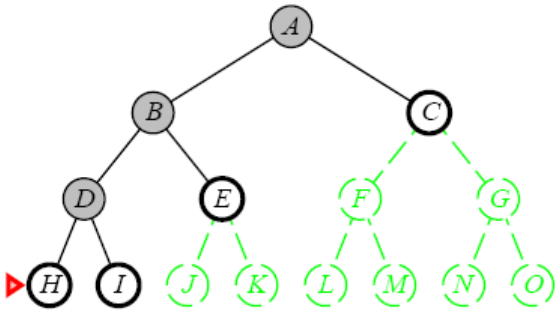
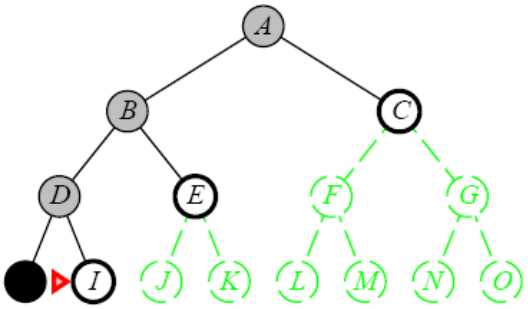
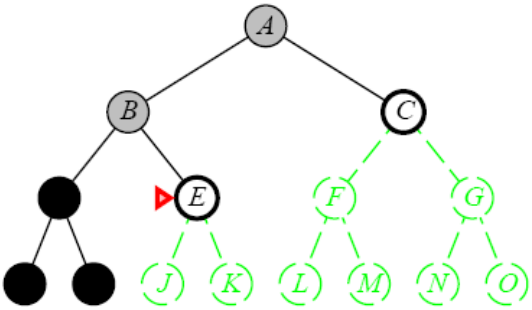
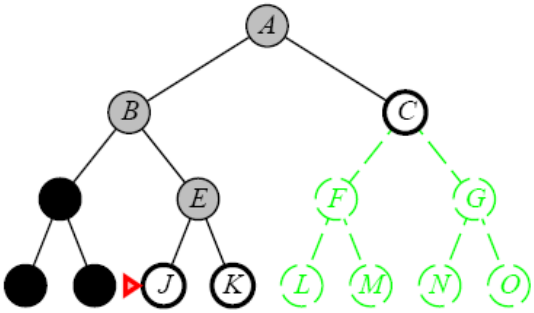
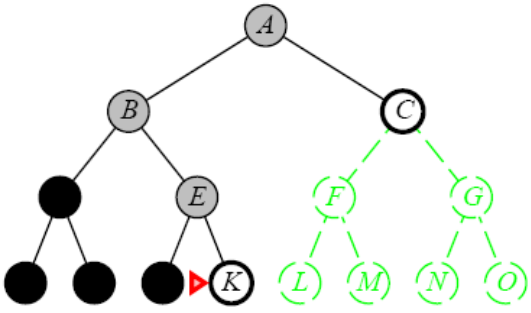
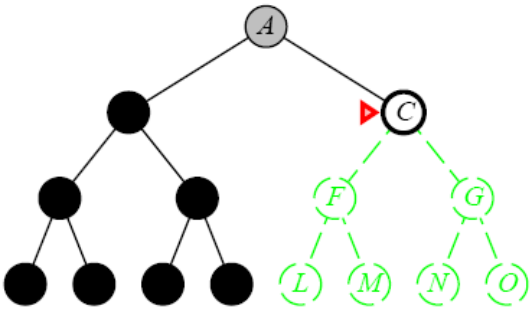
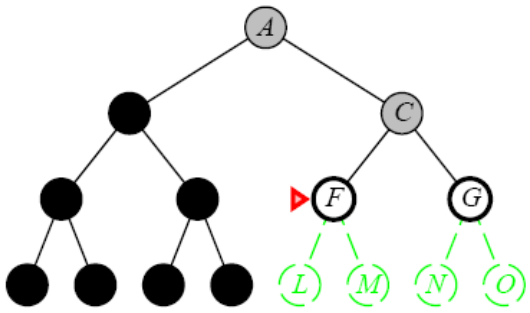


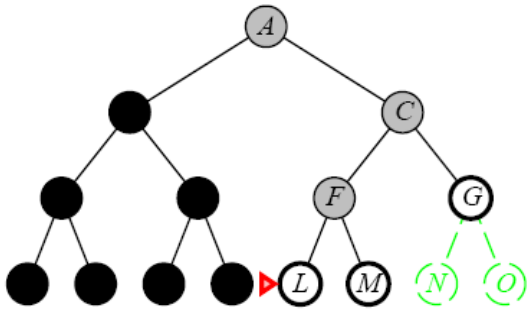
---

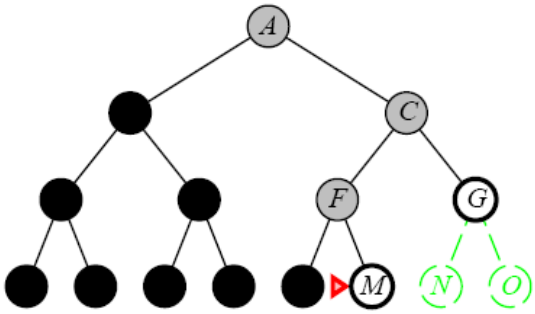
۱ - stack، **یادآوری**- ساختمان داده‌ای است که در آن، عنصرهایی که دورتر وارد شده‌اند، زودتر خارج می‌شوند؛ به عبارت دیگر، پشته به صورت LIFO است؛ *LIFO*، کوتاه شده‌ی «*L*ast *I*n *F*irst *O*ut» می‌باشد.



با استفاده از عمل push(قراردادن)، عنصری را در داخل پشته قرار می‌دهیم و با استفاده از عمل pop، آخرین عنصری را که در داخل پشته قرار داده‌ایم، برمی‌داریم.





ویژگی‌های جستجوی اوّل عمق

کامل بودن؟؟ (☑**کنکور سراسری فنّـآوری اطّلاعات سال ۸۳**– نه)؛ در موردهایی که عمق نامحدود است و در مورد حلقه‌ها درست کار نمی‌کند.

زمان؟؟ ($O(b^m)$؛ در صورتی که m به مراتب بزرگ‌تر از d(عمق راه حلّ بهینه) باشد، بسیار بد است؛ ولی اگر راه حل‌ها پیچیده باشند، این روش به مراتب سریع‌تر از جستجوی اوّل سطح می‌باشد. توجّه کنید که m، بیشینه‌ی عمق درخت می‌باشد. همچنین در برخی از موردها m ممکن است نامحدود باشد.

مطلب مهم:

☑**کنکور سراسری فنّـآوری اطّلاعات سال ۸۵: فضا؟؟** O(bm)، که فضایی خطّی می‌باشد. فقط نیاز به نگهداشتن شاخه‌ای که به تازگی جستجو شده است، داریم و این حسن بزرگ جستجوی اوّل عمق می‌باشد.

پیمایش(جستجوی) معکوس[۱]: گونه‌ای از جستجوی اوّل عمق است که فضا(حافظه)ی کم‌تری مصرف می‌کند. در این روش در هر لحظه به جای تولید همه‌ی جانشینان، فقط یک جانشین تولید می‌شود؛ و هر گره‌ی به صورت جزئی توسعه داده شده، گره‌ی بعدی‌ای را که باید تولید شود، به یاد می‌آورد. در این روش فقط به حافظه‌ی O(m) به جای O(bm) نیاز داریم. این روش حُقّه‌ای برای صرفه‌جویی در فضا و زمان است. در این روش در زمانی که به عقب بر می‌گردیم تا جانشین بعدی را تولید کنیم باید بتوانیم هر تغییر(اصلاح)ی[۲] را لغو[۳] کنیم.[۴]

بهینگی؟؟ نه؛ ما ممکن است یک راه حل را در عمق n، تحت یک فرزند، بدون دیدن یک راه حلّ کوتاه‌تر، تحت فرزند دیگر پیدا نماییم.

---

۱ – Backtracking search؛ برای آگاهی‌های بیش‌تر به فصل «مسأله‌های برآورده‌سازی محدودیّت» همین کتاب مراجعه کنید.

۲ – modification

۳ – undo

۴ – کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌ی ۸۷

اوّل، یک سؤال

چرا ما به الگوریتم‌هایی که تمام گره‌ها را توسعه می‌دهند، می‌پردازیم؟؛ آیا روشی سریع‌تر وجود ندارد؟؛ بسیاری از مسأله‌هایی که به آنها می‌پردازیم، NP– کامل می‌باشند و در مورد آنها هیچ الگوریتم با زمان چندجمله‌ای شناخته شده‌ای وجود ندارد و بدتر از آن، تعدادی هم غیرقابل تخمین می‌باشند.

جستجوی اوّل سطح

اگر جستجوی اوّل سطح را با استفاده از یک صف پیاده‌سازی کنیم، تمام گره‌های در سطح n، قبل از هر گره در سطح n+1 ارزیابی می‌شوند و این کار دارای این حسن است که کامل می‌باشد و در زمانی که هزینه‌ی هر گام به صورت یکسان می‌باشد، راه حلّ بهینه را پیدا خواهد نمود و دارای این ضعف است که به فضایی به صورت نمایی نیازمندیم.

جستجوی اوّل عمق

اگر جستجوی اوّل عمق را با استفاده از پشته پیاده‌سازی نماییم؛ ابتدا یک گره توسعه داده می‌شود، سپس فرزندانش توسعه داده می‌شوند و... و در هر لحظه فقط یک شاخه از درخت جستجو در حافظه نگهداری می‌شود. حسن این روش این است که: به فضایی به صورت خطّی نیازمندیم و عیب آن این است که: ناکامل و غیربهینه می‌باشد.

جلوگیری از جستجوی اوّل عمق نامحدود

مطلب مهم:

چند روش برای جلوگیری از جستجوی اوّل عمق نامحدود وجود دارد، یکی استفاده از closed-list است[۱]؛ که ممکن است همیشه به ما کمک نکند؛ در این صورت ما باید تعداد زیادی گره را به صورت نمایی در حافظه نگهداری نماییم. دو روش دیگر، جستجوی با عمق محدود شده و جستجوی عمیق شونده‌ی تکراری جستجوی اوّل عمق می‌باشند:

جستجوی با عمق محدود شده

مطلب مهم:

با محدود نکردن عمق l(الْ)، با جستجوی اوّل عمق برابر است. *در واقع جستجوی با عمق محدود شده همان جستجوی اوّل عمق است که در آن عمق جستجو محدود شده است (تا یک عمقی در درخت پایین می‌رویم و بیش‌تر از آن پایین نمی‌رویم) و عملّیات جستجو، در این عمق متوقّف می‌شود؛ در اینجا مشکل دیگری به وجود می‌آید و آن این است که اگر راه حل در عمقی بیش‌تر از l باشد، چطور؟؛* چگونه ما یک l خوب را به دست آوریم؟؛

در مسأله‌ی رومانی می‌دانیم که بیست(۲۰) شهر وجود دارد، بنابراین، l=19 است و این روش یک انتخاب خوب می‌باشد. در مورد پازل ۸– تایی چطور؟.

پیاده‌سازی بازگشتی:

الگوریتم

تابع Depth-Limited-Search(problem,limit)، مقادیر soln[۲]، یا fail[۳] و یا cutoff[۴] را برمی‌گرداند

Recursive-DLS(Make-Node(Initial-State[problem]),problem,limit)

---

۱ – **یادآوری**– همان طور که در گذشته هم گفتیم، با استفاده از closed-list، یک لیست از وضعیّت‌های توسعه داده شده را نگهداری می‌نماییم.

۲ – soln(solution)، به معنی «راه حل» است.

۳ – در لغت به معنی «موفّق نشدن» است.

۴ – در اینجا در لغت به معنی «قطع جریان» می‌باشد.

تابع Recursive-DLS(node,problem,limit)، مقادیر soln، یا fail و یا cutoff را برمی‌گرداند

cutoff-occurred[1]? ← false

در صورتی که Goal-Test(problem,State[node]) صحیح بود، node را برگردان (به عبارت دیگر در صورتی که مقدار تابع درست بود، node را برگردان)

در غیر این صورت اگر Depth[node]=limit بود، مقدار cutoff را برگردان

در غیر این صورت

برای هر successor(جانشین) در Expand(node,problem) کارهای زیر را انجام بده:

result ← Recursive-DLS(successor,problem,limit)

اگر result=cutoff بود، آنگاه مقدار true را در cutoff-occurred? بریز

در غیر این صورت اگر result < > failure بود، result را برگردان (< >، علامت نامساوی می‌باشد)

پایان شرط

پایان حلقه

پایان شرط

در صورتی که cutoff-occurred? صحیح بود، cutoff را برگردان و در غیر این صورت failure را برگردان

پایان الگوریتم

جستجوی عمیق شونده‌ی تکراری

مطلب مهم:

بر مبنای ایده‌ی جستجوی با عمق محدود شده توسعه می‌یابد؛ جستجوی با عمق محدود شده را ابتدا با عمق l(اِلْ) برابر یک(l=1)؛ سپس با عمق l=2 و... انجام دهید؛ سرانجام l=d می‌شود، این به این معنی می‌باشد که جستجوی عمیق شونده‌ی تکراری، کامل است. اشکال این روش این است که برخی از گره‌ها تولید شده‌اند و چند بار توسعه داده شده‌اند؛

---

۱ - در لغت به معنی «روی داده، اتّفاق افتاده» می‌باشد.

در سطح یک، تعداد b گره، تعداد d مرتبه تولید می‌شوند؛ در سطح دو، تعداد b^2 گره، d-1 بار تولید می‌شوند و... در نتیجه در سطح d، تعداد b^d گره، یکبار تولید می‌شوند. در این روش مجموع زمان اجرا برابر $O(b^d)$ می‌باشد؛ که اندکی کم‌تر از تعداد گره‌های تولید شده در جستجوی اوّل سطح می‌باشد و هنوز به حافظه‌ای به صورت خطّی نیازداریم. الگوریتم این روش به صورت زیر است:

الگوریتم

تابع Iterative-Deepening-Search(problem)، یک راه حل را برمی‌گرداند

ورودی‌ها، problem، که یک مسأله است، می‌باشد

برای depth مساوی با صفر تا بینهایت(∞)، کارهای زیر را انجام بده

result ← Depth-Limited-Search(problem,depth)[1]

در صورتی که result< >cutoff بود، result را برگردان

پایان حلقه

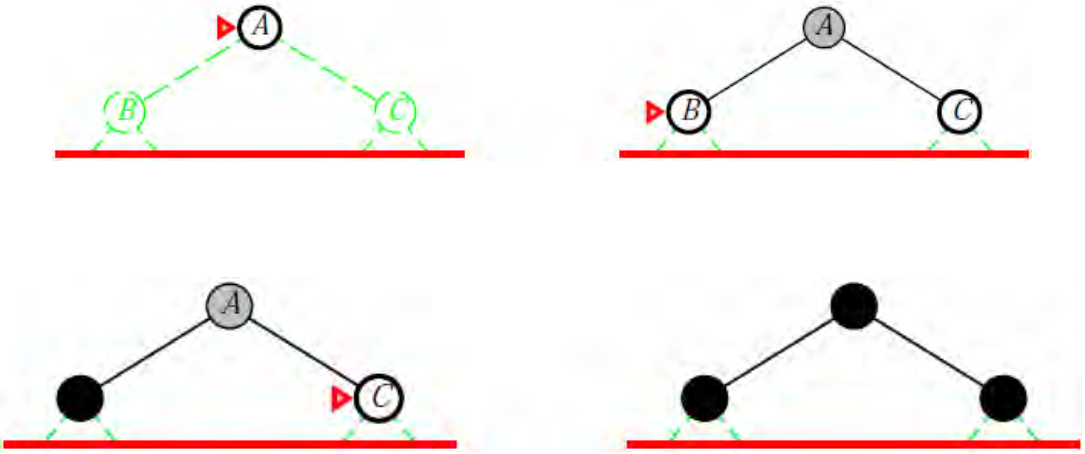
پایان الگوریتم

مثال- شکل‌ها را از سمت چپ به راست(→) دنبال نمایید.

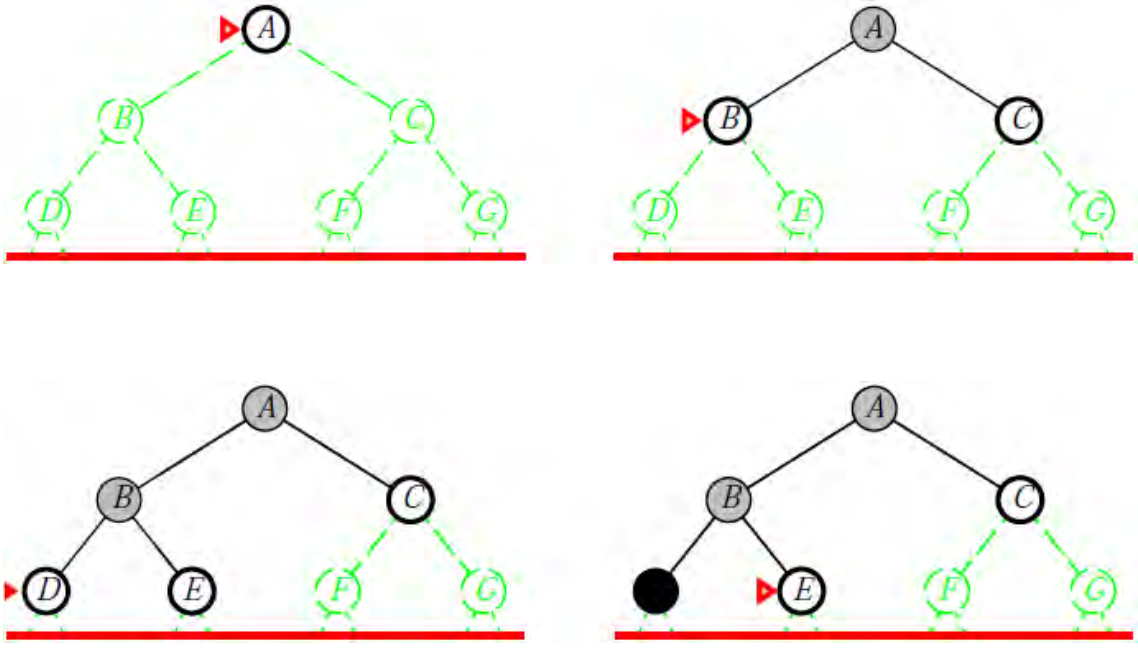
جستجوی عمیق شونده‌ی تکراری، با l=0

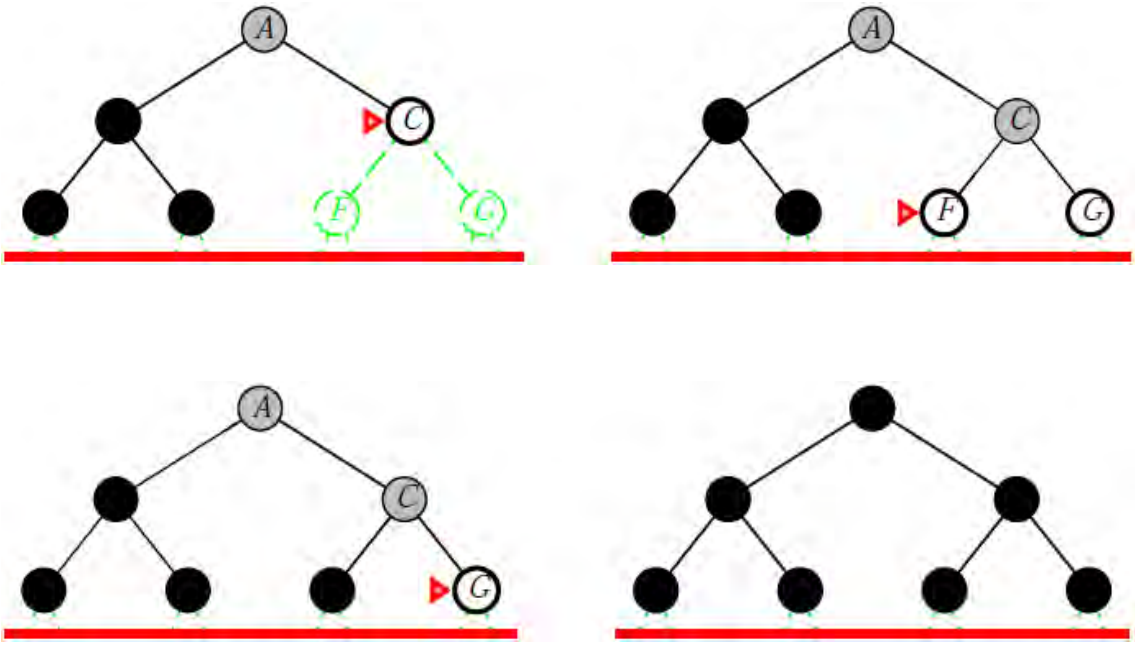
۱ - این تابع همان تابعی است که در قسمت «پیاده‌سازی بازگشتی» «جستجوی با عمق محدود شده» آمده است.

جستجوی عمیق شونده‌ی تکراری، با l=1

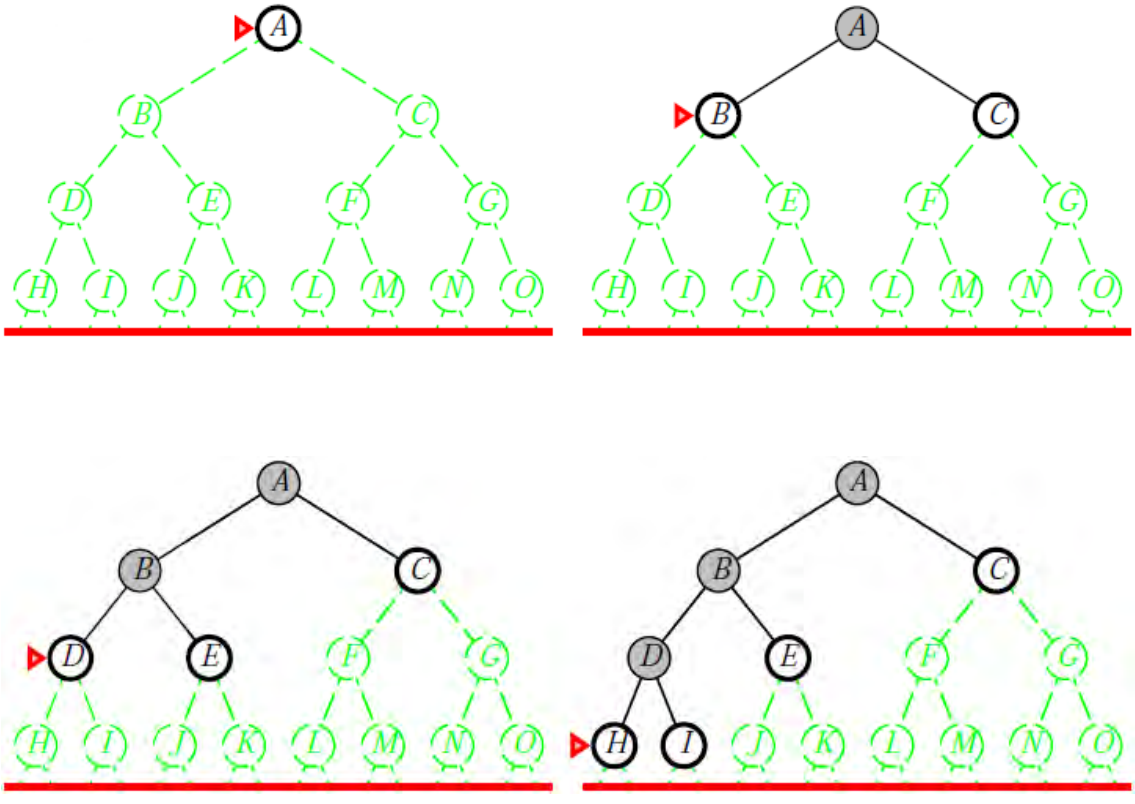


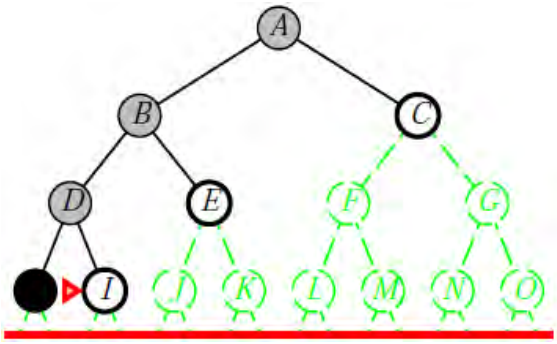
جستجوی عمیق شونده‌ی تکراری، با l=2

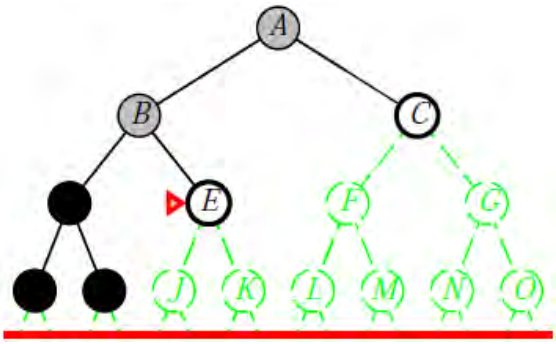


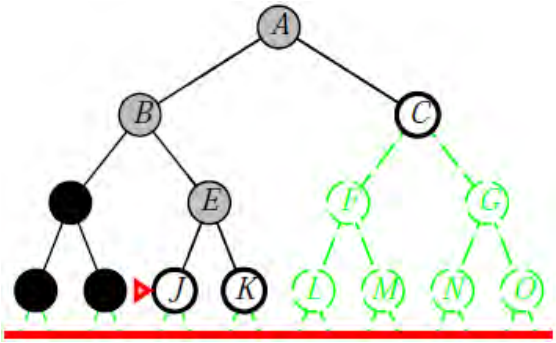


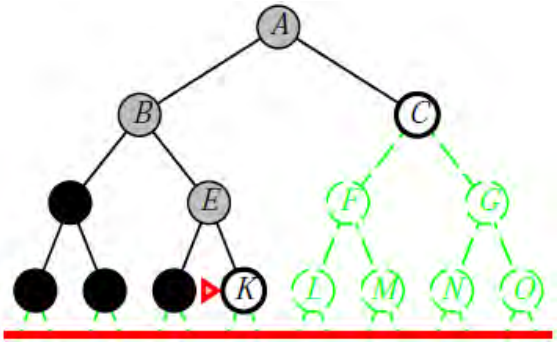
جستجوی عمیق شونده‌ی تکراری، با l=3

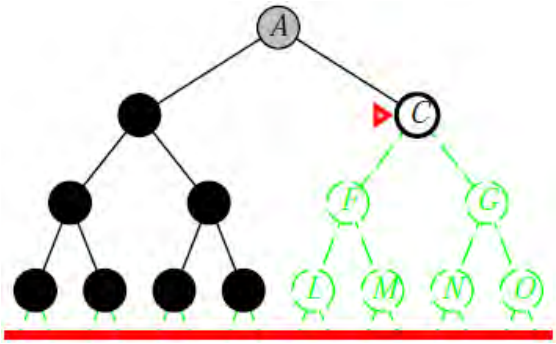


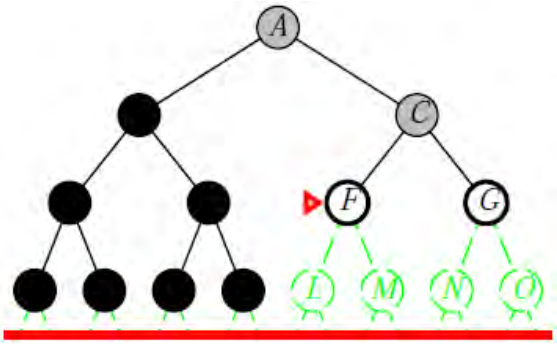

A
B
C
D
E
F
G
I
J
K
L
M
N
O

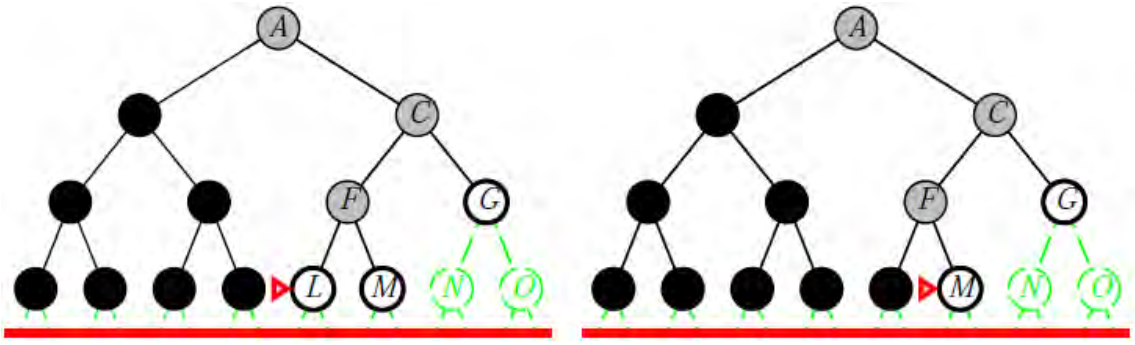

A
B
C
E
F
G
J
K
L
M
N
O


A
B
C
E
F
G
J
K
L
M
N
O


A
B
C
E
F
G
K
L
M
N
O


A
C
F
G
L
M
N
O


A
C
F
G
L
M
N
O



مطلب مهم:

جستجوی عمیق شونده‌ی تکراری، جستجوی اوّل سطح و اوّل عمق را با هم ترکیب می کند و لایه‌ها را شبیه جستجوی اوّل سطح به وجود می‌آورد، امّا در هر اجرا یک اوّل عمق را انجام می‌دهد که باعث صرفه‌جویی در فضا می‌شود. در این روش همان طور که دیدیم، وضعیّت‌ها چند بار توسعه داده می‌شوند.

در واقع جستجوی عمیق شونده‌ی تکراری، شبیه جستجوی اوّل سطح می‌باشد که در آن همه‌ی گره‌های در عمق n، قبل از هر گره در عمق n+1 برّرسی می‌شوند.

ویژگی‌های جستجوی عمیق شونده‌ی تکراری

کامل بودن؟؟ (☑ **کنکور سراسری فنّـآوری اطّلاعات سال ۸۳**- بله)

☑ **مطلب کنکور سراسری مهندسی کامپیوتر سال ۸۶- زمان؟؟**

$$(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$$

فضا؟؟ O(bd)

بهینگی؟؟ (☑ **کنکور سراسری فنّـآوری اطّلاعات سال ۸۳**- بله)، اگر هزینه‌ی گام برابر یک باشد.

جریان معکوس(برگشت به عقب)[۱]

در زمانی که جستجوی اوّل عمق و همتایانش؛ یعنی، روش‌های «اوّل عمق با عمق محدود شده» و «جستجوی اوّل عمق عمیق شونده‌ی تکراری»، که در گذشته در همین فصل دیدیم، به وضعیّت عدم موفّقیّت می‌رسند، چه اتّفاقی می‌افتد؟: به بالای گره‌ی پدر(والد) می‌روند و برادر بعدی را امتحان می‌کنند و فرض را بر این قرار می‌دهند که جدیدترین عملکرد انتخاب شده، عملکردی است که باعث عدم موفّقیّت شده است. این روش، *جریان معکوس به ترتیب وقوع*[۲] نام دارد و جدیدترین کاری را که انجام داده‌اید، بی‌اثر می‌نماید؛ مسأله‌ای که در این مورد وجود دارد، این است که عدم موفّقیّت شاید نتیجه‌ای از یک تصمیم‌گیری قبلی باشد. افزایش تدریجی عمق می‌تواند به شما برای محدود کردن اندازه‌ی فضای جستجو کمک نماید. *جریان معکوس هوشمند*[۳]، برای پیداکردن دلیل برای عدم موفّقیّت و غیرفعّال کردن جستجو برای نقطه‌ای که عدم موفّقیّت برای آن رخ داده است، تلاش می‌نماید و می‌تواند جدیدترین متغیّر ناسازگار را غیرفعّال نماید. جریان معکوس همچنین می‌تواند *برّرسی مستقیم*[۴] را انجام دهد؛ به این صورت که برّرسی می‌کند که «آیا یک انتساب درست مقدارها برای این گره وجود دارد؟».

جستجوی دوطرفه[۵]

دو جستجوی شبیه را*[برای پیدا کردن یک مسیر]* اجرا می‌نماید، یکی در جهت وضعیّت اوّلیّه و دیگری در خلاف جهت و از طرف وضعیّت هدف و در زمانی که دو جستجو به هم می‌رسند، جستجو خاتمه می‌یابد.

---

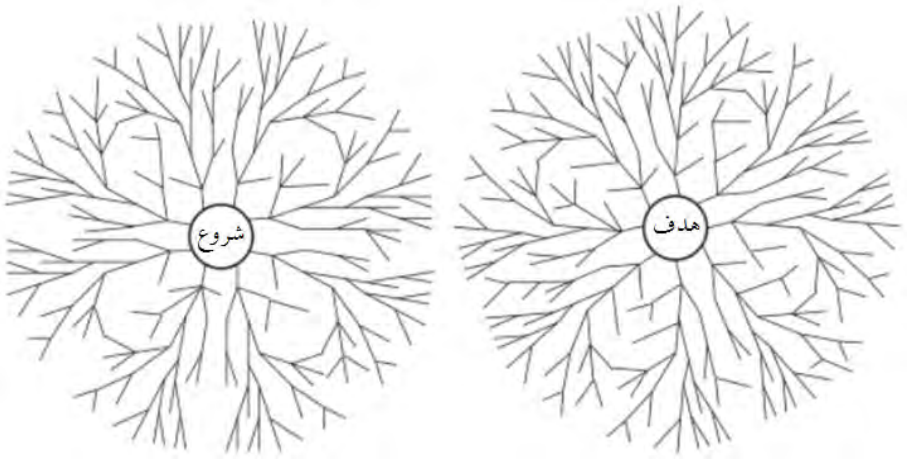
۱ - backtracking؛ کلمه‌ی backtrack در لغت به معنی «از همان راه رفته برگشتن است.»
(http://farsilookup.com/e2p/seek.jsp?lang=fa&word=backtracking؛ این پایگاه اینترنتی، دارای «فرهنگ واژگان انگلیسی به فارسی»، «فرهنگ واژگان فارسی به انگلیسی» و «فرهنگ واژگان فارسی به فارسی» است.)

۲ - chronological backtracking

۳ - intelligent backtracking

۴ - forward checking

۵ - bidirectional search



ویژگی‌های جستجوی دوطرفه

کامل؟؟ در صورتی که هر دو جستجو، اوّل سطح باشند، کامل است.

زمان؟؟ $O(b^{d/2})$

☑ **کنکور سراسری فنّـآوری اطّلاعات سال ۸۸: فضا؟؟** $O(b^{d/2})$

بهینه؟؟ در صورتی که هر دو روش، اوّل سطح باشند، بهینه می‌باشد.

این روش وقتی که فقط یک وضعیّت شروع و یک وضعیّت پایان داشته باشیم، خوب است. گاهی وقت‌ها با این روش راه حل با سرعت بیش‌تری پیدا می‌شود.

به وجود آوردن هماهنگی، میان دو جستجو، یکی از اِشکال‌ها[1]ی این روش است.

مقایسه‌ی روش‌های جستجو

نکته‌ی مهم:

در جدول زیر ملاحظه‌هایی که در مورد پاسخ‌ها وجود دارند را در نظر بگیرید؛ این ملاحظه‌ها در درس هم بیان شده‌اند؛ مثلاً جستجوی اوّل سطح، در صورتی بهینه است که هزینه‌ی گام برابر یک باشد.

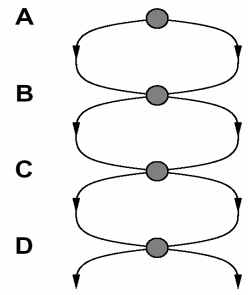
۱- drawbacks

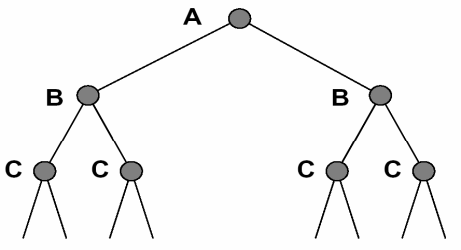
مقیاس[1]↓/روش←	اوّل سطح	با هزینه‌ی یکسان	اوّل عمق	عمق محدود شده	عمیق شونده‌ی تکراری	جستجوی دوطرفه
کامل بودن؟؟	بله	بله[2]	نه	بله، در صورتی که l(الْ)=>d باشد	بله	بله[3]
زمان؟؟	b^{d+1}	$b^{C^*/\varepsilon}$	b^m	b^l	b^d	$b^{d/2}$
فضا؟؟	b^{d+1}	$b^{C^*/\varepsilon}$	bm	bl	bd	$b^{d/2}$
بهینگی؟؟	بله[4]	بله[5]	نه	نه	بله	بله

در جدول بالا، b: فاکتور انشعاب، d: عمق کم‌عمق‌ترین راه حل، m: بیشینه‌ی عمق درخت جستجو، l: محدودیّت عمق، C^*: هزینه‌ی راه حلّ بهینه و ε: دست‌کم(لااقلّ) هزینه‌ی هر عملکرد می‌باشد.

حالت(وضعیّت)های تکرار شده

عدم موفّقیّت در پیداکردن حالت‌های تکرار شده می‌تواند یک مسأله‌ی خطّی را تبدیل به یک مسأله‌ی نمایی کند.





---

۱ – Criterion

۲ – در صورتی که ε بزرگ‌تر از صفر باشد.

۳ – در صورتی که هر دو جستجو اوّلْ سطح باشند، کامل است.

۴ – در صورتی که هزینه‌ی گام‌ها با هم برابر باشند.

۵ – در صورتی که کامل باشد.

راه حلّ این مشکل، نگهداری تاریخچه است؛ گره‌های ملاقات شده (توسعه داده شده) در لیست closed نگهداری می‌شوند و گره‌ی جاری اگر در گذشته در لیست closed بوده، حذف می‌شود. مشکل این روش این است که بهینگی ممکن است از بین برود، چون همه‌ی گره‌ها در حافظه نگهداری می‌شوند.

الگوریتم جستجوی گراف

در مسأله‌هایی که دارای تعداد زیادی حالت‌های تکرار شده هستند، ولی فضای حالت کوچکی دارند، جستجوی گرافی می‌تواند به مراتب مؤثرتر از جستجوی درختی باشد. در جستجوی گرافی، ما از لیستی به نام closed استفاده می‌کنیم، که همه‌ی وضعّیت‌هایی که تا کنون با آنها مواجه شده‌ایم را نگهداری می‌کند و یک گره را تنها در صورتی به fringe اضافه می‌کنیم که در لیست closed نباشد.[1]

الگوریتم

تابع Graph-Search(problem,fringe)، یک راه حل یا عدم موفّقیّت را برمی‌گرداند

یک مجموعه‌ی خالی را به closed تخصیص بده

fringe ← Insert(Make-Node(Initial-State[problem]),fringe)

در حلقه کارهای زیر را انجام بده

در صورتی که fringe خالی است، عدم موفّقیّت را برگردان

node ← Remove-Front(fringe)

در صورتی که Goal-Test(problem,State[node]) موفّقیّت‌آمیز بود، node را برگردان

در صورتی که State[node] در closed نبود

State[node] را به closed اضافه نما

InsertAll(Expand(node,problem),fringe) را در fringe بریز

پایان شرط

پایان حلقه

---

۱ – مطلب‌های درس «هوش مصنوعی» استاد، «اَلِسْتِرْ نُتْ(Alistair Knott)»، دانشکده‌ی علوم کامپیوتر دانشگاه اُتاگوی کشور نیو زیلَند(New Zealand)

پایان الگوریتم

چکیده‌ی مطلب‌های فصل چهارم

فرمول‌بندی مسأله معمولاً به در کنار هم قرار دادن جزئیّات دنیای واقعی برای تعریف یک فضای حالت که بتواند به طور عملی کاوش شود، نیاز دارد.

فرمول‌بندی یک مسأله‌ی جستجو، با استفاده از وضعیّت اوّلیّه، آزمون هدف، عملیّاتی که باید انجام شوند، تابع جانشین(مولّد) و هزینه‌ی مسیر، منجر به جستجو در یک فضای حالت با استفاده از یک درخت جستجو می‌شود.

درخت جستجو، برابر با فضای حالت نمی‌باشد. فضای حالت، از حالت(وضعیّت)ها و عملگرها درست شده است و یک گراف است. درخت جستجو، کاوشی معیّن از فضای جستجو است.

الگوریتم‌های جستجوی ناآگاهانه عبارتند از: جستجوی اوّل سطح، جستجوی اوّل عمق، جستجوی با هزینه‌ی یکسان، جستجوی با عمق محدود شده و جستجوی عمیق شونده‌ی تکراری.

در مسأله‌هایی که دارای تعداد زیادی حالت‌های تکرار شده هستند، ولی فضای حالت کوچکی دارند، جستجوی گرافی می‌تواند به مراتب مؤثرتر از جستجوی درختی باشد.

یادآوری یا تکمیل مطلب‌های فصل چهارم

مطلب- ☑کنکور سراسری فنّـآوری اطّلاعات سال ۸۷: فرمول‌بندی هدف، اوّلین گام در حلّ مسأله است؛ سپس فرمول‌بندی مسأله صورت می‌گیرد.[۱]

تعریف- فضای حالت را تعریف کنید.

جواب- مجموعه‌ی همه‌ی وضعیّت‌هایی که از وضعیّت اوّلیّه، به وسیله‌ی هر رشته(دنباله) از عملکردها قابل دسترس هستند، فضای حالت گفته می‌شود.[۲]

تعریف- مرز(frontier) یا حاشیه(fringe) را تعریف نمائید.

جواب- به مجموعه‌ی همه‌ی گره‌های برگی قابل دسترس برای توسعه، در هر نقطه‌ی داده شده گفته می‌شود.[۳]

تعریف- جستجوی ناآگاهانه(کور) را تعریف کنید.

جواب- این روش‌ها از هیچ اطّلاعات اضافی خارج از تعریف مسأله استفاده نمی‌کنند.[۴]

---

۱ - مطلب‌های درس «روبوتیک هوشمند» استاد، «مارک ای. پِرکُوْسْکی»، استاد مهندسی برق دانشگاه ایالت پُرتلَند کشور آمریکا

۲ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌ی ۶۷

۳ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌ی ۷۵

۴ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌ی ۸۱

تعریف- جستجوی بهینه را تعریف کنید.

جواب- برای پیدا کردن کم هزینه‌ترین راه حل، در میان راه حل‌ها ضمانت شده است.[۱]

تعریف- جستجوی کامل را تعریف کنید.

جواب- برای پیدا کردن یک راه حل، در صورت وجود، ضمانت شده است.[۲]

تست- یک الگوریتم جستجو، در صورتی کامل است که:
۱) همیشه راه حلّ بهینه را پیدا کند.
۲) همیشه یک راه حل را در صورت وجود پیدا کند.
۳) همه‌ی راه حل‌های ممکن را پیدا کند.
۴) هیچ راه حلّی را پیدا نکند.

جواب- گزینه‌ی «۲» درست است.[۳]

تست- کدام گزینه پیچیدگی فضایی را تعریف می‌کند؟
۱) بیشینه‌ی عمق فضای حالت
۲) تعداد گره‌های توسعه داده شده
۳) بیشینه‌ی فاکتور انشعاب درخت جستجو
۴) بیشینه‌ی تعداد گره‌های نگهداری شده در حافظه
۵) عمق کم هزینه‌ترین راه حل

جواب- گزینه‌ی «۴» درست است.[۴]

تعریف- توسعه دادن یک وضعیّت را تعریف کنید.

جواب- به کار بردن هر عمل مُجاز، به وضعیّت، و تولید یک مجموعه‌ی جدید از وضعیّت‌ها.[۱]

---

۱ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۲ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۳ - آزمون میان‌ترم درس «هوش مصنوعی» استاد، «کَتی مِکْکُوْنْ(Kathy McKeown)»، دانشکده‌ی علوم کامپیوتر دانشگاه کلمبیای کشور آمریکا، ۲۶ اکتبر سال ۲۰۰۶ میلادی

۴ - آزمون میان‌ترم درس «هوش مصنوعی» استاد، «روت شولتز»، دانشکده‌ی مهندسی برق و فنّآوری اطلاعات دانشگاه کوئینزلند کشور استرالیا، سال ۲۰۱۲ میلادی

تعریف- فاکتور انشعاب را تعریف کنید.

جواب- بیشینه‌ی تعداد جانشینان هر گره است.[۲]

مسأله‌ی کشیش‌ها[۳] و آدمخواران[۴]- سه کشیش( ) و سه آدمخوار( ) در ساحل سمت چپ یک رودخانه هستند. یک قایق( ) هم هست که می‌تواند یک یا دو نفر را حمل کند. راهی برای بردن همه به سمت راست ساحل رودخانه بیابید بدون اینکه در هر لحظه تعداد کشیش‌ها از تعداد آدمخواران، در هر سمت رودخانه کم‌تر شود.

جواب-

وضعیّت اوّلیّه-

وضعیّت نهائی(هدف)-

وضعیّت‌ها، سه عدد به صورت (i, j, k) هستند که i، بیان کننده‌ی تعداد کشیش‌ها، j، بیان کننده‌ی تعداد آدمخوارها و k، بیان کننده‌ی تعداد قایق‌های سمت چپ رودخانه است. بنابراین،

وضعیّت اوّلیّه، به صورت (3, 3, 1) است و

---

۱ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروایِن دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۲ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروایِن دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۳ - missionaries

۴ - cannibals

عملگرها، بردن «یک کشیش»، «یک آدمخوار»، «دو کشیش»، «دو آدمخوار» و «یک کشیش و یک آدمخوار»، از عرض رودخانه، در جهت داده شده هستند؛ توجّه کنید که ده(۱۰) عملگر داریم.

آزمون هدف– رسیدن به وضعیّت (0, 0, 0)

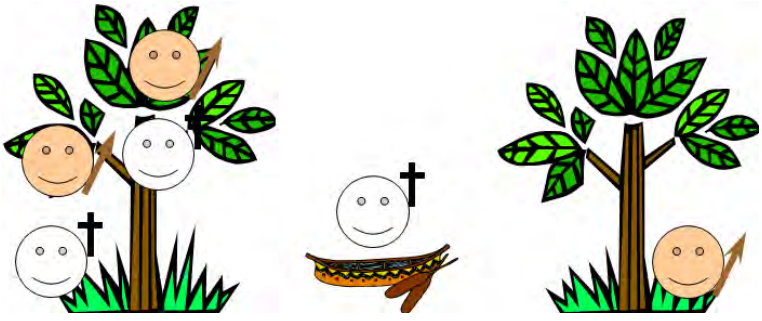
هزینه‌ی مسیر– تعداد عبورها

وضعیّت اوّلیّه– (3, 3, 1):

عبور یک کشیش و یک آدمخوار:

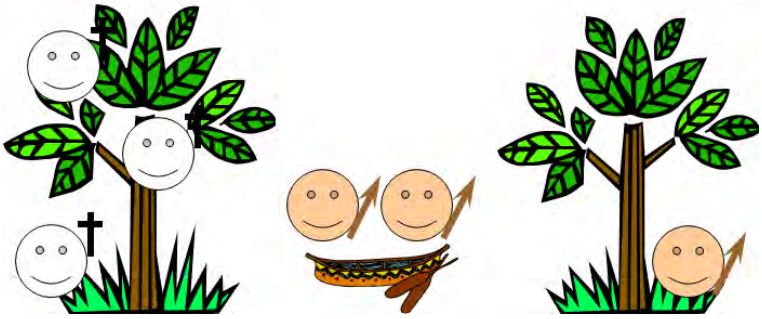
وضعیّت (2, 2, 0):

یک کشیش برمی‌گردد:



وضعیّت (3, 2, 1):

دو آدمخوار عبور می‌کنند:

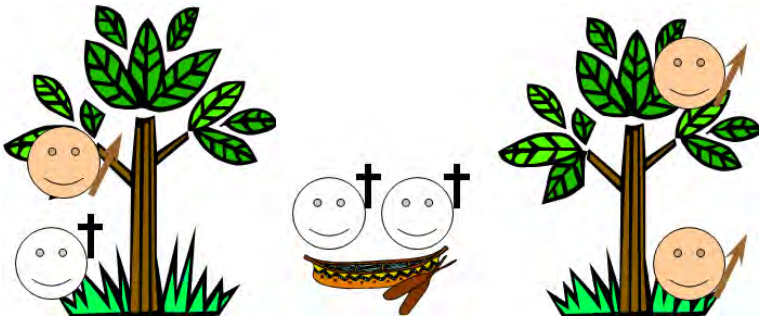


وضعیّت (3, 0, 0):

یک آدمخوار برمی‌گردد:

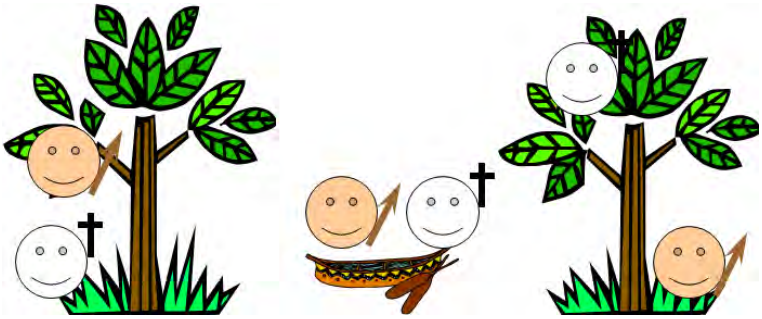
وضعیّت (3, 1, 1):

دو کشیش عبور می‌کنند:



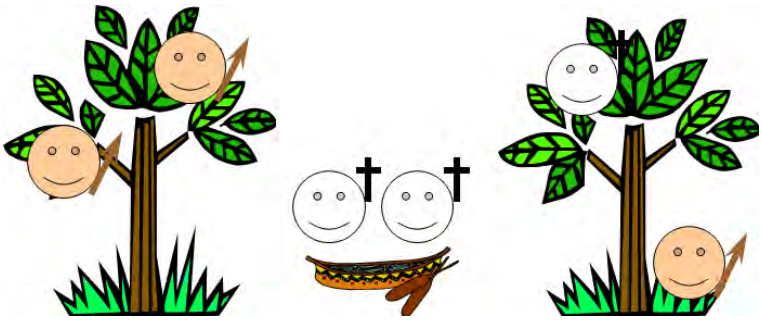
وضعیّت (1, 1, 0):

یک کشیش و یک آدمخوار برمی‌گردند:



وضعیّت (2, 2, 1):

دو کشیش عبور می‌کنند:

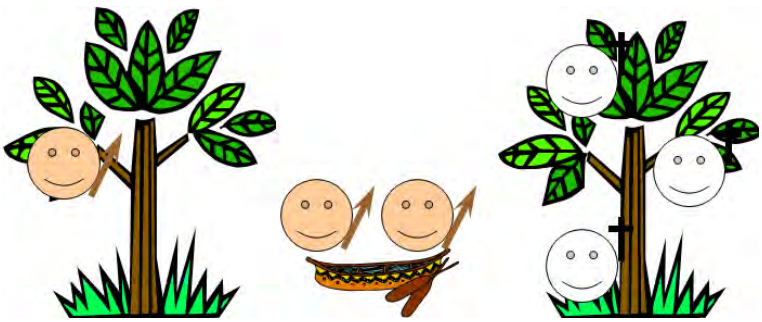


وضعیّت (0, 2, 0):

یک آدمخوار برمی‌گردد:

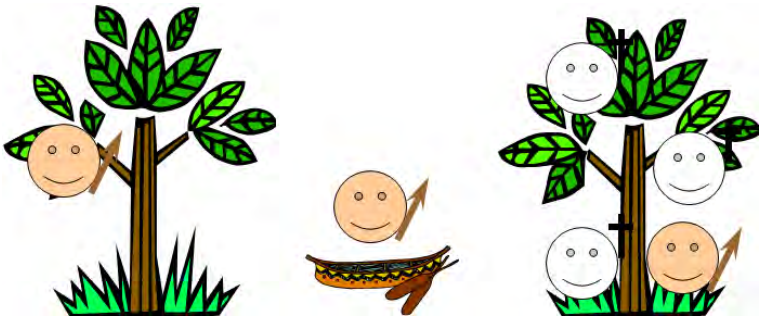
وضعیّت (0, 3, 1):

دو آدمخوار عبور می‌کنند:



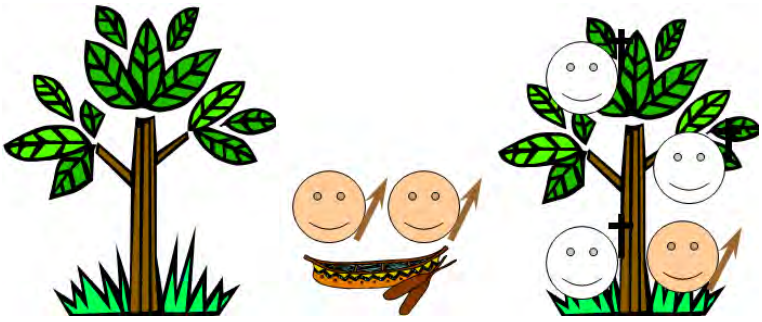
وضعیّت (0, 1, 0):

یک آدمخوار برمی‌گردد:



وضعیّت (0, 2, 1):

دو آدمخوار آخر عبور می‌کنند:



وضعیّت نهایی(0, 0, 0):

در نتیجه راه حل مسیری به صورت زیر است:

$[(3,3,1) \rightarrow (2,2,0) \rightarrow (3,2,1) \rightarrow (3,0,0) \rightarrow (3,1,1) \rightarrow (1,1,0) \rightarrow (2,2,1) \rightarrow (0,2,0) \rightarrow (0,3,1) \rightarrow (0,1,0) \rightarrow (0,2,1) \rightarrow (0,0,0)]$

و هزینه برابر با ۱۱ عبور خواهد بود.[۱]

درست یا غلط– مفهوم‌های«گره»، در یک درخت جستجو و «وضعیّت»، در یک فضای حالت، یکسان هستند.

جواب– «غلط» است؛ یک گره، ساختمان داده‌ای است که شامل یک وضعیّت و اطّلاعات دیگری که به جستجو مربوط هستند، می‌باشد.[۲]

درست یا غلط– جستجوی اوّل سطح همیشه تعداد گره‌های کم‌تری را نسبت به جستجوی اوّل عمق توسعه می‌دهد.

جواب– «غلط» است؛ این به مکان قرار گرفتن گره‌ی هدف بستگی دارد.[۳]

درست یا غلط– جستجوی اوّل سطح حالت خاصّی از جستجوی با هزینه‌ی یکسان است.

جواب– «درست» است؛ زمانی که تمام هزینه‌های گام‌ها یکسان هستند، جستجوی با هزینه‌ی یکسان به جستجوی اوّل سطح تبدیل می‌شود.[۴]

سؤال– اگر جستجوی درختی را روی گراف زیر اجرا کنیم، آیا جستجوی با هزینه‌ی یکسان، هدف بهینه را پیدا می‌کند؟، چرا؟؛ توجّه کنید که هزینه‌ی گام‌ها روی یال‌ها نشان داده شده‌اند.

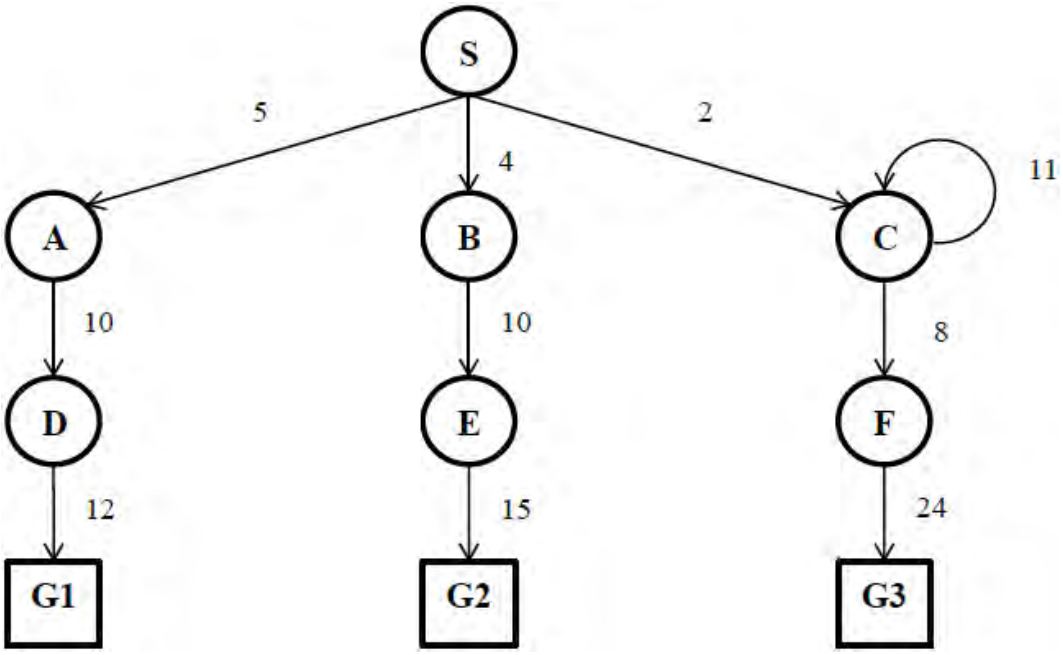
---

۱ – مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

۲ – آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۲۳ سپتامبر سال ۲۰۱۱ میلادی

۳ – آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۷ ژوئن سال ۲۰۱۲ میلادی

۴ – تکلیف‌های خانگی شماره‌ی ۱ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا



جواب– بله؛ چون «هزینه‌ی گام‌ها $\leq \varepsilon >$ صفر» است.[۱]

درست یا غلط– از نظر پیچیدگی فضایی جستجوی اوّل عمق از جستجوی اوّل سطح بهتر است.

جواب– «**درست**» **است؛** از نظر پیچیدگی فضایی روش اوّل عمق به صورت خطّی است، امّا روش اوّل سطح به صورت نمائی است.[۲]

درست یا غلط– هر دو روش جستجوی اوّل سطح و اوّل عمق در بدترین حالت دارای زمان نمائی برای اجرا هستند.

جواب– «**درست**» **است؛** در بدترین حالت هر دو روش دارای پیچیدگی زمانی $O(b^d)$ هستند؛ با فرض اینکه b، فاکتور انشعاب و d، عمق باشد.[۳]

سؤال– مثالی ارائه کنید که نشان دهد روش جستجوی اوّل عمق همیشه بهینه نمی‌باشد.

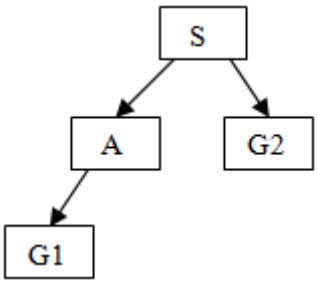
جواب–

---

۱ – آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، زمستان سال ۲۰۱۲ میلادی

۲ – آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۱۴ جولای سال ۲۰۱۱ میلادی

۳ – آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۲۳ سپتامبر سال ۲۰۱۱ میلادی

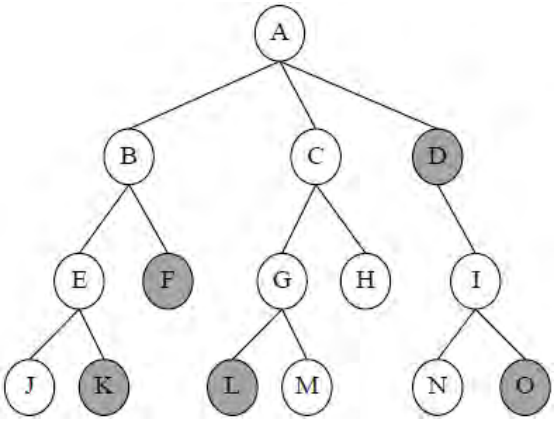


در این مثال هدف بهینه G2 است، ولی روش جستجوی اوّل عمق G1 را پیدا می‌کند.[۱]

درست یا غلط– جستجوی با عمق محدود شده همیشه کامل است.

جواب– **«غلط» است؛** این به عمق گره‌ی هدف بستگی دارد.[۲]

تمرین– با توجّه به درخت زیر به چه ترتیبی گره‌های دارای رنگ زمینه‌ی خاکستری، مثلاً گره‌ی F، در جستجوی عمیق شونده‌ی تکراری توسعه داده می‌شوند؟



جواب– (به ترتیب از چپ به راست):

D FD KFLDO[۳]

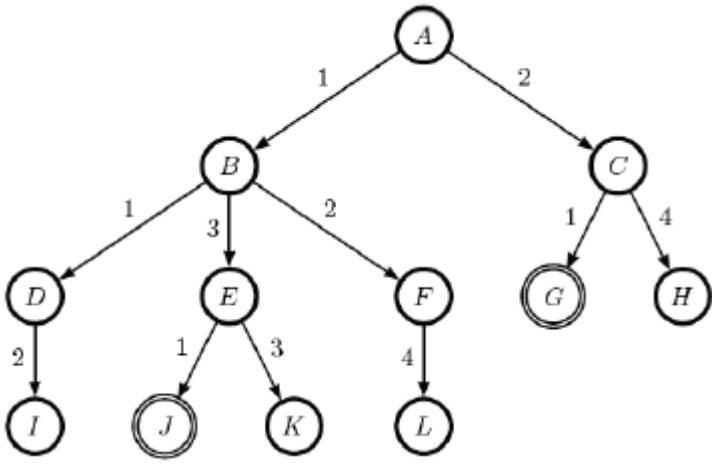
تمرین– درخت زیر یک درخت جستجو برای یک فضای حالت است. [در این درخت] برچسب یال(قُوْسْ)ها، هزینه‌ها را مشخّص می‌کنند و دایره‌های دوخطّی، گره‌های هدف را نشان می‌دهند. برای روش‌های «جستجوی اوّل سطح»، «اوّل عمق» و «جستجوی عمیق شونده‌ی تکراری»، ترتیبی که در آن گره‌ها انتخاب می‌شوند تا اوّلین گره هدف پیدا شود، مشخّص کنید.

---

۱ – تکلیف‌های خانگی شماره‌ی ۲ درس «آشنایی با هوش مصنوعی» استاد، «دیمیتری پاولوف(Dmitry Pavlov)»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، زمستان سال ۲۰۰۱ میلادی

۲ – آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۲۱ فوریّه‌ی سال ۲۰۱۱ میلادی

۳ – آزمون میان‌ترم درس «هوش مصنوعی» استاد، «روت شولتز»، دانشکده‌ی مهندسی برق و فنّآوری اطلاعات دانشگاه کوئینزلند کشور استرالیا، سال ۲۰۱۲ میلادی



جواب-

جستجوی اوّل سطح(از چپ به راست):

A B C D E F G

جستجوی اوّل عمق(از چپ به راست):

A B D I E G

جستجوی عمیق شونده‌ی تکراری(از چپ به راست):

A
A B C
A B C D E F G[1]

درست یا غلط- یک مسیر راه حلّ بهینه برای یک مسأله‌ی جستجوی با هزینه‌های مثبت، هرگز حالت‌های تکراری نخواهد داشت.

جواب- «درست» است؛ برای هر مسیر راه حلّ با وضعیّت‌های تکرار شده، برداشتن حلقه، یک مسیر راه حلّ با هزینه‌ی کم‌تر را به دست خواهد داد.[2]

مطلب- در شکل زیر مسیری که از Arad به Sibiu می‌رود و دوباره به Arad می‌رسد!، نمونه‌ای از یک مسیر حلقه‌ای[3] است.[4]

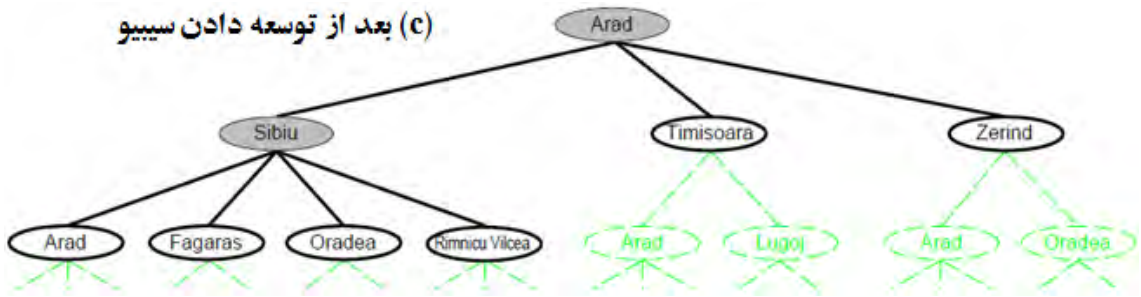
---

۱ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۱۱ ژانویّه‌ی سال ۲۰۱۰ میلادی

۲ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «دن کلین»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۲۰۰۶ میلادی

۳ - loopy path

۴ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌ی ۷۵



تست– در کل، کدامیک از روش‌های جستجوی ناآگاهانه در زمانی که فضای حالت بزرگ است و عمق راه حل مشخّص نیست، ترجیح داده می‌شود؟

۱) جستجوی اوّل عمق

۲) جستجوی اوّل سطح

۳) جستجوی با هزینه‌ی یکنواخت

۴) جستجوی با عمق محدود شده

۵) جستجوی عمیق شونده‌ی تکراری

جواب- گزینه‌ی «۵» درست است.[۱]

سؤال– مسیرهای زائد[۲] چه هنگام به وجود می‌آیند؟

جواب– زمانی که بیش از یک راه برای رسیدن از یک وضعیّت به وضعیّت دیگر وجود داشته باشد.[۳]

مطلب– مسیرهای حلقه‌ای، نوع خاصّی از مفهوم کلّی‌تر مسیرهای زائد هستند.[۴]

مطلب– یک الگوریتم جستجوی درختی کلّی(عمومی)، به تمام مسیرهای ممکن برای پیدا کردن راه حل رسیدگی می‌کند، در حالی که یک الگوریتم جستجوی گرافی، از رسیدگی به مسیرهای زائد اجتناب می‌کند.[۵]

درست یا غلط– جستجوی اوّل عمق گرافی همیشه در مسأله‌های با گراف‌های جستجوی محدود کامل است.

---

۱ – کوییز شماره‌ی ۱ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا

۲ – redundant paths

۳ – کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌ی ۷۶

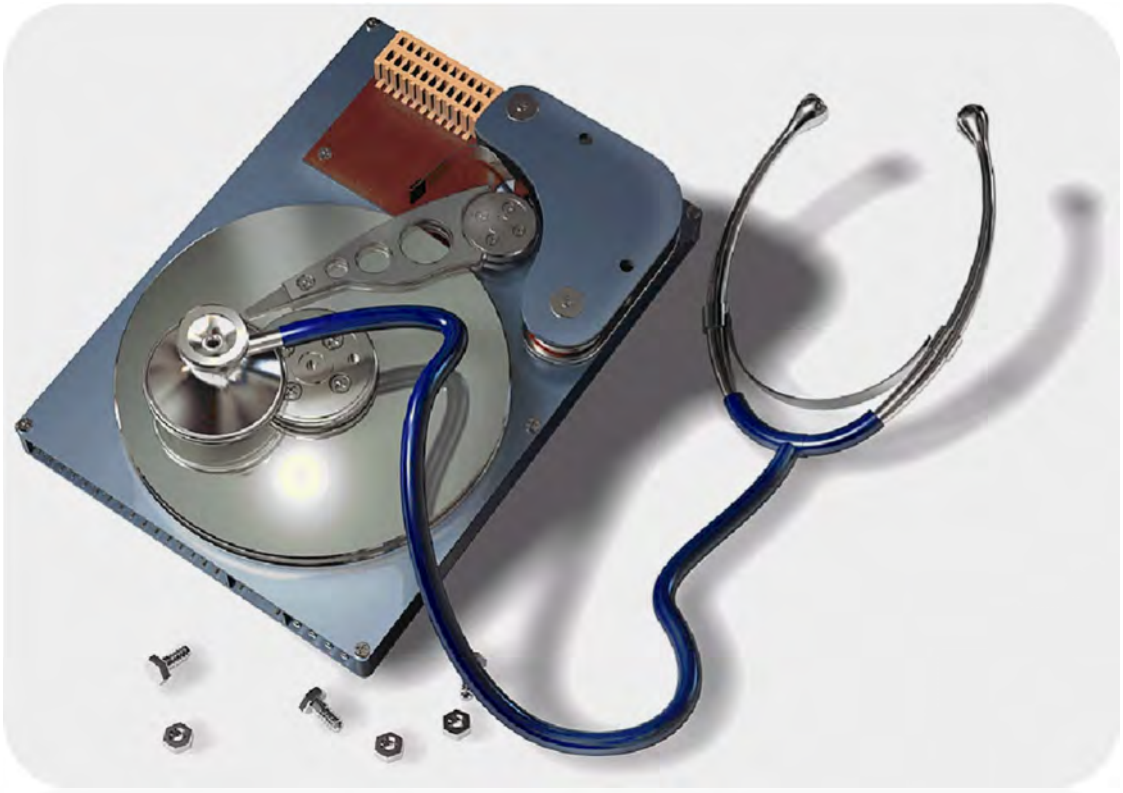
۴ – کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌ی ۷۶

۵ – کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل سوّم، حلّ مسائل با استفاده از جستجو(Solving Problems by Searching)، صفحه‌ی ۱۰۸

جواب- «درست» است؛ این روش به تمام وضعیّت‌های قابل دسترس، در زمان محدود، خواهد رسید، زیرا جستجوی گرافی، وضعیّت‌های تکراری ندارد.[۱]

۱ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «دن کلین»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۲۰۰۶ میلادی

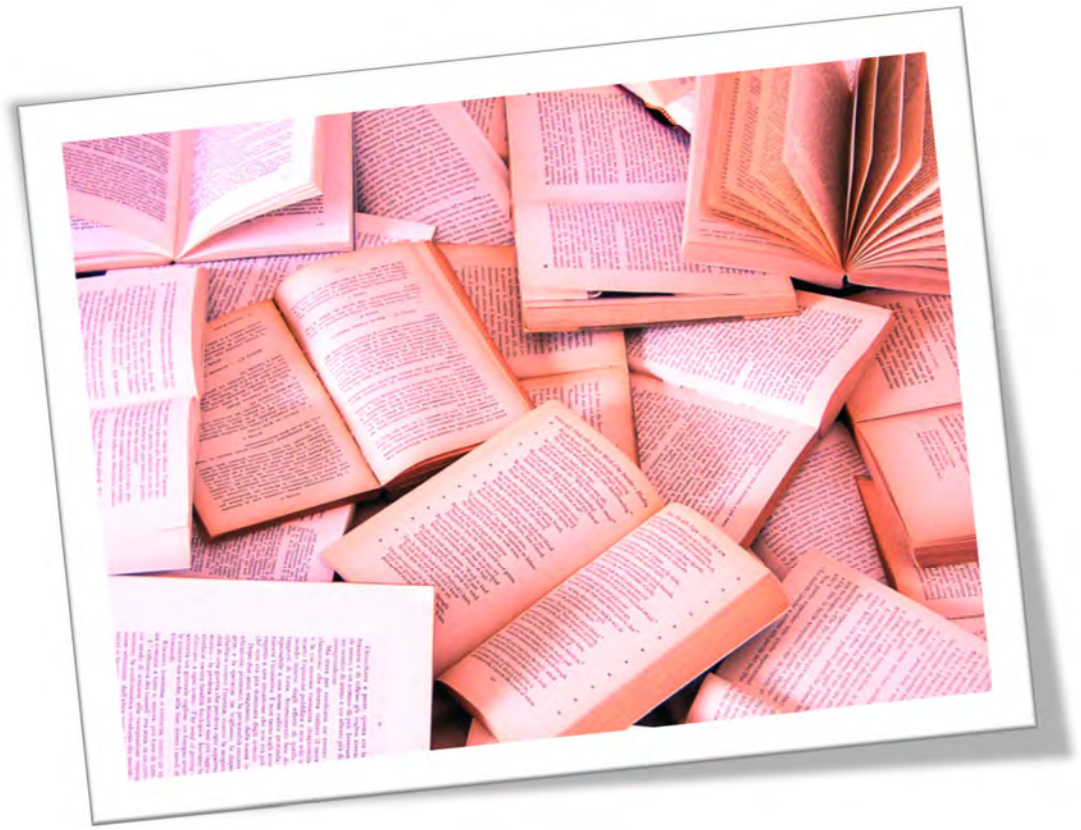
فصل پنجم



۱ جستجوی آگاهانه(مُکاشفه‌ای)[۲]

۱ - بیش‌ترِ برنامه‌های ویروس‌یاب(anti-virus scanners)، از ابتکار(مکاشفه) برای پیداکردن ویژگی‌ها و مشخّصاتی معیّن برای کشف ویروس‌های کامپیوتری استفاده می‌کنند.
(http://en.wikipedia.org/wiki/HEURISTIC_SEARCH)

۲ - Informed(Heuristic) search



فهرست برخی از عنوان‌های نوشته‌ها

آگاهانه کردن جستجو

چرا از دانش در یک مسأله استفاده می‌نماییم؟

چنانچه در فصل گذشته دیدیم، روش‌های جستجوی ناآگاهانه فقط از اطّلاعات قابل دسترس از تعریف مسأله استفاده می‌کردند و می‌توانستند راه حل‌ها را پیدا کنند، امّا ناکارآمد بودند؛ در آنها گره‌ها به صورت نمایی به وجود می‌آمدند. با استفاده از آگاهی‌هایی که در مورد ساختار مسأله داریم، می‌توانیم کارآیی را بهتر کنیم. البتّه باید توجّه کنیم که این دانش را باید از جایی دیگر بگیریم و این دانش باید درست باشد.

ایده‌ی اصلی این است که برای به وجود آوردن گره‌ها از یک تابع ارزیابی[1] استفاده نماییم.

تابع ارزیابی

مطلب مهم:

☑ **کنکور سراسری فنّـآوری اطّلاعات سال ۸۴:** یک تابع ارزیابی f(n)، شایستگی یک وضعیّت را تخمین می‌زند؛

یک وضعیّت با هر گره‌ی درخت جستجو پیوند برقرار می‌کند و بنابراین، تابع می‌تواند برای گره‌های درخت جستجو استفاده شود. در میان گره‌های کاندید، گره‌ای که دارای کم‌ترین مقدار ارزیابی است، انتخاب می‌شود؛ توجّه کنید که ما به دنبال ارزان‌ترین راه حل هستیم.

۱- evaluation function

در پیاده‌سازی این روش، از fringe، که یک صف مرتّب شده به وسیله‌ی تابع ارزیابی است، استفاده می‌نماییم. جستجوی با هزینه‌ی یکسان را به یاد بیاورید؛ در این روش، گره‌ها براساس مجموع هزینه‌ی مسیر توسعه داده می‌شوند و از یک صف اولویّت برای پیاده‌سازی جستجوی با هزینه‌ی یکسان استفاده می‌شود؛ در اینجا *هزینه‌ی مسیر مثالی از یک تابع ارزیابی می‌باشد.*

روش جستجوی اوّل- بهترین[۱]

مطلب مهم:

روش کلّی جستجوی آگاهانه(مکاشفه‌ای) می‌باشد و به این صورت است که اوّل، بهترین گره را برای رسیدن به هدف توسعه دهید؛ *[مشکل در اینجا این است که] در ادامه نمی‌دانیم که این گره هنوز بهترین است یا نه؟*؛ اگر ما این مطلب را بدانیم، آنگاه نیازی به عمل جستجو نداریم! (در صورتی که شما راه خود را بدانید، بر روی یک طرح کار نخواهید کرد)؛ در ادامه گره‌ای را که به نظر می‌رسد بهترین باشد را توسعه دهید. پس ایده‌ی جستجوی اوّل– بهترین، استفاده از یک تابع ارزیابی می‌باشد.

جستجوی حریصانه‌ی اوّل –-بهترین[۲]

مطلب مهم:

نوع خاصّی از جستجوی اوّل- بهترین است؛ در این روش، در ارزیابی فقط از مکاشفه(ابتکار)[۳] استفاده می‌شود؛ یعنی، $f(n)=h(n)$ و الگوریتم گرهی را که به نظر می‌رسد به هدف نزدیک‌تر است را توسعه می‌دهد.

تابع ارزیابی $h(n)$(ابتکاری) برابر است با: تخمین ارزش، از n تا نزدیک‌ترین هدف و [*در مورد مسأله‌ی کشور رومانی، تابع ارزیابی* $h(n)$*]* ($h_{SLD}(n)$)، *برابر است با، فاصله‌ی خطّی مستقیم، از شهر n تا شهر بخارست:*

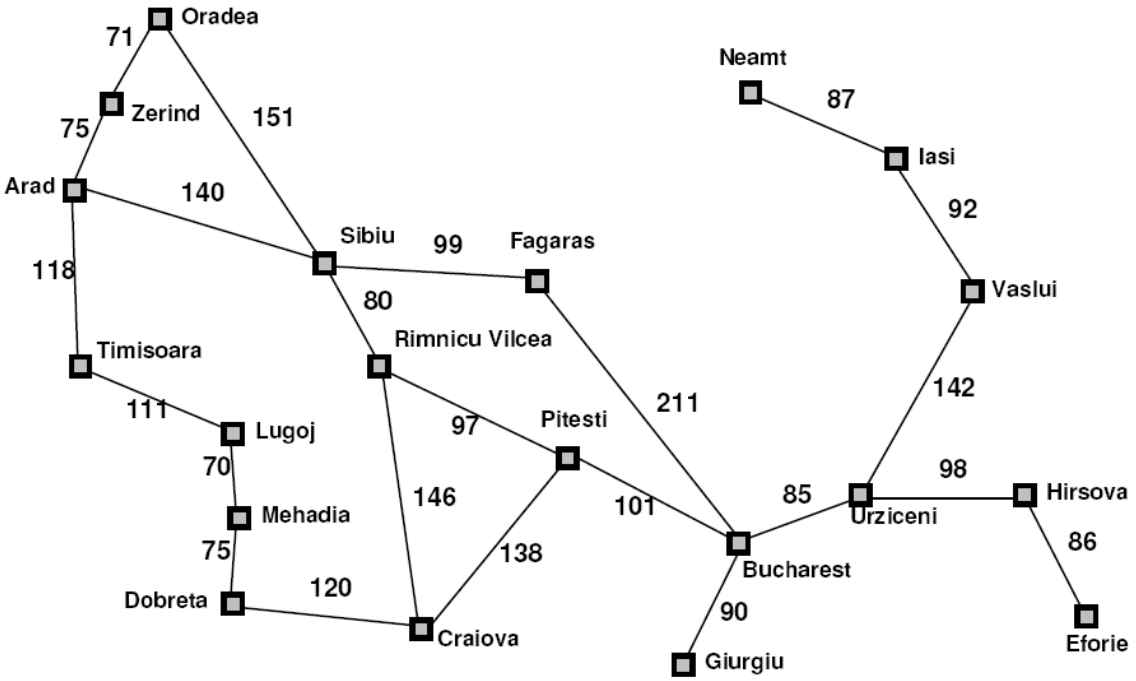
۱- best-first search

۲- Greedy best-first search

۳- Heuristic

مثال- مسأله‌ی کشور رومانی با ارزیابی مراحل به صورت مایل

مکاشفه(ابتکار):



شکل بالا- در شکل بالا توجّه کنید که عددها، فاصله‌ی خطّی مستقیم شهرها با هم هستند؛ به عنوان مثال، فاصله‌ی خطّی مستقیم شهر آراد تا سیبیو طبق شکل بالا برابر با ۱۴۰ مایل است.

آراد	۳۶۶	فَگراز	۱۷۸	مِهادیا	۲۴۱	سیبیو	۲۵۳
بخارست	۰	جیورجیو	۷۷	نِمت	۲۳۴	تیمیسوارا	۳۲۹
کِرایُوا	۱۶۰	هِرسُوا	۱۵۱	اُرادیا	۳۸۰	اِرزیچِنی	۸۰
دابرِیدا	۲۴۲	یاسی	۲۲۶	پیتِستی	۹۸	واسلُوی	۱۹۹
اِفُری	۱۶۱	لوگُوج	۲۴۴	ریمنیکو ویلسیا	۱۹۳	زِرِیند	۳۷۴

جدول بالا- *جدول فاصله‌ی خطّی مستقیم هر شهر تا شهر بخارست(به عنوان مکاشفه یا ابتکار)*

مطلب مهم:

توجّه کنید که *اکتشاف یا ابتکار که به صورت جدولی در* **بالا** *داده شده است و براساس فاصله‌ی خطّی مستقیم هر شهر تا شهر بخارست است، جزئی از تعریف اوّلیّه‌ی مسأله نمی‌باشد و در این مورد به صورت یک جدول خارجی داده شده است.*

حل:

گام نخست-

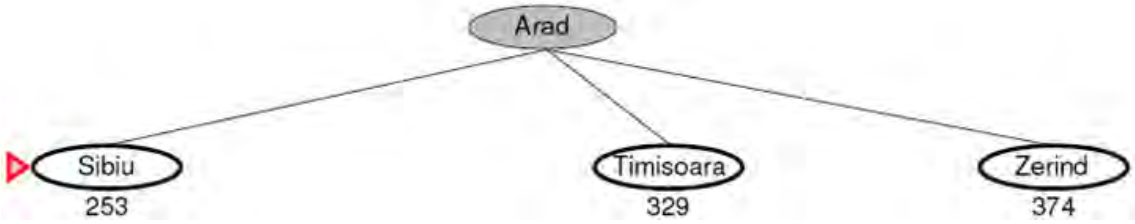
توضیح:

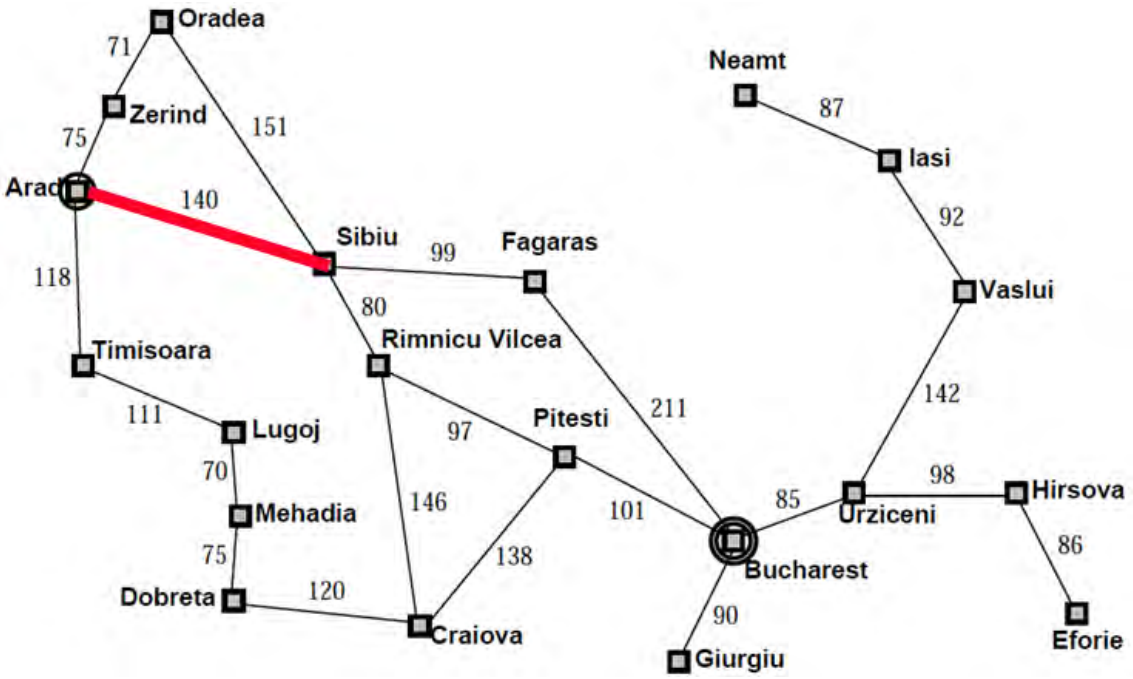
با توجّه به جدول **بالا** فاصله‌ی خطّی مستقیم از شهر آراد به شهر بخارست **۳۶۶** مایل می‌باشد.

گام بعد-

توضیح:

با توجّه به اینکه از بین شهرهای کاندید برای حرکت بعدی؛ یعنی، زرینند، تیمیسوارا و سیبیو، کم فاصله‌ترین شهر تا شهر بخارست، شهر سیبیو، با فاصله‌ی خطّی مستقیم **۲۵۳** مایل می‌باشد، شهر سیبیو را انتخاب می‌نماییم.

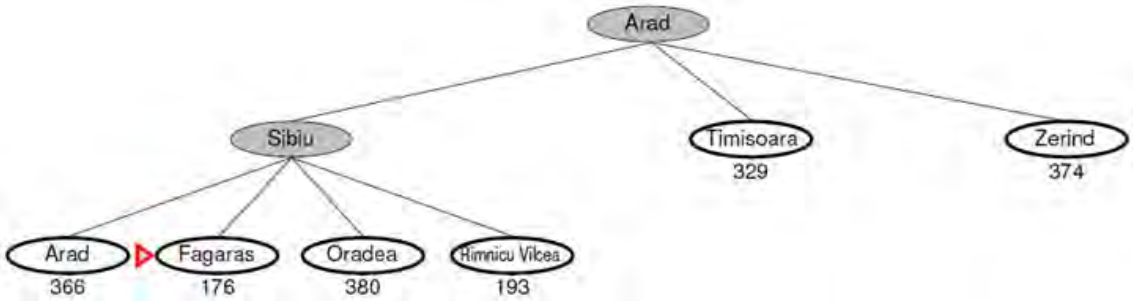


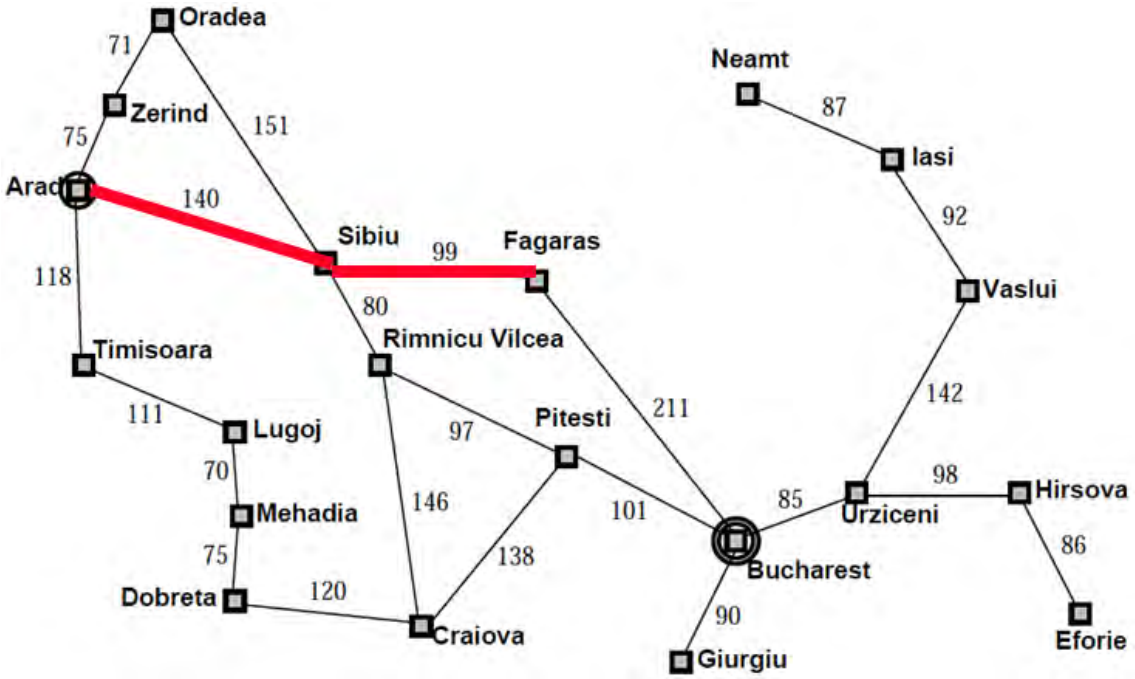


گام بعد–

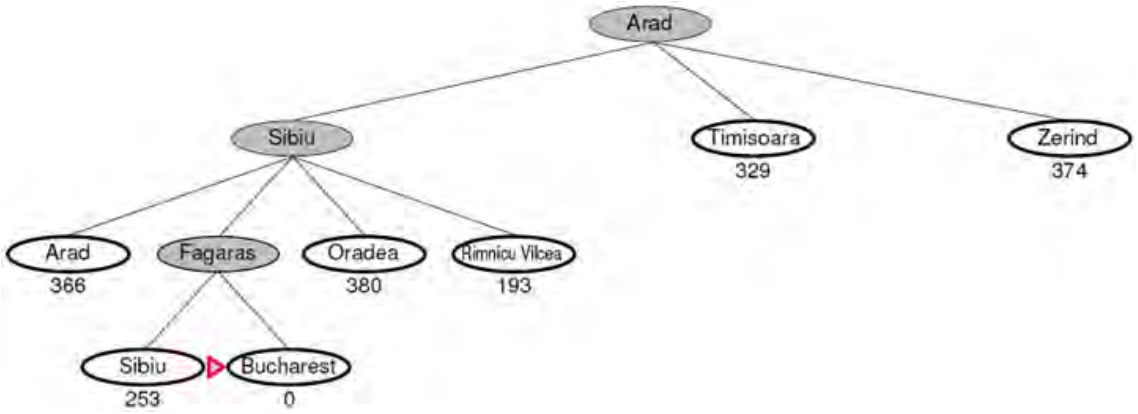
توضیح:

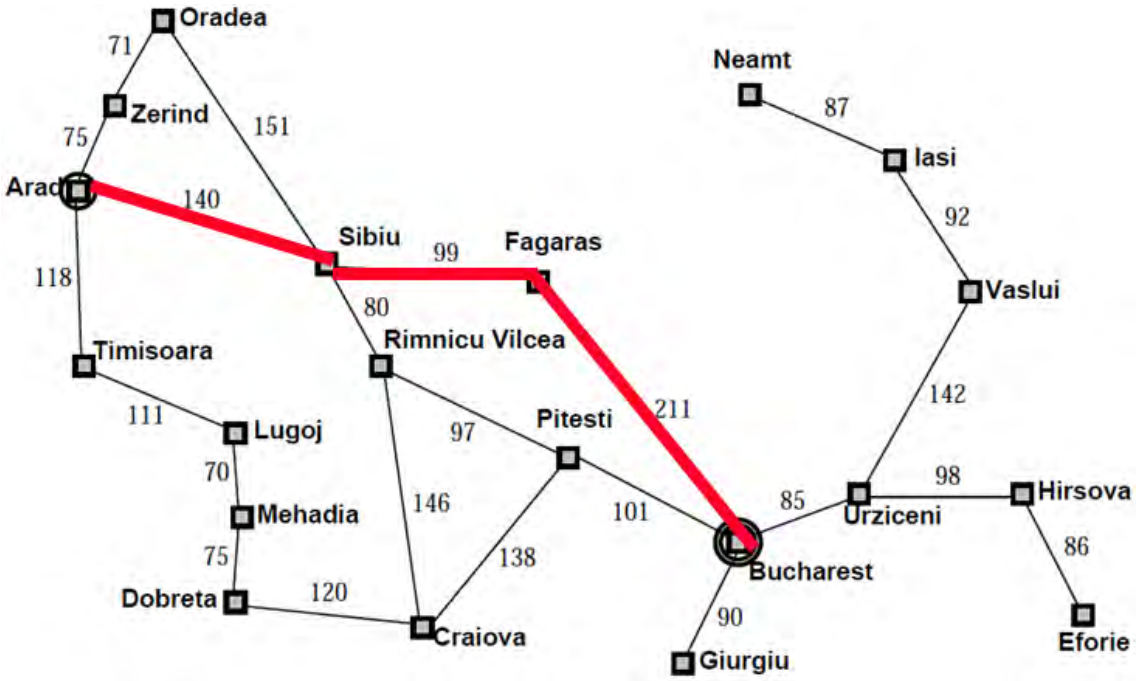
با توجّه به اینکه از بین شهرهای کاندید برای حرکت بعدی؛ یعنی، آراد، فَگراز، اُرادیا و ریمنیکو ویلک، کم فاصله‌ترین شهر تا شهر بخارست، شهر فَگراز، با فاصله‌ی خطّی مستقیم ۱۷۶ مایل می‌باشد، شهر فَگراز را انتخاب می‌نماییم.




Oradea
71
Neamt
87
Zerind
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
Pitesti
211
Lugoj
97
98
70
Hirsova
Mehadia
146
101
85
Urziceni
75
86
138
Bucharest
Dobreta
120
90
Craiova
Giurgiu
Eforie

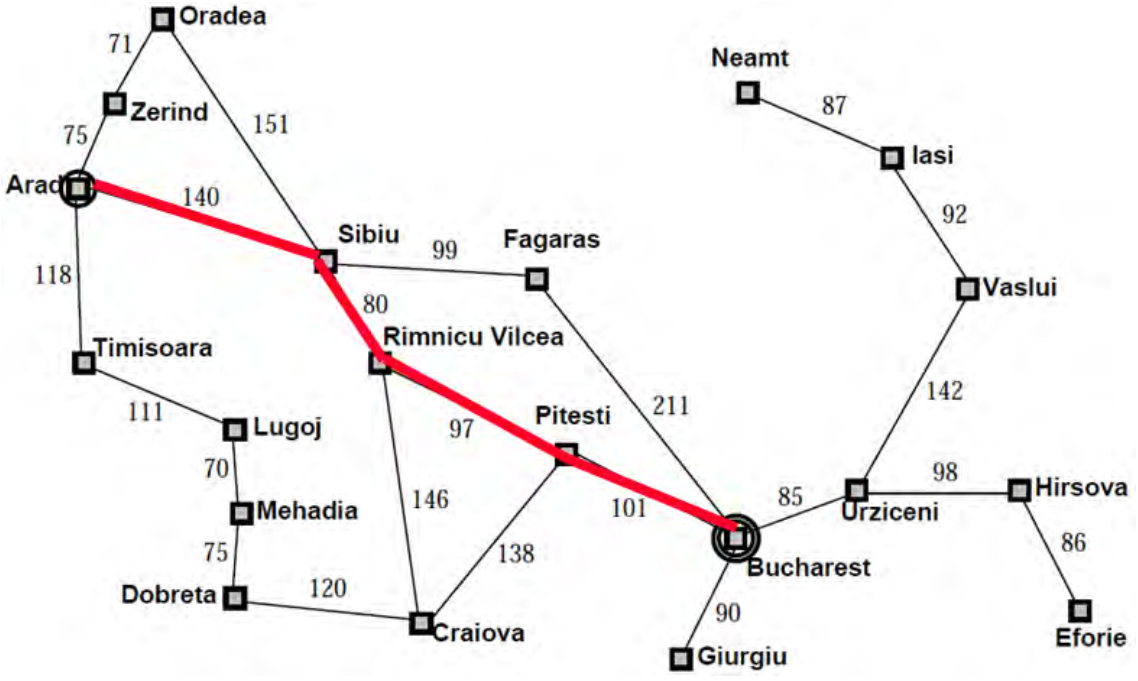
گام بعد:


Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
Oradea
380
Rimnicu Vilcea
193
Sibiu
253
Bucharest
0



توجّه:

امّا مسیر بهینه به صورت زیر است:

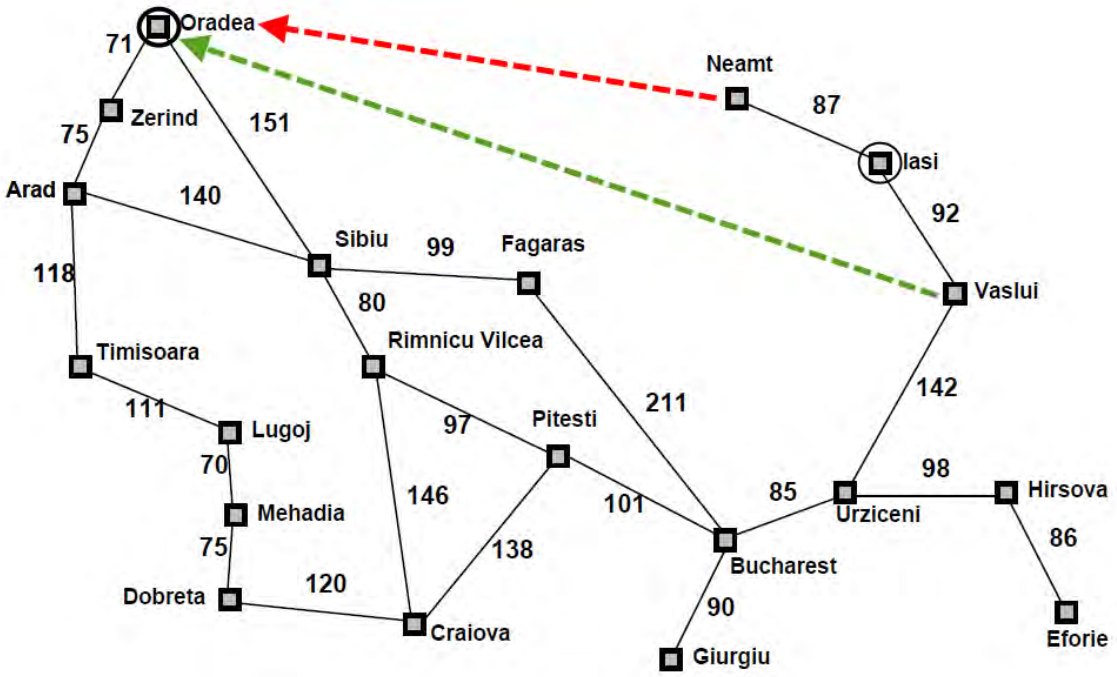


مطلب مهم:

پس این روش(روش حریصانه) بهینه نمی‌باشد.

ویژگی‌های جستجوی حریصانه

کامـل بـودن؟؟ نـه؛ ممکـن اسـت در حلقـه‌ها گیـر کنـد، مـثلاً اگـر مبـدأ، شـهر یاسـی(Iasi) باشـد و مقصـد، شـهر اُرادیا(Oradea) باشد، داریم:



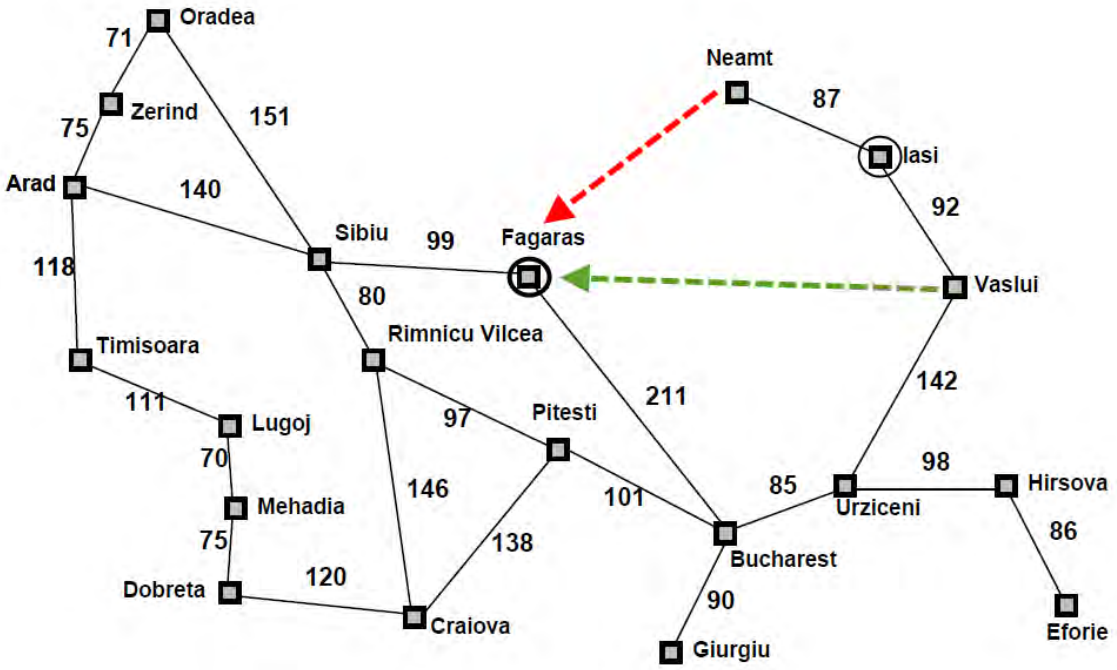
شکل بالا– *در شکل بالا توجّه کنید که مسیرهایی که با فلَش(<- -) نشان داده شده‌اند، واقعاً وجود ندارند و قابل استفاده نیستند!*

Iasi→ Neamt→ Iasi→ Neamt→

توضیح:

با توجّه به شکل بالا، در شهر Iasi، به دلیل اینکه فاصله‌ی خطّی مستقیم شهر Neamt تا شهر Oradea کم‌تر از فاصله‌ی خطی مستقیم شهر Vaslui تا شهر Oradea است، شهر Neamt را انتخاب می‌نماییم(Iasi→ Neamt)؛ در شهر Neamt، دوباره باید به شهر Iasi برگردیم(Iasi→ Neamt→ Iasi)؛ در شهر Iasi، به دلیل اینکه فاصله‌ی خطّی مستقیم شهر Neamt تا شهر Oradea، کم‌تر از فاصله‌ی خطی مستقیم شهر Vaslui تا شهر Oradea است، شهر Neamt را انتخاب می‌نماییم(Iasi→ Neamt→ Iasi→ Neamt)؛ در شهر Neamt ....

به عنوان مثالی دیگر، اگر بخواهیم از شهر یاسی به شهر فَگراز برویم؛ میان شهر یاسی و شهر نِمت، حلقه به وجود می‌آید.



شکل بالا– در شکل بالا توجّه کنید که مسیرهایی که با فلش(<– –) نشان داده شده‌اند، واقعاً وجود ندارند و قابل استفاده نیستند!

برای رفع این مشکل(یعنی، به وجود آمدن حلقه) باید از ملاقات مجدّد وضعیّت‌هایی که در گذشته ملاقات شده‌اند، خودداری نماییم.

این روش، در فضاهای جستجوی محدود و با بررّسی وضعّیت تکرار شده، کامل است.

زمان؟؟: $O(b^m)$، امّا با یک ابتکار صحیح می‌توان بهبود قابل توجّهی به دست آورد؛ b، فاکتور انشعاب و m، عمق درخت جستجو می‌باشد.

فضا؟؟: $O(b^m)$، همه‌ی گره‌ها را در حافظه نگه می‌دارد.

بهینگی؟؟ نه.

جستجوی A^*

مطلب مهم:

بهترین شکل شناخته‌شده‌ی جستجوی اوّل– بهترین است؛ مسأله‌ای که در مورد روش جستجوی حریصانه وجود دارد، این است که هزینه‌ای که مسیر تاکنون داشته است، مورد توجّه قرار نگرفته است. روش جستجوی A^*، از به وجود آوردن(توسعه دادن) مسیرهایی که پرهزینه هستند، اجتناب می‌کند.

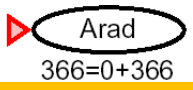
در این روش، تابع ارزیابی، $f(n)=g(n)+h(n)$ می‌باشد؛ که $g(n)$ = هزینه‌ای که تاکنون برای رسیدن به n صرف شده است؛ $h(n)$ = هزینه‌ی تخمین زده شده برای n؛ و $f(n)$ = کلّ هزینه‌ی تخمین زده شده‌ی مسیر، از n تا هدف می‌باشد. روش جستجوی A^* از یک روش ابتکاری مُجاز، برای جستجو استفاده می کند. *در واقع جستجوی A^*، ترکیب جستجوی با هزینه‌ی یکسان و جستجوی حریصانه می‌باشد.*

مثالی برای جستجوی A^*(مسأله‌ی کشور رومانی)

۲۵۳	سیبیو	۲۴۱	مِهادیا	۱۷۸	فگراز	۳۶۶	آراد
۳۲۹	تیمیسوارا	۲۳۴	نِمت	۷۷	جیورجیو	۰	بخارست
۸۰	اِرزیچِنی	۳۸۰	اُرادیا	۱۵۱	هِرسُوا	۱۶۰	کرایوا
۱۹۹	واسلُوی	۹۸	پیتِستی	۲۲۶	یاسی	۲۴۲	دابرِیدا
۳۷۴	زِریند	۱۹۳	ریمنیکو ویلسیا	۲۴۴	لوگُوج	۱۶۱	اِفُری

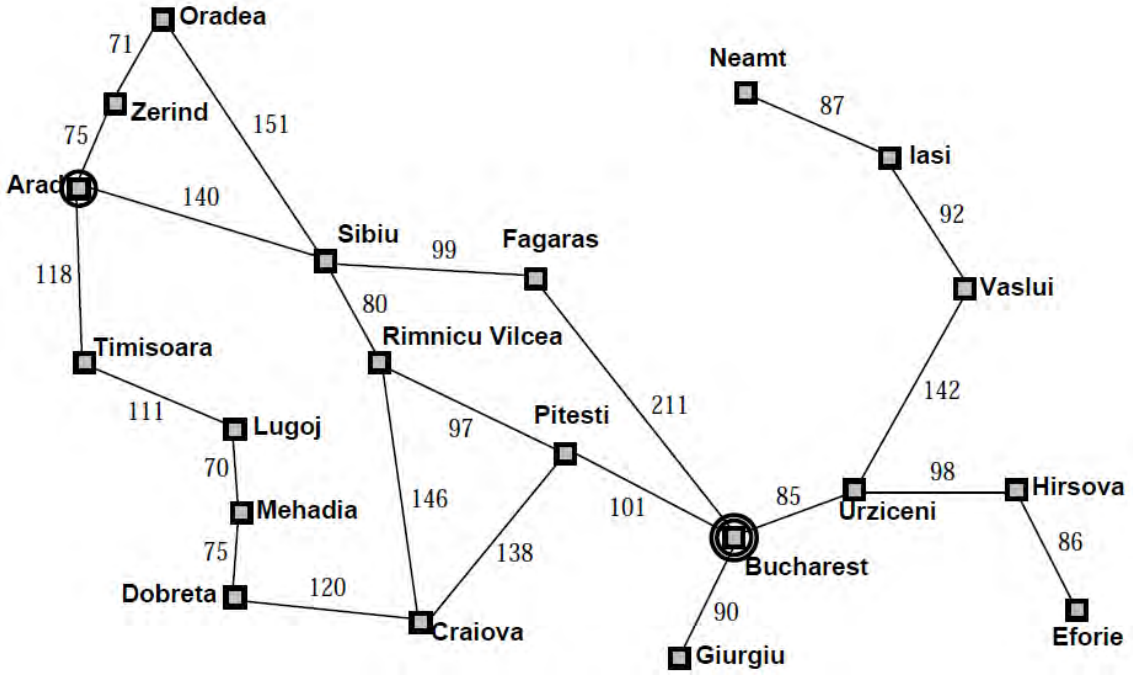
جدول بالا- جدول فاصله‌ی خطّی مستقیم هر شهر تا شهر بخارست

گام نخست-

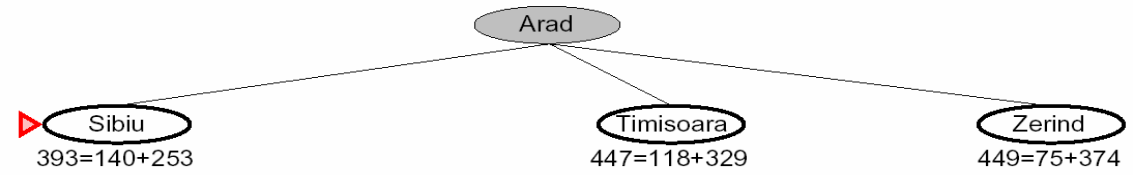


توضیح:

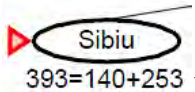
۳۶۶ سمت راست، در تصویر بالا، فاصله‌ی خطّی مستقیم شهر آراد تا شهر بخارست است، که با توجّه به جدول بالا، برابر با ۳۶۶ است؛ *صفر هم فاصله‌ی خطّی‌ای است که تا کنون برای رسیدن به شهر بخارست طی شده است؛* مجموع این دو عدد(۳۶۶+۰)، برابر با ۳۶۶ سمت چپ، در تصویر بالا می‌باشد.



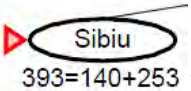
گام بعد:



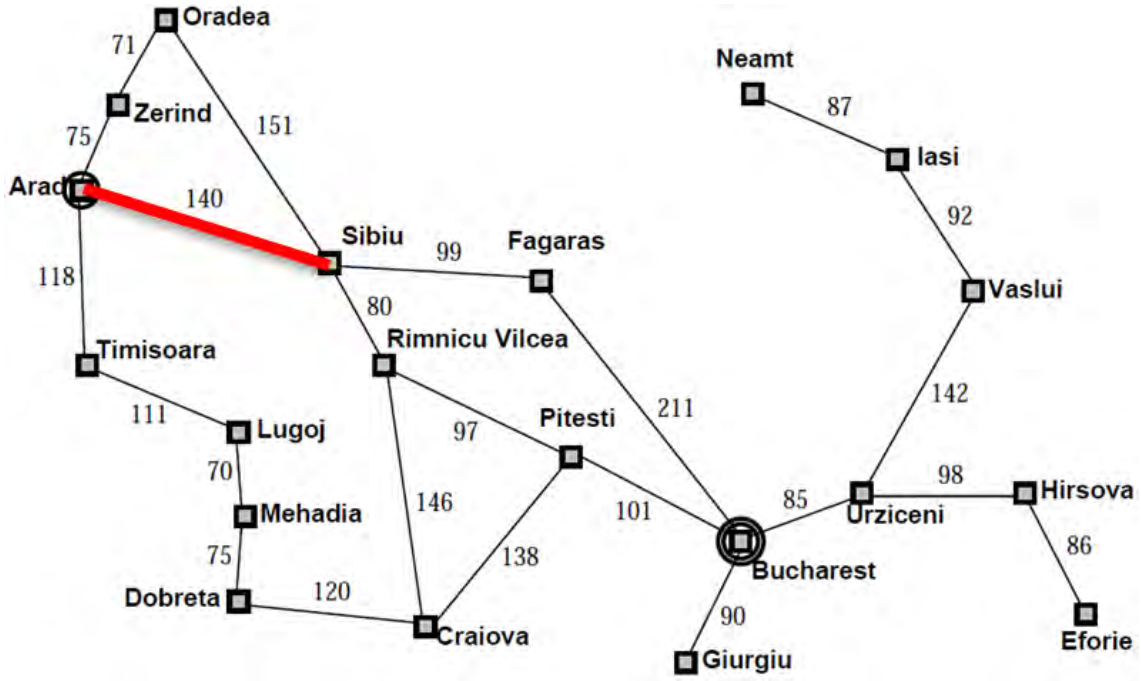
توضیح:



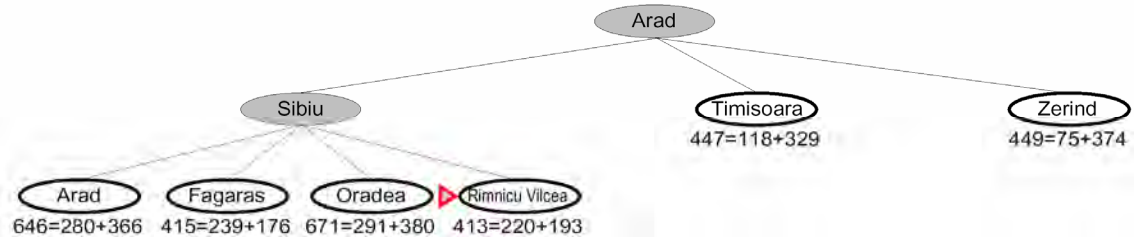
۲۵۳ سمت راست، در سمت چپ تصویر بالا( ) ، فاصله‌ی خطّی مستقیم شهر سیبیو تا شهر بخارست است، که با توجّه به جدول قبل، برابر با ۲۵۳ است؛ ۱۴۰ هم فاصله‌ی خطّی‌ای است که تا کنون برای رسیدن به شهر بخارست طی شده است، که با توجّه به شکل زیر برابر با ۱۴۰ می‌باشد؛ مجموع این دو عدد(۲۵۳+۱۴۰)، برابر با ۳۹۳ سمت چپ، در تصویر



بالا( ) می‌باشد. بقیّه‌ی اعداد هم که برای شهرهای تیمیسوارا و زرینـد، در درخت جستجوی شکل بالا به دست آمده‌اند، به طریقی مشابه محاسبه شده‌اند.

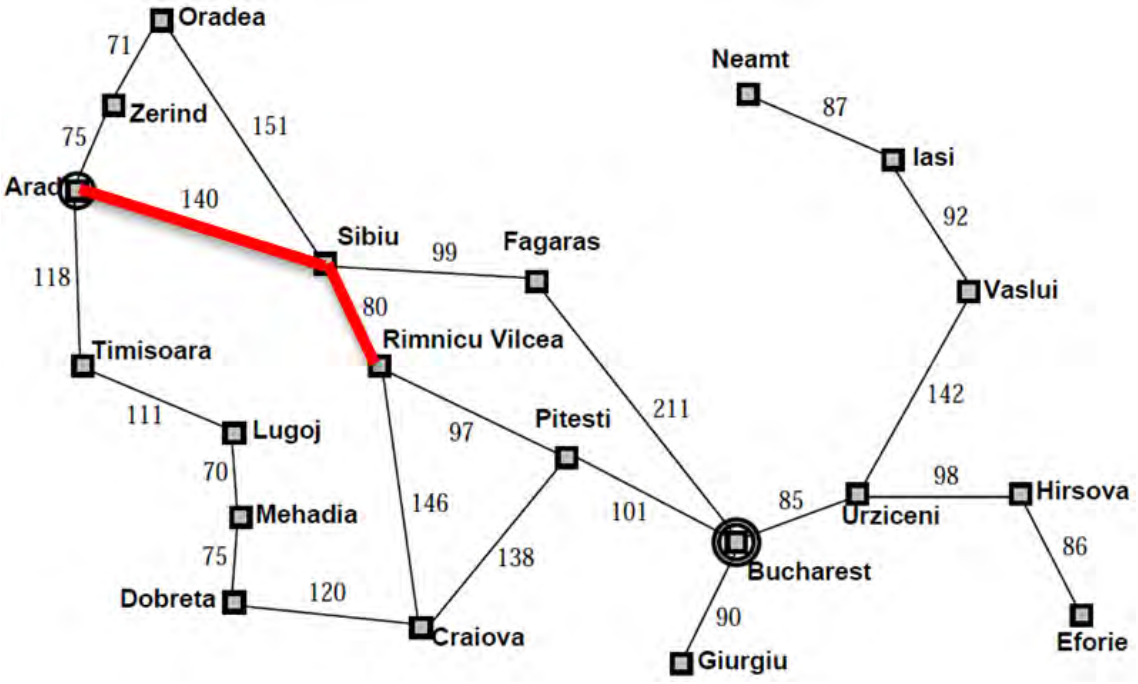


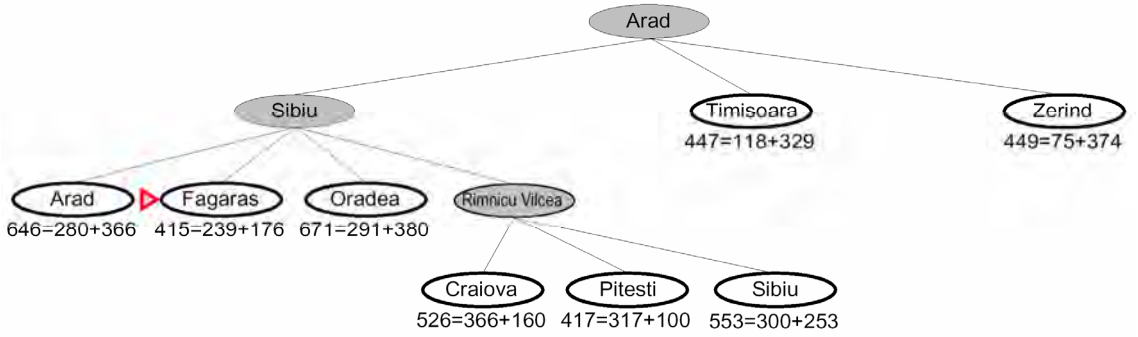
گام بعد:



توضیح:

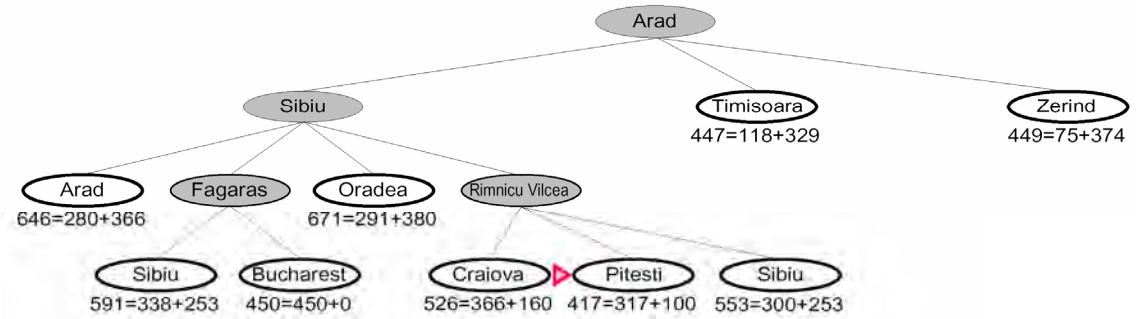
در تصویر بالا، ۱۹۳ سمت راست، در شهری که به عنوان بعدی انتخاب شده است(Rimnicu Vilcea 413=220+193)، فاصله‌ی خطّی مستقیم شهر ریمنیکو ویلسیا تا شهر بخارست است، که با توجّه به جدول قبل، برابر با ۱۹۳ است؛ ۲۲۰ هم فاصله‌ی خطّی‌ای است که تا کنون برای رسیدن به شهر بخارست طی شده است، که با توجّه به شکل زیر برابر با ۱۴۰+۸۰ =۲۲۰ می‌باشد؛ مجموع این دو عدد(۱۹۳+۲۲۰)، برابر با ۴۱۳ سمت چپ، در تصویر بالا(Rimnicu Vilcea 413=220+193) می‌باشد. بقیّه‌ی اعداد هم که در درخت جستجوی شکل بالا، برای سایر شهرها به دست آمده‌اند، به طریقی مشابه محاسبه شده‌اند.

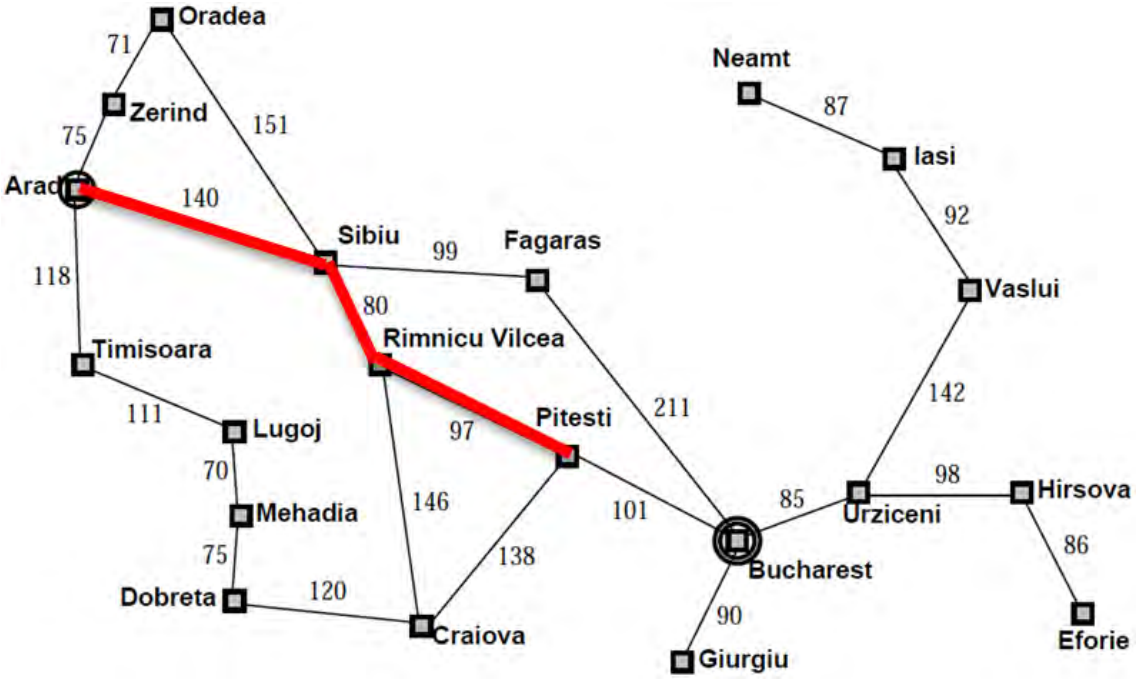




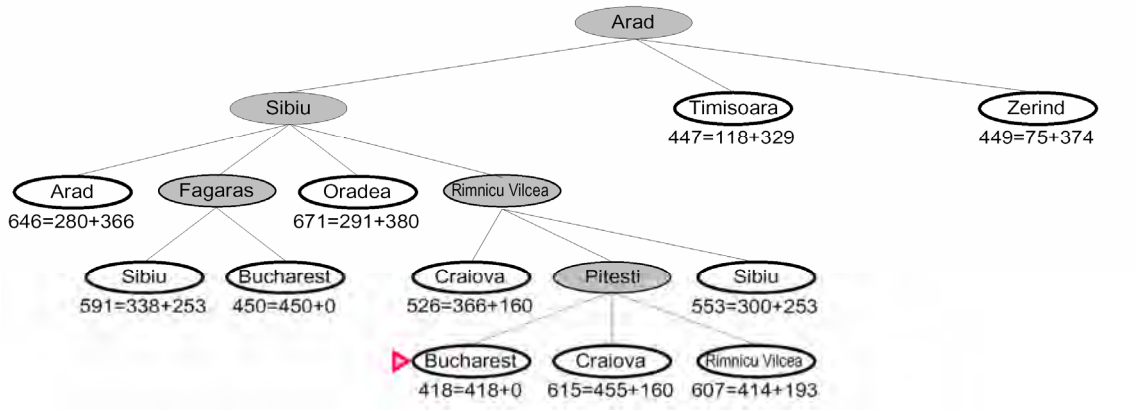
همان طور که در شکل بالا می‌بینید، به دلیل اینکه هزینه‌ی فگراز بیش‌تر از ریمنیکوویلک است، ریمنیکوویلک انتخاب می‌شود.

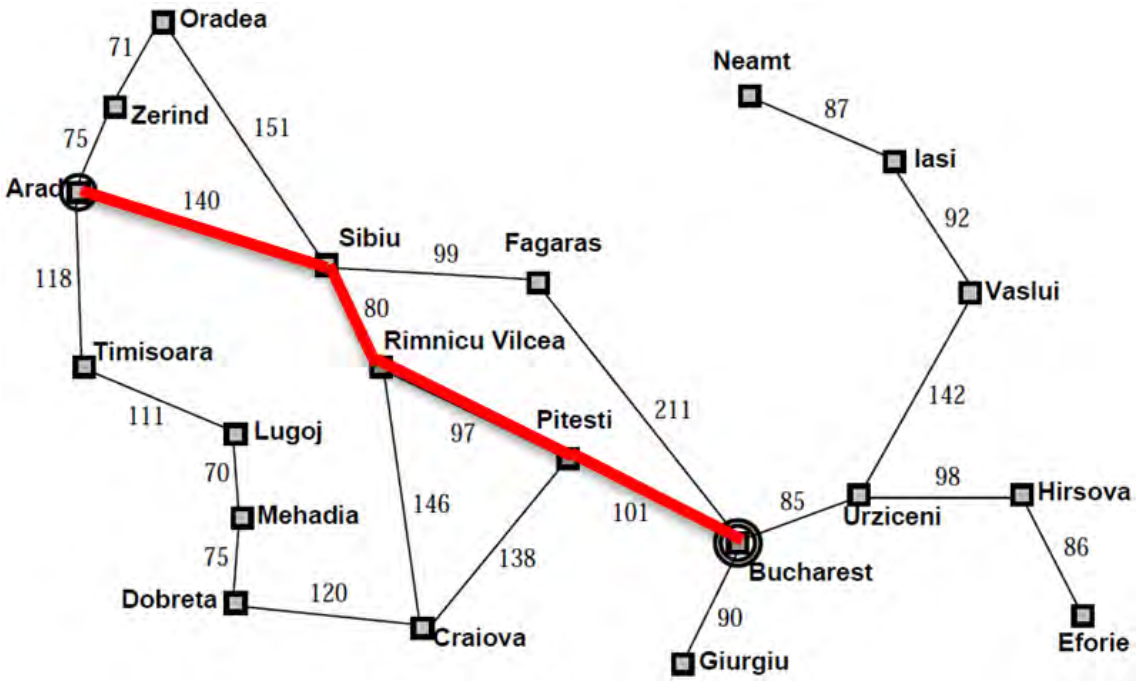
گام بعد:


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253


Oradea
71
Neamt
87
Zerind
151
75
Iasi
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
Pitesti
211
Lugoj
97
98
70
85
Hirsova
Mehadia
146
101
Urziceni
75
86
138
Bucharest
Dobreta
120
90
Craiova
Eforie
Giurgiu

گام بعد(آخر):


Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193


Oradea
71
Neamt
87
Zerind
151
Iasi
75
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
Pitesti
211
Lugoj
97
98
70
Hirsova
Mehadia
146
85
Urziceni
101
75
86
Bucharest
138
Dobreta
120
90
Craiova
Eforie
Giurgiu

مطلب مهم:

بهینگی جستجوی A^*

☑مطلب کنکور سراسری مهندسی کامپیوتر سال ۸۱ و کنکور سراسری فنّـآوری اطّلاعات سال ۸۹-

برای تضمین بهینگی مکاشفه، نباید هزینه‌ی رسیدن به هدف($h(n)$) را زیاد برآورد نماییم[۱]؛ در مکاشفه‌ی قابل قبول[۲] داریم: $h(n) \le h^*(n)$؛ که $h^*(n)$، هزینه‌ی واقعی رسیدن به هدف می‌باشد.

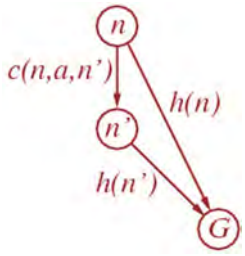
[۳] ***قضیّه:*** *اگر $h(n)$، قابل قبول(admissible) باشد، جستجوی A^*ی که از جستجوی درختی استفاده می‌کند، جستجویی بهینه است.*

مکاشفه‌های سازگار[۱](یکنوا، یکنواخت[۲])

یک مکاشفه سازگار است اگر برای هر جانشین n'، از یک گره n، که به وسیله‌ی هر عملکرد a تولید می‌شود، داشته باشیم:

$$h(n) \le c(n,a,n') + h(n')$$

c، هزینه‌ی گام رسیدن به n' از n، با عمل a است.



مکاشفه‌های قابل قبول معمولاً سازگار هستند.

قضیّه: **☑مطلب کنکور سراسری فنّـآوری اطّلاعات سال ۸۸-** *اگر $h(n)$ سازگار باشد، A^*ی که از جستجوی گرافی استفاده می‌کند، بهینه است.*

---

۱ - overestimate

۲ - admissible

۳ - اوّلین قضیّه‌ای که بیش‌تر ما برای اوّلین بار و در درس ریّاضیّات دوران راهنمائی تحصیلی با آن آشنا شدیم، قضیّه‌ی فیثاغورث(Pythagorean theorem) بود، که شکل زیر یادآور آن می‌باشد:

ویژگی‌های A^*

کامل بودن؟؟ بله.

زمان؟؟ به صورت نمایی می‌باشد؛ پیچیدگی زمانی در این روش به کیفیّت تابع مکاشفه‌ای وابسته است.

مطلب مهم:

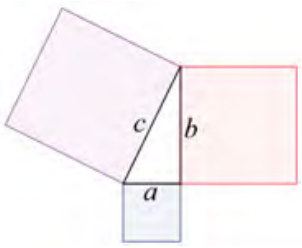
فضا؟؟ تمام گره‌ها را در حافظه نگه می‌دارد و به صورت نمایی می‌باشد. بنابراین، مشکل اصلی این روش، فضا است، نه زمان.

بهینه؟؟ بله، البتّه با در نظر گرفتن دو قضیّه‌ی اخیر.

اگر C^*، هزینه‌ی مسیر راه حلّ بهینه باشد؛

- A^* تمام گره‌های با $f(n) < C^*$ را توسعه می‌دهد.
- A^* بعضی از گره‌هایی که $f(n) = C^*$ است را توسعه می‌دهد.
- A^* گره‌هایی که $f(n) > C^*$ است را توسعه نمی‌دهد.

روش A^* به طور مؤثّری برای هر تابع ارزیابی بهینه می‌باشد. یک الگوریتم ممکن است که راه حلّ بهینه را در صورتی که همه‌ی گره‌های با $f(n) < C^*$ را توسعه ندهد، گم کند.



شکل بالا– $c^2 = a^2 + b^2$

در این کتاب به افتخار فیثاغورث(Pythagoras)، تقریباً ۴۹۵ – ۵۷۰ قبل از میلاد حضرت مسیح (درود بر او باد)، فیلسوف و ریّاضیدان یونانی، برای «قضیّه»ها از مجسّمه‌ی وی به عنوان نماد استفاده شده است:

(http://en.wikipedia.org/wiki/Pythagoras و
http://en.wikipedia.org/wiki/Pythagorean_theorem)

۱- consistent

۲- monotonic

بهترکردن A^*

A^* یکی از الگوریتم‌های کلیدی در هوش مصنوعی می‌باشد، *امّا دارای اشکال در حافظه‌ی مورد نیاز می‌باشد*، چونکه تمام گره‌های ملاقات شده را در حافظه نگه می‌دارد و برای مسأله‌های با ساختار بزرگ، عملی نمی‌باشد؛ روش A^* عمیق شونده‌ی تکراری[۱]، روش جستجوی اوّل-بهترین بازگشتی[۲] و روش A^* با حافظه‌ی محدود شده‌ی(کراندار) ساده شده[۳]، جزء روش‌هایی هستند که برای رفع مشکل فضای روش A^* مورد استفاده قرار می‌گیرند.

روش A^* عمیق شونده‌ی تکراری

بهبود روش A^* می‌باشد و مانند ترکیبی از A^* و عمیق شونده‌ی تکراری است.

مطلب مهم:

در این روش به جای محدود کردن عمق، به صورت تدریجی، محدوده‌ی تابع ارزیابی(یا هزینه‌ی f، یا g+h) افزایش داده می‌شود و جستجوی اوّل عمق با هزینه‌ی محدود شده را برای تابع ارزیابی انجام می‌دهد.

مشکل این روش، مقدار افزایش تدریجی محدوده‌ی ارزیابی است؛ برای رفع این مشکل، یک راه این است که مقدار را در محدوده‌ی مقدار گام قبلی در نظر بگیریم و روش دیگر این است که محدوده را به وسیله‌ی یک مقدار ثابت، به نام ε افزایش دهیم.[۴]

ویژگی‌های روش A^* عمیق شونده‌ی تکراری

روش A^* عمیق شونده‌ی تکراری، کامل است و راه حل‌های بهینه را به ما می‌دهد؛ نسبت به روش A^*، به فضای به مراتب کم‌تری نیاز دارد؛ اگر تعداد گره‌ها به صورت نمایی به‌همراه(با) هزینه رشد نماید، زمان این روش همانند روش A^* خواهد بود.

---

۱ - *I*terative *D*eepening A^*(*IDA**)

۲ - *R*ecursive *b*est-*f*irst search(*RBFS*)

۳ - Simplified *M*emory-Bounded A^*(*SMA**)

۴ - برگرفته از مطالب میلُوسْ هاوْسْکِرِکْتْ(Milos Hauskrecht)، استاد دانشگاه ایالت پیتسبورگ کشور ایالات متّحده‌ی آمریکا

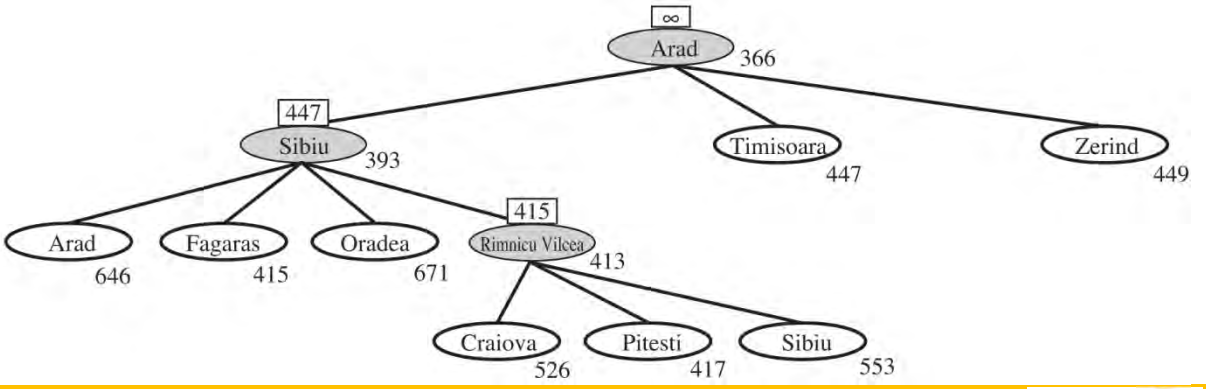
روش جستجوی اوّل–بهترین بازگشتی

مطلب مهم:

شبیه روش جستجوی IDA* است، امّا مقدار f بهترین مسیر جایگزین ممکن را برای هر جد گره جاری نگهداری می‌کند؛ اگر مقدارهای f جاری از این مقدار جایگزین بیش‌تر شد، به بهترین مسیر جایگزین برمی‌گردد. در موقع برگشت، مقدار f هر گره را به بهترین مقدار f فرزندانش تغییر می‌دهد. این روش، مقدار f بهترین برگ زیردرختی که از آن صرفنظر کرده‌ایم[1] را به یاد می‌آورد و بنابراین، در صورت نیاز می‌تواند آن را مجدّداً در آینده برّرسی کند.

مثال-

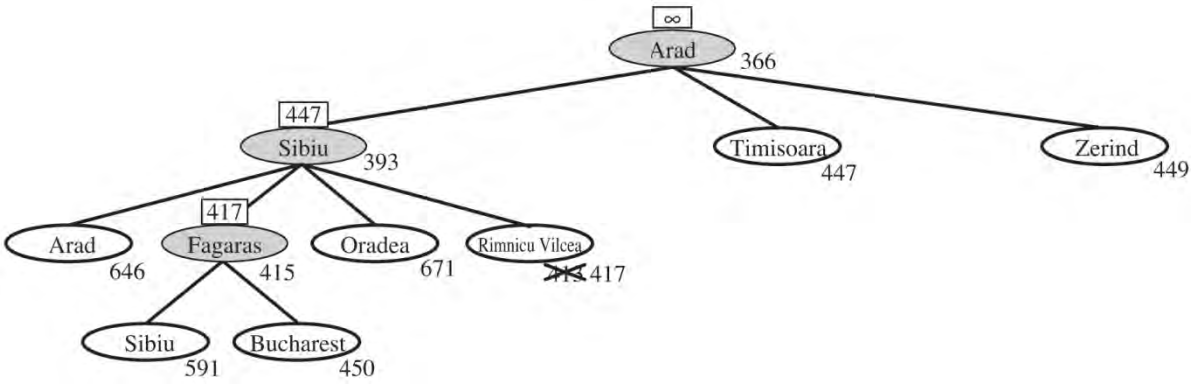
الف) بعد از توسعه‌ی Arad, Sibiu, Rimnicu Vilcea:



توضیح:

در شکل بالا، مسیر Rimnicu Vilcea قبلاً توسعه داده شده است؛ در بالای هر گره، «حدّ f»[2] برای هر فراخوانی بازگشتی نشان داده شده است؛ در پایین گره هم مقدار f(n) نشان داده شده است. مسیر، تا Pitesti(۴۱۷) که دارای مقدار f بدتر از حدّ f(۴۱۵) است، دنبال می‌شود.

ب) بعد از برگشت به Sibiu و توسعه‌ی Fagaras:



---

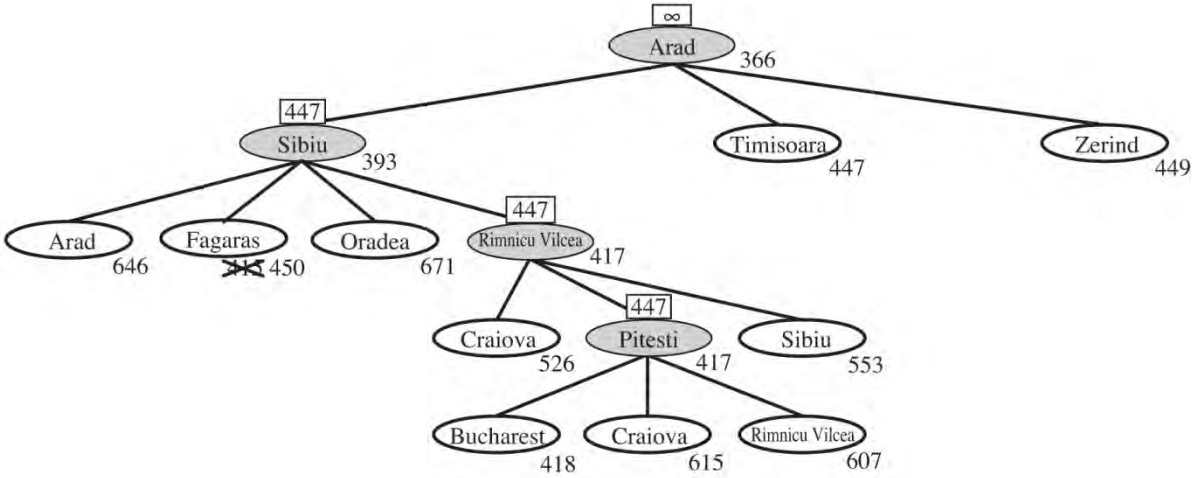
۱ – forgotten

۲ – f-limit

توضیح:

در شکل بالا برگشت صورت می‌گیرد و بهترین مقدار برگ زیر درخت صرفنظر شده (۴۱۷) به Rimnicu Vilcea نسبت داده می‌شود و سپس Fagaras توسعه داده می‌شود و بهترین برگ با مقدار ۴۵۰ به دست می‌آید.

ج) بعد از برگشت به Rimnicu Vilcea و توسعه دادن Pitesti:



توضیح:

در شکل بالا برگشت صورت می‌گیرد و مقدار بهترین برگ زیردرخت صرفنظر شده(۴۵۰) به Fagaras نسبت داده می‌شود و سپس Rimnicu Vilcea توسعه داده می‌شود. این بار با توجّه به اینکه بهترین مسیر جایگزین (به Timisoara) دارای هزینه‌ی دست کم ۴۴۷ است، توسعه به بخارست ادامه می‌یابد.

ویژگی‌های روش RBFS

این روش اندکی کاراتر از IDA^* است؛ گره‌های زیادی تولید می‌کند و تغییرات را ذخیره می‌کند. مانند A^*، اگر $h(n)$ قابل قبول باشد، بهینه است. پیچیدگی فضایی، $O(bd)$ است. تعیین پیچیدگی زمانی دشوار است.

روش A^* با حافظه‌ی محدود شده‌ی(کراندار) ساده شده

در این روش از یک صف دارای اندازه‌ی محدود برای محدود کردن حافظه‌ای که مورد استفاده قرار می‌گیرد، استفاده می‌شود.

اکتشافات(ابتکارات)

پیدا کردن یک تابع مکاشفه‌ای خوب، برای جستجو، کاری بسیار مهم است؛ در مسأله‌ی کشور رومانی، برای مکاشفه از فاصله‌ی مستقیم هر یک از شهرها تا شهر هدف(مقصد) استفاده کردیم، که این کار برای مسأله‌های مسیریابی، خوب بود، امّا مکاشفات را برای مسأله‌های جدید چگونه پیدا کنیم؟، جواب این است که راه استانداردی وجود ندارد، در بیش‌ترِ موردها باید مکاشفات به صورت دستی پیدا شوند.

مثال- دو ابتکار قابل قبول برای پازل ۸-تایی:

مطلب مهم:

☑ **مطلب مورد استفاده در کنکور سراسری مکاترونیک سال ۸۴-** $h_1(N)$ = تعداد کاشی‌های در جای نادرست گذاشته شده.

$h_2(N)$ = مجموع فاصله‌ی هر کاشی در جای نادرست گذاشته شده تا وضعیّت درست[۱].

1	2	3
4	5	6
7	8	

شکل بالا- *حالت نهائی*

7	2	4
5		6
8	3	1

شکل بالا- *حالت اوّلیه*

$h_1(N) = 6$

$h_2(N) = 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14$

مکاشفه‌هایی با استفاده از مسأله‌های راحت شده[۲]

تعریف- مسأله‌ای که نسبت به مسأله‌ی اصلی محدودیّت‌های کم‌تری روی عملکردهایش باشد، مسأله‌ی راحت شده نام دارد.

هزینه‌ی راه حلّ بهینه برای یک مسأله‌ی راحت شده، یک مکاشفه‌ی قابل قبول برای مسأله‌ی اصلی است؛ مثلاً همان طور که در مثال قبل با عنوان «دو ابتکار قابل قبول برای پازل ۸-تایی» دیدیم، اگر قانون‌های پازل ۸-تایی، راحت شوند و یک کاشی بتواند به هر خانه‌ای برود، آنگاه مکاشفه‌ی $h_1(N)$ را داریم، که قابل قبول هم می‌باشد و کوتاه‌ترین راه حل را به ما می‌دهد. یا اگر قانون‌های پازل ۸-تایی، راحت شوند و هر کاشی بتواند به هر خانه‌ی همسایه برود، آنگاه مکاشفه‌ی $h_2(N)$ را خواهیم داشت که قابل قبول هم می‌باشد و کوتاه‌ترین راه حل را به ما می‌دهد.[۳]

۱- Manhattan

۲- relaxed problems

۳- مطلب‌های درس «هوش مصنوعی» استادیار، «سْوِتْلانا لِیْزِبْنیکْ(Svetlana Lazebnik)»، دانشکده‌ی علوم کامپیوتر دانشگاه ایالت ایلینویْزْ کشور ایالات متّحده‌ی آمریکا، پاییز سال ۲۰۱۳ میلادی

چکیده‌ی مطلب‌های فصل پنجم

توابع مکاشفه‌ای هزینه‌های کوتاه‌ترین مسیرها را تخمین می‌زنند.

ابتکارات(مکاشفه‌های) خوب می‌توانند به طور چشمگیری هزینه‌ی جستجو را کاهش دهند.

روش جستجوی اوّل- بهترین، یک روش جستجوی کلّی است، که در آن، اوّل، کم هزینه‌ترین گره‌ها، با توجّه به برخی از معیارها، توسعه داده می‌شوند.

روش جستجوی حریصانه(حریصانه‌ی اوّل- بهترین)، کم هزینه‌ترین $h(n)$ تخمین زده شده برای رسیدن به مقصد را به عنوان معیار در نظر می‌گیرد؛ این روش، زمان جستجو را کاهش می‌دهد، امّا کامل نیست و همیشه هم بهینه نیست.

روش جستجوی A^*، روش‌های جستجوی با هزینه‌ی یکسان و حریصانه را با هم ترکیب می‌کند: $f(n)= g(n)+h(n)$.

اگر $h(n)$ قابل قبول(admissible) باشد، جستجوی A^*ی که از جستجوی درختی استفاده می‌کند، جستجویی بهینه است.

اگر $h(n)$ سازگار باشد، A^*ی که از جستجوی گرافی استفاده می‌کند، بهینه است.

روش‌های A^* عمیق شونده‌ی تکراری، اوّل بهترین بازگشتی و A^* با حافظه‌ی محدود شده‌ی(کراندار) ساده شده، حافظه‌ی مورد نیاز روش A^* را کاهش می‌دهند.

یادآوری یا تکمیل مطلب‌های فصل پنجم

تعریف– تابع مکاشفه‌ای را تعریف کنید.

جواب– هزینه‌ی ارزان‌ترین مسیر از وضعیّت جاری به وضعیّت هدف را تخمین می‌زند.[۱]

تست– یک جستجوی حریصانه از یک تابع مکاشفه‌ای برای .....، استفاده می‌کند.

۱) توسعه‌ی گره‌ای که به نظر می‌رسد نزدیک‌ترین به هدف است

۲) توسعه‌ی گره‌ای که به هدف نزدیک‌ترین است

۳) توسعه‌ی گره‌ای که گران‌ترین است

۴) توسعه‌ی سمت چپ‌ترین گره

جواب– گزینه‌ی «۱» درست است.[۲]

درست یا غلط– جستجوی اوّل عمق، حالت خاصّی از جستجوی درختی اوّل بهترین است.

جواب– «درست» است؛ جستجوی اوّل عمق، جستجوی اوّل بهترین با $f(n) = -depth(n)$ است.[۳]

---

۱ – آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۲ – آزمون میان‌ترم درس «هوش مصنوعی» استاد، «کتی مک‌کون»، دانشکده‌ی علوم کامپیوتر دانشگاه کلمبیای کشور آمریکا، ۲۶ اکتبر سال ۲۰۰۶ میلادی

۳ – تکلیف‌های خانگی شماره‌ی ۱ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا

درست یا غلط- جستجوی اوّل سطح، جستجوی اوّل بهترین با $f(n) = depth(n)$ است.

جواب- «درست» است.[۱]

درست یا غلط- جستجوی با هزینه‌ی یکنواخت، جستجوی اوّل بهترین با $f(n) = g(n)$ است.

جواب- «درست» است.[۲]

تعریف- جستجوی اوّل-بهترین حریصانه را به طور خلاصه تعریف کنید.

جواب- برای توسعه‌ی گره‌ای که به نظر می‌رسد نزدیک‌ترین به هدف است، تلاش می‌کند.[۳]

درست یا غلط- جستجوی اوّل بهترین حریصانه، جستجوی اوّل بهترین با $f(n) = h(n)$ است.

جواب- «درست» است.[۴]

تعریف- جستجوی A^* را به طور خلاصه تعریف کنید.

جواب- برای کمینه کردن مجموع هزینه‌ی راه حلّ تخمین زده شده تلاش می‌کند.[۵]

سؤال- یک توصیف مستقل از دامنه، برای مقدارهای f, g, h موجود در روش A^* ارائه کنید.

جواب-

$g(n)$ = هزینه‌ی بهترین مسیر پیدا شده تاکنون از گره‌ی شروع(S) تا گره‌ی جاری(n)

$h(n)$ = هزینه‌ی تخمین زده شده‌ی بهترین مسیر از گره‌ی n تا گره‌ی هدف(G)

$f(n) = g(n) + h(n)$ = هزینه‌ی تخمین زده شده‌ی بهترین مسیر برای رفتن به هدف از گره‌ی n.[۶]

---

۱ - تکلیف‌های خانگی شماره‌ی ۱ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا

۲ - تکلیف‌های خانگی شماره‌ی ۱ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا

۳ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۴ - تکلیف‌های خانگی شماره‌ی ۱ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا

۵ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۶ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۱۹۹۳ میلادی

تست– در تابع ارزیابی((f(n)) روش جستجوی A^* تفاوت میان تابع هزینه((g(n)) و تابع مکاشفه‌ای((h(n)) چیست؟
۱) تابع هزینه، هزینه‌ی واقعی را از گره‌ی جاری به هدف برمی‌گرداند، در حالی که تابع مکاشفه‌ای، هزینه‌ی تخمین زده شده از گره‌ی جاری تا هدف را برمی‌گرداند.
۲) تابع هزینه، هزینه‌ی تخمین زده شده از گره‌ی شروع تا گره‌ی جاری را برمی‌گرداند، در حالی که تابع مکاشفه‌ای، هزینه‌ی تخمین زده شده از گره‌ی جاری تا هدف را برمی‌گرداند.
۳) تابع هزینه، هزینه‌ی تخمین زده شده از گره‌ی جاری تا هدف را برمی‌گرداند، در حالی که تابع مکاشفه‌ای، هزینه‌ی واقعی از گره‌ی شروع تا گره‌ی جاری را برمی‌گرداند.
۴) تابع هزینه، هزینه‌ی واقعی از گره‌ی شروع تا گره‌ی جاری را برمی‌گرداند، در حالی که تابع مکاشفه‌ای، هزینه‌ی تخمین زده شده از گره‌ی جاری تا هدف را برمی‌گرداند.

جواب– گزینه‌ی «۴» درست است.[۱]

درست یا غلط– جستجوی A^*، جستجوی اوّل بهترین با $f(n) = g(n) + h(n)$ است.

جواب– «درست» است.[۲]

درست یا غلط– جستجوی با هزینه‌ی یکنواخت، جستجوی A^* با $h(n) = 0$ است.

جواب– «درست» است.[۳]

تعریف– «مکاشفه‌ی قابل قبول» را تعریف کنید.

جواب– تابعی که تخمین‌های هزینه‌ی بهترین اجرای یک راه حل به یک گره را به طوری که هیچگاه از هزینه‌ی واقعی تجاوز نکند، ارائه می‌کند.[۴]؛ یا به عنوان تعریفی دیگر، [مکاشفه‌ای که] هرگز هزینه‌ی ارزان‌ترین مسیر به یک وضعیّت هدف را بیش از حد برآورد نمی‌کند.[۵]

درست یا غلط– الگوریتم‌های جستجوی مکاشفه‌ای هیچوقت کامل نیستند.

---

۱ – آزمون میان‌ترم درس «هوش مصنوعی» استاد، «کتی مک‌کون»، دانشکده‌ی علوم کامپیوتر دانشگاه کلمبیای کشور آمریکا، ۲۶ اکتبر سال ۲۰۰۶ میلادی

۲ – تکلیف‌های خانگی شماره‌ی ۱ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا

۳ – تکلیف‌های خانگی شماره‌ی ۱ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا

۴ – آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۱۹۹۳ میلادی

۵ – آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

جواب- «غلط» است؛ به عنوان مثال، دیدیم که اگر تابع مکاشفه‌ای قابل قبول باشد، A^* کامل است.[۱]

درست یا غلط- اگر A^* دارای یک تابع مکاشفه‌ای غیرقابل قبول باشد، در این صورت، هیچوقت یک راه حلّ بهینه را پیدا نمی‌کند.

جواب- «غلط» است؛ این وابسته به مسأله است؛ A^* ممکن است راه حلّ بهینه را پیدا کند. بهتر است که بگوئیم، A^* با یک مکاشفه‌ی غیرقابل قبول، همیشه راه حلّ بهینه را پیدا نمی‌کند.[۲]

درست یا غلط- در درخت جستجویی که توسّط A^* به وجود می‌آید، دو گره با وضعیّت یکسان هیچوقت رُخ نمی‌دهد.

جواب- «غلط» است؛ در درخت جستجویی که توسّط A^* به وجود می‌آید، دو گره با وضعیّت یکسان می‌توانند رخ دهند، امّا ممکن است مقدار g آنها فرق کند.[۳]

تست- مکاشفه‌ی h2 بر h1 در صورتی دارای تسلّط[۴] است که:
۱) h1 برخلاف h2 هرگز هزینه‌ی رسیدن به هدف را بیش از حد برآورد نکند.
۲) h2 برخلاف h1 هرگز هزینه‌ی رسیدن به هدف را بیش از حد برآورد نکند.
۳) به ازای هر n داشته باشیم $h2(n) \geq h1(n)$، و h2 هرگز هزینه‌ی رسیدن به هدف را بیش از حد برآورد نکند.
۴) به ازای هر n داشته باشیم $h1(n) \geq h2(n)$، و h1 هرگز هزینه‌ی رسیدن به هدف را بیش از حد برآورد نکند.
۵) هیچکدام

جواب- گزینه‌ی «۳» درست است.[۵]

مطلب- (☑ کنکور سراسری مهندسی کامپیوتر سال ۸۴- مفهوم تسلّط: اگر به ازای هر n، برای دو تابع مکاشفه‌ای قابل قبول h_1 و h_2 داشته باشیم $h_2(n) \geq h_1(n)$، آنگاه h_2 بر h_1 دارای تسلّط است؛ به عبارت دیگر، h_2 برای جستجو بهتر است.)
به عنوان مثال، تعداد گره‌های توسعه داده شده برای مسأله‌ی پازل ۸-تایی به صورت زیر است:

d=12

---

۱ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۸ فوریّه‌ی سال ۲۰۱۰ میلادی

۲ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۱۰ نوامبر سال ۲۰۱۱ میلادی

۳ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۷ ژوئن سال ۲۰۱۲ میلادی

۴ - dominance

۵ - آزمون میان‌ترم درس «هوش مصنوعی» استاد، «روت شولتز»، دانشکده‌ی مهندسی برق و فنّآوری اطلاعات دانشگاه کوئینزلند کشور استرالیا، سال ۲۰۱۲ میلادی

IDS = 3644035
$A^*(h_1) = 227$
$A^*(h_2) = 73$

d=24

IDS = تعداد بسیار زیاد
$A^*(h_1) = 39135$
$A^*(h_2) = 1641$[۱]

مطلب- ☑مورد استفاده در کنکور سراسری مکاترونیک سال ۸۷- ترکیب مکاشفه‌ها: تصوّر کنید که مجموعه‌ای از مکاشفه‌های قابل قبول $h_1(n), h_2(n), \ldots, h_m(n)$ را داریم، به طوری که هیچکدام از آنها بر دیگری دارای تسلّط نیست، برای ترکیب آنها داریم:

$$h(n) = \max\{h_1(n), h_2(n), \ldots, h_m(n)\}$$

تست- کدامیک از روش‌های زیر جستجوی اوّل سطح و اوّل عمق را با هم ترکیب می‌کند؟

۱) جستجوی حریصانه
۲) جستجوی اوّل بهترین
۳) جستجوی A^*
۴) جستجوی عمیق شونده‌ی تکراری

جواب- گزینه‌ی «۴» درست است.[۲]

سؤال- ☑کنکور سراسری مهندسی کامپیوتر سال ۸۷- روش‌های A^*، IDA^*، RBFS و SMA^*، از لحاظ مصرف حافظه چگونه هستند؟

جواب- به ترتیب نزولی، از راست به چپ عبارتند از: A^*، IDA^*، RBFS و SMA^*.

مطلب- ☑کنکور سراسری مهندسی کامپیوتر سال ۸۲- روش IDA^* از ملاقات مجدّد وضعیّت‌ها نمی‌تواند جلوگیری کند[۳](دوباره‌کاری دارد).

درست یا غلط- ☑مشابه کنکور سراسری مهندسی کامپیوتر سال ۸۳- اگر h1 و h2 مکاشفه‌هایی قابل قبول باشند، آنگاه $(h1+h2)/2$ هم قابل قبول است.

---

۱ - مطلب‌های درس «هوش مصنوعی» استادیار، «سْوِتْلانا لِیزْبِنیکْ»، دانشکده‌ی علوم کامپیوتر دانشگاه ایالت ایلینویز کشور ایالات متّحده‌ی آمریکا، پاییز سال ۲۰۱۳ میلادی

۲ - آزمون میان‌ترم درس «هوش مصنوعی» استاد، «کتی مک‌کون»، دانشکده‌ی علوم کامپیوتر دانشگاه کلمبیای کشور آمریکا، ۲۹ اکتبر سال ۲۰۰۷ میلادی

۳ - مطالب موجود در پایگاه اینترنتی «آزمایشگاه هوش مصنوعی استنفورد»

جواب- «درست» است.[۱]

درست یا غلط- اگر h1 مکاشفه‌ای قابل قبول باشد، آنگاه 2*h1 هم قابل قبول است.

جواب- «غلط» است.[۲]

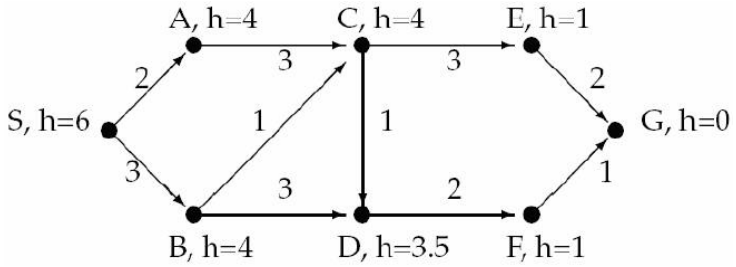
درست یا غلط- اگر h1 و h2 مکاشفه‌هایی قابل قبول باشند، آنگاه max(h1,h2) هم قابل قبول است.

جواب- «درست» است.[۳]

درست یا غلط- اگر h1 و h2 مکاشفه‌هایی قابل قبول باشند، آنگاه روش جستجوی A^* با مکاشفه‌ی H=max(h1,h2) هدف را دست کم به سرعت مکاشفه‌ی F=(h1+h2)/2 پیدا خواهد کرد.

جواب- «درست» است؛ از آنجایی که میانگین کوچک‌تر از بیشینه(max) است، H، دست کم به سرعت F است.[۴]

سؤال- در گراف زیر نام گره‌ها با حروف بزرگ انگلیسی نشان داده شده است، h، هزینه‌ی تخمینی آن گره تا هدف، و عدد روی یال‌ها، هزینه‌ی واقعی انتقال بین گره‌ها است؛ آیا این مکاشفه قابل قبول است؟



جواب- نه؛ چونکه تابع مکاشفه‌ای h در گره D تخمین اضافی ۳٫۵ زده است.[۵]

سؤال- اگر جستجوی درختی را روی گراف زیر اجرا کنیم، آیا A^* هدف بهینه را پیدا می‌کند؟، چرا؟؛ توجّه کنید که هزینه‌ی گام‌ها، روی یال‌ها و hها، که مقدارهای تابع مکاشفه‌ای هستند، در کنار گره‌ها نشان داده شده‌اند.

---

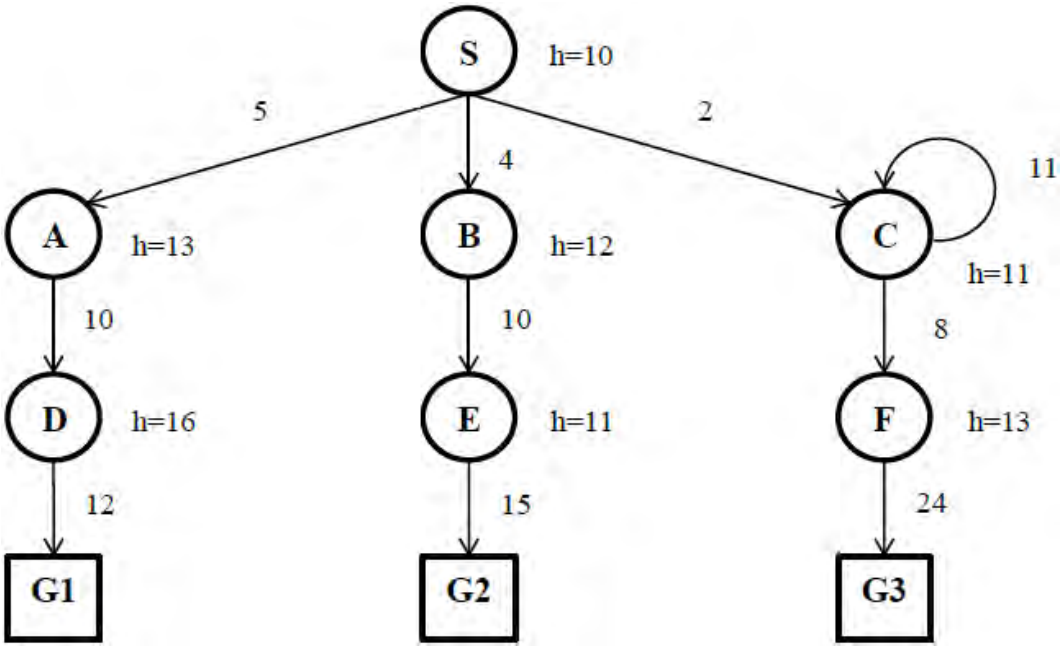
۱ - تکلیف‌های خانگی شماره‌ی ۲ درس «آشنایی با هوش مصنوعی» استاد، «دیمیتری پاولوف(Dmitry Pavlov)»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، زمستان سال ۲۰۰۱ میلادی

۲ - تکلیف‌های خانگی شماره‌ی ۲ درس «آشنایی با هوش مصنوعی» استاد، «دیمیتری پاولوف»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، زمستان سال ۲۰۰۱ میلادی

۳ - تکلیف‌های خانگی شماره‌ی ۲ درس «آشنایی با هوش مصنوعی» استاد، «دیمیتری پاولوف»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، زمستان سال ۲۰۰۱ میلادی

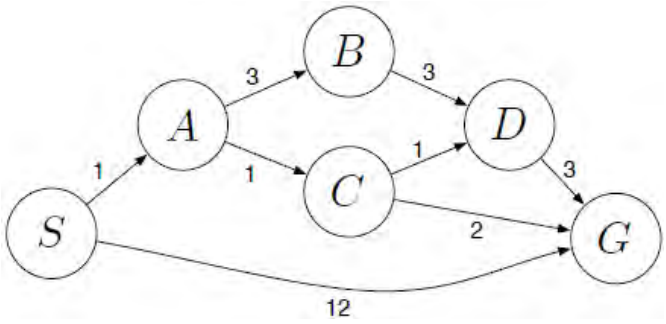
۴ - تکلیف‌های خانگی شماره‌ی ۲ درس «آشنایی با هوش مصنوعی» استاد، «دیمیتری پاولوف»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، زمستان سال ۲۰۰۱ میلادی

۵ - مطلب‌های درس «آشنایی با هوش مصنوعی» استاد، «دَبلیو. اِن. مارتین(W.N. Martin)»، دانشکده‌ی علوم کامپیوتر دانشگاه ایالت ویرجینیای کشور ایالات متّحده‌ی آمریکا، پاییز سال ۲۰۰۴ میلادی



جواب- نه؛ چون مکاشفه در گره‌ی D تخمین اضافی برابر با ۱۶ زده است و قابل قبول نمی‌باشد.[۱]

سؤال مهم- ☑مشابه سؤالات مطرح شده در کنکورهای مختلف سراسری مربوط به رشته‌ی کامپیوتر سال‌های مختلف- به گراف زیر توجّه کنید، در این گراف هزینه‌ی واقعی انتقال بین گره‌ها روی یال‌های گراف نشان داده شده است و می‌خواهیم از گره‌ی S به گره‌ی G برویم؛ اگر در این گراف از یک مکاشفه‌ی یکنوا استفاده کنیم، چه مسیری برای جستجوی A^* برگردانده می‌شود؟[۲]



جواب- (راه حلّ تستی) چون در صورت سؤال، ذکر شده که مکاشفه یکنوا می‌باشد، پس A^* مسیر بهینه‌ی $S - A - C - G$ را که کم هزینه‌ترین مسیر است، برمی‌گرداند. لازم به ذکر است که اگر مکاشفه یکنوا نباشد، نمی‌توان از این «راه حلّ تستی» استفاده کرد.

---

۱ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، زمستان سال ۲۰۱۲ میلادی

۲ - آزمون میان‌ترم درس «هوش مصنوعی» استاد، «پیتِر اَبیل(Pieter Abbeel)»، دانشکده‌ی شهر بِرکلی دانشگاه ایالت کالیفُرنیای کشور ایالات متّحده‌ی آمریکا، بهار سال ۲۰۱۲ میلادی

تست- در کل، کدامیک از روش‌های جستجوی زیر در زمانی که الف) فضای جستجو بزرگ است، ب) عمق راه حل نامعیّن است، پ) دنبال راه حلّ بهینه می‌گردیم، ت) یک مکاشفه‌ی قابل قبول یکنوا داریم، ترجیح داده می‌شود؟

۱) جستجوی اوّل عمق

۲) جستجوی اوّل سطح

۳) جستجوی با هزینه‌ی یکنواخت

۴) جستجوی با عمق محدود شده

۵) جستجوی A^*

۶) جستجوی دوطرفه

۷) جستجوی اوّل بهترین حریصانه

۸) جستجوی عمیق شونده‌ی تکراری

جواب- گزینه‌ی «۵» درست است.[۱]

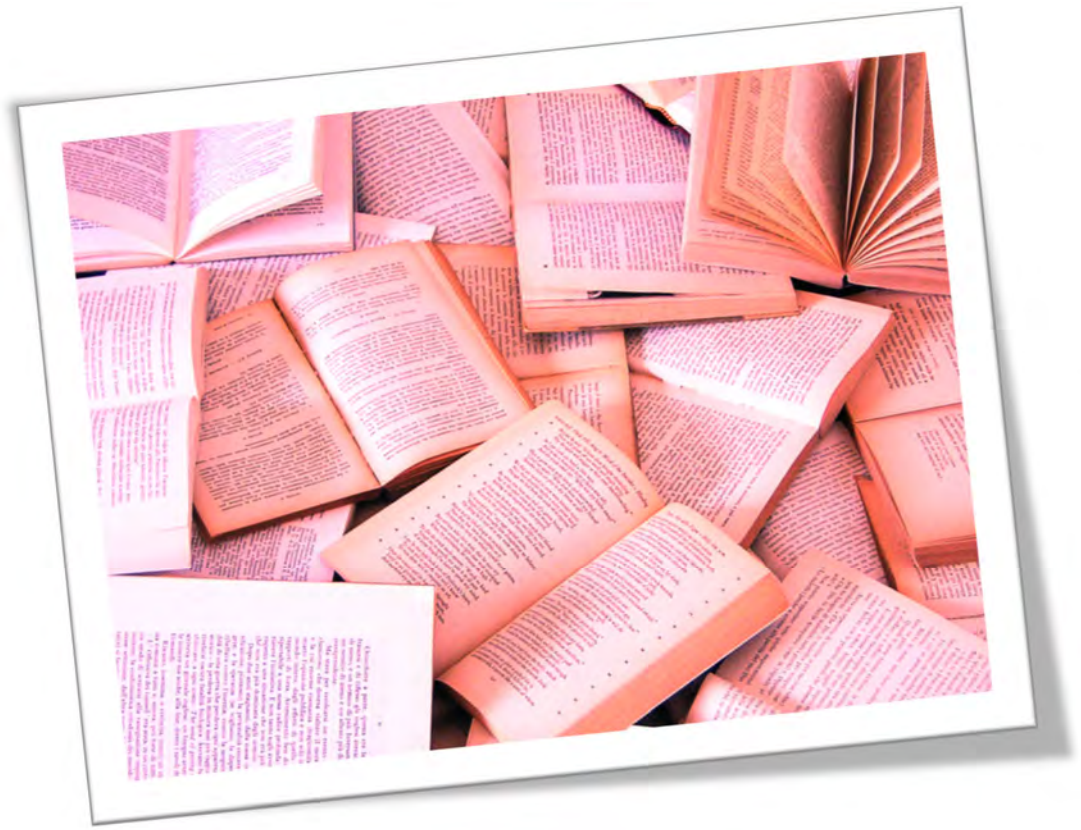
۱ - کوییز شماره‌ی ۲ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا

فصل ششم

۱ الگوریتم‌های جستجوی محلّی ۲

۱ - روش جستجوی تپّه‌نوردی که در همین فصل با آن آشنا خواهیم شد، شبیه به این است که در مه غلیظ در حال بالا رفتن از اِورِسْتْ باشیم و راه خودمان را گم کرده باشیم.

۲ - local search algorithms



فهرست برخی از عنوان‌های نوشته‌ها

مثال- انجام دادن یک الگوریتم ژنتیکی (۸-وزیر)

مسأله‌های بهینه‌سازی[۱]

یادآوری- مسأله‌های بهینه‌سازی: مسأله‌هایی هستند که در آنها باید بهترین راه حل را در بین راه حل‌های موجود پیدا کنیم.[۲]

در برخی از ساختارهای ترکیبی که ما در تلاش برای بهینه‌سازی آنها هستیم، محدودیّت‌ها باید در نظر گرفته شوند و تابع هزینه باید بهینه شود. اغلب پیدا کردن راه حل، آسان است، امّا پیداکردن بهترین راه حل، سخت می‌باشد و پیداکردن همه‌ی راه حل‌های ممکن هم غیرممکن می‌باشد؛ ما دست‌کم یک راه حلّ خوب را می‌خواهیم.

روش بهبود الگوریتم‌های تکرار شونده[۳]

در برخی از مسأله‌های بهینه‌سازی، فضای حالت، یک مجموعه از وضعیّت‌های هدف است؛ که مثلاً ممکن است بخواهید یک مجموعه‌ی بهینه را پیدا کنید، مثل مسأله‌ی فروشنده‌ی دوره‌گرد:

۱- optimization problems

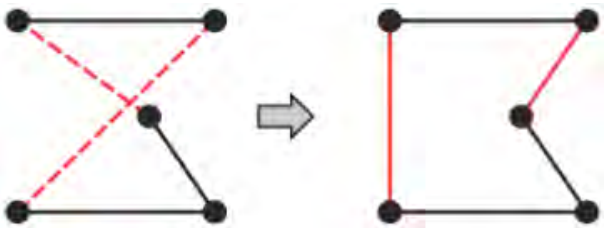
۲- http://en.wikipedia.org/wiki/Optimization_problem

۳- iterative improvement algorithms

شکل بالا- یادآوری صورت مسأله‌ی فروشنده‌ی دوره‌گرد: فروشنده‌ی دوره‌گردی می‌خواهد مسیری که از n شهر عبور می‌کند را طوری پیدا کند که از هر شهر یکبار بگذرد و در ضمن، کوتاه‌ترین مسیر هم باشد.

یا مثلاً ممکن است مثل طرّاحی یک برنامه‌ی زمانی[۱] بخواهید مجموعه‌ای را که محدودیّت‌هایی را برآورده می‌کنند، پیدا کنید؛ در این مورد شما می‌توانید از روش بهبود الگوریتم‌های تکرار شونده استفاده نمایید؛ در این روش با یک راه حل شروع نمایید، که می‌تواند غلط یا غیربهینه باشد و آن را به طرف یک راه حلّ بهینه ببرید؛

مثال- مسأله‌ی فروشنده‌ی دوره‌گرد- با یک مسیر شروع کنید و هزینه‌ی آن را با تعویض شهرها بهبود دهید:



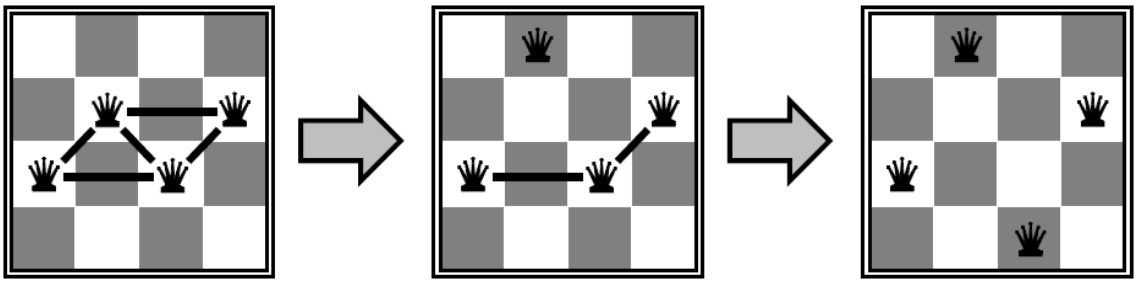
مثال- وزیرهای n-تایی(n-وزیر)[۲]: یادآوری صورت مسأله- n وزیر را در یک صفحه‌ی شطرنجی n×n بدون اینکه هیچ یک از دو وزیر، در سطر[۳] یا ستون[۱] همانند قرار گیرند یا به طور مورّب[۲] همانند قرار گیرند، قرار دهید.

۱- timetable(schedule)

۲- n-queens

۳- row

به شکل زیر که ۴–وزیر را نشان می‌دهد، توجّه نمائید:



در سمت چپ شکل قبل، در ابتدا پنج حالت ناسازگار داریم؛ با حرکت دادن یک وزیر، در وسط شکل قبل، به دو حالت ناسازگار می‌رسیم؛ با حرکت دادن یک وزیر دیگر، در سمت راست شکل قبل، هیچ حالت ناسازگاری نخواهیم داشت.

روش‌های تپّه‌نوردی و شبیه‌سازی گرم و سرد کردن که در ادامه‌ی همین فصل با آنها آشنا خواهیم شد، جزء روش‌های بهبود الگوریتم‌های تکرار شونده هستند.

جستجوی محلّی

الگوریتم‌هایی که در فصل‌های قبل برّرسی کرده‌ایم، تمام مسیر را از وضعیّت اوّلیّه به وضعیّت نهائی نگهداری می‌کنند و این باعث مصرف حافظه(فضا)ی زیاد، در مورد مسأله‌های بزرگ می‌شود. برای برخی از مسأله‌ها اطّلاعات مسیر ضروری می‌باشد؛ مثل مسیریابی و پازل ۸–تایی؛ در این مسأله‌ها هدف را می‌دانیم، امّا اینکه چگونه به آن برسیم را نمی‌دانیم. برای دسته‌ای دیگر از مسأله‌ها ممکن است نگران چگونگی رشته‌ی عملکردها نباشیم و پیدا کردن راه حلّ بهینه چیزی است که مهم می‌باشد؛ مثل زمانبندی، طرح VLSI و پنهانْشناسی(رمزنویسی)[۳]؛ در این موردها ما می‌توانیم با اطمینان برخی از اطّلاعات مسیر را حذف نماییم.

تعریف– *یک الگوریتم جستجو که فقط از وضعّیت جاری استفاده می‌کند، یک الگوریتم جستجوی محلّی نام دارد؛*

مزایای این روش عبارتند از: به حافظه‌ی اندکی نیازمندیم و می‌توانیم راه حل‌هایی باورنکردنی را در مورد فضاهای بزرگ پیدا نماییم. اینکه ممکن است در ماکزیمم محلّی گیر بیفتد، از عیب‌های این روش است. در این روش، فضای حالت، شبیه روی زمین،

---

۱ – column

۲ – diagonal

۳ – cryptography: نوشتن یا فهمیدن کدهای رمز (Babylon> Babylon English - English)

دارای تپّه‌ها و درّه‌ها می‌باشد و *ارتفاع*[۱] به وسیله‌ی تابع هزینه(ارزیابی) داده شده است[، به بیان دیگر، هر چه ارتفاع بیش‌تر باشد، هزینه بیش‌تر خواهد بود؛ مثل شکل زیر].

شکل بالا– تپّه‌نوردی روی یک سطح که از وضعیّت‌ها درست شده است؛ ارتفاع به وسیله‌ی تابع ارزیابی تعریف می‌شود.

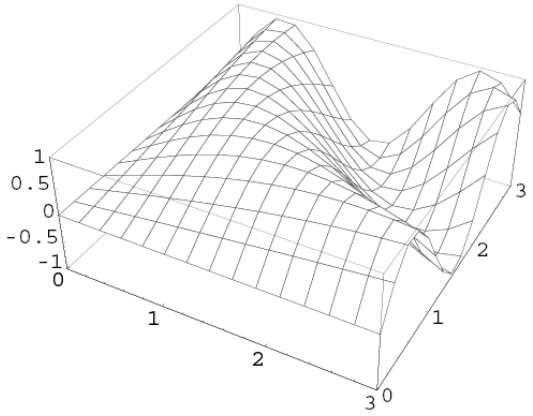
فضای حالت یک بعدی[۲] برای مثال‌ها آسان‌تر نمایش داده می‌شود:

۱ – elevation

۲ – one-dimensional

دورنمای(چشم‌انداز)[۱] جستجو

جستجوی محلّی، زمانی که فضای حالت، ترکیبی از پارامترها می‌باشد، مشکل است. در صورتی که n پارامتر وجود داشته باشد، می‌توانیم یک فضای n+1 بعدی را تصوّر نماییم، که n بعد اوّل، پارامترهای تابع می‌باشند و n+1 اُمین بعد، *تابع هدف*[۲] می‌باشد، که این فضا را یک چشم‌انداز جستجو می‌نامیم. در شکل زیر یک چشم‌انداز دو بعدی نشان داده شده است:



دورنماها یک روش نمایش خوب برای الگوریتم‌های جستجوی محلّی هستند. بالا رفتن از یک تپّه را در نظر مجسّم کنید؛ این مورد یک راه را برای تشخیص مسأله‌های آسان از مسأله‌های سخت ارائه می‌کند. حالت‌های آسان؛ تعداد قلّه‌های کم، سطوح صاف، بدون تپّه‌های دامنه‌ی کوه[۳] یا ماکزیمم محلّی مسطّح(فلات[۴]) هستند و حالت‌های سخت؛ تعداد قلّه‌های زیاد، سطوح دندانه‌دار یا ناپیوسته(گسسته) هستند.

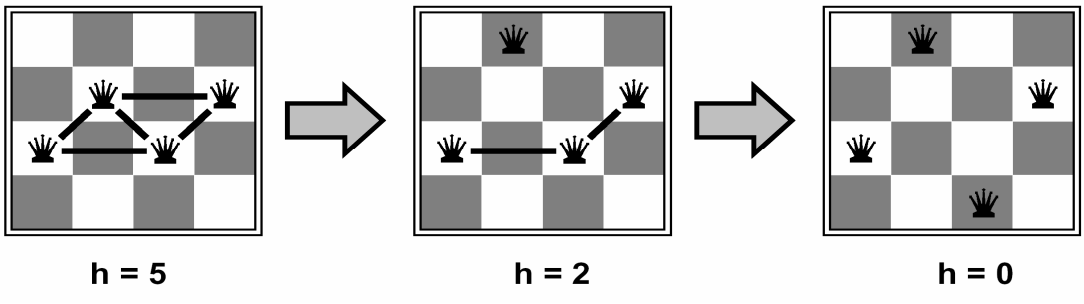
مثال- در شکل زیر که شکل مسأله‌ی ۴-وزیر است، وقتی که h = 0 می‌باشد، هیچ حالت نادرستی را نداریم و در این مورد کاملاً راحت هستیم؛ ولی وقتی که h = 2 است، دارای دو حالت نادرست هستیم (چونکه دو وزیر در یک ردیف قرار گرفته‌اند و دو وزیر دیگر به صورت مورّب قرار گرفته‌اند) و کار ما نسبت به حالتی که h = 0 بود، سخت‌تر است؛ و در حالت h = 5، چونکه پنج مورد نادرست داریم، کار ما نسبت به دو مورد قبلی که h = 0 و h = 2 بود، سخت‌تر است. در حالتی که h = 0 است، می‌توانیم تصوّر کنیم که در جایی مسطّح قرار گرفته‌ایم و در حالتی که h = 5 است، می‌توان تصوّر کرد که در شیب(سربالایی) یک تپّه قرار گرفته‌ایم.

۱- landscape

۲- objective function؛ **نکته:** در بهینه‌سازی، تابع هدف، تابعی است که باید بیشینه(ماکزیمم) یا کمینه(مینیمم) شود.
http://en.wiktionary.org/wiki/objective_function (ویکیشِنری، همانند ویکی‌پدیا، واژه‌نامه‌ای اینترنتی و رایگان است.)

۳- ridge

۴- plateau



تپّه‌نوردی[۱]

مطلب مهم:

ساده‌ترین شکل جستجوی محلّی، جستجوی تپّه‌نوردی می‌باشد و دارای روشی ساده است، به این ترتیب که در هر گام(نقطه) به جانشینان(همسایه‌ها) نگاه کنید و در جهت بهتر؛ یعنی، دارای مقدار بیش‌تر، در صورتی که به دست آوردن یک راه حلّ دارای بالاترین مقدار ممکن مورد نظر باشد، حرکت کنید[۲]؛ *توجّه کنید که در صورتی که به دست آوردن یک راه حلّ دارای پایین‌ترین مقدار ممکن مورد نظر باشد، در جهت دارای مقدار کم‌تر حرکت نمایید.* این روش خیلی شبیه به جستجوی حریصانه می‌باشد و جستجوی محلّی حریصانه[۳] هم نامیده می‌شود و به حافظه‌ی خیلی اندکی نیاز دارد. فلات(ماکزیمم محلّی مسطّح) می‌تواند سبب این شود که الگوریتم، بی‌هدف، سرگردان باشد.

---

۱ - Hill-climbing

۲ - **توجّه**- این گونه، که بیان شد، تپّه‌نوردی از تندترین شیب(steepest ascent hill climbing) نامیده می‌شود؛ در تپّه‌نوردی ساده(simple hill climbing)، اوّلین گره‌ی بهتر انتخاب می‌شود، در حالی که در تپّه‌نوردی از تندترین شیب، همه‌ی جانشینان، مقایسه شده و بهترین گره انتخاب می‌شود. (http://en.wikipedia.org/wiki/Hill_climbing)

۳ - Greedy local search

شکل بالا– روش جستجوی تپّه‌نوردی شبیه به این است که در مه غلیظ در حال بالا رفتن از اِورِست باشیم و راه خودمان را گم کرده باشیم.

الگوریتم تپّه‌نوردی

الگوریتم

تابع Hill-Climbing(problem)، یک وضعیّت(حالت) را که یک ماکزیمم محلّی است، برمی‌گرداند

ورودی‌ها، problem، که یک مسأله است، می‌باشد

متغیّرهای محلّی، current[1] و neighbor، که هر کدام، یک گره هستند، می‌باشند

current ← Initial-State(problem)

در حلقه کارهای زیر را انجام بده

بالاترین مقدار current را در neighbour قرار بده

در صورتی که Value(neighbor)≤Value(current) بود، current را برگردان

---

۱ – در لغت به معنی «جاری، کنونی» است.

current ←n۱۹۹eighbor

پایان حلقه

پایان الگوریتم

ویژگی‌های تپّه‌نوردی

مطلب مهم:

پیاده‌سازی آن آسان می‌باشد و مقدار اندکی از حافظه را استفاده می‌نماید؛ ولی تپّه‌نوردی زیاد تحت تأثیر شکل فضای حالت است؛ در تعداد کم ماکزیمم محلّی، خوب می‌باشد؛ ولی برای فضاهای جوجه تیغی[۲](دندانه‌دار) مانند، بد است؛ در شانه[۳]ها، ماکزیمم محلّی و تپّه‌های دامنه‌ی کوه(رشته‌ای از ماکزیمم‌های محلّی)[۴] دچار مشکل می‌شود؛ در ماکزیمم محلّی نمی‌تواند قلّه‌ی بالاتر را ببیند؛ *با شروع مجدّد تصادفی، الگوریتم تپّه نوردی، بر ماکزیمم محلّی غلبه می‌کند؛* در شانه، راهی به طرف بیرون را نمی‌تواند پیدا نماید و در تپّه‌های دامنه‌ی کوه برای حرکت باید به طرف پایین حرکت نماید و حرکت‌های بد را مُجاز بداند.

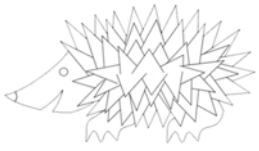
مثالی برای مینیمم(کمینه‌ی) محلّی مسأله‌ی ۸-وزیر

در مسأله‌ی ۸-وزیر فرض کنید تابع هزینه‌ی مکاشفه‌ای h، تعداد جفت وزیرهایی باشد که در حال حمله به یکدیگر هستند(همدیگر را تهدید می‌کنند). مینیمم سراسری(مطلق) این تابع فقط در صورتی صفر است که راه حلّ کامل را داشته باشیم. شکل زیر وضعیّتی را با h=17 نشان می‌دهد؛ این شکل همچنین مقدار همه‌ی جانشینان را نشان می‌دهد، که بهترین جانشینان دارای مقدار h=12 هستند. توجّه کنید که الگوریتم‌های تپّه‌نوردی در صورتی که بیش از یک بهترین جانشین وجود داشته باشد، از میان آنها یکی را به طور تصادفی انتخاب می‌کنند:

---

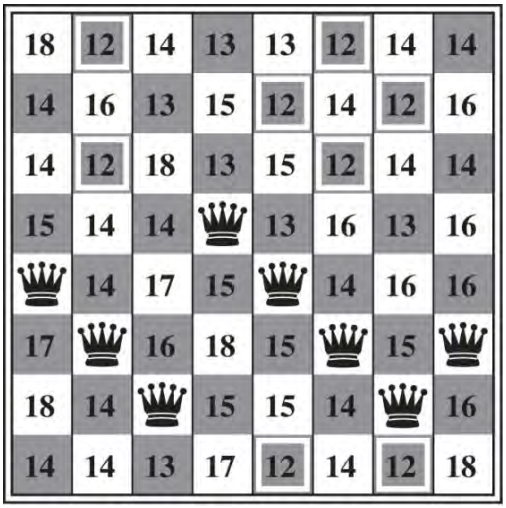
۱ - در لغت به معنی «مقدار» است.

۲ - urchin یا hedgehog:



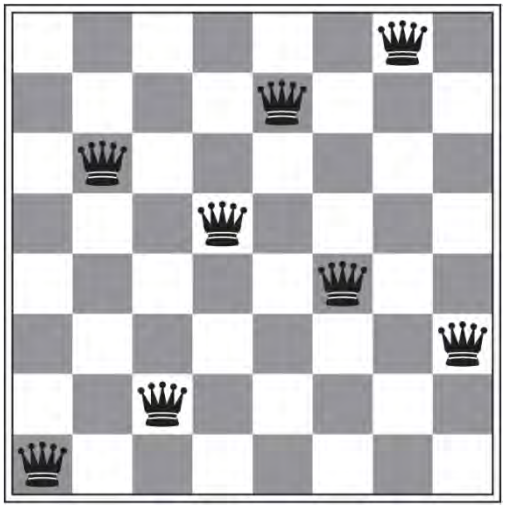
۳ - shoulder

۴ - ridge



شکل بالا– یک وضعیّت ۸–وزیر با تخمین هزینه‌ی مکاشفه‌ی h=17، که مقدار h را برای هر جانشینی ممکن به دست آمده از حرکت یک وزیر در ستونش نشان می‌دهد. بهترین حرکت‌ها مشخّص شده‌اند.

در شکل زیر یک مینیمم محلّی در ۸–وزیر نشان داده شده است؛ با هر حرکت یک وزیر، وضعیّت بدتر خواهد شد:



شکل بالا– یک مینیمم محلّی فضای حالت ۸–وزیر؛ این وضعیّت دارای h=1 است، امّا هر جانشین دارای هزینه‌ی بیش‌تری است.[۱]

گوناگونی‌های تپّه‌نوردی

تپّه‌نوردی اتّفاقی(تصادفی)[۲]: در میان حرکت‌های به طرف سربالایی یکی را به تصادف انتخاب می‌نماید.

تپّه‌نوردی تصادفی با انتخاب بهترین[۳]: همان تپّه‌نوردی تصادفی را آنقدر تکرار می‌کند تا یک مورد بهتر پیدا شود.

---

۱ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل چهارم، فراتر از روش‌های جستجوی کلاسیک(Beyond Classical Search)، صفحه‌های ۱۲۲ و ۱۲۳

۲ - Stochastic hill-climbing

۳ - First-choice hill-climbing

تپّه‌نوردی با شروع مجدّد تصادفی[۱]: [با این کار] تلاش می‌کند که از افتادن در ماکزیمم محلّی جلوگیری نماید؛ در این روش اگر در ابتدا موفّق نشدید، آن قدر تلاش کنید تا موفّق شوید.

روش شبیه‌سازی گرم و سرد کردن[۲]

مطلب مهم:

روش شبیه‌سازی گرم و سرد کردن تلاشی برای برطرف نمودن مشکلات روش تپّه‌نوردی است و در واقع گونه‌ای از روش تپّه‌نوردی می‌باشد. ایده‌ی این روش، گریختن از ماکزیمم محلّی با اجازه دادن به بعضی از حرکت‌های بد یا نادرست(حرکت به طرف پایین تپّه) می‌باشد. امّا به تدریج تعداد و فراوانی[۳] حرکت‌های بد کاهش می‌یابد. در واقع این روش، تپّه‌نوردی را با حرکت تصادفی ترکیب می‌نماید.

شبیه‌سازی گرم و سرد کردن

وقتی که یک فلز، مثل آهن را خیلی گرم می‌کنید، دارای ارتعاش‌های زیادی است، ولی وقتی که به تدریج دما کم می‌شود، ارتعاش‌های آن هم به تدریج کم‌تر می‌شود،

۱- Random-restart hill-climbing
۲- Simulated annealing
۳- frequency

در تپّه‌نوردی، این کار مثل این است که از بیشینه‌ی محلّی با اجازه دادن به برخی از حرکت‌های بد بگریزید، امّا بـه تـدریج تعـداد و فراوانی آنها(حرکت‌های بد) را کاهش دهید.

الگوریتم شبیه‌سازی گرم و سرد کردن

الگوریتم

تابع Simulated-Annealing(problem,schedule)، یک راه حل را برمی‌گرداند

ورودی‌ها، یک مسأله و schedule، که یک نگاشت از زمان به دما می‌باشد، هستند

متغیّرهای محلّی عبارتند از: current، که یک گره است و next، که یک گره است و T، که یک برّرسی کننـده[1]ی دمـا از مراحل پائین‌تر است

current ← Make-Node(Initial-State[problem])

برای مقادیر t = 1 تا ∞(بینهایت) کارهای زیر را انجام بده:

T ← schedule[t]

در صورتی که T=0 بود، گره‌ی current را برگردان

یک جانشین به تصادف انتخاب شده از گره‌ی current را به گره‌ی next نسبت بده

ΔE ←Value[next] – Value[current]

در صورتی که ΔE > 0 بود، محتوای گره‌ی next را در گره‌ی current بریز

در غیر این صورت محتوای گره‌ی next را فقط با احتمال $e^{\Delta(E/T)}$ در گره‌ی current کپی کن

پایان حلقه

پایان الگوریتم

روش شبیه‌سازی گرم و سرد کردن همیشـه حرکت‌هـای خـوب را می‌پـذیرد؛ *ولـی در آن احتمـال پـذیرش یـک حرکت بد وجود دارد.*

---

۱ - پُرُبْ(probe)

(☑**مطلب کنکور سراسری مکاترونیک سال ۸۴**- از روش شبیه‌سازی گرم و سرد کردن در بسیاری از موردها مثل قالب‌بندی VLSI)، زمانبندی خطوط هوایی و غیره استفاده می‌شود. شبیه‌سازی گرم و سرد کردن در صورتی که T به اندازه‌ی کافی با سرعت کم‌تر کاسته شود، کامل و بهینه می‌باشد. روش شبیه‌سازی گرم و سرد کردن به‌سادگی به الگوریتم تپّه‌نوردی اضافه می‌شود. *ضعف این روش در انتخاب یک زمانبندی خنک کننده‌ی خوب می‌باشد.*

جستجوی پرتو محلّی[۱]

ایده: به جای یک حالت k، k حالت را در نظر بگیرید.

الگوریتم

تابع Beam-Search(problem, k)، یک وضعیّت راه حل را برمی‌گرداند

با k وضعیّت به طور تصادفی تولید شده شروع کن

در حلقه کارهای زیر را انجام بده

تمام جانشینان همه‌ی این k حالت را به وجود آورید

اگر هر کدام از آنها(جانشینان) راه حل(هدف) بودند، آن را برگردانید و کار را خاتمه دهید

در غیر این صورت kتا از بهترین جانشینان را انتخاب کرده و به ابتدای حلقه بروید

پایان الگوریتم

اگر به جای انتخاب kتا از بهترین جانشینان، kتا را به صورت تصادفی انتخاب کنیم، *جستجوی پرتو محلّی اتّفاقی(تصادفی)*[۲] خواهیم داشت.

۱- local beam search

۲- stochastic local beam search

الگوریتم‌های ژنتیکی[۱](پیدایشی یا تکوینی) (به طور خلاصه[۲])

مطلب مهم:

ایده‌ی اساسی الگوریتم‌های ژنتیکی این است که برخی از راه حل‌ها را به صورت تصادفی انتخاب نمایید (مثل جستجوی پرتوی محلّی اتّفاقی یا تصادفی) و برای به‌دست‌آوردن راه حل‌های جدید، بهترین تکّه‌های راه حل‌ها را با هم عوض نمایید و این کار را تکرار نمایید.

اصطلاحات الگوریتم ژنتیکی

تعریف- جمعیّت[۳]، به صورت اوّلیّه مجموعه‌ای از k وضعیّت تولید شده به صورت تصادفی است.
به بیانی دیگر، *مجموعه‌ای از راه حل‌ها* که به وسیله‌ی کُرُوموزُوم[۴]ها ارایه می‌شوند، جمعیّت نام دارند.

تعریف- نسل[۵]، جمعیّت در یک نقطه از زمان است.

تعریف- فرد[۶](یک راه حل)، یک عنصر از جمعیّت است و به صورت یک رشته(ژِنْها) با الفبای محدود(مثلاً صفر و یک) توصیف می‌شود. *به هر کاراکتر موجود در رشته هم یک «ژن» می‌گویند.*
مثلاً یک فرد(راه حل) ممکن است به صورت «۱۱۰۰۱۰۱۰۰» باشد و فرد(راه حلّ) دیگری به صورت «۰۱۰۱۱۱۰۰۰۱» باشد.

تعریف- تابع شایستگی[۷]، بهینگی یک راه حل را در یک الگوریتم ژنتیکی تعیین می‌کند.[۸] واژه‌ی شایستگی بیان کننده‌ی خوبی یک راه حل می‌باشد.

تعریف- ژِنوم[۱]، که گاهی کروموزوم هم نامیده می‌شود؛ یک راه حل(فرد) یا وضعیّت است.

۱- Genetic *a*lgorithms(*GA*)؛ برای توضیح‌های بیش‌تر به فصل «الگوریتم‌های ژنتیکی» همین کتاب مراجعه نمایید.
۲- briefly
۳- population
۴- chromosome، این مطلب در ادامه توضیح داده شده است.
۵- generation
۶- individual
۷- fitness function
۸- http://en.wikipedia.org/wiki/Fitness_function

تعریف- ویژگی /(یا) ژِن[۲]- یک راه حل(فرد) از رشته‌ای به نام ژن‌ها تشکیل شده است. همان طور که گفتیم، به هر کاراکتر موجود در رشته، یک ژن می‌گویند.

تعریف- تعویض[۳]: *تعویض، دو قسمت کردن*[۴] *دو راه حل در یک نقطه‌ی تعویض و عوض کردن آنها با هم می‌باشد؛* این کار(عمل تعویض) دو وضعیّت جدید را تولید می‌نماید. در واقع تعویض، قطعات راه حل‌های موفّق، به صورت جزئی را با هم عوض می‌کند.

نکته:

در صورتی که یک فرد(راه حل) از جمعیّت اوّلیّه گم شده باشد، عمل تعویض نمی‌تواند آن را تولید کند.

نقطه‌ی تعویض: برای جلو بردن کار استفاده می‌شود و اغلب به صورت تصادفی انتخاب می‌شود؛ *نقطه‌ی تعویض برای هر دو راه حل باید یکسان باشد* و باید نسل‌های قابل قبول را تولید نماید.

تعریف- جهش[۵]: جهش، که بعد از انجام عمل تعویض انجام می‌شود، تولید مشخّصه‌های جدید، که در والدها وجود ندارد، با تغییرات تصادفی اجزای کد شده‌ی نسل می‌باشد. در واقع جهش، راه حل‌های جدید را به جمعیّت اِعمال می‌کند.

یک الگوریتم ژنتیکی پایه

الگوریتم

یک جمعیّت تصادفی از راه حل‌ها را تولید کن

مادامی که راه حل‌ها به اندازه‌ی کافی خوب نشده‌اند، کارهای زیر را انجام بده(حلقه)

راه حل‌ها را مورد ارزیابی قرار بده

۱- genome: در زیست‌شناسی، جمع تعداد ژن‌های درون یک سلّول تناسلی است. (Babylon> Babylon English-English)

۲- trait/gene: در زیست‌شناسی، بخشی از یک کُروموزوم(chromosome، قسمت میکروسکوپی و نخ مانند(threadlike) سلّول است و اطّلاعات ارثی(hereditary) را به وسیله‌ی ژن‌ها منتقل می‌کند. (Babylon> Britannica.com)) است که ویژگی‌های ارثی معیّنی را منتقل می‌کند. (Babylon> Babylon English-English)

۳- crossover

۴- split

۵- mutation

راه حل‌های تصادفی را انتخاب نما

عمل تعویض را روی راه حل‌ها انجام بده

عمل جهش را روی راه حل‌ها انجام بده

راه حل‌های جدید را تولید کن

پایان حلقه

پایان الگوریتم

مثال– انجام دادن یک الگوریتم ژنتیکی (۸–وزیر)

رمز(کد) کردن مسأله

برای هر ستون صفحه‌ی شطرنجی ۸–وزیر، از سه بیت برای رمز نمودن ردیف وزیر استفاده می‌کنیم = ۳×۸ = ۲۴ بیت. مثلاً در زیر کد کردن یکی از حالت‌های ۸–وزیر نشان داده شده است:

100 101 110 000 101 001 010 110 = 4 5 6 0 5 1 2 6

توجّه:

در این مثال، شماره‌گذاری ردیف‌های صفحه‌ی ۸–وزیر، از پایین و از عدد صفر شروع شده است؛ یعنی، ردیف اوّل، دارای شماره‌ی صفر؛ ردیف دوّم، دارای شماره‌ی ۱ و ...؛ به عنوان مثال، اگر یک وزیر در ردیف چهارم قرار گرفته، به آن عدد ۳ نسبت داده شده است.

برای تولید راه حل‌ها، با تولید رشته‌های بیتی تصادفی شروع می‌کنیم، سپس آنها را با توجّه به تابع شایستگی(تابعی برای بهینه نمودن)، مورد ارزیابی قرار می‌دهیم.

تولید راه حل‌های جدید با استفاده از عمل تعویض

برای انجام عمل تعویض، دو راه حل را به صورت تصادفی انتخاب نمایید. برای به دست آوردن شایستگی مناسب، برای انتخاب یک راه حل؛ شایستگی آن راه حل را بر مجموع تمام شایستگی‌ها تقسیم کنید؛ داریم؛ احتمال انتخاب راه حلّ S_i، که

$$i = \{1, 2, 3, \dots, k\}$$

برابر است با، شایستگی s_i تقسیم بر مجموع شایستگی‌ها:

$$P(s_i) = \frac{\text{Fitness}(s_i)}{\sum_{j=1}^{k} \text{Fitness}(s_j)}$$

در فرمول بالا، $\text{Fitness}(s_i)$(صورت کسر)، شایستگی راه حلّ iاُم است و $\sum_{j=1}^{k} \text{Fitness}(s_j)$(مخرج کسر)، مجموع تمام شایستگی‌ها می‌باشد.

سپس یک نقطه‌ی(مکان هندسی) تصادفی را در رشته‌های بیتی انتخاب نمایید (به این نقطه، نقطه‌ی تعویض گفته می‌شود.) و تکّه‌ی دوّم s1 را با تکّه‌ی دوّم s2، برای به وجود آوردن دو رشته بیتی جدید، با هم تعویض نمایید.

توجّه:

قسمت زیر دارای ارتباط بین رنگ‌ها هم هست.

اجرای عمل تعویض برای ۸–وزیر

اگر s1 و s2 دو راه حلّ کد شده‌ی به صورت تصادفی انتخاب شده برای این مسأله باشند:

s1: 100 101 110 000 101 001 010 110 = 4 5 6 0 5 1 2 6
s2: 001 000 101 110 111 010 110 111 = 1 0 5 6 7 2 6 7

مکان هندسی را به صورت تصادفی برابر با ۹ قرار می‌دهیم:

s1: (100 101 110) (000 101 001 010 110) = 4 5 6 0 5 1 2 6
s2: (001 000 101) (110 111 010 110 111) = 1 0 5 6 7 2 6 7

و تکّه‌ی دوّم s1 را با تکّه‌ی دوّم s2 عوض می کنیم، با این کار s3 و s4 به صورت زیر به دست می‌آیند:

s3 = (100 101 110) (110 111 010 110 111) = 4 5 6 6 7 2 6 7
s4 = (001 000 101) (000 101 001 010 110) = 1 0 5 0 5 1 2 6

با این کار عمل تعویض به پایان رسید. (**مشابه کنکور آزاد سال ۸۳**)

نکته:

تعویض روش‌های دیگری هم دارد؛ به عنوان مثال:

10|111|10 1010110

→ تعویض

01|101|01 0111101

پیاده‌سازی عمل تعویض: از یک ماسک[۱] (m) که به صورت رشته‌ی دودویی می‌باشد، استفاده نمایید، برای دو والد h_1 و h_2 ارائه شده داریم:

$$(h_1 \wedge m) \vee (h_2 \wedge \neg m),\ (h_1 \wedge \neg m) \vee (h_2 \wedge m)$$

برای پیاده‌سازی تعویض بالا، از ماسک m=0011100 استفاده می‌نماییم. در اینجا با استفاده از فرمول بالا برّرسی می‌کنیم که آیا این ماسک درست کار می‌کند یا نه؟!

h_1= 1011110

h_2= 0110101

m= 0011100

¬m= 1100011

در ابتدا درست بودن ماسک را برای فرزند دوّم(0111101) برّرسی می‌کنیم:

---

۱ – Mask؛ **یادآوری**– در رشته‌ی کامپیوتر، ماسک، داده‌ای است که برای عملیّات بیتی(bitwise operations) مورد استفاده قرار می‌گیرد. با استفاده از یک ماسک، چند بیت، در یک بایت، کلمه(word: به دو بایت، یک کلمه گویند.) و غیره می‌توانند از صفر به یک، یا از یک به صفر تبدیل شوند. به عنوان مثال، با استفاده از ماسک «00001000» و استفاده از عملگر بیتی «OR»، چهارمین بیت «10010101»، از سمت راست، به یک تبدیل می‌شود (10011101).
(http://en.wikipedia.org/wiki/Mask_(computing))

$h_1 \wedge m$:

$$\begin{array}{cccccccc} & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ \wedge & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{array}$$

$h_2 \wedge \neg m$:

$$\begin{array}{cccccccc} & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ \wedge & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ \hline & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}$$

$(h_1 \wedge m) \vee (h_2 \wedge \neg m)$:

$$\begin{array}{cccccccc} & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \vee & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ \hline & 0 & 1 & 1 & 1 & 1 & 0 & 1 \end{array}$$

که رشته‌ی بیتی به دست آمده(۰۱۱۱۱۰۱)، فرزند دوّم است.

حال درست بودن ماسک را برای فرزند اوّل(10101**10**) برّرسی می‌نماییم:

$h_1 \wedge \neg m$:

$$\begin{array}{cccccccc} & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ \wedge & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ \hline & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}$$

$h_2 \wedge m$:

$$\begin{array}{cccccccc} & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ \wedge & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{array}$$

$(h_1 \wedge \neg m) \vee (h_2 \wedge m)$:

$$\begin{array}{cccccccc} & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ \vee & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ \hline & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

که رشته‌ی بیتی به دست آمده(۱۰۱۰۱۱۰)، فرزند اوّل است.

پس معلوم شد که ماسک درست کار می‌کند.

یا به عنوان روشی دیگر برای تعویض:

1|01|11|10 0010110

→ تعویض

0|11|01|01 1111101

برای پیاده‌سازی تعویض بالا، از ماسک m=1001100 استفاده می‌نماییم. در اینجا برّرسی می کنیم که آیا این ماسک درست کار می کند یا نه؟!

h_1= 1011110

h_2= 0110101

m= 1001100

¬m= 0110011

در ابتدا درست بودن ماسک را برای فرزند دوّم(1111101) برّرسی می کنیم:

$h_1 \wedge m$:

	1	0	1	1	1	1	0
∧	1	0	0	1	1	0	0
	1	0	0	1	1	0	0

$h_2 \wedge \neg m$:

	0	1	1	0	1	0	1
∧	0	1	1	0	0	1	1
	0	1	1	0	0	0	1

$(h_1 \wedge m) \vee (h_2 \wedge \neg m)$:

	1	0	0	1	1	0	0
∨	0	1	1	0	0	0	1
	1	1	1	1	1	0	1

که رشته‌ی بیتی به دست آمده(۱۱۱۱۱۰۱)، فرزند دوّم است.

حال درست بودن ماسک را برای فرزند اوّل(0010110) برّرسی می‌نماییم:

$h_1 \wedge \neg m$:

$$\begin{array}{cccccccc} & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ \wedge & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ \hline & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{array}$$

$h_2 \wedge m$:

$$\begin{array}{cccccccc} & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ \wedge & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}$$

$(h_1 \wedge \neg m) \vee (h_2 \wedge m)$:

$$\begin{array}{cccccccc} & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ \vee & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

که رشته‌ی بیتی به دست آمده(۰۰۱۰۱۱۰)، فرزند اوّل است.

پس معلوم شد که ماسک درست کار می‌کند.

جهش

بعد از انجام عمل تعویض، جهش را اعمال نمایید؛ برای این کار، در رشته‌ی به دست آمده، یکی از عددها را به عدد دیگری تبدیل کنید؛ به عنوان مثال، در رشته‌ی s3، عدد ۲ را به عدد ۱ تبدیل نمایید:

s3 = 4 5 6 6 7 2 6 7 **جهش** → s3 = 4 5 6 6 7 1 6 7

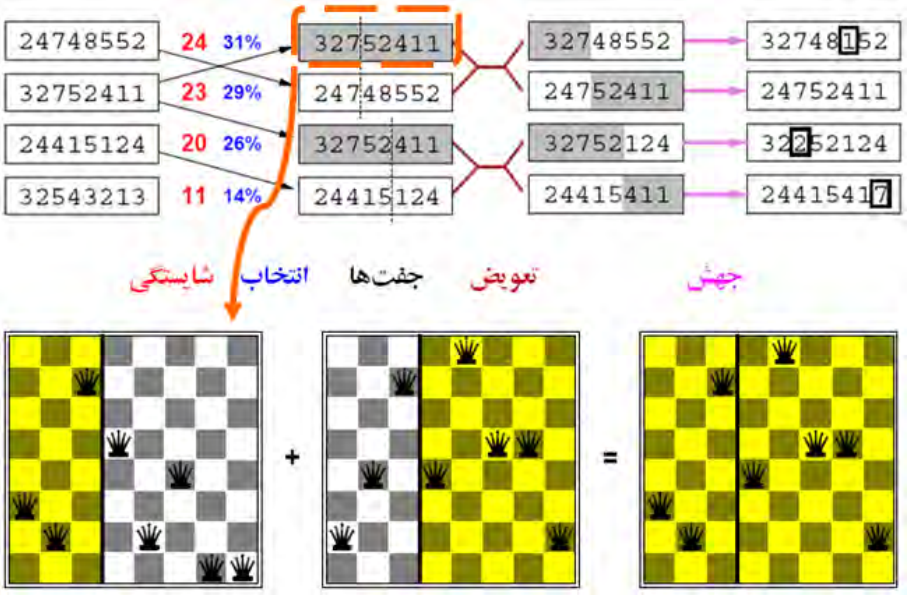
در اینجا اگر بخواهیم تولید راه حل‌های جدید را ادامه بدهیم، باید یک جمعیّت دیگر از راه حل‌ها را به صورت تصادفی تولید کرده و به ابتدای حلقه‌ای که در مطلب با عنوان «یک الگوریتم ژنتیکی پایه» آمده است، برویم.

مثال دیگری برای ۸-وزیر؛

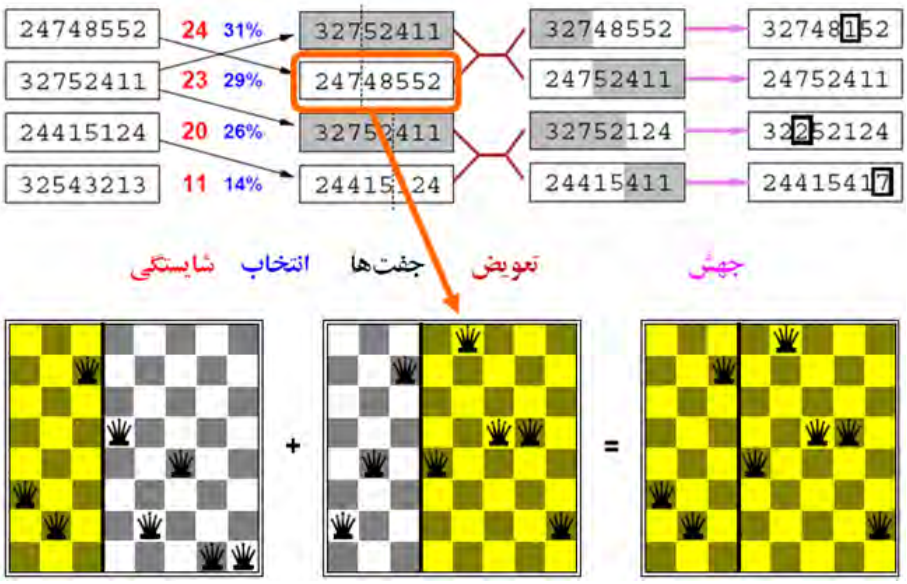
توجّه:

در این مثال توجّه کنید که شماره گذاری ردیف صفحه‌ی ۸-وزیر، از عدد یک و از پایین صفحه آغاز شده است.

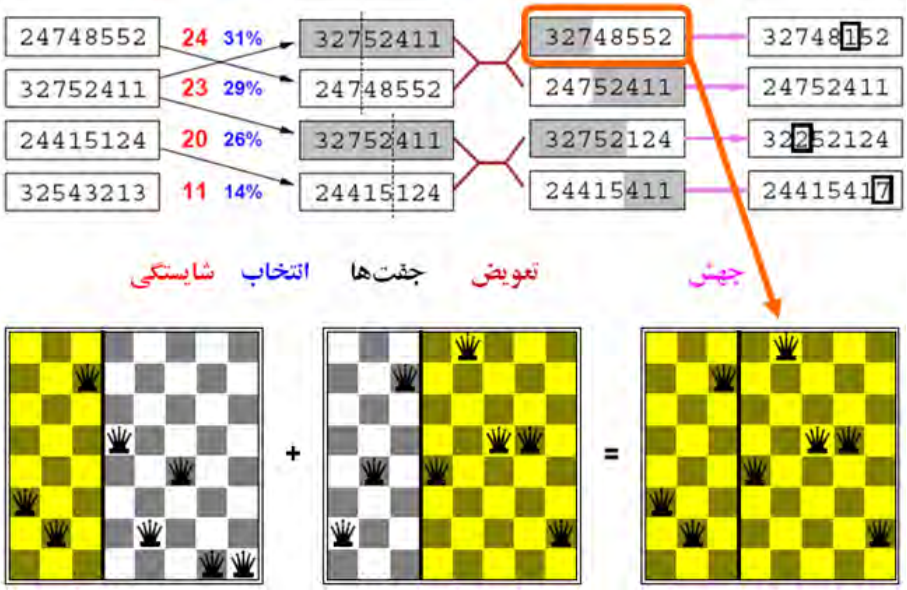
در شکل زیر یکی از راه حل‌ها مشخّص شده است:



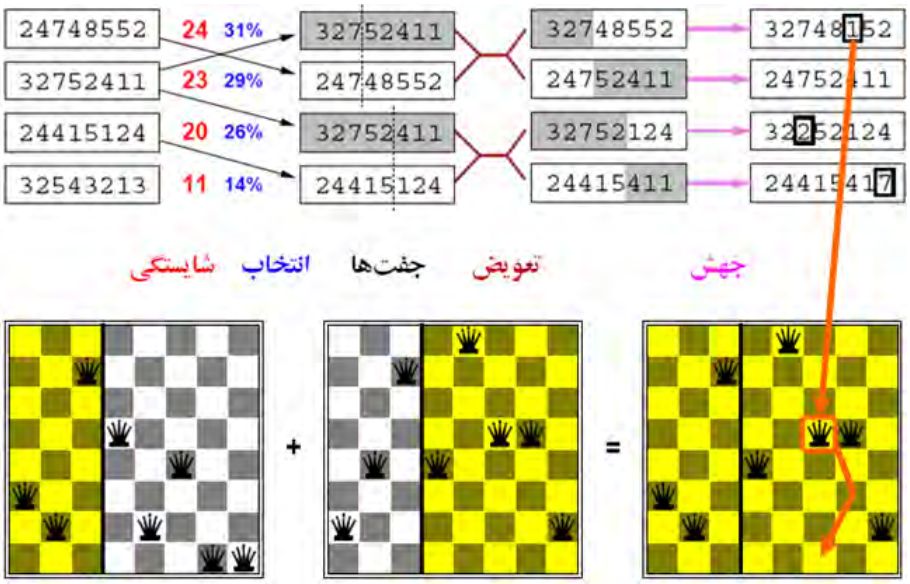
در شکل زیر یکی دیگر از راه حل‌ها که با راه حلّ قبلی می‌خواهیم عمل تعویض را بر روی آنها پیاده‌سازی کنیم، مشخّص شده است:



در شکل زیر راه حلّ اوّل پس از انجام عمل تعویض بر روی آن مشخّص شده است:



در شکل زیر مکان تازه‌ی قرار گرفتن یک وزیر(یعنی، ردیف ۱) پس از انجام عمل جهش بر روی آن مشخّص شده است. توجّه کنید که مکان قبلی قرار گرفتن وزیر، ردیف پنجم بوده است:



توجّه:

پایان قسمت دارای ارتباط بین رنگ‌ها!

الگوریتم‌های ژنتیکی به صورت جستجو

می‌توانیم الگوریتم‌های ژنتیکی را به صورت یک مسأله‌ی جستجو ببینیم؛ به این صورت که وضعیّت‌ها، راه حل‌های ممکن هستند؛ عملگرهای جستجو، جهش، تعویض و انتخاب هستند. جهش و تعویض باید به ما اجازه دهند که از ماکزیمم محلّی خارج شویم (شبیه روش جستجوی شبیه‌سازی گرم و سرد کردن می‌باشند).

کاربردهای الگوریتم ژنتیکی

الگوریتم‌های ژنتیکی اغلب برای مسأله‌های بهینه‌سازی مورد استفاده قرار می‌گیرند، مثل: آرایش مدار[۱]، طرّاحی سیستم و زمانبندی.

به طور خلاصه باید بگوییم که الگوریتم‌های ژنتیکی به اندازه‌ی کافی برای پیداکردن راه حل، مناسب می‌باشند؛ ولی بهبود قابل توجّهی در مورد چند نسل ندارند و دارای محدودیّت زمانی هستند.

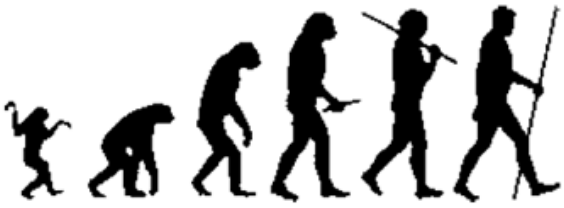
۱ - circuit layout

چکیده‌ی مطلب‌های فصل ششم

الگوریتم‌های تپّه‌نوردی فقط وضعیّت جاری را در حافظه نگه می‌دارند، امّا ممکن است در بیشینه‌ی محلّی گیر بیفتند.

روش شبیه‌سازی گرم و سرد کردن، از بیشینه‌ی محلّی می‌گریزد و در صورت داشتن یک زمانبندی خنک کننده‌ی به اندازه‌ی کافی طولانی، کامل و بهینه است.

الگوریتم‌های ژنتیکی می‌توانند فضایی بزرگ را با الهام گرفتن از تئوری تکامل زیستی جستجو کنند. عکس زیر نمایشگر این تئوری است:



یادآوری یا تکمیل مطلب‌های فصل ششم

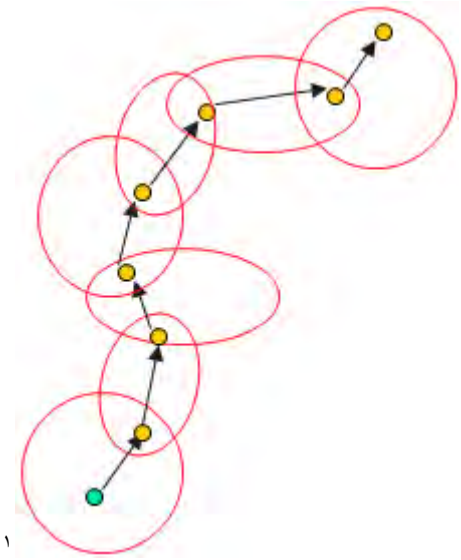
سؤال– ایده‌ی اساسی جستجوی محلّی را به طور خیلی ساده بیان کنید.

جواب–

۱– یک وضعیّت اوّلیّه را به طور تصادفی انتخاب کنید.

۲– اصلاحِ(تغییرِ) محلّی را برای بهبود وضعیّت جاری انجام دهید(وضعیّت جاری را برّرسی کنید و به دیگر وضعیّت‌ها بروید).

۳– مرحله‌ی ۲ را تا زمانی که وضعیّت هدف پیدا شود یا از محدوده‌ی زمانی خارج شوید، تکرار کنید.



۱

۱ – مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

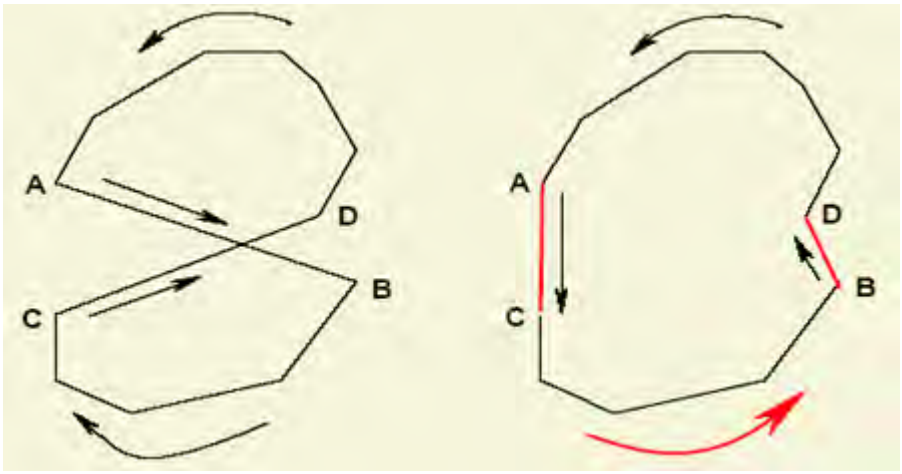
مطلب- مسأله‌ی مسافرت فروشنده‌ی دوره‌گرد: کوتاه‌ترین گشت(تور) را طوری که از همه‌ی شهرها یکبار عبور کند، پیدا کنید.

یک راه، جستجوی کامل (تولید و تست) است؛ که در این صورت، تعداد تمام گشت‌ها، در حدود $(n-1)!/2$ است، که n، تعداد شهرهاست؛ مثلاً اگر $n=36$ باشد، برابر با عدد زیر است:

566573983193072464833325668761600000000

که امکان‌پذیر نیست!.

یک راه حلّ دیگر، شروع از یک راه حلّ اوّلیّه و بهبود آن با استفاده از تغییر محلّی است؛



[1]

درست یا غلط- الگوریتم‌های جستجوی محلّی معمولاً فقط روی یک یا تعداد اندکی گره‌ی جاری عمل می‌کنند.

جواب- «درست» است.[2]

درست یا غلط- الگوریتم‌های جستجوی محلّی معمولاً برای پیدا کردن راه حلّ بهینه‌ی سراسری(مطلق) استفاده می‌شوند.

جواب- «غلط» است.[3]

سؤال- جستجوی تپّه‌نوردی چه هنگام خاتمه می‌یابد؟

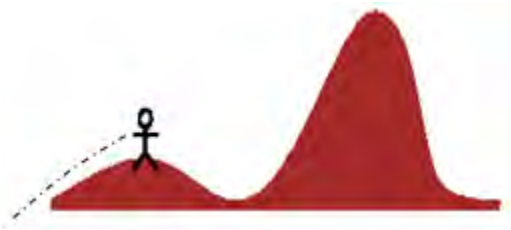
جواب- وقتی که به یک قلّه برسد، به شکل زیر توجّه کنید!.[1]

---

۱ - مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

۲ - کوییز شماره‌ی ۱ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، پاییز سال ۲۰۱۲ میلادی

۳ - کوییز شماره‌ی ۱ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، پاییز سال ۲۰۱۲ میلادی



سؤال- اشکال روش تپّه‌نوردی چیست؟

جواب- بیشینه/کمینه‌ی محلّی: جستجوی محلّی می‌تواند در بیشینه/کمینه‌ی محلّی گیر بیفتد و راه حلّ بهینه را پیدا نکند.[۲]

درست یا غلط- ضعف اصلی تپّه‌نوردی این است که فقط برای کار در فضاهای جستجوی دوبعدی ضمانت شده است.

جواب- «غلط» است.[۳]

درست یا غلط- تپّه‌نوردی به بهترین جانشین وضعیّت جاری حرکت می‌کند.

جواب- «درست» است.[۴]

درست یا غلط- جستجوی تپّه‌نوردی هیچوقت بیشینه‌ی(ماکزیمم) مطلق را پیدا نمی‌کند.

جواب- «غلط» است؛ گاهی اوقات آن را پیدا می‌کند، در اصل این مورد وابسته به وضعیّت شروع است.[۵]

سؤال- در شکل زیر اگر گره‌ی S، گره‌ی شروع و گره‌ی G، گره‌ی هدف باشد و از روش تپّه‌نوردی استفاده کنیم، حاصل چه خواهد بود؟

---

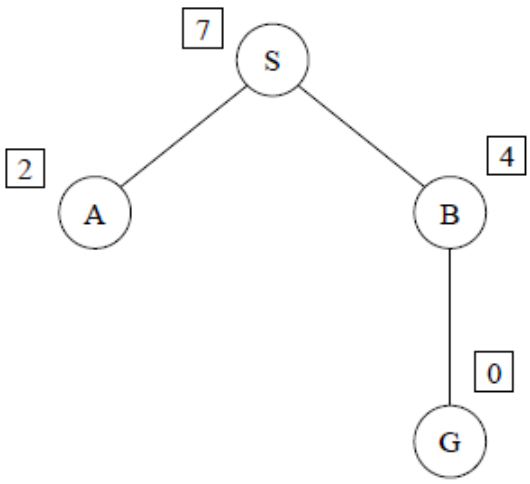
۱ - مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

۲ - مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

۳ - آزمون میان‌ترم درس «مبانی هوش مصنوعی» استاد، «تیم فینین(Tim Finin)»، دانشکده‌ی مهندسی برق و علوم کامپیوتر دانشگاه شهر بالتیمُور ایالت مریلَند کشور آمریکا، ۱۹ اکتبر سال ۲۰۰۹ میلادی

۴ - کوییز شماره‌ی ۱ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، پاییز سال ۲۰۱۲ میلادی

۵ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۱۴ جولای سال ۲۰۱۱ میلادی



جواب– در گره‌ی A، که کمینه‌ی محلّی است، گیر می‌افتیم.[۱]

درست یا غلط– روش شبیه‌سازی گرم و سرد کردن، یک الگوریتم جستجوی ناآگاهانه است.

جواب– **«غلط» است؛** از دانش دامنه استفاده می‌کند.[۲]

درست یا غلط– الگوریتم روش شبیه‌سازی گرم و سرد کردن، حرکت‌های بد را نمی‌پذیرد.

جواب– **«غلط» است؛** حرکت‌های بد پذیرفته می‌شوند، امّا با توجّه به یک زمانبندی گرم و سرد کردن، احتمال پذیرش حرکت‌های بد کاهش می‌یابد.[۳]

درست یا غلط– در شبیه‌سازی گرم و سرد کردن، احتمال پذیرش حرکت بد، در انتها، بیش‌تر از قبل‌تر است.

جواب– **«غلط» است.**[۴]

درست یا غلط– روش شبیه‌سازی گرم و سرد کردن از یک «زمانبندی گرم و سرد کردن» برای انتخاب وضعیّت بعدی استفاده می‌کند.

جواب– **«درست» است.**[۱]

---

۱ – نمونه سؤالات آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «دیمیتری پاولوف»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، زمستان سال ۲۰۰۱ میلادی

۲ – آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۱۱ ژانویه‌ی سال ۲۰۱۰ میلادی

۳ – آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۱۱ ژانویه‌ی سال ۲۰۱۰ میلادی

۴ – کوییز شماره‌ی ۱ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، پاییز سال ۲۰۱۲ میلادی

درست یا غلط- جستجوی پرتو محلّی، k تا از بهترین جانشین‌های k وضعیّتِ در گام قبل را نگه می‌دارد.

جواب- «درست» است.[۲]

درست یا غلط- الگوریتم‌های ژنتیکی، در جستجو برای یک راه حل، با استفاده از مراحل کاملاً قطعی جلو می‌روند.

جواب- «غلط» است؛ مثلاً جهش‌ها.[۳]

درست یا غلط- در الگوریتم‌های ژنتیکی، وضعیّت بعدی معمولاً براساس دو وضعیّت والد محاسبه می‌شود.

جواب- «درست» است؛ تعویض، از دو والد استفاده می‌کند.[۴]

سؤال- تصوّر کنید که شما در حال استفاده از یک الگوریتم ژنتیکی هستید و دو فرد ۱۳۲۴۴۲۱ و ۲۷۵۱۴۲۱ را که به صورت رشته‌هایی از اعداد صحیح ارائه شده‌اند، دارید، در این صورت، نتیجه‌ی اجرای عمل تعویض را میان رقم سوّم و چهارم نشان دهید.

جواب-

1324421 = (132)(4421) = (132)(1421) = 1321421
2751421 = (275)(1421) = (275)(4421) = 2754421[۵]

تست- کدام گزینه جستجو با استفاده از الگوریتم‌های ژنتیکی را شرح می‌دهد؟
۱) جستجو با انتخاب یک عمل که وضعیّت جاری را بهبود می‌دهد.
۲) جستجو با استفاده از عملکردهای تصادفی، با توجّه به کاهش دما.
۳) جستجو با استفاده از عاملی که از مکاشفه استفاده می‌کند.
۴) جستجو با استفاده از یک جمعیّت از وضعیّت‌ها با توجّه به انتخاب تکاملی برای یافتن وضعیّت‌های بهبود یافته.

جواب- گزینه‌ی «۴» درست است.[۶]

---

۱ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۱۴ جولای سال ۲۰۱۱ میلادی

۲ - کوییز شماره‌ی ۱ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، پاییز سال ۲۰۱۲ میلادی

۳ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۳۱ ژانویّه‌ی سال ۲۰۱۱ میلادی

۴ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۷ ژوئن سال ۲۰۱۲ میلادی

۵ - نمونه سؤالات آزمون پایان‌ترم درس «هوش مصنوعی» دانشگاه McGill کشور کانادا، بهار سال ۲۰۱۰ میلادی

۶ - آزمون میان‌ترم درس «هوش مصنوعی» استاد، «روت شولتز»، دانشکده‌ی مهندسی برق و فنّآوری اطلاعات دانشگاه کوئینزلند کشور استرالیا، سال ۲۰۱۲ میلادی

سؤال- جستجوی پرتو محلّی با $k=1$ به چه جستجویی تبدیل می‌شود؟

جواب- تپّه‌نوردی[۱]

تست- کدامیک از الگوریتم‌های جستجوی زیر راه حلّ بهینه را پیدا می‌کند؟

۱) اوّل سطح

۲) اوّل عمق

۳) تپّه‌نوردی

۴) جستجوی حریصانه

جواب- گزینه‌ی «۱» درست است.[۲]

تست- در کدامیک از روش‌های زیر از انتخاب‌های تصادفی برای جلوگیری از گرفتار شدن در ماکزیمم محلّی استفاده می‌شود؟

۱) اوّل سطح

۲) اوّل عمق

۳) شبیه‌سازی گرم و سرد کردن

۴) جستجوی حریصانه

جواب- گزینه‌ی «۳» درست است.[۳]

تست- کدامیک از الگوریتم‌های جستجوی زیر از یک فرموله‌سازی حالت کامل اوّلیّه استفاده می‌کند؟

۱) اوّل عمق

۲) تپّه‌نوردی

۳) A^*

۴) جستجوی حریصانه

جواب- گزینه‌ی «۲» درست است.[۴]

مثالی تصویری برای روش تپّه‌نوردی:

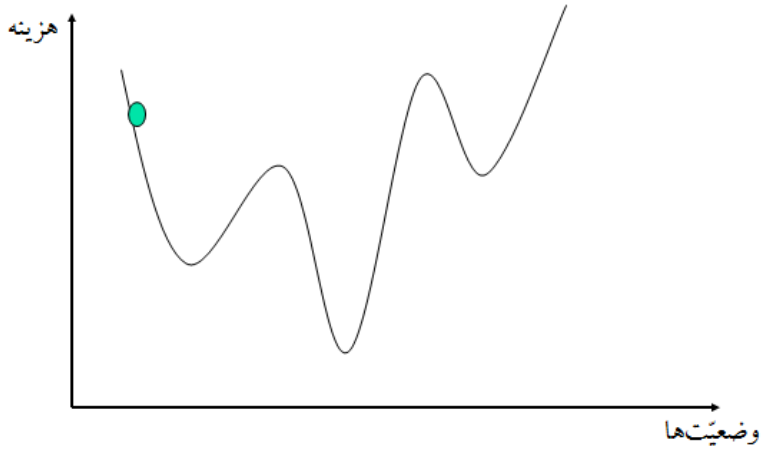
---

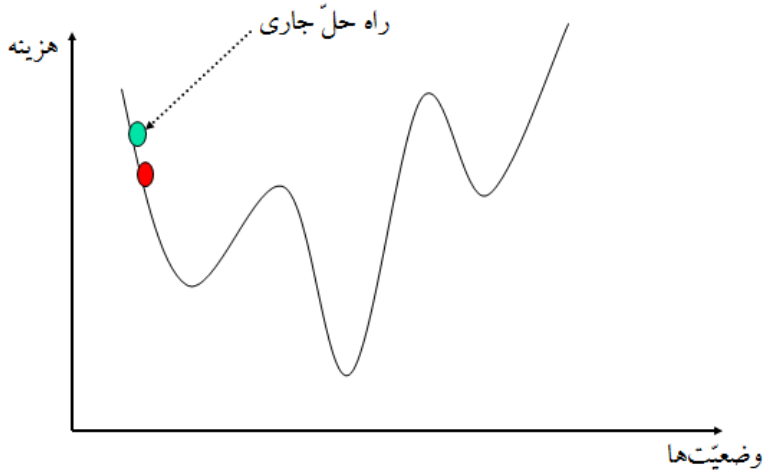
۱ - تکلیف‌های خانگی شماره‌ی ۱ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا

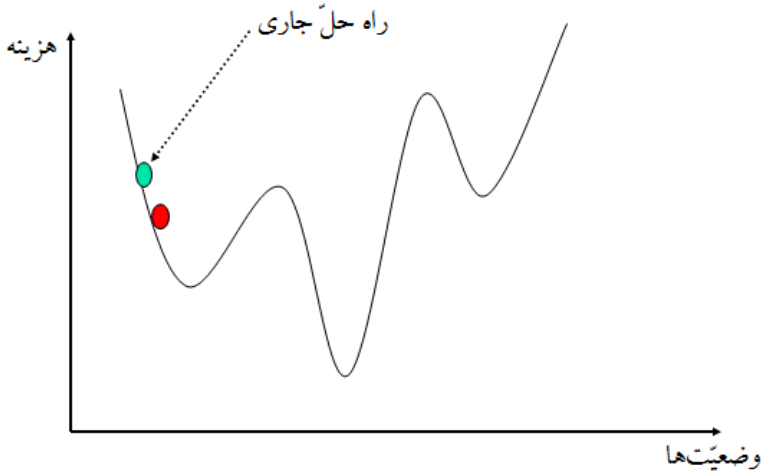
۲ - آزمون میان‌ترم درس «هوش مصنوعی» استاد، «کتی مک‌کون»، دانشکده‌ی علوم کامپیوتر دانشگاه کلمبیای کشور آمریکا، ۲۶ اکتبر سال ۲۰۰۶ میلادی

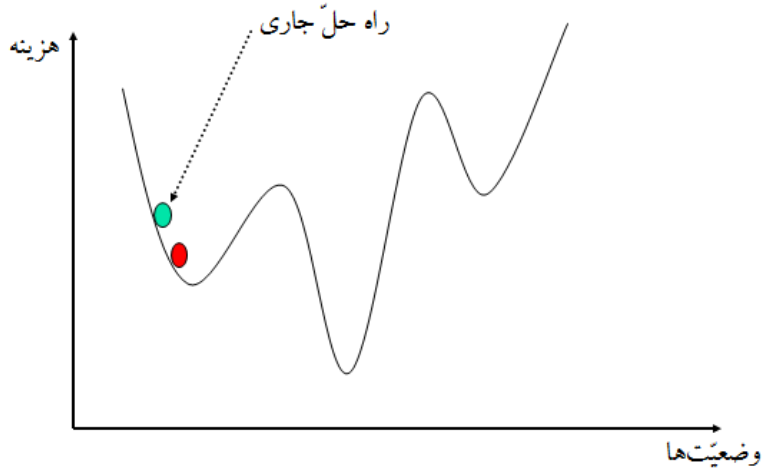
۳ - آزمون میان‌ترم درس «هوش مصنوعی» استاد، «کتی مک‌کون»، دانشکده‌ی علوم کامپیوتر دانشگاه کلمبیای کشور آمریکا، ۲۶ اکتبر سال ۲۰۰۶ میلادی

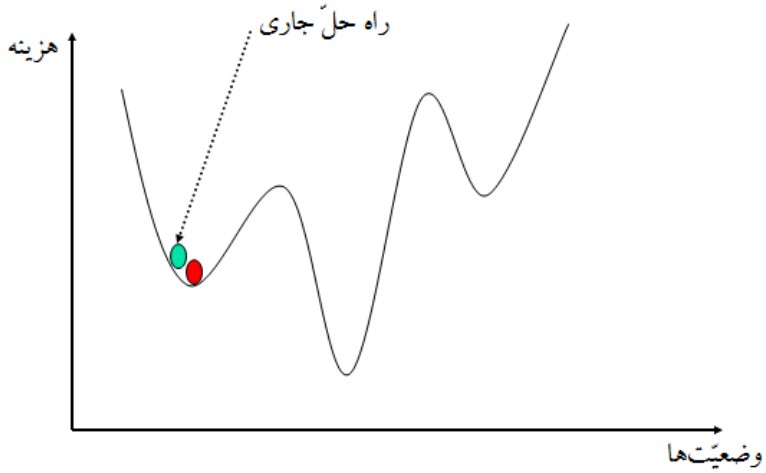
۴ - آزمون میان‌ترم درس «هوش مصنوعی» استاد، «کتی مک‌کون»، دانشکده‌ی علوم کامپیوتر دانشگاه کلمبیای کشور آمریکا، ۲۹ اکتبر سال ۲۰۰۷ میلادی

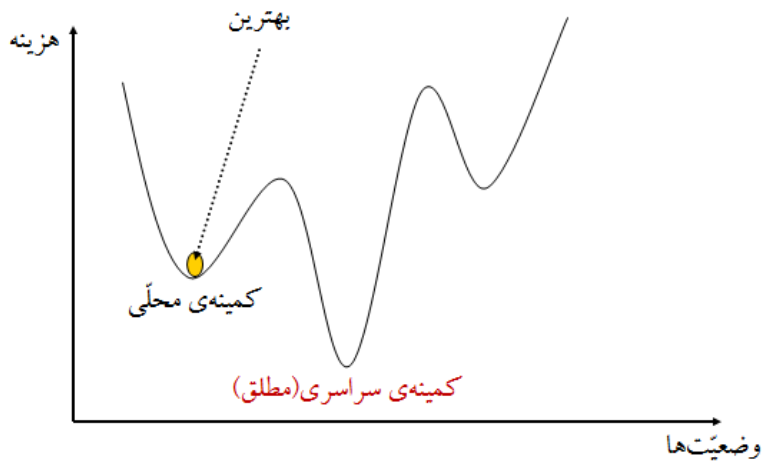

هزینه
وضعیّت‌ها


راه حلّ جاری
هزینه
وضعیّت‌ها


راه حلّ جاری
هزینه
وضعیّت‌ها

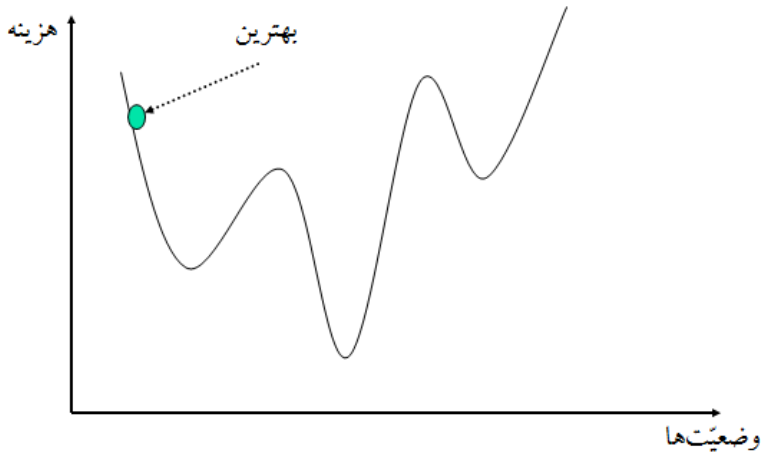


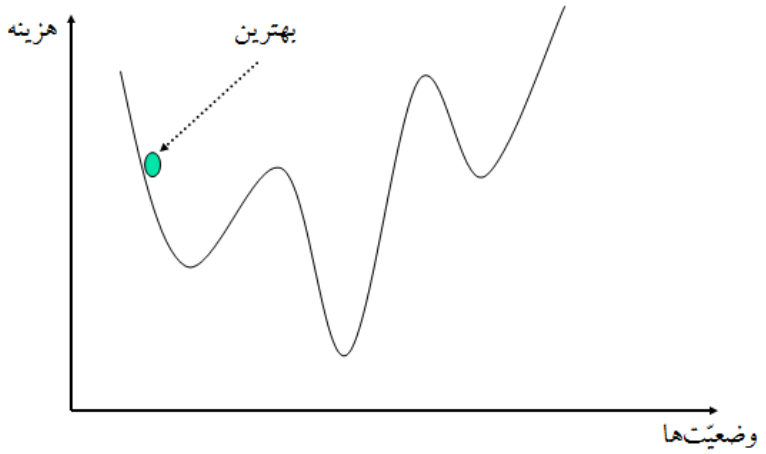




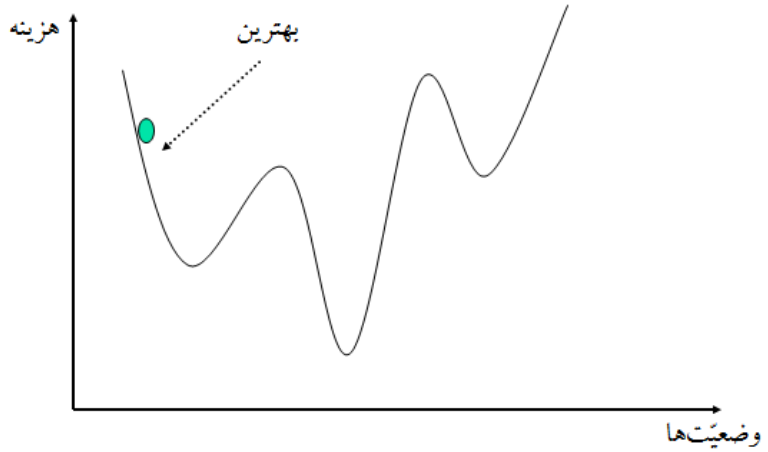
توجّه نمائید که هدف، پیدا کردن بهینه(بیشینه یا کمینه)ی سراسری است:

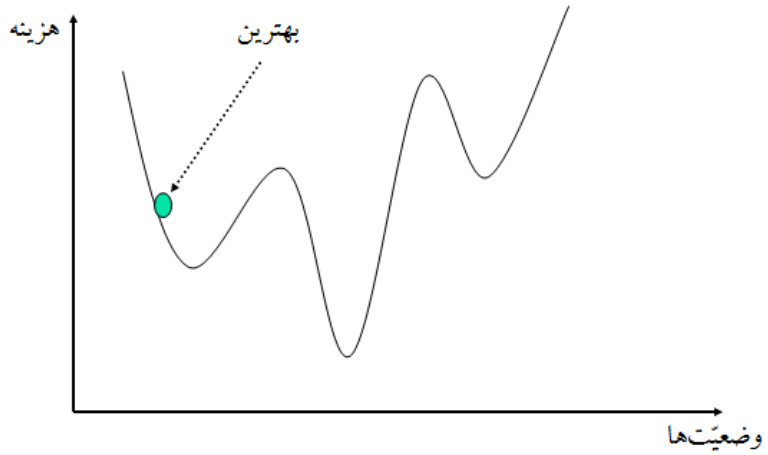
مثالی تصویری برای روش شبیه‌سازی گرم و سرد کردن:

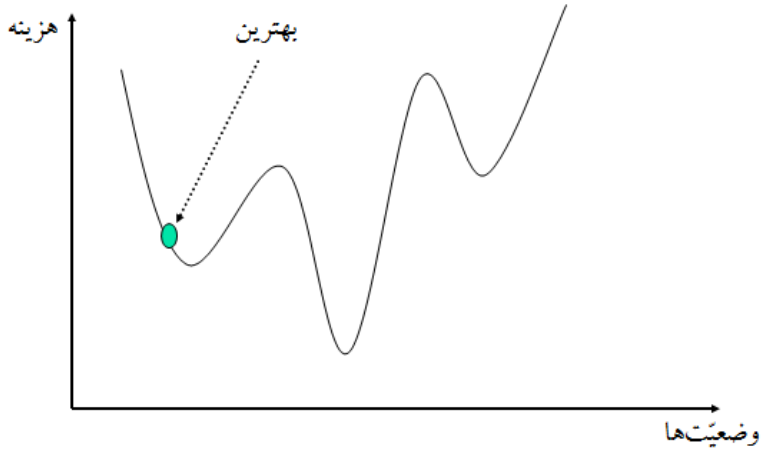


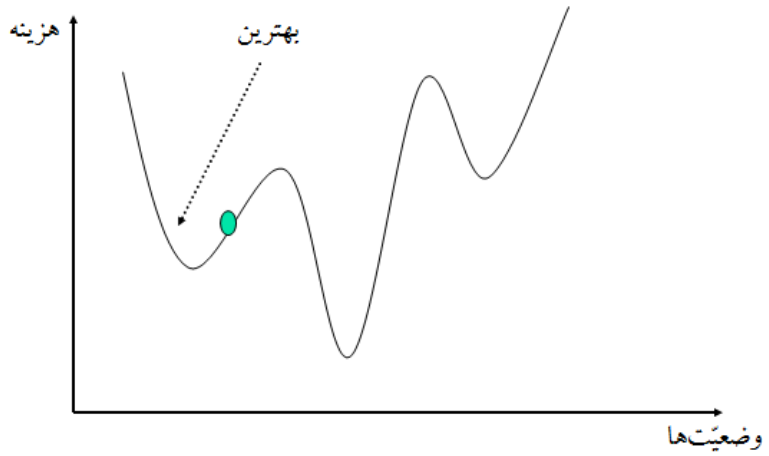


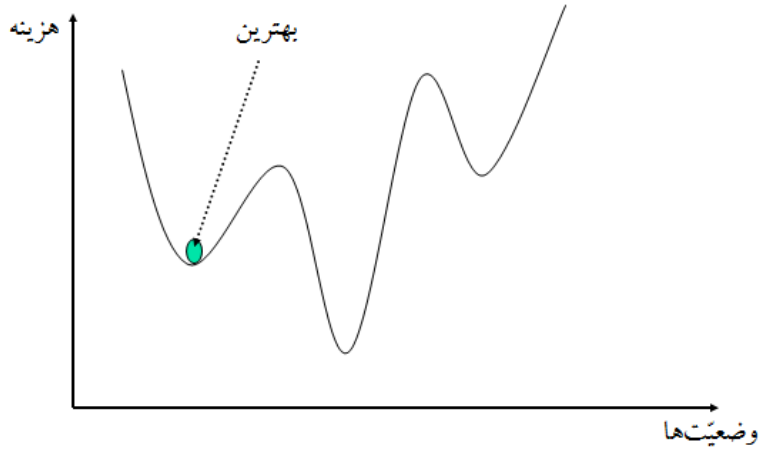
۱ – مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

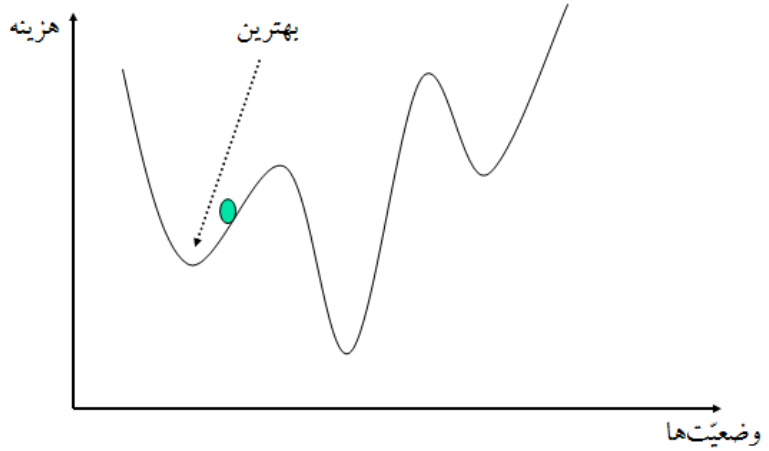

هزینه
بهترین
وضعیّت‌ها

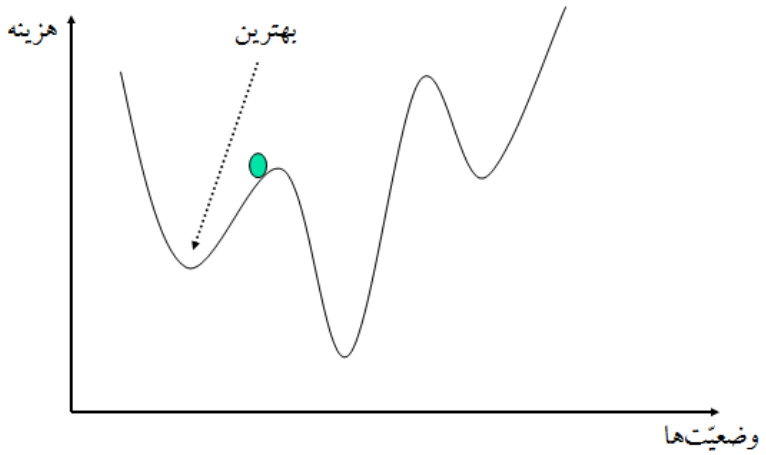

هزینه
بهترین
وضعیّت‌ها

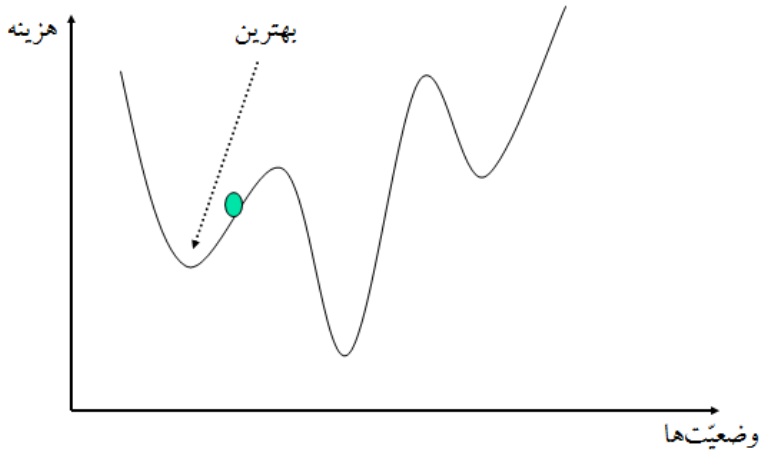

هزینه
بهترین
وضعیّت‌ها

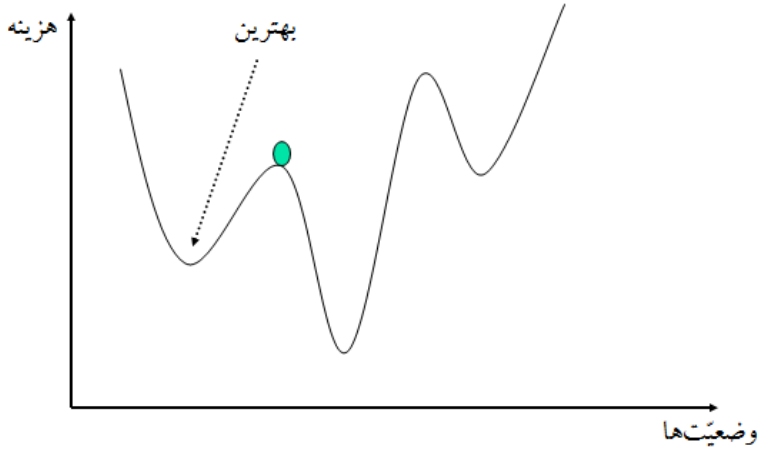

هزینه
بهترین
وضعیّت‌ها

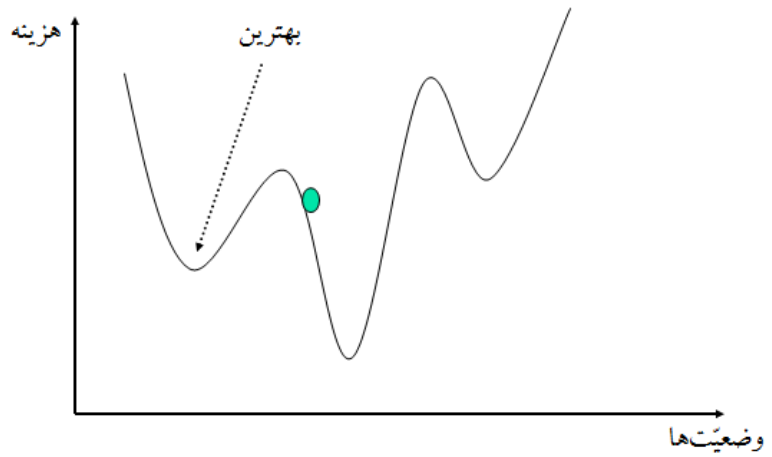

هزینه
بهترین
وضعیّت‌ها

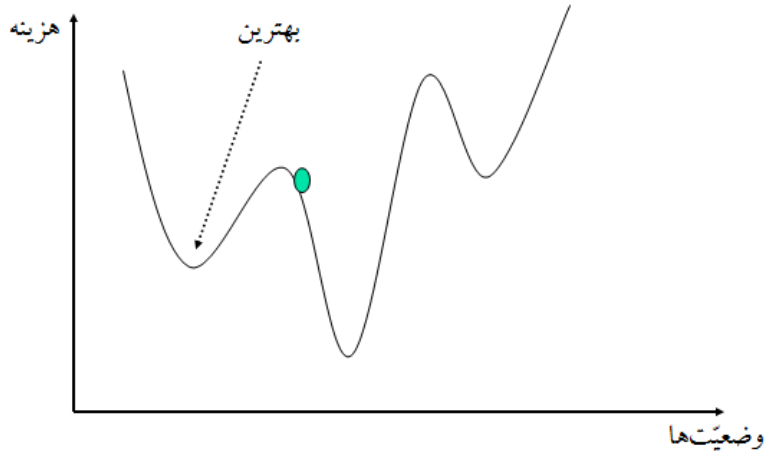

هزینه
بهترین
وضعیّت‌ها

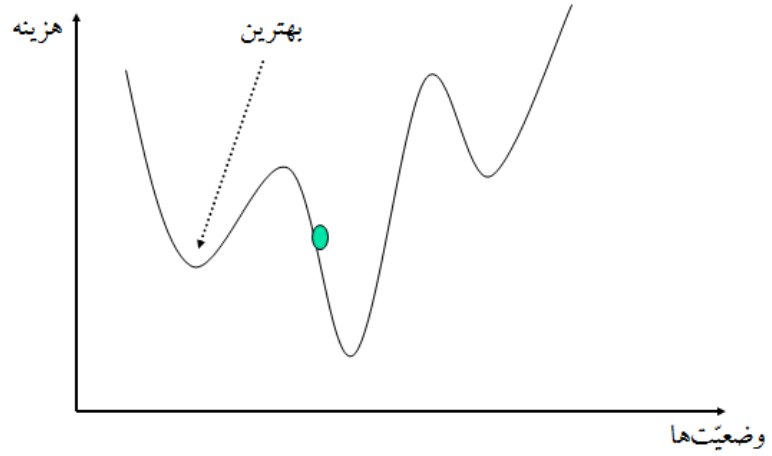

هزینه
بهترین
وضعیّت‌ها

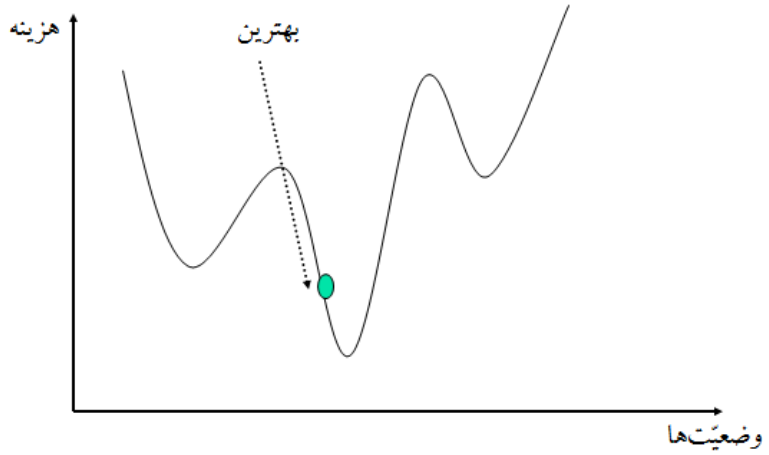

هزینه
بهترین
وضعیّت‌ها

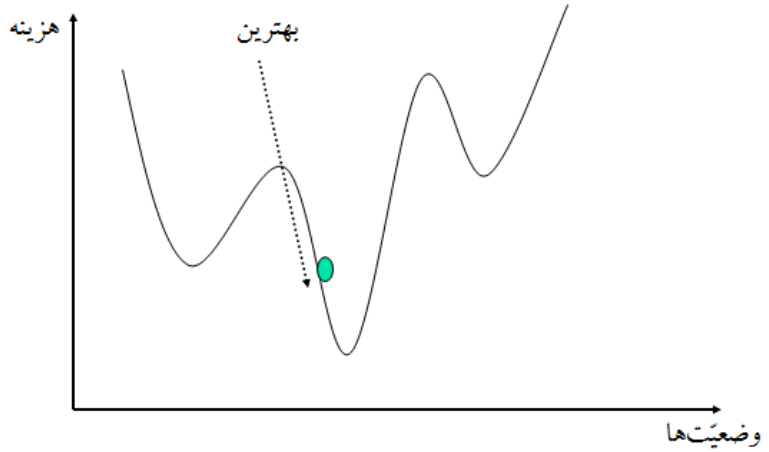

هزینه
بهترین
وضعیّت‌ها

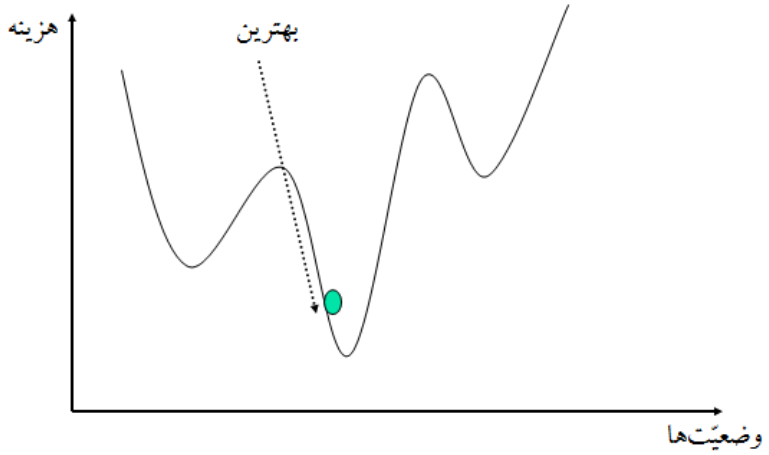

هزینه
بهترین
وضعیّت‌ها

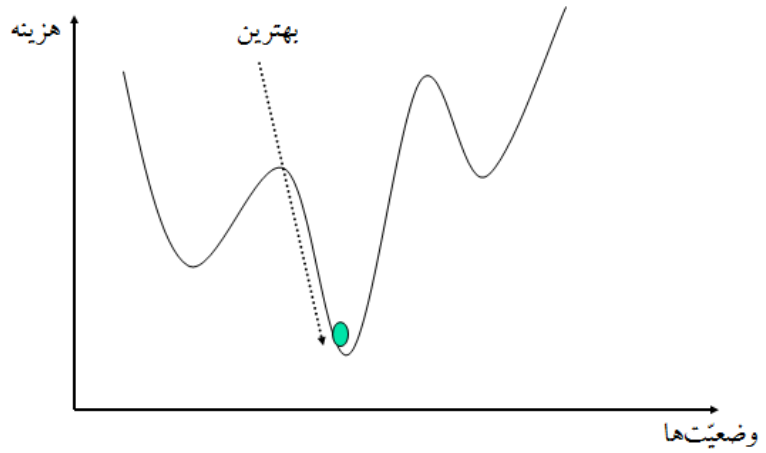

هزینه
بهترین
وضعیّت‌ها

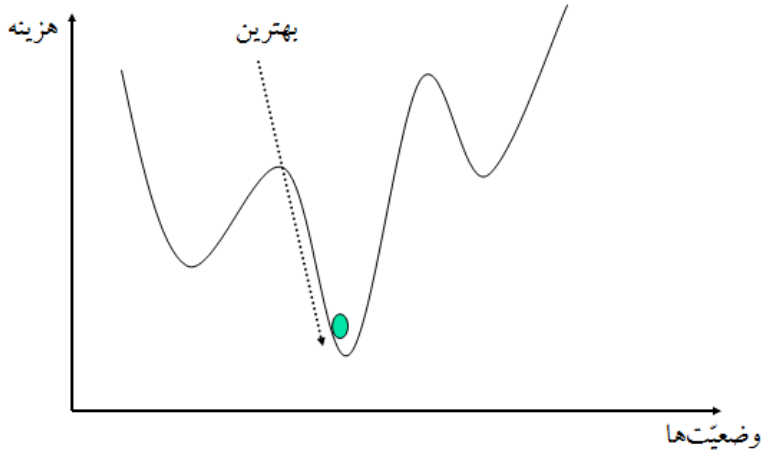

هزینه
بهترین
وضعیّت‌ها

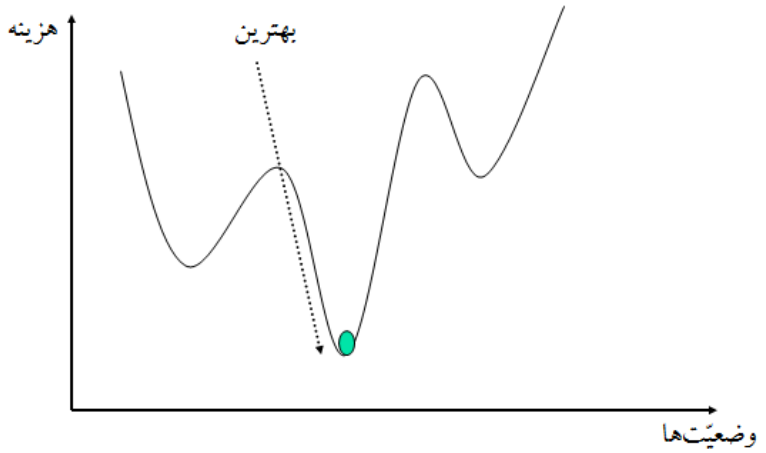

هزینه
بهترین
وضعیّت‌ها

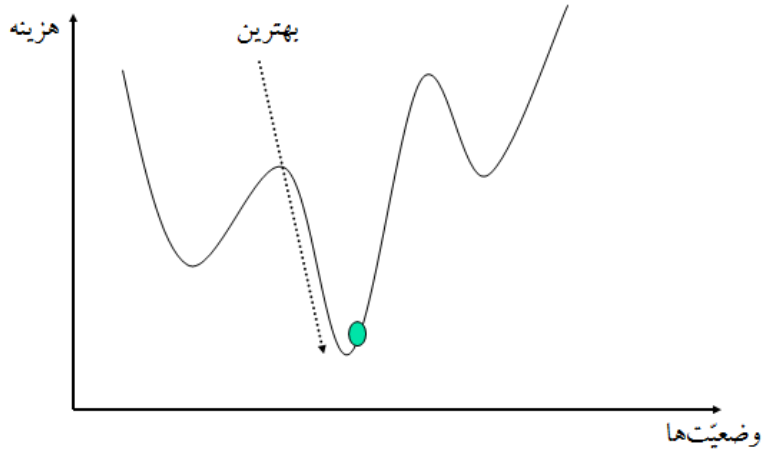

هزینه
بهترین
وضعیّت‌ها

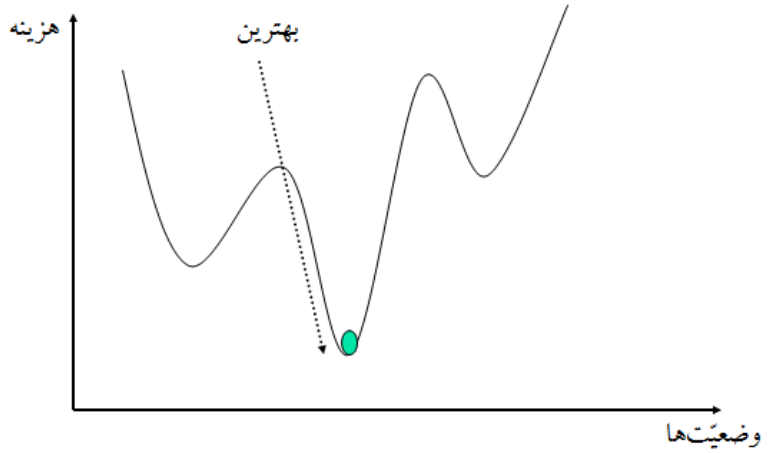

هزینه
بهترین
وضعیّت‌ها

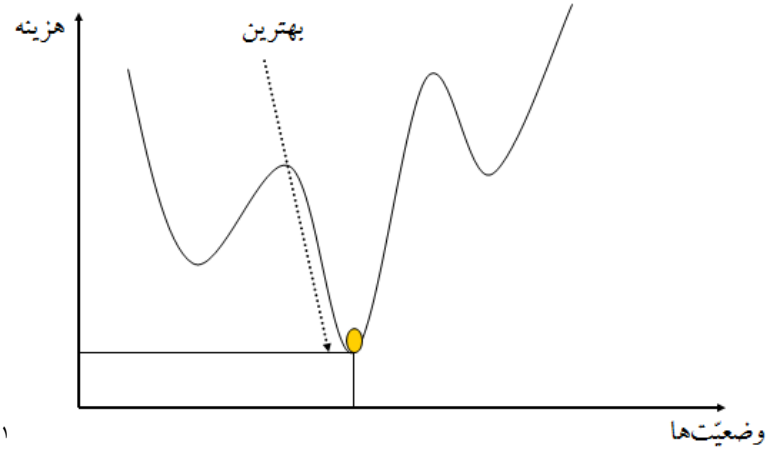

هزینه
بهترین
وضعیّت‌ها


هزینه
بهترین
وضعیّت‌ها


هزینه
بهترین
وضعیّت‌ها



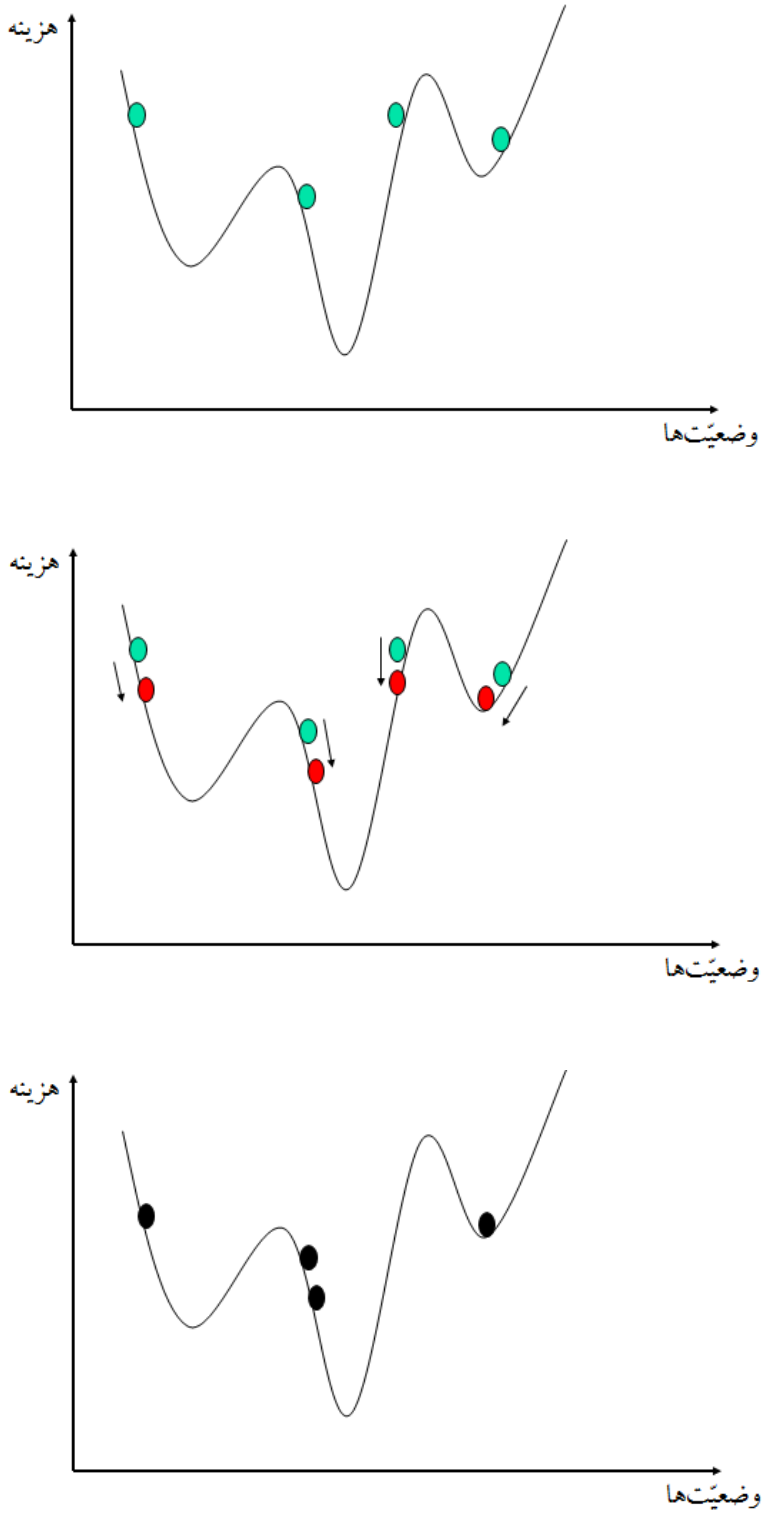




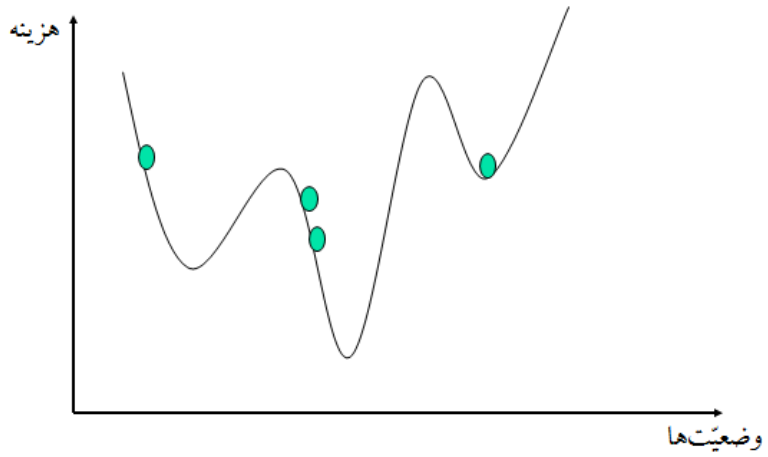
۱

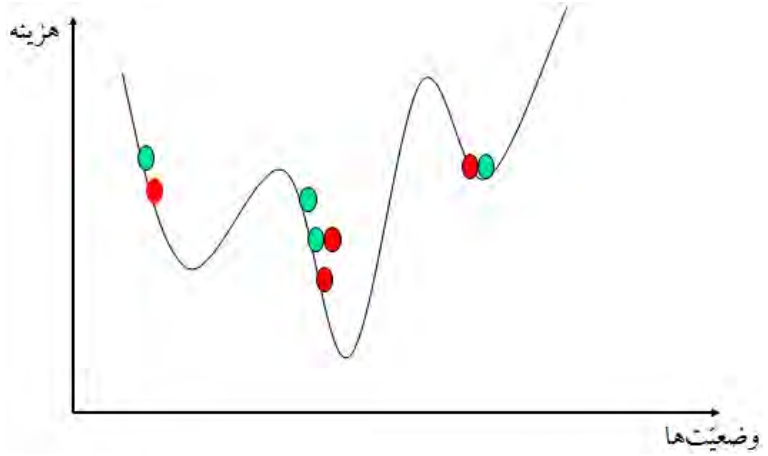
۱ – مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

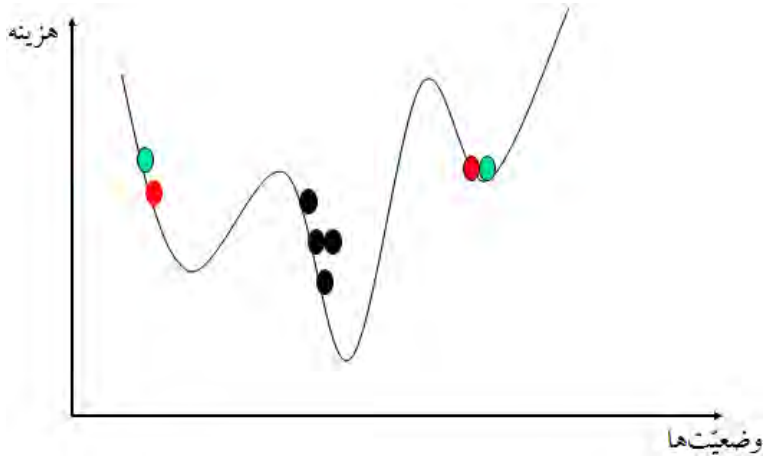
مثالی تصویری برای روش جستجوی پرتو محلّی

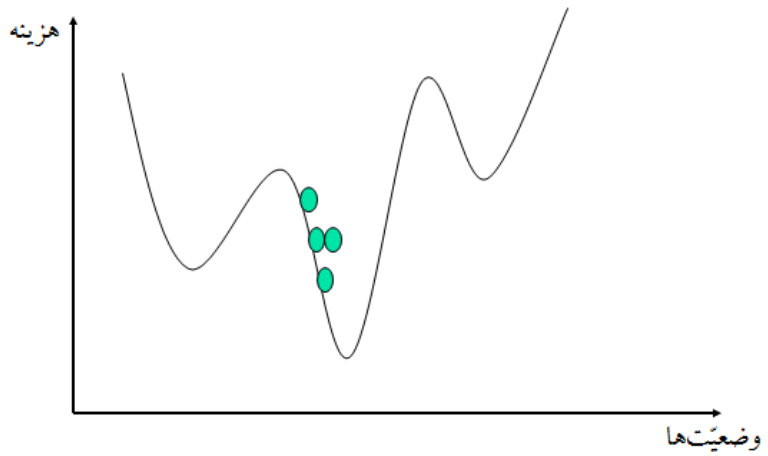


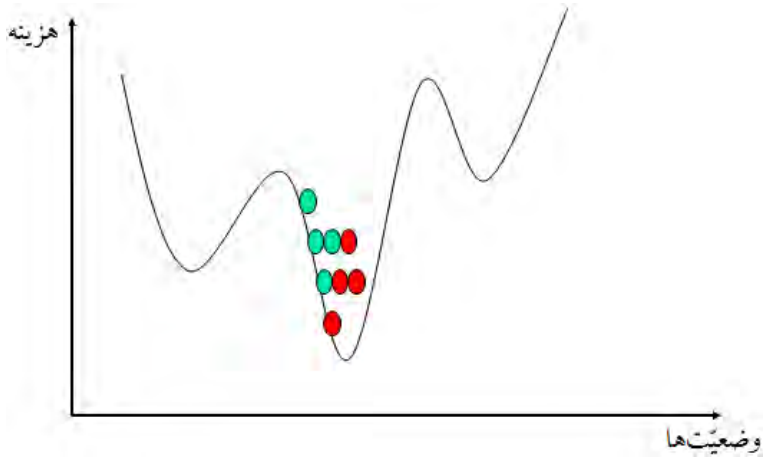
در این شکل‌ها (مثل شکل بالا) توجّه کنید که وضعیّت‌هایی که به رنگ سیاه نشان داده شده‌اند، وضعیّت‌هایی هستند که به عنوان بهترین انتخاب شده‌اند.

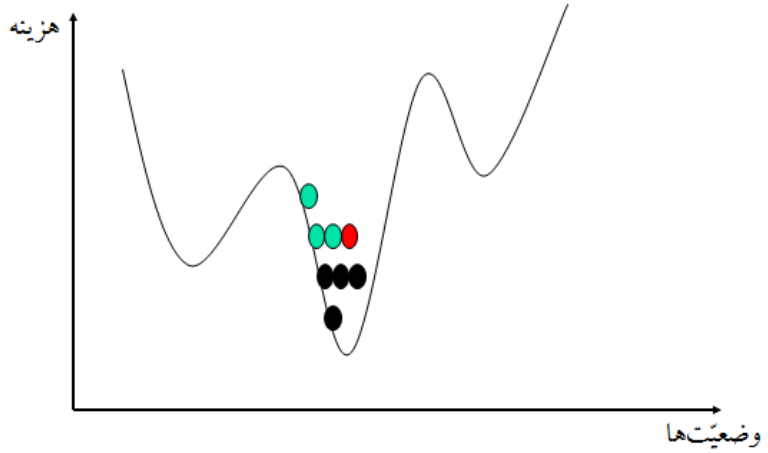

هزینه
وضعیّت‌ها

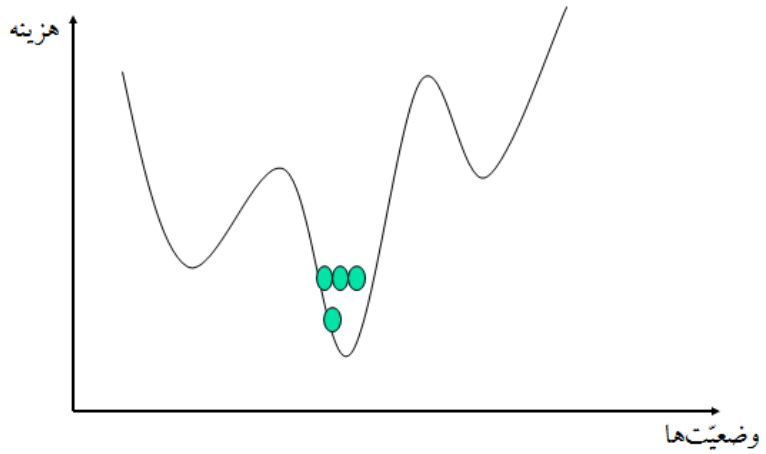

هزینه
وضعیّت‌ها

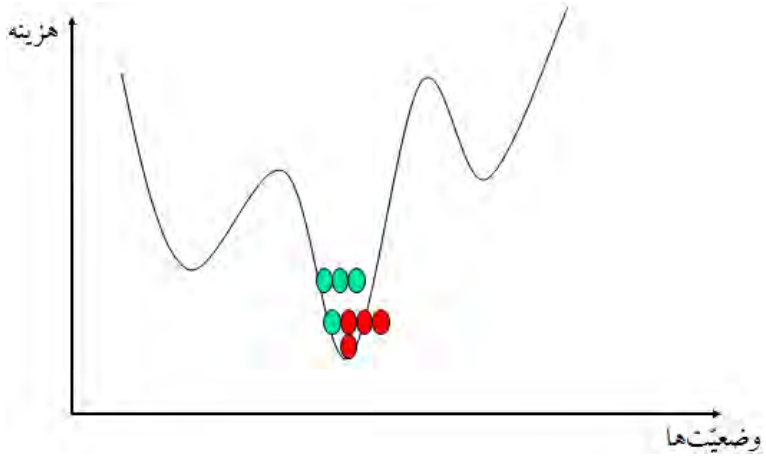

هزینه
وضعیّت‌ها

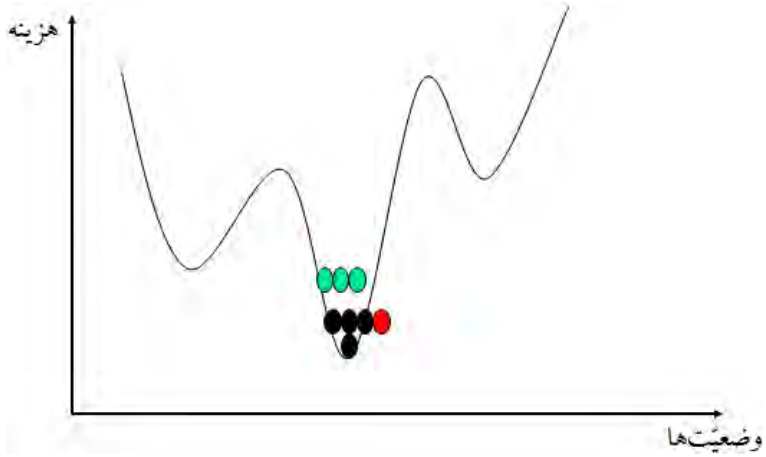

هزینه
وضعیّت‌ها

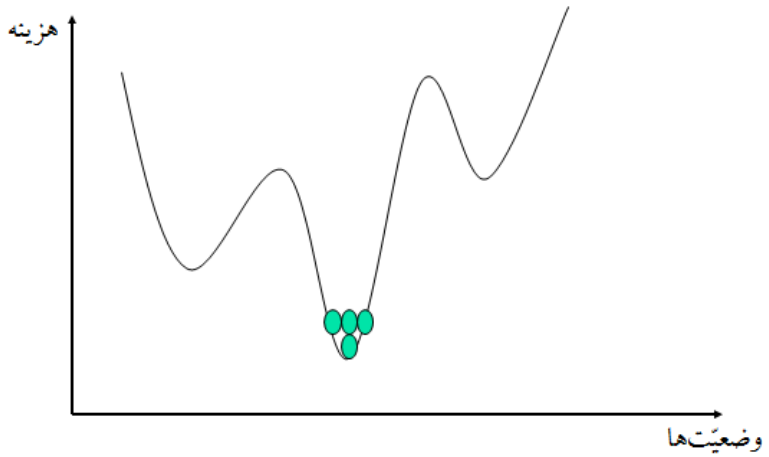

هزینه
وضعیّت‌ها

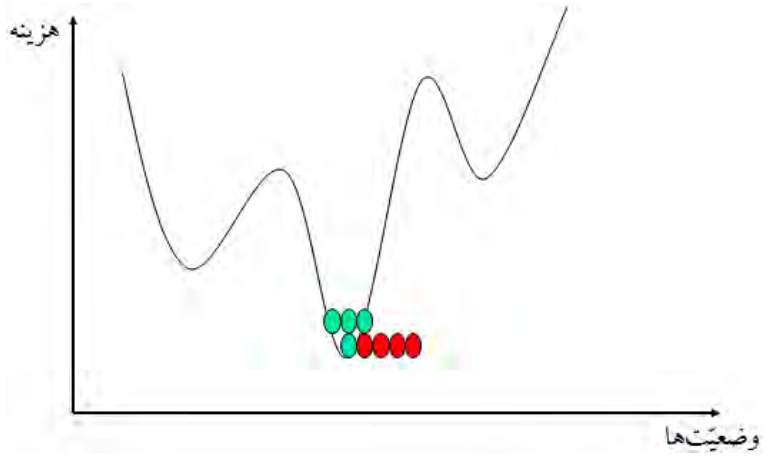

هزینه
وضعیّت‌ها

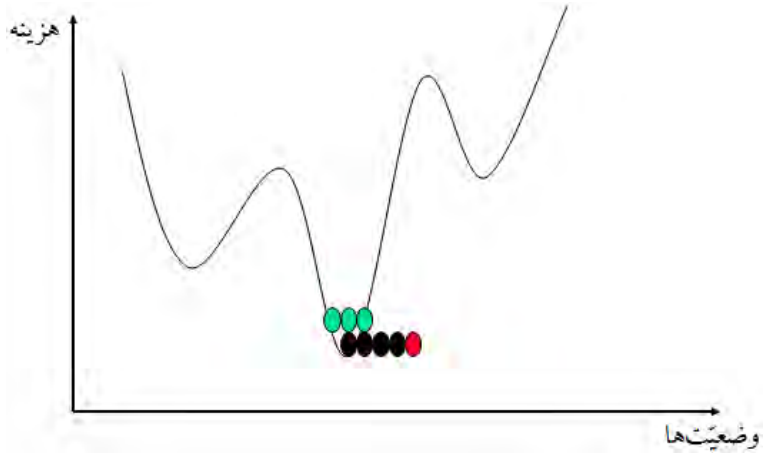

هزینه
وضعیّت‌ها

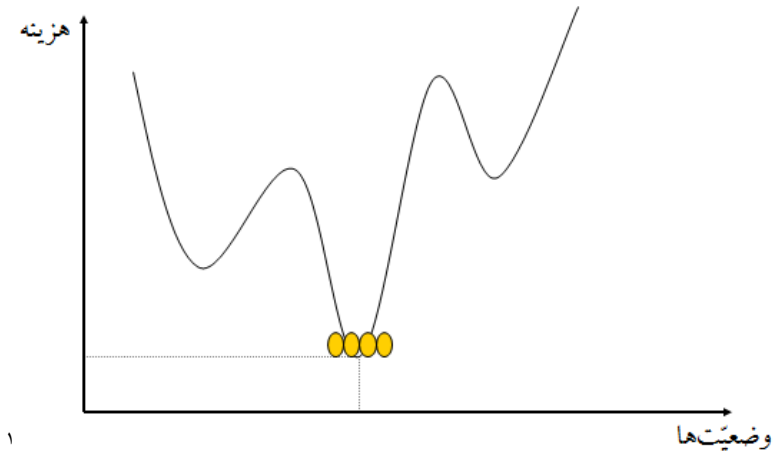

هزینه
وضعیّت‌ها


هزینه
وضعیّت‌ها


هزینه
وضعیّت‌ها


هزینه
وضعیّت‌ها

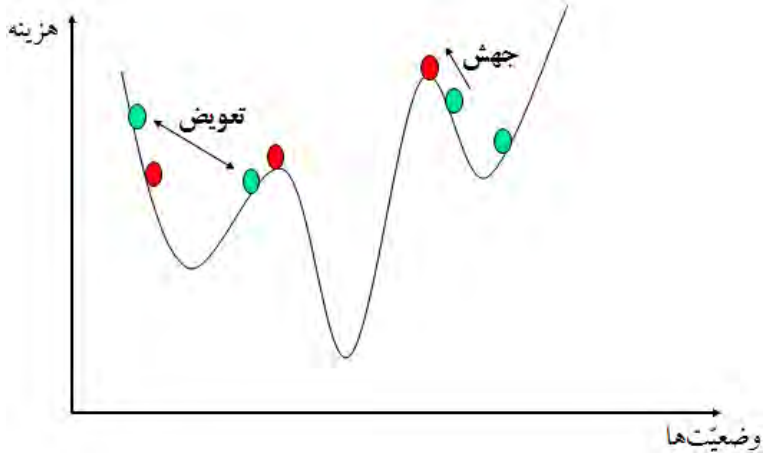

هزینه
وضعیّت‌ها



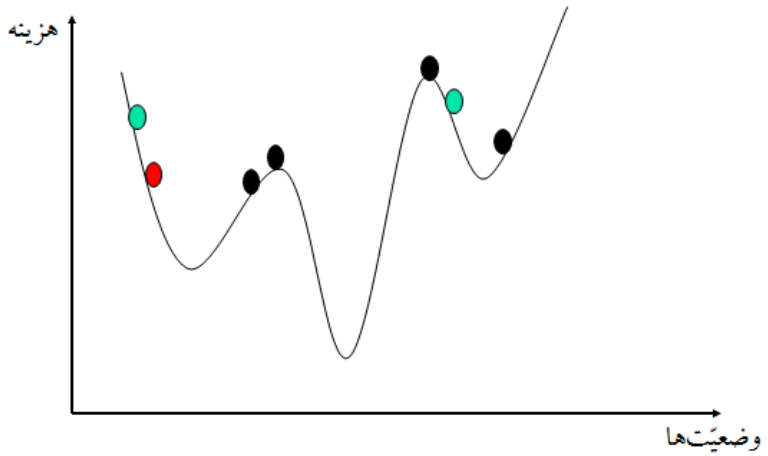
۱

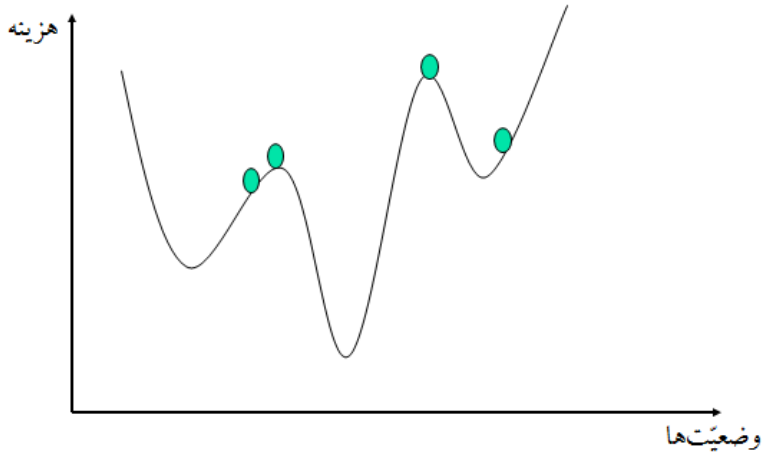
مثالی تصویری برای الگوریتم‌های ژنتیکی

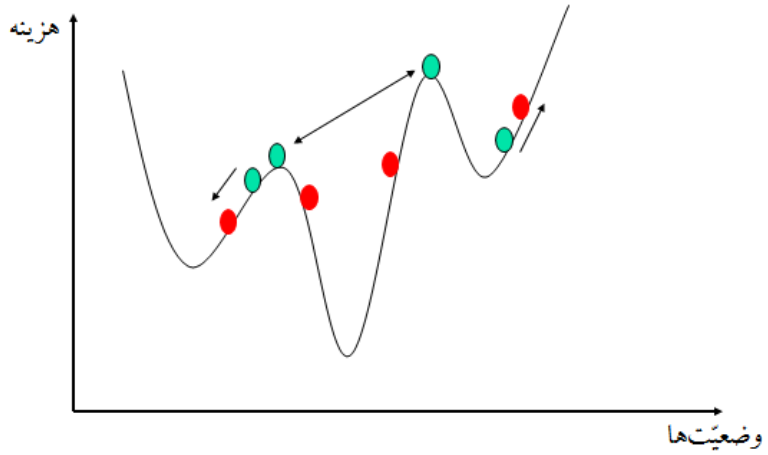
در شکل‌هایی که در ادامه می‌آیند، توجّه کنید که فلش‌های دوطرفه(↔) نمایشگر عمل تعویض و فلش‌های یکطرفه(→) نمایشگر عمل جهش هستند.

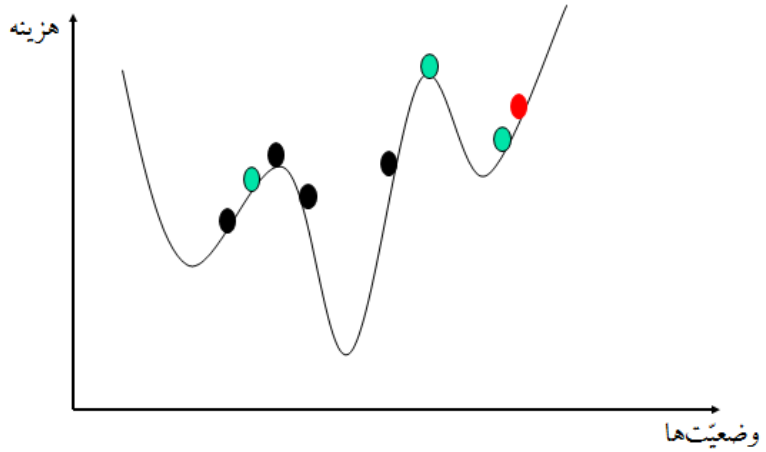


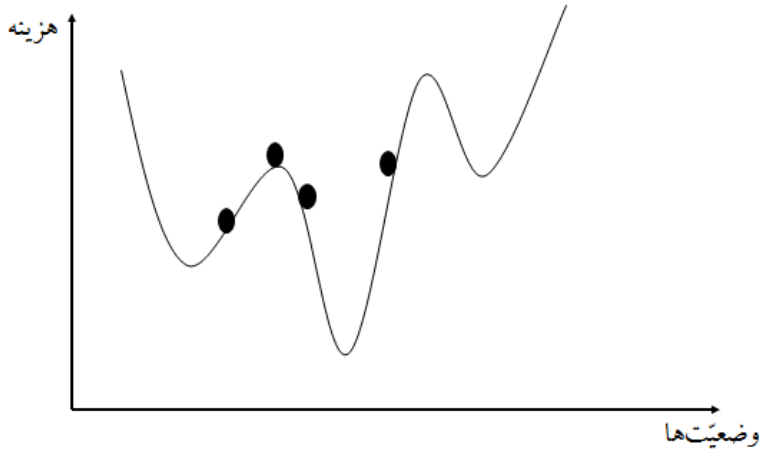
۱ – مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

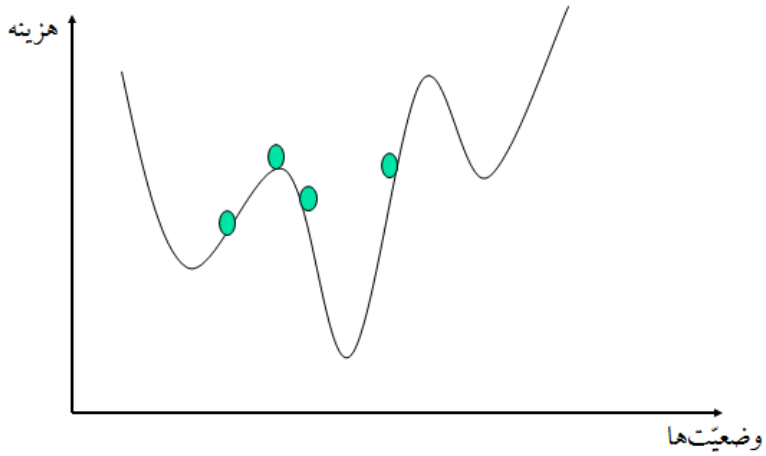

هزینه
وضعیّت‌ها

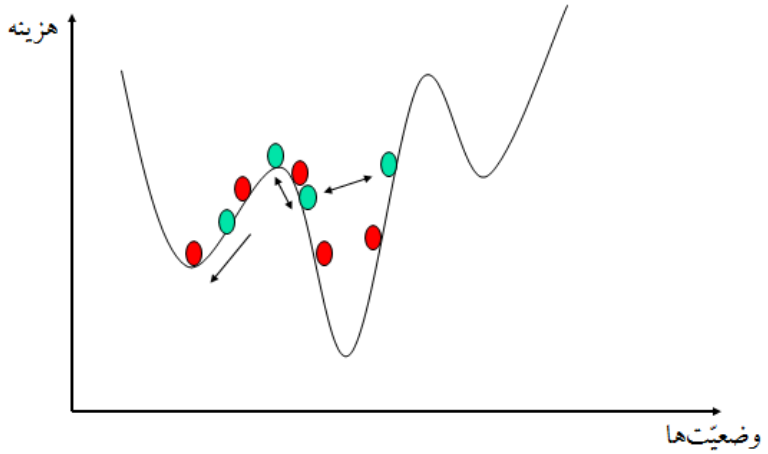

هزینه
وضعیّت‌ها

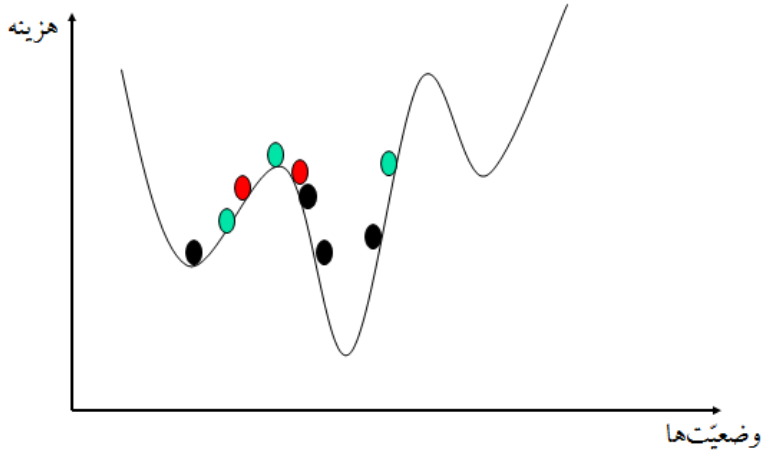

هزینه
وضعیّت‌ها

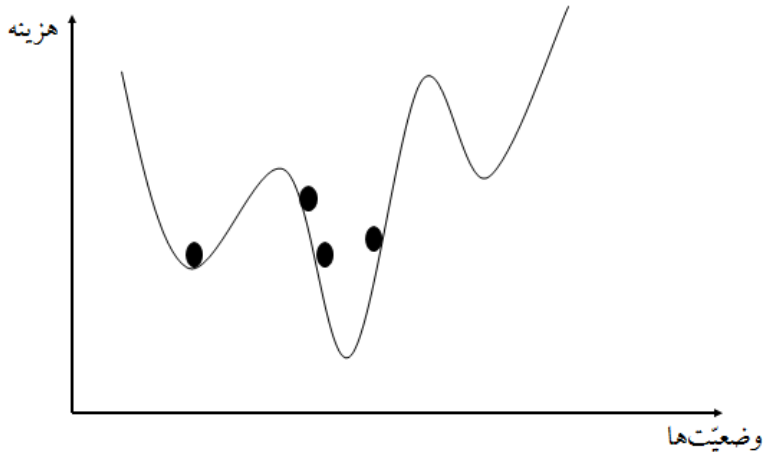

هزینه
وضعیّت‌ها

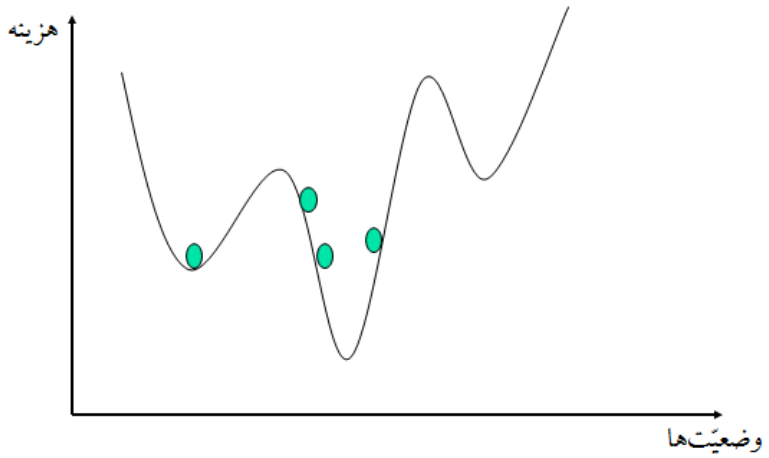

هزینه
وضعیّت‌ها

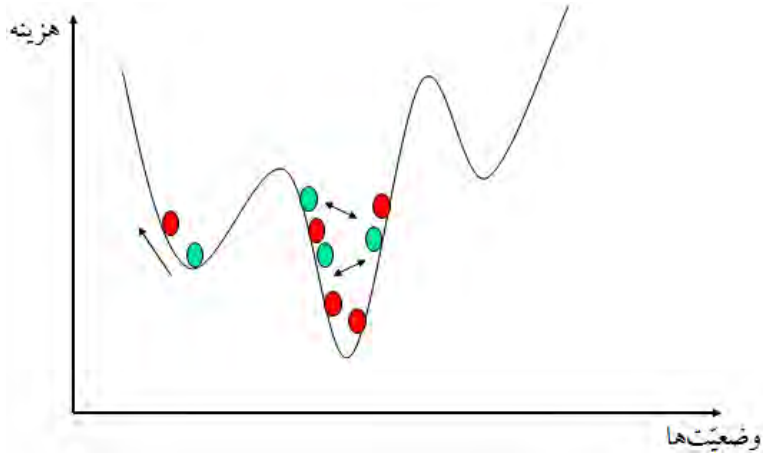

هزینه
وضعیّت‌ها

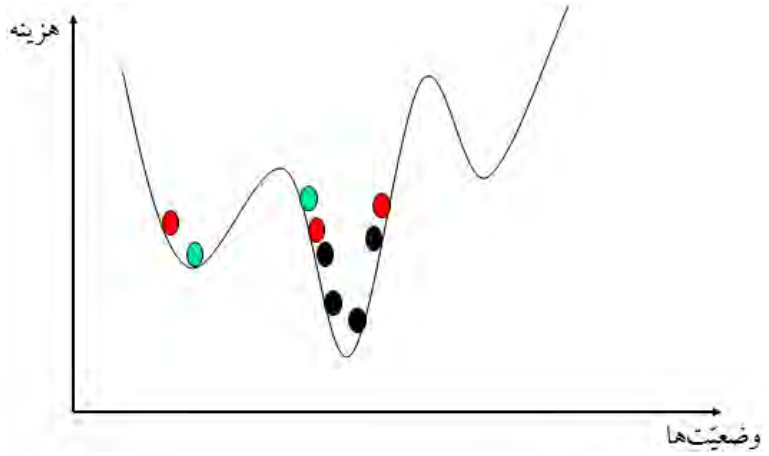

هزینه
وضعیّت‌ها

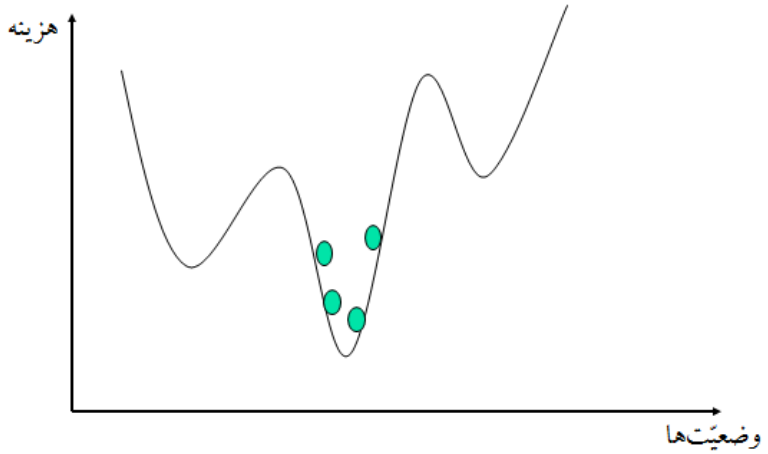

هزینه
وضعیّت‌ها

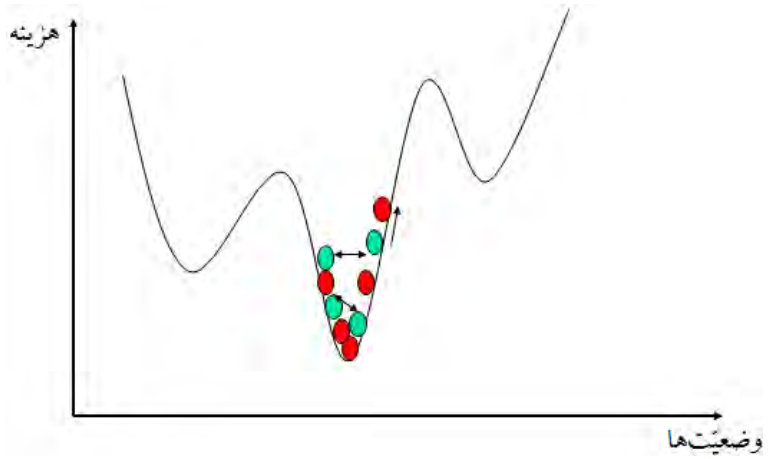

هزینه
وضعیّت‌ها

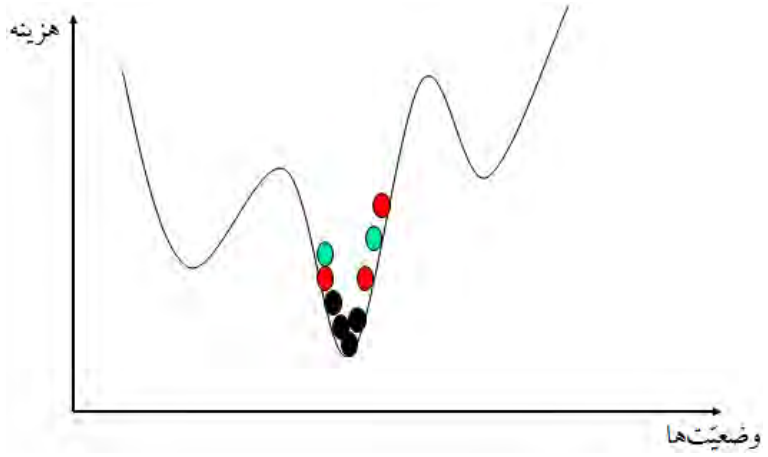

هزینه
وضعیّت‌ها

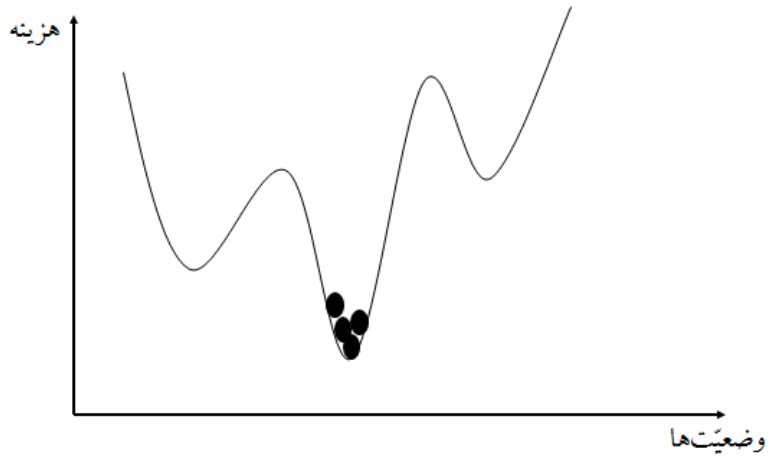

هزینه
وضعیّت‌ها

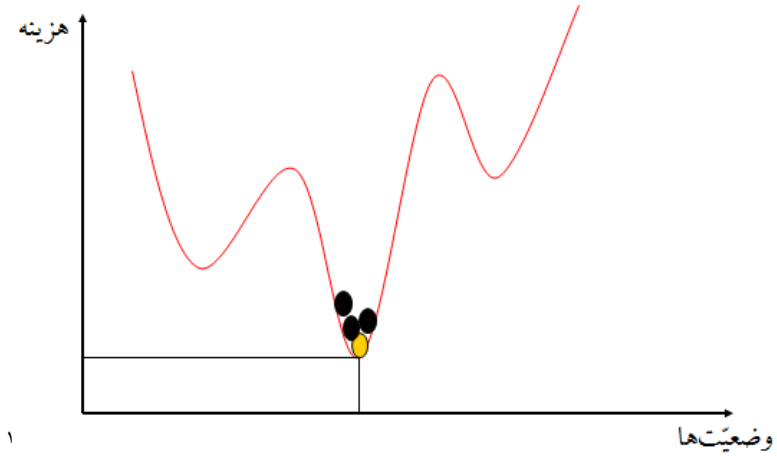

هزینه
وضعیّت‌ها


هزینه
وضعیّت‌ها


هزینه
وضعیّت‌ها


هزینه
وضعیّت‌ها



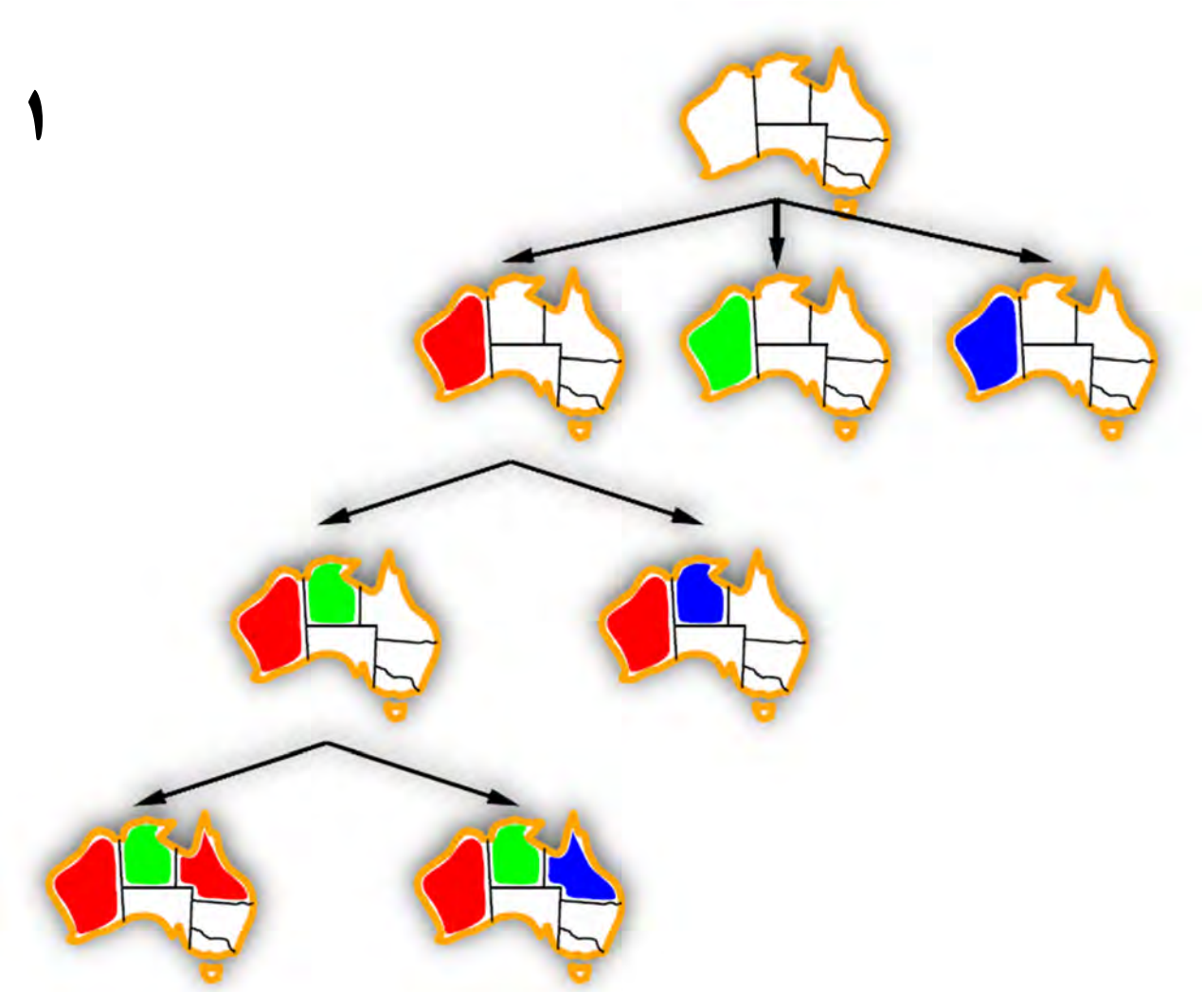


۱

۱ – مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

فصل هفتم

۱

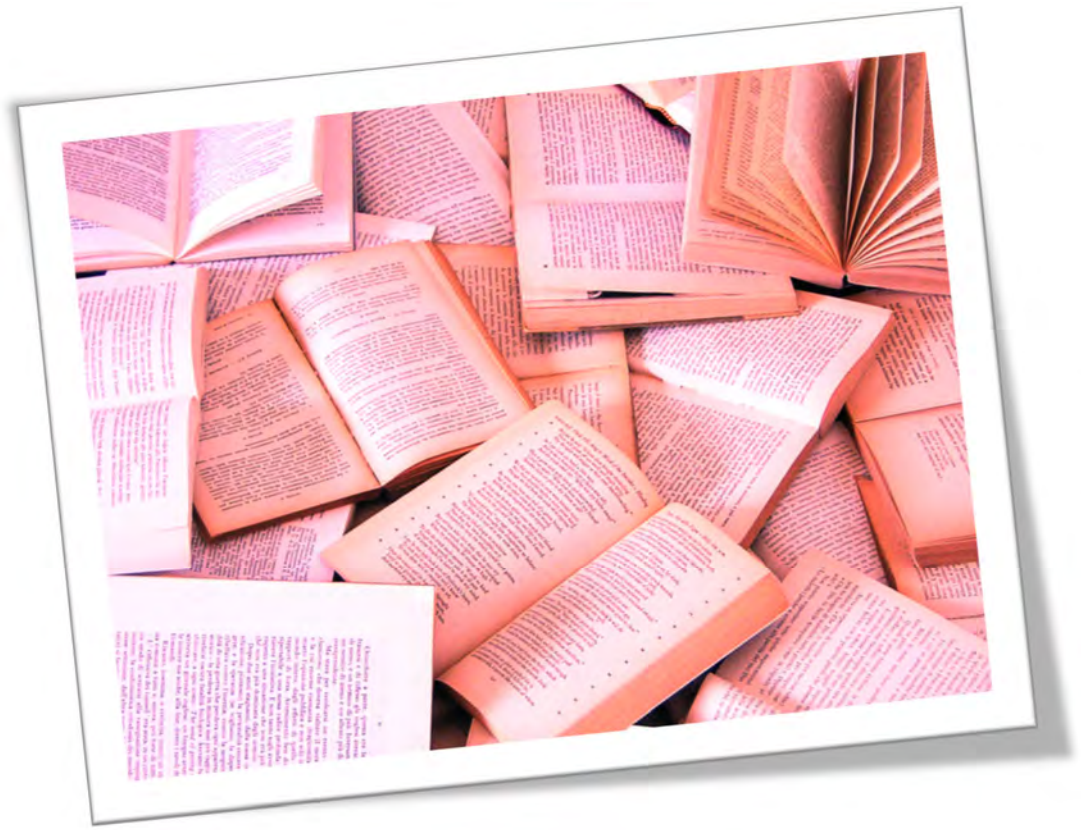


مسأله‌های برآورده‌سازی(اِرضای) محدودیّت[۲] (قُیود)

---

۱ - تصویر، بخشی از درخت جستجوی حلّ مسأله‌ی «رنگ‌آمیزی نقشه‌ی کشور استرالیا» با استفاده از روش «پیمایش(جستجوی) معکوس لیست» را که در ادامه‌ی همین فصل آن را برّرسی خواهیم کرد، نشان می‌دهد. در مسأله‌ی رنگ‌آمیزی نقشه‌ی کشور استرالیا باید ایالت‌های این کشور را با سه رنگ قرمز، سبز و آبی طوری رنگ‌آمیزی نماییم که شهرهای همسایه دارای رنگ‌های متفاوتی باشند.

۲ - Constraint Satisfaction Problems(*CSPs*)



فهرست برخی از عنوان‌های نوشته‌ها

مسأله‌های برآورده‌سازی محدودیّت

تعریف: مسأله‌هایی در ریّاضی هستند که به صورت مجموعه‌ای از اشیاء(چیزها) که وضعیّتشان باید تعدادی از محدودیّت‌ها را برآورده نماید، تعریف شده‌اند.[۱]

در این نوع از مسأله‌ها، مجموعه‌ای از متغیّرهای $\{X_1, X_2, \ldots, X_n\}$ را داریم؛ هر متغیّر X_i، با مقدارهای درون یک دامنه‌ی D_i مقداردهی می‌شود؛ معمولاً D_i به صورت گسسته و محدود می‌باشد. مجموعه‌ای از محدودیّت‌های $\{C_1, C_2, \ldots, C_k\}$ را نیز داریم، که هر محدودیّت C_k، شامل یک زیرمجموعه از متغیّرهاست و ترکیب‌های مُجاز مقدارهای متغیّرها را مشخّص می‌نماید. هدف این است که: یک مقدار را به هر متغیر طوری نسبت دهیم که همه‌ی محدودیّت‌ها برآورده شوند.

مثال- رنگ‌آمیزی نقشه[۲]ی کشور استرالیا

شکل بالا- نقشه‌ی کشور استرالیا

۱- http://en.wikipedia.org/wiki/Constraint_satisfaction_problem

۲- Map Coloring

متغیّرها، عبارتند از: WA[۱], NT[۲], Q[۳], NSW[۴], V[۵], SA[۶], T[۷].

دامنه: D_i= {red, green, blue}.

محدودیّت‌ها: نواحی همسایه(مجاور) باید رنگ‌های متفاوتی داشته باشند؛ به عنوان مثال، رنگ ناحیّه‌ی *W*estern *A*ustralia باید با رنگ ناحیّه‌ی *N*orthern *T*erritory متفاوت باشد($WA \neq NT$)؛ یا:

$(WA,NT) \in^{8}$ {(red,green),(red,blue),(green,red),(green,blue),…}

حل:

راه حلّ ارائه شده در بالا همه‌ی شرایط را لحاظ می‌کند:

{WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}

---

۱ – کوتاه شده‌ی «Western Australia» است.

۲ – کوتاه شده‌ی «Northern Territory» است.

۳ – کوتاه شده‌ی «Queensland(کوئینزلَنْدْ)» است.

۴ – کوتاه شده‌ی «New South Wales» است.

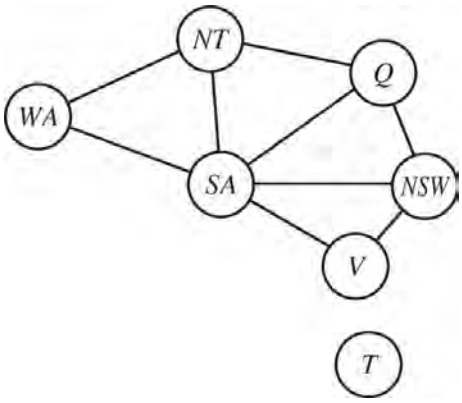
۵ – کوتاه شده‌ی «Victoria(ویکتوریا)» است.

۶ – کوتاه شده‌ی «South Australia» است.

۷ – کوتاه شده‌ی «Tasmania(تاسمانیا)» است.

۸ – از این نماد(∈) برای بیان عضویّت استفاده شده است؛ بخوانید «عضوِ»

گراف محدودیّت[۱]



در گراف محدودیّت، گره‌ها، متغیّرها هستند و کمان‌ها[۲](خط‌های بین گره‌ها)، محدودیّت‌ها را نشان می‌دهند. دو متغیّر، همسایه هستند، اگر با یک کمان به هم وصل شده باشند. الگوریتم‌های همه منظوره‌ی مسأله‌ی برآورده‌سازی محدودیّت، از ساختار گراف برای بالا بردن سرعت جستجو استفاده می‌کنند؛ مثلاً شهر تاسمانیا یک زیر مسأله‌ی مستقل است!

انواع مسأله‌های برآورده‌سازی محدودیّت

مسأله‌های برآورده‌سازی محدودیّتِ با متغیّرهای مجزّا(گسسته)،

در دامنه‌های محدود؛ اگر بیشینه‌ی اندازه‌ی دامنه‌ی هر متغیّر، در یک مسأله‌ی برآورده‌سازی محدودیّت، d باشد و n، تعداد متغیّرها باشد، در این صورت، تعداد انتساب‌های کامل ممکن، $O(d^n)$ خواهد بود.[۳] به عنوان مثال، مسأله‌های برآورده‌سازی محدودیّت دودویی[۴].

در دامنه‌های نامحدود (مثل: integerها(عددهای صحیح)، stringها(رشته‌ها) و.....)، به عنوان مثال، در زمانبند کار[۵]؛ متغیّرها، روزهای شروع یا پایان برای هر کار هستند و به یک زبان با محدودیّت[۶] نیازمندند، به عنوان مثال، $StartJob_1+5 \leq StartJob_3$.

مسأله‌های برآورده‌سازی محدودیّتِ با متغیّرهای پیوسته[۱]،

---

۱ - Constraint graph

۲ - arcs

۳ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش دوّم، فصل پنجم، مسأله‌های برآورده‌سازی محدودیّت(Constraint Satisfaction Problems)، صفحه‌ی ۱۳۹

۴ - Binary CSPs: این گونه، در ادامه‌ی همین درس توضیح داده شده است.

۵ - job scheduling

۶ - constraint language

به عنوان مثال، زمان‌های شروع یا پایان برای مشاهده‌های تلسکوپ فضایی هابل[۲].

گوناگونی محدودیّت‌ها

تعریف- محدودیّت‌های منحصر به فرد(یگانی)[۳]، شامل یک متغیّر تک(منفرد) می‌شوند،

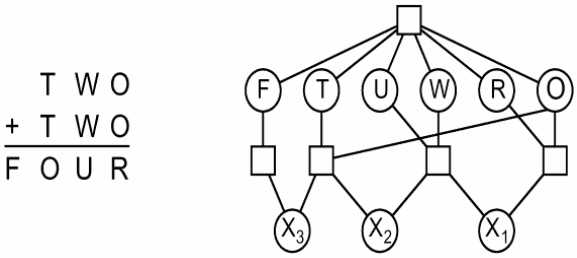
به عنوان مثال، SA≠green.

تعریف- محدودیّت‌های دودویی، شامل جفت‌هایی از متغیّرها می‌باشند،

به عنوان مثال، SA≠WA.

تعریف- محدودیّت‌های مرتبه‌ی بالاتر[۴]، شامل تعداد سه یا بیش‌تر متغیّر می‌باشند،

به عنوان مثال، محدودیّت‌های با ستون‌های پنهان(رمزی)[۵]، در این مسأله‌ها(مسأله‌های رمزی) باید هر حرف را طوری تغییر دهیم که حاصل جمع به دست آید. مثلاً در مثال زیر متغیّرها، $F\ T\ U\ \ W\ R\ O\ X_1\ X_2\ X_3$ و دامنه، {۰،۱،۲،۳،۴،۵،۶،۷،۸،۹} می‌باشد:



مثال(نمونه‌ای از مسأله‌های رمزی): در زیر هر حرف را به گونه‌ای تغییر دهید که حاصلجمع به دست آید.

$$\begin{array}{cccccc} & F & O & R & T & Y \\ + & & & T & E & N \\ + & & & T & E & N \\ \hline & S & I & X & T & Y \end{array}$$

۱- continuous variables
۲- Hubble Space Telescope
۳- unary
۴- Higher-order
۵- cryptarithmetic column constraints

حل:

	2	9	7	8	6
+			8	5	0
+			8	5	0
	3	1	4	8	6

یعنی، F= 2، O= 9، R= 7، ....

محدودیّت‌های اولویّت‌دار[1]، به عنوان مثال، رنگ قرمز بهتر از رنگ سبز است.

مسأله‌های برآورده‌سازی محدودیّت در دنیای واقعی

مسأله‌های انتساب؛ به عنوان مثال، چه فردی درس می‌دهد و در چه کلاسی؟.

مسأله‌های برنامه‌ی زمانی؛ به عنوان مثال، کدام کلاس ارائه می‌شود، چه زمانی و کجا؟.

پیکربندی سخت‌افزاری –صفحات گسترده[2] - زمانبند کننده‌ی انتقال –زمانبند کننده‌ی مرکز تولید.

پِیمایِش(جستجوی) معکوس[3]

انتساب متغیّرها جابجایی‌پذیر می‌باشد؛ مثلاً [WA=red, NT=green] همانند [NT=green, WA=red] می‌باشد.

---

۱ - Preference Constraints

۲ - spreadsheet؛ مثلاً برنامه‌ی اکسل شرکت میکروسافت(Microsoft Excel)

۳ - Backtracking search

مطلب مهم:

تعریف- ☑کنکور سراسری فنّـآوری اطّلاعات و مکاترونیک سال ۸۷- *جستجوی اوّلْ عمق برای مسأله‌های برآورده‌سازی محدودیّت با انتساب‌های متغیّر منفرد، جستجوی معکوس نام دارد.*

جستجوی معکوس، الگوریتم ناآگاهانه‌ی اساسی برای مسأله‌های برآورده‌سازی محدودیّت می‌باشد؛ با ایـن روش می‌تـوان مسـأله‌ی n-وزیر را برای $n \approx 25$ حل نمود.

الگوریتم

تابع Backtracking-Search(csp)، راه حل یا عدم موفّقیّت را برمی‌گرداند

Recursive-Backtracking({},csp) را برگردان

تابع Recursive-Backtracking(assignment[1],csp)، راه حل یا عدم موفّقیّت را برمی‌گرداند

اگر assignment، کامل است، assignment را برگردان

var ← Select[2]-Unassigned[3]-Variable(Variable[csp],assignment,csp)

برای هر value در Order-Domain-Values(var,assignment,csp)، کارهای زیر را انجام بده

در صورتی که value با assignment ارائه شده در Constraint[csp] سازگار است

{ var = value } را به assignment اضافه نما

result ← Recursive-Backtracking(assignment,csp)

در صورتی که result ≠ failure است، result را برگردان

{var = value} را از assignment بردار

پایان شرط

پایان حلقه

عدم موفّقیّت را برگردان

---

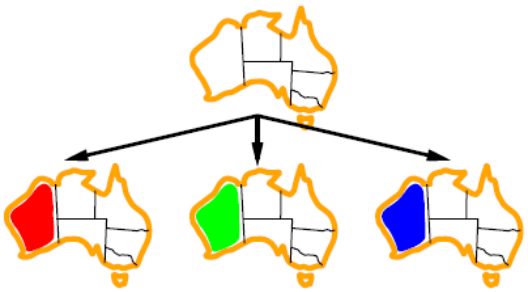
۱ – در لغت به معنی «انتساب، جایگزینی» می‌باشد.

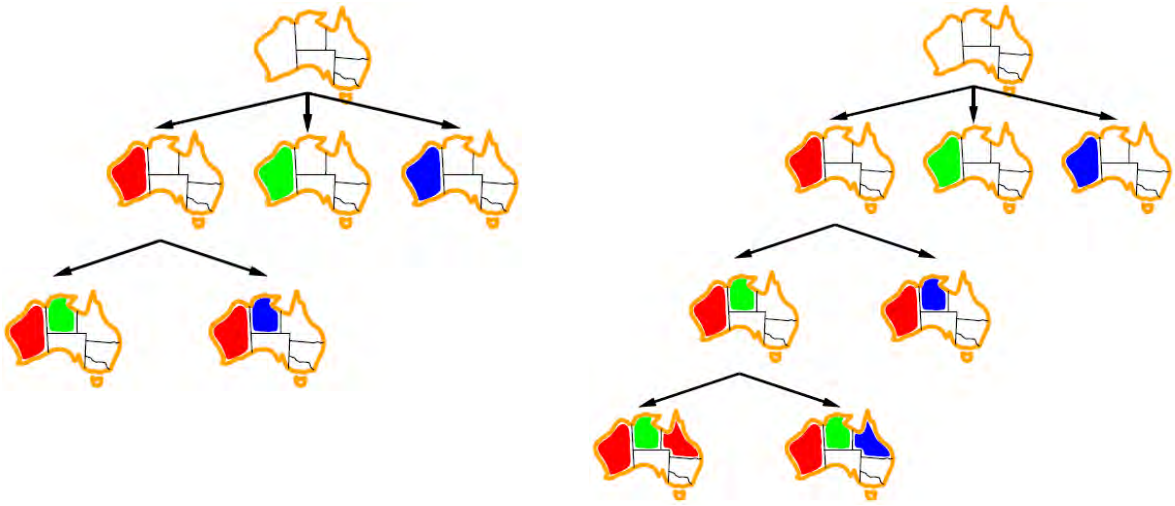
۲ – در لغت به معنی «انتخاب کردن» می‌باشد.

۳ – در لغت به معنی «نسبت داده نشده» می‌باشد.

پایان الگوریتم

مثال پیمایش(جستجوی) معکوس لیست؛ رنگ‌آمیزی نقشه‌ی کشور استرالیا؛ توجّه کنید که فقط بخشی از این درخت جستجو آورده شده است.(تصاویر را از چپ به راست(→) دنبال نمایید):





بهبود دادن کارآیی پیمایش معکوس

شیوه‌هایی برای بالابردن سرعت وجود دارد:

۱- **کدام متغیّر باید به عنوان بعدی انتخاب شود؟**

۲- **به کدام ترتیب، مقدارها باید امتحان شوند؟**

۳- **آیا می‌توان عدم موفّقیّت‌های حتمی را زود پیدا کرد؟**

۴- چگونه می‌توان از ساختار مسأله سود برد؟؛ این مورد را برّرسی نمی‌کنیم.

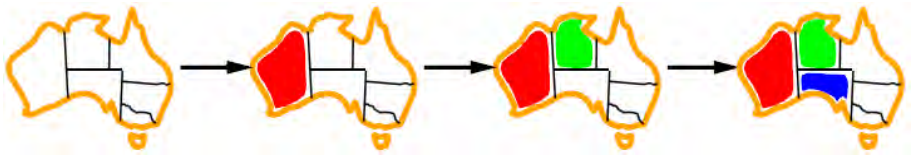
۱– کدام متغیّر باید به صورت بعدی انتخاب شود؟

در این مورد از دو مکاشفه‌ی زیر استفاده می‌کنیم:

الف– مکاشفه‌ی کم‌ترین مقدار باقیمانده[۱]

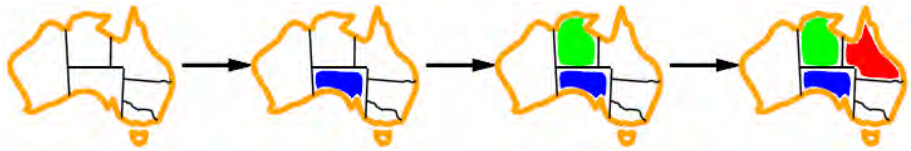
مطلب– ☑ **کنکور سراسری مهندسی کامپیوتر سال ۸۴**– در این مکاشفه، متغیّری که دارای کم‌ترین مقدارهای مُجاز است را انتخاب نمائید؛ مثلاً برای مثال استرالیا داریم:

۱ – *M*inimum *r*emaining *v*alues(*MRV*)



ب– مکاشفه‌ی درجه[۱]

در میان متغیّرهای باقیمانده، محدودترین متغیّر را انتخاب نمائید؛ مثلاً در مثال استرالیا:



۲– به کدام ترتیب، مقدارها باید امتحان شوند؟

در این مورد از مکاشفه‌ی زیر استفاده می‌کنیم:

کم‌ترین مقدار محدود کننده[۲]

وقتی که یک متغیّر را انتخاب کردید، کم‌ترین مقدار محدود کننده را انتخاب نمائید؛ مثلاً در مثال استرالیا، اگر شکل زیر را داشته باشیم:

و سپس Q را به صورت زیر رنگ کنیم،

آنگاه می‌توانیم SA را به رنگ آبی در آوریم:

---

۱ - Degree

۲ - Least constraining value

ولی اگر Q را به صورت زیر رنگ کنیم،

آنگاه دیگر هیچ موردی برای SA مُجاز نمی‌باشد.

۳- آیا می‌توان عدم موفّقیّت‌های حتمی را زود پیدا کرد؟

برّرسی مستقیم[۱]

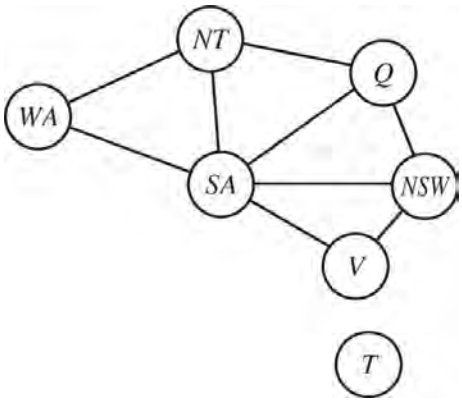
مطلب مهم:

رَوَند برّرسی مستقیم، به این صورت است: بعد از مقداردهی متغیّر X، با یک مقدار v، به هر مقدار متغیّر مقداردهی نشده، مثل Y، که به وسیله‌ی یک محدودیّت به X وصل شده است، نگاه کن و مقدارهایی از دامنه‌ی Y را که با مقدار v ناسازگارند، حذف کن.

۱- Forward Checking(*FC*)

مثال- رنگ‌آمیزی نقشه‌ی کشور استرالیا با استفاده از روش برّرسی مستقیم

گراف محدودیّت این مسأله به صورت زیر است:

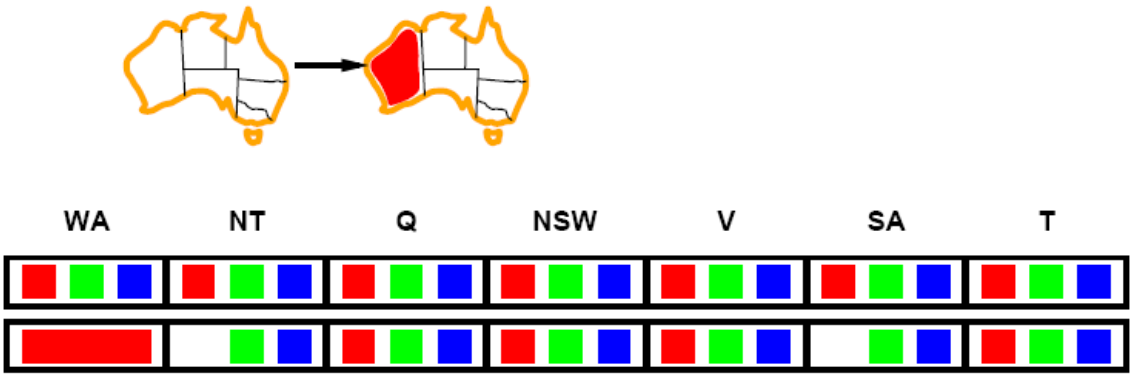


در ابتدا-

توضیح:

همان طور که در قسمت پایین شکل بالا می‌بینید، در ابتدا برای هر کدام از قسمت‌های موجود روی نقشه از هر کدام از سه رنگ قرمز، سبز و آبی می‌توان استفاده نمود.

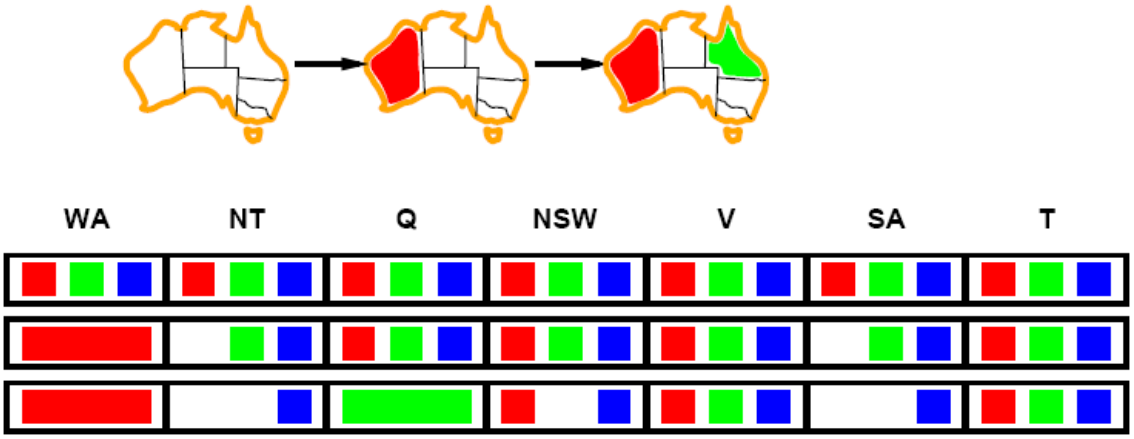
گام نخست-



توضیح:

همان طور که در قسمت پایین شکل بالا می‌بینید، بعد از اینکه WA به رنگ قرمز در آمد، NT و SA دیگر نمی‌توانند به رنگ قرمز درآیند.

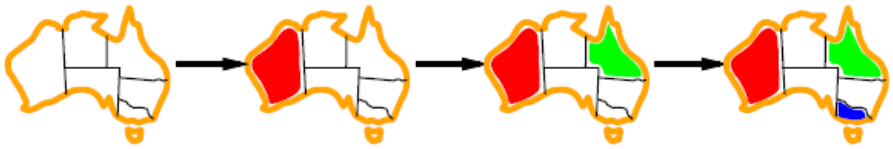
گام بعدی-



توضیح:

همان طور که در قسمت پایین شکل بالا می‌بینید، بعد از اینکه Q به رنگ سبز در آمد، NT، NSW و SA دیگر نمی‌توانند به رنگ سبز درآیند.

گام بعدی–



پخش محدودیّت[۱]

تعریف– پردازه‌ی تصمیم‌گیری در مورد اینکه چگونه مقدارهای ممکن یک متغیّر، مقدارهای ممکن دیگر متغیّرها را تحت تأثیر قرار می‌دهند، می‌باشد.

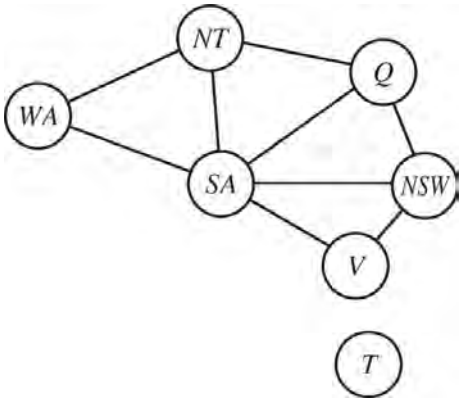
روش برّرسی مستقیم، عدم موفّقیّت‌ها را زود پیدا نمی‌کند:

با توجّه به شکل بالا، به عنوان مثال، NT و SA هر دو نمی‌توانند به رنگ آبی باشند!؛ *پخش محدودّیت، به صورت تکراری، محدودّیت‌ها را به صورت محلّی اجرا می‌کند:*

سازگاری قُوْس[۱]– منظور از قوس، قوس مستقیم، در گراف محدودیّت است، مثل قوسی که از SA به NSW وجود دارد:

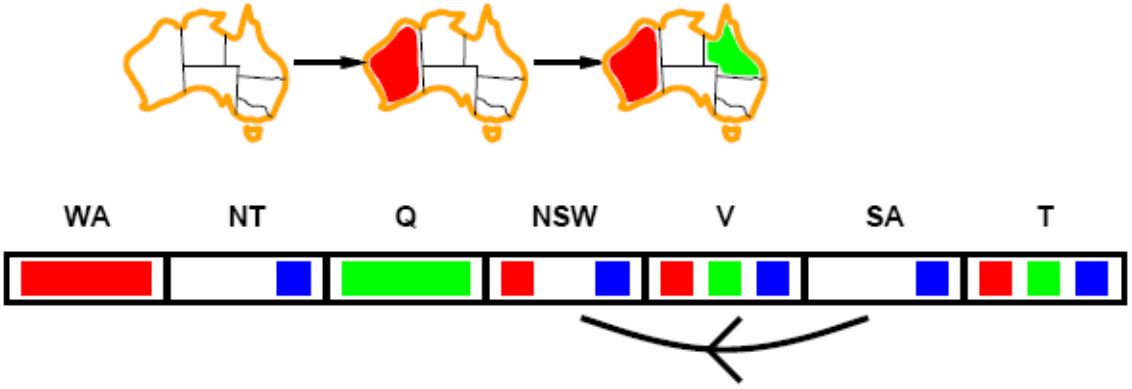
---

۱ – Constraint propagation

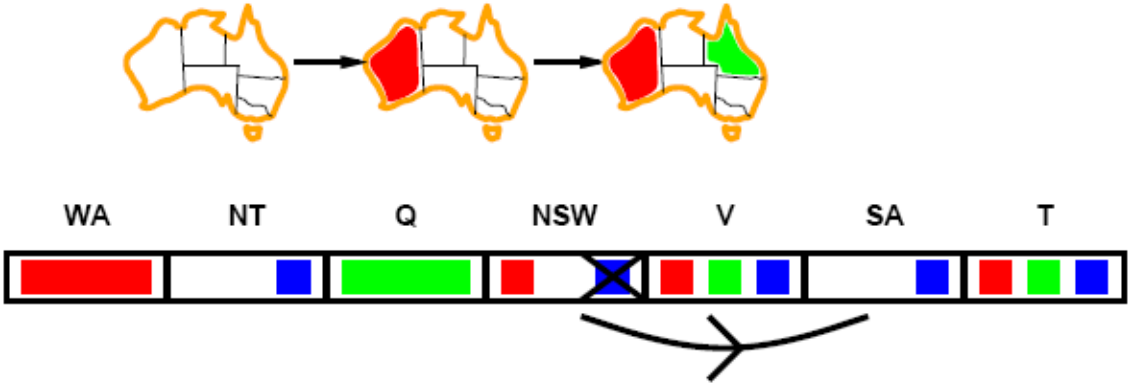


مطلب مهم:

قوس $X \rightarrow Y$ سازگار می‌باشد، اگر برای هر مقدار x از X، تعدادی برای y از Y *مجاز* باشند. به عنوان مثال، $SA \rightarrow NSW$ سازگار است، اگر SA، آبی و NSW، قرمز باشد:



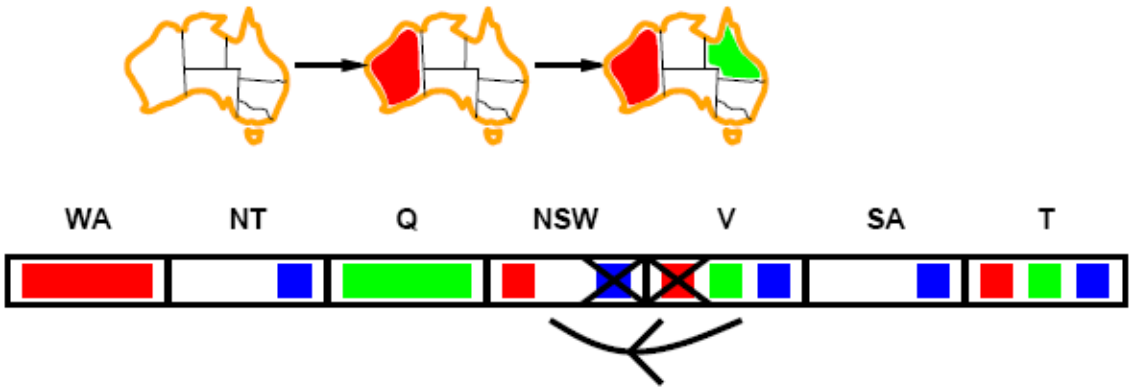
با برداشتن آبی از NSW، قوس می‌تواند سازگار شود:



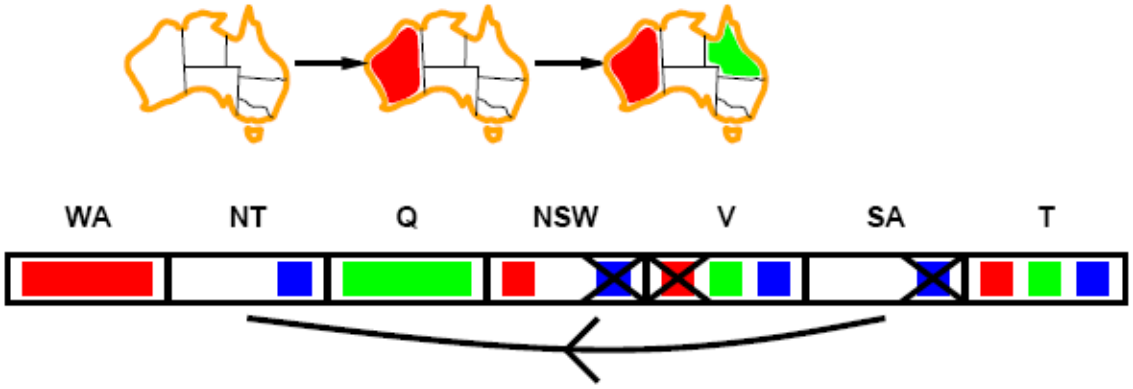
مطلب مهم:

اگر X یک مقدار را از دست داد، همسایه‌های X باید دوباره بررّسی شوند؛ همسایگان NSW را دوباره بررّسی نمایید؛ قرمز را از

V بردارید:



همان طور که گفتیم، اگر X یک مقدار را از دست داد، همسایه‌های X باید دوباره برّرسی شوند:



مطلب مهم:

در حالت کلّی، سازگاری قوس، زودتر از برّرسی مستقیم، عدم موفّقیّت را پیدا می‌کند و می‌تواند به صورت یک پیش پردازنده و یا بعد از هر انتساب اجرا شود.

الگوریتم سازگاری قوس

تابع AC-3(csp)، CSP را برمی‌گرداند، شاید دامنه‌ها کاهش داده شده باشند

ورودی‌ها، CSP، که یک مسأله‌ی برآورده‌سازی محدودیّت دودویی با متغیّرهای $\{X_1, X_2, ..., X_n\}$ می‌باشد

متغیّرهای محلّی، queue، یک صف از قوس‌هاست، که همه به صورت اوّلیّه، قوس‌هایی در csp هستند

مادامی که queue خالی نیست، کارهای زیر را انجام بده

$(X_i, X_j) \leftarrow$ Remove-First(queue)

اگر Remove-Inconsistent-Values(X_i, X_j)، آنگاه

برای هر X_k در $Neighbours[X_i]$ –– X_j کارهای زیر را انجام بده

(X_k,X_i) را به queue اضافه کن

پایان حلقه

پایان شرط

پایان حلقه

تابع Remove-Inconsistent-Values(X_i,X_j)، اگر یک مقدار را برداریم، true را برمی‌گرداند

removed ← false

برای هر x در $Domain[X_i]$ کارهای زیر را انجام بده

در صورتی که هیچ مقدار y در $Domain[X_i]$ اجازه نمی‌دهد که (x,y) محدودیّت‌های میان X_i و X_j را برآورده کند، آنگاه

x را از $Domain[X_i]$ حذف کن؛

removed ← true

پایان شرط

پایان حلقه

removed را برگردان

پایان الگوریتم

سختی مسأله‌ی برآورده‌سازی محدودیّت

به طور کلّی، مسأله‌های برآورده‌سازی محدودیّت، NP–کامل هستند، برای آنها از الگوریتم‌های مکاشفه‌ای، که دارای زمان چندجمله‌ای هستند، استفاده کنید.

برنامه‌نویسی براساس محدودیّت

محدودیّت‌ها یک راه طبیعی برای بیان مسأله‌ها می‌باشند. برنامه‌نویسی مسأله‌ی برآورده‌سازی محدودیّت، با یک زبان برنامه‌نویسی منطقی، نظیر پرولوگ انجام می‌شود.

چکیده‌ی مطلب‌های فصل هفتم

مسأله‌های برآورده‌سازی محدودیّت، نوع خاصی از مسأله‌ها هستند که در آنها حالت‌ها، به وسیله‌ی مقدارهای یک مجموعه‌ی ثابت از متغیّرها تعریف می‌شوند و آزمایش(آزمون) هدف، به وسیله‌ی محدودیّت‌هایی بر مقدار متغیّرها تعریف می‌شود.

بیان مسأله به صورت مسأله‌ی برآورده‌سازی محدودیّت، تحلیل ساختار مسأله را اجازه می‌دهد.

پیمایش معکوس، جستجوی اوّل-عمق، با یک متغیّر انتساب داده شده به هر گره می‌باشد؛ در این مورد، مرتّب کردن متغیّر و انتخاب ابتکاری مقدارها، به طور بسیار مطلوبی به ما کمک می‌کند. برّرسی مستقیم، از مقداردهی‌هایی که عدم موفّقیّت را در آینده به وجود می‌آورند، جلوگیری می‌کند. پخش محدودیّت، به عنوان مثال، سازگاری قوس، کاری تکمیلی را برای مقدارهای محدود و کشف ناسازگاری‌ها انجام می‌دهد.

یادآوری یا تکمیل مطلب‌های فصل هفتم

سؤال– مینی‌سودوکو[۱] را توضیح دهید.

جواب– مینی‌سودوکو، گونه‌ای کوچک از بازی مشهور سودوکو است. مینی‌سودوکو، از یک جدول ۴×۴ درست شده است که به ۴ جدول ۲×۲، که ناحیّه[۲] نامیده می‌شوند، تقسیم شده است. هر خانه باید با عددی از ۱ تا ۴ پر شود. با داشتن یک جدول که به صورت ناتمام پر شده است، هدف این است که بقیّه‌ی اعداد را با توجّه به محدودیّت‌های زیر پر کنیم:

یک عدد فقط یکبار باید در هر سطر ظاهر شود.

یک عدد فقط یکبار باید در هر ستون ظاهر شود.

یک عدد فقط یکبار باید در هر ناحیّه‌ی ۲×۲ ظاهر شود.[۳]

1	2		2
4			
	4		

شکل بالا– یک مینی‌سودوکوی غیر مُجاز؛ عددهای ۲ و ۴ از محدودیّت‌ها تخلّف کرده‌اند.

1	2	3	4
3	4	2	1
4	3	1	2
2	1	4	3

شکل بالا– یک مینی‌سودوکوی مُجاز

سؤال– آیا مینی‌سودوکو می‌تواند به صورت یک مسأله‌ی CSP بیان شود؟

---

۱ – Mini-Sudoku

۲ – region

۳ – پرسش‌های درس «هوش مصنوعی» استاد، «پیتر اَبیل»، دانشکده‌ی شهر برکلی دانشگاه ایالت کالیفرنیای کشور ایالات متّحده‌ی آمریکا، بهار سال ۲۰۱۲ میلادی

جواب- بله[۱].

مطلب- یک انتساب که هیچ محدودیّتی را نقض نمی‌کند، *سازگار* نامیده می‌شود.[۲]

مطلب- انتسابی *کامل* است که در آن همه‌ی متغیّرها مقداردهی شده باشند.[۳]

درست یا غلط- مسأله‌های برآورده‌سازی محدودیّتی وجود دارند که می‌توانند به صورت ۳-متغیّری بیان شوند، امّا به صورت محدودیّت‌های دودویی(باینری) نتوان آنها را بیان کرد.

جواب- **«غلط» است؛** تمام مسأله‌های برآورده‌سازی محدودیّت، دارای یک فرمول‌بندی دودویی هستند.[۴]

تعریف- راه حل برای یک CSP چیست؟

جواب- یک انتساب کامل و سازگار است.[۵]

درست یا غلط- یک مسأله‌ی برآورده‌سازی محدودیّت، از مجموعه‌ای از متغیّرها، مجموعه‌ای از دامنه‌ها (برای هر متغیّر، یکی)، و مجموعه‌ای از محدودیّت‌ها که ترکیبات مجاز مقدارها را مشخّص می‌کنند، تشکیل شده است.

جواب- **«درست» است.**[۶]

درست یا غلط- یک انتساب سازگار، انتسابی است که در آن همه‌ی متغیّرها مقداردهی شده باشند.

جواب- **«غلط» است.**[۷]

---

۱ - پرسش‌های درس «هوش مصنوعی» استاد، «پیتر اَبیل»، دانشکده‌ی شهر برکلی دانشگاه ایالت کالیفرنیای کشور ایالات متّحده‌ی آمریکا، بهار سال ۲۰۱۲ میلادی

۲ - مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

۳ - مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

۴ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «دن کلین»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۲۰۰۶ میلادی

۵ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۶ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۷ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

درست یا غلط- یک انتساب کامل، انتسابی است که در آن هیچ محدودیّتی نقض نشده باشد.

جواب- «غلط» است.[۱]

درست یا غلط- گره‌های موجود در گراف محدودیّت، با متغیّرهای مسأله متناظرند، و یک کمان(لینک، پیوند[۲])، دو متغیّر را که با هم یک در یک محدودیّت، مشترک هستند، به هم وصل می‌کند.

جواب- «درست» است.[۳]

درست یا غلط- مکاشفه‌ی کم‌ترین مقدار باقیمانده(MRV)، متغیّر با کم‌ترین مقدارهای مجاز باقیمانده را برای انتساب بعدی انتخاب می‌کند.

جواب- «درست» است.[۴]

درست یا غلط- مکاشفه‌ی درجه، برای تنظیم دما در متدهایی که برای حلّ مسأله‌های CSP، براساس روش شبیه‌سازی گرم و سرد کردن استفاده می‌شوند، استفاده می‌شود.

جواب- «غلط» است.[۵]

تعریف- «سازگاری گره»[۶] را تعریف کنید.

جواب- تمام مقدارهای درون دامنه‌ی یک متغیّر، محدودیّت‌های یگانیش را برآورده کند.[۷] مثلاً در مسأله‌ی رنگ‌آمیزی نقشه‌ی کشور استرالیا، اگر ساکنان SA، رنگ سبز را دوست نداشته باشند و در شروع، SA، دارای دامنه‌ی {قرمز، سبز، آبی} باشد، می‌توانیم با حذف سبز، SA را دارای سازگاری گره کنیم؛ یعنی، SA، دارای دامنه‌ی کاهش داده شده‌ی {قرمز، آبی} شود.[۱]

---

۱ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۲ - link

۳ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۴ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۵ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۶ - Node Consistency

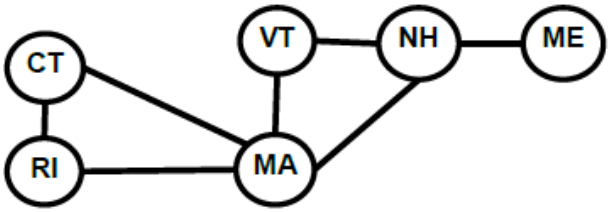
۷ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

سؤال- نقشه‌ی زیر را که بخشی از شرق کشور آمریکا را نشان می‌دهد، در نظر بگیرید:

فرض کنید می‌خواهیم این نقشه را با سه رنگ قرمز(R)، سبز(G) و آبی(B)، رنگ کنیم، طوری که هیچکدام از ایالت‌های کنار هم به یک رنگ نباشند؛

الف) برای این مسأله گراف محدودیّت رسم کنید.

جواب-



ب) تمام مقدارهایی که با استفاده از روش برّرسی مستقیم، پس از اینکه MA مطابق شکل زیر مقدار R را گرفت، باید حذف شوند، ضربدر بزنید.

CT	RI	MA	VT	NH	ME
R G B	R G B	R	R G B	R G B	R G B

جواب-

CT	RI	MA	VT	NH	ME
~~R~~ G B	~~R~~ G B	R	~~R~~ G B	~~R~~ G B	R G B

پ) CT و RI مطابق شکل زیر مقداردهی شده‌اند، امّا پخش محدودیّت صورت نگرفته است. تمام مقدارهایی که با استفاده از سازگاری قوس باید حذف شوند، ضربدر بزنید.

CT	RI	MA	VT	NH	ME
R	G	R G B	R G B	R G B	R G B

جواب-

---

۱ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل ششم، مسأله‌های برآورده‌سازی محدودیّت(Constraint Satisfaction Problems)، صفحه‌ی ۲۰۸

CT	RI	MA	VT	NH	ME
R	G	B	R G	R G	R G B

ت) به انتساب زیر توجّه کنید. RI مقداردهی شده و پخش محدودیّت انجام شده است. تمام متغیّرهای مقداردهی نشده‌ای که می‌توانند به وسیله‌ی مکاشفه‌ی کم‌ترین مقدارهای باقیمانده(MRV) انتخاب شوند، لیست کنید.

CT	RI	MA	VT	NH	ME
R B	G	R B	R G B	R G B	R G B

جواب- CT و MA

ث) به مقداردهی زیر توجّه کنید. RI مقداردهی شده و پخش محدودیّت انجام شده است. متغیّری که با استفاده از مکاشفه‌ی درجه باید به عنوان بعدی انتخاب شود، پیدا کنید.

CT	RI	MA	VT	NH	ME
R B	G	R B	R B	R G B	R G B

جواب- MA[1]

۱ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، پاییز سال ۲۰۱۲ میلادی

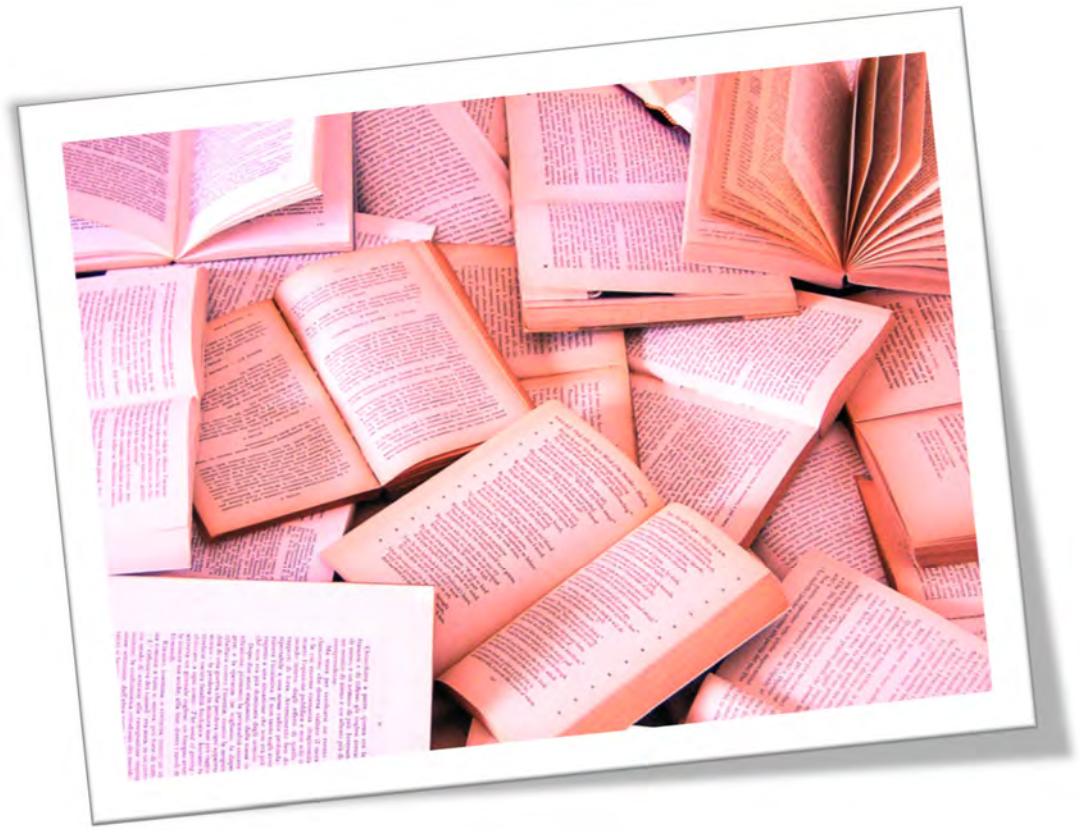
فصل هشتم

تئوری بازی‌ها[۲]

---

۱ - تصویرهای بالا متعلّق به گَری کاسپارُوف(Garry Kasparov)، متولّد ۱۹۶۳ میلادی، قهرمان بزرگ سابق شطرنج جهان، از کشور روسیّه است. (http://en.wikipedia.org/wiki/Garry_Kasparov)

۲ - Game playing؛ توجّه کنید که «بازی‌های خَصمانه(Adversarial games)» که برای آن‌ها از «جستجوی خصمانه(Adversarial search)» استفاده می‌کنیم، بخشی کوچک از تئوری بازی‌ها هستند؛ بازی‌های خصمانه، بازی‌هایی هستند که در آن‌ها یکی می‌برد و دیگران می‌بازند.



فهرست برخی از عنوان‌های نوشته‌ها

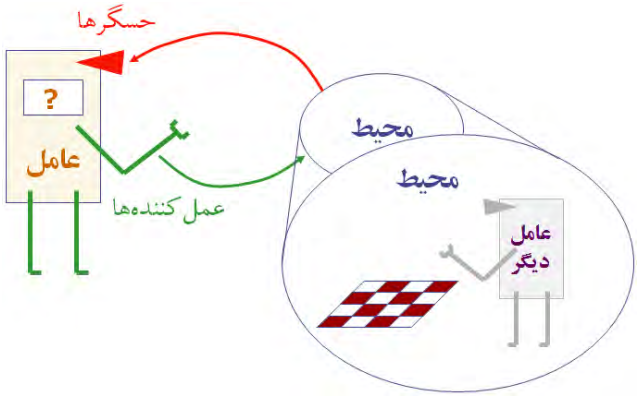
هَرَس(بازبینی) α- β(آلفا-بتا)

بازی‌های قطعی در عمل

بازی‌های غیرقطعی در حالت کلّی

محیط‌های چندعاملی

الگوریتم‌های جستجویی که تا حالا برّرسی کرده‌ایم، دارای یک محیط تک عاملی بودند و عامل، در حال کـار در یـک جهان بی‌اثر[۱] بود؛ به عبارت دیگر، سایر عامل‌ها بر روی آن اثری نداشتند؛ امّا در موردهایی که یک عامل بایـد دیگـر عامل‌هـا را بـه حساب بیاورد، چطور؟:



این محیط‌ها، مثل محیط‌های رقابتی، نظیر بازی‌هـای شـطرنج[۲]، چِکِـرز[۳]، go و...؛ یـا حراجی‌هـا[۴]، فروشـگاه‌های بـرخطّ اینترنتـی؛ و محیط‌های شراکتی[۵]، مثل عامل‌های یک تیم فوتبال می‌باشند:

---

۱ - neutral

۲ - chess

۳ - checkers(دام یا جَنگِ نادِر یا Draughts)

۴ - auctions

۵ - cooperative

شکل بالا- تعریف- رُبوکاپ[۱]: مسابقه‌ای بین المللی بین روبوت‌ها است که در سال ۱۹۹۷ میلادی بنیان آن گذاشته شده است. هدف ربوکاپ، توسعه‌ی روبوت‌های خودکار(خودمختار) بازی فوتبال، با هدف افزایش تحقیق و آموزش در زمینه‌ی هوش مصنوعی است.[۲]

تئوری بازی‌ها

تعریف- تئوری بازی‌ها، شاخه‌ای از علم ریّاضیّات یا علم اقتصاد می‌باشد که به اثرات متقابل میان چند عامل می‌پردازد و روش‌هایی را برای تشخیص رفتار بهینه تعیین می‌نماید.

تئوری بازی‌ها، یکی از قدیمی‌ترین مباحثی است که در هوش مصنوعی راجع به آن زیاد مطالعه شده است؛ دلیل این مطلب آن است که:

اوّل، افراد بازی‌ها را دوست دارند! و خوب می‌توانند با بازی‌ها کار کنند.

دوّم، بیان آنها ساده است و عامل‌ها دارای تعداد اندکی عملیّات هستند.

سوّم، بازی‌ها دارای یک توضیح واضح از محیط می‌باشند (البتّه در آنها فضاهای حالت، خیلی بزرگ و پیچیده‌اند، بنابراین، برخی وقت‌ها اطّلاعات، اتّفاقی و ناکارآمد می‌باشند).

---

۱ - *RoboCup*، کوتاه شده‌ی کلمه‌ی «*Robo*t Soccer World *Cup*»؛ به معنی «گُلدان جایزه‌ی مسابقه( )ی جهانی فوتبال روبوت‌ها» است.

۲ - http://en.wikipedia.org/wiki/RoboCup

نوع‌های بازی‌ها

تعریف- بازی‌های با اطّلاعات کامل[۱]: عامل‌ها به همه‌ی دانش در مورد محیط دسترسی دارند؛ *این محیط را محیط کاملاً قابل مشاهده می‌نامیم*؛ مثل بازی شطرنج.

تعریف- بازی‌های با اطّلاعات ناکامل(ناقص)[۲]: در این مورد عامل باید در مورد جهان یا حریف[۳] خود استنتاج نماید؛ *این محیط را محیط قابل مشاهده به صورت جزئی می‌نامیم*؛ مثال‌ها، بازی‌های شانسی (مثل بازی‌های کارتی پُل[۴] و پوکِر[۵]) و برخی از حراجی‌ها می‌باشند.

	قطعی	شانسی
اطّلاعات کامل	بازی‌های شطرنج، چِکِرز و go	بازی تخته نَرد[۶]
اطّلاعات ناقص	بازی سَنگ، کاغَذ، قِیچی[۷]	بازی‌های پل و پوکر

آشنایی با چند بازی[۸]

در اینجا با بازی‌هایی که در قسمت قبل نام آنها را بردیم، آشنا می‌شویم:

بازی شطرنج- بازی‌ای است که بر روی یک صفحه انجام می‌شود و برای دو بازیگر است که شانزده مهره را با توجّه به قوانینی معیّن حرکت می‌دهند. هدف، مات کردن مهره‌ی شاه طرف مقابل می‌باشد.

۱- Games of perfect information

۲- Games of imperfect information

۳- opposite

۴- bridge

۵- poker

۶- backgammon

۷- rock-paper-scissors

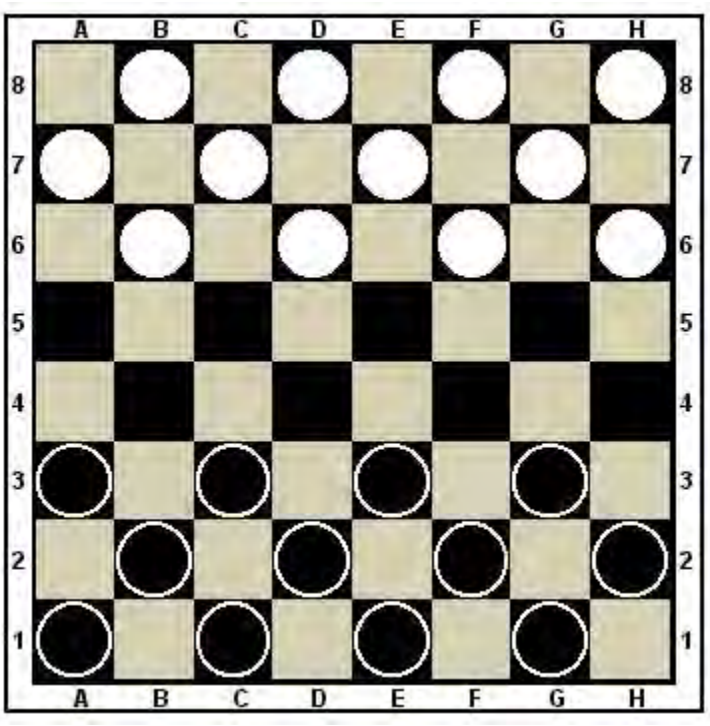
۸- برای آگاهی‌های بیش‌تر به فصل «آشنایی با هوش مصنوعی» همین کتاب الکترونیکی مراجعه کنید.

برای بازی با نسخه‌ای برخط از این بازی می‌توانید از آدرس اینترنتی زیر استفاده کنید:

http://en.lichess.org

در شکل زیر قسمتی از صفحه‌ی این بازی برخط را می‌بینید:

بازی چکِرز(جنگِ نادِر یا دام)- صفحه‌ی بازی چکرز برای دو نفـر می‌باشـد، کـه هـر نفـر دارای دوازده مهـره می‌باشد؛ هدف، زدن(از صفحه خارج کردن) مهره‌های حریف می‌باشد. در زیر تصویری از صفحه‌ی این بازی را می‌بینید:



برای بازی با نسخه‌ای برخط از بازی چکرز می‌توانید از آدرس اینترنتی زیر استفاده نمایید:

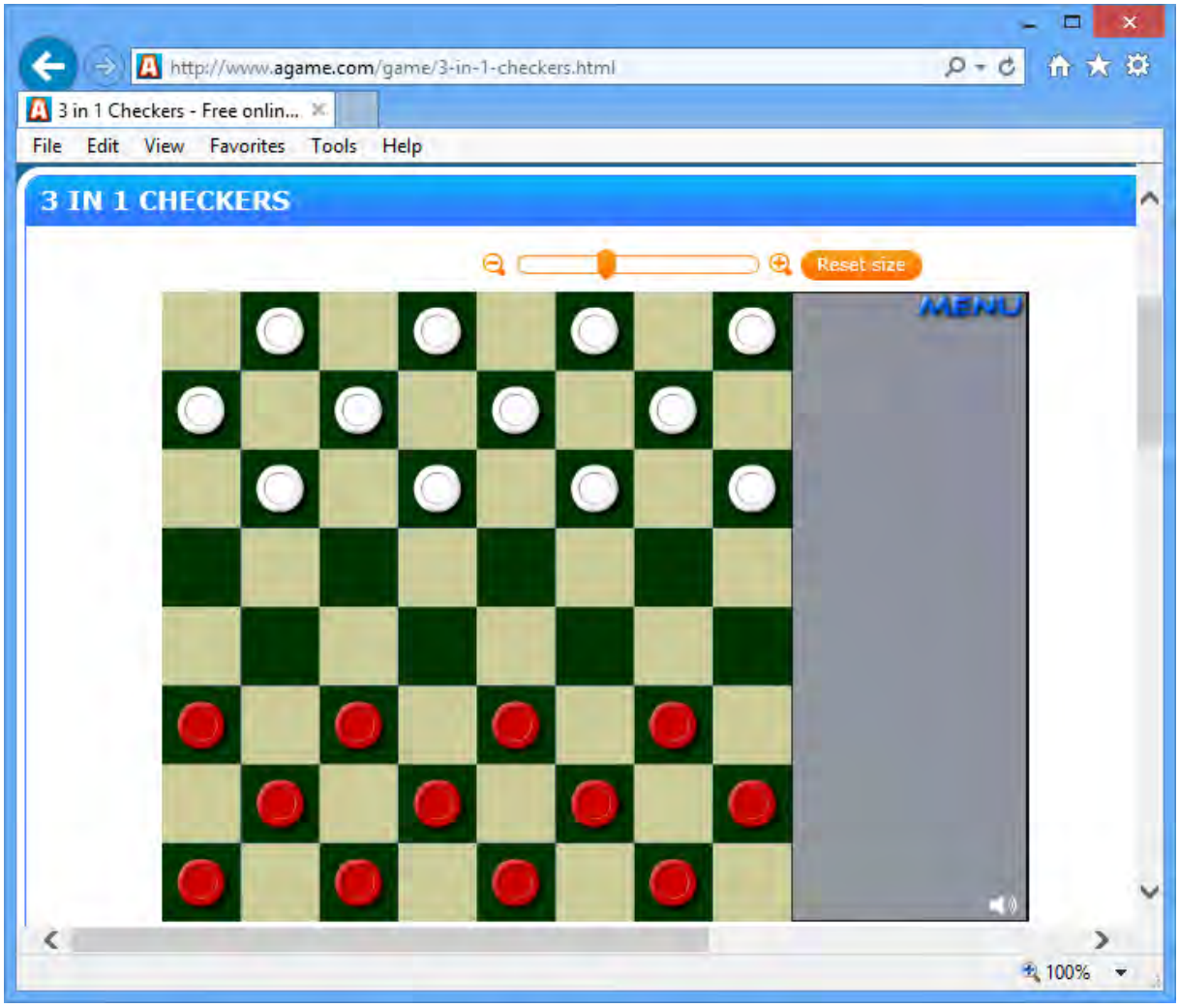
http://www.agame.com/games/checkers/checkers.html

توجّه:

برای اجرای این بازی برخط، نرم‌افزار Adobe Flash Player باید بـر روی کـامپیوتر شـما نصـب شـده باشد؛ چنانچه این نرم افزار را در اختیار ندارید، می‌توانیـد بـا اسـتفاده از آدرس اینترنتـی زیـر آن را دانلـود نمایید:

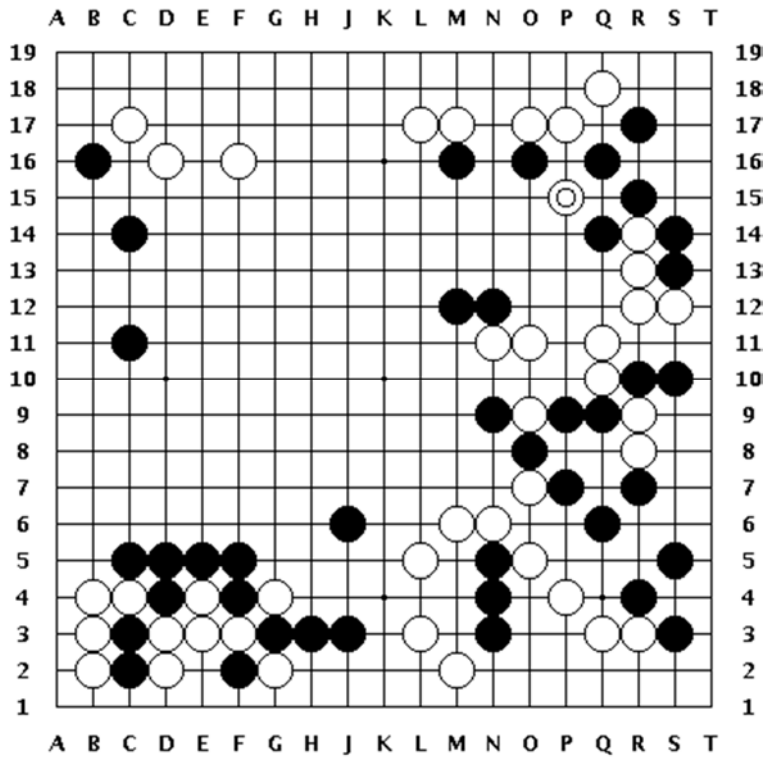
http://www.adobe.com/support/flashplayer/downloads.html

در شکل زیر قسمتی از صفحه‌ی این بازی برخط را می‌بینید:



بازی go- بازی‌ای بر روی صفحه برای دو بازیگر می‌باشد که مهره(سنگ)هایی را روی یک شبکه قرار می‌دهند، هدف، *محاصره کردن* مهره‌های حریف می‌باشد.[۱] در زیر تصویری از صفحه‌ی این بازی را می‌بینید:

۱- http://wordnet.princeton.edu/perl/webwn?s=chess%20game



برای بازی با نسخه‌ای برخط از این بازی می‌توانید از آدرس اینترنتی زیر استفاده نمایید:

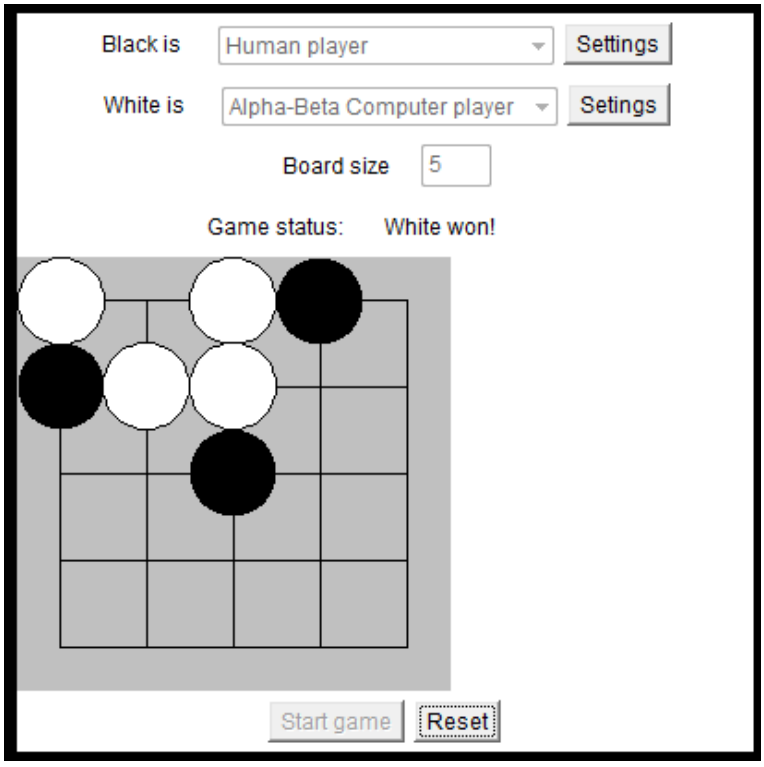
http://www.freegames.ws/games/boardgames/go/go.htm

توجّه:

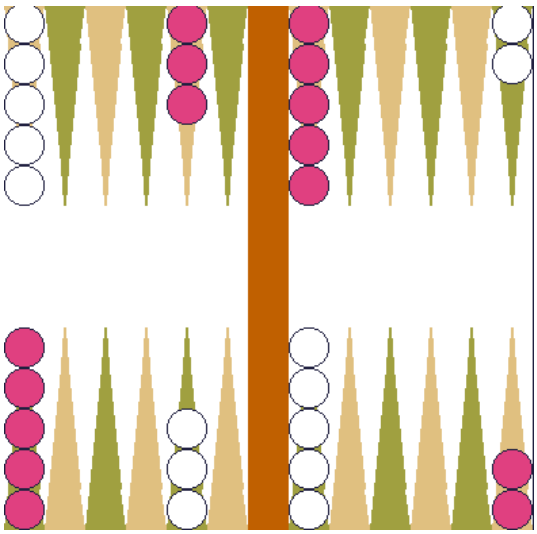
برای اجرای این بازی برخط، نرم‌افزار جاوا باید بر روی کامپیوتر شما نصب شده باشد؛ چنانچه این نرم افزار را در اختیار ندارید، می‌توانید با استفاده از آدرس اینترنتی زیر آن را دانلود نمایید:

http://www.java.com

در شکل زیر قسمتی از صفحه‌ی این بازی برخط را در شرایطی که مهره‌ی سفید، بازی را برده است، می‌بینید:

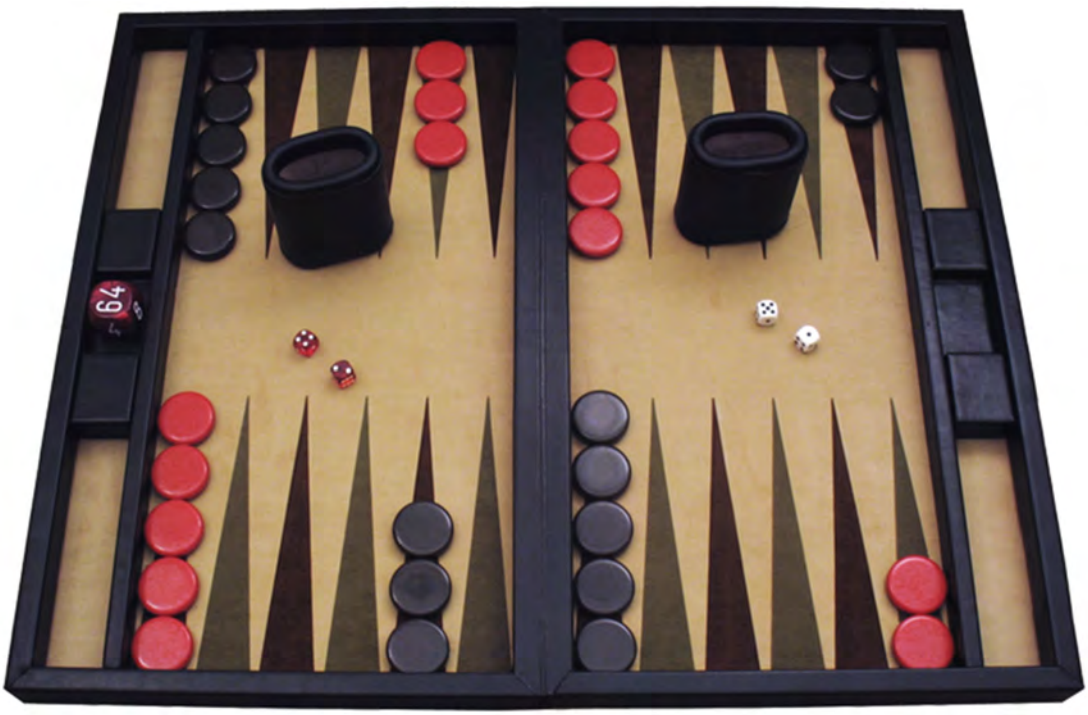


بازی تخته نرد– بازی‌ای بر روی صفحه است و دو بازیگر دارد. هر بازیگر دارای پانزده مهره می‌باشد، که آنها را در بیست و چهار خانه‌ی مثلّثی شکل با توجّه به عدد ظاهر شده روی دو تاس حرکت می‌دهد.[۱] در این بازی، هر کس که بتواند زودتر مهره‌هایش را از صفحه‌ی بازی خارج کند، برنده است.[۲] در زیر تصویرهایی از صفحه‌ی این بازی را می‌بینید:

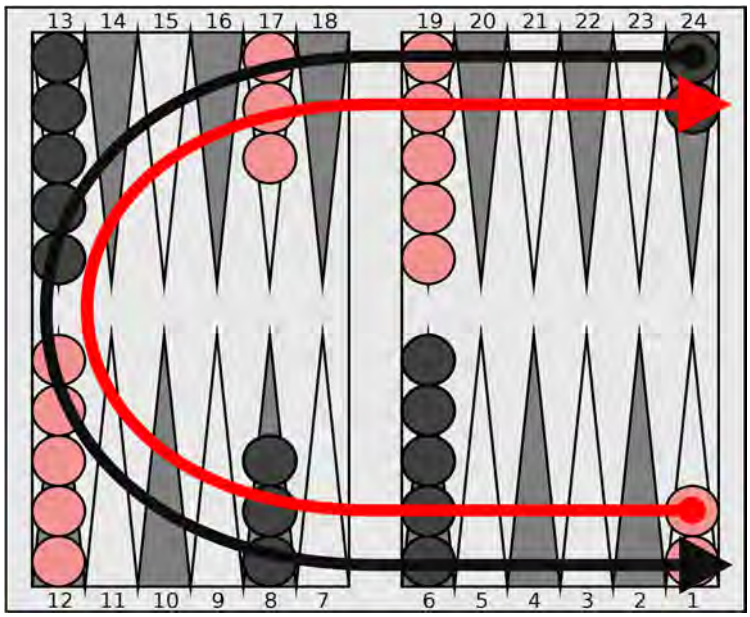


۱ – http://wordnet.princeton.edu/perl/webwn?s=chess%20game

۲ – Babylon> Britannica Concise Encyclopedia



در شکل زیر مسیر حرکت قرمز و سیاه نشان داده شده است:



برای بازی با نسخه‌ای برخط از این بازی می‌توانید از آدرس اینترنتی زیر استفاده نمایید:

http://games.aarp.org/games/backgammon.aspx

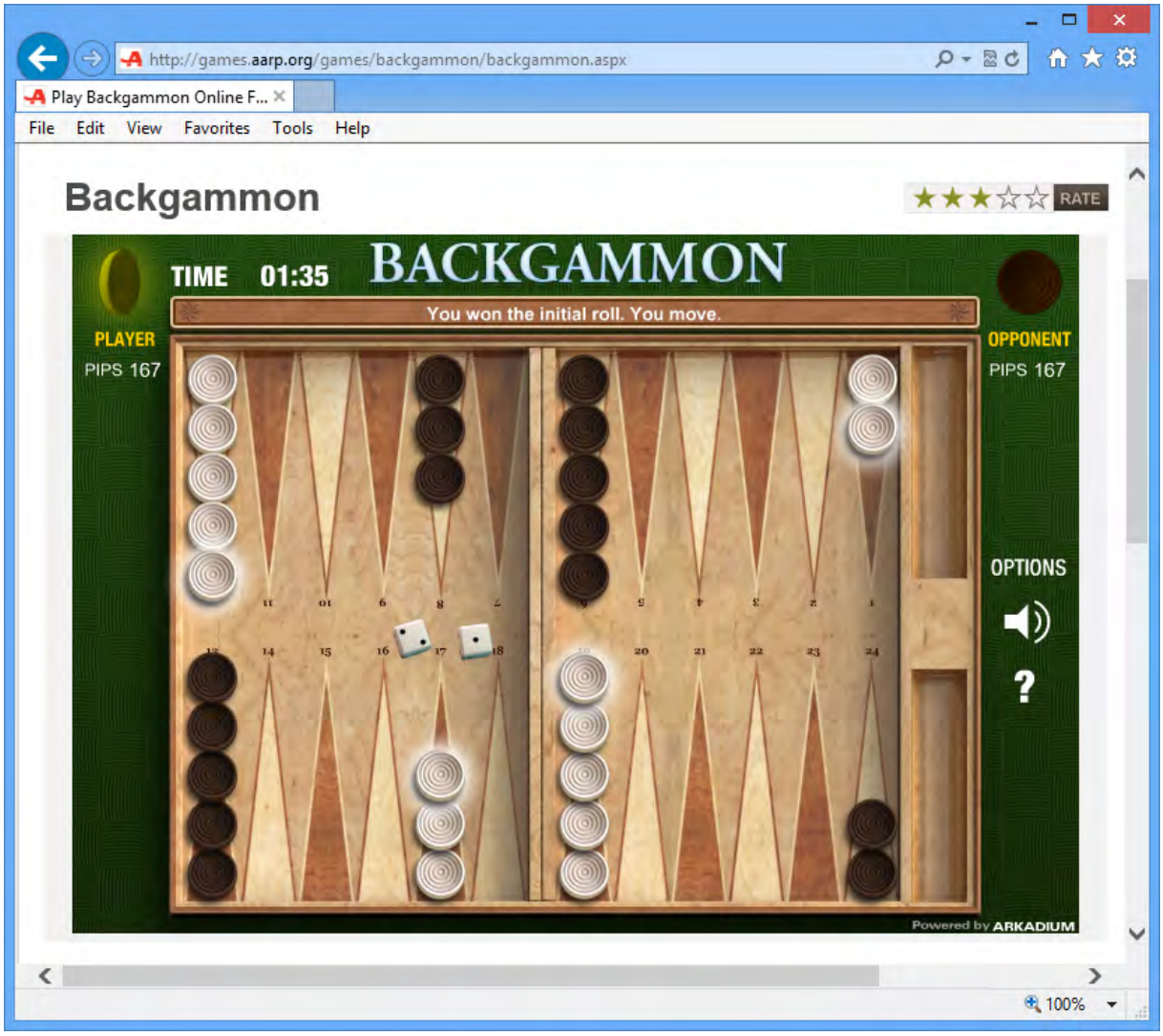
۱ - http://en.wikipedia.org/wiki/Backgammon

توجّه:

برای اجرای این بازی برخط، نرم‌افزار Adobe Flash Player باید بـر روی کـامپیوتر شـما نصـب شـده باشد؛ چنانچه این نرم افزار را در اختیار ندارید، می‌توانید بـا اسـتفاده از آدرس اینترنتـی زیـر آن را دانلـود نمایید:

http://www.adobe.com/support/flashplayer/downloads.html

در شکل زیر قسمتی از صفحه‌ی این بازی برخط را می‌بینید:



بازی سنگ، کاغذ، قیچی– این بازی معمولاً دو نفره است؛ در این بازی، بازیکن‌ها یک دست خود را به صورت مُشت بالا برده، سپس از عدد یک تا سه می‌شمرند یا می گویند: «سنگ، کاغذ، قیچی»؛ وقتی که شمارش(گفتن) تمام شد، بازیکن‌ها دست خود را پایین آورده و به یکی از سه صورت سنگ، کاغذ، یا قیچی که در زیر نشان داده شده است، در می‌آورند:


قیچی


کاغذ


سنگ

برنده با توجّه به نمودار زیر تعیین می‌گردد:

مثلاً اگر دست یک بازیکن به شکل قیچی بود و دست بازیکن دیگر به شکل سنگ بود، بازیکنی که دستش به صورت سنگ است، برنده خواهد بود.[۱]

اگر حالت دست‌ها یکی بود، مثلاً هر دو دست به شکل کاغذ بود، دوباره بازی به همین صورت آنقدر تکرار می‌شود تا حالت دست‌ها با هم فرق کند(یکی نباشد).

بازی پل- یک بازی کارتی حقّه‌ای برای چهار نفر می‌باشد که این چهار نفر به صورت دو گروه دو نفری با هم بازی می‌کنند و در هر طرف، هر گروه مقابل هم می‌نشیند. بازی پل دارای دو مرحله می‌باشد: پیشنهاد و بازی.[۲]

---

۱ - http://en.wikipedia.org/wiki/Rock-paper-scissors

۲ - http://en.wikipedia.org/wiki/Bridge_game

شکل بالا– تصویری از بازی پل

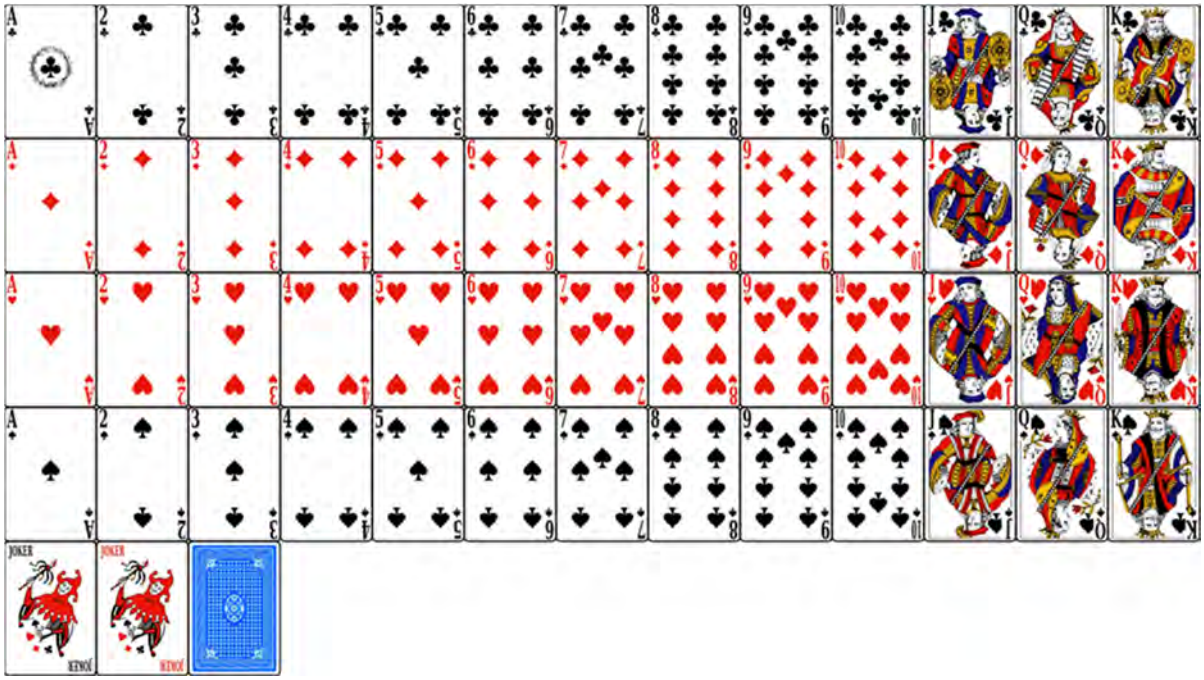
بازی پوکر– بازی‌ای کارتی می‌باشد، محبوب‌ترین بازی از یک دسته از بازی‌ها به نام بازی‌های همچشمی می‌باشد که در آن، بازیگران با کارت‌هایی کاملاً پنهان یا اندکی پنهان روی یک چیزی شرط‌بندی می‌کنند، سپس چیزی که شرط‌بندی روی آن انجام شده است، به بازیگر یا بازیگران باقی مانده‌ای که دارای بهترین ترکیب کارت‌ها هستند، جایزه داده می‌شود.[۱]

۱ – http://en.wikipedia.org/wiki/Poker

شکل بالا- بازی پوکر

شکل بالا- تصویر دیگری از بازی پوکر

در زیر تصویری از کارت‌های دو بازی پل و پوکر را می‌بینید:



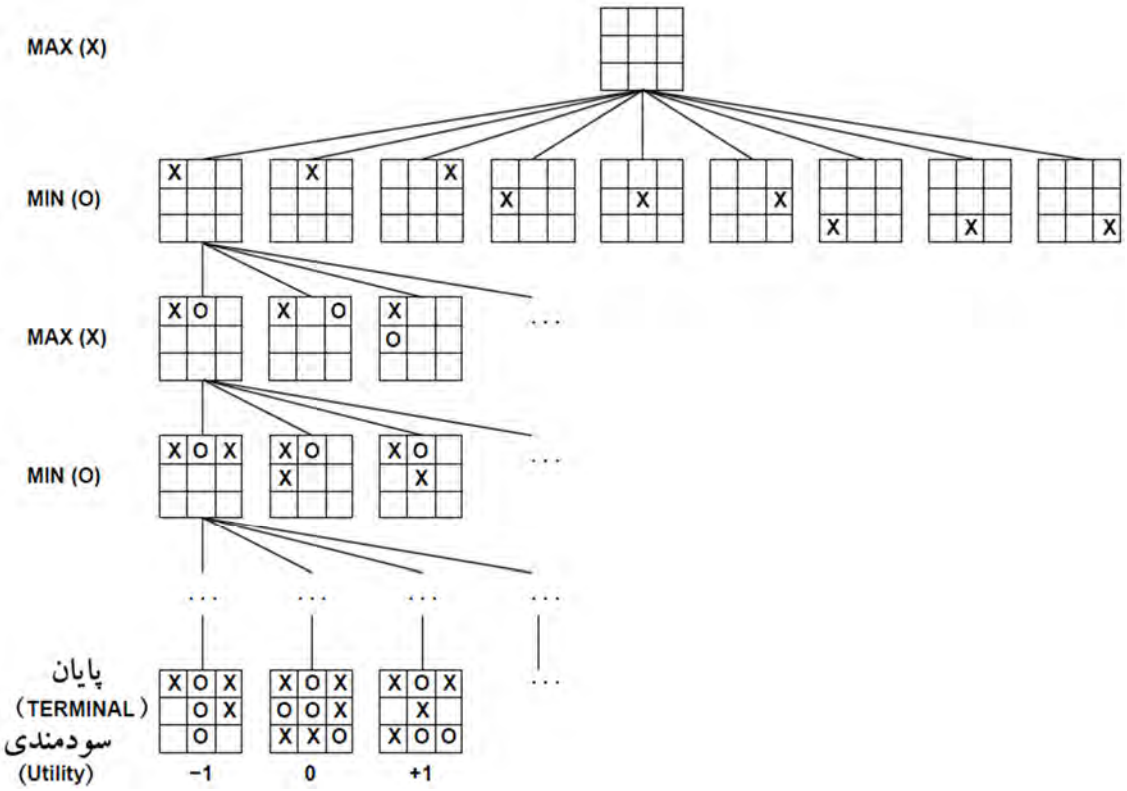
بـــرای اطّلاعـــات بیش‌تـــر می‌توانیـــد بـــه گـــروه بـــازی دانشـــگاه آلبِرتـــای کشـــور کانـــادا، بـــا آدرس اینترنتـــی http://www.cs.ualberta.ca/~games/ مراجعه نمایید.

تئوری بازی‌ها به صورت جستجو

یک بازی دو نفره‌ی با اطّلاعات کامل را در نظر بگیرید؛ آیا می‌توانیم این بازی را به صورت مسأله‌های جستجو فرمول‌بندی نماییم؟؛ برای این کار، وضعیّت صفحه‌ی بازی، وضعیّت جستجوی مسأله می‌باشد؛ حرکت‌های مُجاز، عملگرها هستند؛ و وضعیّت‌های پایانی، وضعیّت‌هایی هستند که درآنها بازی را برده‌ایم یا بازنده شده‌ایم یا بازی بی‌نتیجه مانده است. می‌توانیم به بردن، عدد ۱+؛ به باختن، عدد ۱- و به بازی بی‌نتیجه مانده، عدد ۰(صفر) را نسبت دهیم. ما می‌خواهیم یک روش یا یک راه را برای انتخاب حرکت‌هایی که بازی را می‌برد، پیدا نماییم؛ ما باید تصمیم‌گیری حریفمان را شبیه‌سازی نماییم؛ توجّه کنید که یک بازیگر بیشینه وجود دارد که بیش‌ترین نفع را برای خودش می‌خواهد و یک بازیگر کمینه وجود دارد که کم‌ترین نفع را برای خودش می‌خواهد.

درخت بازی

در درخت بازی، ریشه‌ی درخت، وضعیّت اوّلیّه است؛ سطح بعدی، تمام حرکت‌های MAX[1] است؛ سطح بعدی، تمام حرکت‌های MIN[2] است و ...؛ به عنوان مثال، در درخت بازی تیک – تاک – تو[3]، ریشه، دارای ۹ خانه‌ی خالی است (MAX)؛ سطح یک، دارای ۸ خانه‌ی خالی است (MIN)؛ سطح دو، دارای ۷ خانه‌ی خالی است (MAX) و .... در تابع سودمندی، بردن برای × دارای مقدار ۱+ است و بردن برای O دارای مقدار ۱- است. درخت زیر برای بازی تیک – تاک –-تو رسم شده است:



شکل بالا– قسمتی از درخت بازی برای بازی تیک – تاک – تو. بالاترین گره، وضعیّت آغازین(اوّلیّه) است و MAX اوّل حرکت می‌کند و یک X را در یک خانه‌ی خالی قرار می‌دهد. ما بخشی از درخت را نشان داده‌ایم، با حرکت‌های متناوب MIN(O) و MAX(X) سرانجام به وضعیّت‌های پایانی می‌رسیم که می‌تواند با توجّه به قانون‌های بازی به آنها سودمندی نسبت داده شود.[4]

---

۱ – در ادامه‌ی همین فصل با مفهوم این کلمه آشنا خواهید شد.

۲ – در ادامه‌ی همین فصل با مفهوم این کلمه آشنا خواهید شد.

۳ – برای آگاهی‌های بیش‌تر در مورد این بازی، به فصل «آشنایی با هوش مصنوعی» همین کتاب الکترونیکی مراجعه کنید.

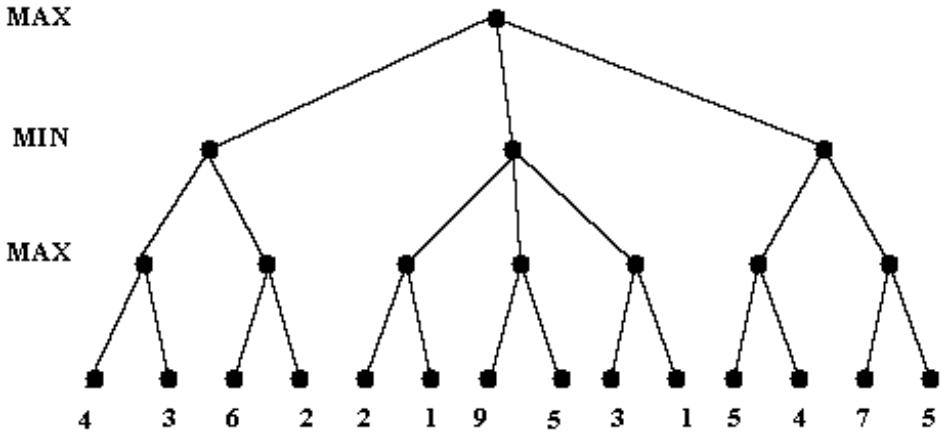
۴ – کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل پنجم، جستجوی خصمانه(Adversarial Search)، صفحه‌ی ۱۶۳

مینیماکس[۱]

در روش مینیماکس، از پردازه‌ی جستجو برای پیدا کردن مسیر راه حل استفاده نمی‌کنیم، بلکه *برای به دست آوردن دقیق‌ترین ارزیابی حرکت‌های ممکن استفاده می‌کنیم.*[۲]

MIN و MAX دو بازیگر هستند که MAX می‌خواهد بازی را ببرد و MIN هم می‌خواهد بازی را ببرد و در واقع MAX و MIN رقیب هم هستند و هر دو بهترین بازی ممکن را انجام می‌دهند.

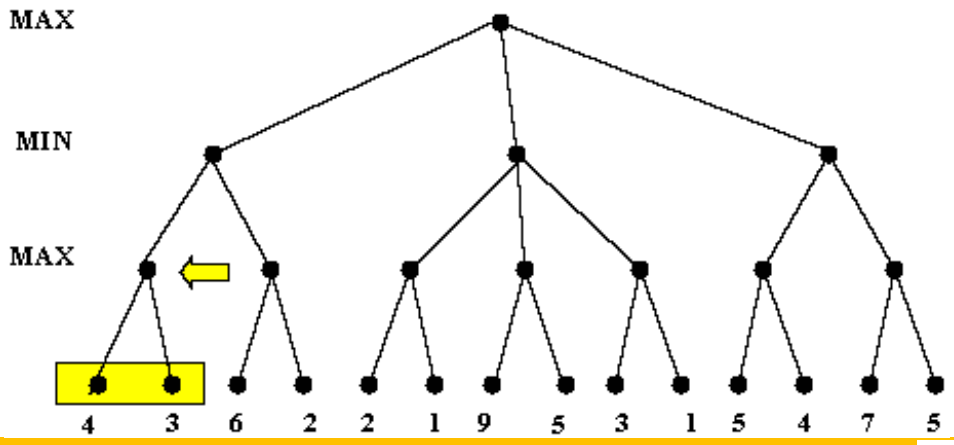
مثال زیر گویای این روش است:



---

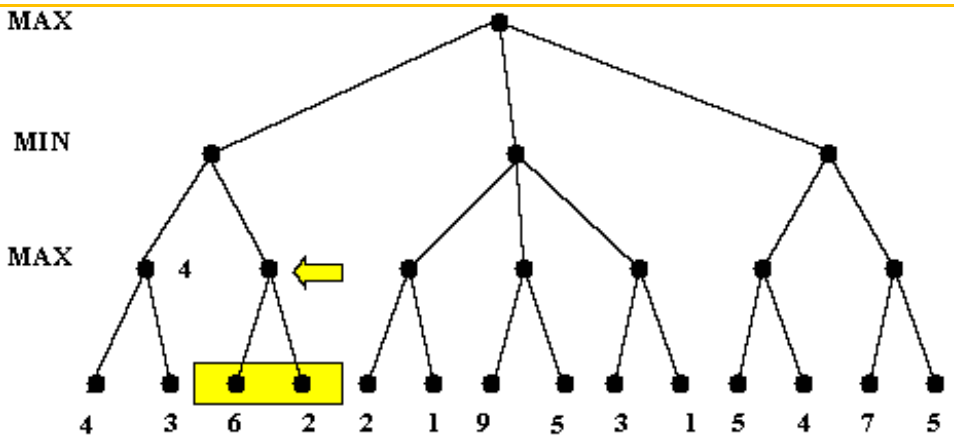
۱ - Minimax

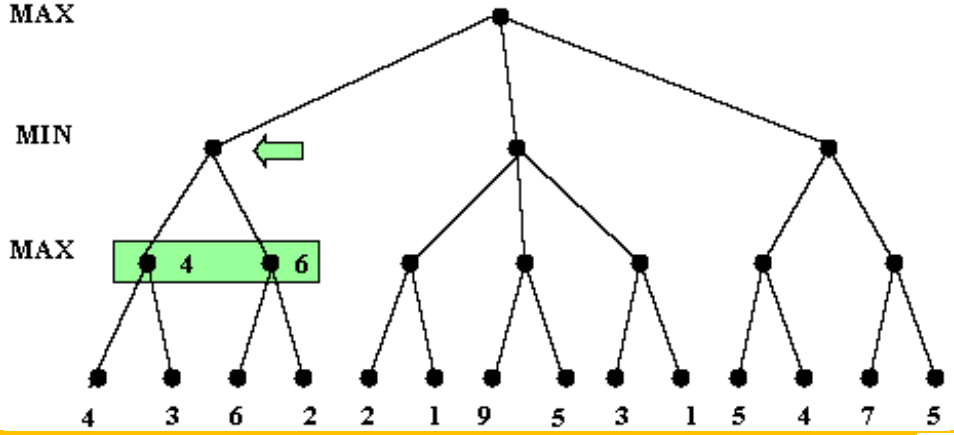
۲ - http://www.informatics.sussex.ac.uk/courses/kr/lec05.html (این پایگاه اینترنتی، متعلّق به دانشگاه ساسِکْس کشور انگلیس است. ساسکس، نام شهری در جنوب شرق کشور انگلیس بوده است و در حال حاضر به دو قسمت ساسکس شرقی و ساسکس غربی تقسیم شده است. (Babylon> Babylon English))



توضیح:

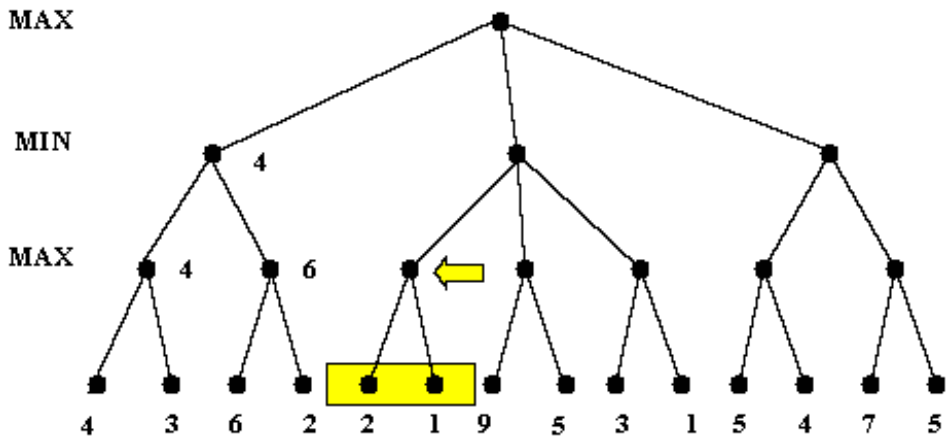
همان طور که در شکل بالا می‌بینید، بیش‌ترین مقدار دو گره‌ی مشخّص شده، ۴ است، پس همان طور که در شکل بعدی می‌بینید، عدد ۴ انتخاب شده است:

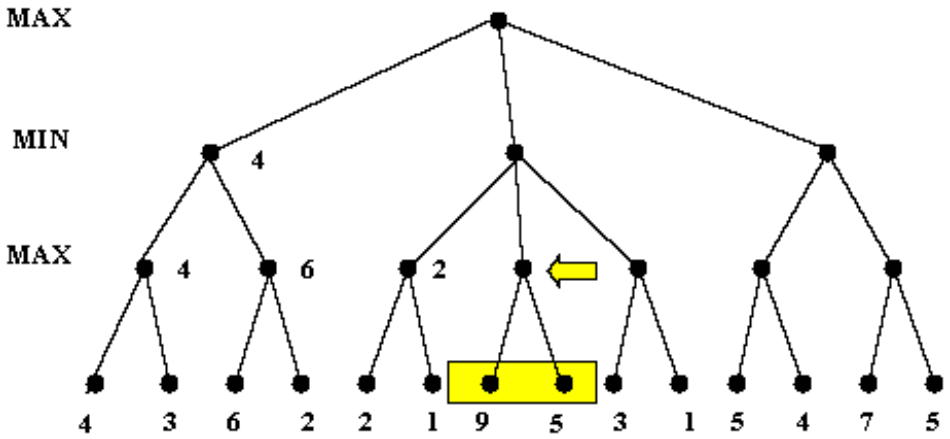


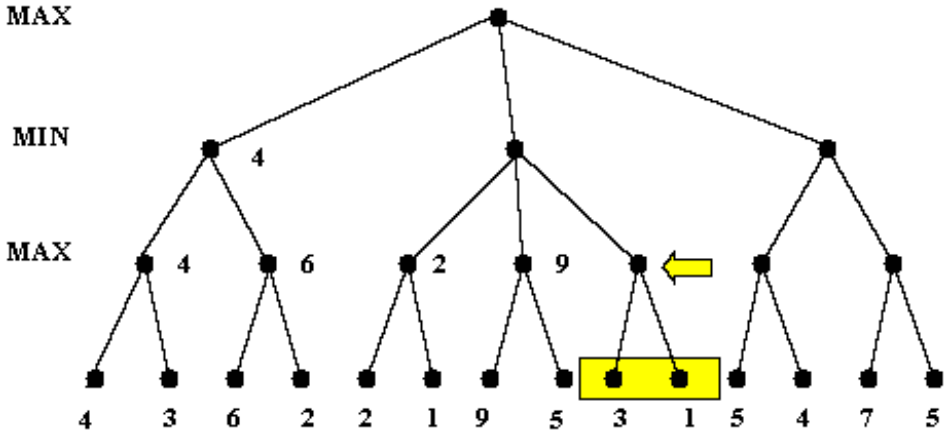


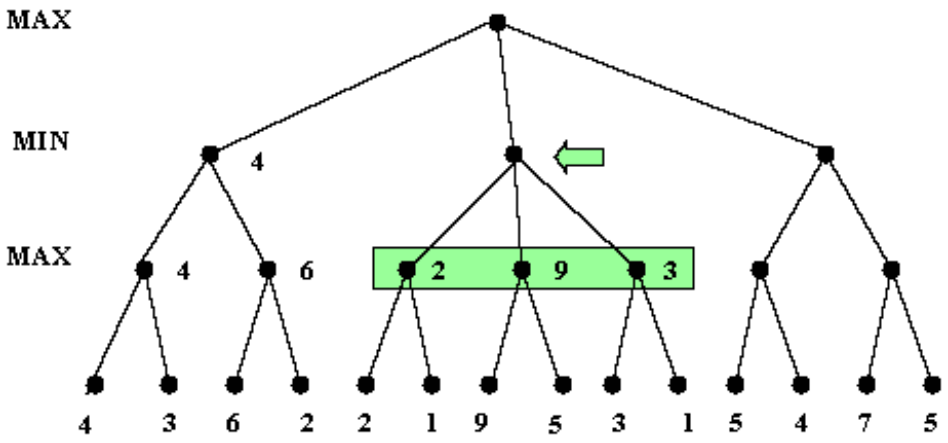
توضیح:

همان طور که در شکل قبلی می‌بینید، کم‌ترین مقدار دو گره‌ی مشخّص شده، ۴ است، پس همان طور که در شکل بعدی می‌بینید، عدد ۴ انتخاب شده است:

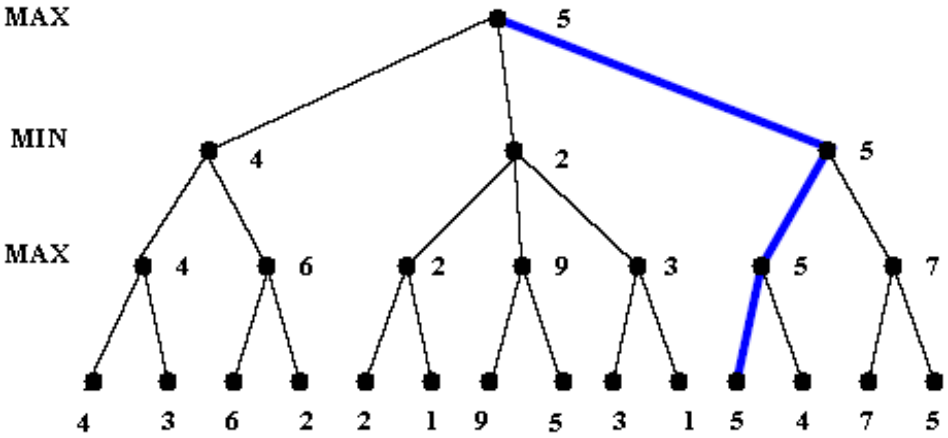

MAX
MIN
MAX
4
4
6
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
4
4
6
2
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
4
4
6
2
9
4 3 6 2 2 1 9 5 3 1 5 4 7 5

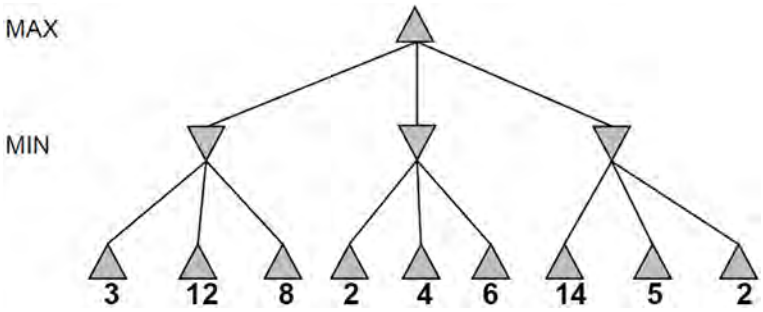


و به همین ترتیب موردهای دیگر را هم به دست می‌آوریم تا در نهایت به شکل زیر برسیم:

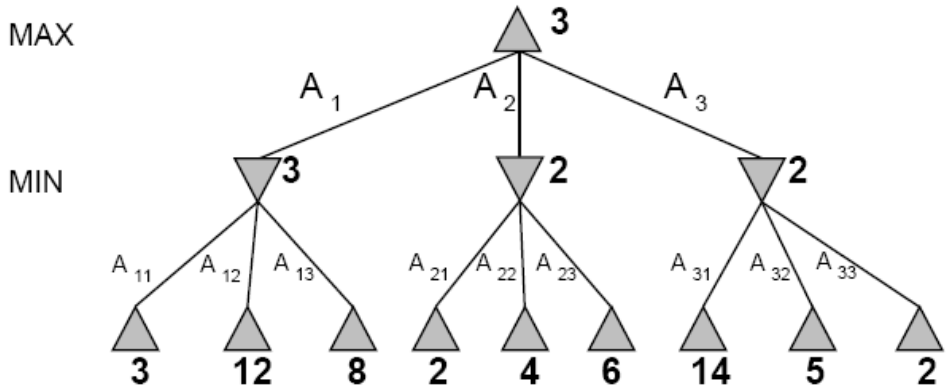


پس جواب این مثال، عدد ۵ است.

به عنوان مثالی دیگر؛ برای بازی‌ای دو نفره داریم:



که در نتیجه با توجّه به روشی که برای مثال قبلی دیدیم، در شکل بعدی، حاصل برابر با ۳ به دست آمده است:



الگوریتم مینیماکس

الگوریتم

تابع Minimax-Decision(state)، یک عملکرد را برمی‌گرداند

ورودی: state، که حالت جاری بازی می‌باشد

aای را که در Actions(state) بیشینه کننده‌ی Min-Value(Result(a, state)) می‌باشد را برگردان

تابع Max-Value(state)، یک مقدار سودمند را برمی‌گرداند

در صورتی که تابع Terminal-Test(state) دارای مقدار بازگشتی درست می‌باشد، Utility(state) را برگردان

$v \leftarrow -\infty$

برای a,s در Successor(state)، $v \leftarrow Max\ (v, Min\text{-}Value(s))$

v را برگردان

تابع Min-Value(state)، یک مقدار سودمند را برمی‌گرداند

در صورتی که Terminal-Test(state) درست است، Utility(state) را برگردان

$v \leftarrow \infty$

برای a,s درون Successors(state)، v ← Min(v, Max-Value(s))

v را برگردان

پایان الگوریتم

ویژگی‌های مینیماکس

کامل بودن؟؟ فقط در صورتی که درخت محدود باشد، کامل است.

بهینه بودن؟؟ بله

پیچیدگی زمانی؟؟ $O(b^m)$

پیچیدگی فضا؟؟ O(bm) (مثل جستجوی اوّل عمق)

برای شطرنج، m ≈100 و b ≈ 35 می‌باشد، در نتیجه راه حلّ دقیق، کاملاً اجرا نشدنی است. *امّا آیا ما احتیاج داریم که همه‌ی مسیرها را جستجو(پیدا یا کاوش) نماییم؟!:*

هَرَس(بازبینی) α- β[1](آلفا–بتا)

شکل بالا– هرس: در باغبانی، شاخه‌های اضافی درختان را می‌برند؛ ما هم می‌توانیم شاخه‌های اضافی درخت بازی را که مورد استفاده قرار نمی‌گیرند، حذف نماییم.

در سال ۱۹۵۶ میلادی، بازبینی یا هرس برای اجازه دادن به جستجوی عمیق‌تر، به وسیله‌ی استاد، جان مِک‌کارتی ارائه شد، که روشی استاندارد برای بازی‌های قطعی و با اطّلاعات کامل می‌باشد؛

مطلب مهم:

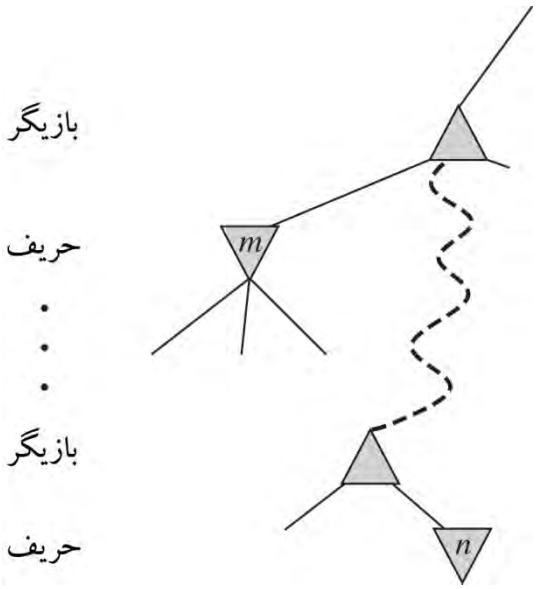
در این روش، شاخه‌ای که هیچوقت مورد استفاده قرار نمی‌گیرد را از درخت جستجو حذف می‌کنیم؛

توجّه کنید که این شاخه‌ها هیچوقت به وسیله‌ی یک بازیکن عاقل مورد استفاده قرار نمی‌گیرند، در ضمن، به وسیله‌ی بازیکن حریف هم مورد استفاده قرار نمی‌گیرند.

---

۱ – α-β pruning

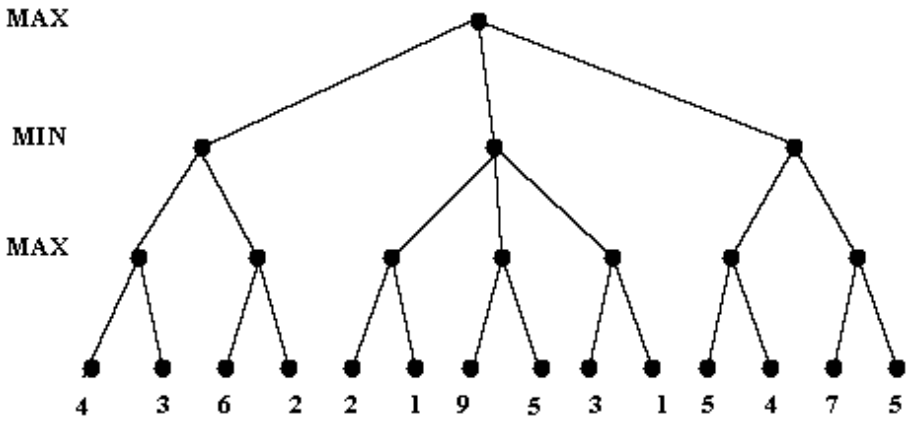
شکل بالا– تصویری از استاد، جان مک‌کارتی[۱]

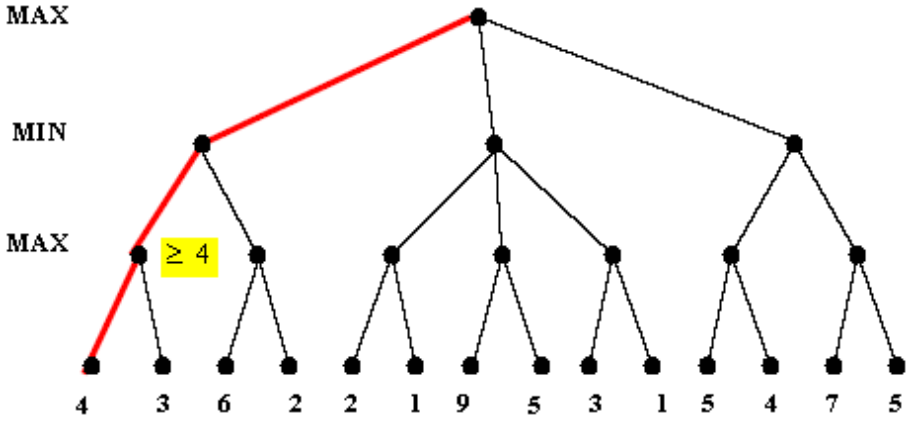


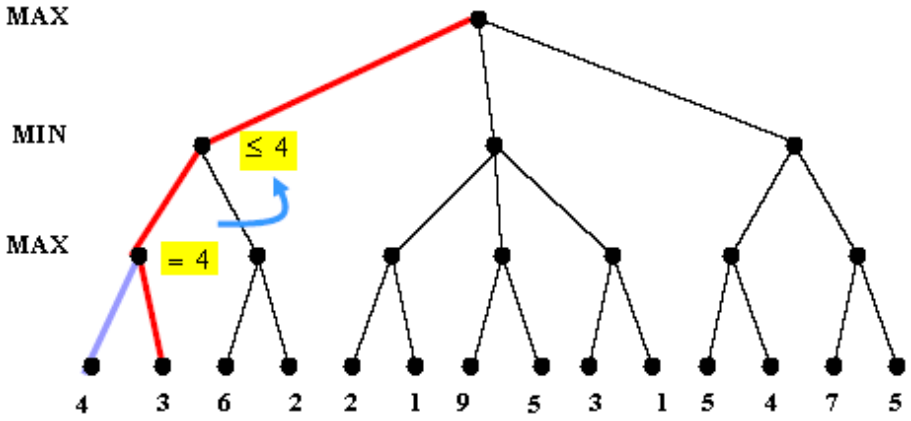
شکل بالا– حالت کلّی هرس آلفا-بتا. اگر m، برای بازیگر، بهتر از n باشد، هیچوقت در بازی به n نخواهیم رفت.

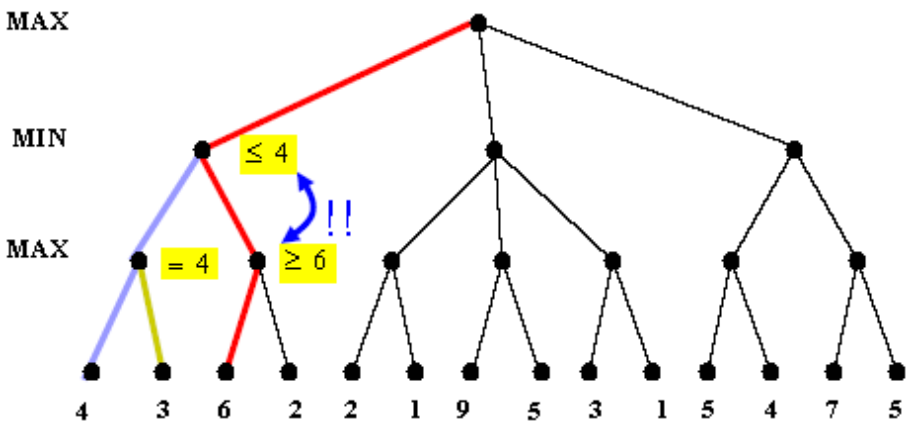
۱ - http://infolab.stanford.edu/pub/voy/museum/pictures/AIlab/3500McCarthy.JPG

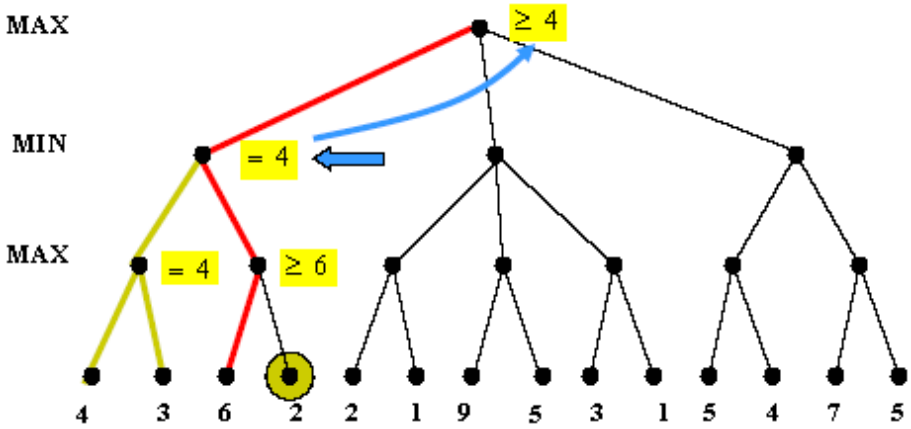
مثال هرس α-β: به درخت زیر توجّه کنید و ***شکل‌ها را بادقّت دنبال نمائید.***



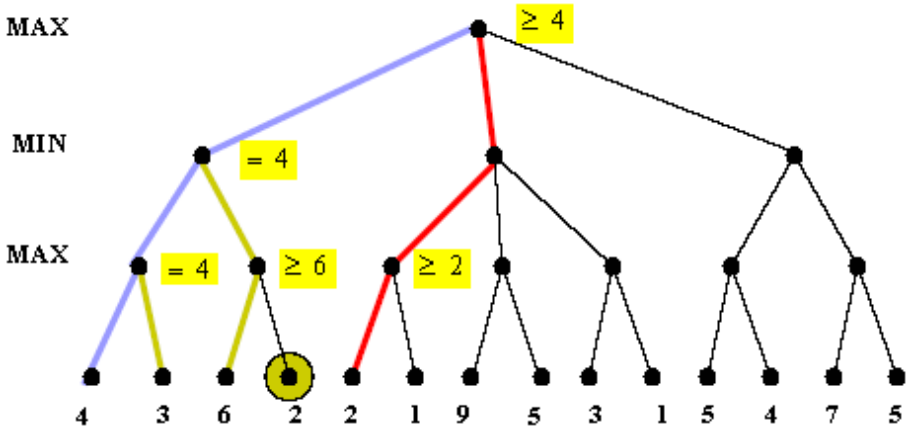


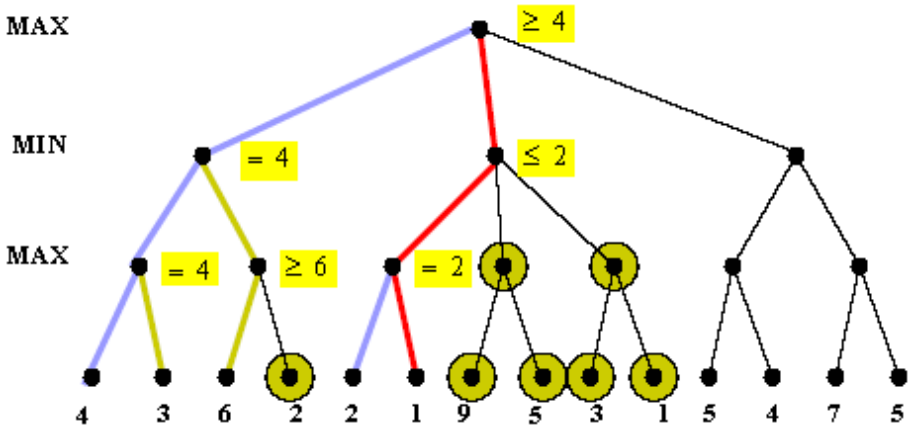


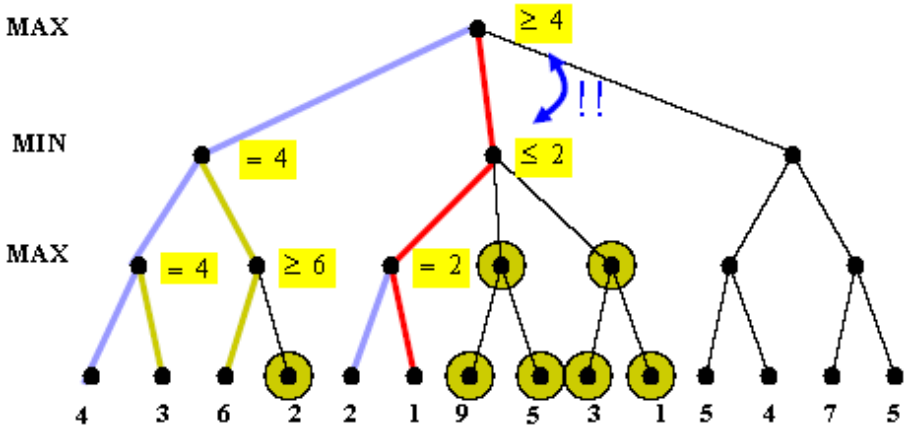


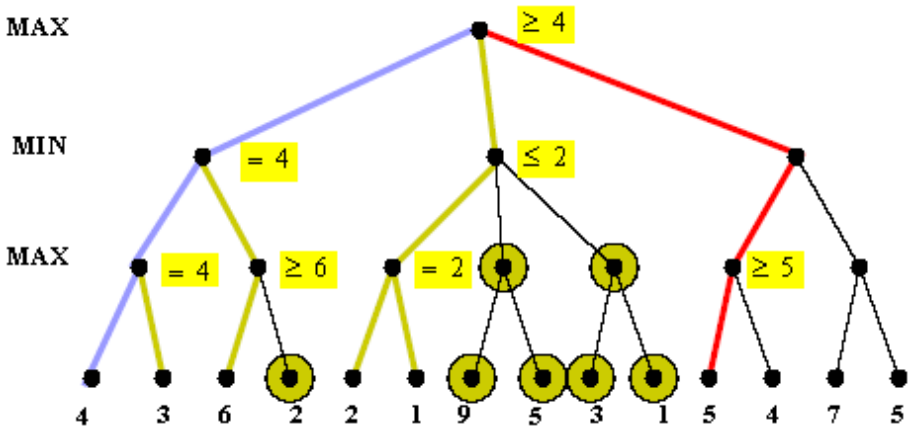


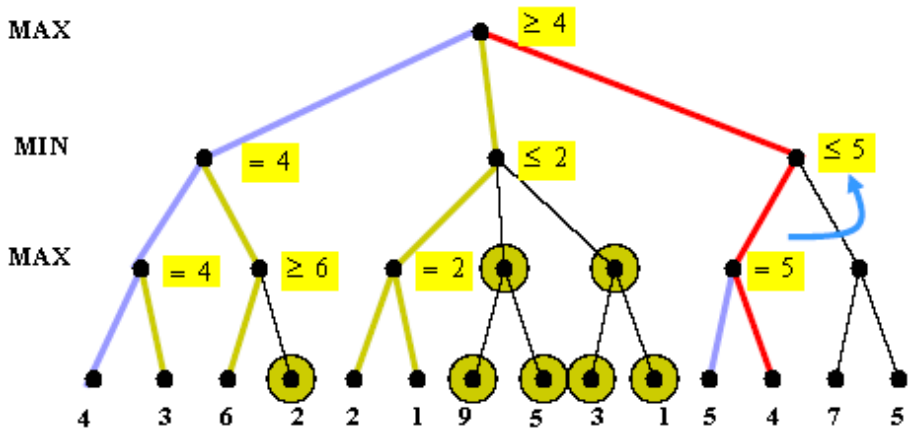
توجّه کنید که گره‌های به شکل ◉ هیچوقت مورد استفاده قرار نمی‌گیرند.

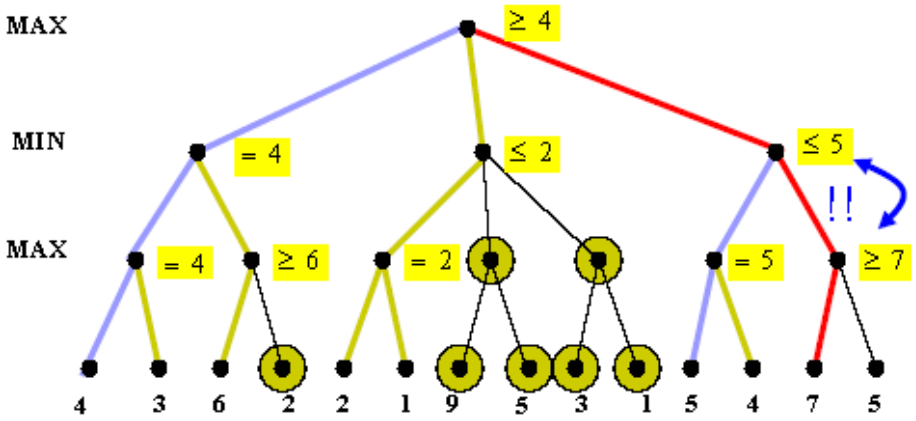


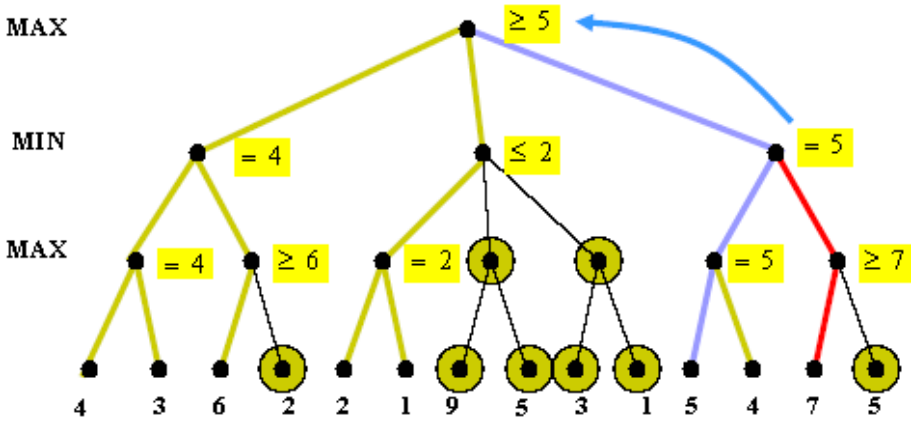

MAX
MIN
MAX
≥ 4
= 4
≤ 2
= 4
≥ 6
= 2
4 3 6 2 2 1 9 5 3 1 5 4 7 5

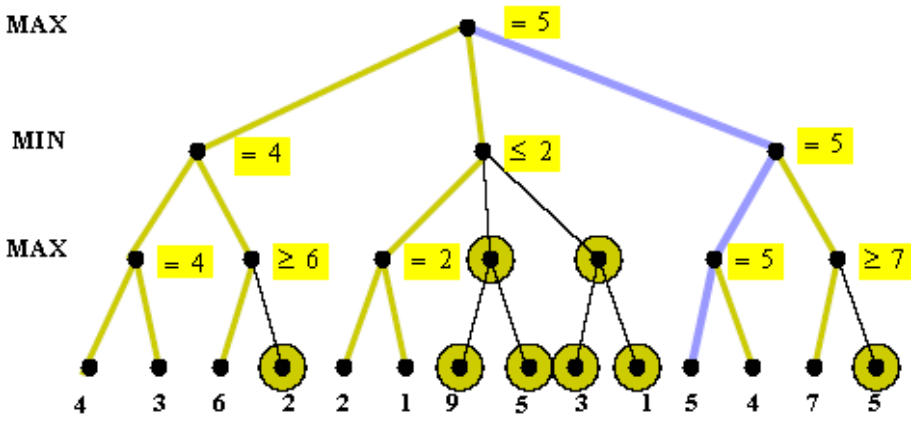

MAX
MIN
MAX
≥ 4
!!
= 4
≤ 2
= 4
≥ 6
= 2
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
≥ 4
= 4
≤ 2
= 4
≥ 6
= 2
≥ 5
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
≥ 4
= 4
≤ 2
≤ 5
= 4
≥ 6
= 2
= 5
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
≥ 4
= 4
≤ 2
≤ 5
!!
= 4
≥ 6
= 2
= 5
≥ 7
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
≥ 5
= 4
≤ 2
= 5
= 4
≥ 6
= 2
= 5
≥ 7
4 3 6 2 2 1 9 5 3 1 5 4 7 5



الگوریتم α-β

الگوریتم

تابع Alpha-Beta-Decision(state)، یک عمل را برمی‌گرداند

aای را که در Actions(state)، بیشینه کننده‌ی Min-Value(Result(a,state)) می‌باشد را برگردان

تابع Max-Value(state,α,β)، یک مقدار سودمند را برمی‌گرداند

ورودی‌ها، state، که حالت جاری در بازی می‌باشد

α مقدار بهترین برای MAX در طول مسیر به طرف state می‌باشد

β مقدار بهترین برای MIN در طول مسیر به طرف state می‌باشد

در صورتی که Terminal-Test(state) درست می‌باشد، Utility(state) را برگردان

v← - ∞

برای a,s در تابع Successors(state) کارهای زیر را انجام بده

v ← Max(v,Min-Value(s,α,β))

در صورتی که v ≥β است، v را برگردان

α←Max(α,v)

انتهای حلقه

v را برگردان

تابع Min-Value(state,α,β)، یک مقدار سودمند را برمی‌گرداند

ورودی‌ها، state، که حالت جاری در بازی می‌باشد

α مقدار بهترین برای MAX در طول مسیر به طرف state می‌باشد

β مقدار بهترین برای MIN در طول مسیر به طرف state می‌باشد

در صورتی که Terminal-Test(state) درست می‌باشد، Utility(state) را برگردان

v← + ∞

برای a,s در تابع Successors(state) کارهای زیر را انجام بده

v ← Min(v,Max-Value(s,α,β))

در صورتی که v ≤β است، v را برگردان

β←Max(β,v)

انتهای حلقه

v را برگردان

پایان الگوریتم

ویژگی‌های α-β

مطلب مهم:

بازبینی یا هرس بر روی نتیجه‌ی نهائی اثری ندارد. «چینشِ حرکت(روش ترتیب‌دهی)»[1] خوب، کارآیی(اثر) هرس را بهبود می‌بخشد.

[اگر در هرس، ترتیب‌دهی می‌توانست به طور کامل انجام شود،] پیچیدگی زمانی برابر با $O(b^{m/2})$ می‌شد، درنتیجه عمق جستجو دوبرابر(مضاعف) می‌شد. بنابراین، به‌سادگی [در درخت جستجو] می‌توانست به عمق هشت(۸) برسد و بازی شطرنج خوبی را ارائه نماید. متأسّفانه ۳۵۵۰ هنوز غیرممکن است!.

---

۱ - move ordering؛ کارآیی هرس آلفا-بتا به شدّت وابسته به ترتیبی است که در آن، وضعیّت‌ها برّرسی می‌شوند. (کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل پنجم، جستجوی خصمانه(Adversarial Search)، صفحه‌ی ۱۶۹)

تابع ارزیابی

همان طور که قبلاً هم داشتیم، یک تابع ارزیابی، تابعی است که خوبی یک وضعیّت را انـدازه‌گیری می‌نمایـد (بـه عنـوان مثال، شانس برنده شدن با آن وضعیّت را اندازه می‌گیرد) و این تابع می‌تواند به وسیله‌ی طرّاح ارائه شود، یا با آزمایش به دسـت‌آید. مثلاً برای بازی شطرنج، این تابع می‌تواند بـه صـورت $w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s)$ باشـد، کـه s، وضـعیّت صفحه‌ی بازی می‌باشد و f_i، تعداد(فراوانی) یک نوع مهره (مثلاً مهره‌ی فیل[۱]) بر روی صفحه‌ی بازی می‌باشـد و w_i، ارزش مهره‌هـا می‌باشد (مثلاً عدد یک، برای مهره‌ی سرباز[۲] و عدد سه، برای مهره‌ی فیل). توجّه کنید که برنامه‌های پیشرفته از ترکیبـات غیرخطّـی هم استفاده می‌نمایند، در ضمن، ترکیبات و توزیع‌ها جزئی از قوانین شطرنج نمی‌باشند.

بازی‌های قطعی در عمل[۳]

بازی چِکِرز

کامپیوتر چینوک[۴]، به چهل سال قهرمانی قهرمان جهان، مِریون تینزلی[۵] در سال ۱۹۹۴ مـیلادی خاتمـه داد؛ چینـوک، نـام نخستین کامپیوتری بود که توانست از قهرمان جهان ببرد؛ برای بازی با نسخه‌ای از چینوک، در اینترنت از آدرس زیر استفاده نمایید:

http://www.cs.ualberta.ca/~chinook

در زیر تصویری از مِریون تینزلی را می‌بینید:

---

۱ - Bishop:

۲ - Pawn:

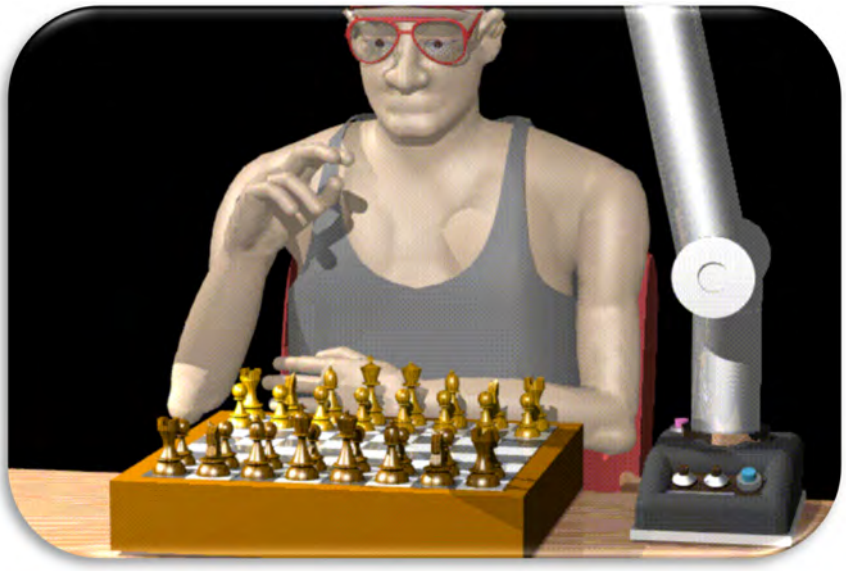
۳ - deterministic games in practice

۴ - Chinook

۵ - Marion Tinsley، ۱۹۹۵ – ۱۹۲۷ میلادی (http://en.wikipedia.org/wiki/Marion_Tinsley)، حرفه‌ی وی درس دادن ریّاضیّات و سرگرمی او بازی چکرز بود.

بازی شطرنج

میانگین فاکتور انشعاب در شطرنج، ۳۵ می‌باشد و فضای حالت در آن، حدود $۱۰^{۷۰}$ می‌باشد.



اَبَررایانه[1]ی Deep Blue یک مثال خوب از ترکیب توان محاسباتی مدرن و تکنیک‌ها(روش‌ها)ی هوش مصنوعی می‌باشد؛ در آن مقدار زیادی از دانش افراد برای به وجود آوردن دقّت در توابع ارزیابی به کار گرفته شد و از پایگاه داده‌های استادان بزرگ شطرنج در بازی‌های قبلی و همچنین از تعداد زیادی کتاب‌های پیشرفته استفاده شده بود. Deep Blue قهرمان شطرنج جهان، گری کاسپارُوف را در شش بازی، در سال ۱۹۹۷ میلادی شکست داد و این کار، یک هدف هوش مصنوعی از دهه‌ی ۵۰ میلادی بود. برنامه‌ی Deep Blue به زبان برنامه‌نویسی سی نوشته شده بود و از روش جستجوی آلفا–بتا استفاده می‌نمود و دارای سخت‌افزار مخصوص محاسبه کننده‌ی توابع ارزیابی بود. Deep Blue دویست میلیون حالت را در هر ثانیه جستجو می‌کرد؛ که ۱۰۰– ۲۰۰ بیلیون[2] وضعیّت را در سه دقیقه ارزیابی می‌کند؛ از یک ارزیابی خیلی ماهرانه استفاده می‌کرد و از متدهای فاش نشده برای توسعه دادن تعدادی از خطوط جستجو تا ۴۰ تا استفاده می‌نمود. در زیر تصویری از مسابقه‌ی گری کاسپاروف و Deep Blue را می‌بینید:

در زیر تصویری از Deep Blue را می‌بینید:

---

۱ – یا اَبَر کامپیوتر(Super computer): کامپیوتر مرکزی که دارای توانایی زیاد محاسباتی است و محاسبه‌های علمی پیچیده را انجام می‌دهد. (Babylon> Babylon English)

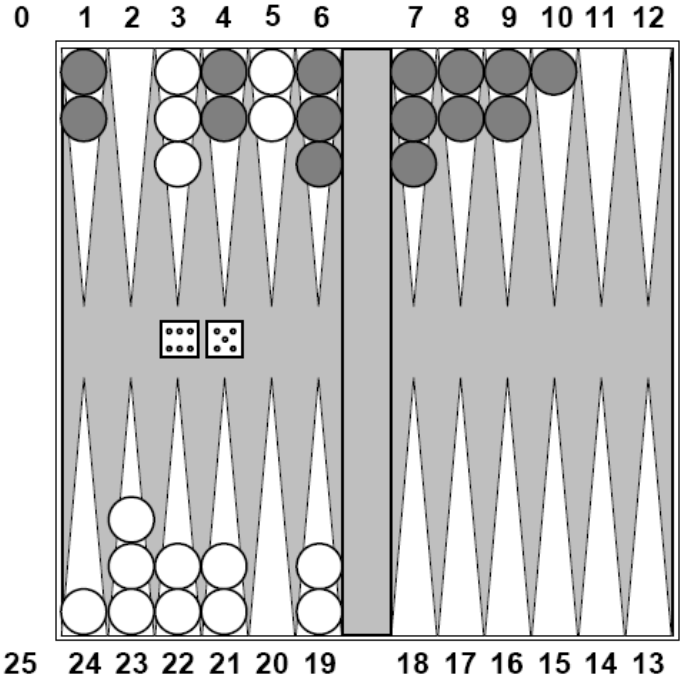
۲ – ۱ بیلیون = ۱ میلیارد = ۱۰۰۰۰۰۰۰۰۰(این عدد دارای نه صفر است.)

بازی Go

تا همین اواخر قهرمانان نمی‌پذیرفتند که با کامپیوتر رقابت کنند؛ زیرا کامپیوترها بازی‌کنندگان خیلی بدی برای بازی Go بودند. Go برای یک درخت جستجوی بازی معمولی، بیش از حد بزرگ است. البتّه در چند سال گذشته برنامه‌های Go به مراتب بهتر شده‌اند.

بازی‌های غیرقطعی در حالت کلّی

در بازی‌های غیر قطعی؛ مثل تخته نرد که تصویری از صفحه‌ی این بازی را در شکل زیر مشاهده می‌کنید؛ شانس شما بستگی به عددی که روی تاس[۱] ظاهر می‌شود و یا بُرْ زَدَن(مخلوط کردن) کارت[۲]های بازی(پاسور) دارد؛ بنابراین، در این دسته از بازی‌ها باید نقش شانس را هم در بازی به حساب آوریم.



۱ - dice

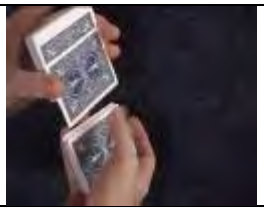
۲ - card-shuffling

پرتاب تاس

۱

۲

۳



۴

۵

۶

۷

۸

نمایش گام به گام یک روش بُر زدن کارت‌های بازی؛ در این روش قسمتی از کارت‌ها لابه‌لای قسمتی دیگر از کارت‌ها قرار می‌گیرند.

چکیده‌ی مطلب‌های فصل هشتم

در محیط‌های چندعاملی، عامل باید دیگر عامل‌ها را هم به حساب بیاورد.

تئوری بازی‌ها، به اثرات متقابل میان چند عامل می‌پردازد.

در بازی‌های با اطّلاعات کامل، عامل‌ها به همه‌ی دانش، در مورد محیط دسترسی دارند.

در بازی‌های با اطّلاعات ناکامل، محیط برای عامل به صورت جزئی قابل مشاهده است.

در درخت بازی، ریشه‌ی درخت، وضعیّت اوّلیّه است؛ سطح بعدی، تمام حرکت‌های MAX است؛ سطح بعدی، تمام حرکت‌های MIN است و ....
MIN و MAX دو بازیگر هستند که MAX می‌خواهد بازی را ببرد و MIN هم می‌خواهد بازی را ببرد و در واقع MAX و MIN رقیب هم هستند و هر دو بهترین بازی ممکن را انجام می‌دهند.

در روش مینیماکس از پردازه‌ی جستجو برای پیدا کردن مسیر راه حل استفاده نمی‌کنیم، بلکه برای به دست آوردن دقیق‌ترین ارزیابی حرکت‌های ممکن استفاده می‌کنیم.

در روش هرس(بازبینی)، شاخه‌ای که هیچوقت مورد استفاده قرار نمی‌گیرد را از درخت جستجو حذف می‌کنیم.

هرس(بازبینی) بر روی نتیجه‌ی نهائی اثری ندارد.

نسبت «بازی» به «هوش مصنوعی»؛ مثل نسبت «مسابقه‌ی بین‌المللی اتومبیل‌رانی فرمول ۱[۱]» به «طرّاحی اتومبیل» است.

---

۱ - Formula One یا Formula 1 یا F1؛ تصویری از یکی از این مسابقه‌ها:


HSBC
HSBC
HSBC
HSBC
HSBC
HSBC
HSBC

یادآوری یا تکمیل مطلب‌های فصل هشتم

تعریف– بازی‌های «جمع صفر»[۱] را تعریف کنید.

جواب– در این گونه، اگر یکی ببرد، دیگری می‌بازد؛ مثل بازی شطرنج.

تعریف– درخت بازی را تعریف کنید.

جواب– درختی که در آن، گره‌ها، وضعیّت‌های بازی هستند و یال‌ها(لبه‌ها)، حرکت‌های بازی هستند.[۲]

تعریف– «استراتژی بازی» را تعریف کنید.

جواب– تابعی که یک حرکت بازیکن را در هر وضعیّت ممکن بازی مشخّص می‌کند.[۳]

تعریف– الگوریتم مینیماکس را تعریف کنید.

جواب– استراتژی بهینه برای بازی‌های جمع صفر(zero-sum) دو نفره‌ی با اطّلاعات کامل است؛ امّا این روش اگر زمان محدود برای انجام هر حرکت داشته باشیم، غیر عملی است.[۴]

---

۱ – zero-sum

۲ – آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۳ – آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۴ – آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

تعریف- «تصمیم‌گیری مینیماکس» را تعریف کنید.

جواب- یک تصمیم‌گیری که بهترین نتیجه‌ی ممکن را در یک بازی ۲-نفره ضمانت می‌کند، با فرض اینکه حریف بهترین بازی را انجام دهد.[۱]

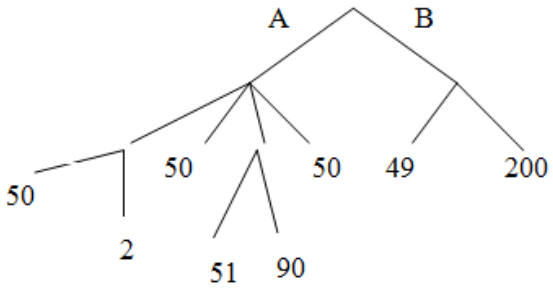
درست یا غلط- مینیماکس، الگوریتمی است که برای پیدا کردن راه حل‌های مسأله‌های برآورده‌سازی محدودیّت استفاده می‌شود.

جواب- «غلط» است؛ زیرا مینیماکس در بازی‌های دو نفره استفاده می‌شود.[۲]

تست- در درخت مینیماکس زیر با فرض آنکه در ریشه، MAX حرکت می‌کند، کدامیک از حرکت‌های A یا B انتخاب خواهند شد؟

۱) A

۲) B



جواب- گزینه‌ی «۱» درست است.[۳]

مطلب- بازی‌های غیر جمع صفر چند نفره[۴]- شبیه مینیماکس هستند: سودمندی‌ها حالا به صورت چندتایی هستند و هر بازیگر، سودمندی خود را در هر نوبت، بیشینه می‌کند.[۵]

مطلب- ☑مشابه کنکور سراسری فنّـآوری اطّلاعات سال ۸۵-

---

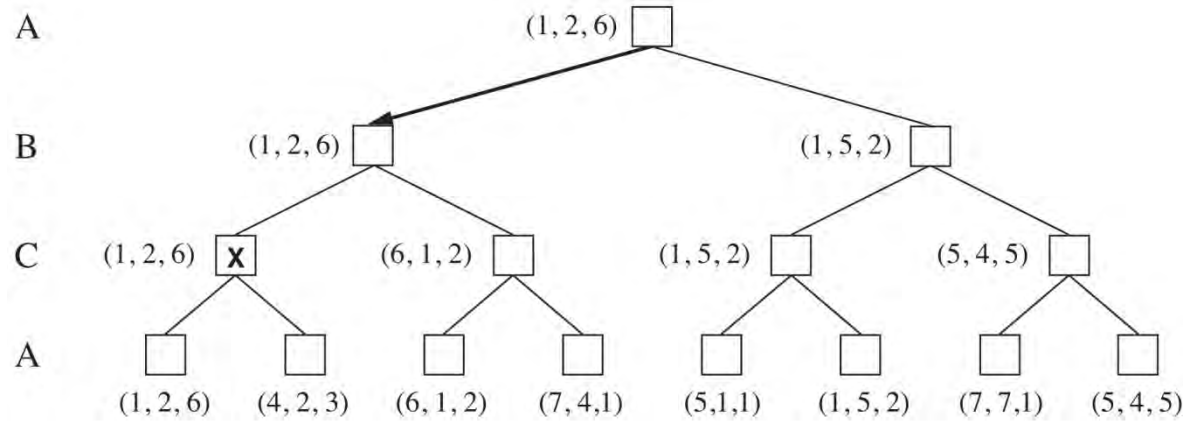
۱ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۱۹۹۳ میلادی

۲ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۱۱ ژانویه‌ی سال ۲۰۱۰ میلادی

۳ - آزمون میان‌ترم درس «هوش مصنوعی» استاد، «کتی مک‌کون»، دانشکده‌ی علوم کامپیوتر دانشگاه کلمبیای کشور آمریکا، ۲۹ اکتبر سال ۲۰۰۷ میلادی

۴ - Multi-player Non-Zero-Sum Games

۵ - مطلب‌های درس «آشنایی با هوش مصنوعی» استاد، «لوکْ زِتْلْمُیرْ(Luke Zettlemoyer)»، دانشکده‌ی علوم کامپیوتر و مهندسی دانشگاه واشینگتن کشور آمریکا، سال ۲۰۱۳ میلادی



شکل بالا- در سه‌تایی‌های بالا، اوّلین عضو، سودمندی بازیکن A، دوّمین عضو، سودمندی بازیکن B، و سوّمین عضو، سودمندی بازیکن C است(A, B, C). هر بازیگر، در هر حرکت خود، سودمندی مربوط به خودش را بیشینه می‌کند. بهترین حرکت، در ریشه‌ی درخت، نشان داده شده است.[۱]

تعریف- «ExpectiMiniMax» را تعریف کنید.

جواب- (☑مطلب کنکور سراسری مهندسی کامپیوتر سال ۸۴- مینیماکس را برای استفاده در مورد بازی‌های شانسی(تصادفی)، با اضافه کردن یک «لایه‌ی شانس»[۲]، میان هر حرکت بازیکن، در درخت جستجو تعمیم می‌دهد.)[۳] به شکل زیر[۴] با دقّت توجّه نمائید:

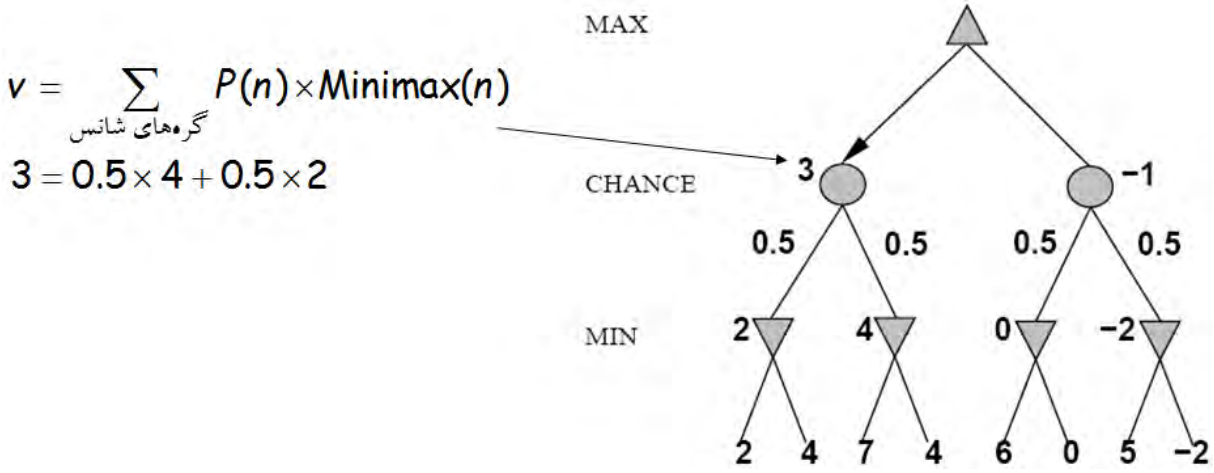
---

۱ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل پنجم، جستجوی خصمانه(Adversarial Search)، صفحه‌ی ۱۶۶

۲ - chance layer

۳ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۴ - مطلب‌های درس «آشنایی با هوش مصنوعی» استاد، «پادْرایْک سْمیتْ(Padhraic Smyth)»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور ایالات متّحده‌ی آمریکا، پاییز سال ۲۰۰۷ میلادی



شکل بالا– توجّه کنید که گره‌های شانس به شکل دایره هستند.

تعریف– هرس آلفا-بتا را تعریف کنید.

جواب– حرکتی مثل مینیماکس را برمی‌گرداند، امّا ممکن است شاخه‌های بیش‌تری را هرس نماید.[۱]

درست یا غلط– برش‌های α-β، در مینیماکس، کارآیی را بهبود می‌دهند، امّا می‌توانند باعث از دست دادن برخی از راه حل‌ها شوند.

جواب– **«غلط» است؛** از آنجایی که برش فقط در زمانی انجام می‌شود که زیردرختی که حذف می‌شود نامربوط به راه حل است، هیچ راه حلّی از دست نمی‌رود.[۲]

درست یا غلط– جستجوی α-β، تمام گره‌های به وجود آمده را در حافظه نگهداری می‌کند.

جواب– **«غلط» است.**[۳]

درست یا غلط– مینیماکس و آلفا-بتا می‌توانند گاهی اوقات نتیجه‌های متفاوتی را برگردانند.

جواب– **«غلط» است؛** آلفا-بتا نسخه‌ی کاراتری از مینیماکس است.[۴]

---

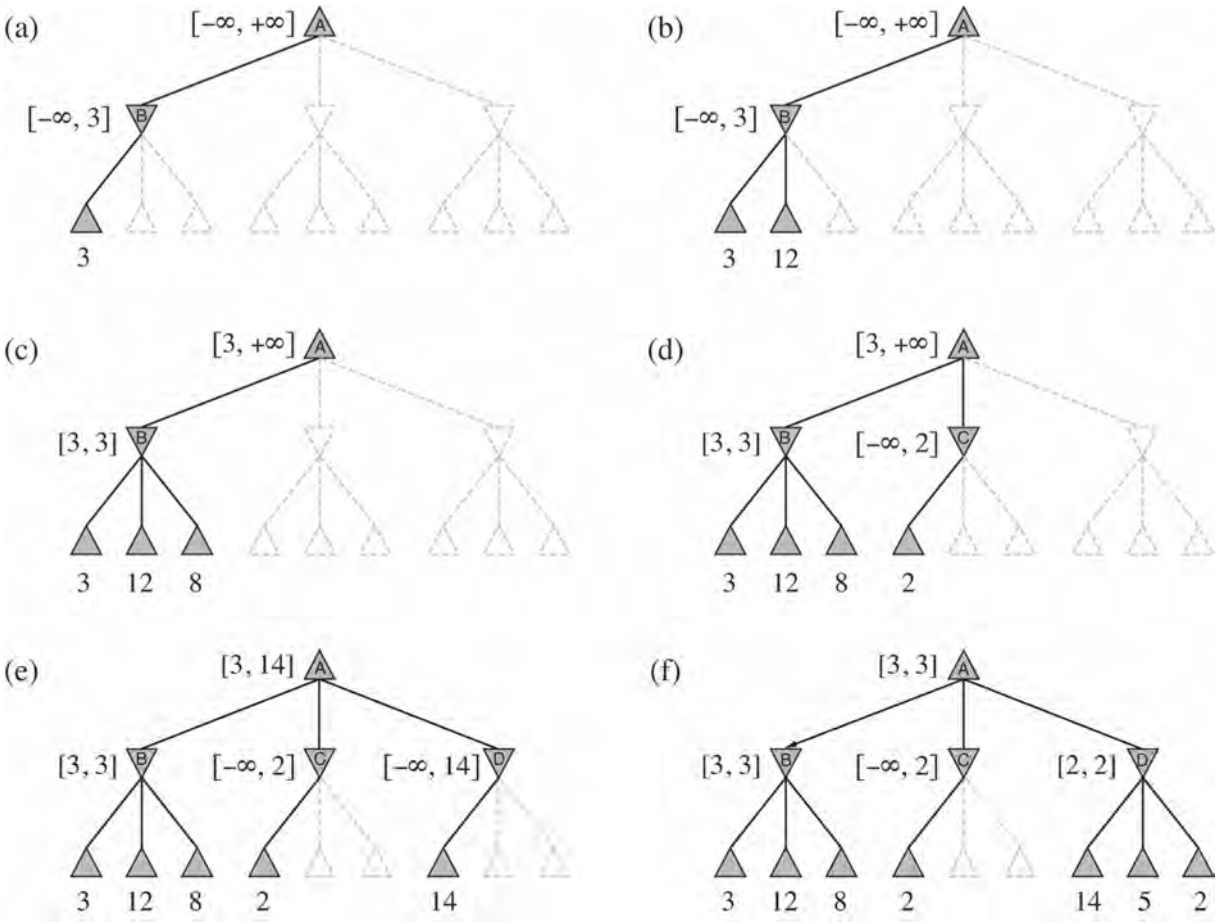
۱ – آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۲ – آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۷ ژوئن سال ۲۰۱۲ میلادی

۳ – آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، پاییز سال ۱۹۹۷ میلادی

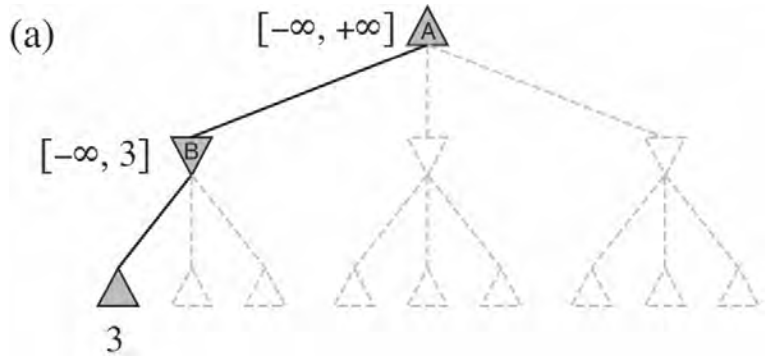
۴ – آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۱۹۹۶ میلادی

مثال هرس آلفا-بتا (مهم)- ☑مشابه سؤالات مطرح شده در کنکورهای مختلف سراسری مربوط به رشته‌ی کامپیوتر سال‌های مختلف و مشابه کنکور آزاد مهندسی کامپیوتر سال ۹۲-

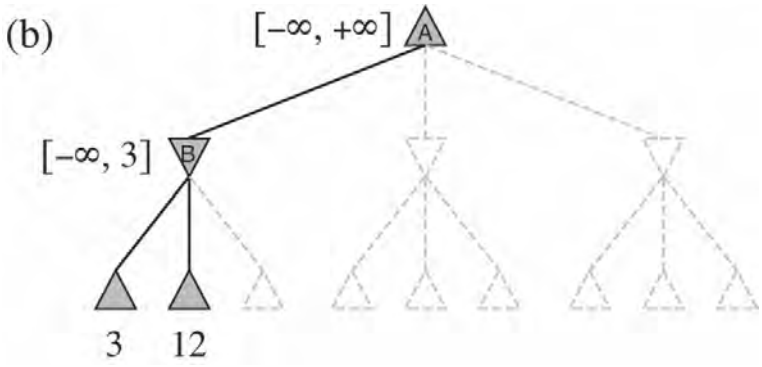


توضیح شکل بالا-

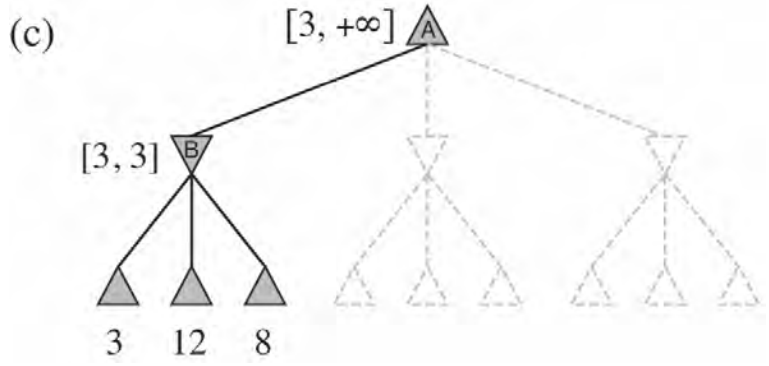
در قسمت (a)، اوّلین برگ دارای مقدار ۳ است. بنابراین، B، که یک گره‌ی MIN است، دارای مقدار حدّاکثر ۳ خواهد بود:



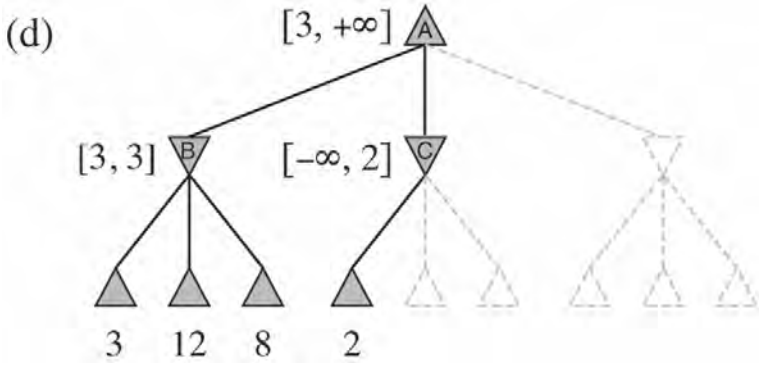
در قسمت (b)، دوّمین برگ، که در زیر B قرار دارد، دارای مقدار ۱۲ است؛ بنابراین، مقدار B هنوز حدّاکثر برابر با ۳ خواهد بود:



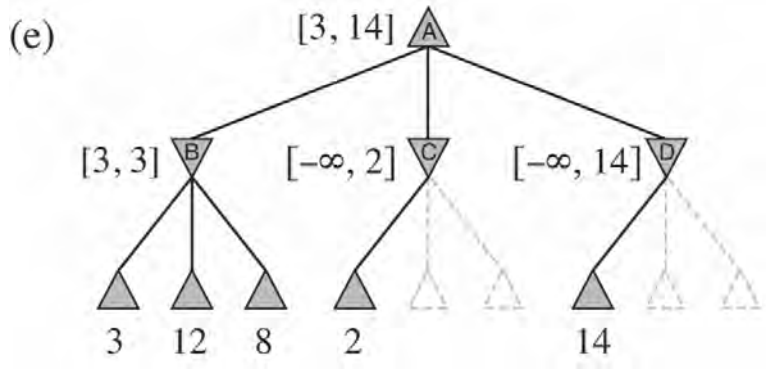
در قسمت (c)، سوّمین برگ زیر B، دارای مقدار ۸ است؛ ما تمام وضعیّت‌های جانشین B را برّرسی کردیم، بنابراین، مقدار B درست برابر با ۳ است. حال ما می‌توانیم نتیجه بگیریم که مقدار ریشه، لااقل برابر با ۳ است، زیرا MAX دارای یک انتخاب با مقدار ۳ در ریشه است:



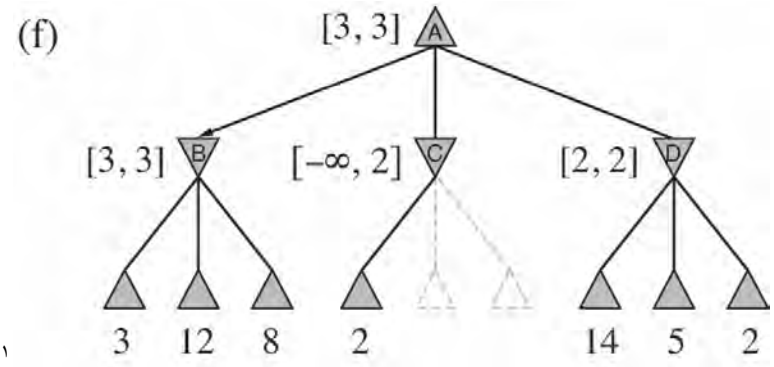
در قسمت (d)، اوّلین برگ زیر C، دارای مقدار ۲ است. بنابراین، C، که یک گره‌ی MIN است، دارای مقدار حدّاکثر ۲ است. *امّا ما می‌دانیم که B دارای مقدار ۳ است، بنابراین، MAX، هرگز C را انتخاب نخواهد کرد. بنابراین، نیازی به برّرسی دیگر وضعیّت‌های جانشین C نمی‌باشد. این مثالی از هرس آلفا-بتا است:*



در قسمت (e)، اوّلین برگ زیر D، دارای مقدار ۱۴ است، بنابراین، D دارای مقدار حدّاکثر ۱۴ است. ۱۴، بیش‌تر از بهترین انتخاب MAX (؛ یعنی، ۳) است، بنابراین، ما نیاز داریم برّرسی وضعیّت‌های جانشین D را ادامه دهیم. همچنین توجّه نمائید که ما حالا دارای حدّ و مرز، روی تمام جانشینان ریشه هستیم، بنابراین، مقدار ریشه همچنان برابر با مقدار حدّاکثر ۱۴ است:

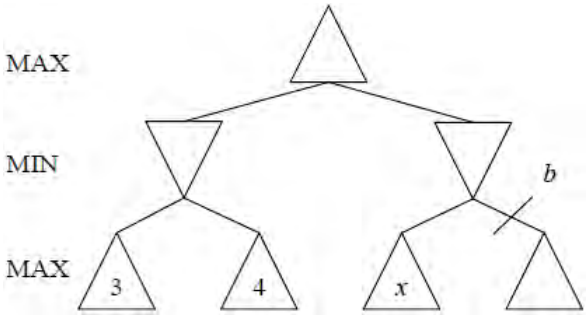


در قسمت (f)، دوّمین جانشین D، دارای مقدار ۵ است، بنابراین، دوباره ما نیاز به ادامه دادن برّرسی داریم. سوّمین جانشین، دارای مقدار ۲ است، بنابراین، حالا D دارای مقدار دقیقاً برابر با ۲ است. تصمیم MAX، در ریشه، حرکت به B، با داشتن مقدار ۳ است:



[1]

تست– در درخت جستجوی بازی دو نفره‌ی زیر به ازای کدام مقادیر x، درخت، در b، در موقع هرس آلفا–بتا، هرس خواهد شد؟



۱) $x \geq 3$

۲) $x \leq 3$

۳) $x \geq 4$

۴) $x \leq 4$

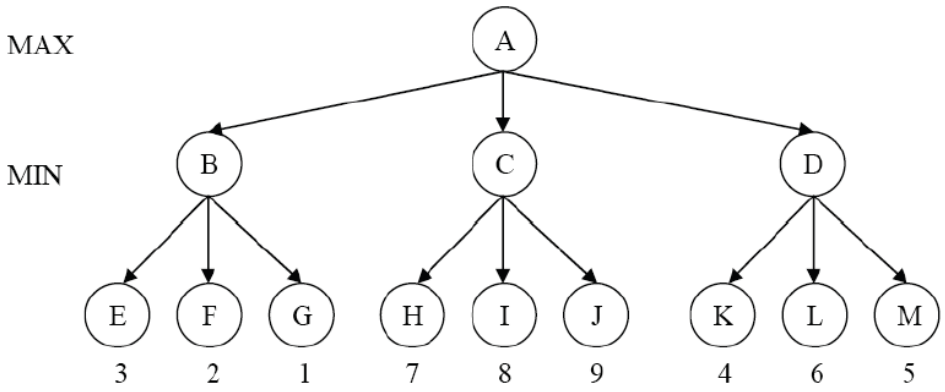
۵) $x < 3$

---

۱ – کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل پنجم، جستجوی خصمانه(Adversarial Search)، صفحه‌ی ۱۶۸

جواب- گزینه‌ی «۲» درست است.[۱]

سؤال- به درخت بازی زیر که دارای گره‌ی ریشه‌ی بیشینه کننده است و فرزندان آن از چپ به راست ملاقات می‌شوند، توجّه کنید:



الف) مقادیر گره‌های A، B، C و D را با استفاده از الگوریتم مینیماکس محاسبه کنید.

جواب- A=7 و D=4، C=7، B=1

ب) چه حرکتی به وسیله‌ی بازیکنی که از روش مینیماکس استفاده می‌کند، انتخاب خواهد شد؟

جواب- C

ج) گره‌های برگی که به وسیله‌ی هرس آلفا-بتا هرس خواهند شد، چیستند؟

جواب- L و M[۲]

سؤال- منظور از حرکت‌های دارای «سکون»[۳] چیست؟

جواب- حرکت‌هایی که به نظر نمی‌رسد تغییر زیادی را در آینده به وجود آورند.[۴]

سؤال- «اَثَر اُفُق»[۵] را توضیح دهید.

---

۱ - آزمون میان‌ترم درس «هوش مصنوعی» استاد، «روت شولتز»، دانشکده‌ی مهندسی برق و فنّآوری اطلاعات دانشگاه کوئینزلند کشور استرالیا، سال ۲۰۱۲ میلادی

۲ - آزمون میان‌ترم درس «مبانی هوش مصنوعی» استاد، «تیم فینین»، دانشکده‌ی مهندسی برق و علوم کامپیوتر دانشگاه شهر بالتیمور ایالت مریلند کشور آمریکا، ۱۹ اکتبر سال ۲۰۰۹ میلادی

۳ - quiescent

۴ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل پنجم، جستجوی خصمانه(Adversarial Search)، صفحه‌ی ۱۷۴

۵ - Horizon Effect

جواب- در حالتی که یک رویداد، بازی را زیاد تغییر می‌دهد، می‌توان برای مدّت طولانی، امّا نه همیشه، جلوی این رویداد را گرفت[۱]؛ مثلاً در شکل زیر *که نوبت حرکت مهره‌ی سیاه است*، اگر سرباز سفید نشان داده شده در شکل، یک خانه به بالا حرکت کند، طبق قوانین بازی شطرنج، به وزیر تبدیل خواهد شد و سیاه را شکست خواهد داد. ولی سیاه، با مهره‌ی رُخ(قلعه‌ی) خود می‌تواند چهارده بار شاه سفید را کیش دهد و برد سفید را به تأخیر بیندازد:

شکل بالا- اثر افق. یک سری ممانعت‌ها به وسیله‌ی مهره‌ی رخ سیاه، باعث تأخیر در به دست آوردن وزیر سفید می‌شود و سبب می‌شود که وضعیّت، این گونه به نظر برسد که سیاه در حال بردن است، در حالی که اینگونه نیست و در واقع این یک برد برای سفید است.[۲]

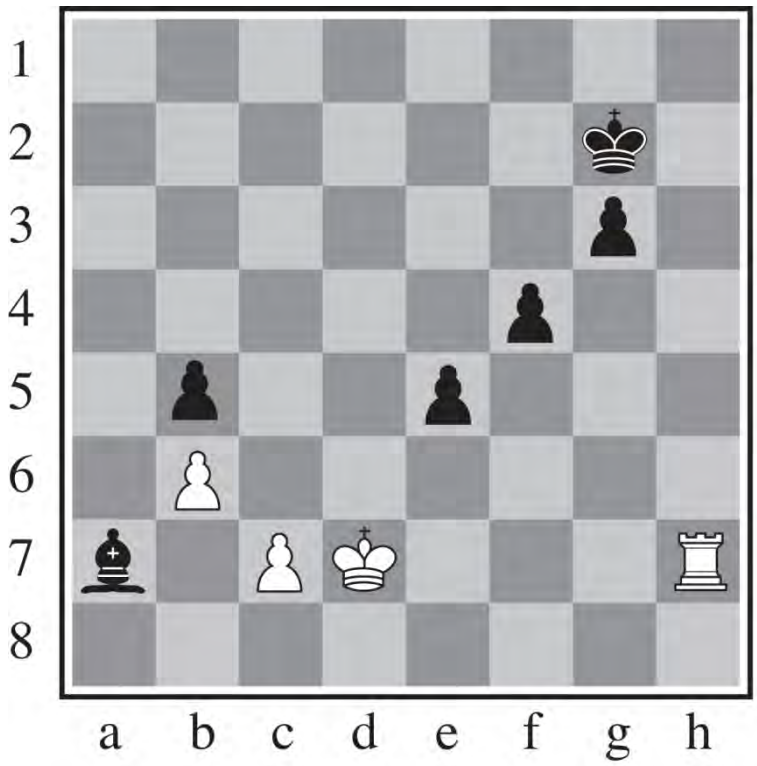
یا به عنوان مثالی دیگر، برای اثر افق، چنانچه در شکل زیر نشان داده شده است، با حرکت سیاه، فیل سیاه حتماً زده خواهد شد، امّا سیاه می‌تواند با استفاده از پیاده‌هایش، در برابر شاه سفید، از این رویداد ممانعت کند و شاه سفید را مجبور به زدن سربازهای سیاه کند. بنابراین، فداکاری مهره‌های پیاده، توسّط الگوریتم جستجو، به عنوان حرکت‌های خوب دیده می‌شوند، نه بد:[۳]

---

۱ - مطلب‌های درس «هوش مصنوعی» استاد، «دِیْوْ میوزیکانْتْ(Dave Musicant)»، دانشگاه کارلْتون(Carleton) کشور کانادا، پاییز سال ۲۰۰۴ میلادی

۲ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش دوّم، فصل ششم، جستجوی خصمانه(Adversarial Search)، صفحه‌ی ۱۷۴

۳ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل پنجم، جستجوی خصمانه(Adversarial Search)، صفحه‌ی ۱۷۵



سؤال– «هرس مستقیم»[1] را توضیح دهید.

جواب– ممکن است که ما از هرس مستقیم استفاده کنیم، به این صورت که برخی حرکت‌هایی که برای یک گره وجود دارند را بلافاصله بدون برّرسی بیش‌تر هرس کنیم؛ همان طور که بیش‌تر انسان‌ها در موقع بازی شطرنج فقط به تعداد اندکی از حرکت‌ها از هر وضعیّتی توجّه می‌کنند. یک روش برای هرس مستقیم، این است که در هر حرکت، فقط به یک پرتو(تعداد) از بهترین حرکت‌ها، با توجّه به تابع ارزیابی توجّه کنیم، به جای اینکه به تمام حرکت‌های ممکن توجّه کنیم. متأسّفانه این روش نسبتاً خطرناک است، چون که هیچ ضمانتی برای اینکه بهترین حرکت حذف نخواهد شد، وجود ندارد.[2]

---

۱ - forward pruning

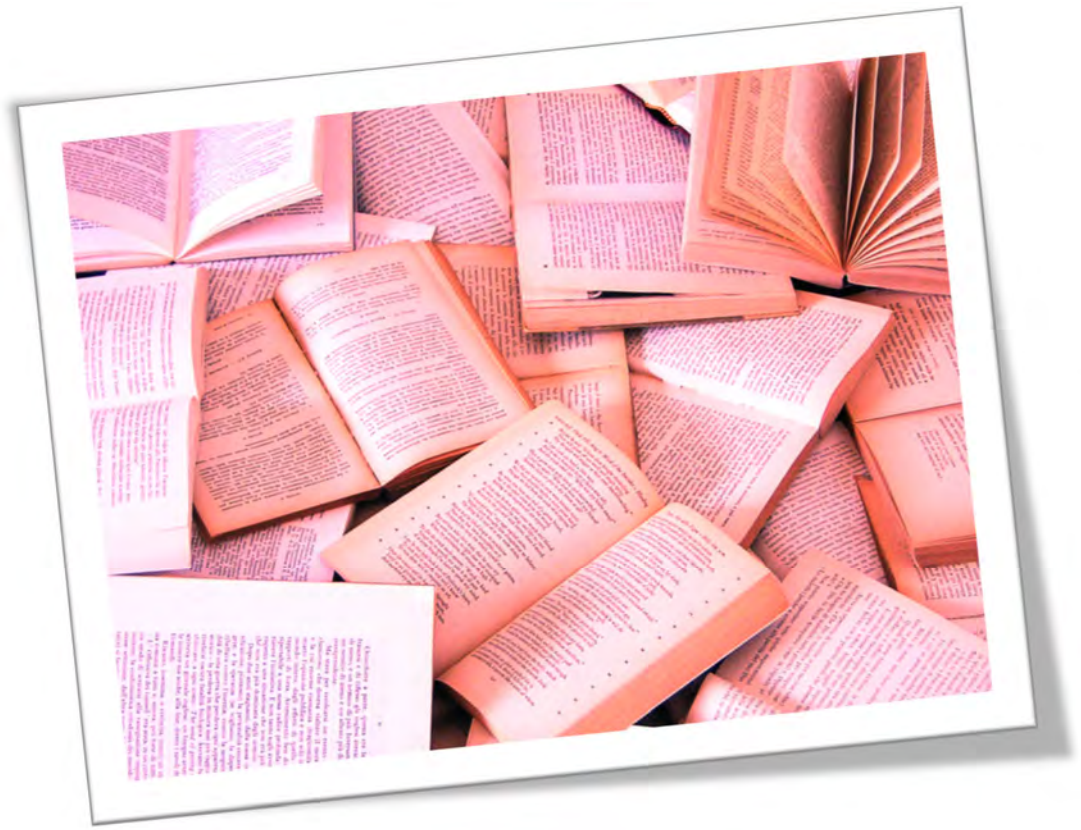
۲ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل پنجم، جستجوی خصمانه(Adversarial Search)، صفحه‌های ۱۷۴ و ۱۷۵

فصل نهم

عامل‌های منطقی[۲]

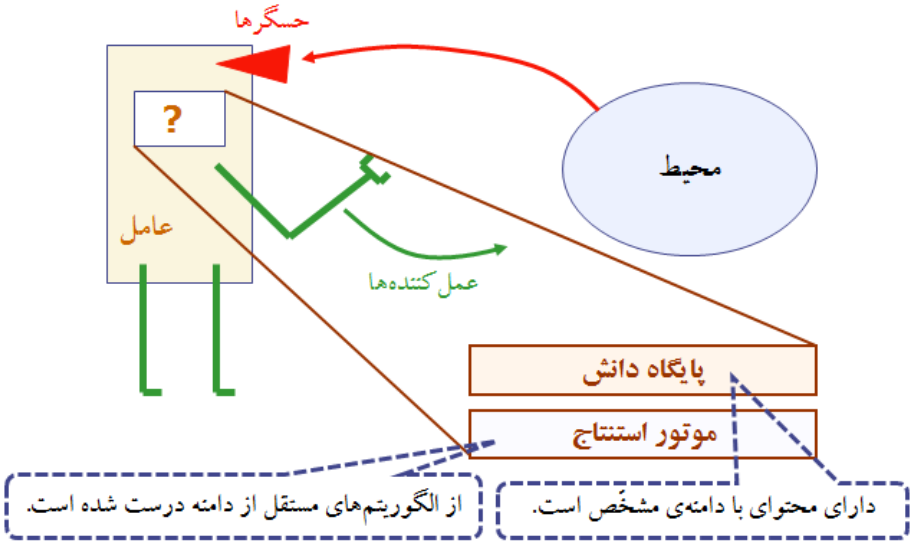
۱ – تصویر، مربوط به بازی‌ای به نام هَیولا(wumpus) است که در ادامه‌ی همین فصل با آن آشنا می‌شویم.

۲ – Logical Agents



فهرست برخی از عنوان‌های نوشته‌ها

عامل براساس دانش[1](عامل منطقی)



عامل‌های بازتابی، راه خود را شانسی و با کندذهنی پیدا می‌کنند.

مطلب مهم:

عامل‌های منطقی(برمبنای دانش، مبتنی بر دانش، براساس دانش)، پیش از انتخاب عملکردها، دانش را با ادراک‌های جاری، برای پی‌بردن به جنبه‌های پنهان وضعیّت ترکیب می‌کنند؛ یک عامل مبتنی بر دانش، از یک پایگاه دانش و یک مکانیزم استنتاج تشکیل شده است. با نگهداری جملات درباره‌ی جهان در پایگاه دانشش و استفاده از مکانیزم استنتاج، برای پی بردن به(استنتاج کردن) جملات جدید و استفاده از این جملات برای تصمیم‌گیری در مورد این که چه عملکردی باید اجرا شود، کار می‌کند.[2]

---

۱ - Knowledge-based agent

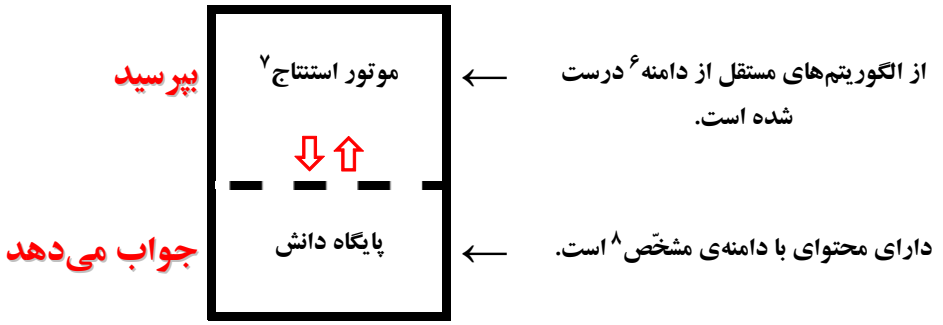
۲ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل هفتم، عامل‌های منطقی(Logical Agents)، صفحه‌ی ۲۷۴

پایگاه‌های دانش

یادآوری- یک پایگاه دانش[۱]، تشکیل شده از جملات یا عبارت‌هایی[۲] که واقعیّت‌هایی را در مورد جهان خود بیان می‌کنند. به بیان دیگر، پایگاه دانش، مجموعه‌ای از عبارت‌ها است که به وسیله‌ی یک زبان قراردادی[۳] بیان شده‌اند.

حال این سؤال مطرح می‌شود که تفاوت میان پایگاه دانش و پایگاه داده[۴] چیست؟: به‌طورکلّی، در طرز بیان و کاربرد آنهاست و در عمل، یک پایگاه دانش ممکن است از یک پایگاه داده استفاده کند.

جملات، در مورد چیزهایی که در جهانشان وجود دارد، مثل هَیولا[۵]ها، طلا، چاله‌ها، فضاها و ارتباط میان عامل‌ها توضیح می‌دهند.



قانون استنتاج: وقتی فردی از پایگاه دانش [چیزی را] می‌پرسد، جواب باید از آنچه که در گذشته به پایگاه دانش گفته‌ایم، بیاید.

به پایگاه دانش **بگویید**( )[۱] که می‌خواهید چه چیزی را بدانید. سپس از خودش **سؤال می‌کند** ( )[۲] که چه کاری باید انجام دهد؛ جواب‌هایی که می‌دهد باید از پایگاه دانش(KB) گرفته شوند.

۱- *K*nowledge *B*ase(*KB*)

۲- sentences

۳- formal

۴- Database

۵- Wumpus

۶- domain-independent algorithms

۷- inference engine

۸- domain-specific content

یک عامل ساده‌ی براساس دانش

الگوریتم:

تابع KB-Agent(percept)، یک action(عمل) را برمی‌گرداند

متغیّرهای استاتیک:

متغیّر KB، که یک پایگاه دانش می‌باشد

t، یک شمارنده با مقدار اوّلیّه‌ی صفر و نشان دهنده‌ی زمان می‌باشد

Tell(KB,Make-Percept-Sentence(percept,t))

action ← Ask(KB,Make-Action-Query[۳](t))

Tell(KB,Make-Action-Sentence(action,t))

t ← t+1

action را برگردان

پایان الگوریتم

عامل پایگاه دانش باید قادر باشد:

حالت‌ها، عملکردها و غیره را بیان نماید؛

دریافت‌های ادراکی[۴] جدید را یکی(ترکیب) کند[۵]؛

تغییرهای جهان را در پایگاه دانش خود ثبت نماید؛

ویژگی‌های پنهان جهان خود را استنباط[۶] نماید؛

---

۱ - Tell ( )

۲ - Ask ( )

۳ - در لغت به معنی «پرس و جو» یا «جستجو» می‌باشد.

۴ - percepts

۵ - Incorporate

۶ - deduce

عملکردهای مناسب را استنباط نماید

توضیح:

پایگاه دانش عامل، شبیه عامل‌های با وضعیّت درونی می‌باشد.

عامل‌ها می‌توانند در سطح‌های مختلف بیان شوند:

۱- **سطح دانش:** به عنوان مثال، چه می‌دانند، بدون توجّه به چگونگی پیاده‌سازی.
۲- **سطح پیاده‌سازی:** به عنوان مثال، ساختمان‌های داده‌ای که در پایگاه دانش به کار می‌روند و الگوریتم‌هایی که آنها را به کار می‌گیرند؛ مثل منطق گزاره‌ای[۱] و تحلیل[۲].

دنیای هَیولا(Wumpus)

۱ - این مطلب در فصل «منطق گزاره‌ای» همین کتاب الکترونیکی یادآوری شده است.
۲ - resolution؛ این مطلب در فصل «منطق گزاره‌ای» همین کتاب الکترونیکی توضیح داده شده است.

هیولا، نام یک بازی کامپیوتری است؛ در این بازی، عامل یک غار را که از اتاق‌های متّصل شده به وسیله‌ی راهرو تشکیل شده است، برّرسی(کاوش) می‌کند. در این کتاب، عامل دنیای هیولا با شکل زیر نشان داده شده است:

کمین گاه، جایی است که در آن هیولا قرار دارد و هیولا، جانوری است که هر عاملی را که به اتاقش وارد شود، می‌خورد. در این کتاب، هیولا با شکل زیر نشان داده شده است:

همچنین برخی از اتاق‌ها دارای چاله[1]های بدون کَفْ هستند و هر عاملی را که در اتاق سرگردان است، به تله می‌اندازند و به صورت مربّع‌هایی با گوشه‌های گِرد شده ودارای رنگ سیاه هستند و روی آنها کلمه‌ی PIT (به معنای چاله) نوشته شده است.

گاهی از وقت‌ها گُپّه‌ای از طلا هم در اتاق هست که برروی آن کلمه‌ی Gold، به معنی طلا نوشته شده است. *هدف ما در این بازی این است که طلا(ها) را جمع کنیم و بدون اینکه [به وسیله‌ی هیولا] خورده شویم، خارج شویم.*

توضیح PEAS دنیای هیولا

اندازه‌گیری عملکرد یا معیارکارآیی: طلا (۱۰۰۰+)، مرگ (۱۰۰۰-)، (منفی یک) در هر گام، (۱۰-) برای استفاده از پیکان(تیر)[2].

محیط: مربّع‌های کنار هیولا، بدبو[3] هستند؛ مربّع‌های کنار چاله[4]، خوش هوا[5] هستند و اگر طلا در همان خانه‌ی خوش هوا باشد، درخشان[6] است؛ اگر به دیوار بخورید، ضربه[7] خواهید خورد؛ اگر هیولا کشته شد، صدای جیغ اندوهناک[8] هیولا را

۱- pit
۲- arrow
۳- smelly
۴- pit
۵- breezy
۶- glitter
۷- bump
۸- woeful scream

خواهید شنید؛ تیراندازی[۱]، هیولا را در صورتی که شما روبه‌روی آن باشید، می‌کشد؛ تیراندازی، فقط پیکان(تیر) را مصرف[۲] می‌نماید؛ رُبایش[۳]، به وسیله‌ی قلّاب[۴]، فقط طلا را در صورتی که در همان مربّع باشد، انتخاب می‌نماید. آزاد کردن[۵]، طلا را در همان خانه باقی می‌گذارد.

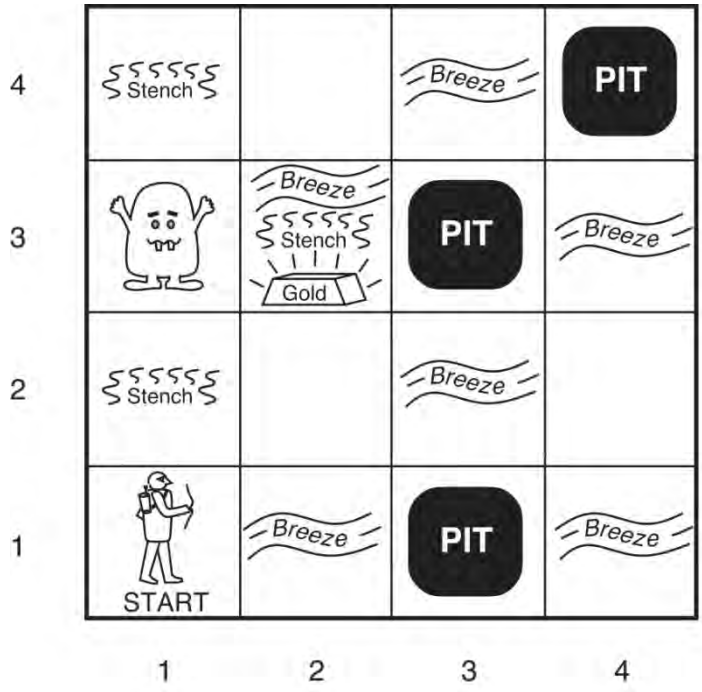
عمل‌کننده‌ها: حرکت به چپ(چپ گرد)[۶]، حرکت به راست(راست گرد)[۷]، حرکت مستقیم[۸]، قلّاب، آزاد کردن، تیراندازی

حسگرها: بوی بد[۹]، بوی خوب[۱۰]، درخشش، ضربه و جیغ زدن[۱۱]

دنیای معمولی هیولا

عامل معمولاً از خانه‌ی [۱و۱] شروع می‌کند. کار عامل این است که طلا را پیدا کند، به خانه‌ی [۱و۱] برگردد و از غار خارج شود.

۱- Shoot
۲- use up
۳- grabbing
۴- Grab
۵- Release
۶- Left turn
۷- Right turn
۸- Forward
۹- stench
۱۰- breeze
۱۱- scream



شکل بالا- یک دنیای معمولی هیولا، در حالی که عامل به طرف راست چرخیده است.

ویژگی‌های دنیای هیولا

قابل مشاهده بودن؟؟ نه- فقط دارای ادراک محلّی است.

قطعیّت؟؟ بله- نتایج دقیقاً مشخّص است.

دوره‌ای؟؟ نه- دارای ترتیب در سطح عملکردها است.

ایستایی؟؟ بله- هیولا و چاله‌ها نمی‌توانند حرکت نمایند.

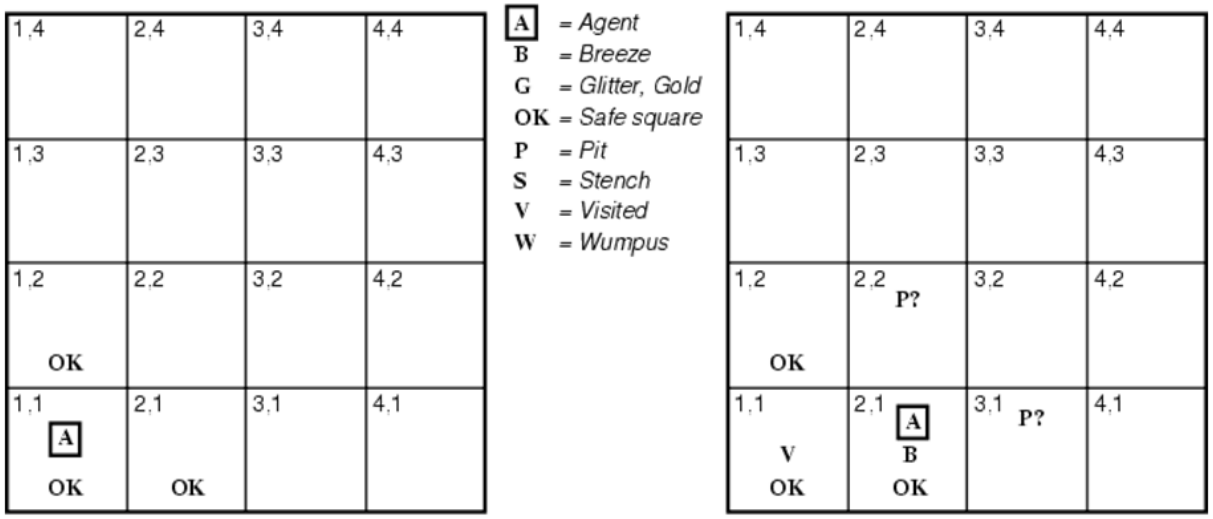
گسسته؟؟ بله

تک عاملی؟؟ بله

حرکت عامل در دنیای هیولا

توجّه:

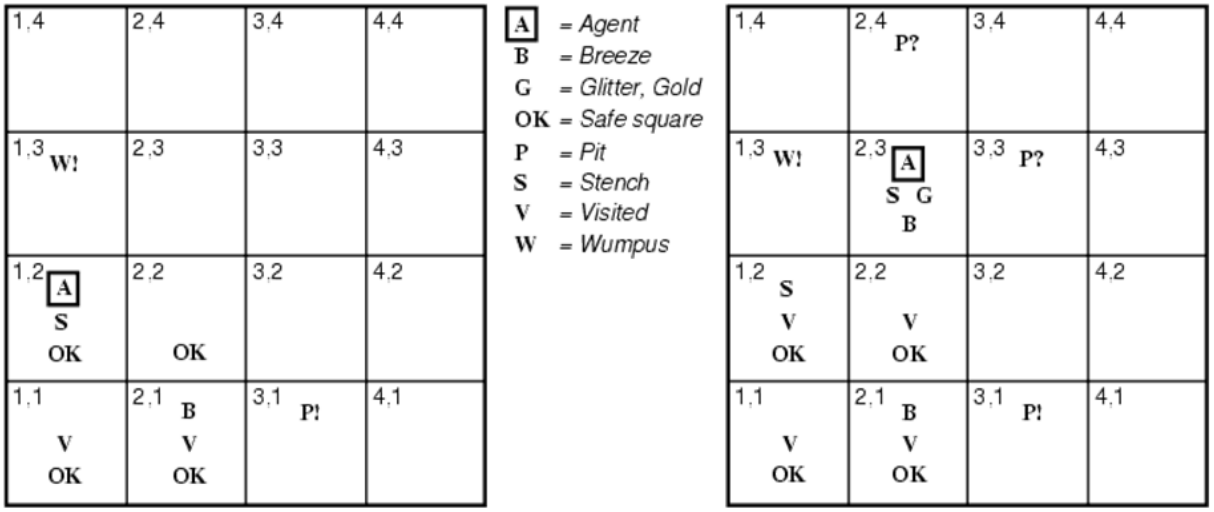
در شکل‌هایی که در ادامه می‌آیند، حرف A، نشان دهنده‌ی عامل؛ حرف B، نشان دهنده‌ی خانه‌ی با بوی خوش؛ حرف G، نشان دهنده‌ی وجود طلا؛ کلمه‌ی OK، نشان دهنده‌ی خانه‌ی اَمن[1]؛ حرف P، برای چاله؛ حرف S، برای بوی بد؛ حرف V، به معنای خانه‌ی رفته شده در گذشته(ملاقات شده)؛ و حرف W، به معنای وجود هیولا در خانه می‌باشد.



توضیح:

همان گونه که در شکل بالا می‌بینیم، در خانه‌ی [۱و۱] پایگاه دانش دارای قانون‌های محیط است. اوّلین دریافت یا ادراک، [هیچ و هیچ و هیچ و هیچ] است. به یک خانه‌ی امن، مثل [۱و۲] می‌رویم؛ خانه‌ی [۱و۲] چون خوش بو است، نتیجه می‌گیریم که چاله در [۲و۲] یا [۱و۳] است؛ در نتیجه به خانه‌ی [۱و۱] برمی‌گردیم و خانه‌ی امن بعدی را امتحان می‌نماییم:

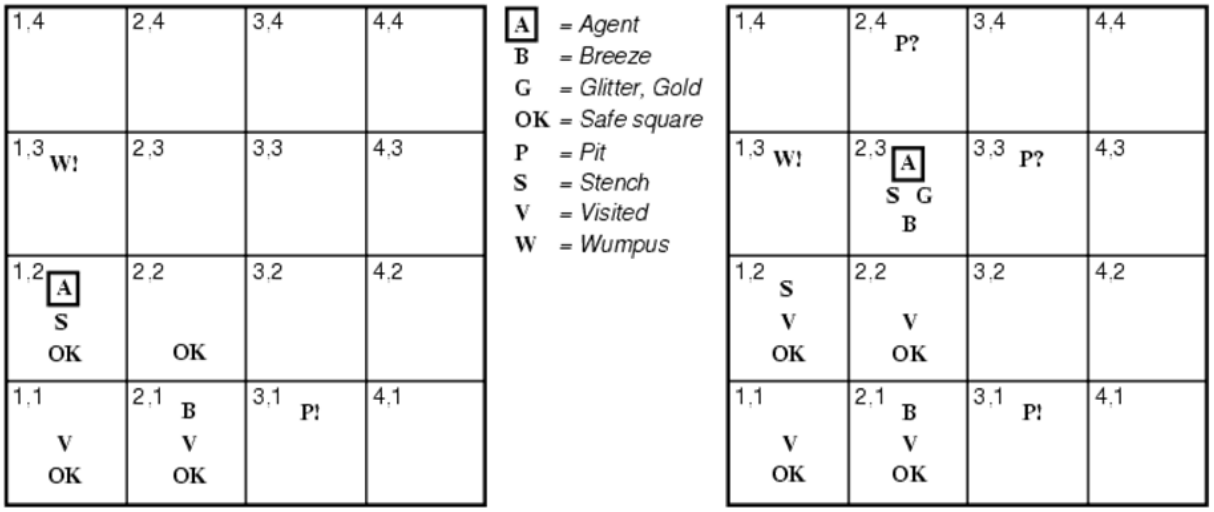
۱- safe square



(a) (b)

توضیح:

در خانه‌ی [۲و۱] از بوی بد(تعفّن) خانه می‌فهمیم که هیولا در خانه‌ی [۳و۱] یا [۲و۲] است. هیولا در خانه‌ی [۱و۱] یا [۲و۲](به دلیل اینکه وقتی در خانه‌ی [۱و۲] بودیم، بوی بد نبود) نمی‌باشد، یا در خانه‌ی [۱و۲](به دلیل اینکه در گذشته به این خانه رفتیم) هم نیست؛ در نتیجه هیولا در خانه‌ی [۳و۱] است؛ بنابراین، خانه‌ی [۲و۲] امن است؛ چون بوی خوش در خانه‌ی [۲و۱] نیست؛ بنابراین، چاله در خانه‌ی [۱و۳] می‌باشد. بنابراین، به خانه‌ی [۲و۲] می‌رویم. از خانه‌ی [۲و۲] به خانه‌ی [۳و۲] می‌رویم:



(a) (b)

توضیح:

در خانه‌ی [۳و۲]، درخشش طلا، بوی بد و بوی خوش را داریم، بنابراین، طلا را برمی‌داریم و از بوی خوش موجود در این خانه نتیجه می‌گیریم که چاله در خانه‌ی [۳و۳] یا [۴و۲] می‌باشد. سپس از مسیر امنی که در گذشته آمده‌ایم، برمی‌گردیم و در نهایت بازی خاتمه می‌یابد.

منبع‌های اینترنتی بیش‌تر برای هیولا

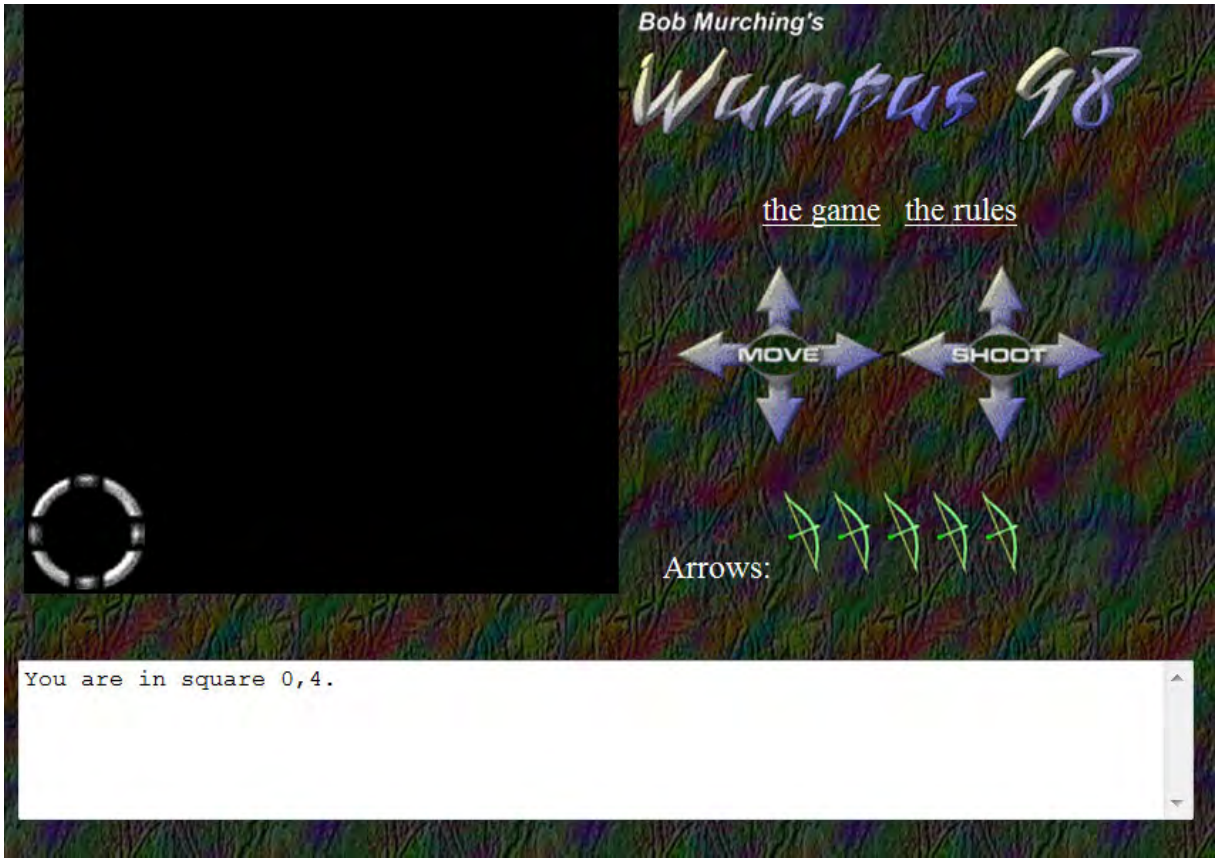
برای به دست آوردن اطّلاعات بیش‌تر در مورد این بازی می‌توانید از آدرس اینترنتی زیر استفاده نمایید:

http://en.wikipedia.org/wiki/Hunt_the_Wumpus

برای اجرای نسخه‌ای برخط از این بازی می‌توانید از آدرس اینترنتی زیر استفاده نمایید:

http://www.inthe70s.com/games/wumpus/index.shtml#

در شکل زیر نمونه‌ای از صفحه‌ی این بازی را بعد از بارگذاری آدرس اینترنتی بالا می‌بینید:



چکیده‌ی مطلب‌های فصل نهم

عامل‌های منطقی(برمبنای دانش، مبتنی بر دانش، براساس دانش) پیش از انتخاب عملکردها، دانش را با ادراک‌های جاری، برای پی‌بردن به جنبه‌های پنهان وضعیّت ترکیب می‌کنند؛ یک عامل مبتنی بر دانش، از یک پایگاه دانش و یک مکانیزم استنتاج تشکیل شده است. با نگهداری جملات درباره‌ی جهان در پایگاه دانشش و استفاده از مکانیزم استنتاج، برای پی بردن به(استنتاج کردن) جملات جدید و استفاده از این جملات برای تصمیم‌گیری در مورد این که چه عملکردی باید اجرا شود، کار می‌کند.

پایگاه دانش عامل، شبیه عامل‌های با وضعیّت درونی می‌باشد.

یادآوری یا تکمیل مطلب‌های فصل نهم

درست یا غلط– می‌توان با استفاده از یک عامل برمبنای دانش، یک عامل کاملاً بازتابی ساخت.

جواب– «**درست» است؛** گرچه پایگاه دانش، یک حافظه است، عامل می‌تواند از آن فقط برای نگهداری قانون‌های بازتابی استفاده کند.[۱]

تعریف– «سازگاری (یک پایگاه دانش)» را تعریف کنید.

جواب– تناقض(مغایرت)ی وجود نداشته باشد؛ بنابراین، در برخی از جهان‌ها، تحت برخی تفسیرها می‌تواند درست باشد.[۲]

---

۱ – آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۸ فوریّه‌ی سال ۲۰۱۰ میلادی و آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، پاییز سال ۱۹۹۷ میلادی

۲ – آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، پاییز سال ۱۹۹۱ میلادی

فصل دهم

۱

منطق گزاره‌ای[۲]

---

۱ - تصویر، جُرج بول(George Boole)، ۱۸۶۴ – ۱۸۱۵ میلادی، ریّاضیدان و فیلسوف انگلیسی و پدیدآورنده‌ی منطق بولی(Boolean Logic) است. منطق بولی، مبنای کار کامپیوترهای دیجیتالی مُدرن می‌باشد. یک عبارت بولی، دارای ارزش درست(True) یا غلط(False) است و عملیّات AND، OR، NOT (در ادامه‌ی همین فصل این عملگرها یادآوری خواهند شد)، XOR(به عنوان یادآوری، حاصل این عملگر بر روی دو عَمَلوَند(operand) A و B در صورتی درست است که یکی از عملوندها درست و دیگری نادرست باشد)، NOR(به عنوان یادآوری، حاصل این عملگر، برعکس عملگر OR است) و NAND(به عنوان یادآوری، حاصل این عملگر، برعکس عملگر AND است) بر روی عبارت‌های بولی تعریف می‌شوند. (http://en.wikipedia.org/wiki/George_Boole و Babylon> Persian Computer Encyclopedia)

۲ - Propositional Logic

فهرست برخی از عنوان‌های نوشته‌ها

عبارت(کلز) صریح(قطعی، مسلّم)

صورت نرمال ربطْ دهنده(CNF)

روش(شیوه یا متد)های اثبات

زنجیره‌ی پیشرو(مستقیم)

زنجیره‌ی پسرو(معکوس)

تحلیل در منطق گزاره‌ای

منطق چیست؟

تعریف‌ها

تعریف نخست: مطالعه‌ی گزاره(عبارت)ها و کاربرد آنها در استدلال(بحث)ها.[۱]

تعریف دوّم: علم استدلال و استنتاج است.[۲]

تعریف سوّم (تعریف بهتر): منطق، روشی کلاسیک برای بیان دانش عامل می‌باشد و به ما اجازه می‌دهد که عامل‌ها را به صورت اعلانی(اظهاری)، برنامه‌ریزی و توصیف کنیم؛ به بیان دیگر، منطق، یک زبان قراردادی است که دارای ظاهر[۳] و سمانتیک[۴] می‌باشد.

در تعریف سوّم، ظاهر می‌گوید که چه عبارت‌هایی مُجازند؛ مثلاً در ریّاضیّات، عبارت x+2=5، به صورت گرامری درست می‌باشد، امّا x+=3 به صورت گرامری درست نمی‌باشد. به عنوان مثالی دیگر، در زبان ریّاضی، x+2≥y، یک جمله است، ولی x2+y>، یک جمله نمی‌باشد. و معنا یا سمانتیک مشخّص می‌نماید که یک عبارت در چه موقعی درست[۵]، یا در چه موقعی غلط(نادرست)[۶] می‌باشد؛ معنای x+2=5، در زمانی درست است که x=3 باشد و در غیر این صورت غلط می‌باشد. به عنوان مثالی دیگر، x+2≥y، در صورتی درست است که عدد x+2 کم‌تر از y نباشد؛ x+2≥y، در جهان، در صورتی که x=7, y=1 باشد، درست می‌باشد، امّا x+2≥y، در جهان، در صورتی که x=0, y=6 باشد، غلط است. *توجّه کنید که عبارت‌های منطقی باید یا درست باشند و یا غلط باشند؛ به بیان دیگر، «درجه‌ی درستی وجود ندارد».*

۱- Babylon> Britannica.com

۲- Babylon> Babylon English - English

۳- گرامر یا نحو(Syntax)

۴- Semantic

۵- true

۶- false

مثال‌هایی از منطق

منطق گزاره‌ای، مثل: $A \wedge B \Rightarrow C$

منطق مرتبه‌ی اوّل[۱]، مثل: $(\forall x)(\exists y)\ Mother(y,x)$

یادآوری و تکمیل– منطق گزاره‌ای

ظاهر

منطق گزاره‌ای، ساده‌ترین منطقی است که ایده‌های اساسی را توضیح می‌دهد. نمادهای گزاره‌ای، P1,P2 و..... می‌باشند.

اگر S، یک جمله باشد؛ $\neg S$، عبارت نقیض(منفی)[۲] آن می‌باشد.

برای عبارت‌های S_1 و S_2؛

$S_1 \wedge S_2$؛ AND این دو خواهد بود؛ در صورتی که ورودی‌ها هر دو صحیح باشند، نتیجه درست خواهد بود.

$S_1 \vee S_2$، OR این دو خواهد بود؛ در صورتی که از S_1 و S_2، دست‌کم یکی درست باشد، نتیجه درست خواهد بود.

$S_1 \Rightarrow S_2$ (بخوانید، اگر S_1، آنگاه S_2)، نتیجه‌گیری(استنباط)[۳] خواهد بود و در صورتی درست است که S_1، غلط باشد یا S_2 درست باشد و در صورتی نادرست است که S_1 درست باشد و S_2 نادرست باشد. مثلاً اگر حالا هوا بارانی باشد، آنگاه حالا هوا ابری می‌باشد. (توجّه کنید که برعکس این عبارت، همیشه، درست نمی‌باشد؛ یعنی، اگر حالا هوا ابری باشد، آنگاه الزاماً حالا هوا بارانی نمی‌باشد.) به عنوان مثال‌هایی دیگر،

---

۱ - First Order Logic(*FOL*)، در فصل «منطق مرتبه‌ی اوّل» این کتاب توضیح داده شده است.

۲ - negation

۳ - implication

$(P \wedge Q) \Rightarrow R$

$(A \Rightarrow B) \vee (\neg C)$

نکته:

استنباط به ما امکان استنتاج را می‌دهد.

یادآوری: $S_1 \Rightarrow S_2$ معادل است با «$\neg S_1 \vee S_2$».

$S_1 \Leftrightarrow S_2$، استنباط دو شرطی (اگر و فقط اگر[1]؛ بخوانید: S_1 اگر و فقط اگر S_2) خواهد بود و در صورتی که $S_1 \Rightarrow S_2$ درست باشد **و** $S_2 \Rightarrow S_1$ هم درست باشد، درست است.

ترتیب اولویّت‌ها[2]

به ترتیب از چپ به راست(→) عبارتند از:

$\neg \quad \wedge \quad \vee \quad \Rightarrow$

به عنوان مثال، $\neg A \vee B \Rightarrow C$ برابر است با: $((\neg A) \vee B) \Rightarrow C$.

مدل[3]های موجود در منطق گزاره‌ای

مطلب مهم:

تعریف- *مدل، نسبت دادن درست یا غلط به هر عبارت اُتمیک(تجزیّه ناپذیر) است؛*

مثلاً اگر A، B، C و D، سَمبُل‌های گزاره‌ای باشند؛ $m = \{A=true, B=false, C=false, D=true\}$ یک مدل است؛ همچنین $m' = \{A=true, B=false, C=false\}$ هم یک مدل است. 2^n *مدل می‌توانند برای* n *سمبل گزاره‌ای تعریف شوند.*[4]

۱- biconditional

۲- Order of Precedence

۳- model

۴- **توجّه- کنکور سراسری فنّـآوری اطّلاعات سال ۸۴-** برای مدل تعریف دیگری هم به این صورت وجود دارد: جاهایی که عبارت در آن‌ها درست است؛ به عبارت دیگر، سطرهایی از جدول صحّت عبارت که عبارت در آن‌ها درست(True) است.

سمانتیک‌ها یا معناهای منطق گزاره‌ای

هر مدل، درست /(یا) غلط بودن را برای هر سمبل یا نشان مشخّص می‌نماید. در زیر مدلی را برای سه نشان $P_{۱و۲}$ و $P_{۲و۲}$، $P_{۳و۱}$ می‌بینید:

$P_{۳و۱}$	$P_{۲و۲}$	$P_{۱و۲}$
↓	↓	↓
غلط	درست	درست

با این سه نشان ($P_{۳و۱}$، $P_{۲و۲}$ و $P_{۱و۲}$) می‌توانیم هشت (۸ = ۲۳) مدل داشته باشیم.

مثال- اگر $P_{1,2} = False$، $P_{2,2} = False$ و $P_{3,1} = True$ باشد، به عنوان مثال داریم:

$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = true \wedge (false \vee true) = true \wedge true = true$

جدول صحّت(درستی)[1] برای رابطه‌ها

P	Q	¬P	P∧Q	P∨Q	P⇒Q	P⇔Q
غلط	غلط	درست	غلط	غلط	درست	درست
غلط	درست	درست	غلط	درست	درست	غلط
درست	غلط	غلط	غلط	درست	غلط	غلط
درست	درست	غلط	درست	درست	درست	درست

بیان جملات دنیای هیولا به صورت گزاره‌ای

مثال- می‌خواهیم جمله‌ی زیر را به صورت گزاره‌ای بنویسیم:

«چاله‌ها باعث ایجاد هوای خوش، در خانه‌های اطراف می‌شوند.»

---

۱ - truth table؛ **یادآوری**- در منطق، جدولی است که همه‌ی ترکیب‌های ممکن ورودی‌ها و خروجی متناظر یک عبارت بولی را لیست می‌نماید. (Babylon> FOLDOC)

اگر داشته باشیم:

$P_{i,j}$، در صورتی که چاله‌ای در [i,j] وجود داشته باشد، درست می‌باشد.

$B_{i,j}$، در صورتی که هوایی خوش در [i,j] وجود داشته باشد، درست می‌باشد.

در نتیجه جمله‌ی بالا («چاله‌ها باعث ایجاد هوای خوش، در خانه‌های اطراف می‌شوند.») را به صورت زیر می‌توان نوشت:

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$
$$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

جمله‌ی «چاله‌ها باعث ایجاد هوای خوش، در خانه‌های اطراف می‌شوند.»، معادل است با، «یک خانه خوش هوا است، اگر و فقط اگر یک چاله در اطرافش وجود داشته باشد.»

استلزام[1]

تعریف- جمله‌ی A مستلزم جمله‌ی B است و می‌نویسیم، A |= B، اگر و فقط اگر B در هر مدلی که A درست است، درست باشد؛ به عنوان مثال، x+y=4 |= 4=x+y

تساوی[2]های منطقی

تعریف- دو عبارت در صورتی معادل منطقی(≡) می‌باشند که در همه‌ی مدل‌ها برابر باشند.

نکته:

α≡β، اگر α |= β و β |= α

برخی از معادل‌های منطقی را در زیر می‌بینید:

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$: جابه‌جایی‌پذیری[3] ∧

۱- entailment

۲- Equivalence

۳- commutativity

$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$: جابه‌جایی‌پذیری $\vee$

$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$: شرکت‌پذیری[۱] $\wedge$

$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$: شرکت‌پذیری $\vee$

$\neg(\neg\alpha) \equiv \alpha$: حذف نقیض دوگانه[۲]

$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$: مفهوم مخالف[۳]

$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$: حذف استنتاج[۴]

$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$: حذف شرط دوطرفه[۵]

$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$: قانون دِمورگِن[۶]

$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$: قانون دمورگن

$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$: توزیع‌پذیری[۷] $\wedge$ روی $\vee$

$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$: توزیع‌پذیری $\vee$ روی $\wedge$

صحّت(اعتبار)[۸] و قابلیّت برآورده‌سازی[۹]

تعریف- یک عبارت در صورتی *دارای صحّت(درستی)* است که در *همه‌ی* مدل‌ها درست باشد.

۱- associativity

۲- double-negation elimination

۳- contraposition

۴- implication

۵- biconditional elimination

۶- آگوسْتوسْ دِ مُورگِنْ(Augustus De Morgan)، ۱۸۷۱ – ۱۸۰۶ میلادی، ریّاضیدان و منطقدانی انگلیسی بود. وی قانون‌های دِمورگِن را تنظیم کرد و اِستِقراءِ ریّاضی(mathematical induction) را معرّفی نمود.
(http://en.wikipedia.org/wiki/Augustus_De_Morgan)

تصویری از او:

۷- distributivity

۸ - validity یا هَمانگویی(tautology)

۹ - satisfiability

مثال‌ها:

True

$A \Rightarrow A$

$A \vee \neg A$

$(A \wedge (A \Rightarrow B)) \Rightarrow B$

تعریف- ☑**مشابه کنکور سراسری فنّآوری اطّلاعات سال ۸۵-** یک عبارت *برآورده کننده* است، اگر در برخی از مدل‌ها درست باشد؛

مثال‌ها:

$A \vee B$

C

تعریف- یک جمله *برآورده کننده* نیست، اگر در هیچ مدلی درست نباشد؛ مثل $A \wedge \neg A$.

عبارت(کِلُز) هُرْن[۱] یا جمله‌ی هرن[۲]

تعریف- یک **لیترال(لفظ یا حرف)**[۳]، یک چیز تجزیّه ناپذیر[۴] یا نقیض آن است؛ مثل P یا نقیض آن که به صورت $\neg P$ است.

تعریف- ☑**کنکور سراسری فنّآوری اطّلاعات سال ۸۹:** یک **کِلُزْ**[۵]، یای لیترال‌ها است.

۱- Horn clause

۲- Horn sentence

۳- literal

۴- atom

۵- clause

تعریف- عبارت‌های هرن، عبارت‌هایی «یایی» هستند که در آنها حدّاکثر یک لیترال مثبت وجود دارد؛ مثلاً دو عبارت

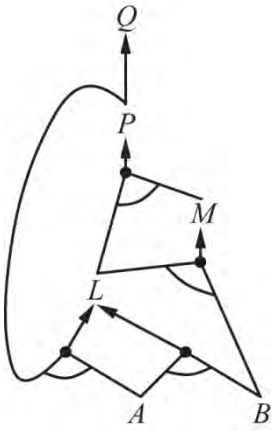
$\neg A \vee \neg B \vee \neg C \vee D$

$\neg A \vee \neg B$

عبارت هرن هستند، ولی $A \vee B$، عبارت هرن نمی‌باشد، چون دارای دو لیترال مثبت A و B است؛ (☑**کنکور سراسری مکاترونیک سال ۸۴**- عبارت‌های هرن می‌توانند به صورت استنتاج‌هایی با یک نتیجه(تالی) نوشته شوند؛ مثلاً $\neg A \vee \neg B \vee \neg C \vee D$ می‌تواند به صورت $A \wedge B \wedge C \Rightarrow D$ نوشته شود.) عبارت‌های هرن، مبنا یا پایه‌ی برنامه‌نویسی منطقی هستند. نامگذاری عبارت هرن، به افتخار آلْفِرِدْ هُرْنْ[1] بود.

مثال- در زیر، مجموعه‌ای از کلزهای هرن به همراه گراف AND-OR متناظر با آنها آورده شده است:[2]

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



عبارت(کلز) صریح(قطعی، مسلّم)[3]

تعریف- عبارت‌های صریح، عبارت‌هایی «یایی» هستند که (☑**کنکور سراسری فنّآوری اطّلاعات سال ۹۱**- در آنها دقیقاً یک لیترال مثبت وجود دارد)؛ مثلاً عبارت

---

۱ - Alfred Horn، ۲۰۰۱ – ۱۹۱۸ میلادی، یکی از استادان ریّاضیّات در دانشگاه شهر لُس آنجِلِسِ ایالت کالیفورنیا(University of California Los Angeles = UCLA)ی کشور ایالات متّحده‌ی آمریکا بود.
(http://www.math.ucla.edu/info/horn.html، این پایگاه اینترنتی، متعلّق به دانشکده‌ی ریّاضیّات دانشگاه UCLA می‌باشد.)

۲ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل هفتم، عامل‌های منطقی(Logical Agents)، صفحه‌ی ۲۵۹

۳ - definite clause

$\neg A \vee \neg B \vee \neg C \vee D$

یک عبارت صریح است؛ در حالی که عبارت‌های زیر صریح نیستند:

$\neg A \vee \neg B$

$\neg A \vee B \vee C$

صورت نرمال ربطْ دهنده[1](CNF)

اغلب مفید است که فرمول‌ها را به صورت‌های نرمال بنویسیم. صورت‌های نرمال، مثل CNF، با آنچه که قبل از نرمال کردن بودند، فرقی ندارند و تفاوت آنها در ظاهر آنهاست و برای استدلال مناسب‌ترند.

تعریف- یک فرمول، به صورت CNF است، اگر به صورت $A_1 \wedge A_2 \wedge ... A_k$ باشد؛ که A_i، تشکیل شده از یای(OR) گزاره‌ها یا نقیض آنها. به عنوان مثال،

$(p \vee q) \wedge r \wedge (\neg p \vee \neg r \vee s)$، به صورت CNF است.

$\neg(p \vee q) \wedge r \wedge (\neg p \vee \neg r \vee s)$، به صورت CNF نمی‌باشد.

$(p \vee q) \wedge r \wedge (p \Rightarrow (\neg r \vee s))$، به صورت CNF نمی‌باشد.

نکته:

هر عبارت منطق گزاره‌ای می‌تواند به فرم نرمال ربط دهنده تبدیل شود.

مثال:

فرمول نهائی (به صورت CNF)	فرمول اوّلیّه
$(A \vee B) \wedge (A \vee C)$	$A \vee (B \wedge C)$
$(B \vee A) \wedge (C \vee A)$	$(B \wedge C) \vee A$

مثالی برای تبدیل یک عبارت به CNF

مثلاً برای تبدیل عبارت $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ به ترتیب زیر عمل می‌کنیم:

۱. حذف کردن $\Leftrightarrow$؛ برای این کار، $\alpha \Leftrightarrow \beta$ را با $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$ جایگزین می‌کنیم:

---

۱ - Conjunctive Normal Form یا به اختصار *CNF*؛ که به آن «فرم نرمال عطفی» هم گفته می‌شود.

۱) $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

۲. حذف کردن $\Rightarrow$؛ برای این کار، $\alpha \Rightarrow \beta$ را با $\neg\alpha \vee \beta$ جایگزین می‌کنیم:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$$

۳. «$\neg$» را با استفاده از قوانین دمورگِن و قرینه‌سازی دوگانه(حذف نقیض دوگانه) به درون عبارات حرکت دهید؛ برای این مثال، با استفاده از قوانین دمورگن داریم:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$$

۴. قوانین توزیع و گسترش را روی $\vee$ و $\wedge$ به کار ببرید:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$

روش(شیوه یا متد)های اثبات[۱]

یک اثبات، یک رشته از جملات است که هر جمله یا یک مقدّم است و یا یک جمله‌ی به دست آمده از جملات قبلی در اثبات که با استفاده از یکی از قانون‌های استنتاج به دست آمده است. آخرین جمله، *قضیّه(هدف یا پرسش یا پرس و جو[۲])* نام دارد و می‌خواهیم آن را ثابت کنیم. به‌طورکلّی، متدهای اثبات به دو دسته‌ی زیر تقسیم می‌شوند:

استفاده از قوانین استنتاج[۳]: در این روش معمولاً به تبدیل(ترجمه‌ی) عبارت‌ها به صورت یک فرم نُرمال، مثل صورت نرمال ربط دهنده نیازمندیم و برای اثبات، یک سری از قوانین استنتاج را به کار می‌بریم؛ مثل: روش‌های زنجیره‌ی پیشرُو(مستقیم)[۴]، زنجیره‌ی پَسرُو(معکوس)[۵] و تحلیل.

برّرسی مدل[۶]: در این روش، از جدول صحّت استفاده می‌نماییم. مثل: روش‌های جریان معکوس(برگشت به عقب) بهبود یافته[۷] و جستجوی مکاشفه‌ای در فضای مدل[۸].

تعریف- صحّت(صحیح بودن)[۱]: در این حالت، استنتاج‌ها فقط جمله‌های مستلزم[۲] را تولید می‌کنند.

۱- Proof methods
۲- query
۳- Inference rules
۴- forward chaining
۵- backward chaining
۶- Model checking
۷- Improved backtracking: *D*avis-*P*utnam-*L*ogemann-*L*oveland(*DPLL*)
۸- Heuristic search in model space: Walksat

تعریف- کامل بودن[۳]: در این حالت، استنتاج‌ها می‌توانند تمام جمله‌های مستلزم را تولید کنند.

زنجیره‌ی پیشرو(مستقیم)

مطلب مهم:

ایده: با جمله‌های اتمیک(واقعیّات، حقایق)[۴] موجود در پایگاه دانش شروع کنید و از قیاس استثنائی[۵]، به سمت جلو، با اضافه کردن جمله‌های اتمیک جدید استفاده کنید، تا اینکه استنتاج‌های بیش‌تر جدید، ممکن نباشند.[۶]؛ به بیان دیگر، هر قانونی که مقدّم‌هایش در پایگاه دانش برآورده می‌شوند را اجرا کن و تالیش را به پایگاه دانش اضافه کن و این کار را تا زمانی که پرسش پیدا شود، ادامه بده.

نکته:

زنجیره‌ی پیشرو، برای پایگاه دانش هرن، صحیح و کامل است.

۱- soundness

۲- entailed

۳- completeness

۴- facts

۵- Modus Ponens؛ از یک استنباط ($\alpha \Rightarrow \beta$) و مقدّم استنباط (α) می‌توانید تالی (β) را استنتاج کنید:

مقدّم $\alpha \Rightarrow \beta,\ \alpha$

تالی β

مثال- از «بارش باران(raining) باعث خیس شدن(soggy) حیات(courts) می‌شود.» و «بارش باران» می‌توان «خیس شدن حیات» را استنتاج کرد. (مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی)

۶- کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل نهم، استنتاج در منطق مرتبه‌ی اوّل(Inference in First-Order Logic)، صفحه‌ی ۳۳۰

زنجیره‌ی پسرو(معکوس)

مطلب مهم:

این الگوریتم‌ها از هدف، به طرف عقب حرکت می‌کنند تا به جمله‌های اتمیک(واقعیّات، حقایق)ی برسند که از اثبات پشتیبانی می‌کنند.[۱]

تحلیل در منطق گزاره‌ای

مطلب مهم:

قانون کاملی برای استنتاج وجود دارد، که این قانون، *تحلیل نام* دارد. قانون تحلیل پایه، شبیه این می‌باشد، $A \vee B$ و $B \vee C$، به ما اجازه می‌دهند که $A \vee C$ را استنتاج کنیم(به بیان دیگر، $(A \vee B) \wedge (\neg B \vee C) \Rightarrow (A \vee C)$). *در موقع تحلیل باید پایگاه دانش ما به صورت CNF باشد.*

قانون تحلیل در منطق گزاره‌ای- اگر داشته باشیم:

$P_1 \vee P_2 \vee \ldots \vee P_n$

$\neg P_1 \vee Q_2 \vee \ldots \vee Q_m$

آنگاه

$$P_2 \vee \ldots \vee P_n \vee Q_2 \vee \ldots \vee Q_m$$

مثال‌ها:

P و $\neg P \vee Q$، Q را نتیجه می‌دهد.

$(\neg P \vee Q)$ و $(\neg Q \vee R)$، $\neg P \vee R$ را نتیجه می‌دهد.

P و $\neg P$، False(غلط) را نتیجه می‌دهد(تناقض).

$(P \vee Q)$ و $(\neg P \vee \neg Q)$، True(درست) را نتیجه می‌دهد.

---

۱ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل نهم، استنتاج در منطق مرتبه‌ی اوّل(Inference in First-Order Logic)، صفحه‌ی ۳۳۷

مطلب مهم:

روال‌های استنتاج براساس تحلیل، با استفاده از اصل اثبات به وسیله‌ی تناقض کار می‌کنند؛ برای نشان دادن اینکه $KB \models \alpha$*، نشان می‌دهیم که* $(KB \wedge \neg\alpha)$ *قابل برآورده‌سازی نمی‌باشد؛ برای این کار،*

۱) ابتدا $KB \wedge \neg\alpha$ *را به معادل* CNF *آن تبدیل کنید.*

۲) قانون تحلیل را روی کلزهای نتیجه شده اعمال کنید؛ با داشتن یکی از دو مورد زیر این پردازه پایان می‌یابد:

الف) کلزهای جدیدی که بتوانند اضافه شوند وجود نداشته باشند، که در این صورت، KB، α *را مستلزم نمی‌کند.*

ب) تحلیل دو کلز، منجر به یک کلز تُهی شود، که در این صورت، KB، α *را مستلزم می‌کند. (**)*

مثال- اگر $KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$ و $\alpha = \neg P_{1,2}$ باشد، نشان دهید که $KB \models \alpha$.

برای این کار، $(KB \wedge \neg\alpha)$ را به CNF تبدیل می‌کنیم و روال استنتاج براساس تحلیل را بر روی آن اعمال می‌کنیم.

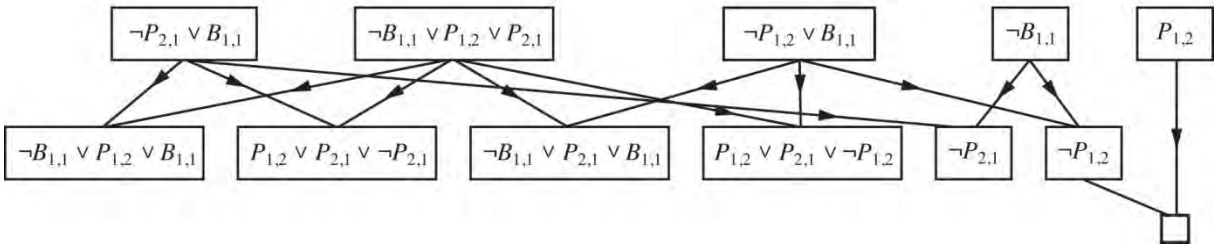
قبلاً تبدیل $(B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}))$ به CNF را که حاصل

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$

شد را دیدیم؛ در نتیجه $KB \wedge \neg\alpha$ به صورت

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1}) \wedge \neg B_{1,1} \wedge P_{1,2}$$

می‌باشد؛ در زیر بخشی از اعمال تحلیل بر روی جفت کلزهای بالا را می‌بینید:



پس چون طبق بند (**)، تحلیل دو کلز $\neg P_{1,2}$ و $P_{1,2}$ منجر به یک کلز تهی شده است، نتیجه می‌گیریم که KB، α را مستلزم می‌کند. *توجّه کنید که کلز تهی، در شکل بالا، با یک مربع کوچک توخالی (□) نشان داده شده است.*[1]

مثالی برای برّرسی استلزام با استفاده از روش برّرسی مدل

اگر $\alpha = A \vee B$ باشد و $KB = (A \vee C) \wedge (B \vee \neg C)$ باشد؛ با استفاده از متد شمارش[2] (برّرسی مدل)، آیا $KB \models \alpha$ درست است؟

---

۱ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل هفتم، عامل‌های منطقی(Logical Agents)، صفحه‌های ۲۵۳، ۲۵۴ و ۲۵۵ و مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

۲ - enumeration method

برای این کار برّرسی می‌کنیم که *آیا در هر جایی که KB درست است آیا α هم درست است یا نه؟*؛ جدول صحّت KB و α به صورت زیر است:

A	B	C	**KB = (A ∨ C) ∧ (B ∨ ¬C)**	**α = A ∨ B**
غلط	غلط	غلط	غلط	غلط
غلط	غلط	درست	غلط	غلط
غلط	درست	غلط	غلط	درست
غلط	درست	درست	درست	درست
درست	غلط	غلط	درست	درست
درست	غلط	درست	غلط	درست
درست	درست	غلط	درست	درست
درست	درست	درست	درست	درست

چنانچه در جدول زیر هم می‌بینید، در هر جایی که KB درست است، α هم درست است، *در نتیجه $KB \models \alpha$ برقرار است:*

A	B	C	**KB = (A ∨ C) ∧ (B ∨ ¬C)**	**α = A ∨ B**
غلط	غلط	غلط	غلط	غلط
غلط	غلط	درست	غلط	غلط
غلط	درست	غلط	غلط	درست
غلط	درست	درست	درست	درست
درست	غلط	غلط	درست	درست
درست	غلط	درست	غلط	درست
درست	درست	غلط	درست	درست
درست	درست	درست	درست	درست

نکته:

متد شمارش(برّرسی مدل)، برای منطق گزاره‌ای، صحیح و کامل است؛ ولی برای n سمبل، دارای پیچیدگی زمانی $O(2^n)$ است؛ پس باید از روش‌های هوشمندتر استفاده نمائیم.[1]

۱ – مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

چکیده‌ی مطلب‌های فصل دهم

منطق، یک زبان قراردادی است که دارای ظاهر(گرامر) و معنا(سمانتیک) می‌باشد؛ ظاهر می‌گوید که چه عبارت‌هایی مُجازند و معنا مشخّص می‌نماید که یک عبارت، در چه موقعی درست یا در چه موقعی غلط(نادرست) می‌باشد.

از منطق می‌توانیم برای بیان دانش عامل استفاده نماییم.

استنباط، به ما امکان استنتاج را می‌دهد.

مدل، نسبت دادن درست یا غلط به هر عبارت اتمیک(تجزیّه ناپذیر) است.

یک فرمول، به صورت CNF است، اگر به صورت $A_1 \wedge A_2 \wedge ... A_k$ باشد؛ که A_i، تشکیل شده از یای(OR) گزاره‌ها یا نقیض آنها.

صورت‌های نرمال، مثل CNF، با آنچه که قبل از نرمال کردن بودند، فرقی ندارند و تفاوت آنها در ظاهر آنهاست و برای استدلال، مناسب‌ترند.

هر عبارت منطق گزاره‌ای می‌تواند به فرم نرمال ربط دهنده تبدیل شود.

عبارت‌های هرن، عبارت‌هایی «یایی» هستند که در آنها حدّاکثر یک لیترال مثبت وجود دارد.

عبارت‌های صریح، عبارت‌هایی «یایی» هستند که در آنها دقیقاً یک لیترال مثبت وجود دارد.

به‌طورکلّی، متدهای اثبات به دو دسته‌ی «استفاده از قوانین استنتاج» و «برّرسی مدل» تقسیم می‌شوند.

در زنجیره‌ی پیشرو(مستقیم)، با جمله‌های اتمیک(واقعیّات، حقایق) موجود در پایگاه دانش شروع کنید و از قیاس استثنائی، به سمت جلو، با اضافه کردن جمله‌های اتمیک جدید استفاده کنید، تا اینکه استنتاج‌های بیش‌تر جدید، ممکن نباشند.؛ به

بیان دیگر، هر قانونی که مقدّم‌هایش در پایگاه دانش برآورده می‌شوند را اجرا کن و تالیش را به پایگاه دانش اضافه کن و این کار را تا زمانی که پرسش پیدا شود، ادامه بده.

زنجیره‌ی پیشرو، برای پایگاه دانش هرن، صحیح و کامل است.

در زنجیره‌ی پسرو(معکوس)، از هدف، به طرف عقب، حرکت کنید تا به جمله‌های اتمیک(واقعیّات، حقایق)ی برسید که از اثبات پشتیبانی می‌کنند.

قانون تحلیل پایه، به صورت $(A \vee B) \wedge (\neg B \vee C) \Rightarrow (A \vee C)$ است.

در موقع تحلیل باید پایگاه دانش ما به صورت CNF باشد.

روال‌های استنتاج براساس تحلیل، با استفاده از اصل اثبات به وسیله‌ی تناقض کار می‌کنند؛ برای نشان دادن اینکه $KB \models \alpha$، نشان می‌دهیم که $(KB \wedge \neg \alpha)$ قابل برآورده‌سازی نمی‌باشد.

یادآوری یا تکمیل مطلب‌های فصل دهم

تعریف- منطق را تعریف کنید.

جواب- سیستم نمادی(سمبلی) رسمی برای ارائه(بازنمائی) و استنتاج.[۱]

تعریف- «صحّت(اعتبار، همانگویی)» را تعریف کنید.

جواب- یک عبارت(جمله)، دارای صحّت است، اگر در تمام جهان‌ها، تحت تمام تفسیرها درست باشد.[۲]

سؤال- در منطق گزاره‌ای، برای چهار(۴) گزاره‌ی A, B, C, D چند مدل ممکن داریم؟

جواب- ۴ متغیّر داریم، بنابراین، ۱۶ = $۲^۴$ مدل خواهیم داشت.[۳]

درست یا غلط- عبارت «$(A \Leftrightarrow B) \wedge (\neg A \vee B)$» برآورده کننده است.

جواب- «**درست» است؛** مثلاً اگر A= false و B= false باشد، یا اگر A= true و B= true باشد، این عبارت درست خواهد بود.[۱]

---

۱ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۲ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، پاییز سال ۱۹۹۲ میلادی

۳ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، پاییز سال ۱۹۹۹ میلادی

درست یا غلط– دو فرمول منطق گزاره‌ای که از لحاظ گرامری(نحوی یا سمانتیکی) متفاوت هستند، هیچوقت نمی‌توانند دارای مدل‌های یکسانی باشند.

جواب– **«غلط» است؛** مثلاً A و $A \vee A$ از لحاظ گرامری، متفاوت هستند، ولی دارای مدل‌های یکسانی هستند.[۲]

درست یا غلط– اگر یک فرمول گزاره‌ای، در یک مدل دارای کاردینالیتی[۳] n، برآورده‌کننده باشد، که n، عدد صحیح مثبتی است، آنگاه تمام مدل‌ها دارای کاردینالیتی n خواهند بود.

جواب– **«غلط» است؛** مثال نقض: $A \vee B$، هم با $\{A, B\}$، که دارای کاردینالیتی ۲ است، برآورده می‌شود و هم با $\{A\}$، که دارای کاردینالیتی ۱ است.[۴]

تعریف– کلز هرن را تعریف کنید.

جواب– کلزی که حداّکثر یک لیترال مثبت دارد.[۵]

درست یا غلط– فرمول گزاره‌ای $(P \wedge \neg R) \rightarrow (Q \rightarrow R)$ می‌تواند به یک عبارت هُرن تبدیل شود.

جواب– **«درست» است:**

$$(P \wedge \neg R) \rightarrow (Q \rightarrow R) = \neg(P \wedge \neg R) \vee (\neg Q \vee R) = \neg P \vee R \vee \neg Q \vee R =$$

$$\boxed{\begin{array}{c} \neg P \vee \neg Q \vee R \\ \text{یا} \\ (P \wedge Q) \rightarrow R^{۶} \end{array}}$$

درست یا غلط– جدول‌های صحّت می‌توانند برای ثابت کردن درستی یا غلط بودن هر عبارت گزاره‌ای استفاده شوند.

---

۱ – آزمون کلاسی درس «آشنایی با هوش مصنوعی» دانشکده‌ی علوم کامپیوتر و مهندسی دانشگاه ایالت تگزاس کشور آمریکا، ۲۰ آوریل سال ۲۰۱۰ میلادی

۲ – آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۳۱ ژانویّه‌ی سال ۲۰۱۱ میلادی

۳ – cardinality، **یادآوری**– تعداد عنصرهای یک مجموعه‌ی ریّاضی داده شده است. (Babylon> Merriam-Webster Collegiate® Dictionary)

۴ – آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۱۴ جولای سال ۲۰۱۱ میلادی

۵ – کوییز شماره‌ی ۳ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۶ – آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۱۵ فوریّه‌ی سال ۲۰۱۲ میلادی و آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، پاییز سال ۱۹۹۷ میلادی

جواب- «**درست**» **است**.[۱]

سؤال- با استفاده از جدول صحّت ثابت کنید $(A \wedge B) \models (A \Leftrightarrow B)$.

جواب- از آنجایی که برای هر سطر از جدول صحّت، در هر جا که $A \wedge B$ درست است، $A \Leftrightarrow B$ هم درست است، این استلزام منطقی ثابت می‌شود:[۲]

A	B	$A \wedge B$	$A \Leftrightarrow B$
T	T	T	T
T	F	F	F
F	T	F	F
F	F	F	T

سؤال- سیستم استنتاجی‌ای که می‌تواند هر جمله‌ای که مستلزم می‌شود را استنتاج کند، چه نام دارد؟

جواب- کامل.[۳]

سؤال- سیستم استنتاجی‌ای که فقط جمله‌های مستلزم را استنتاج می‌کند، چه نام دارد؟

جواب- صحیح(دارای صحّت).[۴]

سؤال- «استلزام» را با «استنتاج» مقایسه کنید.

جواب- اگر نتایج پایگاه دانش مثل یک «پشته‌ی علف»[۵] باشد و α یک سوزن[۱] باشد؛ استلزام، وجود سوزن در پشته‌ی علف و استنتاج، پیدا کردن آن است.[۲]

---

۱ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۱۹۹۶ میلادی

۲ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «چاک دایِر(Chuck Dyer)»، دانشگاه ویسکانسین-مَدیسُون(Wisconsin-Madison) کشور آمریکا، ۲۱ اکتبر سال ۲۰۰۹ میلادی

۳ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۴ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۵ - haystack

سؤال- جمله‌های زیر را به صورت CNF بنویسید.

الف) $Q \Rightarrow S$

جواب- $\neg Q \vee S$

ب) $P \Leftrightarrow Q$

جواب- $(\neg P \vee Q) \wedge (P \vee \neg Q)$[۳]

پ) $B \Leftrightarrow (\neg(P \wedge Q))$

جواب-

۱) حذف $\Leftrightarrow$

$$(B \Rightarrow (\neg(P \wedge Q))) \wedge ((\neg(P \wedge Q)) \Rightarrow B)$$

۱ - needle

۲ - درس هوش مصنوعی دانشگاه Western Australia کشور استرالیا

۳ - کوییز شماره‌ی ۳ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۲) حذف $\Rightarrow$

$$(\neg B \vee (\neg(P \wedge Q))) \wedge (\neg(\neg(P \wedge Q)) \vee B)$$

۳) حرکت دادن $\neg$ به درون، با استفاده از قوانین دمورگن

$$(\neg B \vee \neg P \vee \neg Q) \wedge ((P \wedge Q) \vee B)$$

۴) به کار بردن قانون توزیع $\wedge$ روی $\vee$

$$(\neg B \vee \neg P \vee \neg Q) \wedge (P \vee B) \wedge (Q \vee B)$$

۵) هر کلز را به طور جداگانه، به عنوان یک جمله، در KB بنویسید

$$(\neg B \vee \neg P \vee \neg Q)$$
$$(P \vee B)$$
$$(Q \vee B)^{1}$$

ت) $A \Rightarrow (B \Rightarrow (C \Rightarrow D))$

جواب-

۱) حذف $\Rightarrow$

$$(\neg A \vee (\neg B \vee (\neg C \vee D)))$$

۲) حذف پرانتزهای غیر ضروری و نوشتن کلز، به عنوان یک جمله، در KB

$$(\neg A \vee \neg B \vee \neg C \vee D)^{2}$$

مطلب- زنجیره‌ی پیشرو(مستقیم) در مقایسه با زنجیره‌ی پسرو(معکوس) ممکن است کارهای زیادی را انجام دهد که به هدف نامربوط هستند؛ و (**☑مورد استفاده درکنکور سراسری فنّـآوری اطّلاعات سال‌های ۹۰ و ۹۱-** پیچیدگی زمانی زنجیره‌ی پسرو، بسته به اندازه‌ی پایگاه دانش می‌تواند به مراتب کم‌تر از پیچیدگی زمانی خطی باشد.)[۳]

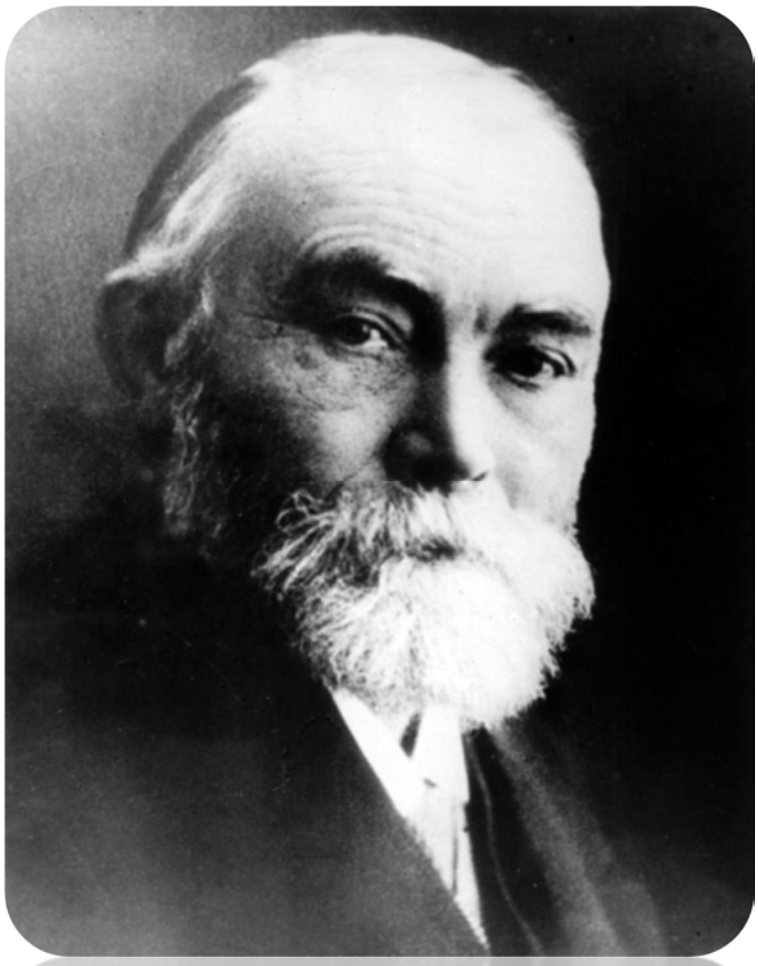
---

۱ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، زمستان سال ۲۰۱۲ میلادی

۲ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، زمستان سال ۲۰۱۲ میلادی

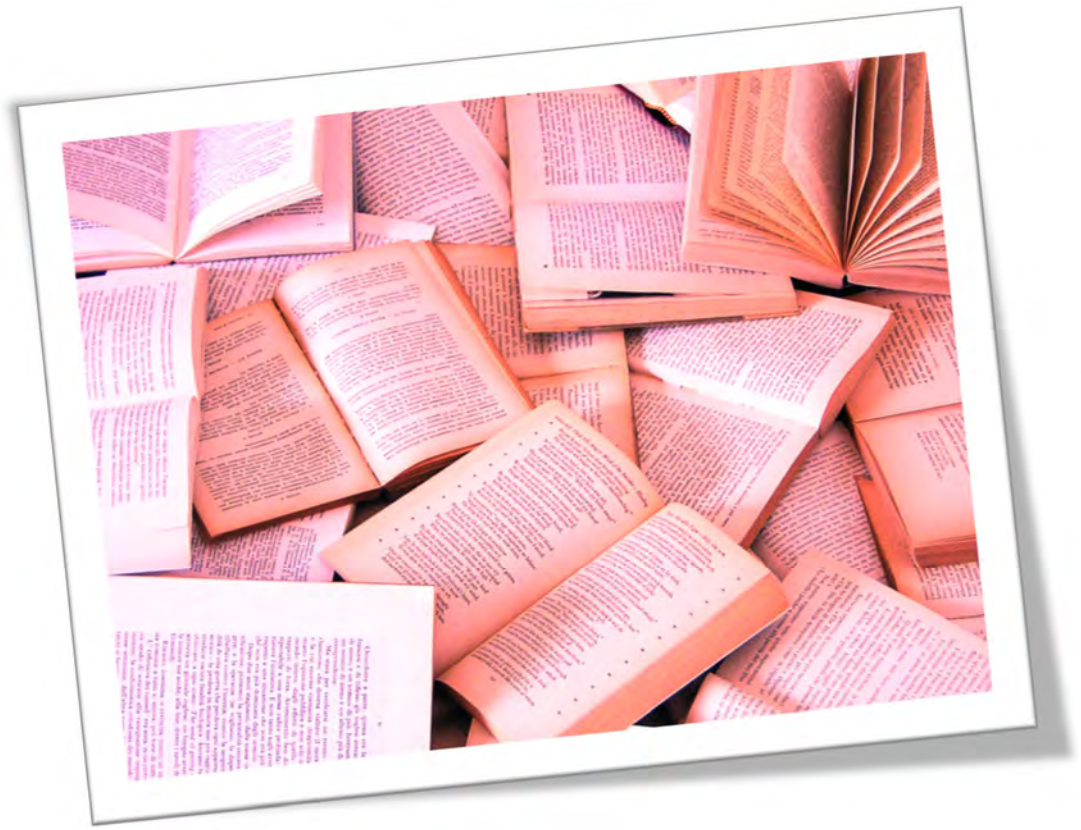
۳ - مطلب‌های درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، زمستان سال ۲۰۱۲ میلادی

فصل یازدهم



۱ منطق مرتبه‌ی اوّل[۱]

۱ - تصویر، گاتْلابْ فریجْ(Gottlob Frege)، ۱۹۲۵ – ۱۸۴۸ میلادی (http://en.wikipedia.org/wiki/Gottlob_Frege)، ریّاضیدان و منطقدان آلمانی می‌باشد؛ او در سال ۱۸۷۹ میلادی منطق مرتبه‌ی اوّل را به وجود آورد.



فهرست برخی از عنوان‌های نوشته‌ها

---

۱ - *F*irst-*O*rder *L*ogic(*FOL*)، که به آن، منطق مُسنَدات(predicate logic) هم گفته می‌شود.

☹منطق گزاره‌ای، زبانی ضعیف است و برخلاف زبان طبیعی، قدرت بیان اندکی دارد

به عنوان مثال، به مشکل بیان اطّلاعات زیر توجّه کنید:

۱- هر انسانی می‌میرد.

۲- باراک اُباما[1] انسان است.

۳- باراک اباما می‌میرد.

چگونه این جمله‌ها را بیان نماییم تا بتوانیم جمله‌ی سوّم را از دو جمله‌ی اوّل نتیجه بگیریم؟ در منطق گزاره‌ای باید سمبل‌های گزاره‌ای را برای همه یا بخشی از جمله به وجود بیاوریم؛ برای مثال، ممکن است از سمبل P، برای «انسان»؛ سمبل Q، برای «میرا»؛ و از سمبل R، برای «باراک اباما» استفاده نماییم. بنابراین، سه جمله‌ی بالا به صورت زیر بیان می‌شوند:

$P \Rightarrow Q$ $R \Rightarrow P$ $R \Rightarrow Q$

در این صورت، گرچه جمله‌ی سوّم به وسیله‌ی دو جمله‌ی اوّل به دست می‌آید، به یک سمبل ساده‌ی R، برای نشان دادن یک فرد (باراک اباما) نیاز داریم، که عضوی از کلاس «انسان» و «میرا» می‌باشد؛

---

۱ - Barack Obama، متولّد ۱۹۶۱ میلادی، چهل و چهارمین رئیس جمهور و رئیس جمهور کنونی کشور ایالات متّحده‌ی آمریکا می‌باشد.
تصویری از او را می‌بینید:

(http://en.wikipedia.org/wiki/Barack_Obama)

مطلب مهم:

برای نمایش دادن فردهای دیگر باید سمبل‌های مختلف دیگری را برای هریک از فردها، برای بیان اینکه همه‌ی افراد که «انسان» هستند، «میرا» هم هستند، مورد استفاده قرار دهیم.

به عنوان مثالی دیگر، در دنیای هیولا(wumpus) با استفاده از منطق گزاره‌ای نمی‌توانیم بگوییم: «چاله‌ها باعث هوای مطلوب در خانه‌های مجاور می‌شوند»، *به جز اینکه برای هر خانه یک عبارت را بنویسیم.*

منطق مرتبه‌ی اوّل

منطق مرتبه‌ی اوّل، عیب‌های منطق گزاره‌ای را ندارد و تمام ویژگی‌های منطق گزاره‌ای را داراست. منطق مرتبه‌ی اوّل، جهان را براساس موردهای زیر مدلسازی می‌کند:

تعریف- اشیا[۱]، معمولاً چیزها هستند؛ به عنوان مثال، مَردُم، خانه‌ها، اعداد، باراک اباما، دانشجویان، استادان، شرکت‌ها، خودروها و خورشید.

تعریف- ویژگی‌ها(خصوصیّت‌ها)[۲]**ی اشیا**، باعث فرق، میان آنها و اشیای دیگر می‌شوند. مثل [رنگ] آبی، بیضی، [عدد] زوج[۳] و دراز(بلند).

تعریف- ارتباط‌ها[۴]، میان مجموعه‌های اشیا وجود دارند؛ به عنوان مثال، «باراک اباما زنده است.»، «خورشید، داغ است.» و «لیلی پاتر[۵]، مادر هَری پاتر[۶] است.»

تعریف- عملکردها(تابع‌ها)[۷]، زیرمجموعه‌ای از ارتباط‌ها هستند، طوری که برای هر ورودی[۸] فقط یک مقدار[۹] وجود داشته باشد. مثل:

father-of(به معنی لغوی «پدرِ»)

۱- objects
۲- properties
۳- even
۴- relations
۵- Lily Potter
۶- Harry Potter
۷- functions
۸- input
۹- value

best-friend(به معنی لغوی «بهترین دوستِ»)

one-more-than(به معنی لغوی «یکی بیش‌تر از»)

end-of(به معنی لغوی «پایانِ»)

ظاهر(گرامر) یک زبان مرتبه‌ی اوّل

شامل موردهای زیر است:

۱- **یک مجموعه از ثابت‌ها**[۱]**؛** به طور حسّی یک ثابت نام یک شیء می‌باشد. مثل:
KingJohn, 2, UCB, Koblenz, C, …

۲- **یک مجموعه از متغیّرها؛** نظیر x، y، z و....

۳- **یک مجموعه از سَمبُل(نشان)ها؛** که برای نام تابع استفاده می‌شوند، نظیر f، g، h، LeftLegOf، Sqrt و....

۴- **رابط‌ها**[۲]**؛** مثل رابط‌های «استاندارد» منطق گزاره‌ای هستند و عبارتند از: ∨، ∧، ¬، → و ....

۵- **تساوی:** =

۶- **کمّیّت‌سنج‌ها(سورها)**[۳]**:** در زبان طبیعی، از «هر...» یا «همه...» استفاده می‌کنیم؛ برای مثال، «هر شخصی دارای یک پدر(والد) است.»، «همه‌ی پرندگان پرواز می‌کنند.»؛ این مطلب‌ها بیان می‌کنند که همه‌ی اشیا دارای یک ویژگی معیّن هستند. همچنین از «برخی...» هم استفاده می‌نماییم؛ برای مثال، «برخی از پرندگان نمی‌توانند پرواز نمایند.»، «برخی از افراد از شطرنج لذّت می‌برند.»؛ این مطلب‌ها بیان می‌کنند که دست‌کم یک شیء، دارای یک خصوصیّت معیّن می‌باشد. دو کمّیّت‌سنج عبارتند از: یکی، ∀، بخوانید «برای همه(هر)»؛ که سور عمومی[۴] نامیده می‌شود و دیگری، ∃، بخوانید «وجود دارد»؛ که سور وجودی[۵] نام دارد. به طور معمول، سورها، با متغیّرها و متغیّرها، با سورها به کار می‌روند.

۱- constants
۲- Connectives
۳- Quantifiers
۴- universal quantifier
۵- existential quantifier

جمله‌ها[۱] از واژگان(تِرِم‌ْها، لغت‌ها)[۲] و اتم‌ها[۳] درست شده‌اند

تعریف- واژه، یک ثابت و یا یک متغیّر می‌باشد، در ضمن، یک عبارت[۴] $f(t_1, \ldots, t_n)$ که در آن، f، سمبل تابعی است، در صورتی یک واژه است که همه‌ی t_iها واژه باشند.

تعریف- واژه‌ای که دارای هیچ متغیّری نباشد، یک واژه‌ی زمینه[۵] است.

تعریف- جمله‌ی اتمیک[۶]، یک گزاره‌ی دارای n واژه است و دارای مقدار درست یا غلط می‌باشد.

تعریف- جمله‌ی پیچیده(مرکّب)[۷]، از جمله‌های اتمیک متّصل شده به وسیله‌ی رابط‌های منطقی تشکیل شده است.

مثال- اگر P و Q، دو جمله باشند؛ گزینه‌های زیر جمله‌های پیچیده هستند.

$\neg P,\ P \vee Q,\ P \wedge Q,\ P \rightarrow Q,\ P \leftrightarrow Q$

تعریف- جمله‌ی سوردار[۸]، دارای سورهای عمومی($\forall$) و وجودی($\exists$) است.

تعریف- فرمول خوبْ ساخته[۹]، جمله‌ای است که دارای هیچ متغیّر آزادی نباشد؛ یعنی، متغیّرهای آن به وسیله‌ی سورهای عمومی و وجودی محدود شده باشند.

مثال- در $(\forall x)P(x,y)$، متغیّر x به وسیله‌ی سور عمومی محدود شده است، ولی متغیّر y آزاد است.

مثال:

$\text{Sibling}^{1}(\text{John}^{2},\text{Richard}^{3}) \Rightarrow \text{Sibling}(\text{Richard},\text{John})$

۱- sentences
۲- terms
۳- atoms
۴- expression
۵- ground term
۶- atomic sentence
۷- complex sentence
۸- quantified sentence
۹- well-formed formula(*wff*)

در عبارت بالا که دارای این معنی است که «اگر John برادر Richard باشد، آنگاه Richard هم برادر John است.»؛ در سمت چپ، Sibling، گزاره می‌باشد. John، واژه است. Richard، واژه است. در طرف راست، Sibling، گزاره است. Richard، واژه است. John، واژه است.

در این عبارت،

Sibling(KingJohn,Richard)، عبارت اتمیک است.

Sibling(Richard,KingJohn)، عبارت اتمیک است.

و کلّ عبارت(Sibling(Richard,John) ⇒ Sibling(John,Richard))، عبارت پیچیده(مرکّب) می‌باشد.

فرمول مرتبه‌ی اوّل

تعریف- در صورتی که P، یک سمبل گزاره‌ای باشد و t_n , ... ,t_1، واژگان باشند، آنگاه $P(t_1, \dots, t_n)$، یک فرمول منطق مرتبه‌ی اوّل می‌باشد؛

در صورتی که φ و ψ، فرمول باشند، آنگاه ψ∨φ، φ∧φ، ¬φ و φ→ψ هم فرمول هستند؛

در صورتی که φ، یک فرمول باشد و x، یک متغیّر باشد، آنگاه ∀x (φ) و ∃x (φ) هم فرمول هستند.

معنی یک فرمول منطق مرتبه‌ی اوّل

در منطق مرتبه‌ی اوّل، معنی ∨، ∧، ¬ و → مثل معنی آنها در منطق گزاره‌ای می‌باشد، به عنوان مثال، در صورتی که φ و ψ دو فرمول باشند، آنگاه φ∨ψ، در صورتی درست می‌باشد که دست‌کم یکی از فرمول‌های φ یا ψ درست باشند.

در زمانی که سورها وجود دارند، درستی(صحّت) یک فرمول باتوجّه به دامنه‌ی سخن تشخیص داده می‌شود؛

تعریف- *یک دامنه، که معمولاً با حرف D نمایش داده می‌شود، فقط یک مجموعه می‌باشد؛*

یک مجموعه می‌تواند مثل: اعداد طبیعی {...،۱،۲،۳}، مجموعه‌ای از افراد و ... باشد.

مثال- به فرمول ∀x: φ و یک دامنه‌ی D توجّه نمایید، فرمول ∀x: φ، در دامنه‌ی D درست می‌باشد، اگر و فقط اگر φ(x|d) برای هر d∈D درست باشد، که φ(x|d)، فرمولی گرفته شده از φ، بعد از جایگزین نمودن هر وقوع x، در φ با d می‌باشد. به عنوان مثال، به فرمول ∀x: has_chair[۴](x) و یک دامنه‌ی D، تشکیل شده از دانشجویانی که در حال حاضر در

---

۱ - در لغت به معنی «برادر یا خواهر، هم‌نژاد» است؛ در اینجا به معنی «برادرِ» می‌باشد.

۲ - جانْ: در زبان انگلیسی نامی برای یک مرد است. (Babylon> Babylon English)

۳ - ریچارد: در زبان انگلیسی نامی برای یک مرد است. (Babylon> Babylon English)

۴ - «chair»، در لغت به معنی «صندلی» می‌باشد و «has_ chair» در اینجا به معنی «دارای صندلی است»، می‌باشد.

کلاس OMB 31[۱] هستند، توجّه نمایید؛ فرمول ∀x: has_chair(x) در دامنه‌ی D درست است، اگر و فقط اگر ∀x: has_chair(John) درست باشد، ∀x: has_chair(Mary[۲]) درست باشد و همین طور این مورد برای هر دانشجوی دیگری درون کلاس OMB 31 هم درست باشد.

مثال- به فرمول ∃x: φ و دامنه‌ی D توجّه نمایید؛ فرمول ∃x: φ، در دامنه‌ی D درست است، اگر وفقط اگر φ(x|d)، برای برخی از مقدارهای d∈D درست باشد، که φ(x|d)، فرمولی گرفته شده از φ، بعد از جایگزین نمودن هر وقوع x، در φ، با d می‌باشد؛ به عنوان مثال، به فرمول ∃x: standing[۳](x) و یک دامنه‌ی D، مرکّب از دانشجویانی که در حال حاضر در کلاس OMB 31 هستند، توجّه نمایید؛ فرمول ∃x: standing(x) در دامنه‌ی D درست است، اگر و فقط اگر دست‌کم یک شخص، در OMB 31 ایستاده باشد و بنابراین، standing(Lecturer[۴]) درست است و داریم: ∃x: standing(x) درست می‌باشد.

نکته:

توجّه کنید که یک فرمول ممکن است در برخی از دامنه‌ها درست باشد، امّا در برخی دیگر نادرست باشد؛ به عنوان مثال، به فرمول ∀x∃y: $(x=y^2)$ توجّه نمایید؛ این فرمول در دامنه‌ی اعداد حقیقی مثبت درست می‌باشد؛ ولی در دامنه‌ی همه‌ی اعداد حقیقی، این عبارت غلط می‌باشد؛ اگر x، منفی باشد، x نمی‌تواند با هر y^2 ای برابر باشد، چون $y^2 \geq 0$ می‌باشد.

تعریف- یک فرمول که در همه‌ی دامنه‌ها درست است، یک *همانگویی(همیشه درست)*[۵] می‌باشد.

تعریف- یک فرمول که در همه‌ی دامنه‌ها نادرست(غلط) است، یک *خلاف گویی(تناقض)*[۶] است.

مثال- بیان نسبت خانوادگی

منطق مرتبه‌ی اوّل یک زبان بیان قدرتمند برای دانش می‌باشد؛ به عنوان مثال، به بیان نسبت خانوادگی توجّه نمایید، گزاره‌ها ممکن است شامل موردهای زیر باشند:

Parent[۷](_,_)

Child[۸](_ , _)

---

۱ - «OMB 31»، در اینجا نام یک کلاس درس است.

۲ - مری، در زبان انگلیسی نامی برای یک زن است. (Babylon> Babylon English)

۳ - در لغت به معنی «ایستاده، سَرپا» می‌باشد.

۴ - در لغت یعنی: «مُدرِّس، سخنران»

۵ - tautology

۶ - contradiction

۷ - در لغت یعنی: «پدر یا مادر»؛ در اینجا یعنی: «پدرِ یا مادرِ»

۸ - در لغت یعنی: «بچّه، فرزند»؛ در اینجا یعنی: «بچّه‌یِ، فرزندِ»

Grandparent[۱](_ , _)

Sibling[۲](_ , _)

Male[۳](_)

Female[۴](_)

- = (_ , _)

راهی کلّی برای تبدیل عبارت‌های انگلیسی به منطق مرتبه‌ی اوّل وجود ندارد؛ در زیر جملاتی بیان شده و معادل آنها در منطق مرتبه‌ی اوّل هم آورده شده است:

«James پدر Harry است.»

Parent(James, Harry)∧Male(James)

«Lily مادر Harry است.»

Parent(Lily, Harry)∧Female(Lily)

«مردان[۵] و زنان[۶] کاملاً متمایز هستند.»

∀x: (Male(x) ↔ ¬Female(x))

p↔q به این صورت هم خوانده می‌شود: «p معادل با q است.»، و چنانچه در فصل «منطق گزاره‌ای» همین کتاب هم یادآوری شد، کوتاه‌نویسی‌ای برای ((p→q)∧(q→p)) می‌باشد.

---

۱ – در لغت یعنی: «پدر بزرگ یا مادر بزرگ»؛ در اینجا یعنی: «پدر بزرگِ یا مادر بزرگِ»

۲ – در لغت یعنی: «برادر یا خواهر، هم‌نژاد»؛ در اینجا یعنی: «برادرِ یا خواهرِ، هم‌نژادِ»

۳ – در لغت یعنی: «مرد»

۴– در لغت یعنی: «زن»

۵ – males

۶ – females

سور(کمّیّت‌سنج)ها در منطق مرتبه‌ی اوّل

∀(هر)

به فرمول $\forall x: P(x)$ توجّه نمایید، برای یک *دامنه‌ی محدود* $D=\{a_1, \dots, a_n\}$، این عبارت دقیقاً در زمانی درست است که $P(a_i)$ برای هر i، که $1 \le i \le n$، درست باشد؛ یعنی: $\forall x: P(x) \equiv^{1} P(a_1) \wedge \dots \wedge P(a_n)$. برای یک *دامنه‌ی نامحدود* $D=\{a_1, \dots, a_n, \dots\}$، این عبارت دقیقاً در زمانی درست است که $P(a_i)$، برای هر i درست باشد؛ یعنی: $\forall x: P(x) \equiv P(a_1) \wedge \dots \wedge P(a_n) \wedge \dots$؛ امّا یک فرمول نمی‌تواند دارای طول بینهایت باشد.

∃(وجود دارد)

به فرمول $\exists x: P(x)$ توجّه نمایید، برای یک دامنه‌ی *محدود* $D=\{a_1, \dots, a_n\}$، این عبارت دقیقاً در زمانی درست است که $P(a_i)$، برای دست‌کم یک مقدار i، که $1 \le i \le n$، درست باشد؛ یعنی: $\exists x: P(x) \equiv P(a_1) \vee \dots \vee P(a_n)$. برای یک دامنه‌ی *نامحدود* $D=\{a_1, \dots, a_n, \dots\}$، این عبارت دقیقاً در زمانی درست است که $P(a_i)$، برای دست‌کم یک مقدار i درست باشد؛ یعنی: $\exists x: P(x) \equiv P(a_1) \vee \dots \vee P(a_n) \vee \dots$؛ امّا یک فرمول، چنانچه گفتیم، نمی‌تواند دارای طول بینهایت باشد.

دو اشتباه معمول در مورد سورهای عمومی و وجودی که باید از آنها خودداری کنیم!

مطلب مهم:

سورهای عمومی اغلب با استنباط(⇒) به کار می‌روند؛

مثلاً اگر بخواهیم بگوئیم «همه‌ی دانشجویان، باهوش هستند»، می‌نویسیم:

$(\forall x)\ student(x) \Rightarrow smart(x)$

و به ندرت با عطف(∧) به کار می‌روند؛ زیرا این گونه عبارات، به ندرت درست(True) هستند؛ مثلاً عبارت زیر به این معنی است که «همه، در جهان، دانشجو و باهوش هستند»:

$(\forall x)\ student(x) \wedge smart(x)$

مطلب مهم:

سورهای وجودی معمولاً با عطف(∧) به کار می‌روند؛

مثلاً عبارت زیر به این معنی است که «دانشجویی وجود دارد که باهوش است»:

$(\exists x)\ student(x) \wedge smart(x)$

---

۱ - «≡»، در اینجا به معنی «برابر است با» یا «معادل است با» می‌باشد.

یک اشتباه رایج، این است که این جمله را با استنباط(⇒)، به صورت زیر بیان کنیم:

$(\exists x)\ student(x) \Rightarrow smart(x)$

این عبارت حتّا اگر هیچ دانشجویی در دامنه وجود نداشته باشد هم درست است، زیرا $student(x) \Rightarrow smart(x)$، برای هر فردی که دانشجو نباشد، درست است؛ *جدول صحّت استنباط(⇒) را به یاد بیاورید؛ در این جدول، اگر p نادرست بود، آنگاه $p \Rightarrow q$ همیشه درست بود!.*

بیان یکتایی(یگانگی، منحصر به فرد بودن، $\exists!$)[1]

گاهی اوقات می‌خواهیم بگوئیم که یک شیء تک و منحصر به فرد وجود دارد که شرط معیّنی را برآورده می‌کند.

مثال- «x منحصر به فردی وجود دارد که king(x) درست(true) است.»؛ این جمله را می‌توان به دو صورت زیر بیان کرد:

$\exists x\ king(x) \wedge \forall y\ (king(y) \rightarrow x=y)$

$\exists x\ king(x) \wedge \neg\exists y\ (king(y) \wedge x\neq y)$

این جمله را می‌توان به صورت زیر هم بیان کرد:

$\exists!\ x\ king(x)$

ویژگی‌های کمّیّت‌سنج‌ها(سورها)

مطلب مهم:

$\forall y\ \forall x$، مثل $\forall x\ \forall y$ می‌باشد. $\exists y\ \exists x$، مثل $\exists x\ \exists y$ می‌باشد؛ امّا $\forall y\ \exists x$، مثل $\exists x\ \forall y$ نمی‌باشد.

مثال:

$\forall x\ \exists y\ likes(x,y)$؛ یعنی، «هر فرد، فردی را دوست دارد.»

$\exists y\ \forall x\ likes(x,y)$؛ یعنی، «فردی وجود دارد که همه او را دوست می‌دارند.»

۱- expressing uniqueness

نقیض و سورها

مطلب مهم:

هر کدام از سورها می‌تواند با استفاده از دیگری بیان شود داریم:

$\neg\forall x: P(x) \equiv \exists x: \neg P(x)$

$\forall x: \neg P(x) \equiv \neg\exists x: P(x)$

$\neg\forall x: \neg P(x) \equiv \exists x: P(x)$

$\forall x: P(x) \equiv \neg\exists x: \neg P(x)$

مثالی برای نقیض و سورها:

$\forall x\ Likes(x,IceCream^{1}) \equiv \neg\exists x\ \neg Likes(x,IceCream)$

$\exists x\ Likes(x,Broccoli^{2}) \equiv \neg\forall x\ \neg Likes(x, Broccoli)$

---

۱ – لغت «ice cream» در زبان انگلیسی به معنی «بستنی» است.

۲ – در لغت نام نوعی گُلِ کَلَم است:

چکیده‌ی مطلب‌های فصل یازدهم

منطق گزاره‌ای، زبانی ضعیف است و برخلاف زبان طبیعی، قدرت بیان اندکی دارد.

منطق مرتبه‌ی اوّل، یک زبان بیان قدرتمند برای دانش می‌باشد.

منطق مرتبه‌ی اوّل، عیب‌های منطق گزاره‌ای را ندارد و تمام ویژگی‌های منطق گزاره‌ای را داراست.

منطق مرتبه‌ی اوّل، جهان را براساس اشیا، ویژگی‌ها، ارتباط‌ها و عملکردها مدلسازی می‌کند.

در منطق مرتبه‌ی اوّل، معنی $\vee$، $\wedge$، $\neg$ و $\rightarrow$ مثل معنی آنها در منطق گزاره‌ای می‌باشد.

در صورتی که P یک سمبل گزاره‌ای باشد و t_n , ... , t_1، واژگان باشند، آنگاه $P(t_1, \ldots , t_n)$، یک فرمول منطق مرتبه‌ی اوّل می‌باشد.

در زمانی که سورها وجود دارند، درستی(صحّت) یک فرمول باتوجّه به دامنه‌ی سخن تشخیص داده می‌شود.

یک فرمول ممکن است در برخی از دامنه‌ها درست باشد، امّا در برخی دیگر نادرست باشد.

راهی کلّی برای تبدیل عبارت‌های انگلیسی به منطق مرتبه‌ی اوّل وجود ندارد.

سورهای عمومی اغلب با استنباط($\Rightarrow$) به کار می‌روند.

سورهای وجودی معمولاً با عطف($\wedge$) به کار می‌روند.

$\forall x \ \forall y$، مثل $\forall y \ \forall x$ می‌باشد. $\exists x \ \exists y$، مثل $\exists y \ \exists x$ می‌باشد؛ امّا $\exists x \ \forall y$، مثل $\forall y \ \exists x$ نمی‌باشد.

هر کدام از سورها می‌تواند با استفاده از دیگری بیان شود.

یادآوری یا تکمیل مطلب‌های فصل یازدهم

تعریف- «واژه(ترم، لغت)» را تعریف کنید.

جواب- یک ساختار نحوی(گرامری) است که یک شیء را مشخّص می‌نماید و می‌تواند یک سمبل ثابت، یک متغیّر و یا یک سمبل تابعی به کار گرفته شده برای ترم‌ها باشد.[۱]

درست یا غلط- عبارت $\forall x\ x=x$، یک عبارت برآورده کننده است.

جواب- **«درست» است؛** همچنین دارای صحّت(اعتبار، همانگویی) هم می‌باشد.[۲]

درست یا غلط- عبارت $\exists x\ \forall y\ x=y$، برآورده کننده است.

جواب- **«غلط» است؛** مگر اینکه دامنه فقط از یک شیء تشکیل شده باشد.[۳]

سؤال- دارای صحّت(همانگویی) بودن هر یک از جملات زیر را مشخّص نمائید.

الف) $(\exists x\ x = x) \Rightarrow (\forall y \exists z\ y = z)$

ب) $\forall x\ P(x) \vee \neg P(x)$

---

۱ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، پاییز سال ۱۹۹۱ میلادی

۲ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، پاییز سال ۱۹۹۹ میلادی

۳ - آزمون میان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۱۹۹۶ میلادی

جواب- هر دو عبارت دارای صحّت هستند.[۱]

درست یا غلط- عبارت «سهراب و سَهَند با هم برادرند» برابر است با عبارت «سهراب برادر است و سهند برادر است»

جواب- «غلط» است؛ زیرا عبارت دوّم بیان نمی‌کند که آنها با هم برادر هستند.[۲]

درست یا غلط- «نه(neither) p و نه(nor) q» برابر است با «هر دوی p و q غلط هستند»

جواب- «درست» است؛ زیرا neither و nor، در زبان انگلیسی، دارای معنی «$\neg p \wedge \neg q$» است.[۳]

سؤال- جمله‌های زیر را به صورت منطق مرتبه‌ی اوّل بنویسید.

الف) تمام انواع موجود پرندگان می‌توانند پرواز کنند

جواب-

$\forall x \; Bird(x) \rightarrow Fly(x)$[۴]

ب) برخی از انواع موجود پرندگان می‌توانند پرواز کنند

جواب-

$\exists x \; Bird(x) \wedge Fly(x)$[۵]

درست یا غلط- جمله‌ی «همه‌ی گربه‌ها پستاندار هستند.»، در منطق مرتبه‌ی اوّل به صورت زیر می‌باشد:

$\forall x \; Cat(x) \wedge Mammal(x)$

جواب- «غلط» است؛ $\forall x \; Cat(x) \Rightarrow Mammal(x)$[۶]

سؤال- مشابه کنکور آزاد مهندسی کامپیوتر سال ۹۲- جمله‌ی «Bill دست کم دو خواهر دارد.» را به منطق مرتبه‌ی اوّل بنویسید.

جواب-

---

۱ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «چاک دایر»، دانشگاه ویسکانسین-مَدیسُون کشور آمریکا، ۲۲ دسامبر سال ۲۰۰۹ میلادی

۲ - آزمون پایان‌ترم درس «هوش مصنوعی» دانشکده‌ی علوم کامپیوتر دانشگاه نیویورک کشور آمریکا، بهار ۲۰۰۷ میلادی

۳ - آزمون پایان‌ترم درس «هوش مصنوعی» دانشکده‌ی علوم کامپیوتر دانشگاه نیویورک کشور آمریکا، بهار ۲۰۰۷ میلادی

۴ - آزمون میان‌ترم درس «هوش مصنوعی» دانشگاه McGill کشور کانادا، زمستان سال ۲۰۱۰ میلادی

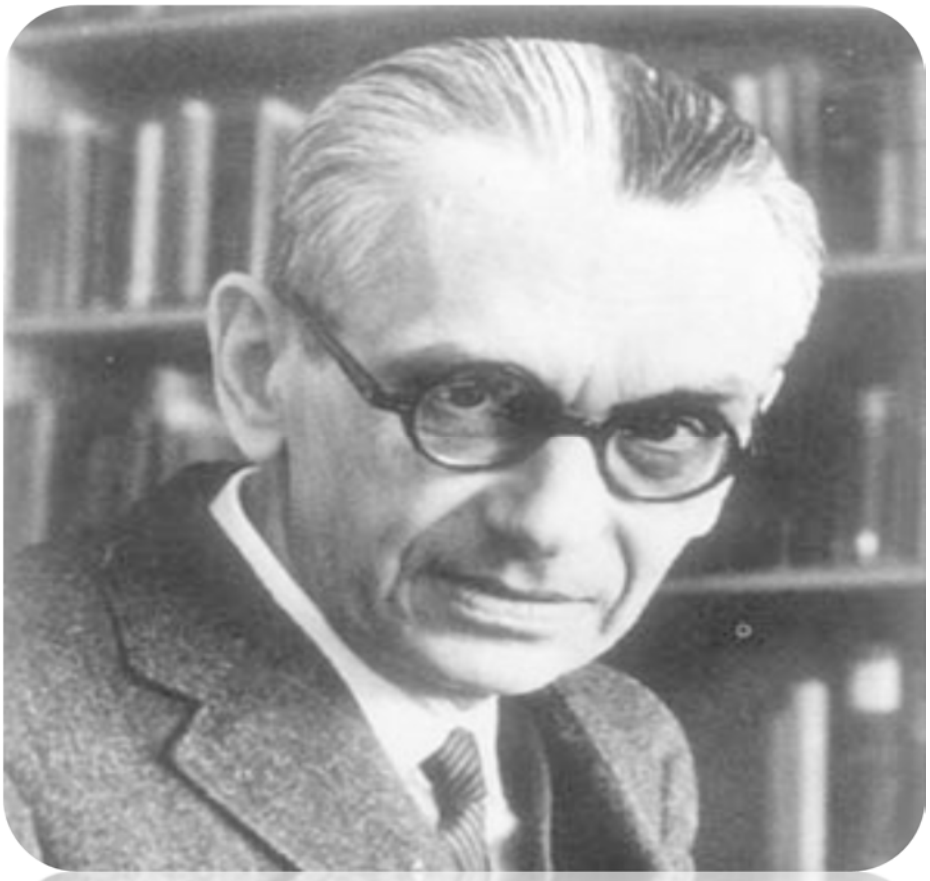
۵ - آزمون میان‌ترم درس «هوش مصنوعی» دانشگاه McGill کشور کانادا، زمستان سال ۲۰۱۰ میلادی

۶ - کوییز شماره‌ی ۳ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

$\exists x, y\ (SisterOf(x, Bill) \wedge SisterOf(y, Bill) \wedge \neg(x = y))$[۱]

۱ - مطلب‌های موجود در پایگاه اینترنتی دانشگاه شهر بابل(Babylon) کشور عراق

فصل دوازدهم

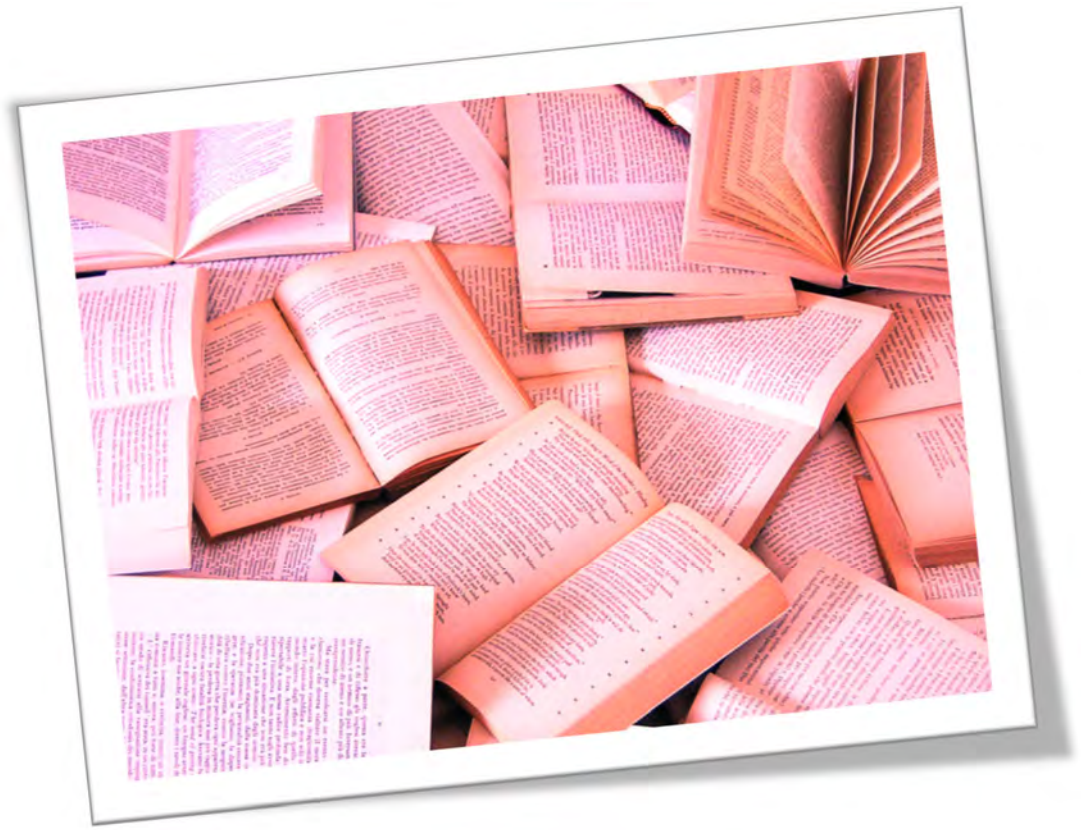


۱ استنتاج در منطق مرتبه‌ی اوّل[۲]

---

۱ - تصویر، کِرْتْ گادُلْ(Kurt Godel)، ۱۹۷۸ – ۱۹۰۶ میلادی، ریّاضیدان، فیلسوف و منطقدان آمریکایی متولّد کشور اُتریش است (http://en.wikipedia.org/wiki/Kurt_G%C3%B6del)؛ او گفت: «الگوریتم کاملی برای منطق مرتبه‌ی اوّل وجود دارد.»

۲ - Inference in First-order Logic



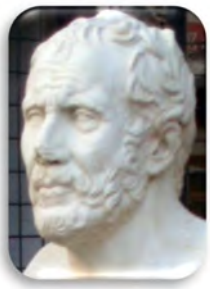
فهرست برخی از عنوان‌های نوشته‌ها

زنجیره‌ی پیشرو(مستقیم)

زنجیره‌ی پسرو(معکوس)

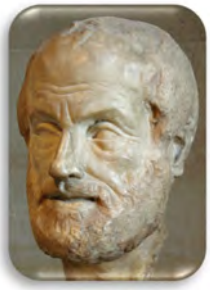
تحلیل

تاریخچه‌ی کوتاهی از استدلال



زِنُو[1]

در سال ۴۵۰ قبل از میلاد حضرت مسیح (درود بر او باد)، پیروان فلسفه‌ی رَواقیّون[2] منطق گزاره‌ای و (شاید) استدلال را بنیان نهادند.



اَرَسطو

در سال ۳۲۲ قبل از میلاد حضرت مسیح (درود بر او باد)، اَرَسطو قیاس منطقی[3] (قوانین استدلال) و سورها را بیان کرد.

---

۱ - Zeno Of Citium، ۲۶۳ – ۳۳۵ سال قبل از میلاد حضرت مسیح (درود بر او باد)، فیلسوفی یونانی بوده است. (Babylon> Britannica Concise Encyclopedia)

۲ - Stoics؛ رَواقی به کسی می‌گفتند که عضوی از مدرسه‌ی فلسفه‌ای که به وسیله‌ی زِینُو(Zeno) به وجود آمد، بود و دارای این عقیده بودند که انسان عاقل نباید تحت تأثیر هوای نَفْس، غم یا شادی قرار بگیرد و باید مطیع قانون‌های طبیعی باشد. (Babylon> Merriam-Webster Collegiate® Dictionary)

۳ - syllogisms؛ **یادآوری**- نوعی از استدلال است که دارای مقدّم و تالی می‌باشد؛ به عنوان مثال، اگر A=B و B=C باشد، آنگاه A=C خواهد بود. (Babylon> Britannica Concise Encyclopedia)

جِرولامُو کاردانُو

در سال ۱۵۶۵ میلادی، کاردانو[۱] تئوری احتمال[۲](منطق گزاره‌ای[۳] + نامعلومی[۴]) را به وجود آورد.

جُرج بول

در سال ۱۸۴۷ میلادی، بول[۵] منطق گزاره‌ای را برّرسی کرد.

جوانی گاٹلابْ فیریجْ

در سال ۱۸۷۹ میلادی، فیریجْ[۶] منطق مرتبه‌ی اوّل را به وجود آورد.

---

۱ - Gerolamo Cardano، ۱۵۷۶ – ۱۵۰۱ میلادی، ریّاضیدان، پزشک، ستاره‌شناس(منجّم) و قُمارْ بازی ایتالیایی بود.
(http://en.wikipedia.org/wiki/Gerolamo_Cardano)

۲ - probability theory

۳ - propositional logic

۴ - uncertainty

۵ - George Boole، ۱۸۶۴ – ۱۸۱۵ میلادی، ریّاضیدان و فیلسوفی انگلیسی بود.
(http://en.wikipedia.org/wiki/George_Bool)

۶ - Gottlob Frege

لودویگ ویتگِنستاین

در سال ۱۹۲۲ میلادی، لودْویگْ ویتْگِنْستایْنْ[۱] اثبات به وسیله‌ی جدول صحّت را برّرسی کرد.

کِرت گادُل

در سال ۱۹۳۰ میلادی، کِرْتْ گادُلْ[۲] گفت: «الگوریتم کاملی برای منطق مرتبه‌ی اوّل وجود دارد.»

ژاک هِربرَند

در سال ۱۹۳۰ میلادی، ژاکْ هِرْبرْنْدْ[۳] الگوریتم کاملی برای منطق مرتبه‌ی اوّل به وجود آورد.

در سال ۱۹۳۱ میلادی، کِرت گادُل گفت: الگوریتم کاملی برای محاسبه وجود ندارد.

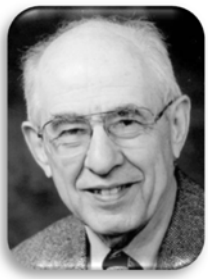
---

۱ - Ludwig Wittgenstein، ۱۹۵۱ – ۱۸۸۹ میلادی، فیلسوفی انگلیسی و متولّد کشور اتریش بود. تعداد زیادی باور دارند که او بزرگ‌ترین فیلسوف قرن بیستم است. (Babylon> Britannica Concise Encyclopedia)

۲ - Kurt Godel

۳ - Jacques Herbrand: ۱۹۳۱ – ۱۹۰۸ میلادی، ریّاضیدانی فرانسوی بود.
(http://en.wikipedia.org/wiki/Jacques_Herbrand)


مارتین دِیویس


هیلاری پاتنام

در سال ۱۹۶۰ میلادی، مارْتینْ دِیْویسْ[۱] و هیلاری پاتْنامْ[۲] الگوریتمی کاربردی برای منطق گزاره‌ای ارائه کردند.


جان آلِن رابینسُون

در سال ۱۹۶۵ میلادی، رابینْسُونْ[۳] الگوریتمی کاربردی را برای تحلیل[۴] در منطق مرتبه‌ی اوّل به وجود آورد.

---

۱ - Martin Davis: متولّد ۱۹۲۸ میلادی و ریّاضیدانی آمریکایی است. (http://en.wikipedia.org/wiki/Martin_Davis)

۲ - Hilary Putnam: متولّد ۱۹۲۶ میلادی و فیلسوفی آمریکایی است.
(http://en.wikipedia.org/wiki/Hilary_Putnam)

۳ - John Alan Robinson: متولّد ۱۹۳۰ میلادی، فیلسوف، ریّاضیدان و دانشمند علوم کامپیوتری از کشور آمریکا می‌باشد.
(http://en.wikipedia.org/wiki/John_Alan_Robinson)

۴ - Resolution، **توضیح**- روشی برای اثبات عبارت‌های منطق مرتبه‌ی اوّل است؛ این روش به وسیله‌ی جان آلِن رابینسون(J. A. Robinson) در سال ۱۹۶۵ میلادی معرّفی شد. (Babylon> FOLDOC) در فصل «منطق گزاره‌ای» همین کتاب هم این روش را داشتیم؛ در آینده در همین فصل با این روش بیش‌تر آشنا خواهیم شد.

استنتاج در منطق مرتبه‌ی اوّل

می‌توانیم انواع استنتاج را با منطق مرتبه‌ی اوّل داشته باشیم.

برداشتن سورها

نمونه‌سازی(معرّفی) عمومی[۱]

متغیّر می‌تواند با هر واژه‌ی زمینه[۲](ی ممکن) جایگزین شود؛ مثلاً در مثال زیر؛

$\forall x\ \text{LivesIn}^{۳}(x,\text{Springfield}^{۴}) \Rightarrow \text{knows}^{۵}(x,\text{Jack}^{۶})$

از آنجایی که این عبارت برای همه‌ی xها(ی ممکن) صحیح می‌باشد، می‌توانیم $\{x=\text{Bart}^{۷}\}$ را جایگزین نماییم:

$\text{LivesIn}(\text{Bart},\text{Springfield}) \Rightarrow \text{knows}(\text{Bart},\text{Jack})$

پس:

$$\frac{\forall \upsilon : \alpha}{Subst\left(\{\upsilon / g\}, \alpha\right)}$$

برای هر متغیّر v و واژه‌ی زمینه‌ی g برقرار می‌باشد. در بالا، Subst، مخفّف substitution، به معنی «جایگزینی، جانشینی» می‌باشد.

مثال، نتیجه‌های $\forall x: \text{King}^{۸}(x) \wedge \text{Greedy}^{۹}(x) \Rightarrow \text{Evil}^{۱۰}(x)$ عبارتند از:

---

۱ - Universal instantiation(*UI*)

۲ - ground term؛ **یادآوری**- همان طور که در گذشته هم گفتیم، واژه‌ای که دارای هیچ متغیّری نباشد، یک واژه‌ی زمینه است.

۳ - «lives in»، یعنی: «زندگی می‌کند در»

۴ - نام شهری در کشور ایالات متّحده‌ی آمریکا و مرکز ایالت ایلینُیْز(Illinois) است. (Babylon> Babylon English)

۵ - در اینجا یعنی: «می‌شناسد»

۶ - جَک، در زبان انگلیسی نامی برای یک مرد است. (Babylon> Babylon English)

۷ - بارْتْ، در زبان انگلیسی نامی برای یک مرد است. (Babylon> Babylon English)

۸ - در لغت یعنی: «شاه»

۹ - در لغت یعنی: «آزمند، حریص، طمّاع»

۱۰ - در لغت یعنی: «شرور، بد»

King(John) ∧ Greedy(John) ⇒ Evil(John),
King(Richard) ∧ Greedy(Richard) ⇒ Evil(Richard),
King(Father(John)) ∧ Greedy(Father(John)) ⇒ Evil(Father(John)), …

نمونه‌سازی(معرّفی) وجودی[۱]

در مورد $\exists x\ LivesIn(x,Springfield) \wedge knows(x,Jack)$ می‌دانیم که این باید برای دست‌کم یک شیء درست باشد. بیایید این شیء را k بنامیم؛ داریم، $LivesIn(k,Springfield) \wedge knows(k,Jack)$ که در این مورد، k یک *ثابت اِسکُلِم*[۲] نامیده می‌شود. پس برای هر عبارت α و متغیّر v و سمبل ثابت k، *که در جای دیگری در پایگاه دانش وجود ندارد*، داریم:

$$\frac{\exists \upsilon:\alpha}{\mathrm{Subst}(\{v/k\},\alpha)}$$

مثال- نتیجه‌ی $\exists x\ Crown^{3}(x) \wedge OnHead^{4}(x,John)$ به صورت $Crown(C_1) \wedge OnHead(C_1,John)$ می‌تواند باشد، که C_1، به وجود آمده از یک سمبل ثابت جدید، به نام ثابت اِسکُلِم می‌باشد.

مثالی دیگر: از $\exists x:\ d(x^y)/dy=x^y$، نتیجه می‌گیریم، $d(e^y)/dy=e^y$ (eای که به وجود آمده است، یک سمبل ثابت جدید(اسکلم) می‌باشد).

---

۱ - *E*xistential *I*nstantiation(*EI*)

۲ - Skolem constant؛ تُرالْفْ اِسکُلِم(Thoralf Skolem)، ۱۹۶۳ – ۱۸۸۷ میلادی، ریّاضیدانی نُروژی بود که شهرت او به خاطر کار بر روی منطق ریّاضی و نظریّه‌ی مجموعه‌ها(set theory) می‌باشد.
(http://en.wikipedia.org/wiki/Thoralf_Skolem)
تصویر، وی را نشان می‌دهد:

۳ - در لغت یعنی: «تاج»؛ تصویری از یک تاج:

۴ - on head، در لغت به معنی «روی سر» می‌باشد.

گزاره‌بندی(گزاره‌سازی)[۱]

می‌توانیم هر عبارت دارای سور وجودی را با یک نوع اِسکُلمایز شده جایگزین نماییم. برای عبارت‌های با سور عمومی می‌توانید هر جایگزین ممکن را جایگزین نمایید؛ *که این کار به ما اجازه می‌دهد که از قانون‌های استنتاج گزاره‌ای استفاده نماییم؛* البتّه این کار خیلی ناکارآمد می‌باشد؛ تا سال ۱۹۶۰ میلادی از همین روش استفاده می‌کردند.

تبدیل منطق مرتبه‌ی اوّل به استدلال گزاره‌ای

تصوّر نمایید که پایگاه دانش فقط شامل عبارات زیر باشد:

$\forall x: King(x) \wedge Greedy(x) \Rightarrow Evil(x)$,
$King(John)$,
$Greedy(John)$,
$Brother(Richard,John)$

با جایگزینی برای عبارت دارای سور عمومی در همه‌ی موردهای ممکن داریم:

$King(John) \wedge Greedy(John) \Rightarrow Evil(John)$,
$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$,
$King(John)$,
$Greedy(John)$,
$Brother(Richard,John)$

که پایگاه دانش جدید، گزاره‌بندی(گزاره‌سازی) شده[۲] می‌باشد، سمبل‌های گزاره‌ای عبارتند از:

$King(John), Greedy(John), Evil(John), King(Richard)$

و.....

– **ادّعا:** یک عبارت زمینه به وسیله‌ی پایگاه دانش جدید در صورتی به وجود می‌آید که به وسیله‌ی پایگاه دانش اصلی هم به وجود آید.

– **ادّعا:** هر پایگاه دانش منطق مرتبه‌ی اوّل برای حفظ استلزام می‌تواند به صورت گزاره‌بندی شده بیان شود.

---

۱ – propositionalization

۲ – propositionalized

- **ایده:** برای این کار، پایگاه دانش و پرس‌وجو را گزاره‌بندی نمایید، تحلیل را به کار ببرید و نتیجه را برگردانید.

مطلب مهم:

- **مسأله:** *گزاره‌بندی می‌تواند تعداد زیادی جمله‌های نامربوط را به وجود آورد؛* مثلاً با استفاده از توابع، واژگان زمینه به صورت نامحدود وجود خواهند داشت.

مثال- Father(Father(Father(John)))

قضیّه‌ی هِربرَند (۱۹۳۰ میلادی): در صورتی که عبارت α به وسیله‌ی یک پایگاه دانش منطق مرتبه‌ی اوّل به وجود بیاید، به وسیله‌ی یک زیرمجموعه‌ی محدود از پایگاه دانش گزاره‌بندی شده هم به وجود می‌آید.

ایده:

برای $n = 0$ تا بینهایت کارهای زیر را انجام بده

یک KB(پایگاه دانش) گزاره‌ای، با عمق n بسازید

برّرسی نمایید که آیا α به وسیله‌ی این KB تولید می‌شود؟

ایراد: اگر α به وجود آید، کار می‌کند؛ ولی اگر α به وجود نیاید، در حلقه می‌افتد.

ایرادهای گزاره‌بندی

همان طور که گفتیم، *گزاره‌بندی، تعداد زیادی عبارت‌های نامربوط را به وجود می‌آورد؛ در نتیجه استنتاج ممکن است خیلی ناکارآمد باشد.*

مثال- از

$\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$, $King(John)$, $\forall y\ Greedy(y)$, $Brother(Richard,John)$

حاصل به نظر می‌رسد که $Evil(John)$ باشد، امّا گزاره‌بندی، تعداد زیادی واقعیّت‌ها، نظیر $Greedy(Richard)$ که نامربوط هستند را به وجود می‌آورد. با سمبل‌های تابعی(تابع‌ها)، معرّفی‌های با مقادیر به مراتب بدتری وجود خواهد داشت!؛ مثل

Father(Father(Father(John))). بیایید ببینیم که آیا می‌توانیم استنتاج را به طور مستقیم، با استفاده از جمله‌های منطق مرتبه‌ی اوّل به وجود آوریم؟.

یکی‌سازی(یکسان‌سازی)[۱]

مطلب مهم:

تعریف- *یکی‌سازی این است که [برخلاف گزاره‌بندی،] فقط برای عباراتی که به ما کمک می‌کنند تا چیزهایی را ثابت نماییم، می‌خواهیم جایگزین‌هایی پیدا کنیم.*

مثال- اگر p = Knows(John,x) و q = Knows(John,Jane[۲]) باشد؛ با جایگزین کردن x با مقدار Jane، دو عبارت p و q با هم یکی می‌شوند.

مثال- اگر بدانیم:

$\forall x$: LivesIn(x,Springfield) $\wedge$ WorksAt[۳](x,PowerPlant[۴]) $\Rightarrow$ knows(x,Jack)

$\forall y$: LivesIn(y,Springfield)

WorksAt(Tom[۵],PowerPlant)

آنگاه ما باید قادر باشیم با جایگزینی

{x/Tom,y/Tom}

به طور مستقیم نتیجه بگیریم: knows(Tom,Jack).

در جدول زیر با جایگزینی θ در p و q باید این دو عبارت با هم یکی شوند:

---

۱ - unification

۲ - جِیْنْ، در زبان انگلیسی نامی برای یک زن است. (Babylon> Babylon English)

۳ - works at، یعنی: «کار می‌کند در»

۴ - power plant: نیروگاه برق، کارخانه‌ی برق

۵ - تامْ، در زبان انگلیسی نامی برای یک مرد است و شکلی از کلمه‌ی توماس(Thomas) می‌باشد. (Babylon> Babylon English)

θ	q	p
{x/Jane}	Knows(John,Jane)	Knows(John,x)
{x/Ernest,y/John}	Knows(y,Ernest[1])	Knows(John,x)
{y/John,x/Mother(John)}	Knows(y,Mother(y))	Knows(John,x)
چنین موردی وجود ندارد.	Knows(x,Ernest)	Knows(John,x)

مطلب مهم:

توضیح- در جدول بالا آخرین یکی‌سازی تنها به این دلیل موفّق نشد که x نمی‌تواند مقدارهای John و Ernest را به صورت همزمان داشته باشد؛ مشکل از اینجا به وجود آمده است که خواسته‌ایم در هر دو عبارت، از متغیّر یکسان x استفاده نماییم. *استاندارد کردن*، از *روی هم افتادن*[2] متغیّرها جلوگیری می‌کند، به عنوان مثال، در اینجا می‌توانیم به جای Knows(x,Ernest)، از Knows(z_{17},Ernest) استفاده نماییم و این مشکل را برطرف نماییم.

روش تعمیم قیاس استثنائی[3]

مطلب مهم:

ایده‌ی اساسی این است که فرض کنیم یک استنتاج، به شکل $P_1 \wedge P_2 \wedge \ldots \wedge P_i \Rightarrow Q$ را داریم و عبارات، به صورت $P_1', P_2', \ldots, P_i'$ می‌باشند، اگر مجموعه‌ای از جانشین‌ها، به صورت $P_1=P_1', P_2=P_2', \ldots, P_i=P_i'$ وجود داشته باشند، در این صورت می‌توانیم جانشین‌ها را به کار بگیریم و این روش (یعنی، روش تعمیم قیاس استثنائی) را برای به‌دست‌آوردن Q به کار بگیریم. پس پردازه‌ی استنتاج ما حالا پیداکردن جانشین‌هایی که به ما اجازه خواهند داد عبارات جدید را به وجود آوریم، می‌باشد.

تعریف- الگوریتم یگانی: دو عبارت را می‌پذیرد و مجموعه‌ای از جانشین‌هایی که عبارات را به صورت یکتا می‌سازند، برمی‌گرداند. مثلاً

WorksAt(x,PowerPlant) و WorksAt(Jack,PowerPlant) ، {x/Jack} را تولید می‌نماید.

---

۱ - اِرْنِسْتْ، در زبان انگلیسی نامی برای یک مرد است. (Babylon> Babylon English)

۲ - overlap

۳ - Generalized *M*odus *P*onens(*GMP*)؛ به این روش، «قانون انتزاع تعمیم یافته» هم گفته می‌شود.

WorksAt(x,PowerPlant) و WorksAt(Jack,y)، {x/Jack,y/PowerPlant} را تولید می‌نماید.

WorksAt(x,PowerPlant) و WorksAt(FatherOf[۱](Bart),y)

، {x/FatherOf(Bart),y/PowerPlant} را تولید می کند.

WorksAt(x,PowerPlant),WorksAt(Jack,x)، غلط می‌باشد– چون x نمی‌تواند هم دارای مقدار Jack باشد و هم دارای مقدار PowerPlant باشد.

آخرین عبارت، یک اشتباه است؛ فقط به خاطر اینکه خواسته‌ایم x را در هر دو عبارت استفاده نماییم. همان طور که دیدیم، *برای رفع این مشکل می‌توانیم یکی از xها را با یک متغّیر منحصر به فرد (مثل $x_{۲۱}$)، که یک عبارت است، جایگزین نماییم [و این مشکل را از بین ببریم]، که این کار، استانداردسازی نامیده می‌شود.*

مطلب مهم:

اگر بیش از یک جانشین وجود داشته باشد که بتوانند دو عبارت که به نظر می‌رسد یکسان هستند را بسازند، چطور؟؛ مثلاً:

Sibling(Bart,x) و Sibling(y,z) می‌تواند {Bart/y,x/z} یا {x/Bart,y/Bart,z/Bart} را تولید نماید، که اوّلین یکی‌سازی، کلّی‌تر از دوّمی است، چون الزام‌های کم‌تری را سبب می‌شود؛ پس *ما می‌خواهیم کلّی‌ترین یکی کننده[۲]ها را در زمانی که استنتاج را انجام می‌دهیم، پیدا نماییم.*

زنجیره‌ی پیشرُو(مستقیم)، زنجیره‌ی پَسرُو(معکوس) و تحلیل[۳]

مثال– پایگاه دانش– قانون می گوید: این یک جنایت است که یک آمریکایی به کشورهای مخالف[۴] اسلحه[۵] بفروشد؛ کشور نونو[۱] یک دشمن[۲] آمریکا می‌باشد و دارای[۳] تعدادی موشک[۴] می‌باشد و تمام این موشک‌ها به وسیله‌ی کُلُنِل وِست[۵] که یک آمریکایی است به این کشور فروخته[۶] شده است. ثابت کنید که کلنل وست، یک جنایت‌کار[۷] می‌باشد.

---

۱ – یعنی: «پدرِ»

۲ – *m*ost *g*eneral *u*nifier(*MGU*)

۳ – **توجّه**– برای یادگیری بهتر این سه روش، بهتر است آن‌ها را در فصل «منطق گزاره‌ای» همین کتاب هم مطالعه کنید!

۴ – hostile(adjective)

۵ – weapon

حل– در ابتدا این مثال را به صورت منطق مرتبه‌ی اوّل می‌نویسیم:

... برای یک آمریکایی این جنایت است که به کشورهای دشمن اسلحه بفروشد:

$$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

کشور نونو دارای تعدادی موشک می‌باشد.... توجّه کنید:

$$\exists x\ Owns(Nono,x) \wedge Missile(x)$$

$$Owns(Nono,M_1),\ Missile(M_1)$$

... تمام این موشک‌ها به وسیله‌ی کلنل وست فروخته شده‌اند:

$$\forall x\ Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$$

موشک‌ها اسلحه هستند:

$$Missile(x) \Rightarrow Weapon(x)$$

«دشمن آمریکا»، «مخالف» است:

$$Enemy(x,America) \Rightarrow Hostile(x)$$

وست آمریکایی است:

$$American(West)$$

کشور نونو یک دشمن آمریکا است:

$$Enemy(Nono,America)$$

---

۱ – Nono؛ در اینجا نامی برای یک کشور فرضی است.
۲ – enemy(noun)
۳ – own
۴ – missile
۵ – Colonel West
۶ – sell
۷ – criminal

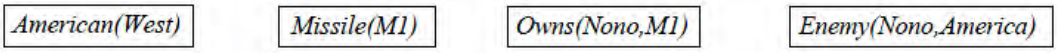
زنجیره‌ی پیشرُو(مستقیم)

مطلب مهم:

ایده- در این روش، اثبات‌ها با واقعیّت‌ها یا مقدّم‌ها شروع می‌شوند و جمله‌های جدید، با استفاده از روش تعمیم قیاس استثنائی به دست می‌آیند، تا زمانی که جمله‌ی هدف یا پرسش(پرس و جو)[1] به وجود آید.

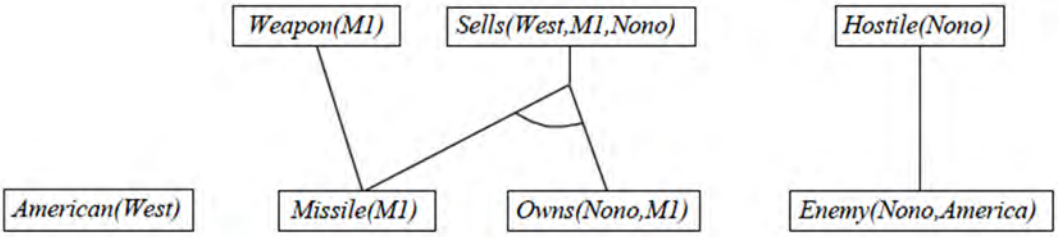
مثال- اثبات جنایت‌کار بودن کلنل وست به روش زنجیره‌ی پیشرو

در ابتدا-



$$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$$
$$\forall x \;\; Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$$
$$Owns(Nono,M_1) \qquad Missile(M_1)$$
$$Missile(x) \Rightarrow Weapon(x) \qquad Enemy(x,America) \Rightarrow Hostile(x)$$
$$American(West) \qquad Enemy(Nono,America)$$

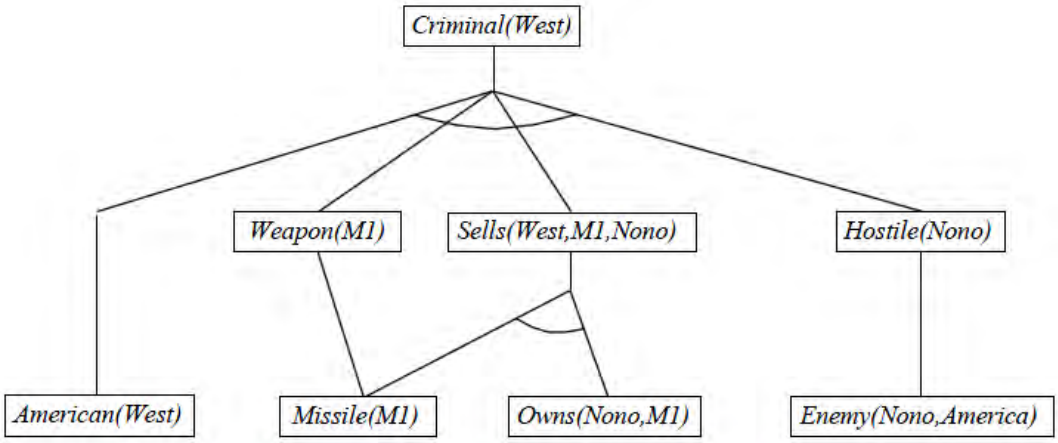
گام نخست-



$$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$$
$$\forall x \;\; Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$$
$$Owns(Nono,M_1) \qquad Missile(M_1)$$
$$Missile(x) \Rightarrow Weapon(x) \qquad Enemy(x,America) \Rightarrow Hostile(x)$$
$$American(West) \qquad Enemy(Nono,America)$$

۱- query

در انتـها-



$$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$$
$$\forall x \;\; Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$$
$$Owns(Nono,M_1) \qquad Missile(M_1)$$
$$Missile(x) \Rightarrow Weapon(x) \qquad Enemy(x,America) \Rightarrow Hostile(x)$$
$$American(West) \qquad Enemy(Nono,America)$$

کاربردی از زنجیره‌ی پیشرو

زنجیره‌ی پیشرو به طور گسترده‌ای در *پایگاه داده‌های قیاسی(استقرائی یا استنتاجی)*[۱] استفاده می‌شود. یک سیستم پایگاه داده‌ی قیاسی، سیستمی پایگاه داده‌ای است که می‌تواند نتیجه‌گیری را براساس قوانین و واقعیّت‌های نگهداری شده در پایگاه دانش قیاسی انجام دهد. سیستم‌های پایگاه دانش قیاسی اساساً به قوانین و واقعیّت‌ها می‌پردازند، در آنها از یک زبان اعلانی (نظیر پرولوگ) برای مشخّص کردن این قوانین و واقعیّت‌ها استفاده می‌شود و از یک موتور استنتاج که می‌تواند قوانین و واقعیّت‌های جدید را از آنهایی که ارائه شده‌اند، نتیجه‌گیری نماید، استفاده می‌شود.[۲]

۱- deductive databases

۲- http://en.wikipedia.org/wiki/Deductive_database

زنجیره‌ی پَسرُو(معکوس)

مطلب مهم:

این روش، به طور معکوس، از هدف به طرف واقعیّت‌هایی که باید برای هدف نشان داده شوند، حرکت می کند؛

ایده-*اثبات‌ها با پرسش هدف شروع می‌شوند و قانون‌هایی که دارای آن تالی هستند، پیدا می‌شوند و سپس هر یک از مقدّم‌های در استنباط، اثبات می‌شوند و این روال تا رسیدن به مقدّم‌ها ادامه می‌یابد.*

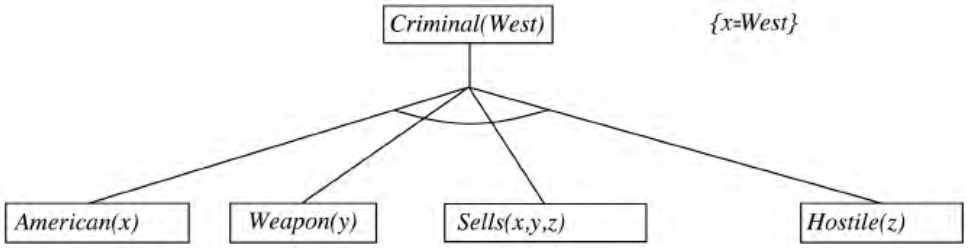
مثال- اثبات جنایت‌کار بودن کلنل وست با استفاده از روش زنجیره‌ی پسرو

در ابتدا-

$Criminal(West)$

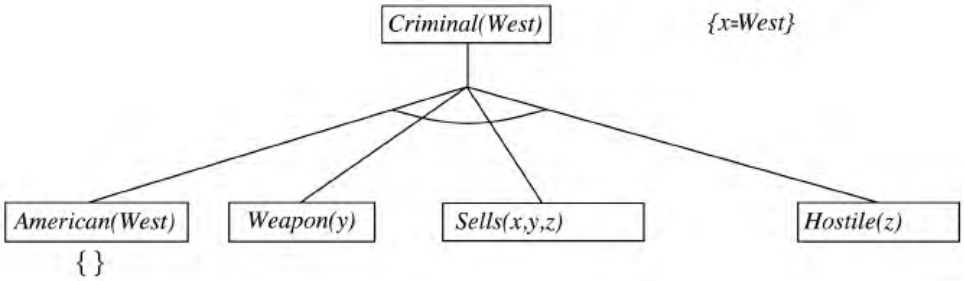
$$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$$
$$\forall x \ \ Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$$
$$Owns(Nono,M_1) \qquad Missile(M_1)$$
$$Missile(x) \Rightarrow Weapon(x) \qquad Enemy(x,America) \Rightarrow Hostile(x)$$
$$American(West) \qquad Enemy(Nono,America)$$

گام بعدی-



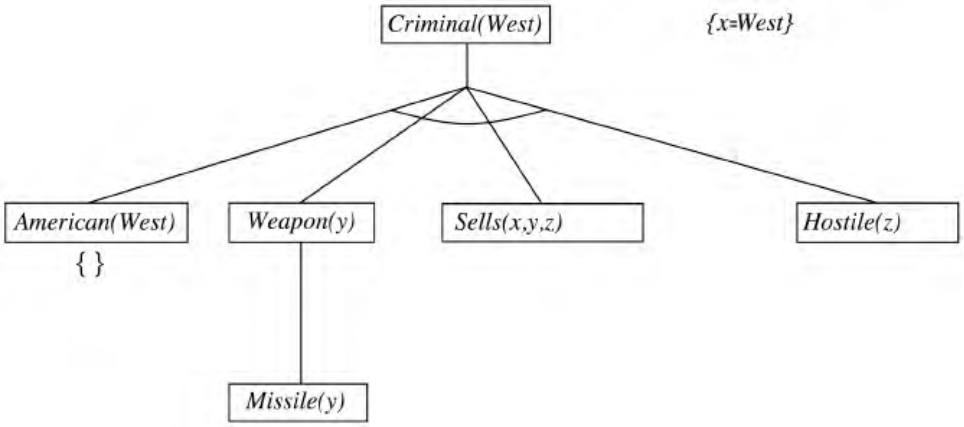
$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
$\forall x \;\; Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$
$Missile(x) \Rightarrow Weapon(x)$ $\quad$ $Enemy(x, America) \Rightarrow Hostile(x)$
$American(West)$ $\quad$ $Enemy(Nono, America)$

گام بعدی-



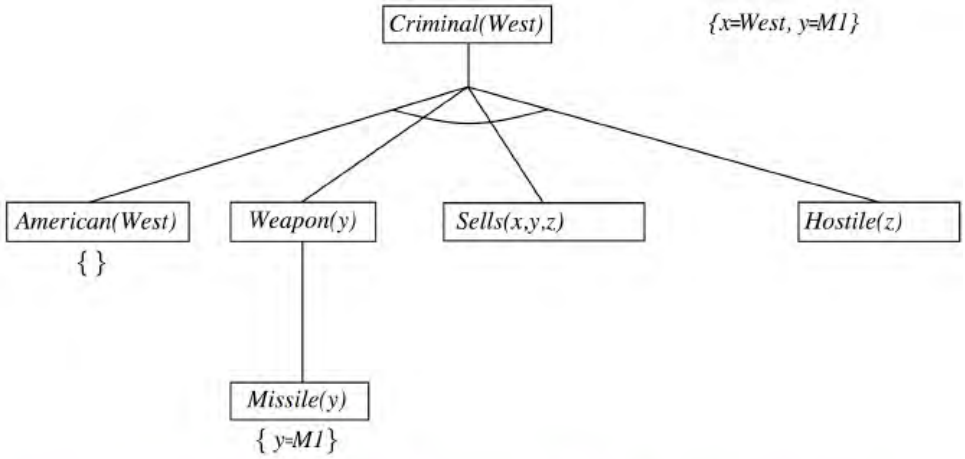
$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
$\forall x \;\; Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$
$Missile(x) \Rightarrow Weapon(x)$ $\quad$ $Enemy(x, America) \Rightarrow Hostile(x)$
$American(West)$ $\quad$ $Enemy(Nono, America)$

گام بعدی-



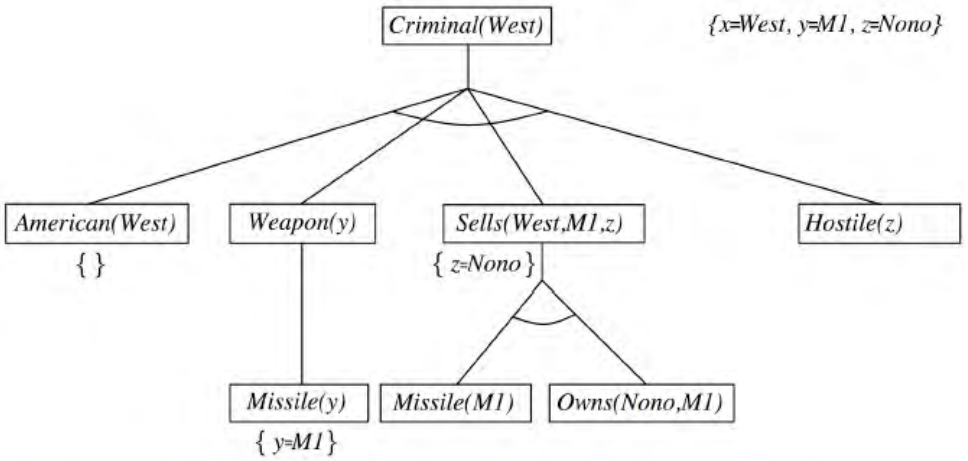
$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
$\forall x \;\; Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$
$Missile(x) \Rightarrow Weapon(x)$ $\quad$ $Enemy(x, America) \Rightarrow Hostile(x)$
$American(West)$ $\quad$ $Enemy(Nono, America)$

گام بعدی-



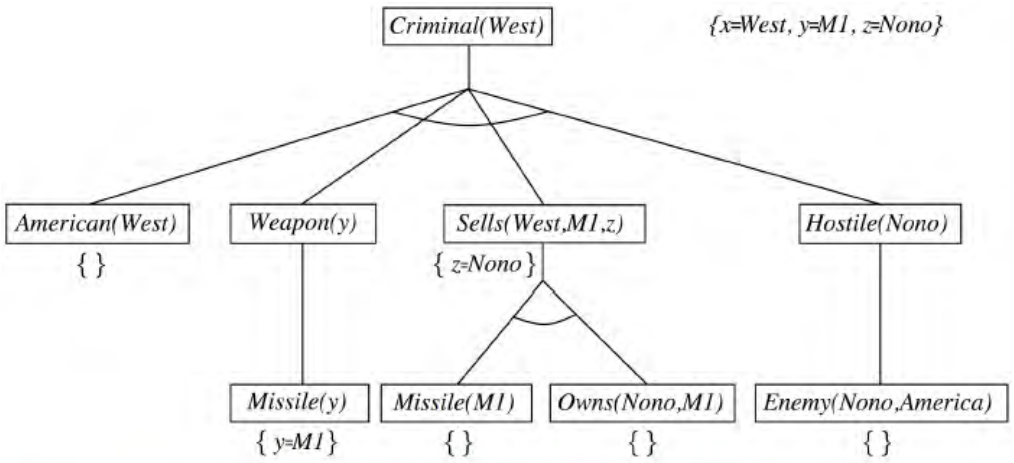
$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
$\forall x \;\; Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$
$Missile(x) \Rightarrow Weapon(x)$ $\quad$ $Enemy(x, America) \Rightarrow Hostile(x)$
$American(West)$ $\quad$ $Enemy(Nono, America)$

گام بعدی-



$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
$\forall x \;\; Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$
$Missile(x) \Rightarrow Weapon(x)$ $\quad$ $Enemy(x, America) \Rightarrow Hostile(x)$
$American(West)$ $\quad$ $Enemy(Nono, America)$

در انتـها-



$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
$\forall x \;\; Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
$Owns(Nono, M_1)$ $\quad$ $Missile(M_1)$
$Missile(x) \Rightarrow Weapon(x)$ $\quad$ $Enemy(x, America) \Rightarrow Hostile(x)$
$American(West)$ $\quad$ $Enemy(Nono, America)$

کاربردی از زنجیره‌ی پسرو

زنجیره‌ی پسرو می‌تواند برای سیستم‌های برمبنای پرس‌وجو[۱] خیلی مؤثّر باشد؛ در ضمن، این روش برای برنامه‌نویسی منطقی[۲] به طور گسترده‌ای استفاده می‌شود.

روش تحلیل: خلاصه

همان طور که گفتیم، تحلیل، روشی برای اثبات عبارت‌های منطق مرتبه‌ی اوّل است؛ همان طور که دیدیم، تحلیل، در منطق گزاره‌ای هم به این صورت استفاده می‌شد: $(A \vee B) \wedge (\neg B \vee C) \Rightarrow (A \vee C)$؛ تحلیل در منطق مرتبه‌ی اوّل به طریقی مشابه عمل می‌کند. *توجّه کنید که لازم است عبارات به فرم CNF[۳] باشند.*

مطلب مهم:

تحلیل در منطق مرتبه‌ی اوّل- اگر عبارت‌های زیر را به صورت CNF داشته باشیم:

$P_1 \vee \ldots \vee P_n$

$Q_1 \vee \ldots \vee Q_m$

و P_j و $\neg Q_k$، با جایگذاری لیست θ، یکسان شوند، در این صورت عبارت زیر به دست می‌آید:

$subst(\theta, P_1 \vee \ldots \vee P_{j-1} \vee P_{j+1} \ldots P_n \vee Q_1 \vee \ldots Q_{k-1} \vee Q_{k+1} \vee \ldots \vee Q_m)$

مثال:

از عبارت‌های

$P(x, f(a)) \vee P(x, f(y)) \vee Q(y)$

$\neg P(z, f(a)) \vee \neg Q(z)$

با استفاده از $\theta = \{x/z\}$، عبارت زیر به وجود می‌آید:

$P(z, f(y)) \vee Q(y) \vee \neg Q(z)$

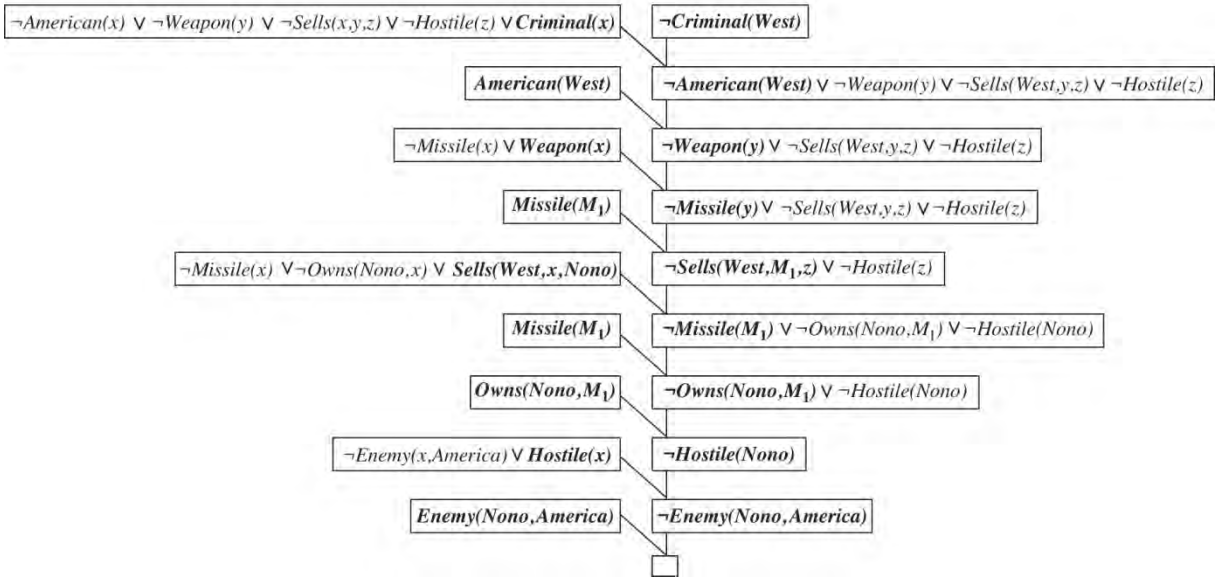
---

۱ - query–based systems

۲ - logic programming؛ **توضیح**- روشی از برنامه‌نویسی است که براساس منطق مرتبه‌ی اوّل می‌باشد. زبان برنامه‌نویسی منطقی اصلی، پرولوگ بود. (Babylon> FOLDOC)

۳ - این مورد در فصل «منطق گزاره‌ای» همین کتاب توضیح داده شده است.

مثال- اثبات جنایت‌کار بودن کلنل وست به روش تحلیل: با کلزهای صریح



کامل بودن برخی از روش‌های استنتاج

جدول صحّت، برای منطق مرتبه‌ی اوّل، کامل نیست، چون ممکن است اندازه‌ی جدول صحّت، نامحدود باشد.

تعمیم قیاس استثنائی، برای منطق مرتبه‌ی اوّل، کامل نیست؛ (☑ **کنکور سراسری مهندسی کامپیوتر سال ۸۸**- تعمیم قیاس استثنائی، برای پایگاه‌های دانشی که فقط شامل کلزهای صریح منطق مرتبه‌ی اوّل هستند، کامل است.)

تحلیل، برای منطق مرتبه‌ی اوّل، صحیح و کامل است.

چکیده‌ی مطلب‌های فصل دوازدهم

متغیّری که دارای سور عمومی است، می‌تواند با هر مقدار ممکن، جایگزین شود.

متغیّری که دارای سور وجودی است، باید دست‌کم برای یک مقدار درست باشد، می‌توانیم این مقدار را k بنامیم، که در این صورت، به k، یک ثابتِ اِسکُلِم می‌گوئیم.

هر پایگاه دانش منطق مرتبه‌ی اوّل می‌تواند به صورت گزاره‌بندی شده بیان شود؛ در این مورد، مسأله‌ای که وجود دارد این است که *گزاره‌بندی می‌تواند تعداد زیادی جمله‌های نامربوط را به وجود آورد.*

یکی‌سازی، این است که [برخلاف گزاره‌بندی،] فقط برای عباراتی که به ما کمک می‌کنند تا چیزهایی را ثابت نماییم، می‌خواهیم جایگزین‌هایی پیدا کنیم.

اگر بیش از یک جانشین وجود داشته باشد که بتوانند دو عبارت که به نظر می‌رسد یکسان هستند را بسازند، کلّی‌ترین یکی کننده‌ها را در نظر می‌گیریم.

در روش زنجیره‌ی پیشرُو، اثبات‌ها با واقعیّت‌ها یا مقدّم‌ها شروع می‌شوند و جمله‌های جدید با استفاده از روش تعمیم قیاس استثنائی به دست می‌آیند، تا زمانی که جمله‌ی هدف یا پرسش(پرس و جو) به وجود آید.

در روش زنجیره‌ی پسرُو، به طور معکوس، از هدف به طرف واقعیّت‌هایی که باید برای هدف نشان داده شوند، حرکت می‌کنیم.

تحلیل، در منطق مرتبه‌ی اوّل، شبیه تحلیل، در منطق گزاره‌ای عمل می‌کند.

یادآوری یا تکمیل مطلب‌های فصل دوازدهم

تعریف- «یکی کننده» را تعریف کنید.

جواب- یک، جانشینی برای متغیّرهای درون دو عبارت اتمیک است، طوری که یکسان(یکی) شوند.[۱]

مطلب- حساب وضعیّت[۲]، برای نمایش(بازنمایی) و استدلال در دامنه‌های پویا(دینامیک) می‌باشد. در منطق مرتبه‌ی اوّل، جمله‌ها یا درست هستند و یا غلط هستند و همین طور هم باقی می‌مانند و تغییر نمی‌کنند. حساب وضعیّت، تغییر در سناریوها را به صورت مجموعه‌ای از فرمول‌های منطق مرتبه‌ی دوّم[۳] نمایش می‌دهد. منطق مرتبه‌ی دوّم، دارای قدرت بیان بیش‌تر از منطق مرتبه‌ی اوّل است. به عنوان مثال، در منطق مرتبه‌ی اوّل، راهی برای بیان اینکه افراد a و b دارای دست کم یک خصوصیّت مشترک هستند، به طور صریح وجود ندارد، امّا در منطق مرتبه‌ی دوّم می‌توانیم بگوییم:

$$\exists P\ (P(a)\wedge P(b))$$

مثال‌های بیش‌تر در مورد منطق مرتبه‌ی دوّم:

$$\exists P\ P(John)$$

خصوصیّتی به نام P وجود دارد که John دارای آن است.

$$\forall F\ F(John)\vee\neg F(John)$$

(اصل دوظرفیّتی(دوبنیانی، دووالانسی)[۴])

برای هر خصوصیّت، یا John دارای آن خصوصیّت هست و یا نیست.[۱]

---

۱ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۱۹۹۳ میلادی

۲ - situation calculus

۳ - *Second-Order Logic*(*SOL*)

۴ - principle of bivalence

سؤال- آیا عبارت‌های زیر یکسان هستند؟

$\forall x\ \forall y\ likes(sohrab, dinner^{۲}(x)) \rightarrow (today^{۳}(y) \rightarrow dinner(x))$

$\forall z\ \forall t\ likes(sohrab, dinner(cooked\_by^{۴}(z))) \rightarrow (today(t) \rightarrow dinner(cooked\_by(sara^{۵})))$

جواب- **«بله»**؛ با جانشینی:

$\{x/\ cooked\_by(z),\ z/\ sara,\ t/y\}^{۶}$

سؤال- با داشتن جمله‌های منطق مرتبه‌ی اوّل زیر که به صورت CNF هستند:

1. $\neg R(x) \vee L(x)$
2. $\neg D(y) \vee \neg L(y)$
3. $D(A)$
4. $I(A)$

از روش تحلیل برای اثبات پرس و جوی زیر با به وجود آوردن یک درخت اثبات استفاده نمائید:

$\exists z\ (I(z) \wedge \neg R(z))$

جواب- اوّل، پرس و جو را منفی کرده و آن را به CNF تبدیل کنید:

$\neg I(z) \vee R(z)$

سپس درخت اثبات رَد(تکذیب، تناقض)[۷] را به وجود آورید:

---

۱ - http://en.wikipedia.org/wiki/Situation_calculus و مطالب «واسیلاْیزْ پاپاتِکسیرِزْ(Vassilis Papataxiarhis)»، عضو گروه تحقیقاتی دانشکده‌ی انفورماتیک و مخابرات دانشگاه شهر آتن کشور یونان

۲ - در لغت به معنی «شام» است.

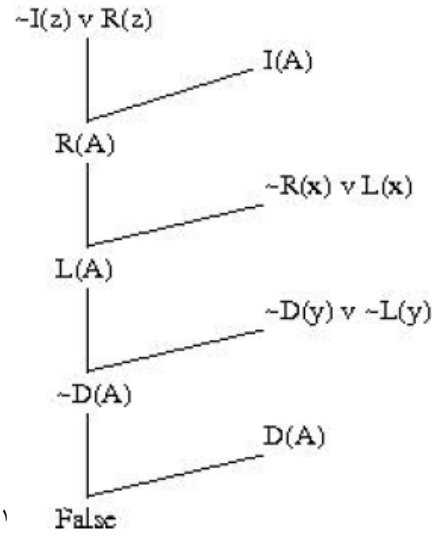
۳ - در لغت به معنی «امروز» است.

۴ - «cooked by»، در لغت به معنی «پخته‌شده به وسیله‌ی» است.

۵ - «سارا»، در زبان انگلیسی نامی است که برای یک زن استفاده می‌شود. (Babylon> Babylon English)

۶ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۳۱ ژانویّه‌ی سال ۲۰۱۱ میلادی

۷ - refutation

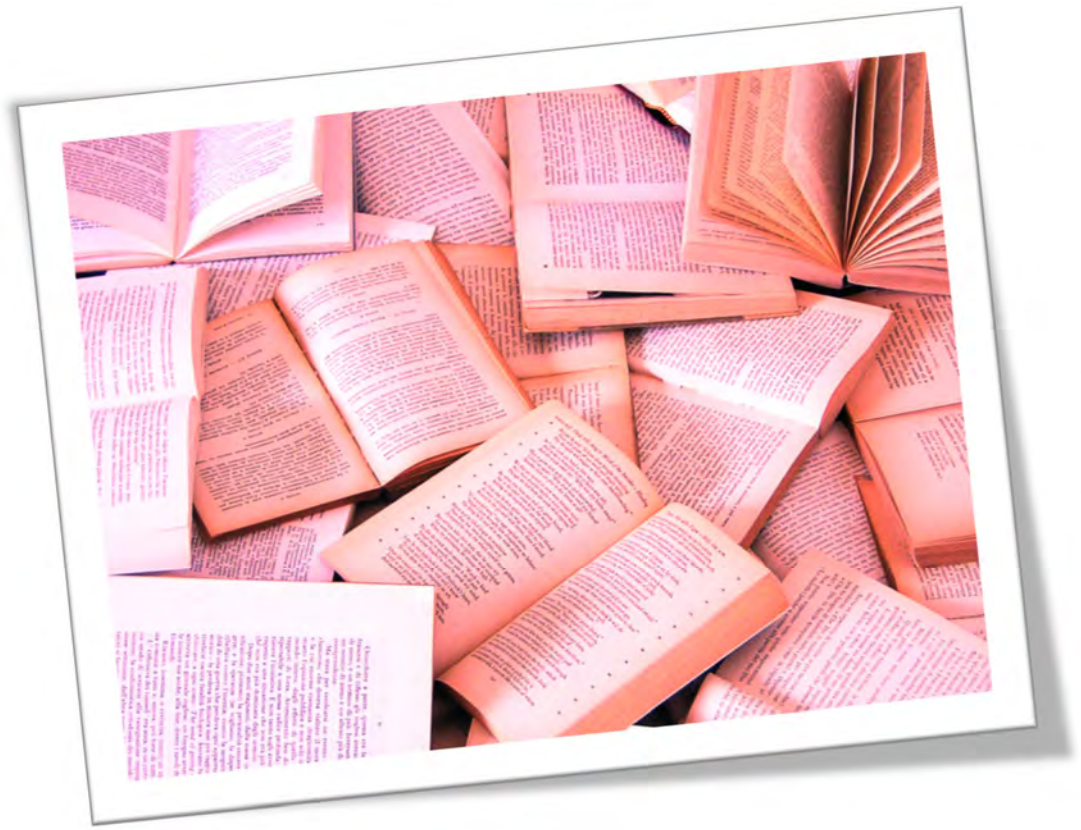


۱ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «چاک دایر»، دانشگاه ویسکانسین-مَدیسُون کشور آمریکا، ۲۲ دسامبر سال ۲۰۰۹ میلادی

فصل سیزدهم

نامعلومی[۱] (عدم‌قطعیّت)

۱ - uncertainty یا ابهام یا عدم‌قطعیّت



فهرست برخی از عنوان‌های نوشته‌ها

تصمیم‌گیری با وجود(تحت) نامعلومی

مبانی احتمال

استنتاج با توزیع‌های پیوسته‌ی احتمال

احتمال شرطی

نُرمال‌ْسازی

استقلال

استقلال شرطی

قانون بِیْزْ

انگیزه

مثال- بیایید عملکرد A_t را ترک کردن به مقصد فرودگاه، t دقیقه قبل از پرواز قرار دهیم؛ آیا در زمان A_t به موقع در فرودگاه خواهم بود؟

مشکلات:

۱) مشاهده‌پذیری جزئی[۱]؛ مثل: حالت جادّه، برنامه‌ی رانندگان دیگر خودروها و.....

۲) حسگرهای پر پارازیت(اغتشاش)[۲]؛ مثلاً گزارش‌های ترافیک ممکن است درست نباشند.

۳) نامعلومی در نتایج عملکرد[۳]؛ مثلاً ممکن است تایر خودرومان پنچر شود و.....

۴) پیچیدگی بیش از حدّ[۴] مدلسازی[۵] و پیش‌بینی ترافیک[۶]

بنابراین، یک شیوه‌ی منطقی یا با ریسک‌های دروغ[۷] است؛ مثلاً [اینکه بگوییم،] «با $A_{۲۵}$، به موقع به فرودگاه خواهم رسید»؛ یا به نتایجی منجر می‌شود که خیلی برای تصمیم‌گیری، ضعیف هستند؛ مثلاً «$A_{۲۵}$، در صورتی مرا به موقع خواهد رساند که تصادفی در راه رخ ندهد؛ باران نبارد؛ لاستیک خودرو من بی‌عیب باشد و.....» ($A_{۱۴۴۰}$، معقولانه است که مرا به موقع برساند، امّا من باید در طول شب در فرودگاه بمانم!.....).

۱- partial observability

۲- noisy sensors

۳- uncertainty in action outcomes

۴- immense

۵- modelling

۶- predicting traffic

۷- risks falsehood

نامعلومی

در برخی از محیط‌های جالب عامل‌ها، نامعلوم بودن یا عدم‌قطعیّت، دارای نقش زیادی می‌باشد؛ به عبارت دیگر؛ *عملکردها ممکن است دارای اثرات نامعلومی باشند؛* مثل پرتاب یک پیکان به هدف:

عامل‌ها ممکن است وضعیّت درست جهان را ندانند؛ در ضمن، ارتباط‌های میان واقعیّت‌ها ممکن است به صورت معلوم نباشند؛ مثلاً گاهی وقت‌ها، در زمانی که هوا ابری می‌باشد، باران می‌بارد؛ به عبارت دیگر، هر وقت که هوا ابری باشد، باران نمی‌بارد!:

عامل‌های هوشمند باید نامعلوم بودن یا ابهام را برّرسی نمایند.

نکته:

منطق فازی[۱]، از درجه‌ی درستی[۲] استفاده می‌کند، نه نامعلومی[۳].

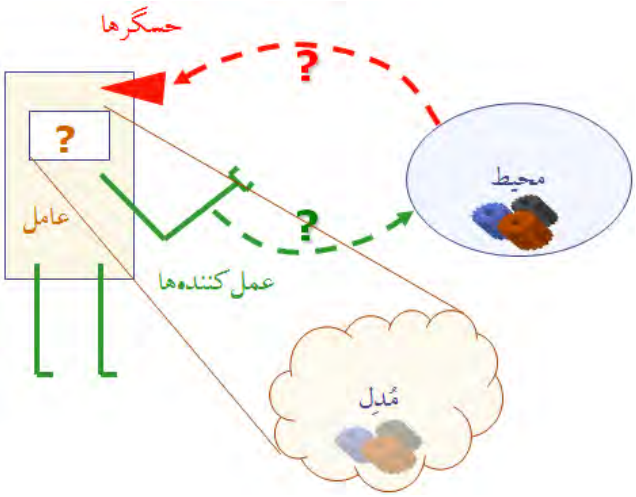
---

۱ - Fuzzy logic: **تعریف**- یک شکل از منطق ریاضی است که در آن مقادیر بین صفر و یک هم دارای ارزش درست هستند. (Babylon> WordNet 2.0)

۲ - degree of truth

۳ - uncertainty

عامل نامعلوم[۱]



انواع نامعلومی

نامعلومی در دانش قبلی

به عنوان مثال، بعضی از دلایل بیماری ناشناخته‌اند و در دانش یک عامل دستیار پزشک بیان نشده‌اند.

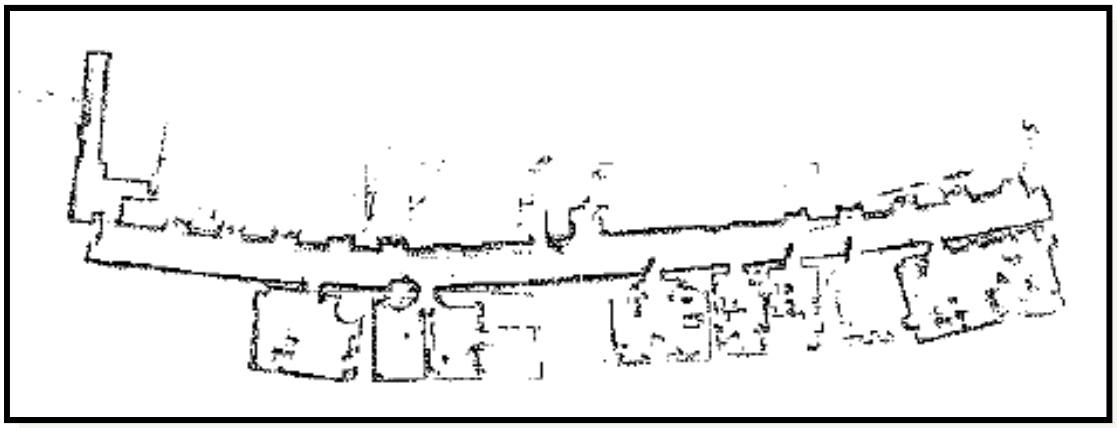
نامعلومی در عملکردها

همان طور که گفتیم، مثل پرتاب یک پیکان به هدف.

نامعلومی در ادراک

مثلاً ممکن است حسگرها اطّلاعات کامل یا دقیق را در مورد جهان برنگردانند؛ مثلاً ممکن است یک رُبُوت، مسیر درست را پیدا نکند:

۱ - uncertain agent



چگونه با نامعلومی برخورد کنیم؟

به صورت ناآشکار(ضمنی یا تلویحی)

به موردهایی که نامعلوم هستند، در زمانی که شما می‌توانید کار خود را انجام دهید، بی‌اعتنا باشید و عملیّاتی را بـه وجـود بیاورید که در مقابل نامعلومی، مقاوم هستند.

به صورت آشکار

مدلی از جهان را بسازید که نامعلومی را در وضعیّت‌ها، حرکت‌ها و مشاهده‌هایش در نظر می‌گیرد و در مورد اثر عملیّاتی که مدل انجام می‌دهد، دلیل بیاورید.

منطق و نامعلومی

به زودی خواهیم دید که چگونه منطق را با نامعلومی به کار ببریم؛ مثال‌هایی از منطق را در زیر مشاهده می‌نمایید:

$Studies^{1}(Bart) \vee WatchesTV^{2}(Bart)$

$Hungry^{1}(Jack) \Rightarrow Eats^{2}(Jack, HotDog^{3}) \vee Eats(Jack, Pizza^{4})$

---

۱ - یعنی: «مطالعه می‌کند»

۲ - watches TV، یعنی: «تلویزیون تماشا می‌کند»

$\exists x: Hungry(x)$

متأسّفانه نگرش منطقی دارای برخی اشکالات می‌باشد.

ضعف‌های منطق[۵]

برخی از آنها:

نمی‌توانیم همه‌ی نتیجه‌های ممکن را تعیین کنیم؛ مثلاً نمی‌دانیم که اگر حالا از خانه خارج شویم، آیا به موقع در محلّ مورد نظر خواهیم بود یا نه؟. یا اگر پدال گاز خودرو خود را زیادتر فشار دهیم آیا باعث زودتر رسیدن ما به مقصد می‌شود یا اینکه باعث می‌شود که تصادف کنیم؟!.

ممکن است همه‌ی نتیجه‌های ممکن را ندانیم؛ مثلاً «در صورتی که یک بیمار، دارای دندان درد[۶] باشد، وی ممکن است پوسیدگی[۷] در دندان خود داشته باشد، یا ممکن است دارای بیماری لثه[۸] باشد، یا ممکن است دلیلی دیگر داشته باشد که ما چیزی در مورد آن نمی‌دانیم.»

هیچ راهی برای صحبت در مورد احتمال حوادث نداریم؛ مثلاً «امروز ممکن است که من با تیر چراغ برق برخورد کنم.»

کمّیّت و کیفیّت

منطق، یک نگرش کمّی به نامعلومی را ارائه می‌دهد؛ می‌توانیم بگوییم که یک اتّفاق بیش‌تر از دیگری روی می‌دهد، یا اینکه ممکن است یک اتّفاق روی دهد؛ این روش در موردهایی که آماری نداریم یا می‌خواهیم به صورت فرضی‌تری دلیل بیاوریم، مفید می‌باشد. *احتمال، به ما اجازه می‌دهد که به صورت کیفی بحث کنیم و احتمال رخ دادن یک اتّفاق را با یک عدد بیان کنیم.*

---

۱ - در لغت یعنی: «گرسنه»
۲ - یعنی: «می‌خورد»
۳ - هات‌داگ؛ نوعی سوسیس(sausage) است.
۴ - پیتزا (نوعی غذای ایتالیایی)
۵ - weaknesses with logic
۶ - toothache
۷ - cavity
۸ - gum

نامعلومی و عقلانیّت

تعریف ما از عقلانیّت را به یاد بیاورید؛ یک عامل عقلانی، عاملی است که با بیشینه نمودن معیارکارآیی، عمـل می‌نمایـد؛ چگونه عامل هوشمند را در یک جهان دارای نامعلومی تعریف نماییم؟؛ می‌گـوییم، عملکـرد عامـل، دارای نتیجه‌هایی اسـت و هـر نتیجه را با یک عدد، که نشان دهنده‌ی احتمال آن است، بیان می‌نماییم. سپس یک عامل می‌تواند به هر کدام از نتایج ممکن و عدد آنها(احتمال رخ دادن نتیجه) توجّه نماید و عملی را که دارای بالاترین سود مورد انتظار می‌باشد، انتخاب نماید.

توجّه:

بیش‌تر مطلب‌هایی که در ادامه‌ی این فصل آمده‌اند را در درس‌هایی که قبلاً - نه در این کتاب - گذرانده‌اید، خوانده‌اید.

احتمال[۱]

یک چارچوب مشهور و قابل فهم برای نامعلومی می‌باشد.

چرا از احتمال استفاده می‌کنیم؟ برخی از رویدادهای منطقی وابسته باید روابط احتمالاتی داشته باشند.

اساس احتمال

به عنوان مثال، احتمال‌های P(BartStudied[۱]) = 0.01[۲] و P(Hungry(Jack)) = 0.99 را در نظر بگیرید؛ توجّـه کنید که خود گزاره(BartStudied و (Hungry(Jack) ممکن است درست یا غلط باشد و فرق دارد با اینکه بگوییم، عبـارت، به صورت جزئی درست است.

۱ - Probability

متغیّرهای تصادفی[۳]

یک متغّیر تصادفی، متغّیر یا گزاره‌ای است که مقدارش ناشناخته است؛ به بیان دیگر، یک گزاره که با احتمال p درست است و با احتمال 1-p نادرست است، یک متغّیر تصادفی، با توزیع (p,1-p) است و دارای دامنه‌ای از مقادیر است که می‌تواند آنها را به خود بگیرد؛ این متغّیرها می‌توانند از موردهای زیر باشند:

- **بولیّن(درست یا غلط)؛** مثل: Hungry(Jack) و isRaining[۴]
- **گسسته؛** مقادیر، از یک دامنه‌ی *قابل شمارش* گرفته می‌شوند؛ مثل دما[۵]: < گرم[۶]، خنک[۷]، ملایم[۸] > و پیش‌بینی وضع هوا: < آفتابی[۹]، ابری[۱۰]، بارانی >
- **پیوسته؛** مقادیر می‌توانند از یک *فاصله(دامنه‌ی غیرقابل شمارش)* گرفته شوند، نظیر: [۱و ۰]؛ مثل سرعت[۱۱]، زمان و مکان[۱۲]

در اینجا تمرکز ما بیش‌تر بر روی مورد گسسته خواهد بود.

به عنوان مثال، اگر یک ظرف دارای مهره‌هایی به سه رنگ قرمز، زرد و آبی باشد، *رنگ یک مهره که به طور تصادفی از ظرف بیرون آورده شده است، یک متغّیر تصادفی دارای سه مقدار ممکن قرمز یا زرد یا آبی است.* توزیع تصادفی یک متغّیر تصادفی X، که دارای n مقدار $x_1, x_2, \ldots, x_n$ است، به صورت $(p_1, p_2, \ldots, p_n)$ می‌باشد.

توزیع پیوسته[۱۳]

برای k متغّیر تصادفی $X_1, \ldots, X_k$، توزیع پیوسته‌ی این متغّیرها، جدولی است که در آن هر عدد احتمال یک ترکیب از مقدارهای $X_1, \ldots, X_k$ را ارائه می‌نماید. توزیع پیوسته می‌تواند هر پرسش درباره‌ی یک دامنه را پاسخ دهد.

---

۱ - Bart studied، در اینجا یعنی: «بارت مطالعه کرده باشد»

۲ - P(BartStudied) = 0.01، یعنی: «احتمال این که آقای بارت مطالعه کرده باشد برابر است با یک صدم»

۳ - random variables

۴ - در اینجا یعنی: «باران می‌بارد[؟]»

۵ - temperature

۶ - hot

۷ - cool

۸ - mild

۹ - sunny

۱۰ - overcast

۱۱ - velocity

۱۲ - position

۱۳ - joint distribution

مثال- توزیع احتمال، مقادیر را برای همه‌ی انتساب‌های ممکن ارائه می‌دهد:

P(Weather) = <برفی، ابری، بارانی، آفتابی>

P(Weather) = <0.72,0.1,0.08,0.1>

توجّه نمایید که مجموع اعداد، برابر با عدد «یک» می‌باشد (۰,۷۲+۰,۱+۰,۰۸+۰,۱=۱,۰۰).

مثال:

	Toothache	¬Toothache
Cavity	0.04	0.06
¬Cavity	0.01	0.89

P(¬Cavity∧Toothache) P(Cavity∧¬Toothache)

تصمیم‌گیری با وجود(تحت) نامعلومی

در مثال رفتن به فرودگاه، که در گذشته در مورد آن صحبت کردیم، تصوّر نمایید موردهای زیر را داریم:

P(مرا به موقع برساند، A_{25})= ۰,۰۴

P(مرا به موقع برساند، A_{90})= ۰,۷۴

P(مرا به موقع برساند، A_{120})= ۰,۹۵

P(مرا به موقع برساند، A_{1440})= ۰,۹۹۹

کدام مورد بالا را باید انتخاب نماییم؟ انتخاب، وابسته به صلاح‌دیدهای ما برای از دست ندادن پرواز، وضعیّت فرودگاه و..... می‌باشد.

قضیّه‌ی(تئوری) سودمندی[1]، برای بیان صلاح‌دیدها و استدلال در مورد صلاح‌دیدها استفاده می‌شود.

قضیّه‌ی تصمیم‌گیری[1]، برابر است با، قضیّه‌ی سودمندی + قضیّه‌ی احتمال. به بیانی دیگر، قضیّه‌ی ترکیب کننده‌ی برتری(اولویّت)ها با احتمال رخداد یک نتیجه، تئوری تصمیم‌گیری نام دارد.

۱- Utility theory

مبانی احتمال

با یک مجموعه از فضای نمونه‌ی ساده‌ی Ω[۲] شروع می‌کنیم؛ مثل شش حالت ممکن در پرتاب یک تاس.

Ω(فضای نمونه) = {۱، ۲، ۳، ۴، ۵، ۶}

اگر $\omega \in \Omega$ باشد. در یک بار پرتاب یک تاس، احتمال آمدن هر کدام از اعداد ۱ تا ۶ روی شش وجه تاس، بین ۰ تا ۱ است؛ پس داریم:

$$۰ \le P(\omega) \le ۱$$

در ضمن، مجموع احتمال‌های آمدن هر وجه تاس، برابر ۱ است؛ یعنی، $\Sigma_{\omega} P(\omega) = ۱$.

مثال- در پرتاب یک تاس هَمْگِن[۳] داریم:

P(1)=P(2)=P(3)=P(4)=P(5)=P(6)=1/6؛ می‌بینید که احتمال آمدن هر وجه تاس، برابر با $\frac{1}{6}$ می‌باشد.

یک رویداد A، زیرمجموعه‌ای از Ω(فضای نمونه) می‌باشد:

$$P(A) = \sum_{\{\omega \in A\}} P(\omega)$$

مثلاً احتمال اینکه عددی که در یک بار پرتاب یک تاس ظاهر می‌شود، کم‌تر از ۴ باشد، برابر است با: احتمال ۱ آمدن + احتمال ۲ آمدن + احتمال ۳ آمدن = ۱/۶+۱/۶+۱/۶ = ۱/۲

---

۱ - Decision theory

۲ - اُمِگا(Omega)، بیست و چهارمین و آخرین حرف الفبای یونانی است و با حرف بزرگ، به صورت Ω نوشته می‌شود و با حرف کوچک، به صورت ω نوشته می‌شود. (http://en.wikipedia.org/wiki/Omega)

۳ - تاس همگن، تاسی است که احتمال آمدن هر وجه آن با هم برابر باشد.

استنتاج با توزیع‌های پیوسته‌ی احتمال

آسان‌ترین راه انجام استنتاج احتمالی، این است که یک جدول ارائه کننده‌ی توزیع پیوسته‌ی احتمال را داشته باشیم:

	Humidity[1]=High Outlook[2]=Overcast	Humidity=High Outlook=Sunny	Humidity=Normal Outlook=Overcast	Humidity=Normal Outlook=Sunny
Rain	0.1	0.05	0.15	0.05
¬ Rain	0.2	0.15	0.1	0.2

می‌توانیم توزیع پیوسته‌ی احتمال را برای تشخیص *احتمال نهائی*[3] متغیّر وابسته، با جمع کردن همه‌ی موردهای وابسته به متغیّر که می‌توانند درست باشند، به دست آوریم:

P(Rain)=0.1+0.05+0.15+0.05=0.35

اگر P، یک توزیع تصادفی را برای هر متغیّر تصادفی، نظیر X بیان نماید؛ داریم:

$$P(X = x_i) = \sum_{\{\omega : X(\omega) = x_i\}} P(\omega)$$

مثال- در یک بار پرتاب یک تاس، احتمال اینکه عدد فرد[4] ظاهر شود، برابر است با:

P(Odd=true)=P(1)+P(3)+P(5)=1/6+1/6+1/6=1/2

گزاره‌ها- یک گزاره را به صورت رویداد مجموعه‌ای از فضاهای نمونه، در جایی که گزاره صحیح می‌باشد، تصوّر نمایید. برای متغیّرهای تصادفی بولیّن A و B، اگر رویداد a، مجموعه‌ای از فضاهای نمونه‌ای، به شرط آن که A(ω)=true باشد، آنگاه رویداد ¬a، مجموعه‌ای از فضاهای نمونه‌ای، به شرط آن که A(ω) = false باشد، خواهد بود و رویداد a∧b، موردهایی هستند که A(ω) و B(ω) [، هر دو]، True می‌باشند.

اغلب در هوش مصنوعی، *فضای نمونه‌ای، به وسیله‌ی مقادیر یک مجموعه از متغیّرهای تصادفی تعریف می‌شود*. توجّه نمایید که فضای نمونه‌ای، حاصلضرب کارتزیَن[1] محدوده‌ی متغیّرها می‌باشد. با متغیّرهای بولیّن، که دارای مقدار درست یا غلط می‌باشد، فضای نمونه، با مدل منطق گزاره‌ای، برابر می‌باشد.

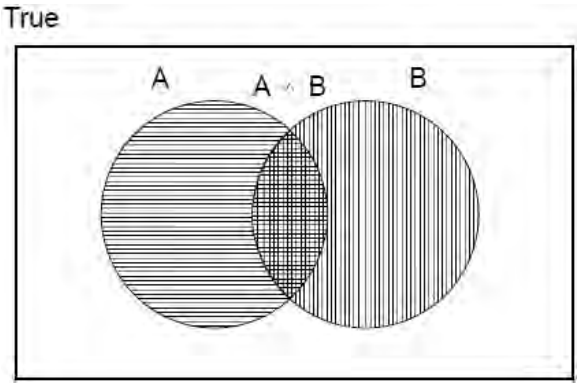
---

۱ - رطوبت
۲ - پیش‌بینی
۳ - marginal probability
۴ - odd

چند فرمول

یک فرمول:

$(a \vee b) \equiv (\neg a \wedge b) \vee (a \wedge \neg b) \vee (a \wedge b) \Rightarrow P(a \vee b) = P(\neg a \wedge b) + P(a \wedge \neg b) + P(a \wedge b)$



فرمولی دیگر:

$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$

فرمولی دیگر:

$P(A \vee \neg A) = P(A) + P(\neg A) - P(A \wedge \neg A) \Rightarrow P(True) = P(A) + P(\neg A) - P(False) \Rightarrow 1 = P(A) + P(\neg A) - 0 \Rightarrow \mathbf{P(A) = 1 - P(\neg A)}$

---

۱ - Cartesian product، **یادآوری**- حاصلضرب کارتزیّن دو مجموعه‌ی X و Y که به صورت $X \times Y$ نشان داده می‌شود، مجموعه‌ای از تمام زوج‌های مرتّبی است که اوّلین عنصر، عضو X و دوّمین عنصر، عضو Y است:

$$X \times Y = \{(x,y) | \ x \in X, \ y \in Y\}$$

مثال‌ها:

$\{1,2\} \times \{3,4\} = \{(1,3), (1,4), (2,3), (2,4)\}$
$\{3,4\} \times \{1,2\} = \{(3,1), (3,2), (4,1), (4,2)\}$

استثناء:

$A \times \phi = \phi \times A = \phi$

(ϕ = تهی)

نکته:

$X \times Y \neq Y \times X$

حاصلضرب کارتزیّن n-تائی به صورت زیر تعریف می‌شود:

$X_1 \times \ldots \times X_n = \{(x_1, \ldots, x_n): x_i \in X_i\}$

(http://en.wikipedia.org/wiki/Cartesian_product)

ظاهر گزاره‌ها

متغیّرهای تصادفی گزاره‌ای یا بولیّن: مثلاً اگر Cavity، برای «آیا من کرم خوردگی دندان دارم؟»، باشد؛ آنگاه Cavity = true، یک گزاره می‌باشد.

احتمال شرطی[1]

در گذشته دیدیم که اگر [به عنوان مثال،] هوا ابری باشد، احتمال بارندگی بالا می‌رود؛ که این، یک احتمال شرطی نام دارد و به صورت P(Rain|Cloudy) نوشته می‌شود.

تعریف احتمال شرطی

$$P(a|b) = \frac{P(a \wedge b)}{P(b)}، \text{ با شرط } P(b) \neq 0$$

با استفاده از فرمول بالا قانون ضرب[2] را به صورت زیر خواهیم داشت:

$$P(a \wedge b) = P(a \mid b)P(b) = P(b \mid a)P(a)$$

مثال:

$$P(Weather,Cavity)=P(Weather|Cavity)P(Cavity)$$

مثال- اگر $P(Rain \wedge Cloudy) = 0.15$ و $P(Cloudy) = 0.3$ باشد؛ P(Rain|Cloudy) چقدر است؟

$$P(Rain \mid Cloudy) = \frac{P(Rain \wedge Cloudy)}{P(Cloudy)} = \frac{0.15}{0.3} = 0.5$$

۱- conditional probability

۲- Product rule

تعمیم- $P(A \wedge B \wedge C) = P(A|B,C)\, P(B|C)\, P(C)$

قانون زنجیره‌ای[1]- برای کاربرد موفّقیّت‌آمیز قانون ضرب به وجود آمده است:

$P(X_1,\ldots,X_n)$

$= P(X_1,\ldots,X_{n-1})\, P(X_n|X_1,\ldots,X_{n-1})$

$= P(X_1,\ldots,X_{n-2})\, P(X_{n-1}|X_1,\ldots,X_{n-2})\, P(X_n|X_1,\ldots,X_{n-1})$

$= \ldots$

$= \prod_{i=1}^{n} P(X_i|X_1, \ldots, X_{i-1})$

مثال:

	Toothache	¬Toothache
Cavity	0.04	0.06
¬Cavity	0.01	0.89

P(¬Cavity∧Toothache) P(Cavity∧¬Toothache)

با توجّه به جدول توزیع پیوسته‌ی بالا داریم:

$P(\text{Toothache}) = P((\text{Toothache} \wedge \text{Cavity}) \vee (\text{Toothache} \wedge \neg\text{Cavity}))$

$= P(\text{Toothache} \wedge \text{Cavity}) + P(\text{Toothache} \wedge \neg\text{Cavity}) = 0.04 + 0.01 = 0.05$

$P(\text{Toothache} \vee \text{Cavity}) = P((\text{Toothache} \wedge \text{Cavity}) \vee (\text{Toothache} \wedge \neg\text{Cavity}) \vee (\neg\text{Toothache} \wedge \text{Cavity})) = 0.04 + 0.01 + 0.06 = 0.11$

می‌توانیم همچنین احتمالات شرطی را محاسبه نماییم:

$P(\text{Cavity} \mid \text{Toothache}) = P(\text{Cavity} \wedge \text{Toothache}) / P(\text{Toothache})$

$P(\text{Cavity} \wedge \text{Toothache}) = ?$

با توجّه به جدول برابر است با ۰.۰۴

$P(\text{Toothache}) = ?$

حاصل این احتمال را هم که در گذشته در همین مثال محاسبه کردیم؛ برابر بود با: ۰.۰۵ ؛ درنتیجه داریم:

$P(\text{Cavity} \mid \text{Toothache}) = 0.04/0.05 = 0.8$

۱- Chain rule

نُرمال‌ْسازی- در فرمول احتمال شرطی، مقسومٌ علیه[۱] می‌تواند به صورت یک ثابت نرمال سازی α دیده شود؛ مثلاً برای مثال قبل داریم:

$$P(Cavity|Toothache) = P(Cavity \wedge Toothache) / P(Toothache) = \alpha\,[P(Cavity \wedge Toothache)] = \alpha \times 0.04 = \frac{1}{0.05} \times 0.04 = 20 \times \frac{4}{100} = 0.8$$

استقلال[۲]

در برخی از موردها می‌توانیم چیزها را با توجّه به اینکه یک متغیّر دارای هیچ اثری بر روی دیگری نمی‌باشد، ساده نماییم؛ برای مثال، اگر ما یک متغیّر چهارم DayOfWeek(روز هفته) را به محاسبه‌ی بارندگی اضافه نماییم، چه تغییری به وجود می‌آید؟؛ از آنجایی که روز هفته بر احتمال باران تأثیری نخواهد داشت، داریم:

$$P(Rain|Cloudy,Monday^{3}) = P(Rain|Cloudy,Tuesday^{4}) \ldots = P(Rain|Cloudy)$$

گفتیم که DayOfWeek و Rain مستقل می‌باشند؛ بنابراین، می‌توانیم توزیع پیوسته‌ی احتمال بزرگ‌تر را به زیر جدول‌هایی مجزّا تقسیم نماییم؛ استقلال به ما کمک خواهد کرد که دامنه را به تکّه‌های مجزّا تقسیم نماییم.

A و B در صورتی مستقل هستند که $P(A|B)=P(A)$، یا $P(B|A)=P(B)$، یا $P(A,B)=P(A)P(B)$.

استقلال مطلق، خیلی مفید است؛ ولی نادر(کمیاب)[۵] می‌باشد؛ مثلاً دندانپزشکی، زمینه‌ی بزرگی با صدها متغیّر می‌باشد که هیچکدام مستقل نیستند. در این مورد چه کاری انجام بدهیم؟.

استقلال شرطی[۶]

اگر A و B مستقل شرطی باشند و C هم داده شده باشد، داریم:

$$P(A,B|C) = P(A|C)P(B|C)$$
$$P(A|C,B) = P(A|C)$$

۱- denominator
۲- Independence
۳- دوشنبه
۴- سه‌شنبه
۵- rare
۶- conditional independence

مطلب مهم:

در بیش‌تر موردها استفاده از استقلال شرطی اندازه‌ی ارائه‌ی توزیع پیوسته را از حالت نمایی n، به حالت خطّی n کاهش می‌دهد. استقلال شرطی، پایه‌ای‌ترین و نیرومندترین دانش، درباره‌ی محیط‌های نامعّین می‌باشد.

قانون بِیْز[۱]

با استفاده از قانون ضرب داریم: $P(a\wedge b) = P(a|b)\ P(b)$ و $P(a\wedge b) = P(b|a)\ P(a)$، در نتیجه می‌توانیم این تساوی‌ها را برابر هم قرار دهیم:

$P(a\wedge b) = P(a|b)\ P(b) = P(b|a)\ P(a)$ ، در نتیجه قانون بیز را به صورت زیر خواهیم داشت:

$$P(a|b) = \frac{P(b|a)P(a)}{P(b)}$$

یا به شکل نرمال داریم:

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)} = \alpha P(X \mid Y)P(Y)$$

مثال برای قانون بیز– این مفروضات را در نظر بگیرید: مِنَنْژیت[۲]، باعث خشکی گردن[۳] در ۵۰٪ از بیماران می‌شود؛ پس $P(stiffNeck|Meningitis)=0.5$؛ احتمال مننژیت برابر است با، $\frac{1}{50000}$؛ در نتیجه $P(meningitis)=0.00002$؛ احتمال خشکی گردن برابر است با، $\frac{1}{20}$؛ در نتیجه $P(stiffNeck)=0.05$؛ حال یک بیمار با خشکی گردن مراجعه می‌نماید؛ احتمال اینکه او دارای مننژیت باشد چقدر است؟

$$P(meningitis|stiffNeck) = \frac{P(stiffNeck|meningitis)\times P(meningitis)}{P(stiffNeck)} = \frac{0.5\times 0.00002}{0.05} = 0.0002$$

---

۱ - Bayes' Rule

۲ - meningitis؛ به سه پرده که مغز(brain) و نخاع(spinal cord) را می‌پوشانند، مِنَنْژ(meninges) یا پرده‌های مغز گفته می‌شود (Babylon> Babylon English) و به التهاب(inflammation) این سه پرده، مننژیت گفته می‌شود. (Babylon> Babylon English)

۳ - stiff neck

مثـال- اگـر داشـته باشـیم: P(Cavity)=0.1، P(Toothache)=0.05 و P(Cavity|Toothache)=0.8؛ در ایـن صورت P(Toothache|Cavity) را به دست آورید.

با استفاده از قانون بیز داریم:

$$P(\text{Toothache}|\text{Cavity}) = (0.8 \times 0.05) / 0.1 = 0.4$$

تعمیم

با توجّه به دو مورد زیر

$$P(A \wedge B \wedge C) = P(A \wedge B|C)\, P(C) = P(A|B,C)\, P(B|C)\, P(C)$$
$$P(A \wedge B \wedge C) = P(A \wedge B|C)\, P(C) = P(B|A,C)\, P(A|C)\, P(C)$$

داریم:

$$P(B|A, C) = \frac{P(A|B, C)P(B|C)}{P(A|C)}$$

چکیده‌ی مطلب‌های فصل سیزدهم

در برخی از محیط‌های عامل‌ها، عملکردها ممکن است دارای اثرات نامعلومی باشند.

نگرش منطقی دارای برخی اشکالات می‌باشد؛ از جمله اینکه نمی‌توانیم همه‌ی نتیجه‌های ممکن را تعیین کنیم؛ ممکن است همه‌ی نتیجه‌های ممکن را ندانیم؛ و هیچ راهی برای صحبت در مورد احتمال حوادث نداریم.

برای تعریف عامل هوشمند، در یک جهان دارای نامعلومی می‌گوییم، عملکرد عامل، دارای نتیجه‌هایی است و هر نتیجه را با یک عدد، که نشان دهنده‌ی احتمال آن است، بیان می‌نماییم؛ سپس یک عامل می‌تواند به هر کدام از نتایج ممکن و عدد آنها(احتمال رخ دادن نتیجه) توجّه نماید و عملی را که دارای بالاترین سود مورد انتظار می‌باشد، انتخاب نماید.

احتمال، یک چارچوب مشهور و قابل فهم برای نامعلومی می‌باشد.

در بیش‌تر موردها استفاده از استقلال شرطی، اندازه‌ی ارائه‌ی توزیع پیوسته را از حالت نمایی n، به حالت خطّی n کاهش می‌دهد. استقلال شرطی، پایه‌ای‌ترین و نیرومندترین دانش درباره‌ی محیط‌های نامعیّن می‌باشد.

یادآوری یا تکمیل مطلب‌های فصل سیزدهم

تعریف– تئوری احتمال را تعریف کنید.

جواب– به هر جمله، درجه‌ای از باور(گمان، عقیده) در محدوده‌ی صفر تا یک نسبت می‌دهد.[۱]

سؤال– $P(a \vee b)$ برابر با چیست؟

جواب– $P(a) + P(b) - P(a \wedge b)$[۲]

مثالی برای فرمول $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$– در پرتاب دو سکّه، رویدادها به صورت زیرند (H، برای شیر آمدن و T، برای خط آمدن):

$\{(H,H),(H,T),(T,H),(T,T)\}$

اگر رویداد E، شیر آمدن اوّلین سکّه باشد:

$\{(H,H),(H,T)\}$

و رویداد F، شیر آمدن دوّمین سکّه باشد:

$\{(H,H),(T,H)\}$

آنگاه:[۳]

$P(E \vee F) = P(E) + P(F) - P(E \wedge F) = ½ + ½ - ¼ = ¾$

---

۱ – آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۲ – کوییز شماره‌ی ۴ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۳ – مطلب‌های درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا

سؤال- فرمول استقلال چیست؟

جواب- $P(a \wedge b) = P(a)\ P(b)$ [۱]

سؤال- فرمول استقلال شرطی چیست؟

جواب- $P(a \wedge b|c) = P(a|c)\ P(b|c)$ [۲]

سؤال- فرمول قانون ضرب(قانون زنجیره‌ای) چیست؟

جواب- $P(a \wedge b \wedge c) = P(a|\ b \wedge c)\ P(b|c)\ P(c)$

سؤال- با فرض اینکه A و B مستقل هستند، سایر مقدارهای جدول توزیع پیوسته‌ی زیر را به دست آورید.

	A = T	A = F
B = T	۳/۱۲	۶/۱۲
B = F		

جواب- از آنجایی که A و B مستقل هستند، داریم:

$P(A \cap B) = P(A, B) = P(A) \times P(B)$

در اینجا داریم:

$P(A, B) = 3/12 = P(A) \times P(B)$
$P(\neg A, B) = 6/12 = P(\neg A) \times P(B)$

بنابراین، می‌توانیم $P(A)$ و $P(B)$ را به صورت زیر به دست آوریم:

$P(\neg A)\ P(B) = (1 - P(A)) \times P(B) = P(B) - P(A) \times P(B) = P(B) - 3/12 = 6/12 \Rightarrow P(B) = 6/12 + 3/12 = 9/12 = 3/4,\ 3/12 = P(A) \times 9/12 \Rightarrow P(A) = 3/9 = 1/3$

پس داریم:

$P(A, \neg B) = P(A) \times P(\neg B) = P(A) \times (1 - P(B)) = 1/3 \times (1 - 3/4) = 1/12$
$P(\neg A, \neg B) = P(\neg A) \times P(\neg B) = (1 - P(A)) \times (1 - P(B)) = (1 - 1/3) \times (1 - 3/4) = 2/12 = 1/6$

در نتیجه کامل شده‌ی جدول توزیع پیوسته‌ی داده شده در صورت سؤال به صورت زیر است:[۳]

---

۱ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۲ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۳ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «چاک دایر»، دانشگاه ویسکانسین-مَدیسُون کشور آمریکا، ۲۲ دسامبر سال ۲۰۰۹ میلادی

	A = T	A = F
B = T	۳/۱۲	۶/۱۲
B = F	۱/۱۲	۱/۶

تست– استنتاج با شمارش[۱] براساس چیست؟

۱) احتمال‌های شرطی میان رویدادهای اتمیک

۲) مدرک تصادفی به دست آمده از رویدادهای اتمیک

۳) توزیع پیوسته‌ی کامل رویدادهای اتمیک

۴) لیستی از رویدادهای اتمیک، به تنهایی

جواب– گزینه‌ی «۳» درست است.[۲] مثلاً اگر جدول توزیع پیوسته‌ی احتمال زیر را داشته باشیم:

	Toothache	¬Toothache
Cavity	0.04	0.06
¬Cavity	0.01	0.89

برای محاسبه‌ی P(Toothache)، تمام قسمت‌هایی که در آنها Toothache، True است را با هم جمع می‌کنیم:

	Toothache	¬Toothache
Cavity	0.04	0.06
¬Cavity	0.01	0.89

$P(Toothache) = P((Toothache \wedge Cavity) \vee (Toothache \wedge \neg Cavity)) = P(Toothache \wedge Cavity) + P(Toothache \wedge \neg Cavity) = 0.04 + 0.01 = 0.05$

یا برای محاسبه‌ی $P(Toothache \vee Cavity)$ داریم:

	Toothache	¬Toothache
Cavity	0.04	0.06
¬Cavity	0.01	0.89

$P(Toothache \vee Cavity) = P((Toothache \wedge Cavity) \vee (Toothache \wedge \neg Cavity) \vee (\neg Toothache \wedge Cavity)) = 0.04 + 0.01 + 0.06 = 0.11$

تست– در جامعه، ۱۰٪ افراد، سرماخوردگی دارند، ۲۰٪ افراد، سرفه می‌کنند، و یک فرد دارای سرماخوردگی، به احتمال ۳۰٪ سرفه می‌کند. اگر کسی سرفه کند، احتمال اینکه سرما خورده باشد، چقدر است؟

---

۱ – inference by enumeration

۲ – آزمون میان‌ترم درس «هوش مصنوعی» استاد، «کتی مک‌کون»، دانشکده‌ی علوم کامپیوتر دانشگاه کلمبیای کشور آمریکا، ۲۶ اکتبر سال ۲۰۰۶ میلادی

۱) ۰,۳ / (۰,۲ × ۰,۱)

۲) ۰,۲ / (۰,۳ × ۰,۱)

۳) ۰,۱ / (۰,۳ × ۰,۲)

۴) (۰,۳ × ۰,۲) / ۰,۱

۵) هیچکدام

جواب- گزینه‌ی «۲» درست است.[۱]

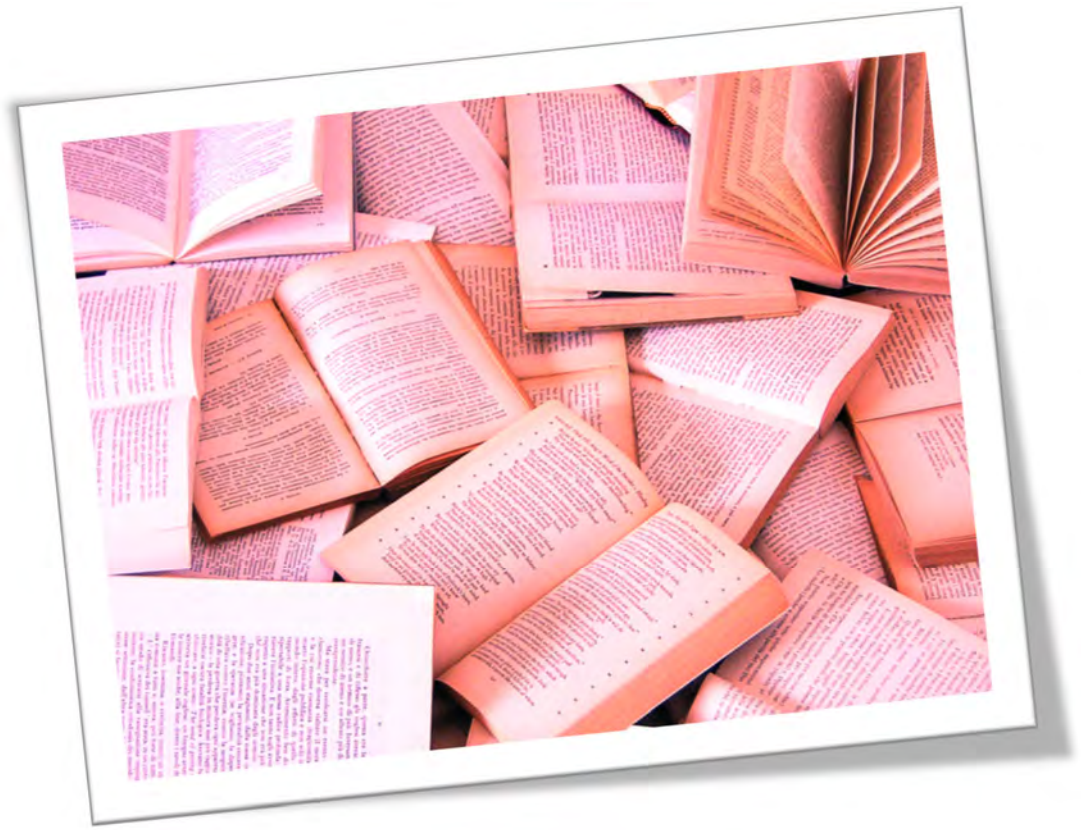
۱ - آزمون میان‌ترم درس «هوش مصنوعی» استاد، «روت شولتز»، دانشکده‌ی مهندسی برق و فنّآوری اطلاعات دانشگاه کوئینزلند کشور استرالیا، سال ۲۰۱۲ میلادی

فصل چهاردهم

شبکه‌های بِیْزی[۲]

۱ - تصویر، مربوط به توماس بِیْز(Thomas Bayes)، ۱۷۶۱ – ۱۷۰۱ میلادی، ریّاضیدان و کشیش انگلیسی می‌باشد. شهرت وی به خاطر قضیّه‌ی بیز است که پس از مرگش پخش شده است.
(http://en.wikipedia.org/wiki/Thomas_Bayes)

۲ - Bayesian networks



فهرست برخی از عنوان‌های نوشته‌ها

استنتاج احتمالی

در گذشته در مورد سیستم‌های برمبنای قانونی که می‌توانند استنتاج‌های منطقی را انجام دهند، صحبت کردیم. مثلاً اگر جَک گرسنه شود، به بازار می‌رود:

$$Hungry(Jack) \Rightarrow GoesTo(Jack,Mart^{1})$$

تمایل داریم این نوع از عملکرد را به جهان‌هایی با نامعلومی توسعه دهیم. مثلاً:

$$P(Hungry(Jack))=0.98$$
$$P(GoesTo(Jack,Mart)|Hungry(Jack))=0.5$$
$$P(GoesTo(Jack,Mart))=?$$

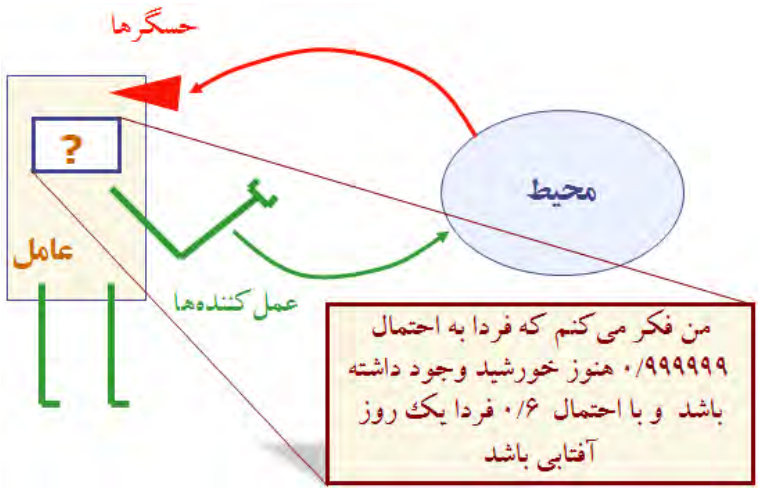
در کل می‌توانیم از قانون بیز برای این کار استفاده نماییم، *در این مورد مشکل در کار کردن با توزیع پیوسته‌ی احتمال می‌باشد، چون دارای جدولی بزرگ، به صورت نمایی می‌باشد.*

ما به ساختمان داده‌ای که در آن تعدادی متغّیر بر هم تأثیر نمی‌گذارند، نیازمندیم. به عنوان مثال، رنگ کلاه بارت بر اینکه جک گرسنه می‌باشد، تأثیری نمی‌گذارد؛ *این ساختار را یک شبکه‌ی بیزی می‌نامیم.*

---

۱ – در لغت یعنی: «بازار»

عامل احتمالی



مسأله

در زمان مشخّص t، پایگاه دانش(KB) یک عامل، مجموعه‌ای از باور(حدس)[۱]ها(موردها) است؛ در زمان t، حسگرهای عامل، یک دید را به وجود می‌آورند که احتمال یکی از حدس‌ها را افزایش می‌دهد؛ *حال سؤال این است که در این موقع، عامل باید چگونه احتمال حدس‌های(موردهای) دیگر را به‌روز نماید؟*.

هدف شبکه‌های بیزی

شرح یک مجموعه از حدس‌ها را با به وجود آوردن ارتباط‌های صریح، میان حدس‌ها و پیدا کردن استقلال شرطی میان حدس‌ها آسان می‌کنند و یک روش مناسب‌تر از استفاده از جدول‌های توزیع پیوسته را برای به‌روز کردن قدرت حدس‌ها، در زمانی که مدرک(شاهد یا گواه)[۲] جدید مشاهده می‌شود، به وجود می‌آورند.

نام‌های دیگر شبکه‌های بیزی

نام‌های دیگر شبکه‌های بیزی عبارتند از: شبکه‌های باور(حدسی)[۳]، شبکه‌های احتمالی[۴] و شبکه‌های بیان کننده‌ی سببی[۱].

۱- belief
۲- evidence
۳- belief networks
۴- probabilistic networks

شبکه‌های بیزی

مطلب مهم:

تعریف– یک روش ساده و گرافیکی برای وضعیّت‌های مستقل شرطی در یک نمایش فشرده برای توزیع پیوسته‌ی کامل است. به عبارت دیگر، یک شبکه‌ی بیزی، یک گراف مستقیم است که در آن هر گره، با اطّلاعات احتمالی تفسیر می‌شود. یک شبکه[ی بیزی] دارای موردهای زیر است:

- یک مجموعه از متغیّرهای تصادفی، که همان گره‌های درون شبکه هستند.
- یک مجموعه از یال‌ها(لبه‌ها یا پیکان‌ها)، که گره‌ها را متّصل می‌کنند. یال‌ها، *تأثیرها* را نمایش می‌دهند. اگر پیکانی از گره‌ی X به طرف گره‌ی Y وجود داشته باشد، آنگاه گره‌ی X، پدر گره‌ی Y می‌باشد.

در این شبکه(شبکه‌ی بیزی)، هیچ حلقه‌ای(دُوْری) وجود ندارد؛ به عبارت دیگر، این شبکه به صورت یک گراف مستقیم بدون حلقه(دُوْر) می‌باشد.

ساختار شبکه‌ی بیزی: همان طور که گفتیم، در این ساختار، هر متغیّر، دارای یک گره است و شکل آن به صورت یک گراف بدون دُوْر(غیرحلقه‌ای) مستقیم می‌باشد؛

مطلب مهم:

در ضمن، یک توزیع شرطی، برای هر گره‌ی ارائه شده به وسیله‌ی والدهای خودش، به صورت $P(X_i|Parents(X_i))$ می‌باشد. در ساده‌ترین حالت، توزیع شرطی ارائه شده، به صورت یک *جدول احتمال شرطی*[۲]، توزیع را برمبنای X_i، برای هر ترکیب از مقادیر پدر(والد) ارائه می‌دهد؛ به بیان دیگر، هر گره، دارای یک جدول احتمال شرطی(CPT) است که این جدول، احتمال هر مقدار آن گره و مقادیر والدهای آن را ارائه می‌کند.

مثال دزدی[۳]– من سرکار هستم و همسایه‌ام، جان، با من تماس می‌گیرد و می‌گوید: «آژیر[۴] خانه‌ی من فعّال است.»، امّا همسایه‌ی دیگرم، مِری[۵]، این کار را انجام نمی‌دهد؛ بعضی از وقت‌ها آژیر با کم‌ترین لرزه‌ای فعّال می‌شود؛ آیا در خانه‌ی من واقعاً دزد است؟

متغیّرها، در این مثال عبارتند از: دزد، لرزه (ی زمین)[۱]، آژیر، تماس جان[۲]، تماس مری[۳].

---

۱ – causal networks

۲ – **conditional probability table(*CPT*)**

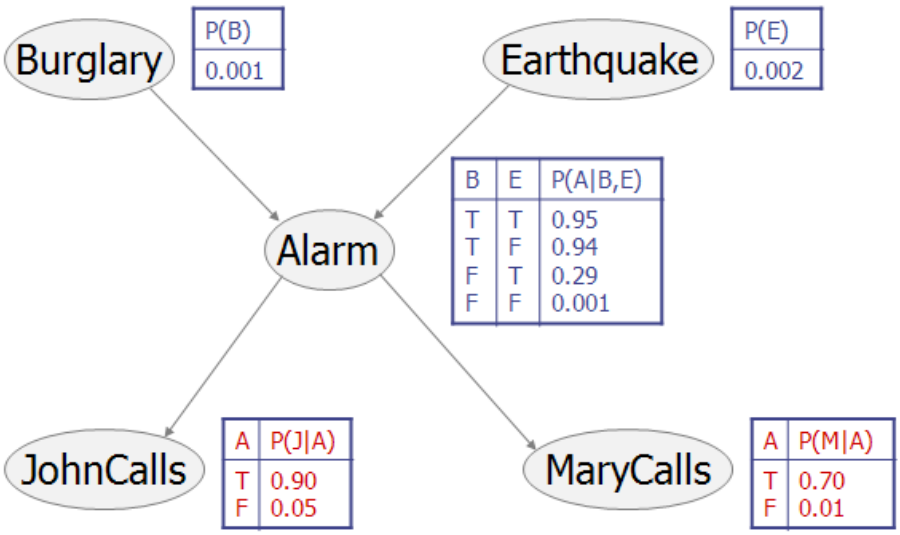
۳ – **burglar**، که به طور خلاصه، با حرف ***B*** نشان داده می‌شود.

۴ – **alarm**، که به طور خلاصه، با حرف ***A*** نشان داده می‌شود.

۵ – Mary

به موردهایی مثل موردهای زیر هم توجّه نمایید:

- دزد می‌تواند آژیر را در وضعیّت خاموش قرار دهد.
- لرزه‌ی زمین می‌تواند آژیر را خاموش نماید.
- آژیر می‌تواند سبب تماس مری شود.
- آژیر می‌تواند سبب تماس جان شود.



تراکم(به هم فشردگی)[۴]

شبکه‌های بیزی، ارائه‌ی شبکه‌های بزرگ را امکان‌پذیر می‌سازند؛ در این شبکه‌ها، اطّلاعات اضافی[۵] برداشته می‌شود.

یک جدول احتمال شرطی(CPT)، برای X_i بولیّن، با والدهای بولیّن k، دارای 2^k سطر برای ترکیب مقدارهای والد می‌باشد. هر سطر، یک عدد p را برای X_i = true نیاز دارد (عدد مورد نیاز برای X_i برابر با false، فقط 1-p می‌باشد). اگر n متغیّر داشته باشیم و در صورتی که هر متغیّر دارای بیش از k والد نباشد، شبکه‌ی کامل به تعداد بیش‌تر از $n.2^k$ سطر نیاز ندارد. توجّه کنید که [تراکم،] به صورت خطّی با n رشد می‌کند و دارای مرتبه‌ی $O(2^n)$ برای توزیع پیوسته‌ی کامل می‌باشد.

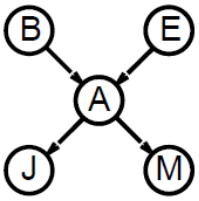
---

۱ – **earthquake، که به طور خلاصه، با حرف *E* نشان داده می‌شود.**
۲ – **JohnCalls، که به طور خلاصه، با حرف *J* نشان داده می‌شود.**
۳ – **MaryCalls، که به طور خلاصه، با حرف *M* نشان داده می‌شود.**
۴ – compactness
۵ – redundant

اگر X_n, ... ,X_1، همه‌ی متغیّرهای تصادفی باشند، در این صورت، با استفاده از قانون زنجیری و استقلال شرطی داریم:

$$P(x_1,..., x_n) = \prod_{i=1}^{n} P(x_i \mid parents\ (X_i))$$

مثال(با توجّه به مثال دزدی):



$$P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) = P(j \mid a)P(m \mid a)P(a \mid \neg b, \neg e)P(\neg b)P(\neg e)$$
$$= 0.9 \times 0.7 \times 0.001 \times 0.999 \times 0.998 \approx 0.00063$$

بیان نامعلومی(عدم‌قطعیّت) به طور خلاصه

مطلب مهم:

[در مثال دزدی،] توجّه نمایید که ما نیازی به داشتن گره‌هایی برای همه‌ی موردهایی که مری ممکن است تماس نگیرد، نداریم. یک روش احتمالی به ما اجازه می‌دهد که این اطّلاعات را به صورت ¬M خلاصه نماییم. این روش، به یک عامل ساده اجازه می‌دهد که به جهان‌های بزرگی که دارای تعداد زیادی از نتایج نامعلوم ممکن هستند، رسیدگی کند.

چکیده‌ی مطلب‌های فصل چهاردهم

توزیع پیوسته‌ی احتمال برای عامل‌هایی که محیط آنها دارای نامعلومی است، خیلی بزرگ و به صورت نمایی می‌باشد؛ در این مورد ما به ساختمان داده‌ای که در آن تعدادی متغیّر بر هم تأثیر نمی‌گذارند، نیازمندیم، که این ساختار را یک شبکه‌ی بیزی می‌نامیم.

شبکه‌های بیزی، ارائه‌ی شبکه‌های بزرگ را امکان‌پذیر می‌سازند؛ در این شبکه‌ها، اطّلاعات اضافی برداشته می‌شود.

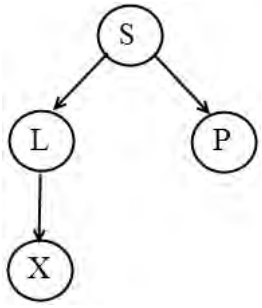
شبکه‌های بیزی، شرح یک مجموعه از حدس‌ها را با به وجود آوردن ارتباط‌های صریح، میان حدس‌ها و پیدا کردن استقلال شرطی میان حدس‌ها آسان می‌کنند و یک روش مناسب‌تر از استفاده از جدول‌های توزیع پیوسته را برای به‌روز کردن قدرت حدس‌ها، در زمانی که مدرک جدید مشاهده می‌شود، به وجود می‌آورند.

شبکه‌های بیزی، از یک مجموعه از متغیّرهای تصادفی، که همان گره‌های درون شبکه هستند و یک مجموعه از یال‌ها(لبه‌ها یا پیکان‌ها)، که گره‌ها را متّصل می‌کنند و *تأثیرها* را نمایش می‌دهند، درست شده‌اند.

یادآوری یا تکمیل مطلب‌های فصل چهاردهم

سؤال- ☑مشابه کنکور سراسری مکاترونیک سال ۸۷- یک شبکه‌ی بیزی، با استفاده از چهار متغیّر تصادفی بولیّن سیگاری[۱](S)، سرطان ریّه[۲](L)، بُنیّه‌ی ضعیف[۳](P) و اشعه‌ی X[۴](X)، با توجّه به این عبارت‌ها، رسم کنید: احتمال بیش‌تری است که سیگاری‌ها سرطان ریّه و همچنین بنیّه‌ی ضعیف داشته باشند؛ و در صورتی که سرطان ریّه وجود داشته باشد، در عکسی که با اشعه‌ی X گرفته می‌شود، ممکن است نقطه‌ای سیاه رنگ وجود داشته باشد و در نتیجه متغیّر X درست باشد.[۵]

جواب-



۱ - smoker

۲ - lung-cancer

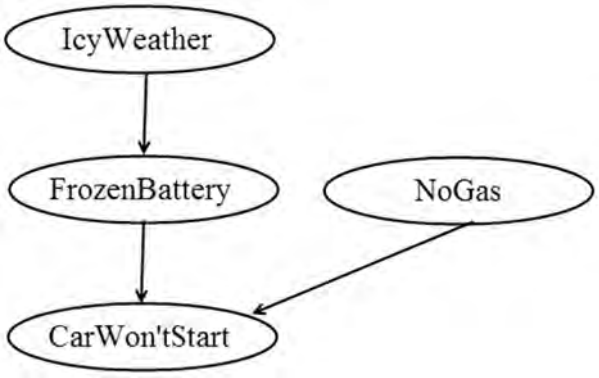
۳ - poor-stamina

۴ - X-ray

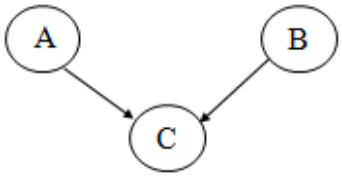
۵ - آزمون درس «مبانی هوش مصنوعی» استاد، دکتر، «تارک حلمی»، دانشکده‌ی علوم کامپیوتر و اطّلاعات دانشگاه پادشاه فهد کشور عربستان سعودی

سؤال- ☑مشابه کنکور سراسری مکاترونیک سال ۸۷- مناسب‌ترین شبکه‌ی بیزی را برای استدلال در مورد مشکلات اتومبیل، با استفاده از چهار متغیّر تصادفی بولیّن FrozenBattery(باتری یخ زده)، IcyWeather(هوای یخبندان)، CarWon'tStart(اتومبیل روشن نخواهد شد) و NoGas(تمام شدن بنزین) رسم کنید.[۱]

جواب-



سؤال- در شبکه‌ی بیزی زیر P(A,B,C) چیست؟



جواب-

$$P(A,B,C) = P(C|A,B)\ P(A|B)\ P(B)$$

با توجّه به اینکه A و B از هم مستقل هستند، $P(A|B) = P(A)$، در نتیجه داریم:[۲]

$$P(A,B,C) = P(C|A,B)\ P(A)\ P(B)$$

سؤال- در شبکه‌ی بیزی زیر P(A,B,C) چیست؟

جواب-[۳]

$$P(A,B,C) = P(A)\ P(B)\ P(C)$$

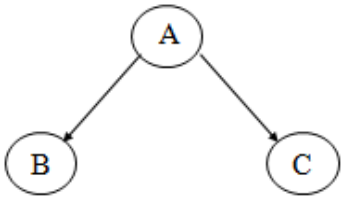
سؤال- در شبکه‌ی بیزی زیر P(A,B,C) چیست؟

---

۱ - آزمون درس «مبانی هوش مصنوعی» استاد، دکتر، «تارک حلمی»، دانشکده‌ی علوم کامپیوتر و اطّلاعات دانشگاه پادشاه فهد کشور عربستان سعودی

۲ - مطلب‌های درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا

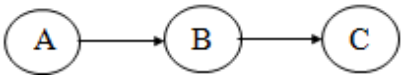
۳ - مطلب‌های درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا



جواب-[۱]

$$P(A,B,C) = P(B|A)\ P(C|A)\ P(A)$$

سؤال- در شبکه‌ی بیزی زیر $P(A,B,C)$ چیست؟



جواب-[۲]

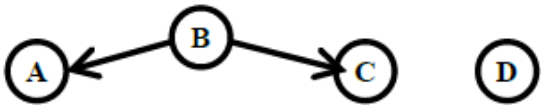
$$P(A,B,C) = P(C|B)\ P(B|A)\ P(A)$$

سؤال- شبکه‌های بیزی مناسب معادله‌های احتمال شرطی زیر را رسم کنید.[۳]

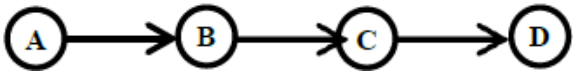
الف) $P(B|A,C)\ P(A)\ P(C|D)\ P(D)$

ب) $P(A)\ P(B)\ P(C)\ P(D)$

پ) $P(A|B)\ P(C|B)\ P(B)\ P(D)$



ت) $P(D|C)\ P(C|B)\ P(B|A)\ P(A)$



ث) $P(B|A)\ P(A)\ P(C|D)\ P(D)$

---

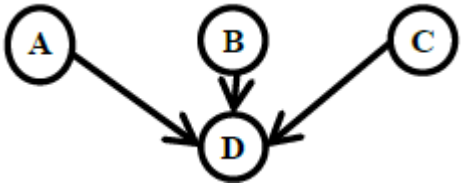
۱ - مطلب‌های درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا

۲ - مطلب‌های درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا

۳ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، زمستان سال ۲۰۱۲ میلادی

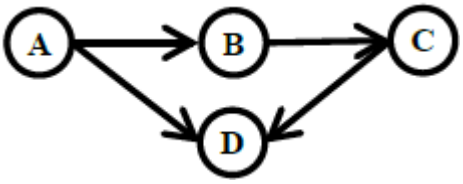
سؤال- ☑مشابه کنکور سراسری فنّـآوری اطّلاعات سال ۹۲- معادله‌های احتمال شرطی مناسب شبکه‌های بیزی زیر را بنویسید.[۱]

الف)



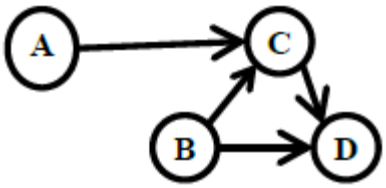
P(D|A,B,C) P(A) P(B) P(C)

ب)



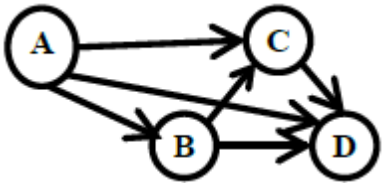
P(D|A,C) P(C|B) P(B|A) P(A)

پ)



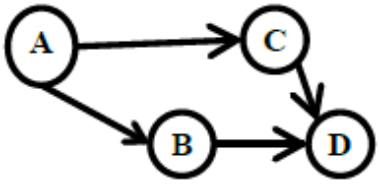
P(D|B,C) P(C|A,B) P(B) P(A)

ت)



P(D|A,B,C) P(C|A,B) P(B|A) P(A)

ث)



P(D|B,C) P(C|A) P(B|A) P(A)

---

۱ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، زمستان سال ۲۰۱۲ میلادی

سؤال- به شبکه‌ی بیزی $A \rightarrow B \leftarrow C$ با سه متغیّر تصادفی بولیّن و CPT آنها که به صورت زیر تعریف شده‌اند، توجّه نمائید:

$P(A) = 0.2$
$P(B \mid A, C) = 0.25$
$P(B \mid A, \neg C) = 0.5$
$P(B \mid \neg A, C) = 0.3$
$P(B \mid \neg A, \neg C) = 0.8$
$P(C) = 0.55$

الف) $P(\neg A, B, C)$ را محاسبه کنید.

جواب-

$P(\neg A, B, C) = P(B \mid \neg A, C)\, P(\neg A)\, P(C) = 0.3 \times 0.8 \times 0.55 = 0.132$

ب) $P(\neg A \mid \neg C)$ را محاسبه کنید.

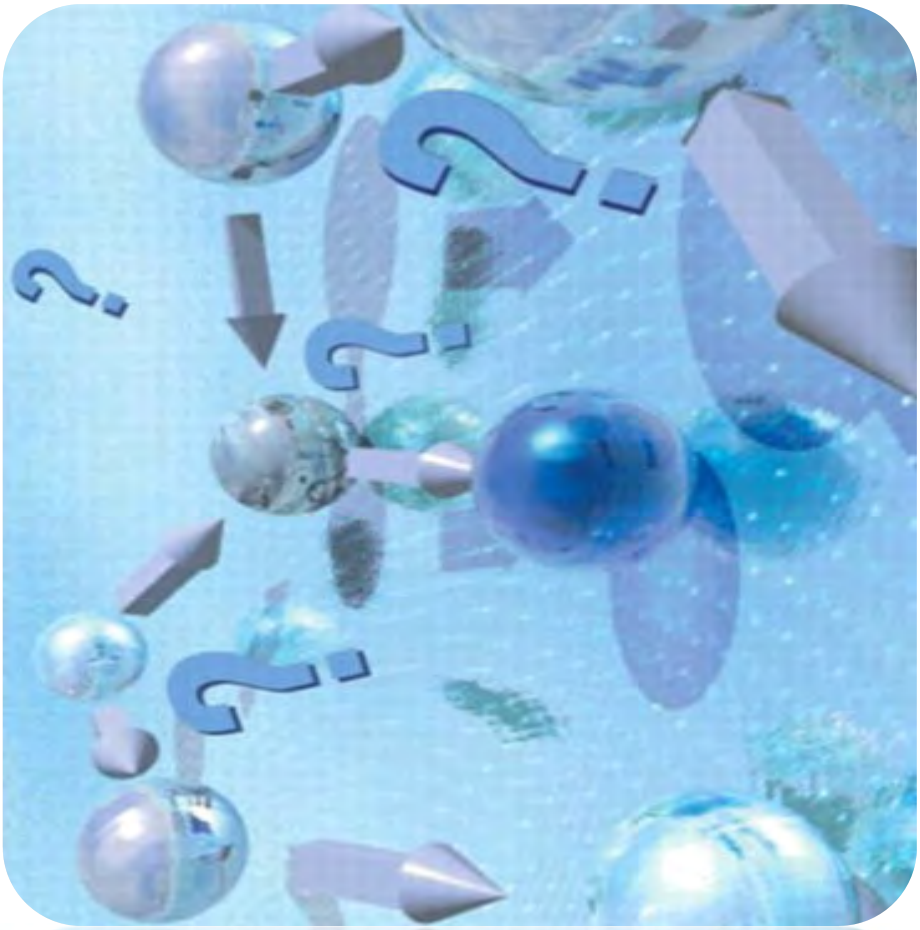
جواب- با توجّه به اینکه A و C از هم مستقل هستند، داریم:

$P(\neg A \mid \neg C) = P(\neg A)$[1]

---

۱ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «چاک دایر»، دانشگاه ویسکانسین-مَدیسُون کشور آمریکا، ۲۰ دسامبر سال ۲۰۱۰ میلادی

فصل پانزدهم



استنتاج در شبکه‌های بیزی

فهرست برخی از عنوان‌های نوشته‌ها

E را متغّیرهای مدرکی که دیده شده‌اند، قرار دهید؛ مثلاً در مثال دزدی، که در فصل شبکه‌های بیزی با آن آشنا شدیم، $\{JohnCalls,MaryCalls\}$ را متغّیرهای مدرک قرار دهید. X را متغّیر پرس‌وجو قرار دهید؛ به عنوان مثال، Burglary و فرض کنید می‌خواهیم $P(X|E)$ را به دست آوریم.

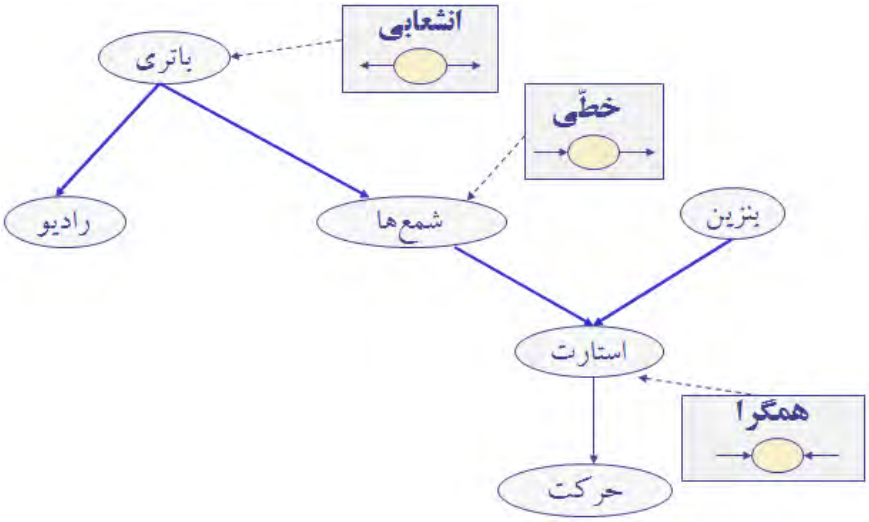
انواع گره‌های موجود در یک مسیر

با داشتن یک مجموعه‌ی E از گره‌های مدرک، دو گره که به وسیله‌ی یک مسیر غیرمستقیم، به هم وصل شده‌اند، در صورتی *مستقل* هستند که دارای یکی از سه شرط زیر باشند:

۱- یک گره‌ی موجود در مسیر، به صورت خطّی[۱] و در E باشد.
۲- یک گره‌ی موجود در مسیر، به صورت انشعابی[۲] و در E باشد.
۳- یک گره‌ی موجود در مسیر، به صورت همگرا باشد و نه این گره و نه هیچ یک از فرزندانش در E باشند.

مثال خودرو؛ مسیر زیر را برای خودرو در نظر بگیرید:

۱- linear
۲- diverging



با توجّه به شکل بالا، *بنزین و رادیو، با توجّه به مدرک شمع‌ها[1]، مستقل هستند. بنزین و رادیو، با توجّه به مدرک باتری، مستقل هستند. امّا بنزین و رادیو، با وجود یکی از مدرک‌های استارت یا حرکت، مستقل نمی‌باشند.*

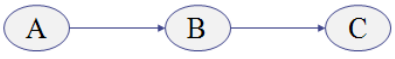
استنتاج در شبکه‌های بیزی

در ساده‌ترین حالت داریم:

$$P(B) = P(a)P(B|a) + P(\sim a)P(B|\sim a)$$

$$\Rightarrow P(B) = \sum_{A} P(A)P(B \mid A)$$

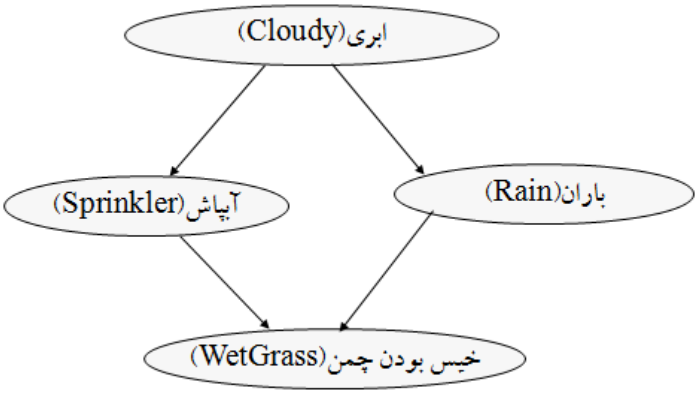
حال سؤال اینجاست که اگر شکل زیر را داشته باشیم، آنگاه P(C) برابر چیست؟.



اگر یک زنجیره به صورت زیر داشته باشیم، پیچیدگی زمانی برای محاسبه‌ی P(Xn) چقدر است؟.

۱- Spark plugs

مثال- با داشتن شکل زیر احتمال خیس بودن چمن را به دست آورید.



$$P(W)=\sum_{R,S,C}P(w|r,s)P(r|c)P(s|c)P(c)$$

مطلب مهم:

در مثال بالا راه حل در حال محاسبه‌ی احتمال‌ها به صورت تکی نمی‌باشد، بلکه در حال محاسبه‌ی تمام موردها است. در زیر ایده‌ای برای جلوگیری از انفجار نمایی محاسبه‌ها بیان شده است، *که به آن، **روش حذف متغیّر** هم گفته می‌شود:*

۱- با توجّه به ساختار شبکه‌ی بیزی، برخی از زیر عبارت‌ها، در محل‌های اتّصال، فقط به تعداد کمی از متغیّرها وابسته‌اند.

۲- با یکبار محاسبه‌ی آنها و نگهداری نتایج می‌توانیم از به وجود آمدن آنها به صورت نمایی، در بسیاری از موردها جلوگیری نماییم:

$$=\sum_{R,S}P(w|r,s)\sum_{C}P(r|c)P(s|c)P(c)$$

اگر $\sum_{C}P(r|c)P(s|c)P(c)$ را $f_C(R,S)$ در نظر بگیریم، داریم:

$$P(W)=\sum_{R,S}P(w|r,s)\,f_C(r,s)$$

روش حذف متغیّر[۱]

ایده‌ی اصلی- پرس‌وجو را به صورت زیر بنویسید:

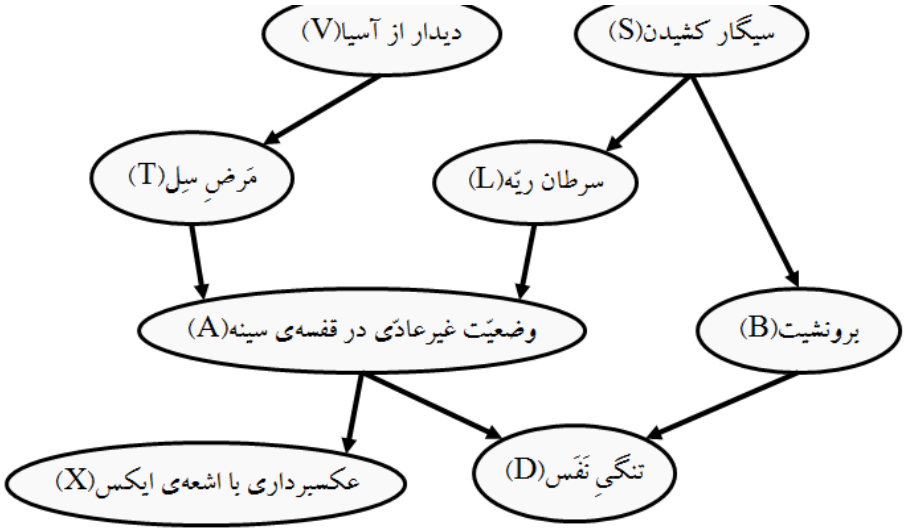
$$P(X_n, \boldsymbol{e}) = \sum_{x_k} \cdots \sum_{x_3} \sum_{x_2} \prod_i P(x_i \mid pa_i)$$

مطلب مهم:

برای این کار، به صورت تکراری کارهای زیر را انجام دهید:

۱- تمام عبارت‌های نامربوط را به خارج از داخلی‌ترین جمع انتقال دهید.
۲- داخلی‌ترین جمع را انجام دهید و عبارت جدید را به دست آورید.
۳- عبارت جدید را در ضرب وارد نمایید.

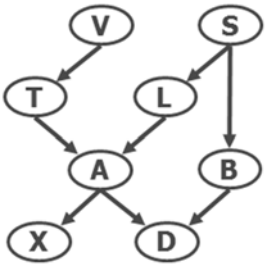
یک مثال پیچیده‌تر برای روش حذف متغیّر(دیدار از آسیا)



شکل بالا- شکل مثال دیدار از آسیا؛ در پاورقی، بیماری‌های سِل[۱] و بُرونشیت[۲] توضیح داده شده است.

۱- variable elimination

شکل بالا را به صورت زیر در نظر می‌گیریم:



می‌خواهیم احتمال تنگی نفس(P(D)) را محاسبه نماییم؛ باید v, s, x, t, l, a, b را حذف نماییم.

در ابتدا– عامل‌های اوّلیّه عبارتند از:

$$P(v)P(s)P(t|v)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

گام نخست– برای حذف v، به جای احتمال‌هایی که دارای متغیّر v هستند، $f_v(t)$ را قرار می‌دهیم؛ پس فرض می‌کنیم:

$$f_v(t) = \sum_v P(v)P(t|v)$$

در نتیجه داریم:

$$f_v(t)P(s)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

نکته:

توجّه کنید که $f_v(t) = P(t)$؛ پس در حالت کلّی، نتیجه‌ی حذف، لزوماً یک عبارت احتمالی نمی‌باشد.

گام بعدی– برای حذف s، فرض می‌کنیم:

$$f_s(b,l) = \sum_s P(s)P(b|s)P(l|s)$$

در نتیجه داریم:

$$f_v(t)\, f_s(b,l)P(a|t,l)P(x|a)P(d|a,b)$$

---

۱ – سِل(tuberculosis): نوعی بیماری واگیردار(مُسْری) ریّه(شُش یا جگر سفید) است. (Babylon> Babylon English - English)

۲ – بُرونشیت: به التهاب یک یا بیش‌تر نایچه[ی شُش]، بیماری برونشیت(Bronchitis) گفته می‌شود. (Babylon> Babylon English - English)

نکته:

همان طور که در $f_s(b,l)$ می‌بینید، در حالت کلّی، نتیجه‌ی حذف ممکن است یک تابع دارای چند متغیّر باشد.

گام بعدی- برای حذف x، فرض می‌کنیم:

$$f_x(a) = \sum_x P(x \mid a)$$

در نتیجه داریم:

$$f_v(t)\, f_s(b,l)\, f_x(a) P(a \mid t,l) P(d \mid a,b)$$

نکته:

$f_x(a)$، برای تمام مقدارهای a، برابر با یک است.

گام بعدی- برای حذف t، فرض می‌کنیم:

$$f_t(a,l) = \sum_t f_v(t) P(a \mid t,l)$$

در نتیجه داریم:

$$f_s(b,l)\, f_x(a)\, f_t(a,l) P(d \mid a,b)$$

گام بعدی- برای حذف l(الْ)، فرض می‌کنیم:

$$f_l(a,b) = \sum_l f_s(b,l)\, f_t(a,l)$$

در نتیجه داریم:

$$f_l(a,b)\, f_x(a) P(d \mid a,b)$$

گام بعدی- برای حذف a، فرض می‌کنیم (توجّه کنید که نماد $\sum$، در عبارت زیر دارای اندیس پایین a است.):

$$f_a(b,d) = \sum_a f_l(a,b) f_x(a) p(d \mid a,b)$$

در نتیجه داریم:

$f_a(b,d)$

گام بعدی- برای حذف b، فرض می‌کنیم (توجّه کنید که نماد $\sum$، در عبارت زیر دارای اندیس پایین b است.):

$f_b(d) = \sum_b f_a(b, d)$

در نتیجه داریم:

$f_b(d)$

حالا فهمیدیم که حذف متغیّر، به صورت یک رشته از عملیّات *بازنویسی* می‌باشد، محاسبه‌ی واقعی، در مرحله‌ی حذف انجام می‌شود و محاسبه، وابسته به ترتیب حذف می‌باشد.

روش‌های استنتاج

به دو دسته‌ی استنتاج دقیق[۱] و استنتاج تقریبی[۲] تقسیم می‌شوند. روش‌های استنتاج در زنجیره‌های ساده وحذف متغیّر، جزء روش‌های استنتاج دقیق هستند و روش‌های شبیه‌سازی احتمالی یا تصادفی[۳] یا روش‌های نمونه‌برداری[۴]، جزء روش‌های استنتاج تقریبی می‌باشند.

۱- Exact inference
۲- Approximate inference
۳- Stochastic simulation
۴- sampling methods

چکیده‌ی مطلب‌های فصل پانزدهم

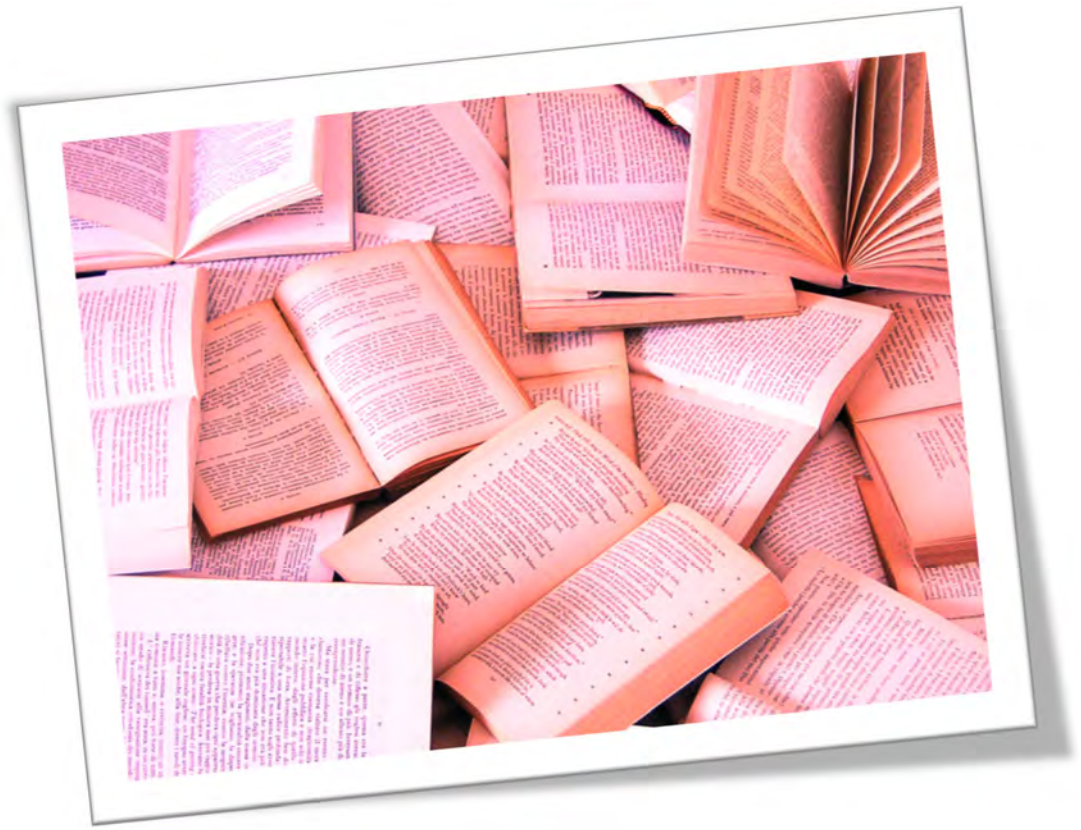
روش حذف متغیّر، که جزء روش‌های استنتاج دقیق است، برای جلوگیری از انفجار نمایی محاسبه‌ها، در توزیع پیوسته می‌باشد.

در روش حذف متغیّر، محاسبه‌ها به جای اینکه همه با هم انجام شوند، به صورت محلّی انجام می‌شوند.

فصل شانزدهم

شناخت سُخَن یا سخن‌شناسی[۱]

۱- Speech recognition



فهرست برخی از عنوان‌های نوشته‌ها

سخن‌شناسی

تعریف نخست: عملیّات لازم برای توانمندسازی یک کامپیوتر برای شناسایی و واکنش دادن به صداهای به وجود آمده در سخن انسان می‌باشد[۱].

تعریف دوّم: سخن‌شناسی یا تشخیص صدا[۲]، توانایی سیستم‌های کامپیوتری برای دریافت سخن، به صورت ورودی، و پردازش بر روی آن، یا بیان آن به صورت نوشته می‌باشد.[۳]

کاربردهای عملی سخن‌شناسی، شامل سیستم‌های پرس‌وجوکننده از پایگاه داده[۴] و سیستم‌های بازیابی اطّلاعات[۵] می‌باشد. سخن‌شناسی، دارای کاربرد در روبوتیک و مخصوصاً توسعه‌ی روبوت‌هایی که می‌توانند «بشنوند»، می‌باشد.[۶]

سخن به صورت استدلال احتمالی[۷]

سیگنال‌های سخن، پارازیت‌دار(اغتشاش‌دار)[۱]، متغیّر و مبهم[۲] می‌باشند؛ [در این صورت،] شبیه‌ترین ترتیب کلمات و سیگنال سخن ارائه شده چیست؟، برای این کار از قانون بیز استفاده نمایید:

۱- Babylon> Concise Oxford English Dictionary
۲- voice recognition
۳- Babylon> Britannica Concise Encyclopedia
۴- database-query systems
۵- information retrieval systems
۶- Babylon> Britannica Concise Encyclopedia
۷- speech as probabilistic inference

$$P(Words|signal) = \alpha P(signal|Words)P(Words)$$

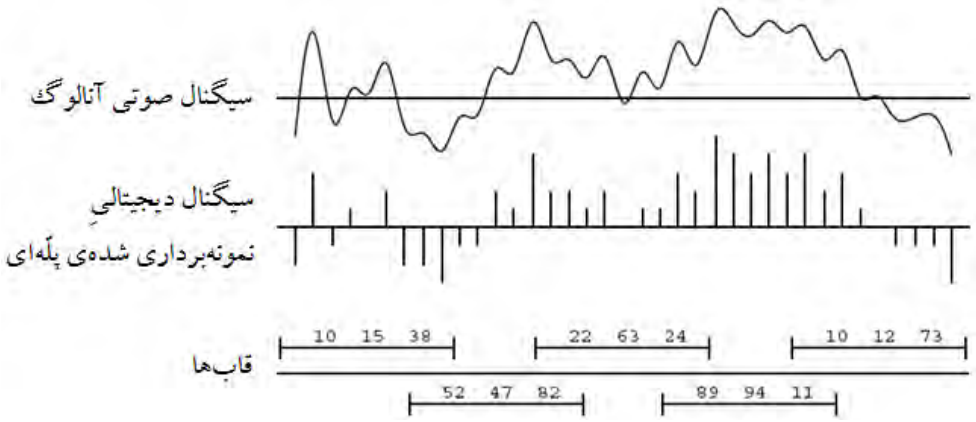
تمام سخنان انسان، ترکیبی از ۴۰ تا ۵۰ صُوْت[۳] می‌باشد. آرْپابِتْ[۴]، الفبایی برای بیان صداهای موجود در انگلیسی آمریکایی می‌باشد و به صورت زیر است:

[iy]	beat	[b]	bet	[p]	pet
[ih]	bit	[ch]	Chet	[r]	rat
[ey]	bet	[d]	debt	[s]	set
[ao]	bought	[hh]	hat	[th]	thick
[ow]	boat	[hv]	high	[dh]	that
[er]	Bert	[l]	let	[w]	wet
[ix]	roses	[ng]	sing	[en]	button
⋮	⋮	⋮	⋮	⋮	⋮

برای مثال، برای کلمه‌ی «ceiling» داریم: [s iy l en] / [s iy l ix ng] / [s iy l ih ng]

صداهای سخن[۵]– سیگنال خام میکروفون، به صورت تابعی از زمان می‌باشد؛ سیگنال‌های صوتی، که در ابتدا به صورت آنالوگ هستند، به صورت سیگنال دیجیتالی نمونه‌برداری شده‌ی پلّه‌ای در می‌آیند، در پردازش، قاب‌ها روی هم می‌افتند و همگی به وسیله‌ی پستی و بلندی‌اشان تشخیص داده می‌شوند.

۱- noisy
۲- ambiguous
۳- phone
۴- ARPAbet
۵- speech sounds



صوت‌های سه بخشی[۱]: هر صوت، دارای سه بخش می‌باشد(آغاز[۲]، وسط[۳] و پایان[۴])، به عنوان مثال، حرف [t] دارای ابتدای آرام[۵]، وسط قوی[۶] و انتهای خَشِن[۷] می‌باشد.

مدل‌های پنهان مارکوف[۸]

به طور گسترده‌ای در تعداد زیادی از سیستم‌ها مورد استفاده قرار می‌گیرند. سیستم‌های سخن‌شناسی پیشرفته، براساس مدل‌های پنهان مارکوف می‌باشند؛ مدل‌های پنهان مارکوف، مدل‌هایی آماری هستند که رشته‌ای از سمبل‌ها را به وجود می‌آورند. از مدل‌های پنهان مارکوف به این دلیل در سخن‌شناسی استفاده می‌شود که سیگنال سخن می‌تواند به صورت تکّه سیگنالی ثابت یا سیگنالی ثابت کوتاه مدّت دیده شود؛ در زمانی کوتاه، مثلاً ده میلی ثانیه، سخن می‌تواند به صورت یک پردازش ثابت، تخمین زده شود. دلیل دیگری که از این مدل‌ها استفاده می‌کنیم، این است که از نظر محاسباتی امکان‌پذیر هستند.[۹]

۱- Three-state phones
۲- Onset
۳- Mid
۴- End
۵- silent Onset
۶- explosive Mid
۷- hissing End
۸- *H*idden *M*arkov *m*odels(*HMMs*)
۹- http://en.wikipedia.org/wiki/Speech_recognition

چکیده‌ی مطلب‌های فصل شانزدهم

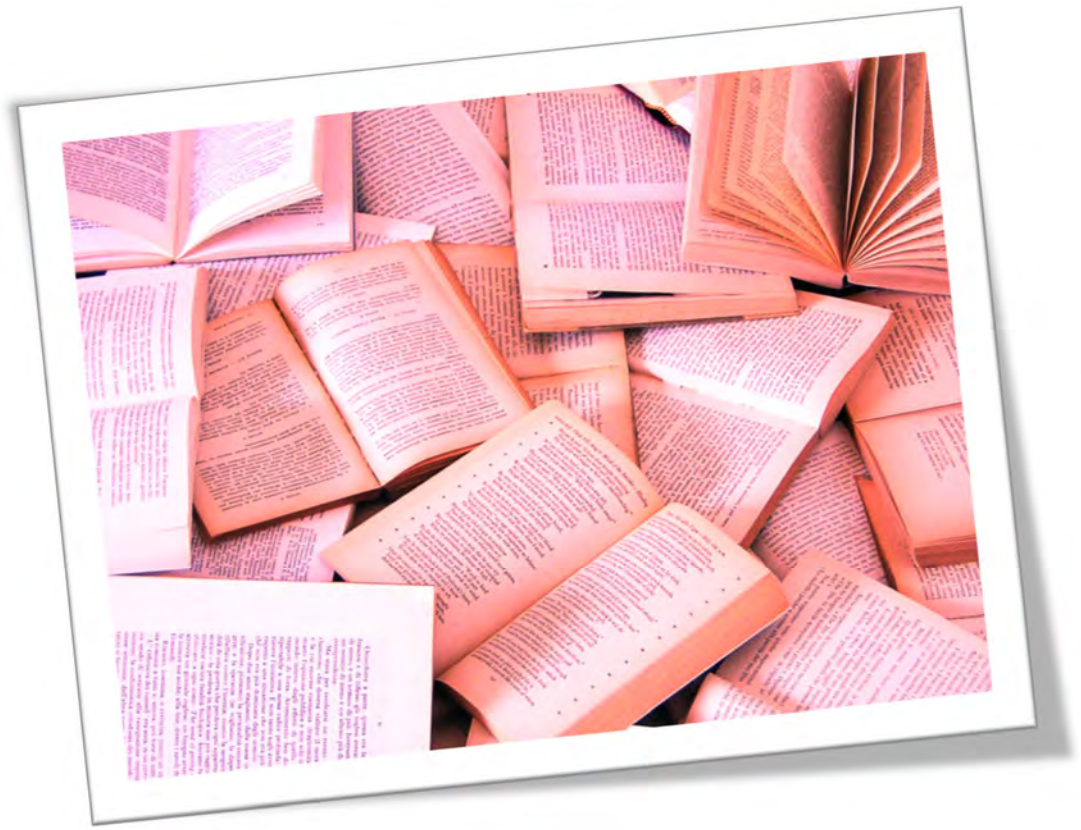
سخن‌شناسی، توانایی سیستم‌های کامپیوتری برای دریافت سخن، به صورت ورودی، و پردازش بر روی آن، یا بیان آن به صورت نوشته می‌باشد.

سیستم‌های سخن‌شناسی پیشرفته، براساس مدل‌های پنهان مارکوف می‌باشند؛ مدل‌های پنهان مارکوف، مدل‌هایی آماری هستند که رشته‌ای از سمبل‌ها را به وجود می‌آورند.

فصل هفدهم

شبکه‌های عصبی[۱]

۱ - neural networks



فهرست برخی از عنوان‌های نوشته‌ها

شبکه‌های عصبی

تعریف اوّل- سیستمی که از برنامه‌ها و پایگاه‌های داده‌ای که شامل چند پردازشگر موازی هستند، تشکیل شده است و عملیّات مغز انسان را شبیه‌سازی می‌کند.[۱]

تعریف دوّم- **مشابه کنکور آزاد مهندسی کامپیوتر سال ۸۱**- یک سیستم کامپیوتری که از روی مغز انسان و سیستم عصبی، مدلسازی شده است.[۲]

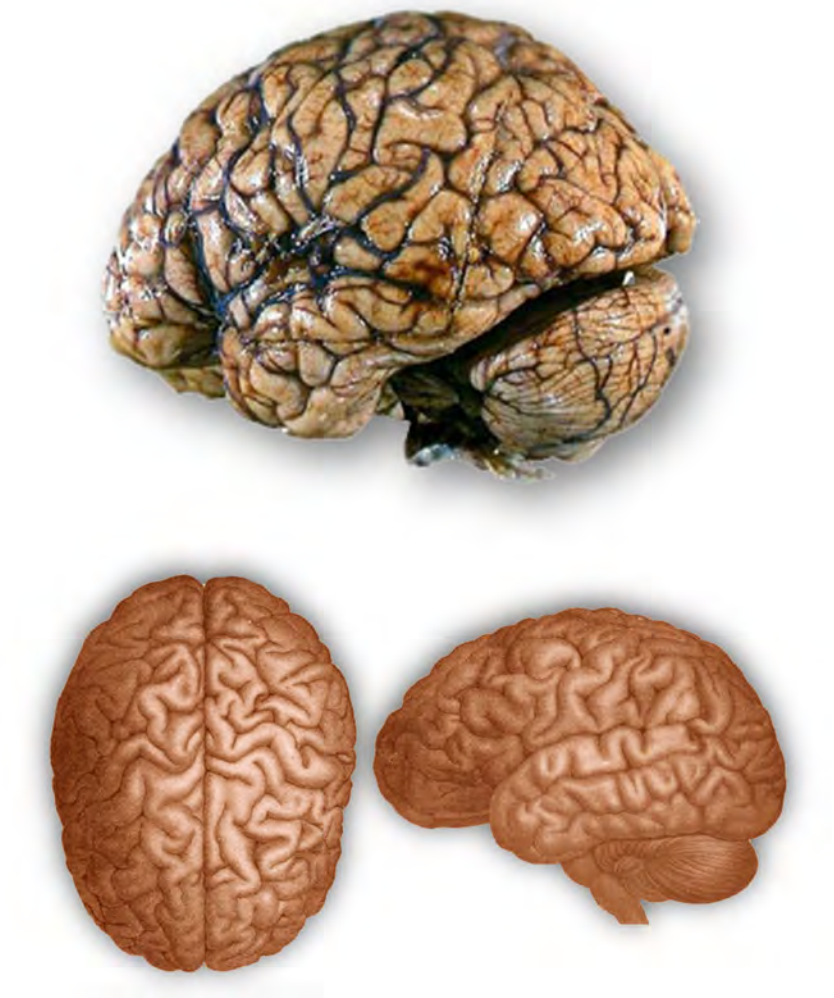
زیست‌شناسی و علم کامپیوتر

مغز انسان به نظر نمی‌رسد که دارای یک واحد پردازنده‌ی مرکزی[۳] باشد.

۱- Babylon> Babylon English-English

۲- Babylon> Concise Oxford English Dictionary

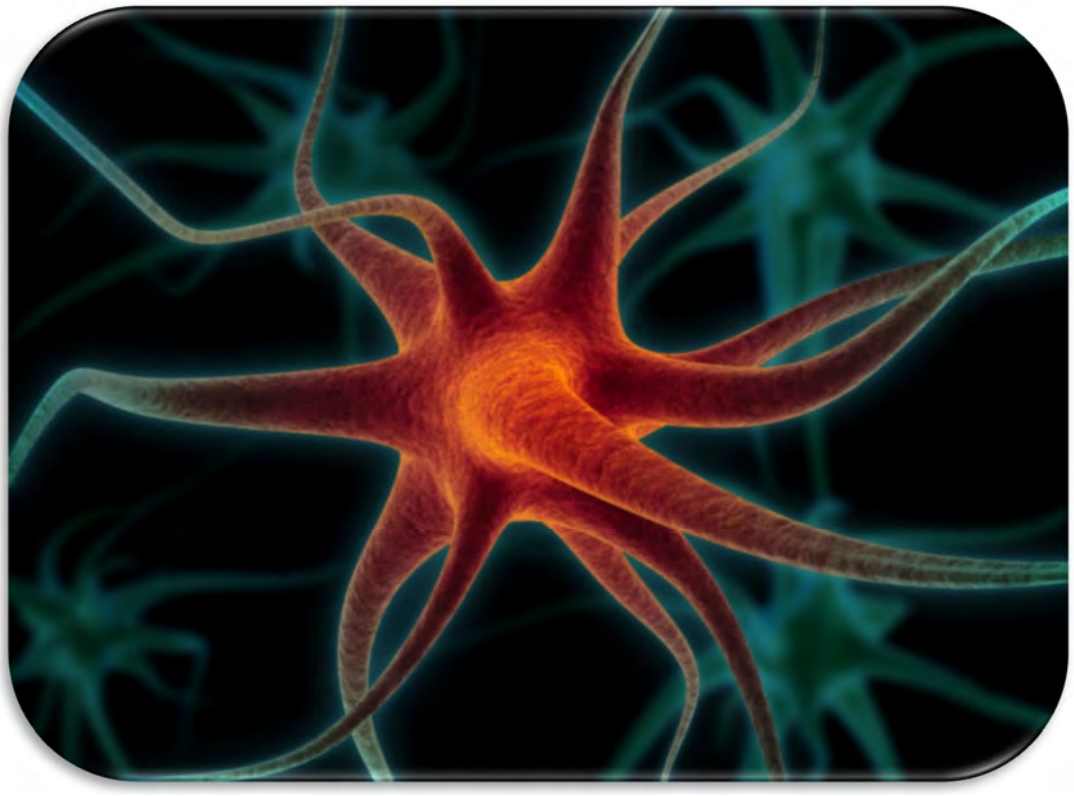
۳- *C*entral *P*rocessing *U*nit(*CPU*)



سه شکل بالا– مغز انسان

در عوض، دارای تعداد زیادی واحدهای ساده، ناهمزمان[۱] و موازی[۲] به نام نورُون[۳] است. مغزانسان دارای تقریباً 10^{11} نورون می‌باشد. نورون‌ها دارای درحدود بیست نوع هستند.

۱ – asynchronous
۲ – parallel
۳ – neuron



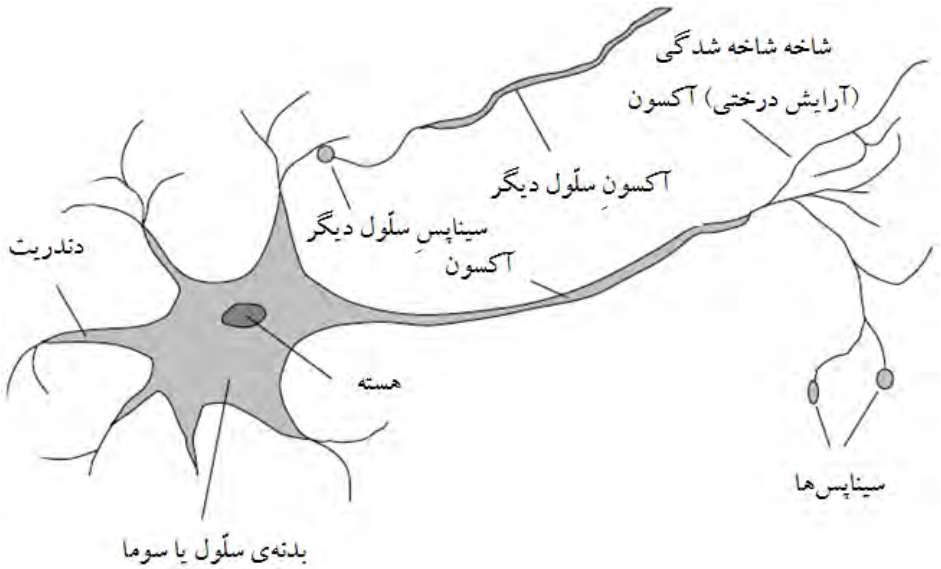
شکل بالا– نورون

هر نورون، یک سلّول تکی است که دارای تعدادی لیف(فیبر)[۱]های نسبتاً کوتاه به نام دِنْدْرِیْتْ(دَنْدْرِیْتْ یا دِنْدْرایْتْ)[۲] است. یکی از دندریت‌ها، که بزرگ‌تر از بقیّه است، آکسون[۳] نام دارد. انتهای آکسون به تعداد زیادی لیف‌های کوچک تقسیم می‌شود. هر لیف، دندریت‌ها و بدنه‌های سلّولی دیگر نورون‌ها را وصل می‌کند. اتّصال، در واقع فاصله[۴]ای کوتاه به نام سیناپْس[۵] است. آکسون‌ها، حمل‌کننده[۶] هستند و دندریت‌ها، دریافت‌کننده[۷] هستند. در حدود ۱۰[۱۴] اتّصال وجود دارد.

۱ – fiber
۲ – dendrites
۳ – axon
۴ – gap
۵ – synapse
۶ – transmitter
۷ – receiver

شکل بالا– نورون‌ها

همان طور که بیان شد، ( **کنکور آزاد مهندسی فنّـآوری اطّلاعات و شبکه‌های کامپیوتری سال ۸۴**– در نورون‌های زیست‌شناسی، سیگنال‌ها به وسیله‌ی دندریت‌ها دریافت می‌شوند و از طریق آکسون به دیگر نورون‌ها می‌رسند.) فکر و رفتار انسان‌ها از بر هم کنش هزاران نورون به وجود می‌آیند.



شکل بالا– نورون

انواع گره‌ها در شبکه‌های عصبی کامپیوتری

در شبکه‌های عصبی کامپیوتری سه نوع گره داریم:

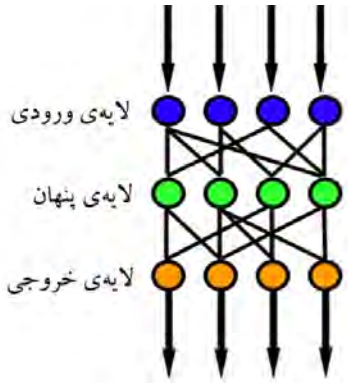
۱- گره‌های ورودی

۲- گره‌های خروجی

۳- گره‌های پنهان (که در پرسپترون‌های چندلایه‌ای وجود دارند.)

انواع شبکه‌های عصبی

۱- **شبکه‌های با تغذیه‌ی مستقیم**[۱]: در این شبکه‌ها، علامت(سیگنال)ها در یک جهت حرکت می‌کنند و بدون دُوْر(سیکل یا حلقه) می‌باشند.



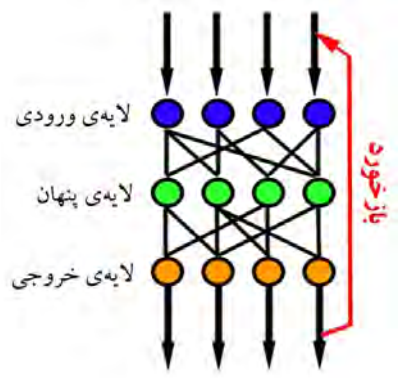
شکل بالا– نمونه‌ای از شبکه‌های با تغذیه‌ی مستقیم

۲- **شبکه‌های بازگشت کننده**[۲]: در این شبکه‌ها، در انتشار علامت(سیگنال)، دُوْر(سیکل یا حلقه) وجود دارد.

---

۱ - feed-forward networks

۲ - recurrent networks

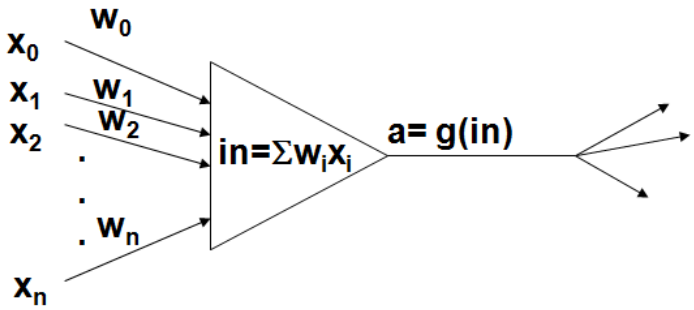


شکل بالا- نمونه‌ای از شبکه‌های بازگشت کننده

ما بیش‌تر روی شبکه‌های با تغذیه‌ی مستقیم تمرکز می‌نماییم.

پِرْسِپْتْرُونِ[1] (تک لایه‌ای)

تعریف- پرسپترون، الگوریتمی ابتدایی برای یادگیری شبکه‌های عصبی تک لایه‌ای ساده است و در دهه‌ی ۱۹۵۰ میلادی به وجود آمد.



در شکل قبل، X_iها، ورودی‌های پرسپترون هستند. W_iها، وزن‌ها هستند؛ از W_0، به عنوان شروع کننده(آستانه)[2]، با X_0 برابر با منفی یک استفاده می‌شود. in، در مرحله‌ی فعّال‌سازی، جمع ورودی‌ها و آستانه است. g، تابع فعّال‌سازی است. a، فعّال‌سازی یا خروجی است. خروجی، با استفاده از یک تابع، که تشخیص می‌دهد تا کجا سطح فعّال‌سازی پرسپترون، بالا یا پایین صفر است، محاسبه می‌شود.

پرسپترون، $a = g(in) = g(X.W)$ را محاسبه می‌نماید:

$$X.W = w_0*-1 + w_1*x_1 + w_2*x_2 + \ldots + w_n*x_n$$

g، معمولاً تابع آستانه است و داریم: اگر $z > 0$ باشد، $g(z)$، برابر با یک است و در غیر این صورت، برابر با صفر است.

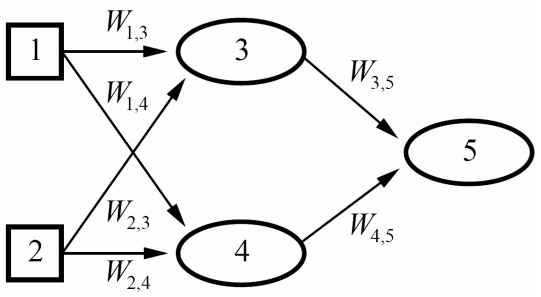
۱- perceptron

۲- threshold

نکته:

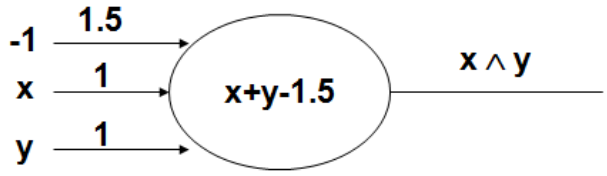
پرسپترون می‌تواند به صورت یک دروازه‌ی منطقی[۱] عمل کند؛ که *مقدار «یک»، «درست» و مقدار «صفر یا منفی یک»، «غلط»* است.

مثال تغذیه‌ی مستقیم



$$a_5 = g(W_{3,5}.a_3 + W_{4,5}.a_4)$$
$$= g(W_{3,5}.g(W_{1,3}.a_1 + W_{2,3}.a_2) + W_{4,5}.g(W_{1,4}.a_1 + W_{2,4}.a_2))$$

مثال- تابع(دروازه‌ی) منطقی AND

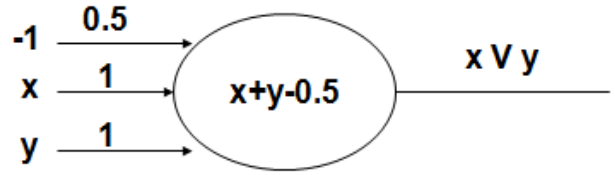


شکل بالا- در شکل بالا توجّه نمایید که x و y هرکدام دارای مقدارهای ۱ یا ۰ می‌توانند باشند.

۱- logic gate

x	y	x+y-1.5	خروجی
1	1	0.5	1
1	0	-0.5	0
0	1	-0.5	0
0	0	-1.5	0

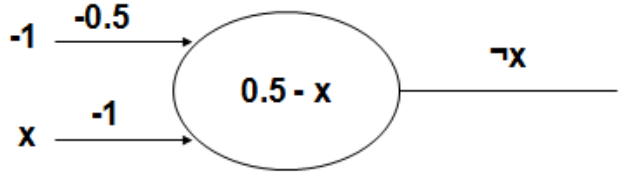
مثال- تابع منطقی OR



شکل بالا- در شکل بالا توجّه نمایید که x و y هرکدام دارای مقدارهای ۱ یا ۰ می‌توانند باشند.

x	y	x+y-0.5	خروجی
1	1	1.5	1
1	0	0.5	1
0	1	0.5	1
0	0	-0.5	0

مثال- تابع منطقی NOT



شکل بالا- در شکل بالا توجّه نمایید که x دارای مقدارهای ۱ یا ۰ می‌تواند باشد.

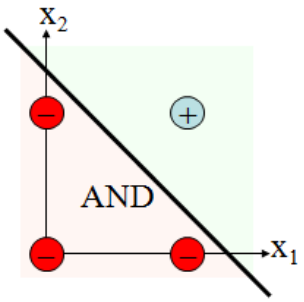
x	0.5 - x	خروجی
1	-0.5	0
0	0.5	1

ایراد پرسپترون‌های تک لایه‌ای

مطلب مهم:

با پرسپترون‌های تک لایه‌ای فقط می‌توان تابع‌هایی را که به صورت خطّی جدا شدنی[1] هستند را نشان داد و نمی‌توان XOR را که به صورت خطّی جدا نشدنی است، نشان داد؛ برای رفع این مشکل می‌توانیم از *پرسپترون‌های چند لایه‌ای* استفاده کنیم.

مثال- تابع AND به صورت خطّی جدا شدنی است.

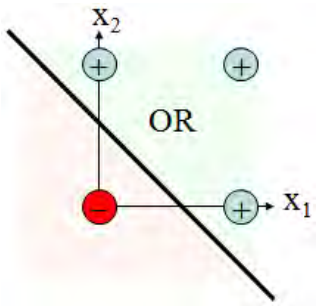


توضیح:

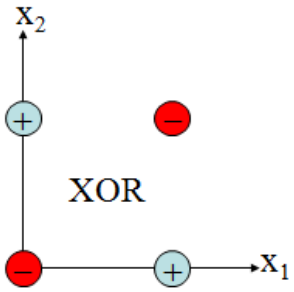
در شکل بالا، نقطه‌ی مثبت، نقطه‌ای است که مقدار تابع به ازای آن (x_1,x_2)، یک می‌شود و نقطه‌های منفی، نقطه‌هایی هستند که به ازای آنها (x_1,x_2)، مقدار تابع، صفر می‌شود و نقطه‌ی مثبت و نقطه‌های منفی، در شکل بالا، به وسیله‌ی یک خطّ مستقیم، قابل جدا شدن هستند.

۱- linear separability

مثال- تابع OR به صورت خطّی جدا شدنی است.



مثال- تابع XOR به صورت خطّی *جدا شدنی نیست*.



توضیح:

همان طور که در شکل بالا می‌بینید، نمی‌توانیم یک خطّ مستقیم را طوری رسم کنیم که مقدارهای مثبت را از مقدارهای منفی جدا نماید.

طبقه‌بندی با استفاده از شبکه‌های عصبی

شبکه‌های عصبی، طبقه‌بندی را هم خیلی خوب انجام می‌دهند؛ ورودی‌ها را به یک یا بیش‌تر خروجی تبدیل می‌نمایند و دامنه‌ی خروجی به کلاس‌هایی مجزّا تقسیم می‌شود و برای کارهای یادگیری، در جایی که نمی‌دانیم «در جستجوی چه هستیم»، خیلی مفید می‌باشند، مثل: صورت‌شناسی[۱]، دست‌خط‌شناسی[۲] و رانندگی یک خودرو.

۱- face recognition

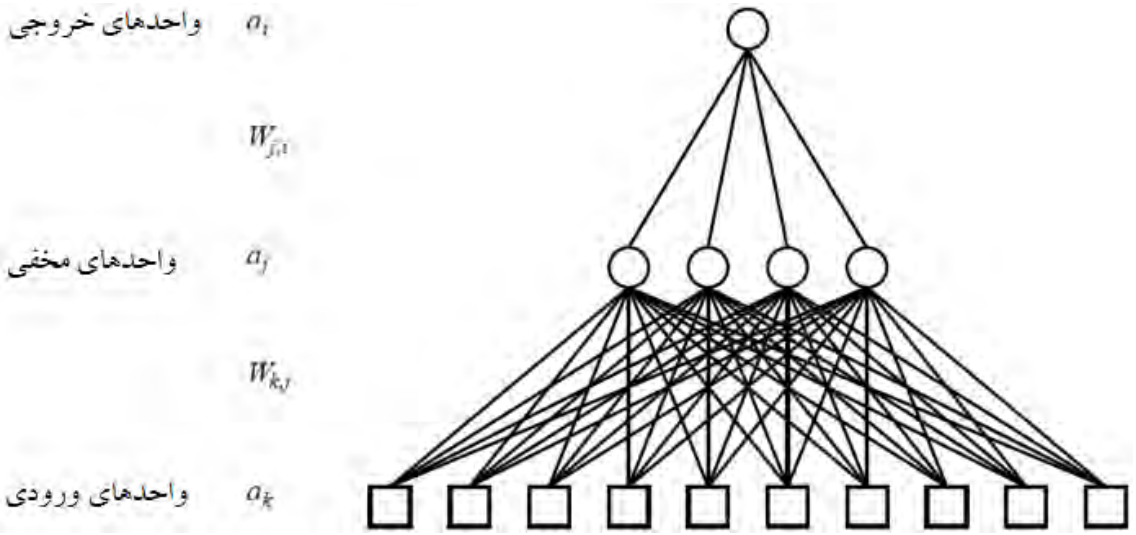
۲- handwriting recognition

پرسپترون‌های چند لایه‌ای

مطلب مهم:

در پرسپترون‌های چند لایه‌ای، لایه‌ها معمولاً به صورت کامل متّصل می‌شوند و به طور معمول، تعدادی از واحدهای پنهان(مخفی)، به صورت دستی انتخاب می‌شوند. *پرسپترون‌ها[ی تک لایه‌ای] دارای این مزّیت هستند که یک الگوریتم یادگیری ساده دارند؛ ولی عیب آنها در این است که دارای محدودیّت‌های محاسباتی هستند؛* حال سؤال این است که اگر ما یک لایه‌ی مخفی اضافه نماییم، به عبارتی دیگر، پرسپترون چند لایه‌ای با یک لایه‌ی پنهان بسازیم، چه اتّفاقی می‌افتد؟، پاسخ، این است که *توان محاسباتی افزایش می‌یابد؛* با یک لایه‌ی پنهان، [پرسپترون] می‌تواند هر تابع پیوسته را نمایش دهد و با دو لایه‌ی پنهان(پرسپترون چند لایه‌ای با دو لایه‌ی پنهان) می‌تواند هر تابعی را نمایش دهد؛

در این مورد مشکل این است که چگونه وزن‌های صحیح را برای گره‌های پنهان، پیدا نماییم؟.

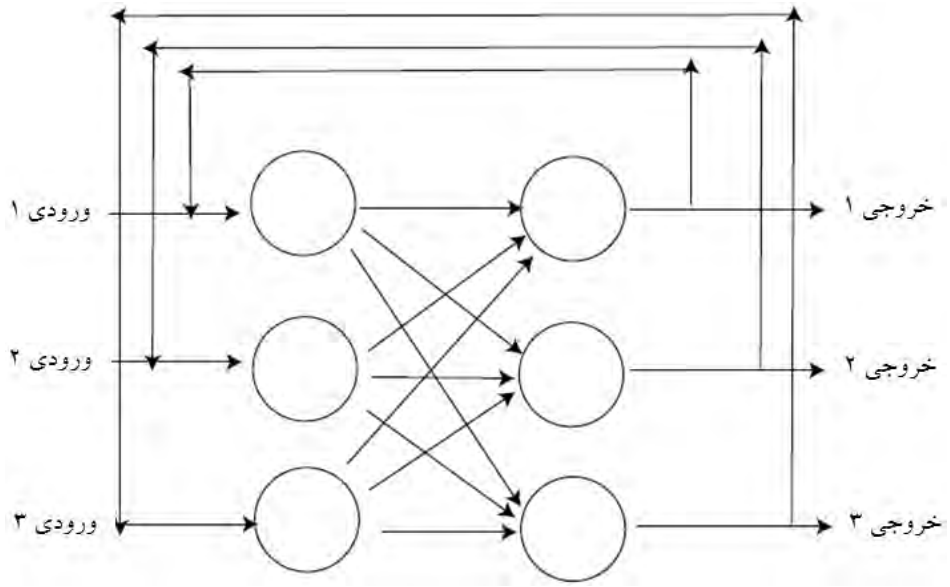


شکل بالا– نمونه‌ای از یک پرسپترون چند لایه‌ای

شبکه‌های عصبی برگشت کننده[1]

تاکنون فقط در مورد شبکه‌های با تغذیّه‌ی مستقیم صحبت کرده‌ایم. در این شبکه‌ها، علامت‌ها(سیگنال‌ها) در یک جهت پخش می‌شوند، خروجی بلافاصله در دسترس می‌باشد و دارای الگوریتم‌هایی هستند که به‌سادگی قابل فهم می‌باشند. تعداد زیادی کار می‌تواند با شبکه‌های عصبی برگشت کننده انجام شود؛ در این شبکه‌ها، دست‌کم یکی از خروجی‌ها به یکی از ورودی‌ها متّصل شده است. در زیر یک شبکه عصبی برگشت کننده‌ی تک لایه‌ای را می‌بینید:

۱- recurrent *NNs*(*N*eural *N*etworks)



شبکه‌های هاپْفیلد[۱]


جان جُوزِف هاپفیلد

شبکه‌های هاپفیلد به وسیله‌ی دکتر، جانْ جُوزِفْ هاپْفیلْدْ[۲] به وجود آمده‌اند و جزء شبکه‌های عصبی برگشت کننده می‌باشند. یک شبکه‌ی هاپفیلد دارای هیچ گره‌ی معیّن ورودی یا خروجی نمی‌باشد؛ هر گره، یک ورودی و روال‌های یک خروجی را دریافت می‌نماید؛ هر گره، به گره‌های دیگر متّصل شده است و معمولاً از توابع آستانه‌ای استفاده می‌شود؛ شبکه بلافاصله یک خروجی را تولید نمی‌نماید و مردّد می‌باشد؛ تحت برخی از وضعیّت‌های به آسانی قابل دسترس، سرانجام به حالت موازنه می‌رسد؛ وزن‌ها با استفاده از روش شبیه‌سازی گرم و سرد کردن به دست می‌آیند. شبکه‌های هاپفیلد می‌توانند برای ساختن یک *حافظه‌ی شرکت‌پذیر(انجمنی)*[۳] *به کار روند، در این شبکه‌ها، یک قسمت از یک الگو برای شبکه ارائه می‌شود و شبکه تمام الگو را به یاد می‌آورد.* این شبکه‌ها، برای حرف‌شناسی[۴] و برای مسأله‌های بهینه‌سازی هم به کار می‌روند و اغلب برای مدل فعّالیّت مغز استفاده می‌شوند.

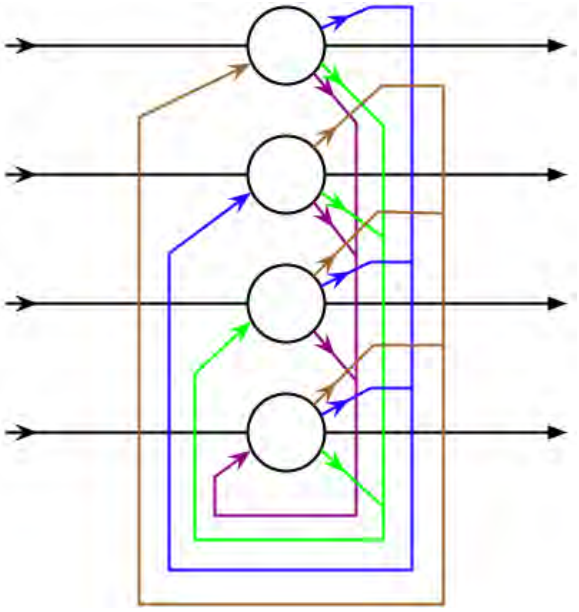
---

۱ - Hopfield networks

۲ - John Joseph Hopfield، متولّد ۱۹۳۳ میلادی، دانشمندی آمریکایی است.
(http://en.wikipedia.org/wiki/John_Joseph_Hopfield)

۳ - associative memory یا content addressable memory(*CAM*) یا حافظه‌ی قابل آدرس‌دهی به‌وسیله‌ی محتوا، حافظه‌ای است که در آن، آدرس‌دهی، با استفاده از محتوای حافظه انجام می‌شود.

۴ - letter recognition



شکل بالا– نمونه‌ای از یک شبکه‌ی هاپفیلد[۱]

۱ - http://en.wikipedia.org/wiki/Hopfield_net

چکیده‌ی مطلب‌های فصل هفدهم

یک شبکه‌ی عصبی، یک سیستم کامپیوتری است که از روی مغز انسان و سیستم عصبی مدلسازی شده است.

دو نوع شبکه‌ی عصبی داریم: یکی شبکه‌های با تغذیه‌ی مستقیم، که در این شبکه‌ها علامت(سیگنال)ها در یک جهت حرکت می‌کنند و بدون دُوْرْ(سیکل) می‌باشند؛ و دیگری شبکه‌های بازگشت کننده، که در این شبکه‌ها در انتشار علامت(سیگنال)، دُوْرْ(سیکل) وجود دارد؛ در این شبکه‌ها دست‌کم یکی از خروجی‌ها به یکی از ورودی‌ها متّصل شده است.

پرسپترون تک لایه‌ای، الگوریتمی ابتدایی برای یادگیری شبکه‌های عصبی تک لایه‌ای ساده است.

با پرسپترون‌های تک لایه‌ای فقط می‌توان تابع‌هایی را که به صورت خطّی جدا شدنی هستند را نشان داد و نمی‌توان XOR را که به صورت خطّی جدا نشدنی است، نشان داد؛ برای رفع این مشکل می‌توانیم از پرسپترون‌های چند لایه‌ای استفاده کنیم.

پرسپترون‌ها[ی تک لایه‌ای] دارای این مزیّت هستند که یک الگوریتم یادگیری ساده دارند؛ ولی عیب آنها در این است که دارای محدودیّت‌های محاسباتی هستند. اگر ما یک لایه‌ی مخفی اضافه نماییم، به عبارتی دیگر، پرسپترون چند لایه‌ای با یک لایه‌ی پنهان بسازیم، توان محاسباتی افزایش می‌یابد؛ با یک لایه‌ی پنهان، [پرسپترون] می‌تواند هر تابع پیوسته را نمایش دهد و با دو لایه‌ی پنهان(پرسپترون چند لایه‌ای با دو لایه‌ی پنهان) می‌تواند هر تابعی را نمایش دهد.

یادآوری یا تکمیل مطلب‌های فصل هفدهم

سؤال- شبکه‌های عصبی می‌توانند برای جواب دادن به چه مواردی به کار روند؟

جواب-

الگوشناسی: به عنوان مثال، آیا در این عکس، یک چهره(صورت) وجود دارد؟.

مسأله‌های طبقه‌بندی: به عنوان مثال، آیا این باتری، معیوب است؟

پیش‌بینی: به عنوان مثال، با داشتن این علائم، بیمار، دارای بیماری X است.

پیش‌بینی: به عنوان مثال، پیش‌بینی رفتار موجودی انبار بازار.

دست‌خط: به عنوان مثال، آیا این کاراکتر است؟.

بهینه‌سازی: به عنوان مثال، کوتاه‌ترین مسیر را برای مسأله‌ی مسافرت شخص دوره‌گرد بیابید.[1]

تعریف- پرسپترون را تعریف کنید.

جواب- یک شبکه‌ی عصبی تک لایه‌ای است، که در آن، گره‌های ورودی به طور مستقیم به گره‌های خروجی، بدون هیچ واحد پنهانی وصل شده‌اند.[2]

درست یا غلط- یک پرسپترون می‌تواند هر تابع بولیّنی را یاد بگیرد و ارائه نماید.

جواب- «غلط» است.[1]

---

۱ - مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

۲ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، بهار سال ۱۹۹۳ میلادی

درست یا غلط- شبکه‌های عصبی می‌توانند فقط خروجی ۰ یا ۱ داشته باشند.

جواب- «درست» است.[۲]

درست یا غلط- مغز انسان، در نگاه اوّل می‌تواند به صورت یک شبکه‌ی عصبی با تغذیه‌ی مستقیم چندلایه‌ای خیلی بزرگ توصیف شود.

جواب- «غلط» است؛ یک شبکه‌ی عصبی با تغذیه‌ی مستقیم، دارای هیچ وضعیّت ورودی‌ای نمی‌باشد و بنابراین، دارای حافظه نیست.[۳]

مطلب- شبکه‌های عصبی مصنوعی، یک تلاش پایین به بالا برای مدلسازی کارکرد مغز هستند.[۴]

---

۱ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۲ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، پاییز سال ۱۹۹۷ میلادی

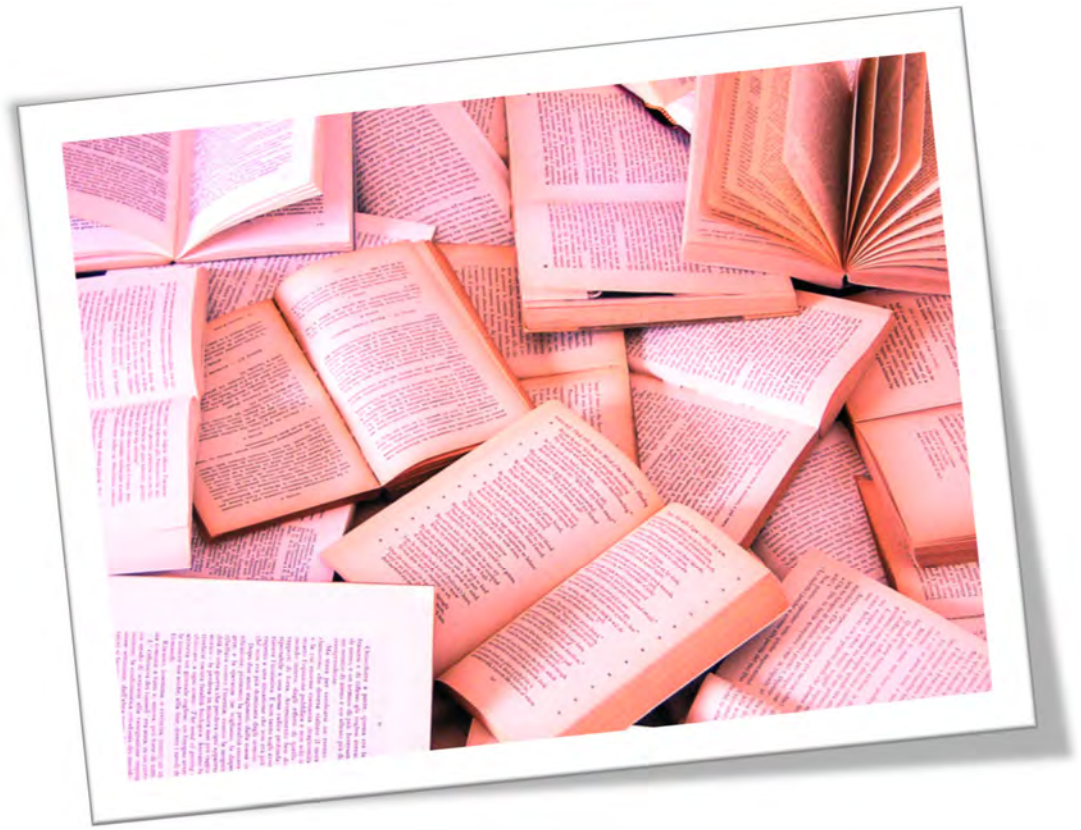
۳ - آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، پاییز سال ۱۹۹۹ میلادی

۴ - مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

فصل هیجدهم

الگوریتم‌های ژنتیکی[۱]

۱ - Genetic Algorithms



توجّه:

برای یادگیری بهتر مطلب‌های این فصل، بهتر است قسمت «الگوریتم‌های ژنتیکی» موجود در فصل «الگوریتم‌های جستجوی محلّی» همین کتاب الکترونیکی را هم بخوانید.

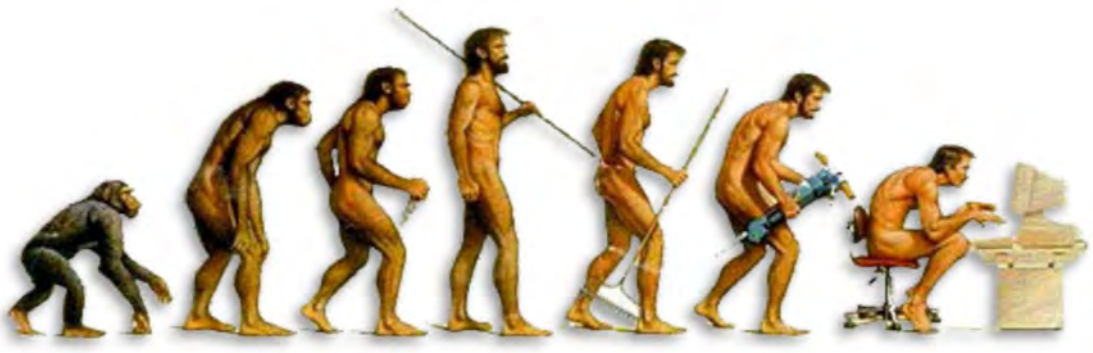
فهرست برخی از عنوان‌های نوشته‌ها

تئوری تکاملی داروین(باقی ماندن شایسته‌ترین[۱] یا مناسب‌ترین)

هِربِرت سپنسِر

این تئوری به وسیله‌ی هِرْبِرْتْ سْپِنْسِرْ[۲] و چارْلْزْ رابِرْتْ دارْوینْ[۳] به وجود آمد. این تئوری می‌گوید: گونه‌هایی باقی می‌مانند که می‌توانند بر مشکلات غلبه کنند.[۴]

چارلز داروین



شکل بالا– این شکل بیان کننده‌ی نظریّه‌ی تکامل در انسان است.

---

۱ – survival of the fittest

۲ – Herbert Spencer، ۱۹۰۳ – ۱۸۲۰ میلادی، فیلسوف، زیست‌شناس(biologist) و جامعه‌شناس(sociologist)، از کشور انگلیس بود.
(http://en.wikipedia.org/wiki/Herbert_Spencer)

۳ – Charles Robert Darwin، ۱۸۸۲ – ۱۸۰۹ میلادی، طبیعت‌شناس(naturalist)، از کشور انگلیس بود.
(http://en.wikipedia.org/wiki/Charles_Robert_Darwin)

۴ – Babylon> Babylon English-English

تاریخچه

جان هِنری هالِند

الگوریتم‌های ژنتیکی در دهه‌ی هفتاد میلادی به وسیله‌ی جانْ هِنْری هالِنْد[۱] بیان شد؛ در دهه‌ی هشتاد میلادی عمومیّت یافت و براساس تئوری تکاملی داروین(باقی ماندن شایسته‌ترین یا مناسب‌ترین) بود. الگوریتم‌های ژنتیکی می‌توانند برای حلّ انواعی از مسأله‌ها که با استفاده از روش‌های دیگر، برای حل، آسان نمی‌باشند، مورد استفاده قرار گیرند.

تعریف الگوریتم‌های ژنتیکی

تعریف اوّل- یک الگوریتم جستجو که رشته‌های دودویی بهینه را با پردازش یک جمعیّت اولیّه‌ی تصادفی از رشته‌ها، با استفاده از جهش مصنوعی، عمل تعویض[۲] و عملگرهای انتخاب تولید می‌کند. (گُلْدبِرْگ[۳]، ۱۹۸۹ میلادی).[۴]

تعریف دوّم- یک الگوریتم *تکاملی* که هر فرد(راه حل) را با استفاده از برخی فرم‌های رمزشده که کُرُوموزوم[۵] یا ژِنوم نامیده می‌شوند، به وجود می‌آورد.[۶]

تکامل در دنیای واقعی

هر سلّول[۷] یک موجود زنده دارای یک هسته[۸] است؛ و در هسته، *کروموزوم*هایی وجود دارد؛ هر کروموزوم، شامل یک مجموعه از ژِنْ[۹]ها می‌باشد؛ هر ژن، برخی از ویژگی‌های(خواصّ) موجود زنده (مانند رنگ چشم) را تعیین می‌کند. یک مجموعه از

۱- John Henry Holland؛ ۲۰۱۵ - ۱۹۲۹ میلادی، دانشمندی آمریکایی و استاد روانشناسی، مهندسی برق و علوم کامپیوتر در دانشگاه میشیگان کشور آمریکا بود.
(http://en.wikipedia.org/wiki/John_Henry_Holland)

۲- crossover

۳- Goldberg؛ با این شخص در فصل «سیستم‌های طبقه‌بندی کننده»ی همین کتاب آشنا خواهید شد.

۴- Babylon> Electronic Statistics Textbook

۵- chromosome

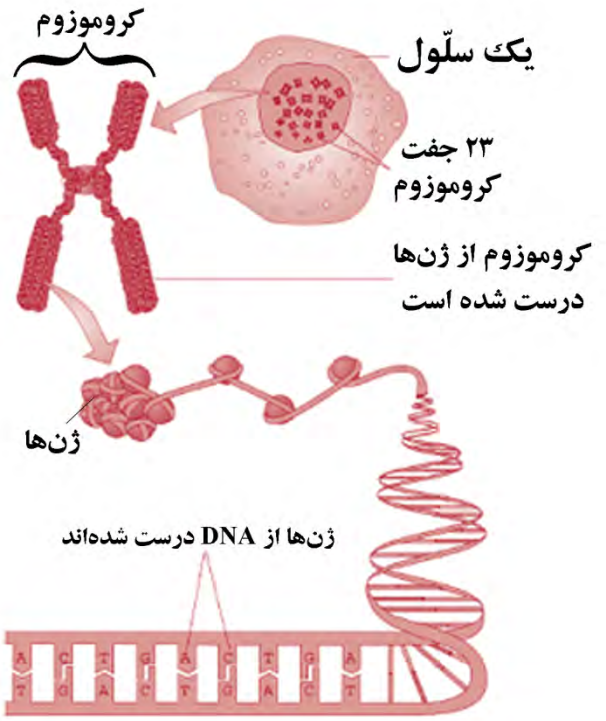
۶- Babylon> FOLDOC

۷- cell

۸- Nucleus

۹- gene

ژن‌هـا، در برخـی از موردهـا *ژنوتیپ(ژنوتایـپ)*[۱] نامیـده می‌شـود. یـک مجموعـه از ویژگی‌هـا (نظیـر رنـگ چشـم)، گـاهی *فنوتیپ(فنوتایپ)*[۲] نامیده می‌شود. انتشار(زاد و ولد)، شامل جفت‌گیری[۳] ژن‌های والدین و سپس مقدار کـوچکی *جهـش*[۴] می‌باشـد. تکامل، براساس «بقای مناسب‌ترین یا شایسته‌ترین» می‌باشد.



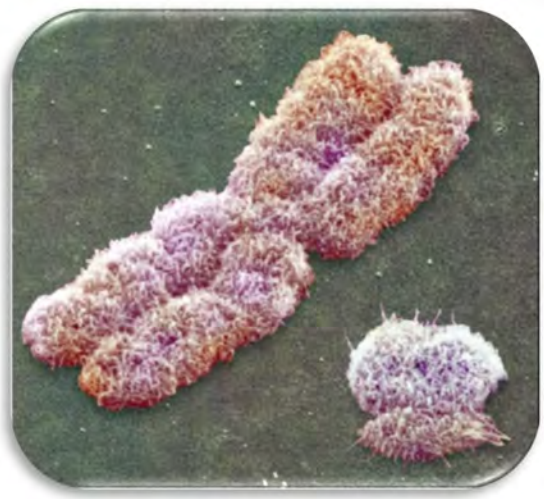
شکل بالا– سلّول، هسته، کروموزوم و ژن

۱ – genotype

۲ – phenotype

۳ – recombination؛ در زیست‌شناسی، پردازه‌ای است که در آن، ژن‌ها شکسته می‌شوند و به دیگر ژن‌ها می‌چسبند و در الگوریتم‌های ژنتیکی، همان تعویض(crossover) می‌باشد.
(http://en.wikipedia.org/wiki/Recombination)

۴ – mutation



شکل بالا– کروموزوم واقعی در زیر میکروسکوپ

الگوریتم‌های ژنتیکی

شروع با یک تصوّر

فرض کنید که شما مشکلی دارید و نمی‌دانید که چگونه آن را حل نمایید. برای حلّ این مشکل چه کار می‌توانید بکنید؟، آیا می‌توانید به طریقی از یک کامپیوتر برای پیدا کردن یک راه حل استفاده نمایید؟؛ این کار باید خوب باشد! می‌توانید آن را انجام دهید؟.

یک راه حلّ گنگ

یک الگوریتم «تولید و تست گنگ»:

تکرار کن

یک راه حلّ ممکن تصادفی را تولید کن

راه حل را امتحان کن و خوب بودن آن را بسنج

تا موقعی که راه حل به اندازه‌ی کافی خوب باشد

پایان الگوریتم

آیا ما می‌توانیم از این ایده‌ی گنگ استفاده نماییم؟

برخی وقت‌ها، بله: در صورتی که تعداد راه حل‌های اندکی وجود داشته باشد و شما به اندازه‌ی کافی زمان در اختیار داشته باشید، آنگاه روش‌های این فُرمی می‌توانند استفاده شوند. *امّا برای بیش‌ترِ مسأله‌ها نمی‌توانیم از این راه حل‌ها استفاده نماییم:* در زمانی که راه حل‌های ممکن، زیاد باشند و شما به اندازه‌ی کافی زمان برای امتحان همه‌ی آنها نداشته باشید، این راه حل‌ها نمی‌توانند استفاده شوند.

ایده‌ای که کم‌تر دارای گنگ بودن می‌باشد (الگوریتم ژنتیکی)

الگوریتم

یک مجموعه‌ی تصادفی از راه حل‌ها را تولید نمایید

تکرارکن

هر راه حلّ موجود در مجموعه را امتحان کن و آنها را رُتبه‌بندی کن

برخی از راه حل‌های بد را از مجموعه بَردار

برخی از راه حل‌های خوب را تکثیر(زیاد) کن

برخی از تغییرات کوچک را در مورد آنها به کار بِبَر

تا زمانی که بهترین راه حل به اندازه‌ی کافی خوب شود

پایان الگوریتم

چگونه شما یک راه حل را کد می‌کنید؟

بدیهی است که این وابسته به مسأله می‌باشد! الگوریتم‌های ژنتیکی اغلب، راه حل‌ها را بـه صـورت رشـته‌های بیتـیِ[۱] باطول ثابت(ژنوتیپ یا کروموزوم)، کد می‌کنند (به عنوان مثال، ۱۰۱۱۱۰، ۱۱۱۱۱۱، ۰۰۰۱۰۱)؛ هر بیت(ژن)، برخی از ویژگی‌هـای راه حل‌های ارائه شده برای مسأله را بیان می‌کند. برای اینکه الگوریتم‌های ژنتیکی کار کنند، نیاز به این داریم که هر رشته را تسـت نماییم و به آن امتیازی بدهیم که نشان دهنده‌ی چگونگی خوب بودن آن باشد.

مثال – حَفر برای نفت

شکل بالا– تصویری از یک چاه نفت

تصوّر کنید که شما باید برای نفت، جایی را در طول یک جادّه‌ی بیابانی یک کیلومتری حفر نمایید.

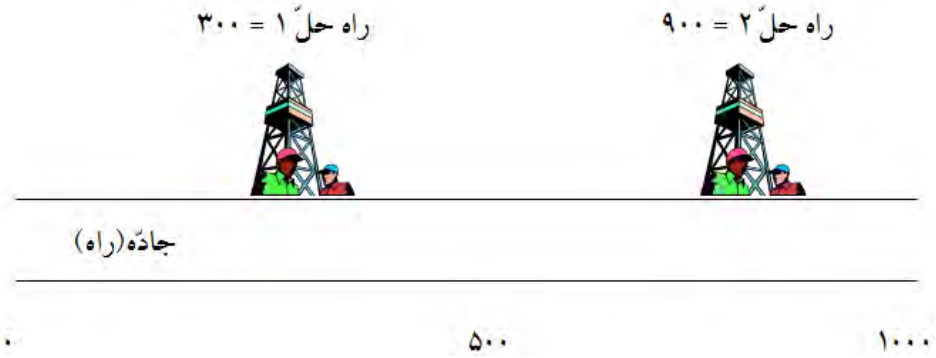
مسأله: بهترین مکان در جادّه را که بیش‌ترین مقدار نفت را در روز تولید می‌کند، انتخاب نمایید.

می‌توانیم هر راه حل را به صورت یک وضعیّت در جادّه نمایش دهیم. تصوّر نمایید که تمام اعداد، بین [۱۰۰۰، ۰] باشند.

---

۱ - bitstrings

کجا را برای نفت حفر نماییم؟

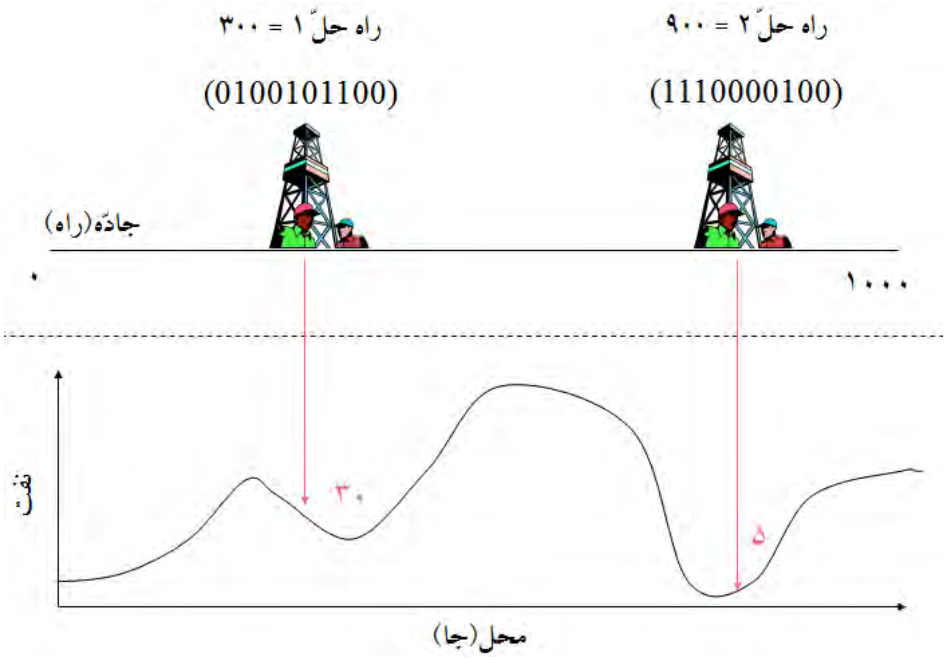


مجموعه‌ی همه‌ی راه حل‌های ممکن [۰،۱۰۰۰]، فضای جستجو یا فضای حالت نام دارد. در این مورد این فقط یک عدد است؛ امّا می‌تواند تعداد زیادی عدد یا سمبل باشد. همان طور که گفتیم، اغلب، الگوریتم‌های ژنتیکی، اعداد را به صورت دودویی(باینری) کد می‌کنند. در مورد این مثال، ده بیت را که برای نمایش ۰ ... ۱۰۰۰ کافی است، انتخاب می‌نماییم.

تبدیل به رشته‌ی باینری

	۵۱۲	۲۵۶	۱۲۸	۶۴	۳۲	۱۶	۸	۴	۲	۱
۹۰۰	۱	۱	۱	۰	۰	۰	۰	۱	۰	۰
۳۰۰	۰	۱	۰	۰	۱	۰	۱	۱	۰	۰
۱۰۲۳	۱	۱	۱	۱	۱	۱	۱	۱	۱	۱

تعریف– در الگوریتم‌های ژنتیکی، این رشته‌های کد شده، گاهی ژنوتیپ یا کروموزوم نامیده می‌شوند و بیت‌های تکی، گاهی وقت‌ها ژن نامیده می‌شوند.



خلاصه

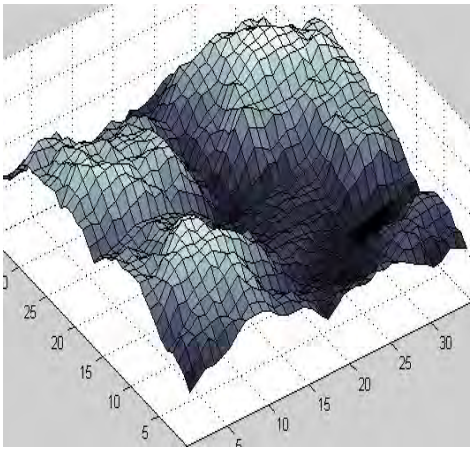
تاکنون دیدیم که چگونه راه حل‌ها را به صورت یک عدد بیان نماییم؛ یک عدد را به صورت یک رشته‌ی بیتی کد کردیم. یک رتبه یا درجه را برای هر عدد داده شده به منظور تعیین میزان خوب بودن آن راه به آن عدد نسبت می‌دهیم، که اغلب، *تابع شایستگی*[1] نامیده می‌شود؛ در شکل قبلی، این اعداد برای دو راه حلّ ۱ و ۲، به رنگ قرمز نشان داده شده‌اند(عددهای ۳۰ و ۵). مثال نفت، در واقع بهینه‌سازی با استفاده از یک تابع $f(x)$ است، که ما باید پارامتر x را تطبیق دهیم.

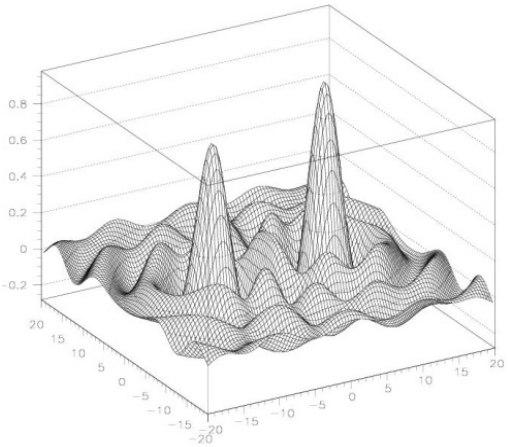
فضای جستجو

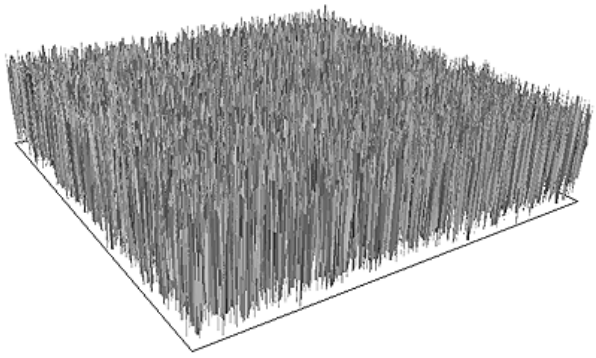
برای یک تابع ساده‌ی $f(x)$، فضای جستجو یک بعدی است، امّا با کد کردن چند مقدار به کروموزوم، ابعاد زیادی می‌تواند به وجود آید؛ به عنوان مثال، برای حالت دو بعدی، تابع f، به صورت $f(x,y)$ خواهد بود. فضای جستجو می‌تواند به صورت یک سطح باشد، که در آن، هر چه شایستگی بیش‌تر باشد، ارتفاع نقطه بیش‌تر است. هر ژنوتایپ(کروموزوم یا راه حلّ ممکن، یک نقطه در فضا است. یک الگوریتم ژنتیکی تلاش می‌کند که نقطه‌ها را به جاهای بهتر(با شایستگی بالاتر)، در فضا منتقل نماید. در زیر سه نمونه از منظره‌های شایستگی را مشاهده می‌نماییم:

---

۱ - fitness function







بدیهی است که نوع فضای جستجو تعیین می‌کند که چگونه یک الگوریتم ژنتیکی کار کند. یک فضای کاملاً تصادفی، برای یک الگوریتم ژنتیکی، بد خواهد بود؛ همچنین الگوریتم‌های ژنتیکی، اگر فضاهای جستجو شامل تعداد زیادی از موردهای تصادفی باشند، ممکن است در ماکزیمم محلّی بیفتند.

اضافه کردن جنسیّت– عمل تعویض

گرچه الگوریتمی که ما گفتیم برای فضاهای جستجوی ساده کار می‌کند، امّا این الگوریتم هنوز خیلی ساده است. این الگوریتم به جهش‌های تصادفی برای یافتن یک راه حلّ خوب وابسته می‌باشد.

مطلب مهم:

با اضافه نمودن جنسّیت به این الگوریتم، می‌توانیم نتایج بهتری را به دست آوریم؛ این کار با انتخاب دو والد در موقع تکثیر[۱] و ترکیب ژن‌های آنها برای تولید فرزند انجام می‌شود؛ به بیان دیگر، دو رشته‌ی بیتی(کروموزوم) والد با امتیاز بالا انتخاب می‌شوند و با استفاده از تعویض با هم ترکیب می‌شوند؛ در نتیجه دو فرزند(رشته‌ی بیتی) به وجود می‌آیند و سپس هر فرزند هم ممکن است به صورت تصادفی تغییر داده شود، که به این کار، جهش گفته می‌شود.

انتخاب والدها

روش‌های زیادی برای انتخاب کروموزوم‌های با امتیاز بهتر وجود دارد. امتیاز اغلب *شایستگی*[۲] نامیده می‌شود. [برای این کار،] از روش «چَرخِشِ رولِت»[۳] می‌توان به صورت زیر استفاده نمود:

- شایستگی همه‌ی کروموزوم‌ها را باهم جمع نمایید.
- یک *عدد تصادفی* R را در محدوده‌ی آن(حاصل‌جمع) به وجود آورید.
- اوّلین کروموزوم موجود در جمعیّت را با این شرط که زمانی که همه‌ی شایستگی‌های کروموزوم‌های قبل از آن باهم جمع می‌شوند، دست‌کم مقدار R را بدهد، انتخاب نمایید.

توجّه:

قسمت زیر دارای ارتباط بین رنگ‌ها هم هست.

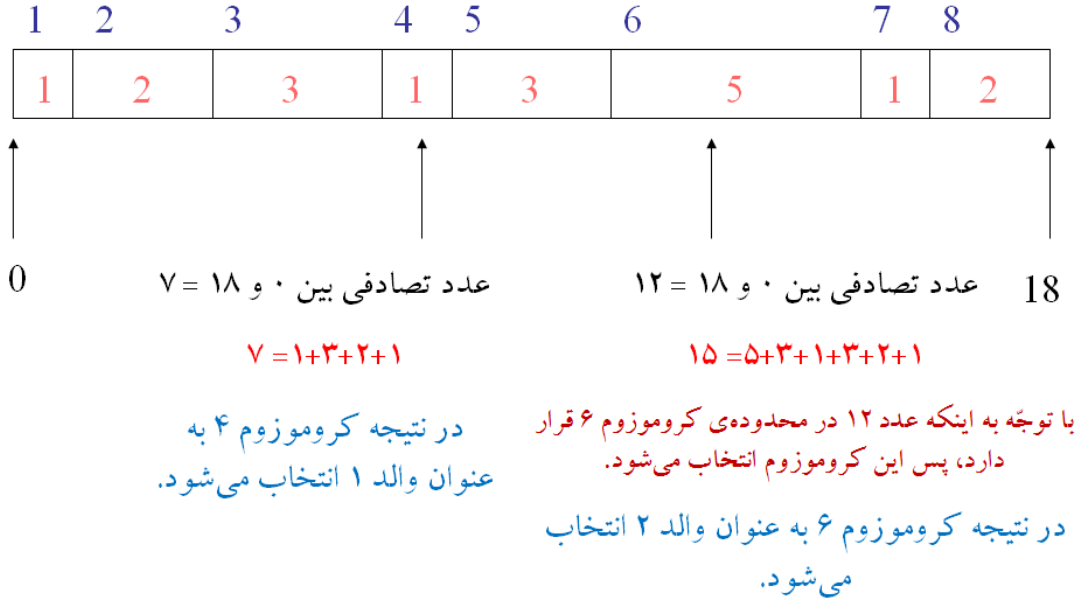
۱- reproduction
۲- fitness
۳- Roulette Wheel

جمعیّت نمونه

شماره	کروموزوم	شایستگی
1	1010011010	1
2	1111100001	2
3	1011001100	3
4	1010000000	1
5	0000010000	3
6	1001011111	5
7	0101010101	1
8	1011100111	2

توجّه کنید که جمع کلّ شایستگی‌ها برابر است با: ۱+۲+۳+۱+۳+۵+۱+۲ = ۱۸

انتخاب والدها با استفاده از روش چرخش رولت(برای جدول قبلی)



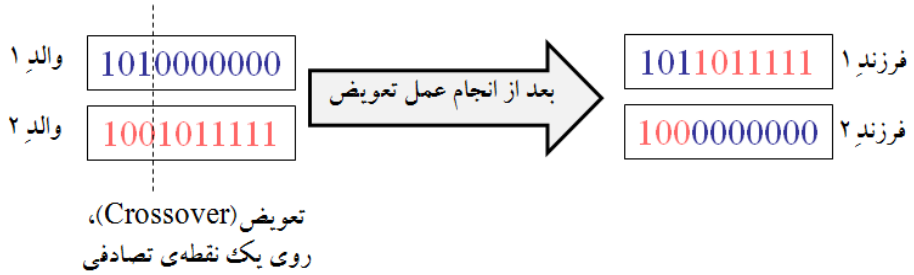
در شکل بالا توجّه کنید که جمع کلّ شایستگی‌ها برابر است با: ۱+۲+۳+۱+۳+۵+۱+۲ = ۱۸ و به همین خاطر، محدوده از صفر تا ۱۸ در نظر گرفته شده است.

چرخش رولت، نام یک بازی هم می‌باشد، که در آن، اعدادی به طور نامنظّم روی یک صفحه‌ی دایره‌ای شکل نوشته شده‌اند؛ این صفحه را می‌چرخانند و سپس رهایش می‌کنند تا خودش از حرکت بایستد و عددی را که مقابل نقطه‌ای خاص قرار گرفته را برمی‌گزینند.

شکل بالا– صفحه‌ی بازی چرخش رولت

تعویض(جفت‌گیری)[۱]

برای کروموزوم‌های شماره‌ی۴(1010000000) و ۶(1001011111)، که در قسمت قبل انتخاب کردیم، نقطه‌ی تعویض را به صورت تصادفی، برابر با ۳ در نظر می‌گیریم؛ داریم:



جهش

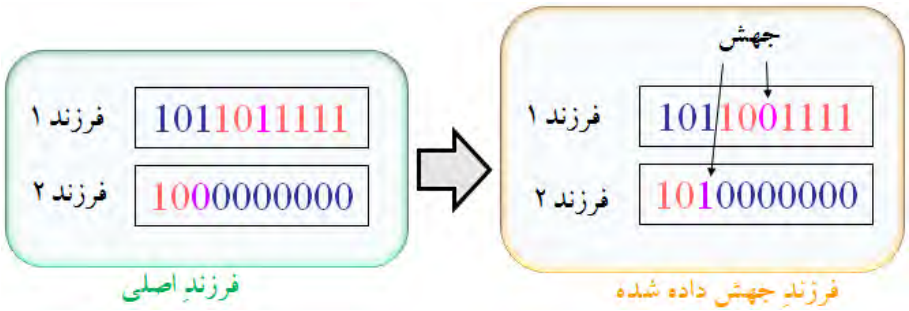
یک یا بیش‌تر ژن(بیت)، به صورت تصادفی، به یک مقدار تصادفی، جهش داده می‌شوند، مثلاً:

(بعد از انجام عمل جهش)۱۱۰۱۰۱۱۱۱۱ ← ۱۱۱۱۱۱۱۱۱۱(قبل از انجام عمل جهش)

در مورد مثال قبلی داریم:

---

۱ - Crossover(Recombination)



توجّه:

پایان قسمت دارای ارتباط بین رنگ‌ها!

برگشت به الگوریتم ژنتیکی [و تکمیل آن]

الگوریتم

یک جمعیّت را با استفاده از کروموزوم‌های تصادفی تولید کن

برای هر نسل کارهای زیر را انجام بده

شایستگی هر کروموزوم را محاسبه کن

کارهای زیر را تکرار کن

انتخاب رولت را برای انتخاب جفت‌های والدها به کار ببر

فرزند را با استفاده از انجام عمل تعویض و جهش تولید نما

تا زمانی که جمعیّت جدید تولید شود

تا زمانی که بهترین راه حل به اندازه‌ی کافی خوب شود

پایان الگوریتم

گوناگونی‌های زیاد الگوریتم‌های ژنتیکی

به خاطر موردهای زیر الگوریتم‌های ژنتیکی، دارای گوناگونی زیادی هستند:

- انـواع مختلـف انتخـاب، کـه [بـا اسـتفاده از روش چـرخش] رولـت نیسـتند؛ مثـل: روش‌هـای مسـابقه‌ای[۱]، گـزینش بهترین‌ها(نخبه‌سالاری)[۲] و غیره.
- تفاوت در جفت‌گیری(تعویض)؛ مثل: روش تعویض چند نقطه‌ای.
- انواع مختلف کد کردن رشته‌ی بیتی
- انواع مختلف جهش

تعویض چگونه کار می‌کند؟

تعداد زیادی نظریّه در این مورد وجود دارد، البتّه برخی مخالفت‌ها هم با این نظریّه‌ها وجود دارد؛ هالِند نظریّه‌ی «الگو» را معرّفی کرد؛ این ایده می‌گوید که تعویض، بیت‌های خوب والدین مختلف را نگهداری می‌کند و آنهـا را بـرای تولیـد راه حل‌هـای بهتر ترکیب می‌نماید؛ بنابراین، یک الگوی خوب کد کننده باید بیت‌های خوب را در طول تعویض و جهش نگهداری نماید.

کاربردهای الگوریتم‌های ژنتیکی

الگوریتم‌های ژنتیکی در بسیاری از موردها و زمینه‌های پژوهشی، نظیر مسأله‌های عددی؛ مسأله‌ی مسافرت شخص دوره گرد[۳]؛ مدار(دُوْر) هَمیلتُونی[۴]؛ پیدا کردن شکل مولکول‌های پروتئین؛ مسأله‌های NP–سخت؛ طرّاحی شبکه‌های عصبی؛ توابع، برای به وجود آوردن تصاویر؛ مدلسازی شناختی؛ و تئوری بازی‌ها به کار می‌روند.

---

۱ - Tournament

۲ - Elitism

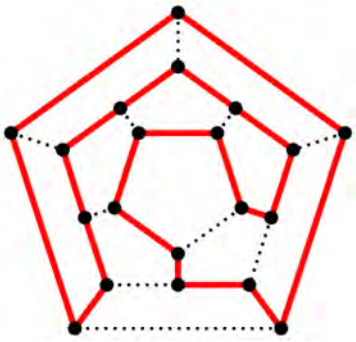
۳ - همان طور که در فصل‌های قبل هم بیان شد، در مسأله‌ی مسافرت شخص دوره گرد، یک دوره گرد باید کوتاه‌ترین مسیر را که یک مجموعه از شهرها می‌باشد و باید آن‌ها را طی کند، پیدا نماید. فرض کنید که فاصله‌ی میان هر شهر را می‌دانیم. این مسأله‌ای مشکل می‌باشد، زیرا تعداد مسیرهای ممکن، برابر !N می‌باشد، که N، برابر با تعداد شهرها می‌باشد. الگوریتم ساده‌ای که در این مورد بهترین جواب را به سرعت ارائه نماید، وجود ندارد.

۴ - Hamiltonian cycle(circuit)؛ مسیر هَمیلتونی(Hamiltonian path) و دُور همیلتونی: این نامگذاری به افتخار ویلیام رُوان هَمیلتُون(William Rowan Hamilton)، ۱۸۶۵ – ۱۸۰۵ میلادی، فیزیکدان(physicist)، ستاره‌شناس(منجّم(astronomer)) و ریّاضیدان ایرلندی(Irish) صورت گرفته است؛ تصویری از این شخص را در زیر می‌بینید:

برنامه‌نویسی ژنتیکی

تعریف– شاخه‌ای از الگوریتم‌های ژنتیکی و یادگیری ماشینی[۱] می‌باشد و یک روش برنامه‌نویسی است که الگوریتم ژنتیکی را برای تمام برنامه‌های کامپیوتری گسترش می‌دهد.[۲] برنامه‌نویسی ژنتیکی به وسیله‌ی گروه دکتر، جان کُوزا[۳] به وجود آمد.

در نظریّه‌ی گراف‌ها، مسیر همیلتونی، *در یک گراف بدون جهت (undirected graph)*، مسیری است که از هر رأس(vertex) دقیقاً یک بار عبور می‌کند؛ اگر پس از طی کردن مسیر، دوباره به گره‌ی شروع برگردیم، در این صورت دور همیلتونی داریم. به عنوان مثال، شکل زیر دور همیلتونی دارد (دُور در این شکل به رنگ قرمز نشان داده شده است و رأس‌ها با رنگ *سیاه پر رنگ* نشان داده شده‌اند).



مسأله‌ی همیلتون، یک مسأله‌ی NP–کامل است.

(http://en.wikipedia.org/wiki/William_Rowan_Hamilton و
http://en.wikipedia.org/wiki/Hamiltonian_path)

۱ – machine learning؛ **تعریف**– شاید بهترین تعریف برای یادگیری ماشینی، «مطالعه‌ی الگوریتم‌های کامپیوتری‌ای که به صورت خودکار و با استفاده از تجربه بهتر می‌شوند»، باشد.

۲ – منبع‌ها:

http://pages.cpsc.ucalgary.ca/~jacob/Courses/Winter2000/CPSC533/Slides/04.3.3-GP.ppt(این پایگاه اینترنتی، متعلّق به دانشگاه شهر کَلگِریِ کشور کانادا است.)، http://www.geneticprogramming.com/Tutorial/ و <Babylon و FOLDOC.

۳ – John Koza، دانشمند علوم کامپیوتر و استاد دانشگاه اِستَنفُورد کشور ایالات متّحده‌ی آمریکا می‌باشد.
(http://en.wikipedia.org/wiki/John_Koza)

شکل بالا– دکتر، جان کوزا

چکیده‌ی مطلب‌های فصل هیجدهم

یک الگوریتم ژنتیکی، یک الگوریتم جستجو است که رشته‌های دودویی بهینه را با پردازش یک جمعیّت اولیّه‌ی تصادفی از رشته‌ها، با استفاده از جهش مصنوعی، عمل تعویض و عملگرهای انتخاب تولید می‌کند.

الگوریتم‌های ژنتیکی اغلب، راه حل‌ها را به صورت رشته‌های بیتی باطول ثابت(ژنوتیپ یا کروموزوم)، کد می‌کنند؛ هر بیت(ژن)، برخی از ویژگی‌های راه حل‌های ارائه شده برای مسأله را ارائه می‌کند. برای اینکه الگوریتم‌های ژنتیکی کار کنند، نیاز به این داریم که هر رشته را تست نماییم و به آن امتیازی بدهیم که نشان دهنده‌ی چگونگی خوب بودن آن باشد.

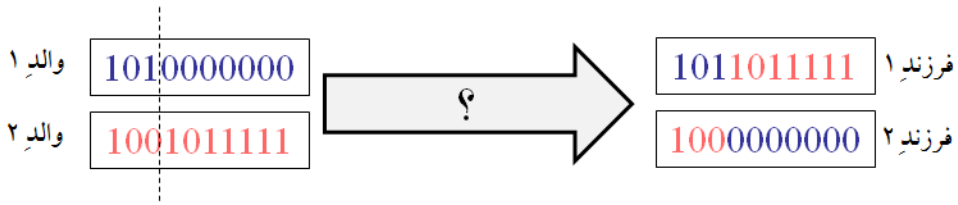
در الگوریتم‌های ژنتیکی، جفت‌گیری(recombination)، همان تعویض(crossover) می‌باشد.

تعویض، با انتخاب دو والد(رشته‌ی بیتی یا کروموزوم) با امتیاز بالا در موقع تکثیر و ترکیب ژن‌های آنها برای تولید دو فرزند(رشته‌ی بیتی) انجام می‌شود. سپس هر فرزند هم ممکن است به صورت تصادفی تغییر داده شود، که به این کار، جهش گفته می‌شود.

یکی از روش‌های انتخاب کروموزوم‌های با امتیاز بهتر، روش چرخش رولت است.

یادآوری یا تکمیل مطلب‌های فصل هیجدهم

سؤال- مشابه کنکور آزاد مهندسی کامپیوتر سال ۸۳- شکل زیر کدام عملگر ژنتیکی را نشان می‌دهد؟



جواب- تعویض (تقاطع).

فصل نوزدهم

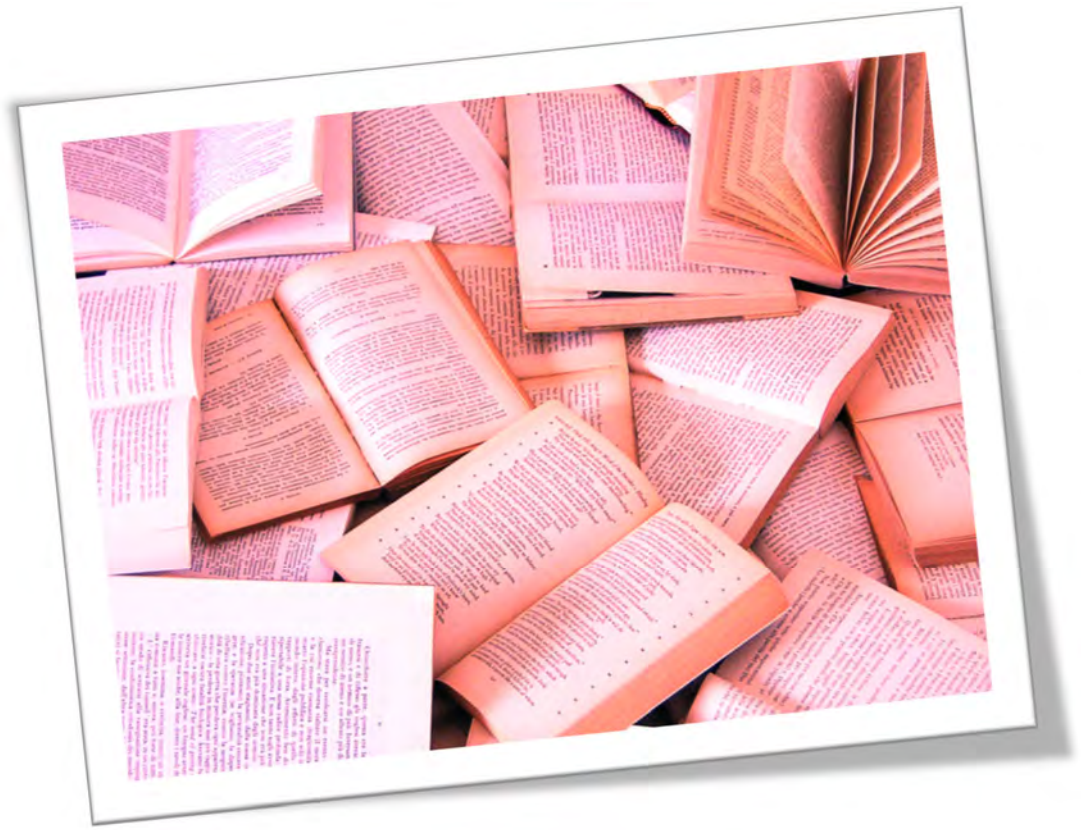
۱

سیستم‌های خِبره[۲]

---

۱ - تصویر، اِدْوارْدْ فایْگِنْباوْمْ(Edward Feigenbaum)، متولّد ۱۹۳۶ میلادی، دانشمند علوم کامپیوتری از کشور آمریکا را نشان می‌دهد؛ وی در زمینه‌ی هوش مصنوعی کار می‌کند و اغلب به عنوان پدر سیستم‌های خبره شناخته می‌شود (http://en.wikipedia.org/wiki/Edward_Feigenbaum)؛ همچنین وی یکی از کسانی است که اوّلین سیستم خبره به نام DENDRAL را در دهه‌ی ۱۹۶۰ میلادی به وجود آورد؛ DENDRAL برای کمک به شیمیدان‌های [شیمی] آلی در تشخیص مولکول‌های آلی ناشناخته، با بررّسی توده‌ی آن‌ها و با استفاده از دانش شیمی به وجود آمد. (http://en.wikipedia.org/wiki/Dendral). شیمی آلی(organic chemistry): بخشی از شیمی است که ترکیباتی که منبع آنها گیاهان و حیات وحش می‌باشند را برّرسی می‌کند. (Babylon> Babylon English) ترکیبات آلی یا زیستی(organic compounds): گروه بزرگی از ترکیبات شیمیایی هستند که معمولاً شامل کربن، هیدروژن، نیتروژن و اکسیژن می‌باشند. تمام موجودات زنده از این ترکیبات آلی تشکیل شده‌اند. (Babylon> Environment Engineering)

۲ - Expert systems



فهرست برخی از عنوان‌های نوشته‌ها

سیستم‌های خبره

تعریف- خبره- در صورتی که ما بگوییم فردی، در رشته‌ای، یک خبره می‌باشد، منظور ما این است که در آن تخصّص محدود، شایستگی بالایی را به نمایش می‌گذارد.

تعریف- سیستم‌های خبره- برنامه‌هایی طرّاحی شده برای مدلسازی و استدلال در مورد دانش انسان می‌باشند و معمولاً حول یک دامنه، مثل تشخیص بیماری، پرواز فضایی، رُجِسْتیک(اقدامات مربوط به تهیه و توزیع) و عیب‌یابی نرم‌افزار تمرکز می‌کنند. در این سیستم‌ها، برنامه‌نویسان، *واقعیّت‌ها* و *قوانین* را اضافه می‌کنند و سیستم، پردازه‌ی استنتاج را به صورت خودکار انجام می‌دهد و می‌توانند یا از زنجیره‌ی پیشرُو[۱] یا از زنجیره‌ی پَسرُو[۲] استفاده نمایند.

برخی از کاربردهای سیستم‌های خبره

این سیستم‌ها ممکن است برای طبقه‌بندی، تشخیص عیب، پیداکردن خرابی، تفسیر اطّلاعات، طرّاحی، پیکربندی و یا پیش‌بینی به کار روند؛ آنها شاید برای آموزش دادن، نمایش دادن، تحلیل، مشاوره، تجدید نظر یا کنترل به کار روند. استفاده‌ی تجاری سیستم‌های خبره شامل این موردها می‌باشند: یک سیستم استفاده شده برای مشاوره(ارائه‌ی نصیحت) در مورد وام خانه؛ یک سیستم استفاده شده به وسیله‌ی یک تولید کننده‌ی کامپیوتر، برای برّرسی اجرای کامل دستورات مشتریان؛ یک سیستم استفاده شده در بیمارستان، برای تفسیر اندازه‌گیری‌های ریّوی، جهت مشاهده‌ی علائم ناخوشی ریّه؛ یک سیستم استفاده شده به وسیله‌ی شیمیدان‌ها برای تفسیر توده‌ی داده‌های طیف‌سنج، برای کمک به پی‌بردن به ساختار مولکولی ترکیبات آلی(زیستی)؛ و یک سیستم برای کمک به زمین‌شناسان، برای ارزیابی مکان‌های معدنی، برای ذخایر.

---

۱ - این مطلب در فصل «استنتاج در منطق مرتبه‌ی اوّل» همین کتاب الکترونیکی توضیح داده شده است.

۲ - این مطلب در فصل «استنتاج در منطق مرتبه‌ی اوّل» همین کتاب الکترونیکی توضیح داده شده است.

چرا یک سیستم خبره را می‌سازیم؟

برای تکثیر خبره‌ی نادر(کمیاب) و پرهزینه؛ برای فرمول‌بندی دانش خبره؛ و برای جمع‌آوری منبع‌های دانش متمایز، یک سیستم خبره را می‌سازیم. دلایل ممکن دیگری هم وجود دارند؛ برخی از افراد دلیل می‌آورند که سیستم‌های خبره در مقایسه با انسان‌ها کم‌تر خطا می‌کنند، مناسب‌تر هستند و رُک‌تر هستند، یا در مقایسه با خبره‌های انسانی، بی‌طرف‌تر می‌باشند؛ البتّه سیستم‌های خبره عیب‌های زیادی هم دارند که باید استفاده از آنها را به دامنه‌های مشخّص، محدود نماییم. سیستم‌های خبره، دارای بینش، دلسوزی، فهم انگیزه‌ی انسانی، توانایی حدس زدن، معمولاً توانایی یادگیری، دانش قضاوت صحیح(عقل سلیم) اندکی می‌باشند.

ویژگی‌های خوب یک سیستم خبره

در صورتی که سیستم خبره به صورت محاوره‌ای باشد، یک رابط کاربری خوب، بسیار ضروری می‌باشد. توجّه کنید که مکالمه‌ای که سیستم خبره با شخص کاربر انجام می‌دهد باید به صورت «طبیعی» به وسیله‌ی کاربر مطرح شده باشد، که شامل موردهایی نظیر این‌ها می‌باشد: شیوه‌ی سؤالاتی که سیستم می‌پرسد باید طبیعی باشد؛ سؤالات احمقانه نباید وجود داشته باشند (کسانی که به سیستم جواب می‌دهند باید با دلیل تدبیر کرده باشند)؛ سیستم باید قادر باشد توضیح دهد که چرا یک سؤال را می‌پرسد و هر نتیجه‌ای که به آن می‌رسد را توجیه نماید و باید به کاربر در مورد سیستم اطمینان دهد.

در استدلال باید یک سیستم خبره قادر به انجام این موردها باشد: باید استنتاج‌هایش باورکردنی و نه الزاماً صحیح باشند؛
باید قادر به کار با دامنه‌ی دانش ناقص و موردهایی که اطّلاعات، ناقص هستند، باشد؛ اغلب، دانش و یا اطّلاعات، ناکامل می‌باشند و دانش و یا داده‌ها ممکن است قابل اعتماد نباشند (مثلاً ممکن است دارای خطا باشند) و شاید دانش و یا داده‌ها به صورت نادرست بیان شده باشند (به عنوان مثال، «اگر دارای دندان بلندی باشد، آنگاه این خطرناک می‌باشد.»)؛
همچنین باید سیستم، رقابت همزمان مفروضات را در نظر داشته باشد.

سیستم‌های خبره باید بتوانند به‌سادگی اطّلاعات جدید را یکی نمایند(تطبیق دهند) (یا از طریق مهندسی دانش بیش‌تر یا با استفاده از یادگیری ماشینی).

سیستم خبره‌ی قانون‌گرا[1]

تعریف- سیستم خبره‌ی قانون‌گرا، در علم کامپیوتر، سیستمی خبره است که براساس مجموعه‌ای از قانون‌ها که یک انسان خبره برای رسیدگی کردن به یک مسأله دنبال می‌کند، باشد.[2]

---

۱ - rule-based

۲ - http://www.answers.com/topic/rule-based-expert-system

روش‌های زیادی می‌توانند برای ساخت سیستم‌های خبره استفاده شوند. امّا اکثر سیستم‌های خبره که تاکنون ساخته شده‌اند، سیستم‌های قانون‌گرا می‌باشند؛ یعنی، پایگاه دانش آنها شامل واقعیّت‌ها و قانون‌ها می‌باشد.

پایگاه دانش

پایگاه دانش، شامل واقعیّت‌ها و قوانین می‌باشد. تصوّر کنید که ما در حال ساخت یک سیستم خبره‌ی قانون‌گرا برای تشخیص پزشکی باشیم؛ قانون‌ها، ارتباط‌های میان علائم و عبارت‌های پزشکی را کد خواهند کرد؛ واقعیّت‌ها، دانش را در مورد بیماری جاری، کد خواهند کرد؛ در سیستم‌های قانون‌گرا، [قانون‌ها،] اغلب به صورت «اگر... آنگاه...» نوشته می‌شوند؛ به صورت برابر می‌توانند به یکی از شکل‌هایی که برای موردهای زیر استفاده می‌شوند، نوشته شوند:

if p_1 AND p_2 … AND p_n then q

$(p_1 \wedge \ldots \wedge p_n) \Rightarrow q$

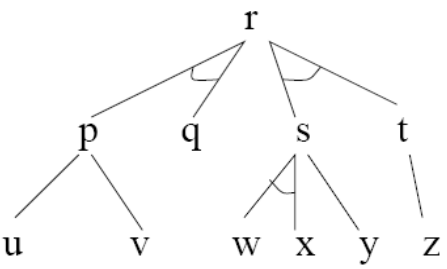
$q \vee \neg p_1 \vee \ldots \vee \neg p_n$

برای سیستم‌های قانون‌گرا استفاده از منطق گزاره‌ای زمانی که گزاره‌ها دارای هیچ آرگومانی نمی‌باشند، کاملاً معمولی است؛ بنابراین، در این موقع، متغّیرها و سمبل(نام)های تابعی، در واقعیّت‌ها و قانون‌ها وجود ندارند؛ در مثال‌های زیر خودمان را با این وضعیّت محدود نموده‌ایم. نتیجه‌ی یک قانون شاید یک قلم تجزیّه ناپذیر باشد که در قانون قبلی آمده است. می‌توانیم این زنجیره را به شکل گرافیکی، به صورت یک گراف AND-OR نمایش دهیم.

مثال- برای قانون‌های زیر:

if p AND q **then** r
if s AND t **then** r
if u **then** p
if v **then** p
if w AND x **then** s
if y **then** s
if z **then** t

گراف زیر را داریم:



مثال- در زیر مثالی از پایگاه دانشی برای شناسایی حیوان‌ها را می‌بینید:

if *animal*[۱]*Gives*[۲]*Milk*[۳] **then** *animalIsMammal*[۴]

if *animalHasHair*[۵] **then** *animalIsMammal*

if *animalIsMammal* AND *animalChews*[۶]*Cud*[۷] **then** *animalIsUngulate*[۸]

if *animalIsUngulate* AND *animalHasLongNeck*[۹] **then** *animalIsGiraffe*[۱۰]

if *animalIsUngulate* AND *animalIsStriped*[۱۱] **then** *animalIsZebra*[۱۲]

از این مثال در آینده استفاده خواهیم کرد.

موتور استنتاج

موتور استنتاج، استنتاج‌ها را با استفاده از دانش موجود در پایگاه دانش، استخراج می‌کند و این کار را با استفاده از استنتاج قانون‌گرا[۱۳] انجام می‌دهد؛ از یک نقطه نظر منطقی، اساساً از روش تحلیل[۱۴] استفاده می‌کند. یک چشم‌انداز(دورنمای) استنتاج(استدلال) قانون‌گرا این است که در حال انجام یک جستجوی گرافی AND-OR می‌باشد؛ به طور مؤثّر، ما به دنبال یک مسیر که ریشه و برگ‌ها را به هم متّصل می‌کند، می‌گردیم؛ در صورتی که یک گره، یک برگ باشد، در این صورت، این گره، یک واقعیّت را که باید درست باشد، نمایش می‌دهد.

---

۱ – حیوان

۲ – می‌دهد

۳ – شیر

۴ – پستاندار

۵ – مو، زُلف

۶ – می‌جَوَد

۷ – نُشخوار می‌کند؛ نشخوار: در دسته‌ای از حیوان‌ها، مثل گوسفند، بخش کوچکی از غذا، از معده‌ی اوّل حیوان، به دهان حیوان برمی‌گردد تا دوباره جویده شود. (Babylon> Babylon English-English)

۸ – سُم‌دار

۹ – گردن

۱۰ – زرّافه

۱۱ – راه‌راه یا خط‌دار

۱۲ – گورخَر

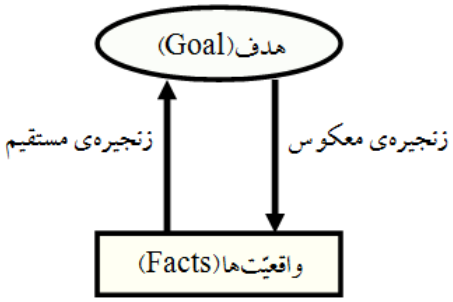
۱۳ – Rule-*B*ased *R*easoning(*RBR*)؛ **تعریف**– نوع خاصّی از استدلال است که از عبارت‌هایی به صورت «اگر – آنگاه – در غیر این صورت(If – Then - Else)» استفاده می‌کند.
(http://www.xs4all.nl/~synotix/robbieng/docs/inferenceengine.doc)

۱۴ – «روش تحلیل»، در فصل‌های «منطق گزاره‌ای» و «استنتاج در منطق مرتبه‌ی اوّل» همین کتاب الکترونیکی توضیح داده شده است.

در سیستم‌های خبره‌ی محاوره‌ای، فرض نمی‌کنیم که همه‌ی واقعیّت‌ها شناخته شده‌اند و در گذشته در پایگاه دانش وجود داشته‌اند. بنابراین، در زمانی که در تلاش برای این هستیم که ببینیم آیا یک گره‌ی برگ(واقعیّت) درست است یا نه، پایگاه دانش را برّرسی خواهیم نمود، اگر در پایگاه دانش نباشد، در این صورت از شخص کاربر پرسش می‌کنیم که آیا واقعیّت درست است یا نه؟؛ جواب کاربر می‌تواند به پایگاه دانش اضافه شود.

زنجیره‌ی پَسرُو[1]- زنجیره‌ی پسرو، استدلال به وجود آمده(مشتق شده) از هدف می‌باشند. در عبارت‌های گراف AND-OR، این نوع از استدلال از ریشه‌ی گراف شروع می‌شود و برای پیداکردن مسیری از ریشه به برگ‌ها تلاش می‌نماید.

زنجیره‌ی پیشرُو[2]- این روش، استدلال مشتق شده از داده‌ها هم نامیده می‌شود؛ در عبارات گراف AND-OR، این نوع از استدلال از برگ‌ها شروع می‌کند و تلاش می‌کند که مسیری را از برگ‌ها به طرف ریشه پیدا نماید.



برخی از سیستم‌ها منحصراً از یکی از روش‌های زنجیره‌ی پیشرو یا زنجیره‌ی پسرو استفاده می‌کنند. امّا تعداد زیادی از سیستم‌ها از هر دو روش استفاده می‌کنند، که این کار، خیلی طبیعی می‌تواند باشد؛ به عنوان مثال، به یک مشاوره با یک پزشک توجّه نمایید؛ بیمار برخی از علائم را تشریح می‌کند؛ نتایج با استفاده از زنجیره‌ی پیشرو به دست می‌آید (شاید فقط به طور آزمایشی). پزشک یک فرض را انتخاب می‌کند و با استفاده از زنجیره‌ی پسرو به علائمی می‌رسد که به وسیله‌ی بیمار بیان شده‌اند. بیمار دوباره صحبت می‌کند و شاید به پرسشی دیگر جواب دهد و یا از او اطّلاعات دیگری خواسته شود و این کار(پروسه) باز هم تکرار می‌شود.

۱ - این مطلب در فصل «استنتاج در منطق مرتبه‌ی اوّل» همین کتاب هم توضیح داده شده است.

۲ - این مطلب در فصل «استنتاج در منطق مرتبه‌ی اوّل» همین کتاب هم توضیح داده شده است.

توضیح‌های استدلال

در گذشته اهمّیّت داشتن توضیح روان(سلیس) را در یک سیستم خبره بیان کردیم. سیستم‌های قانون‌گرا معمولاً دو امکان را برای ارائه‌ی توضیحات به کاربران ارائه می‌کنند؛ کاربران ممکن است یا بپرسند، چرا؟ و یا بپرسند، چگونه؟؛ سیستم، این سؤالات را با نمایش برخی از قوانین مربوط جواب می‌دهد؛ برای مثال، تصوّر نمایید که سیستم خبره‌ی شناسایی حیوان‌ها، که در گذشته در همین فصل آن را بیان کردیم، از کاربر این پرسش را می‌پرسد که «آیا حیوان نُشخوار می‌کند؟»؛ در این مورد کاربر می‌تواند به جای جواب دادن به این سؤال، بپرسد که: «چرا شما این سؤال را می‌پرسید؟»؛ برای جواب دادن به این سؤال کاربر، سیستم خبره با نمایش یک گراف AND-OR پاسخ می‌دهد؛ برای مثال، ممکن است سیستم پاسخ دهد:

«من از شما پرسیدم که آیا حیوان نشخوار می‌کند؟، زیرا این در تشخیص اینکه حیوان جانور سُم‌دار است، کمک می‌کند؛ قبلاً تشخیص داده شده که جانور پستاندار است؛ بنابراین، اگر حیوان نُشخوار می‌کند، آنگاه حیوان سم‌دار می‌باشد؛ این در تشخیص اینکه آیا حیوان زرّافه است؟، کمک می‌کند؛ در صورتی که حیوان سم‌دار باشد و دارای گردن بلند باشد، زرّافه می‌باشد»؛

تصوّر نمایید که یک سیستم خبره تشخیص داده که برخی از گره‌ها درست هستند؛ در گفتن نتیجه به کاربر، کاربر می‌تواند بپرسد: «چگونه به این نتیجه رسیدی؟»؛ برای این کار، سیستم خبره با نمایش قسمت‌های موفّق گراف AND-OR جواب می‌دهد. ایده این است که برای توجیه کردن یک استنتاج، سیستم باید نشان دهد که کدام قانون‌ها در رسیدن به آن نتیجه اجرا می‌شوند. برای مثال، تصوّر کنید در موردی که سیستم تشخیص می‌دهد که حیوان یک جانور سُم‌دار می‌باشد، کاربر ممکن است درخواست کند که چگونه به این نتیجه رسیدی؟، و جواب سیستم ممکن است به صورت زیر باشد:

این قانون برای تشخیص اینکه آیا حیوان سم‌دار است، استفاده شده است: اگر حیوان پستاندار است و نشخوار می‌کند، سم‌دار می‌باشد.

این قانون برای تشخیص اینکه آیا جانور پستاندار است، استفاده شده است: حیوان دارای مو(زلف) می‌باشد، پس پستاندار می‌باشد.

شما به من گفتید که: حیوان دارای مو می‌باشد.

شما به من گفتید که: حیوان نشخوار می‌کند.

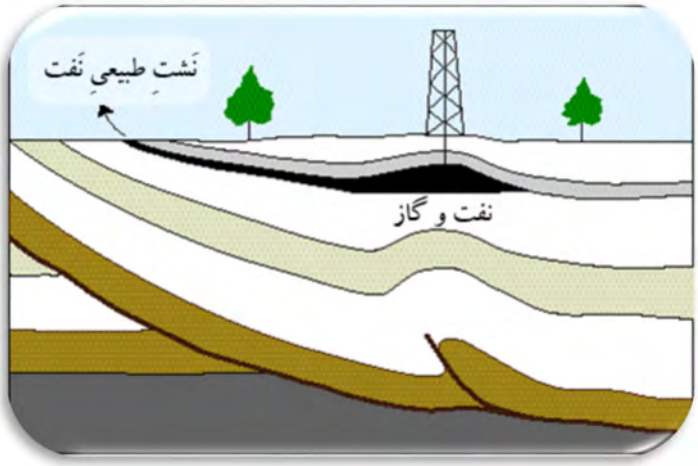
آشنایی با چند سیستم خبره

بیش‌تر پژوهش‌های انقلابی در سیستم‌های قانون‌گرا در مورد سیستمی به نام MYCIN انجام شده است. MYCIN برای استفاده‌ی یک پزشک برای تشخیص عفونت‌های باکتریایی خون طرّاحی شد و در حدود ۴۵۰ قانون دارد.[۱] بازده MYCIN و طبیعی بودن مکالمه‌ی آن با مشارکت قطعه‌های خیلی کوچک دانش که شاید در مدّت استنتاج به کار گرفته شوند، بهبود یافت.

---

۱ - در ویکی‌پدیا این تعداد در حدود ۶۰۰ تا ذکر شده است. (http://en.wikipedia.org/wiki/Mycin)

یکی دیگر از سیستم‌های قانون‌گرای خبره‌ی مشهور، PROSPECTOR[۱] (برای ارزیابی مکان‌های معدنی دارای استعداد برای استخراج) می‌باشد.



سیستم خبره‌ی دیگر، R1 (که XCON هم نامیده شده است)، برای برّرسی، کامل کردن و پیکربندی تقاضاهای تجهیزات کامپیوتری مشتری می‌باشد.

PUFF، یکی دیگر از سیستم‌های خبره می‌باشد که در پزشکی برای فهمیدن وضعیّت‌های تنفّسی از آن استفاده می‌شود:

---

۱ – در لغت به معنی «اکتشاف کننده، معدن‌یاب، معدن کاو» می‌باشد. (Babylon> hFarsi – advanced version)

بعد از برخی از آزمایش‌ها، با ساخت چند سیستم خبره‌ی قانون‌گرا، پژوهشگران هوش مصنوعی دریافتند که فقط پایگاه دانش، وابسته به دامنه می‌باشد. موتور استنتاج و رابط کاربر (نسبتاً) مستقل از دامنه بودند. به‌طورکلّی برای ساخت یک سیستم خبره‌ی جدید برای یک دامنه‌ی مختلف به یک پروسه‌ی مهندسی دانش[1] برای دریافت کردن قانون‌ها نیاز داریم.

۱ - Knowledge Engineering(*KE*)، **تعریف**- روشی است که مهندسان دانش از آن برای ساخت سیستم‌های هوشمند استفاده می‌کنند. (http://pages.cpsc.ucalgary.ca/~kremer/courses/CG/CGlecture_notes.html)

چکیده‌ی مطلب‌های فصل نوزدهم

سیستم‌های خبره، برنامه‌هایی طرّاحی شده برای مدلسازی و استدلال در مورد دانش انسان می‌باشند.

سیستم خبره‌ی قانون‌گرا، در علم کامپیوتر، سیستمی خبره است که براساس مجموعه‌ای از قانون‌ها که یک انسان خبره برای رسیدگی کردن به یک مسأله دنبال می‌کند، باشد.

برای سیستم‌های خبره‌ی قانون‌گرا، استفاده از منطق گزاره‌ای، زمانی که گزاره‌ها دارای هیچ آرگومانی نمی‌باشند، کاملاً معمولی است؛ بنابراین، در این موقع متغیّرها و سمبل(نام)های تابعی، در واقعیّت‌ها و قانون‌ها وجود ندارند.

یادآوری یا تکمیل مطلب‌های فصل نوزدهم

سؤال– ضعف‌های سیستم‌های خبره کدامند؟

جواب–

(**مشابه کنکور آزاد مهندسی کامپیوتر سال ۸۲**– درست جواب دادن آنها، به پایگاه دانش آنها بستگی دارد؛ خطا در قوانین آنها باعث خطا در تشخیص آنها می‌شود.)

تصحیح خودکار خطا؛ یادگیری، سخت است (گرچه تحقیقات یادگیری ماشینی می‌تواند این مورد را تغییر دهد).

استخراج دانش از یک خبره و کدکردن آن به شکل قابل استنباط به وسیله‌ی ماشین، سخت‌ترین قسمت پیاده‌سازی سیستم خبره است.[۱]

تست– MYCIN و XCON نمونه‌هایی از ..... هستند.

۱) سیستم‌های خبره

۲) منطق فازی

۳) روبوت‌ها

۴) شبکه‌های عصبی

جواب– گزینه‌ی «۱» درست است.[۲]

---

۱ – مطلب‌های درس «هوش مصنوعی» استاد، دکتر، «محمّدشوقی الحسن بعطوش»، دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی

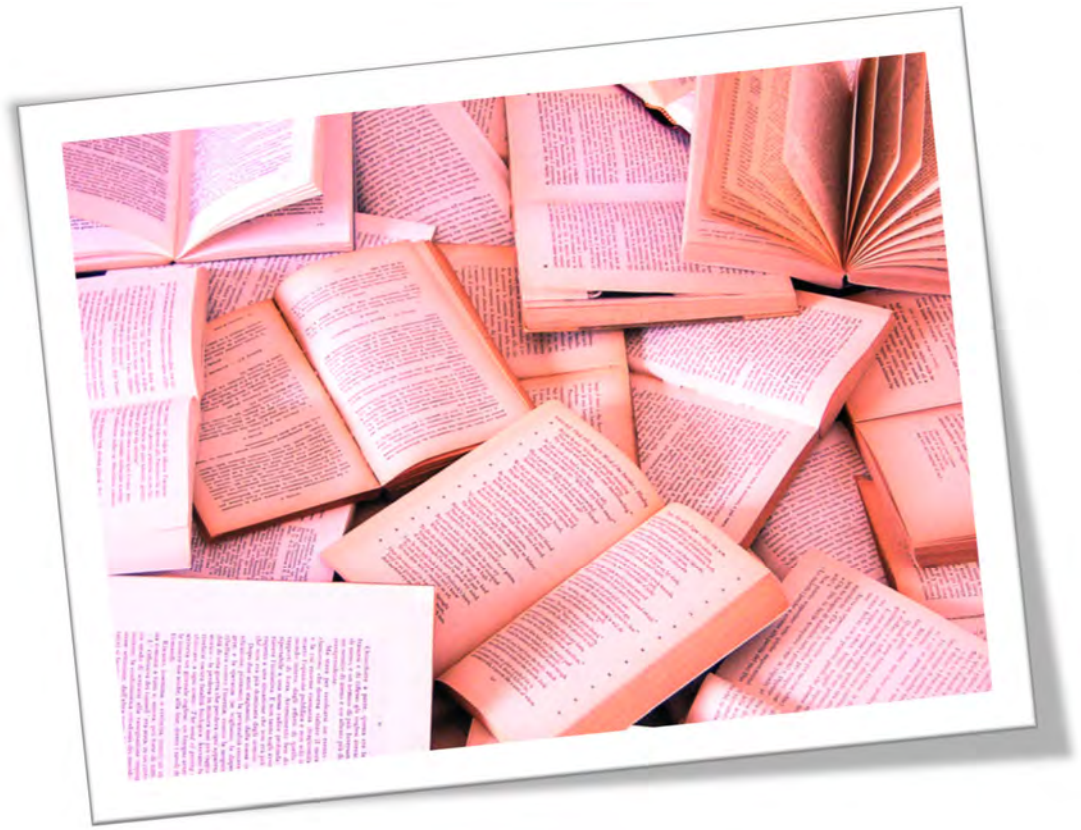
۲ – تست‌های(پرسش‌های چندگزینه‌ای) مربوط به «هوش مصنوعی» موجود در پایگاه اینترنتی http://quegrande.org، که این پایگاه اینترنتی مربوط به جامعه‌ای از زبان‌آموزان کشور اسپانیا می‌باشد، سال ۲۰۰۸ میلادی

فصل بیستم

۱ سیستم‌های طبقه‌بندی‌کننده[۲]

۱ - تصویر، دِیوید گُلدبِرگ(David E. Goldberg)، متولّد ۱۹۵۳ میلادی، استاد و دانشمند علوم کامپیوتر از کشور ایالات متّحده‌ی آمریکا می‌باشد؛ او به خاطر آثاری که در زمینه‌ی الگوریتم‌های ژنتیکی دارد، معروف می‌باشد.
(http://en.wikipedia.org/wiki/David_E._Goldberg)

۲ - Classifier Systems



فهرست برخی از عنوان‌های نوشته‌ها

سیستم‌های طبقه‌بندی‌کننده

تعریف– *طبقه‌بندی*، پردازه‌ی انتساب یک ورودی به یکی از کلاس‌های چندگانه می‌باشد.

تعریف گُلدبِرگ

تعریف دیوید گلدبرگ از سیستم طبقه‌بندی‌کننده چنین است: یک سیستم یادگیری ماشینی است که رشته قانون‌های دستوری ساده را برای راهنمایی عملکردش یاد می‌گیرد.

دِیوید گُلدبِرگ

سیستم‌های تولید[1]

تعریف– یک سیستم تولید، از تعداد زیادی قانون؛ حافظه‌ای که در آن واقعیّت‌ها قرار می‌گیرند؛ و یک الگوریتم که با استفاده از روش زنجیره‌ی پیشرو اجرا می‌شود و واقعیّت‌های جدید را با استفاده از قبلی‌ها به وجود می‌آورد، تشکیل شده است.

یک قانون، زمانی اجرا می‌شود که مجموعه‌ای از شرایط که در حافظه هستند، برقرار باشند.[2] سیستم‌های تولید، یک روش رایج در هوش مصنوعی هستند. در آنها از «اگر–آنگاه» یا قانون‌های «شرط–عملکرد» استفاده می‌شود. مثلاً در دنیای جاروبرقی داریم: در صورتی که خانه‌ی [۱و۱] جارو نشده باشد و در خانه‌ی مجاور یا همسایه باشد، آنگاه به خانه‌ی [۱،۱] برو. چنانچه بیان شد، یک سیستم تولید، از تعداد زیادی قانون تشکیل شده است؛ برخی از قوانین ممکن است باعث عمل کردن قانون‌های دیگر شوند؛ در سیستم‌های تجاری، مسأله این است که وقتی برای یک مورد بیش از یک قانون داریم، چه کار کنیم؟ [و کدام را اجرا نماییم؟]، در آینده به این مورد خواهیم پرداخت.

---

۱ – Production systems

۲ – Babylon> FOLDOC

اجزای یک سیستم طبقه‌بندی کننده

یک سیستم طبقه‌بندی کننده دارای سه جزء می‌باشد: یک قانون و سیستم پیام؛ یک سیستم انتساب اعتبار؛ و یک الگوریتم ژنتیکی، برای تولید قانون‌های جدید. حال این اجزا را مورد برّرسی قرار می‌دهیم:

قانون و سیستم پیام

حسگرهای عامل، اطّلاعات را که به صورت یک رشته‌ی بیتی، کد شده‌اند، دریافت می‌نمایند؛ این اطّلاعات، پیامی از محیط می‌باشد. این پیام، طبقه‌بندی‌کننده‌ها(قانون‌ها) را با تطبیق شرط‌ها فعّال می‌نماید. طبقه‌بندی‌کننده‌ها پیام‌هایشان را به لیست پیام ارسال می‌نمایند؛ این پیام‌ها ممکن است دیگر طبقه‌بندی‌کننده‌ها یا عمل‌کننده‌های عامل را فعّال نمایند.

مثال- فرض کنید سیستم ما دارای طبقه‌بندی‌کننده‌های زیر می‌باشد؛ توجّه کنید که قانون‌ها به وسیله‌ی ۰،۱ و #(*) به صورت یک رشته‌ی بیتی کد می‌شوند:

1. 01##: 0000
2. 00#0: 1100
3. 11##: 1000
4. ##00: 0001

اگر پیام ۰۱۱۱ از محیط دریافت شود؛ در ابتدا قانون یک اجرا می‌شود، پس ۰۰۰۰ در لیست پیام قرار می‌گیرد؛ سپس قانون‌های دو و چهار اجرا می‌شوند، در نتیجه ۱۱۰۰ و ۰۰۰۱ در لیست پیام قرار می‌گیرند؛ بعد قانون سه اجرا می‌شود، پس ۱۰۰۰ در لیست پیام قرار می‌گیرد که با قانون شماره‌ی ۴ مطابقت دارد که پیامش در لیست پیام‌ها قرار دارد، حالا چند پیام در لیست پیام قرار دارد؛ کدام پیام به عمل‌کننده‌ها[ی عامل] فرستاده می‌شود؟:

سیستم انتساب اعتبار

در زیر روشی را برّرسی می‌نمائیم:

روش گزینش براساس سابقه‌ی بهتر- در الگوریتم گزینش براساس سابقه‌ی بهتر، قانون‌ها براساس عملکرد قبلی انتخاب می‌شوند؛ به بیان دیگر، قانونی که دارای گذشته‌ی بهتری است، اجرا می‌شود؛ وقتی که یک قانون برقرار می‌شود، در یک «حراج(مزایده)[1]» [با قانون‌های دیگر] شرکت می‌نماید؛ هر قانون براساس عملکرد قبلی دارای یک توانایی(قوّت) می‌باشد و بالاترین قانون‌های شرکت کننده، بَرَنده می‌شوند و به طبقه‌بندی‌کننده(ها)ای که آنها را فعّال کرده(اند)، فرستاده می‌شوند.

الگوریتم ژنتیکی

تولید قوانین جدید- گزینش براساس سابقه‌ی بهتر، یک روش انتخاب قوانین و *انتساب اعتبار* را ارائه می‌نماید؛ حال سؤال این است که چگونه قوانین جدید را به دست بیاوریم؟؛ در *الگوریتم ژنتیکی* پایه‌ی ما، همه‌ی جمعیّت، در زمان t، در زمان

۱- auction

t+1 جایگزین می‌شود؛ این روش به‌خوبی برای بهینه‌سازی عمل می‌کند، ولی برای یادگیری، زیاد مناسب نمی‌باشد؛ به همین خاطر از روش گزینش بهترین‌ها(نخبه‌سالاری)[۱] برای حفظ برخی از قانون‌ها استفاده می‌نماییم؛ از روش انتخاب رولت[۲] هم برای حفظ قانون‌ها می‌توانید استفاده نمایید، البتّه این روش، کندتر استنتاج می‌کند.

۱ – elitism

۲ – roulette selection: این مورد در فصل «الگوریتم‌های ژنتیکی» همین کتاب الکترونیکی توضیح داده شده است.

چکیده‌ی مطلب‌های فصل بیستم

طبقه‌بندی، پردازه‌ی انتساب یک ورودی به یکی از کلاس‌های چندگانه می‌باشد.

سیستم طبقه‌بندی‌کننده، یک سیستم یادگیری ماشینی است که رشته قانون‌های دستوری ساده را برای راهنمایی عملکردش یاد می‌گیرد. (گُلدبِرگ)

یک سیستم تولید، از تعداد زیادی قانون؛ حافظه‌ای که در آن واقعیّت‌ها قرار می‌گیرند؛ و یک الگوریتم که با استفاده از روش زنجیره‌ی پیشرو اجرا می‌شود و واقعیّت‌های جدید را با استفاده از قبلی‌ها به وجود می‌آورد، تشکیل شده است.

در یک سیستم تولید، برخی از قوانین ممکن است باعث عمل کردن قانون‌های دیگر شوند؛ در سیستم‌های تجاری، مسأله این است که *وقتی برای یک مورد بیش از یک قانون داریم، چه کار کنیم؟ [و کدام را اجرا نماییم؟].*

یک سیستم طبقه‌بندی کننده دارای سه جزء می‌باشد: یک قانون و سیستم پیام؛ یک سیستم انتساب اعتبار؛ و یک الگوریتم ژنتیکی برای تولید قانون‌های جدید.

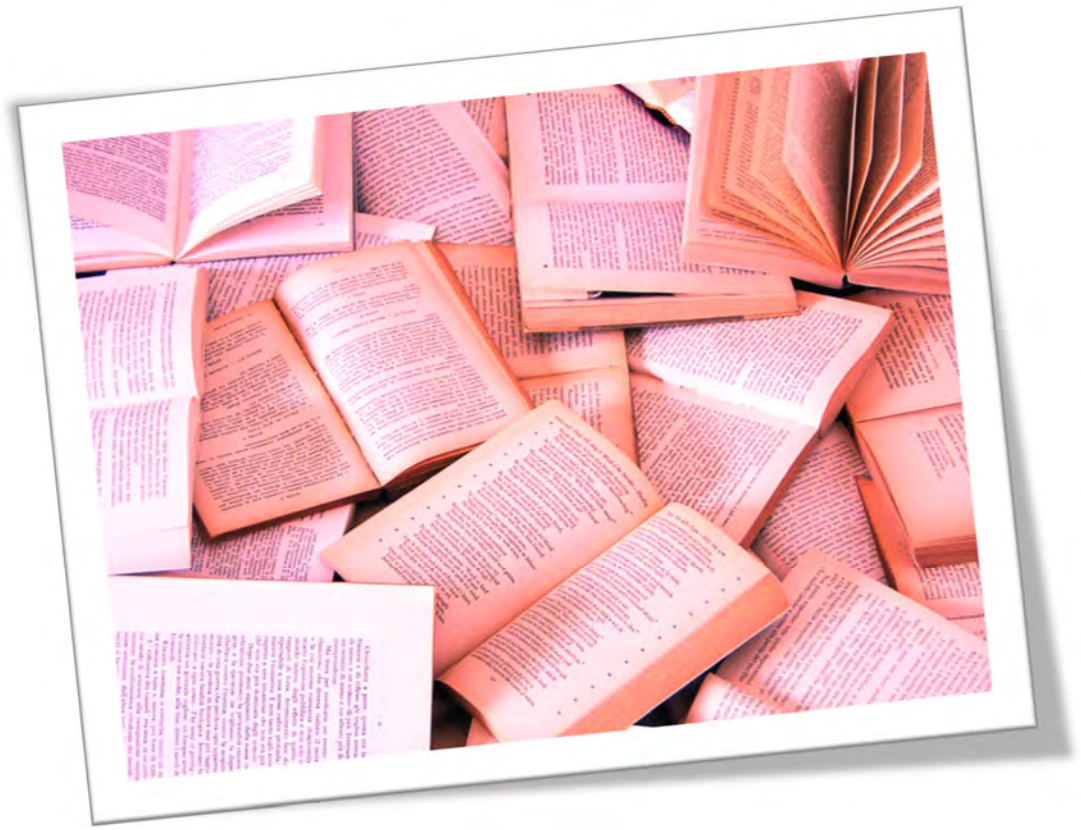
در الگوریتم گزینش براساس سابقه‌ی بهتر، قانون‌ها براساس عملکرد قبلی انتخاب می‌شوند؛ به بیان دیگر، قانونی که دارای گذشته‌ی بهتری است، اجرا می‌شود.

در الگوریتم ژنتیکی پایه‌ی ما، همه‌ی جمعیّت، در زمان t، در زمان $t+1$ جایگزین می‌شود؛ این روش به‌خوبی برای بهینه‌سازی عمل می‌کند، ولی برای یادگیری، زیاد مناسب نمی‌باشد؛ به همین خاطر از روش گزینش بهترین‌ها(نخبه‌سالاری) برای حفظ برخی از قانون‌ها استفاده می‌نماییم.

فصل بیست و یکم

یادگیری با استفاده از مشاهده‌ها[۱]؛ و درخت‌های تصمیم‌گیری[۲]

۱ - learning from observations
۲ - Decision trees



فهرست برخی از عنوان‌های نوشته‌ها

در گذشته فرض کردیم که دانش اوّلیّه به وسیله‌ی خبره‌ها به ما داده شده است و بر چگونگی استفاده از این دانش تمرکز کردیم. حال می‌خواهیم در این مورد صحبت کنیم که *چگونه دانش را از راه مشاهده به دست آوریم* و بر قانون‌های گزاره‌ای تمرکز می‌کنیم؛ مثلاً اگر هوا(وضعیّت جوّی) آفتابی و گرم باشد، آنگاه تنیس بازی کنید و [به صورت گزاره‌ای] می‌نویسیم:

$$\text{sunny} \wedge \text{warm} \rightarrow \text{PlayTennis}$$

یا اگر هوا(وضعیّت جوّی) خنک باشد و بارانی باشد یا باد شدید بوزد، آنگاه تنیس بازی نکنید و می‌نویسیم:

$$\text{cool} \wedge (\text{rain} \vee \text{strongWind}) \rightarrow \neg\text{PlayTennis}$$

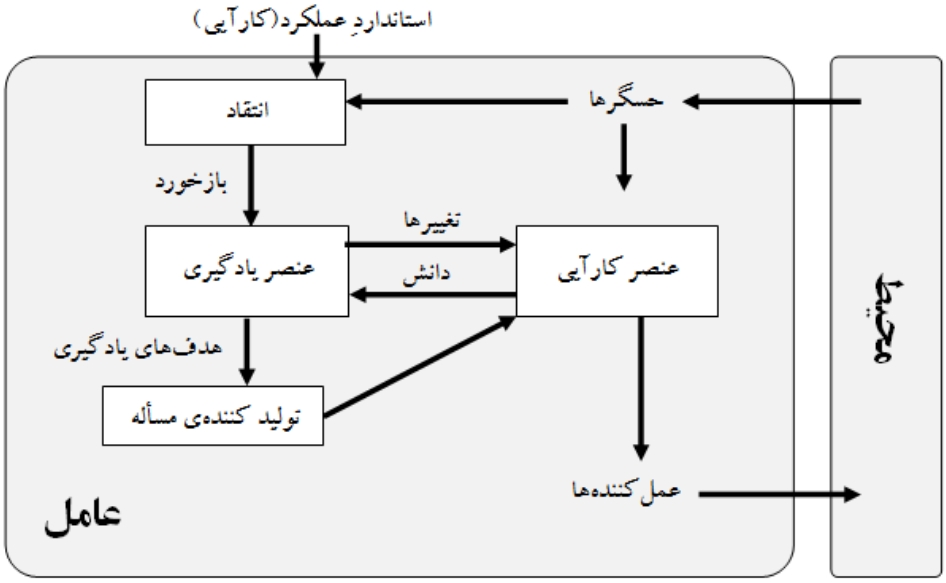
یادگیری

برای یک عامل، یادگیری به چه معنی می‌باشد؟:

تعریف– به این معنی است که عامل، دانش جدید را دریافت می‌نماید، دانش جدید را به کار می‌گیرد، رفتارش را تغییر می‌دهد و در یک کار معیّن، معیار کارآیی خود را بهبود می‌بخشد.

عامل‌های یادگیرنده[۱]

به یاد بیاورید که در گذشته در مورد عامل‌های یادگیرنده صحبت کردیم:



یک عامل یادگیرنده، دارای یک *عنصر کارآیی* و یک *عنصر یادگیری* می‌باشد. عنصر کارآیی، چیزی است که یک عامل برای تصمیم‌گیری در مورد اینکه چه کاری انجام دهد، از آن استفاده می‌کند و این چیزی است که تاکنون مطالعه کرده‌ایم. عنصر یادگیری، چیزی است که به عامل برای بازنگری عنصر کارآیی اجازه می‌دهد، این ممکن است به معنی اضافه نمودن یا تغییر قانون‌ها یا واقعیّت‌ها، بازنگری یک مکاشفه(ابتکار) و تغییر یک تابع جانشین[۲] باشد. یک عامل برای بازنگری کردن رفتارش، به اطّلاعاتی که به عامل بگوید چگونه به‌خوبی کارش را انجام می‌دهد، نیازمند می‌باشد، این اطّلاعات، بازخورد[۳] نام دارد.

انواع بازخورد

در اصل *سه نوع عملکرد یادگیری وجود* دارد که هر کدام بازخورد متفاوتی دارد:

---

۱ – این مورد در فصل «عامل‌های هوشمند» همین کتاب الکترونیکی توضیح داده شده است.

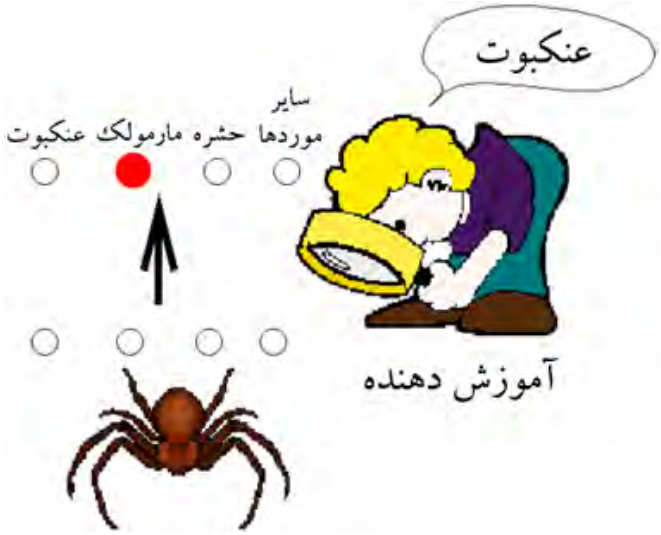
۲ – successor function؛ این مورد در فصل «حل مسأله و جستجو»ی همین کتاب الکترونیکی توضیح داده شده است.

۳ – feedback

یادگیری نظارت شده[۱](کنترل شده)

تعریف- یکی از عمومی‌ترین شکل‌های یادگیری می‌باشد؛ در این مورد یک منبع خارجی (که اغلب یاد(آموزش) دهنده[۲] نام دارد)، *عامل را با نمونه‌های برچسب زده شده[۳] ارائه می‌نماید، که باید از این داده‌ها برای تشخیص قانون‌های کلّی‌تر استفاده نماید؛*

مثلاً نمونه‌ها می‌توانند این موردها باشند: لیست بیماران و ویژگی‌ها؛ چه عامل‌هایی مرتبط با سرطان می‌باشند؟؛ چه عواملی فردی را دارای خطر می‌داند؟؛ بهترین سؤالات برای طبقه‌بندی حیوان‌ها چیست؟؛ صورت چه فردی یا چه جانوری در این تصویر می‌باشد؟؛ این پردازش یادگیری(به دست آوردن) قانون‌های کلّی از واقعیّت‌های مشخّص، *استنتاج*[۴] نام دارد. عامل موردهای(عملکردهای) معیّنی را در طول طبقه‌بندی‌اشان می‌بیند.



در شکل بالا وقتی که عامل عنکبوت را به اشتباه مارمولک می‌پندارد، آموزش دهنده این مطلب را به او می‌گوید.

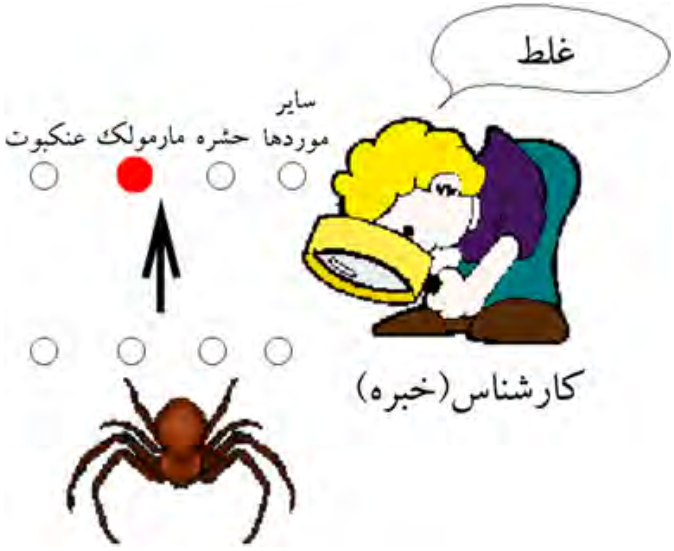
یادگیری بدون نظارت[۵](کنترل نشده)

تعریف- در این مورد آموزش دهنده‌ای برای ارائه‌ی نمونه‌ها وجود ندارد و عامل معمولاً برای پیداکردن الگوهایی در داده‌ها تلاش می‌نماید.

۱- supervised learning
۲- teacher
۳- labled examples
۴- induction
۵- unsupervised learning

تقویّت یادگیری(یادگیری تقویّتی)[1]

تعریف- یک نوع مخصوص از یادگیری است که در آن، عامل فقط درست بودن یا غلط بودن را برای انجام یک عمل دریافت می‌نماید و ممکن است «بهترین» عملکرد را برای انجام نداند.



در شکل بالا وقتی که عامل به اشتباه عنکبوت را مارمولک می‌پِندارد، کارشناس(خبره)، غلط بودن این پندار را به او می‌گوید و عامل دوباره خود را به‌روز کرده و تلاش می‌کند.

۱ - reinforcement learning

[۱] تعریف یک مسأله‌ی یادگیری

می‌توانیم مسأله‌ی یادگیری را با تخمین به صورت یک تابع f که به ما می‌گوید چگونه یک مجموعه از ورودی‌ها را طبقه‌بندی نماییم، تفسیر کنیم؛ یک مثال در این مورد یک مجموعه از ورودی‌های x و f(x) متناظر می‌باشد:

<<Mammal[۲],Eats-Meat[۳],Black-Stripes[۴],Tawny[۵]>,Tiger[۶]>

شکل بالا- بَبر

[۷]

درخت‌های تصمیم‌گیری

تعریف نخست: درخت تصمیم‌گیری، روشی(نموداری درختی) برای بیان تصمیم‌گیری‌های ترتیبی و نتیجه‌های ممکن از این تصمیم‌گیری‌ها است.[۱]

---

۱ - این قسمت دارای ارتباط بین رنگ‌ها است.

۲ - یعنی: «پستاندار»

۳ - یعنی: «گوشت می‌خورد»

۴ - یعنی: «دارای نوارهای(خط‌های) سیاه است»

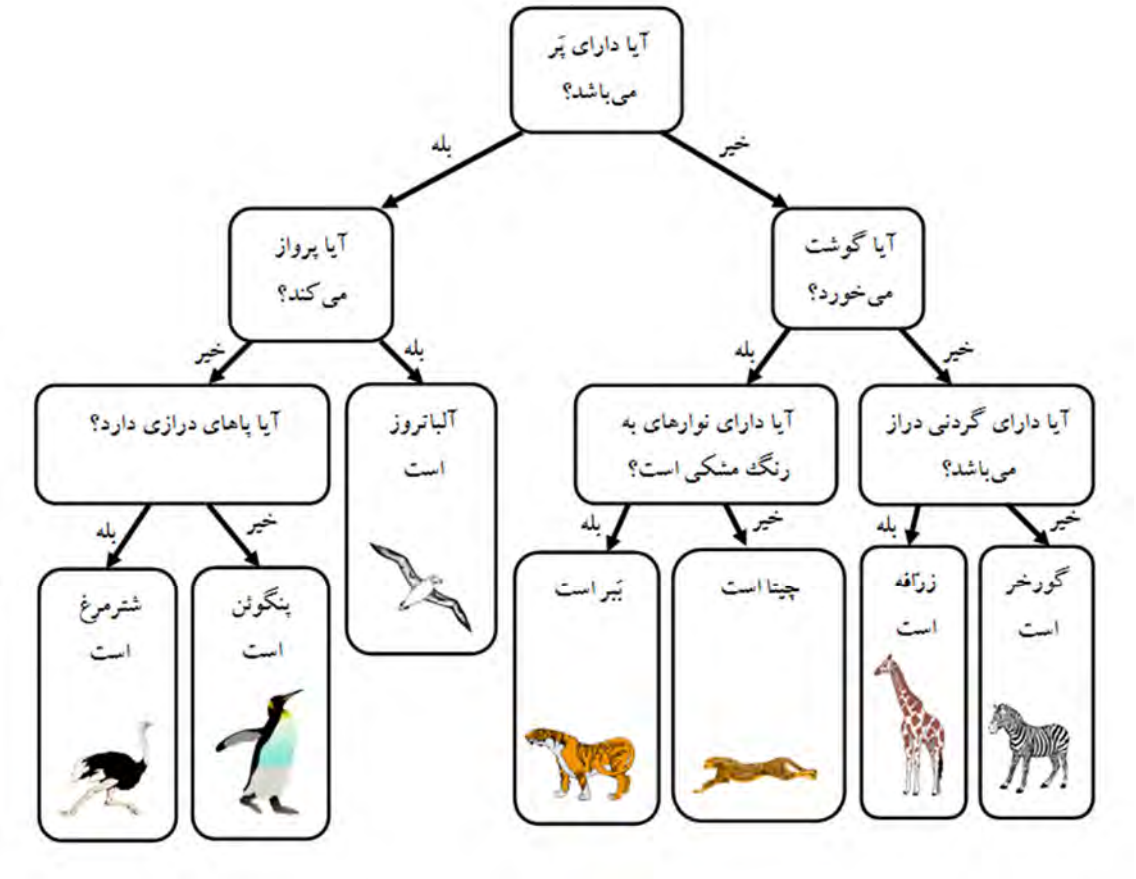
۵ - یعنی: «دارای رنگ زرد مایل به قهوه‌ای است»

۶ - یعنی: «بَبر»

۷ - پایان ارتباط بین رنگ‌ها

تعریف دوّم: درخت تصمیم‌گیری، نموداری درختی است که برای تصمیم‌گیری، در تجارت یا برنامه‌نویسی کامپیوتر مورد استفاده قرار می‌گیرد و در آن، گره‌ها(گزینه‌ها) با در نظر گرفتن ریسک‌ها، هزینه‌ها، نتیجه‌ها یا احتمال‌ها بیان می‌شوند.[۲]

در هر گره در درخت، یک ویژگی آزمایش(تست) می‌شود. در زیر یک درخت تصمیم‌گیری را می‌بینید:[۳]



۱- Babylon> Campbell R. Harvey's HyperTextual Finance Glossary

۲- Babylon> Merriam-Webster Collegiate® Dictionary

۳- در متن ترجمه نشده‌ی اینترنتی‌ای که این قسمت از کتاب با استفاده از آن ترجمه شده است، شکل‌های حیوان‌ها وجود نداشته است.

چکیده‌ی مطلب‌های فصل بیست و یکم

برای یک عامل، یادگیری به این معنی است که عامل، دانش جدید را دریافت می‌نماید، دانش جدید را به کار می‌گیرد، رفتارش را تغییر می‌دهد و در یک کار معیّن، معیارکارآیی خود را بهبود می‌بخشد.

یک عامل برای بازنگری کردن رفتارش، به اطّلاعاتی که به عامل بگوید چگونه به‌خوبی کارش را انجام می‌دهد، نیازمند می‌باشد، این اطّلاعات، بازخورد نام دارد.

سه نوع عملکرد یادگیری وجود دارد که هر کدام بازخورد متفاوتی دارد: (**مشابه کنکور آزاد مهندسی کامپیوتر سال ۸۱**- یادگیری نظارت شده، یادگیری بدون نظارت و تقویّت یادگیری.)

درخت تصمیم‌گیری، نموداری درختی برای بیان تصمیم‌گیری‌های ترتیبی و نتیجه‌های ممکن از این تصمیم‌گیری‌ها است.

یادآوری یا تکمیل مطلب‌های فصل بیست و یکم

تعریف- یادگیری نظارت(کنترل) شده را تعریف کنید.

جواب- در این گونه، عامل، زوج‌های ورودی- خروجی را مشاهده می‌کند و یاد می‌گیرد که ورودی را به خروجی نگاشت کند.[۱]

تعریف- یادگیری نظارت(کنترل) نشده را تعریف کنید.

جواب- در این گونه، عامل، الگوهای در ورودی را بدون بازخورد صریح یاد می‌گیرد.[۲]

درست یا غلط- یک درخت تصمیم‌گیری می‌تواند هر تابع بولّینی را یاد بگیرد و بیان کند.

جواب- «درست» است.[۳]

---

۱ - کوییز شماره‌ی ۴ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، زمستان سال ۲۰۱۲ میلادی

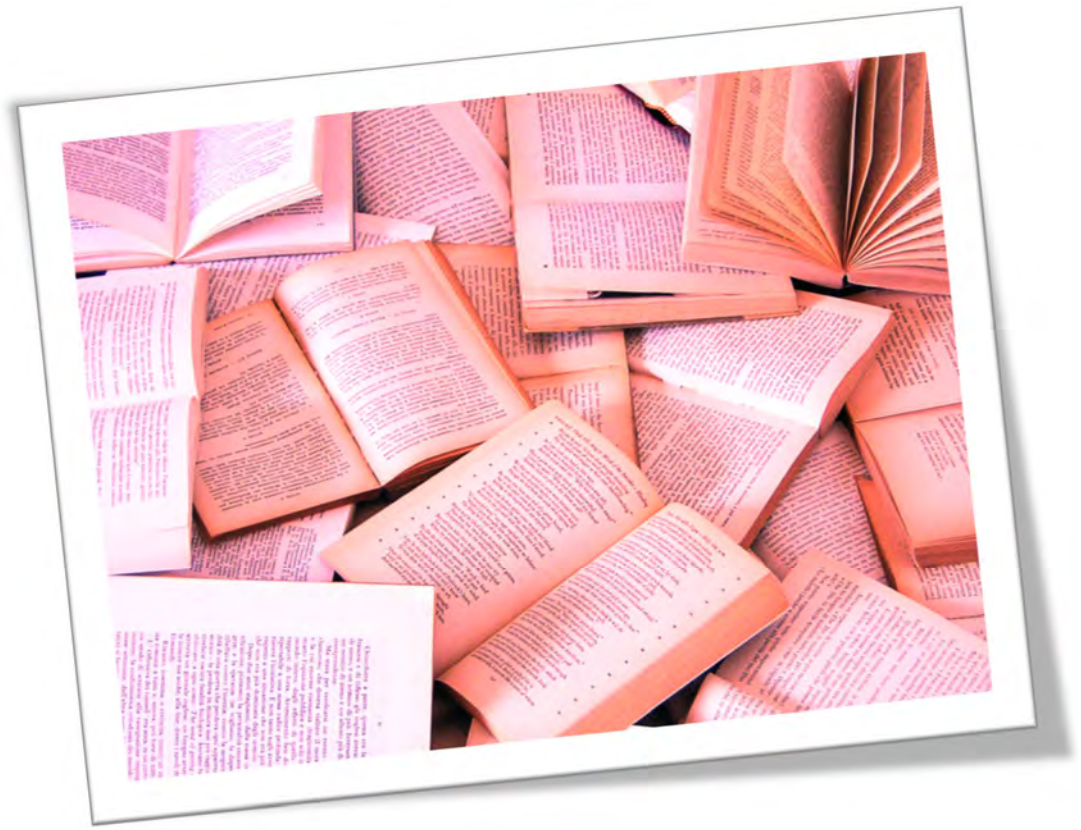
۲ - کوییز شماره‌ی ۴ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، زمستان سال ۲۰۱۲ میلادی

۳ - کوییز شماره‌ی ۴ درس «آشنایی با هوش مصنوعی» استاد، «ریچارد اچ. لاترپ»، دانشکده‌ی شهر اروین دانشگاه ایالت کالیفورنیای کشور آمریکا، پاییز سال ۲۰۱۱ میلادی

فصل بیست و دوّم

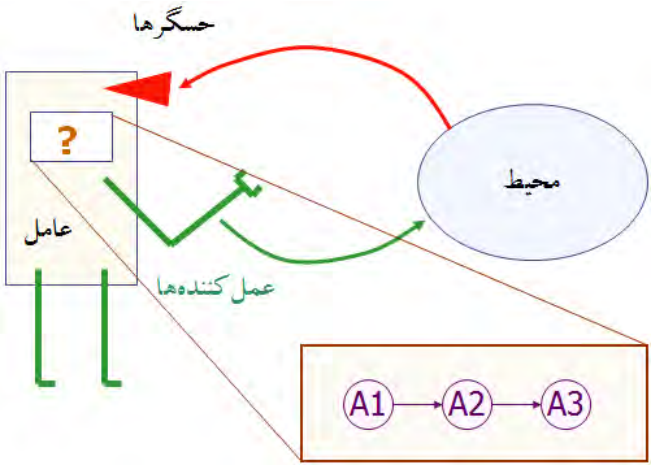
برنامه‌ریزی[۱]

۱ - planning



فهرست برخی از عنوان‌های نوشته‌ها

عامل برنامه‌ریزی



برنامه‌ریزی

تعریف– برنامه‌ریزی، تعبیه کردن یک رشته از عملکردها برای دسترسی به یک هدف می‌باشد.

محیط برنامه‌ریزی کلاسیک[۱]، کاملاً قابل مشاهده، قطعی، محدود، ثابت[۲] و گسسته است. زبان مشخّصی برای ارائه‌ی مسأله‌های برنامه‌ریزی کلاسیک وجود دارد(STRIPS) که در ادامه‌ی همین فصل به بررّسی آن می‌پردازیم.

---

۱ – classical planning

۲ – static

مقایسه‌ی برنامه‌ریزی و حلّ مسأله[۱]

برخی از موردها عبارتند از: روش‌های برنامه‌ریزی و حلّ مسأله اغلب می‌توانند گونه‌های یکسانی از مسأله‌ها را حل کنند. برنامه‌ریزی به خاطر استفاده از نمایش‌ها و روش‌هایی که استفاده می‌کند، قدرتمندتر از روش حلّ مسأله است. [در برنامه‌ریزی،] وضعیّت‌ها، هدف‌ها و عملیّات، به مجموعه‌هایی از عبارت‌ها که معمولاً به صورت منطق مرتبه‌ی اوّل[۲] هستند، تجزیّه می‌شوند.

زبانی برای مسأله‌های برنامه‌ریزی

حل کننده‌ی مسأله‌ی مؤسّسه‌ی تحقیق اِستَنفورد(اِستِریپس)[۳]: در هوش مصنوعی، استریپس نامی برای *برنامه‌ریزی کننده‌ای خودکار*[۴] است که به وسیله‌ی ریچارد فایْکْز[۵] و نیلز نیلسون[۶] در سال ۱۹۷۱ میلادی به وجود آمد. استریپس همچنین نامی برای زبانی است که بعداً برای ارائه‌ی ورودی‌های این برنامه‌ریز به وجود آمده است. این زبان بهترین زبانی است که برای ارائه‌ی مسأله‌های برنامه‌ریزی کلاسیک وجود دارد و پایه‌ای برای اکثر زبان‌هایی است که امروزه برای بیان *برنامه‌ریزی خودکار*[۷] مورد استفاده قرار می‌گیرند. در اینجا ما فقط زبان استریپس را مورد برّرسی قرار می‌دهیم و برنامه‌ریز استریپس را برّرسی نمی‌کنیم.[۸]

یک نمونه‌ی استریپس از موردهای زیر تشکیل شده است:

- یک **وضعیّت اوّلیّه؛**
- تعیین **وضعیّت‌های هدف؛** که وضعیّت‌هایی هستند که برنامه‌ریز در تلاش برای رسیدن به آنهاست؛
- یک مجموعه از **عملکردها؛** که برای هر عملکرد موارد زیر وجود دارد:

---

۱ - برای آگاهی‌های بیش‌تر در مورد روش «حلّ مسأله»، به فصل «حلّ مسأله و جستجو»ی همین کتاب الکترونیکی مراجعه کنید.

۲ - برای آگاهی‌های بیش‌تر در این مورد به فصل «منطق مرتبه‌ی اوّل» همین کتاب الکترونیکی مراجعه کنید.

۳- *ST*anford *R*esearch *I*nstitute *P*roblem *S*olver(*STRIPS*)

۴ - automated planner

۵ - Richard Fikes، متولّد ۱۹۴۲ میلادی، دانشمند علوم کامپیوتری و در حال حاضر استاد بازنشسته‌ی دانشکده‌ی علوم کامپیوتر دانشگاه استنفورد می‌باشد. (http://en.wikipedia.org/wiki/Richard_Fikes) تصویری از او را در زیر می‌بینید:

(http://engineering.stanford.edu/research/layout.php?sunetid=fikes)

۶ - Nils John Nilsson؛ در مورد این استاد بزرگ در فصل «آشنایی با هوش مصنوعی» همین کتاب الکترونیکی صحبت شده است.

۷ - automated planning

۸ - http://en.wikipedia.org/wiki/STRIPS

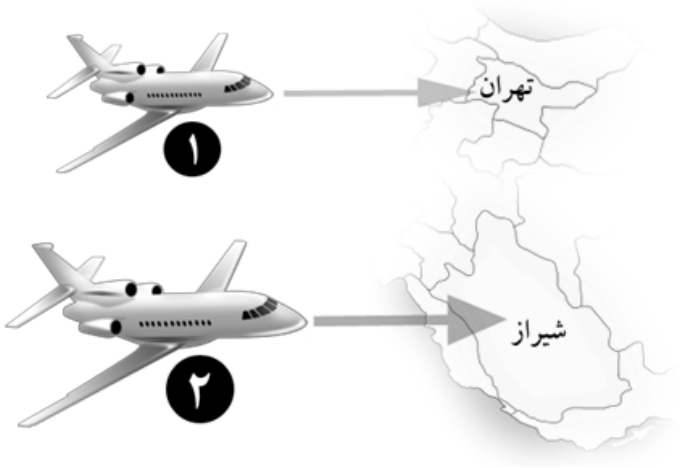
پیش‌شرط‌ها[۱]، مواردی هستند که باید قبل از اینکه عمل انجام شود برقرار باشند؛

پس‌شرط‌ها[۲]، وضعیّت(حالت)ی هستند که بعد از اینکه عمل انجام شد باید برقرار باشند.

وضعیّت

می‌تواند به صورت گزاره‌ای؛ به عنوان مثال، $Happy \wedge Hungry$ برای نمایش وضعیّت عامل باشد و یا به صورت منطق مرتبه‌ی اوّل باشد؛ به عنوان مثال،

$At^{3}(Plane^{4}_{1},Tehran) \wedge At(plane_2,Shiraz)$



فرض: هر گزاره‌ای که نام برده نشده، غلط می‌باشد.

هدف

وضعیّتی است که به صورت جزئی مشخّص شده است؛ به عنوان مثال،

$At(Plane_1,Tehran) \wedge At(plane_2,Shiraz)$

هدف $At(plane_2,Shiraz)$ را برآورده می‌نماید.

---

۱ - preconditions

۲ - postconditions

۳ - در لغت یعنی: «دَر(برای مکان)»

۴ - در لغت یعنی: «هواپیما»

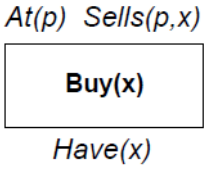
عملکردهای استریپس

عملکردها به وسیله‌ی موردهای زیر مشخّص می‌شوند:

پیش‌شرط‌ها: زمانی را مشخّص می‌کنند که عملکرد می‌تواند به کار گرفته شود.

پس‌شرط‌ها(اثرات): وضعیّت به وسیله‌ی عملکردها تغییر می‌یابد. پس‌شرط‌ها، وضعیّت(حالت)ی هستند که بعد از اینکه عمل انجام شد باید برقرار باشند.

مثلاً در برنامه‌ریزی عملکرد Buy[1](x)، به صورت زیر:



پیش‌شرط، $At(p) \wedge Sells^{2}(p,x)$ می‌باشد و پس‌شرط(اثر)، $Have^{3}(x)$ است و باید به این نکته توجّه کنیم که هیچ اطّلاعاتی در مورد چگونگی اجرای عملکرد(Buy(x)) وجود ندارد!.

مثال- خرید

عملکردها عبارتند از:

Buy(x):

پیش‌شرط: At(store[4]) و Sells(store,x)(فروشگاه، x را می‌فروشد)

پس‌شرط(اثر): Have(x)

Go(x,y)(از مکان x به مکان y برو)

پیش‌شرط: At(x)

پس‌شرط(اثر): At(y) و ¬At(x)(در مکان x نباشیم)

وضعیّت اوّلیّه(شروع):

---

۱ - یعنی: «خریدن»

۲ - یعنی: «می‌فروشد»

۳ - یعنی: «داشتن»

۴ - یعنی: «فروشگاه»

At(Home[۱]) ∧ Sells(SM[۲],Milk) ∧ Sells(SM,Banana[۳]) ∧ Sells(HWS[۴],Drill[۵])

وضعیّت هدف:

Have(Milk) ∧ Have(Banana) ∧ Have(Drill)

برنامه‌ریز با مرتبه‌ی جزئی[۶]

مطلب مهم:

برنامه‌ریزی‌ای است که در آن برخی از مرحله‌ها به ترتیب هستند؛ مثلاً در پوشیدن جورابِ چپ و جوراب راست، ترتیب اهمّیّتی ندارد، به عبارت دیگر، اینکه اوّل جورابِ راست را بپوشیم یا اوّل جورابِ چپ را بپوشیم، مهم نیست؛ ولی در پوشیدن جورابِ راست و کفشِ راست، ترتیب مهم است، به عبارت دیگر، اوّل نمی‌توان کفشِ راست را پوشید و سپس جوراب راست را پوشید!؛ در پوشیدن جورابِ راست، کفشِ چپ، و کفشِ راست، ترتیب در برخی از عملکردها مهم است.

---

۱ - یعنی: «خانه»

۲ - در اینجا نامی برای یک فروشگاه است.

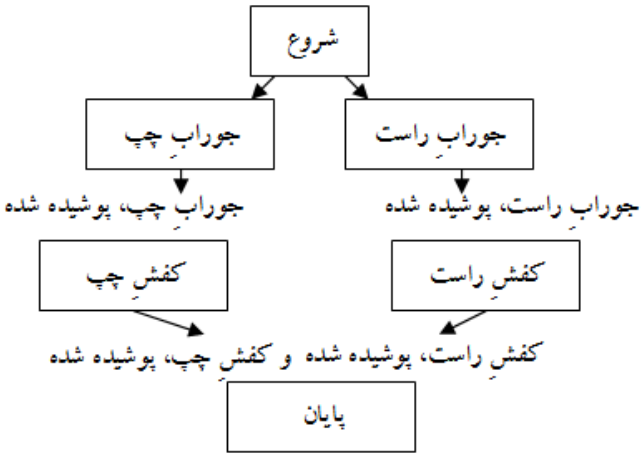
۳ - یعنی: «موز»

۴ - در اینجا نامی برای یک فروشگاه است.

۵ - یعنی، «دستگاه سوراخ کننده»:

۶ - partial-order planner

مثال- برنامه‌ریز با مرتبه‌ی جزئی:



برنامه‌ریز با مرتبه‌ی کلّی[1]

در این روش از لیست‌های ساده‌ای از مرحله‌ها استفاده می‌کنیم.

مثال- برنامه‌ریز با مرتبه‌ی کلّی:

۱- total-order planner

چکیده‌ی مطلب‌های فصل بیست و دوّم

برنامه‌ریزی، تعبیه کردن یک رشته از عملکردها برای دسترسی به یک هدف می‌باشد.

زبان مشخّصی به نام اِستریپس(STRIPS) برای ارائه‌ی مسأله‌های برنامه‌ریزی کلاسیک وجود دارد.

یک نمونه‌ی استریپس، از وضعیّت(های) اوّلیّه؛ وضعیّت(های) هدف؛ و عملکرد(ها) تشکیل شده است؛ در ضمن، برای هر عملکرد، پیش‌شرط(ها) و پس‌شرط(هایی) وجود دارد.

برنامه‌ریز با مرتبه‌ی جزئی، برنامه‌ریزی‌ای است که در آن برخی از مرحله‌ها به ترتیب هستند.

در برنامه‌ریز با مرتبه‌ی کلّی، از از لیست‌های ساده‌ای از مرحله‌ها استفاده می‌کنیم.

یادآوری یا تکمیل مطلب‌های فصل بیست و دوّم

تعریف– «برنامه‌ریز با مرتبه‌ی جزئی» را تعریف کنید.

جواب– در این گونه، گام‌های برنامه‌ریزی، در طول پردازه‌ی برنامه‌ریزی کاملاً به ترتیب نیستند.[۱]

درست یا غلط– «جستجو» حالت خاصّی از «برنامه‌ریزی» است.

جواب– **«درست» است.**[۲]

مطلب– سیستم‌های برنامه‌ریزی، الگوریتم‌های حلّ مسأله‌ای هستند که روی نمایش‌های گزاره‌ای یا رابطه‌ای[۳] وضعیّت‌ها و عملکردها کار می‌کنند. این نمایش‌ها به دست آوردن مکاشفه‌های مؤثّر و توسعه‌ی الگوریتم‌های نیرومند و انعطاف‌پذیر را برای حلّ مسأله‌ها ممکن می‌سازند.[۴]

---

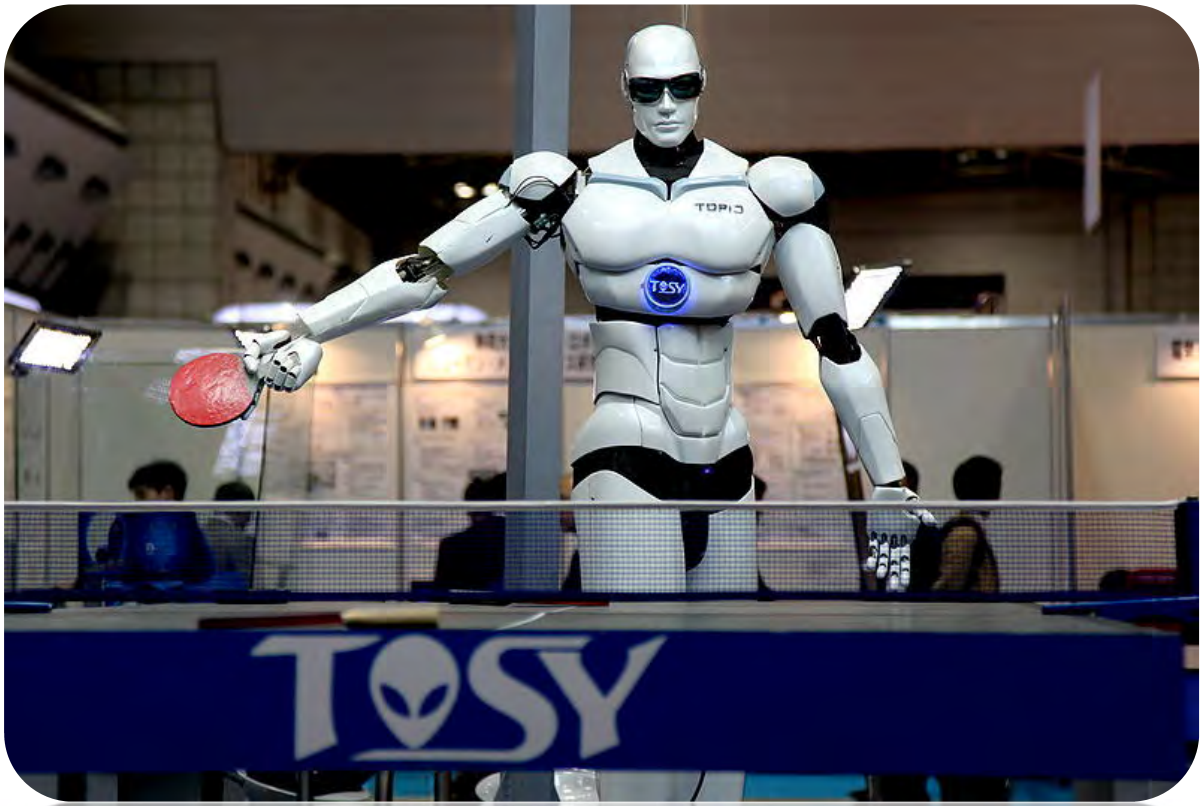
۱ – آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، پاییز سال ۱۹۹۱ میلادی

۲ – آزمون پایان‌ترم درس «آشنایی با هوش مصنوعی» استاد، «استوارت راسل»، دانشگاه ایالت کالیفرنیا، شهر برکلی کشور آمریکا، پاییز سال ۱۹۹۷ میلادی

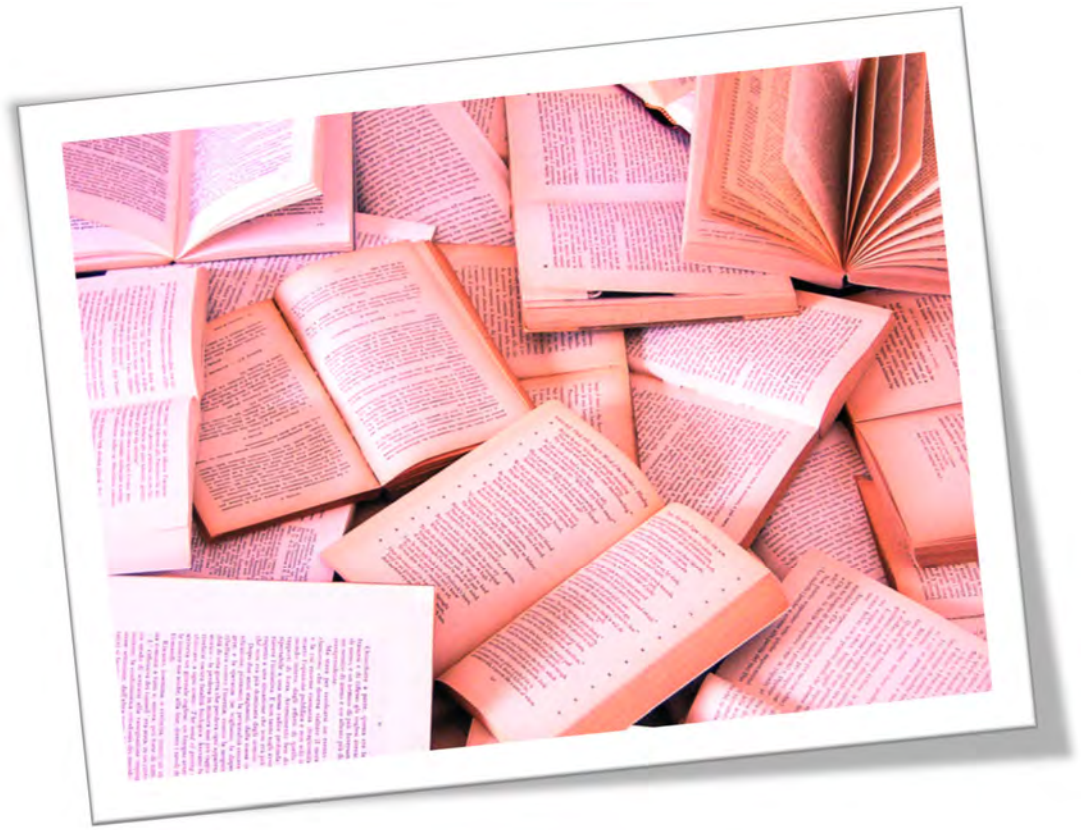
۳ – relational

۴ – کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل دهم، برنامه‌ریزی کلاسیک(Classical Planning)، صفحه‌ی ۳۹۳

فصل بیست و سوّم



آشنایی با روبوتیک



فهرست برخی از عنوان‌های نوشته‌ها

طرز کار(مکانیزم) روبوت

حسگرها

عمل‌کننده‌ها

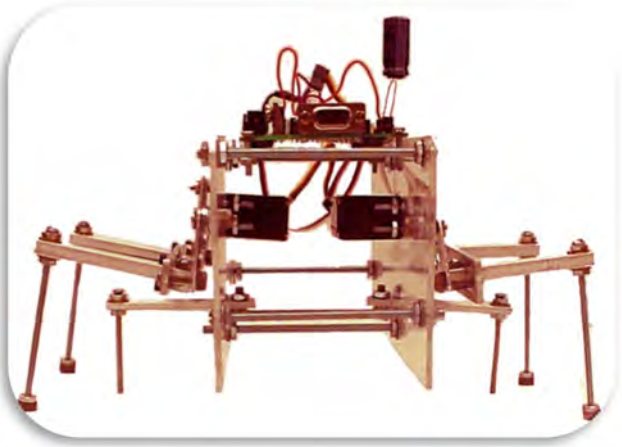
کنترل کننده

سخت‌افزار کنترل کننده

صنایعی که از روبوت‌ها استفاده می‌کنند

روبوت‌های صنعتی چه کارهایی می‌توانند انجام دهند؟

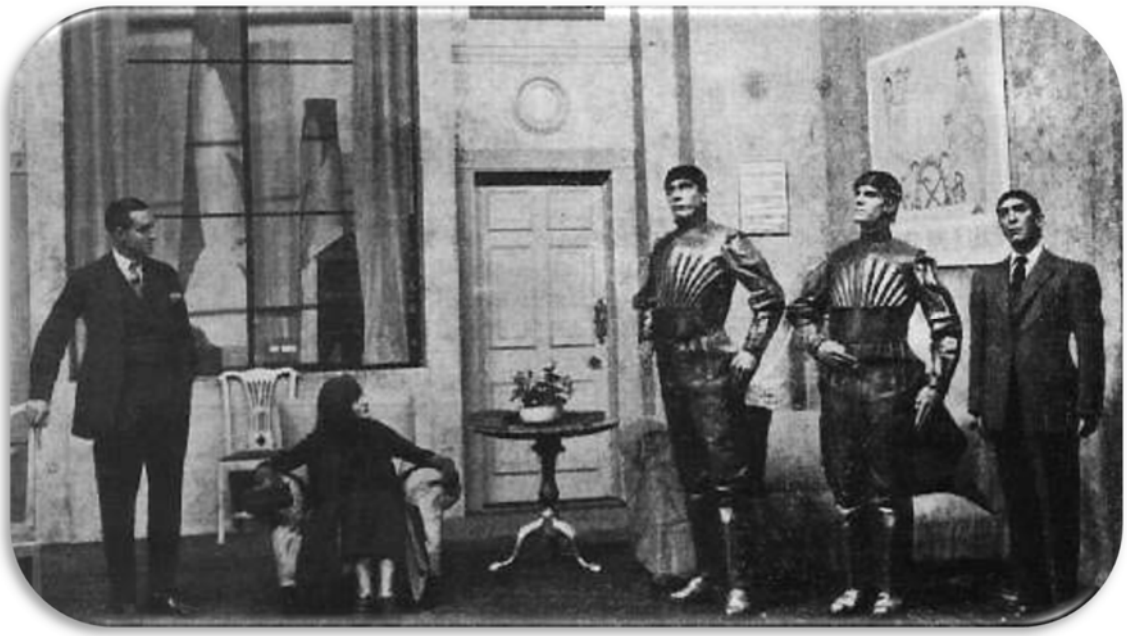
چند نمونه روبوت:



مقدّمه

کِرُل کاپِک

کلمه‌ی رُوبُوت به وسیله‌ی یک رمان نویس کشور چِک[۱] به نام کِرُلْ کاپِک[۲] در سال ۱۹۲۰ میلادی در یک نمایش با عنوان *R*assum's *U*niversal *R*obots(*RUR*) به کار گرفته شد؛ این واژه(روبوت) از کلمه‌ی چِکی روبوتا[۳] که در زبان چکی واژه‌ای برای «کار اجباری، بیگاری، کارگر یا پیشخدمت» می‌باشد، گرفته شده است.



شکل بالا– صحنه‌ای از نمایش *R*assum's *U*niversal *R*obots(*RUR*) کِرُل کاپِک در سال ۱۹۲۰ میلادی؛ در این تصویر، در سمت راست، سه روبوت نشان داده شده است.

---

۱ – Czech

۲ – Karel Capek؛ ۱۹۳۸ – ۱۸۹۰ میلادی (http://en.wikipedia.org/wiki/Karel_Capek)

۳– robota

تعریف‌های روبوت

تعریف اوّل– هر ماشین ساخته شده به وسیله‌ی یکی از اعضای ما.☺[۱]

تعریف دوّم– یک روبوت، یک دست ماشینی قابل برنامه‌ریزی مجدّد و چندکاربردی طرّاحی شده، برای حرکت دادن مواد، قطعه‌ها، ابزار یا وسایل مخصوص، با استفاده از حرکت‌های برنامه‌ریزی شده‌ی متغیّر و برای کاربردهای مختلف می‌باشد.[۲]

تعریف سوّم– وسیله‌ای مکانیکی برای انجام کاری که به وسیله‌ی انسان انجام می‌شود، می‌باشد، مثل رَنگ کردن اتومبیل.[۳]

تعریف چهارم– وسیله‌ای مکانیکی کامپیوتری برای انجام کارهای پیچیده.[۴]

تعریف پنجم– هر ماشینی که به صورت خودکار و به جای انسان کار می‌کند، گرچه ممکن است در ظاهر یا شیوه‌ی انجام کارها شبیه انسان نباشد.[۵]

تعریف ششم– یک روبوت، یک عامل هوشمند مکانیکی است که می‌تواند خودش کارها را انجام دهد یا می‌تواند برای انجام کارها راهنمائی شود. در عمل، یک روبوت معمولاً یک ماشین الکترومکانیکی است که به وسیله‌ی کامپیوتر و برنامه‌نویسی الکترونیکی راهنمائی می‌شود. روبوت‌ها می‌توانند خودمختار یا نیمه‌خودمختار باشند و دارای دو نوع اصلی هستند:

یکی، آنهایی که برای پژوهش در زمینه‌ی سیستم‌های شبیه انسان[۶] استفاده می‌شوند، مثل روبوت‌های ASIMO و TOPIO

---

۱ – مؤسّسه‌ی روبوت آمریکا(Robot Institute of America)

۲ – مؤسّسه‌ی روبوت آمریکا، ۱۹۷۹ میلادی

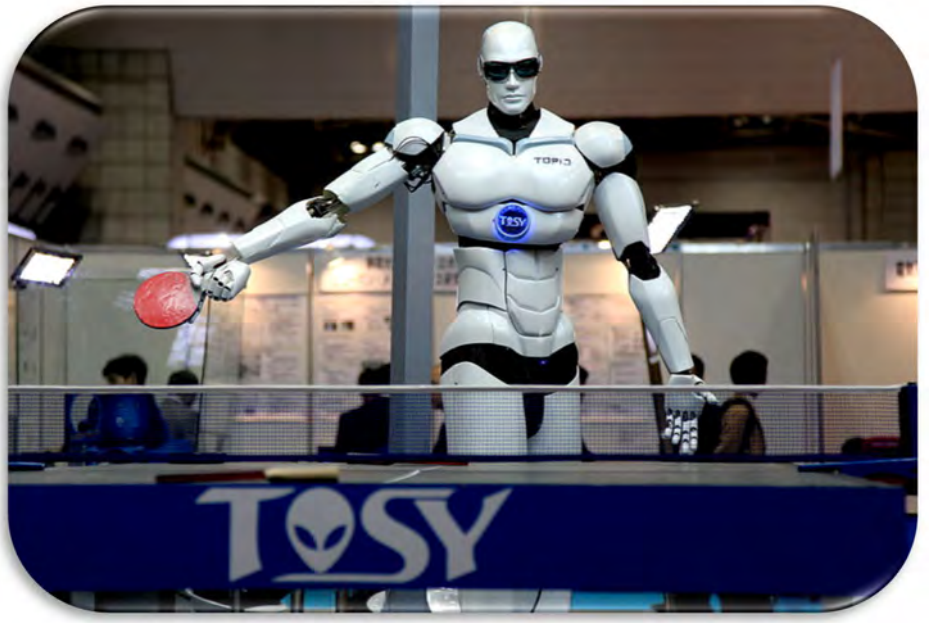
۳ – Babylon> FOLDOC

۴ – Babylon> Babylon English-English

۵ – Babylon> Britannica.com

۶ – humanoid robot

شکل بالا– روبوت ASIMO 2000 یک روبوت شبیه انسان است که در سال ۲۰۰۵ میلادی در نمایشگاه Expo 2005، در کشور ژاپُن به نمایش گذاشته شده است.



شکل بالا– TOPIO که یک روبوت شبیه انسان است، در نمایشگاه بین المللی روبوت IREX[1] در سال ۲۰۰۹ میلادی در شهر توکیوی کشور ژاپن، پینگ پنگ بازی کرد.

۱ – *IREX*، کوتاه شده‌ی International *R*obot *Ex*hibition است.

و دیگری، آنهایی که دارای نقش‌های تعریف شده‌تر و خاص‌تر هستند، مثل نانُو روبوت‌ها[۱] و کُپّه روبوت‌ها(روبوت‌های کپّه‌ای)[۲]؛ و روبوت‌های کمک کننده[۳]، که برای ساختن یا حرکت دادن چیزها یا انجام کارهای پَست یا خطرناک استفاده می‌شوند، مثل روبوت‌های صنعتی[۴] یا روبوت‌های متحرّک[۱] یا روبوت‌های سرویس‌دهنده(مُستَخدِم)[۲].

---

۱ - Nano robots، روبوت‌هایی هستند که اندازه‌ی اجزای آنها نزدیک به یک نانومتر($۱۰^{-۹}$ متر) است.
(http://en.wikipedia.org/wiki/Nanorobotics)

۲ - Swarm robots، سیستم‌های چند روبوتی‌ای هستند که از تعداد زیادی روبوت‌هایی که از لحاظ فیزیکی کوچک هستند، درست شده‌اند. در زیر نمونه‌ای از یک کپّه روبوت را می‌بینید:

(http://en.wikipedia.org/wiki/Swarm_Robotics)

۳ - helper robots

۴ - Industrial robot؛ در زیر یک روبوت صنعتی شش محوره که از آن برای جوشکاری استفاده می‌شود، نشان داده شده است:

(http://en.wikipedia.org/wiki/Industrial_robot)

۱ - Mobile robots، می‌توانند در محیط خود حرکت کنند.

(http://en.wikipedia.org/wiki/Mobile_robot)

۲ - servicing robots، مثل روبوت جارو کننده‌ی زیر که فرش‌های محیط‌های خانگی را جارو می‌کند؛ نام این روبوت، رومبا(Roomba) است:

یا روبوت زیر:

خصوصیّت مشترک دیگر برای روبوت‌ها این است که ظاهر یا حرکت آنها این حس را به ما می‌دهد که آنها از خودشان دارای قصد(نیّت)[۱] و عمل(عاملیّت)[۲] [برای انجام کار] هستند.[۳]

(http://en.wikipedia.org/wiki/Domestic_robot)

۱- Intent

۲- Agency

۳- http://en.wikipedia.org/wiki/Robot

انواع روبوت

شکل بالا– دست ماشینی

شکل بالا– دست ماشینی

شکل بالا– روبوت پادار

شکل بالا– روبوت چرخ‌دار

شکل بالا– وسیله‌ی زیر آبی خودمختار

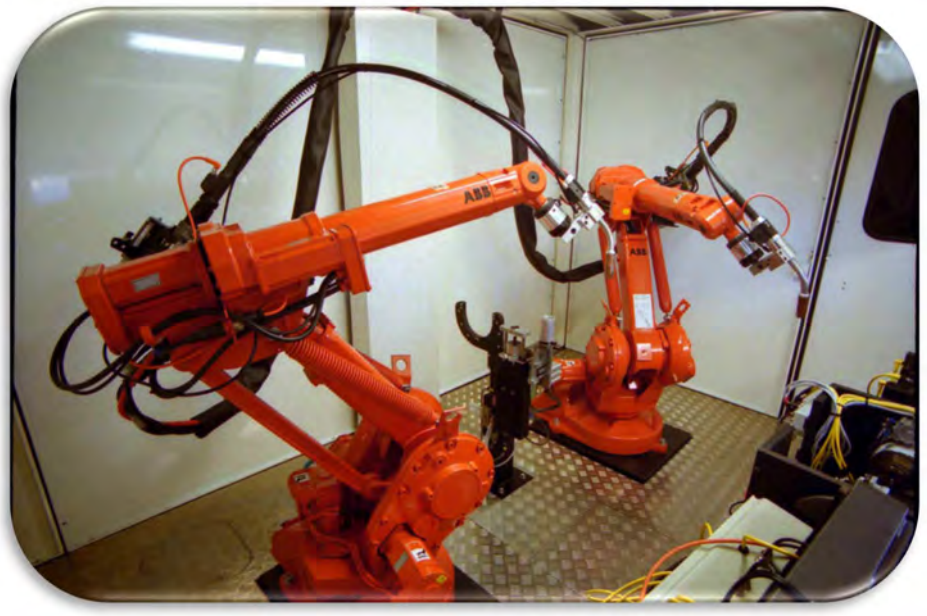
شکل بالا– وسیله‌ی بی‌سرنشین هوایی

کاربردهای روبوت

کارهایی که برای انسان خطرناک می‌باشند

روبوت ضدّعفونی کننده‌ی نشان داده شده در بالا در نیروگاه اتمی برای پاک(تمیز) کردن به کار می‌رود.

کارهای تکراری که برای انسان کِسِل کننده، پردَغدَغه یا کار بر می‌باشند



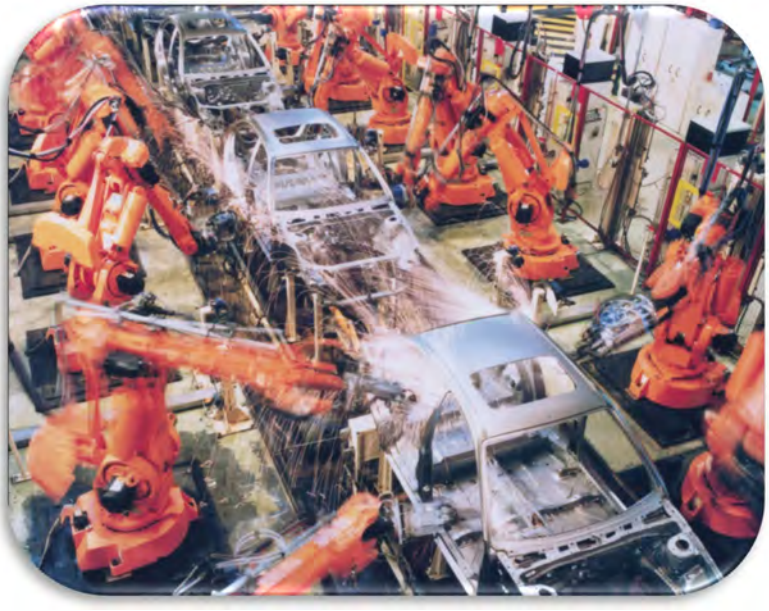
روبوت جوشکاری کننده (تصویر بالا)

کارهای پست که انسان دوست ندارد آنها را انجام دهد

روبوت اِسکرابمیت[۱] (تصویر بالا) در حال پاک کردن دستشویی

چه کارهایی برای انسان سخت و برای روبوت‌ها آسان است؟

کارهای تکراری، کار مداوم، انجام محاسبه‌های پیچیده، کار با پایگاه داده‌های بزرگ، برای روبوت‌ها آسان و برای انسان دشوار است.



شکل بالا– روبوت‌ها در خطّ تولید یک کارخانه‌ی خودروسازی

۱ – SCRUBMATE Robot

چه کارهایی برای انسان آسان و برای روبوت دشوار است؟

دلیل آوردن(استدلال)، تطبیق با شرایط جدید، آمیختن اطّلاعات نامربوط، آفرینندگی(خلّاقیّت)[۱]، برای روبوت دشوار و برای انسان [در مقایسه با روبوت] آسان است.

فرق روبوت با ماشین‌آلات خودکار[۲]

ماشین‌آلات خودکار، برای انجام کار مشخّصی طرّاحی شده‌اند؛ مثل: ماشین‌هایی که بطری‌ها را پُر می‌کنند[۳]، ماشین ظرفشویی[۴] و ماشین رَنگ پاش[۵]. ماشین‌آلات خودکار، همیشه بهتر از روبوت‌ها هستند، زیرا آنها می‌توانند به صورت بهینه برای انجام کاری خاص طرّاحی شوند.

۱- creativity
۲- automated machinery
۳- Bottling machine
۴- Dishwasher
۵- Paint sprayer

شکل بالا– یک ماشین که بطری‌ها را پر می‌کند

شکل بالا– ماشین ظرفشویی

شکل بالا– ماشین رنگ پاش

ولی روبوت‌ها، برای انجام کارهای گوناگون طرّاحی شده‌اند؛ مثل: بازوهای برداشتن و گذاشتن[۱] و روبوت‌های متحرّک.

شکل بالا– بازوی برداشتن و گذاشتن

شکل بالا– یک روبوت متحرّک

۱ – pick and place arms

تعریف علم روبوتیک[۱]

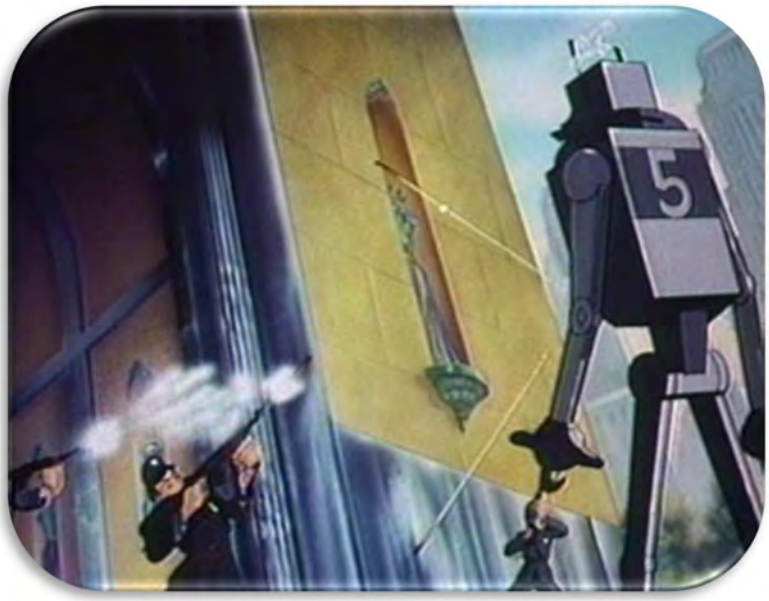
تعریف نخست– طرّاحی، ساخت و استفاده از ماشین‌هایی یا روبوت‌هایی که کارهایی را انجام می‌دهند که آن کارها معمولاً به وسیله‌ی انسان‌ها انجام می‌شوند.[۲]

تعریف دوّم– شاخه‌ای از هوش‌مصنوعی است که به طرّاحی، تولید و به کارگیری روبوت می‌پردازد.[۳]

تعریف سوّم– فنّآوری ساخت و استفاده از روبوت‌ها.[۴]

تعریف چهارم– شاخه‌ای از فنّآوری است که به طرّاحی، تولید، اداره، حالت ساختاری و کاربردهای روبوت‌ها می‌پردازد. روبوتیک، به علم‌های الکترونیک، مهندسی، مکانیک و نرم‌افزار، وابسته است.[۵]

قوانین روبوتیک



شکل بالا– یک روبوت معمولی، قبل از قوانین آسیموف، این تصویر در کارتون «سوپِر مَن» نشان داده شده است. قانون اوّل آسیموف روبوت را از حمله به انسان نهی می‌کند.[۱]

---

۱ – Robotics

۲ – Babylon> Britannica Concise Encyclopedia

۳ – Babylon> Concise Oxford English Dictionary

۴ – Babylon> Babylon English-English

۵ – http://en.wikipedia.org/wiki/Robotics


اِسحاق آسیمُوف

اِسْحاقْ آسیمُوف[۲] قانون‌های سه گانه‌ی روبوتیک را به صورت زیر بیان نمود:

قانون اوّل، روبوت نباید به انسان صدمه(آسیب) وارد کند.

قانون دوّم، روبوت باید دستوراتی که به وسیله‌ی انسان به آن داده شده و به انسان صدمه نمی‌زند(قانون اوّل را نقض نمی‌کند) را اجرا نماید.

قانون سوّم، روبوت باید از خودش مواظبت کند و این مواظبت نباید باعث صدمه زدن به انسان(نقض قانون اوّل) یا اجرا نشدن دستوراتی که انسان به آن داده است(نقض قانون دوّم) شود.

---

۱ - http://en.wikipedia.org/wiki/Three_Laws_of_Robotics

۲ - Isaac Asimov، ۱۹۹۲ - ۱۹۲۰ میلادی، نویسنده و دانشمند آمریکایی متولّد کشور روسیّه بود. (Babylon> Babylon English - English)

تاریخچه‌ی کوتاهی از روبوتیک

در سال ۱۹۵۴ میلادی اوّلین روبوت قابل برنامه‌ریزی صنعتی به وسیله‌ی جُورْجْ دِوُلْ[۱] با نام Universal Automation طرّاحی شد؛ این روبوت در سال ۱۹۶۲ میلادی به Unimation که نام اوّلین شرکت روبوت شد، تغییر نام پیدا نمود. شرکت Unimate در ابتدا ساخت لامپ تصویر تلویزیون را خودکار نمود.

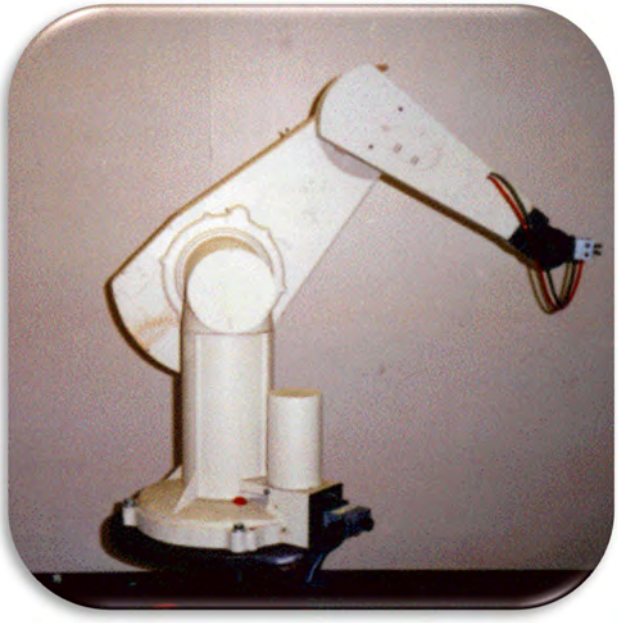
در سال ۱۹۷۸ میلادی، روبوت پوما(ماشین همه منظوره‌ی قابل برنامه‌ریزی برای مونتاژ یا اسِمبلی)[۲] به وسیله‌ی شرکت Unimation با پشتیبانی شرکت General Motors به وجود آمد (شکل زیر).

---

۱ - George Devol، ۲۰۱۱ - ۱۹۱۲ میلادی، طرّاحی آمریکایی بود. تصویری از او:

(http://en.wikipedia.org/wiki/George_Devol)

۲ - *Puma*(*P*rogrammable *U*niversal *M*achine for *A*ssembly)



شکل بالا– دست ماشینی پوما ۵۶۰

در دهه‌ی ۱۹۸۰ میلادی، صنعت روبوت به یک مرحله‌ی رشد سریع رسید. تعداد زیادی مؤسّسه، برنامه‌ها و زمینه‌هایی را در علم روبوتیک معرّفی نمودند. زمینه‌های روبوتیک، در مهندسی مکانیک، مهندسی برق و دانشکده‌های علوم کامپیوتر پخش شدند.

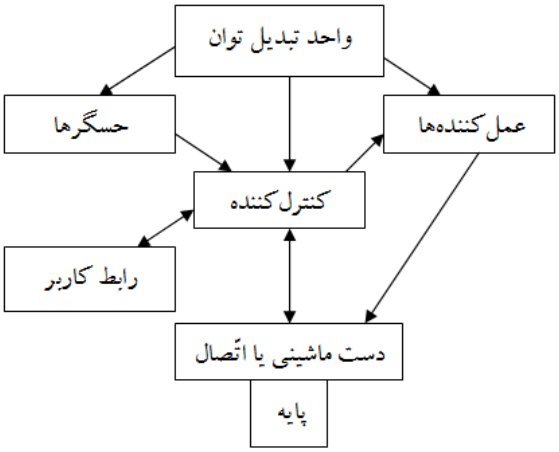
شکل بالا– دست ماشینی فنّآوری بارت

شکل بالا– روبوت Cognex

شکل بالا– روبوت ماهر SCARA

از سال ۱۹۹۵ میلادی تاکنون، تحوّلاتی کاربردی در روبوتیک و گردانندهٔ روبوت‌های متحرّک، یک دور دوّم رشد کمپانی‌ها و تحقیق را به وجود آورده است. در سال ۲۰۰۳ میلادی، کاوشگر کرهٔ مرّیخ ناسا[1] جستجویی را در سیّارهٔ مرّیخ برای تاریخچهٔ آب، در آن سیّاره اجرا نمود(شکل زیر).

اجزای کلیدی روبوت



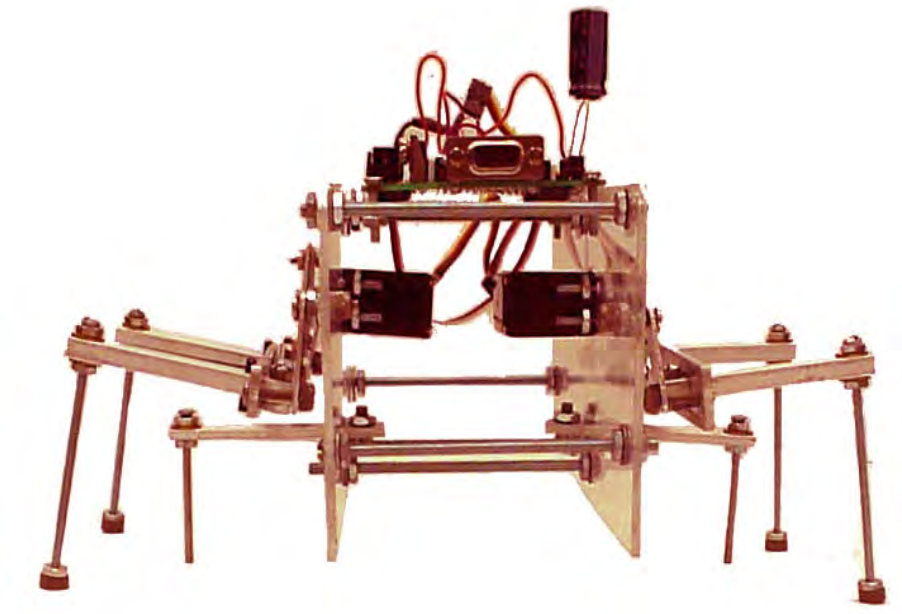
۱ - *NASA*، که کوتاه شدهٔ *N*ational *A*eronautics and Space *A*dministration است؛ سازمانی آمریکایی است که در سال ۱۹۵۸ میلادی برای تحقیق و ساخت وسایل و فعّالیّت‌هایی برای علوم هوانوردی و اکتشاف فضا تأسیس شد. (Babylon> Britannica Concise Encyclopedia)

پایه‌ی روبوت

پایه‌ی روبوت می‌تواند بـه صـورت ثابـت یـا متحـرّک باشـد؛ بازوهـای ماشـینی روبوتیـک در کارخانـه‌ها، نمونـه‌هایی از روبوت‌های ثابت می‌باشند؛ آنها نمی‌توانند پایه‌ی خود را حرکـت دهنـد. در زیـر تصـویری از یـک روبـوت دارای پایـه‌ی ثابـت را مشاهده می‌نمایید:

شکل بالا– روبوت A255

در روبوت‌های با پایه‌های متحرّک، پایه‌ها معمولاً دارای چرخ هستند و برخی نیز دارای پا مانندهایی برای حرکت می‌باشند:

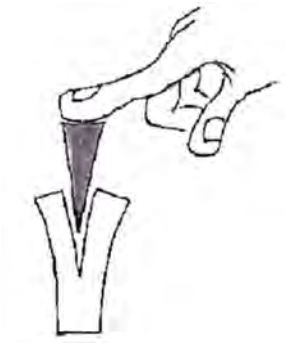


طرزکار(مکانیزم) روبوت

اجزای مکانیکی

در زیر چند نمونه از اجزای مکانیکی را می‌بینید:

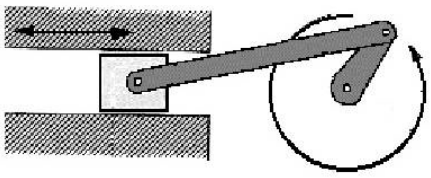
شکل بالا– سطح شیبدار



شکل بالا– گُوِه

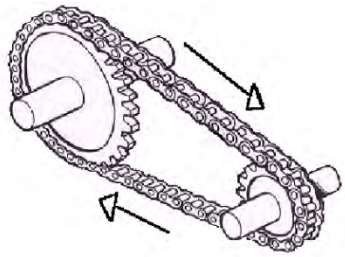
شکل بالا– حلقه‌های زنجیر

شکل بالا– چرخ‌دنده، دنده‌ی هَرزگرد و غیره

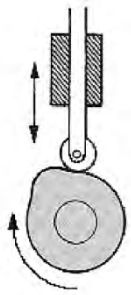


شکل بالا– میل‌لَنگ و پیستون

شکل بالا– اَهرُم



شکل بالا– زنجیر و چرخ‌دنده



شکل بالا– میل و بادامَک

حسگرها

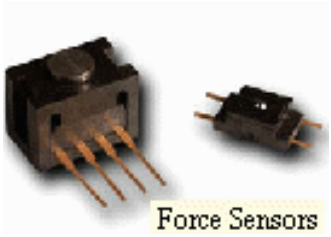
حس‌های انسانی؛ نظیر جا، صدا، لمس، مزّه و بو، اطّلاعات حیاتی را برای کار و بقای ما مهیّا می‌کنند. حسگرهای روبوت؛ پیکربندی یا وضعیّت و محیطش را ارزیابی می‌کنند و اطّلاعات این چنینی را به کُنتُرُلِر روبوت، به صورت علامت‌های الکترونیکی می‌فرستند (به عنوان مثال، وضعیّت بازو و حسّ گاز مَسموم).

شکل بالا– شتابْ سنج

روبوت‌ها اغلب به اطّلاعاتی که ماورای پنج حسّ انسان هستند، نیاز دارند (به عنوان مثال، دیدنْ در تاریکی، تشخیص مقدارهای کم پرتوهای نامرئی، اندازه‌گیری حرکت‌هایی که آن قدر سریع یا کوچک هستند که برای انسان‌ها قابل دیدن نمی‌باشند).

در ادامه برخی از گونه‌های حسگرها معرّفی شده‌اند:

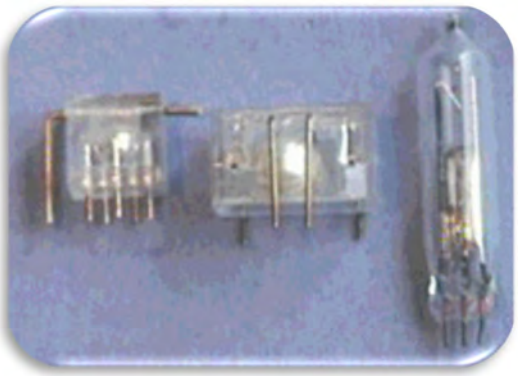
حسگرهای نیرو



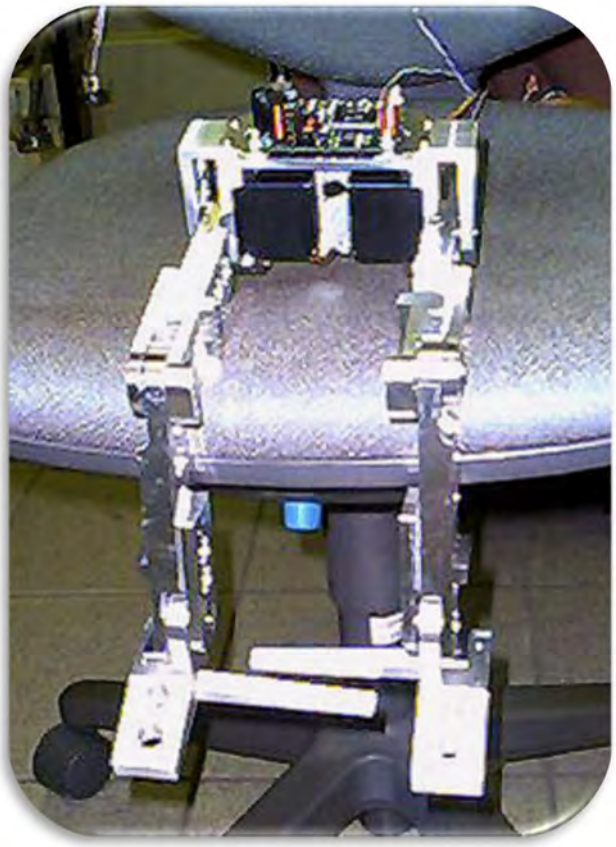
روبوت‌ها می‌توانند جاسازی دقیق و جاسازی قطعات ماشین را با استفاده از حسگر نیرو انجام دهند.

حسگرهای شیب(کجی)

حسگرهای شیب، به عنوان مثال برای متعادل کردن(بالانس کردن) یک روبوت به کار می‌روند.



در شکل‌های بالا دو نمونه حسگر شیب که به طور مثال می‌توانند در روبوت برنامه‌ریز نشان داده شده در شکل زیر به کار روند را می‌بینید:



حسگر بینایی

به عنوان مثال، برای برداشتن زباله‌ها، نظارت و ...

شکل بالا- یک روبوت دارای حسگرهای بینایی

عمل‌کننده‌ها[۱]

عمل‌کننده‌های رایج روبوتیک از ترکیب ابزارهای الکترومکانیکی مختلف استفاده می‌کنند؛ به عنوان مثال، موتور همزمان[۲]، موتور پِلّه‌ساز[۳]، موتور سِروُی با جریان متناوب[۴]، موتور سروی بی زُغال با جریان مستقیم[۵] و موتور سروی با زغال دارای جریان مستقیم[۶].

شکل بالا– موتور پلّه‌ساز

شکل بالا– سیلندر گازی[۷]

---

۱ – actuators

۲ – Synchronous motor ؛ **تعریف**– موتوری که دارای سرعت ثابت چرخش است و فقط با زیاد و کم کردن نیرو می‌توان این سرعت را تغییر داد. (Babylon> Britannica.com)

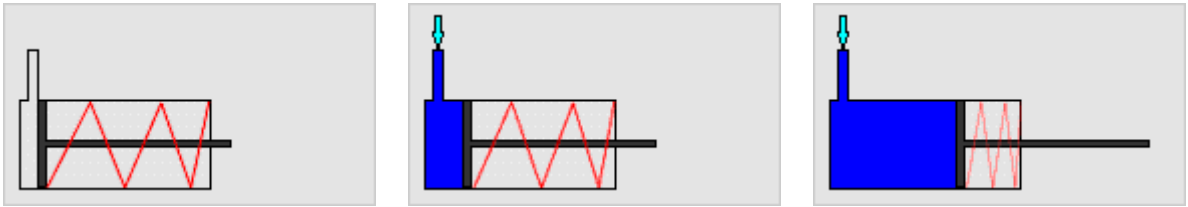
۳ – Stepper motor؛ **تعریف**– موتوری که میله‌ی حرکت کننده‌ی آن به اندازه‌ای خاص می‌چرخد و دورهای متوالی نمی‌زند. (Babylon> Merriam – Webster Collegiate® Dictionary)

۴ – AC servo motor؛ **تعریف**– موتورهای سِروُ، موتورهایی هستند که حرکت‌های با قدرت پایین را به حرکت‌های با قدرت بالا تبدیل می‌کنند. (Babylon> Babylon English-English)

۵ – Brushless DC servo motor

۶ – Brushed DC servo motor

۷ – Pneumatic Cylinder: **تعریف**– وسایلی مکانیکی هستند که از نیروی گاز فشرده شده برای تولید نیرو در یک حرکت خطّی مستقیم استفاده می‌نمایند. برای درک بهتر مطلب، تصویرهای زیر را که یک دوره(سیکل) کاری یکی از نوع‌های یک سیلندر گازی را نشان می‌دهند، به ترتیب از چپ به راست دنبال نمایید:



شکل بالا– موتور رُوغنی[۱]

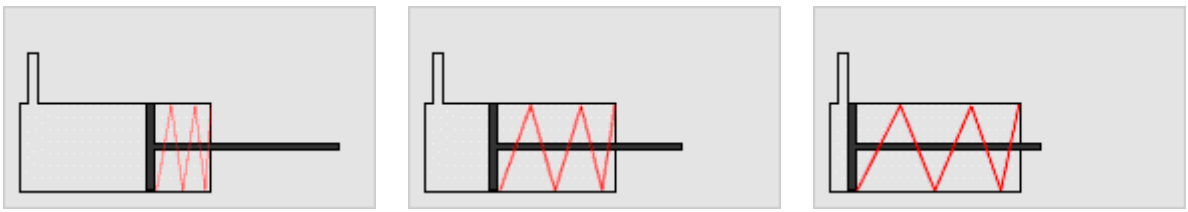
شکل بالا– موتور سروی

شکل بالا– موتور جریان مستقیم

شکل بالا– موتور گازی[۲]

کنترل کننده

هوش لازم برای کنترل روبوت را به وجود می‌آورد؛ اطّلاعات حس شده را پردازش می‌کند و دستورهای کنترلی را برای عمل‌کننده‌ها محاسبه می‌کند تا کارهای خاصّی را انجام دهند.



در شکل‌های بالا، فنر، به رنگ قرمز، گاز، به رنگ آبی، بدنه‌ی سیلندر، به رنگ سیاه نازک و پیستون، به رنگ سیاه ضخیم نشان داده شده‌اند.
(http://en.wikipedia.org/wiki/Pneumatic_cylinder)

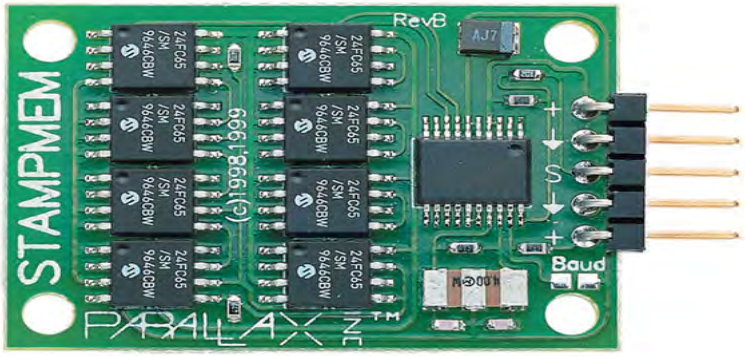
۱ – Hydraulic motor: **تعریف**– در این گونه، از فشار روغن برای به وجود آوردن چرخش استفاده می‌شود.
(http://en.wikipedia.org/wiki/Hydraulic_motor)

۲ – Pneumatic motor: **تعریف**– در این گونه، از فشار گاز برای به وجود آوردن چرخش یا حرکت خطّی استفاده می‌شود.
(http://en.wikipedia.org/wiki/Pneumatic_motor)

سخت‌افزار کنترل کننده

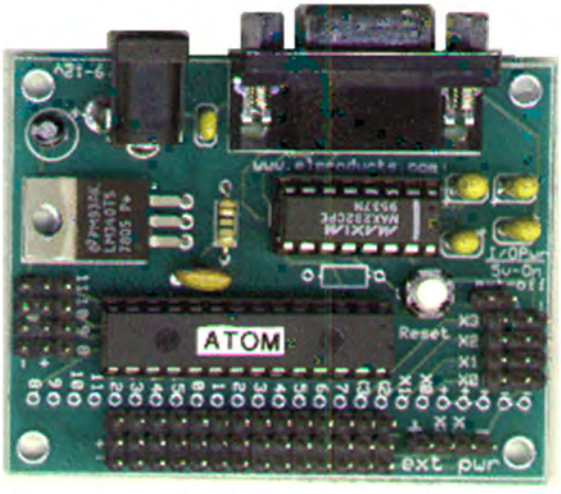
وسایل ذخیره سازی

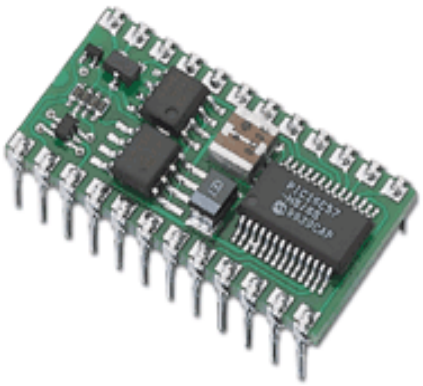
به عنوان مثال، حافظه، برای نگهداری برنامه‌ی کنترل و وضعیّت سیستم روبوت که با استفاده از حسگرها به دست می‌آیند.



موتور محاسبه

دستورهای کنترل را محاسبه می‌کند.





واحدهای رابط

سخت‌افزاری برای ارتباط بین محیط و کنترل کننده که به صورت دیجیتالی می‌باشد، هستند ([به عبارت دیگر، واسط بین] حسگرها و عمل کننده‌ها [می‌باشند]).

تقویّت کننده‌های عملیّاتی			تبدیل کننده‌ی آنالوگ به دیجیتال
 LM1458 **تقویّت کننده‌ی عملیّاتی دوتایی**	 LM358	 LM358	

صنایعی که از روبوت‌ها استفاده می‌کنند

عبارتند از: کشاورزی؛ خودروسازی؛ ساختمان؛ سرگرمی؛ در موردهای بهداشتی، مثل بیمارستان‌ها، نگهداری از بیمار، جرّاحی، تحقیق و غیره؛ آزمایشگاه‌های علمی، مهندسی و غیره؛ نظارت بر اجرای قانون؛ تولید؛ امور نظامی، مثل مین جنگی‌یابی، نظارت و حمله؛ حمل و نقل هوایی، زمینی، ریلی و فضایی؛ تسهیلات زندگی، مثل آب، برق و گاز؛ انبار کردن.

روبوت‌های صنعتی چه کارهایی می‌توانند انجام دهند؟

حمل مواد؛ انتقال مواد؛ بار قرار دادن در ماشین و یا خالی کردن آن؛ نُقْطِهْ جوشْ؛ جوش پیوسته؛ اِسپِری کردن روکِش؛ سرهم‌بندی قطعات و بازرسی.

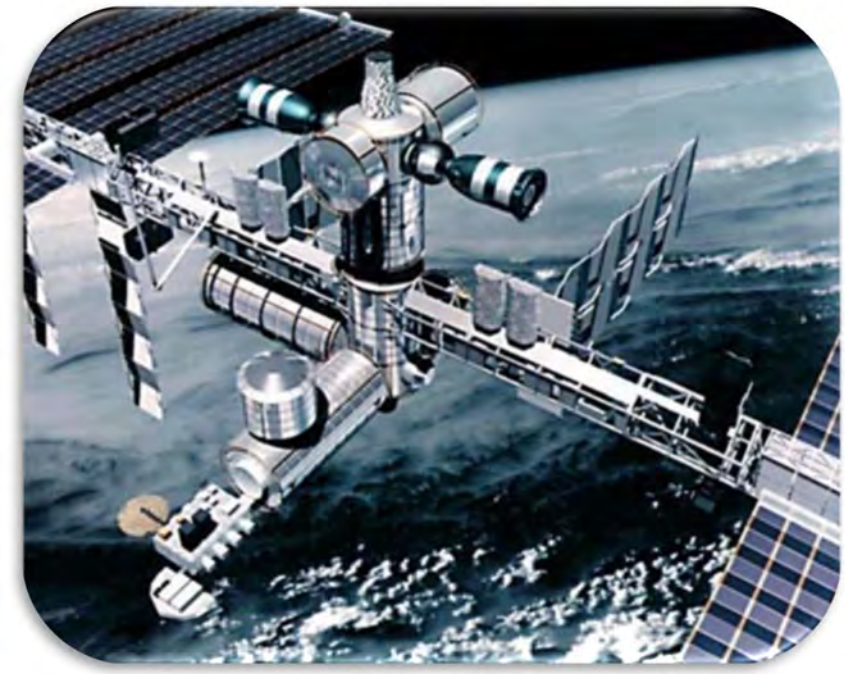
شکل بالا– بازوی حمل کننده‌ی مواد

شکل بالا– بازوی سرهم‌بندی کننده

شکل بالا– بازوی نقطه جوشْ زَن

روبوت‌ها در فضا

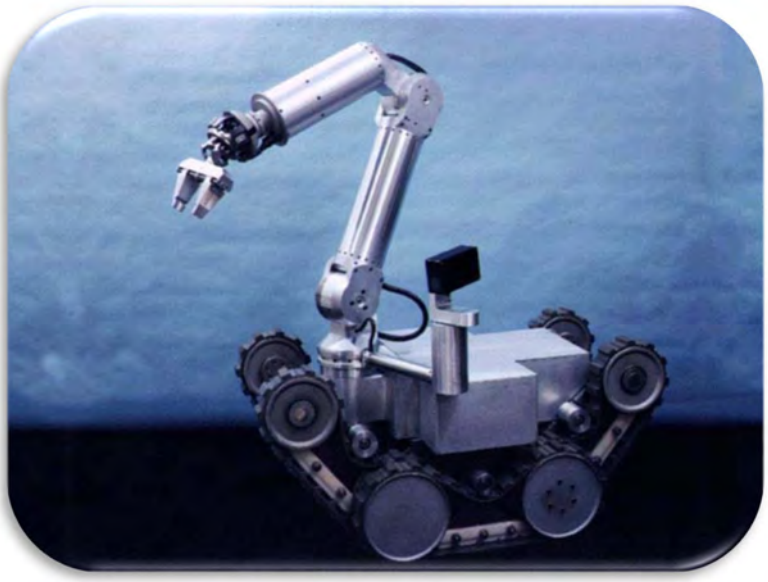
به عنوان مثال:



شکل بالا- ایستگاه فضایی شرکت ناسا

روبوت‌ها در جاهای خطرناک

به عنوان مثال:



شکل بالا- روبوت HAZBOT، در محیط قابل انفجار

شکل بالا- روبوت TROV، در زیر دریا

روبوت‌ها در پزشکی

به عنوان مثال، روبوت زیر برای جرّاحی‌های ریز به کار می‌رود:

روبوت‌ها در خانه

به عنوان مثال، روبوت‌های سرگرم کننده‌ی زیر:

چکیده‌ی مطلب‌های فصل بیست و سوّم

یک روبوت، یک عامل هوشمند مکانیکی است که می‌تواند خودش کارها را انجام دهد یا می‌تواند برای انجام کارها راهنمائی شود. در عمل، یک روبوت معمولاً یک ماشین الکترومکانیکی است که به وسیله‌ی کامپیوتر و برنامه‌نویسی الکترونیکی راهنمائی می‌شود. روبوت‌ها می‌توانند خودمختار یا نیمه‌خودمختار باشند و دارای دو نوع اصلی هستند:

یکی، آنهایی که برای پژوهش در زمینه‌ی سیستم‌های شبیه انسان استفاده می‌شوند، مثل روبوت‌های ASIMO و TOPIO

و دیگری، آنهایی که دارای نقش‌های تعریف شده‌تر و خاص‌تر هستند، مثل نانو روبوت‌ها و کپّه روبوت‌ها(روبوت‌های کپّه‌ای)؛ و روبوت‌های کمک کننده که برای ساختن یا حرکت دادن چیزها یا انجام کارهای پَست یا خطرناک استفاده می‌شوند، مثل روبوت‌های صنعتی یا روبوت‌های متحرّک یا روبوت‌های سرویس‌دهنده(مستخدم).

خصوصیّت مشترک دیگر برای روبوت‌ها این است که ظاهر یا حرکت آنها این حس را به ما می‌دهد که آنها از خودشان دارای قصد(نیّت) و عمل(عاملیّت) [برای انجام کار] هستند.

ماشین‌آلات خودکار برای انجام کار مشخّصی طرّاحی شده‌اند؛ ولی روبوت‌ها برای انجام کارهای گوناگون طرّاحی شده‌اند.

روبوتیک، شاخه‌ای از فنّـآوری است که به طرّاحی، تولید، اداره، حالت ساختاری و کاربردهای روبوت‌ها می‌پردازد.

قانون‌های سه گانه‌ی روبوتیک به وسیله‌ی اِسحاق آسیمُوف به این صورت بیان شده‌اند: قانون اوّل، روبوت نباید به انسان صدمه(آسیب) وارد کند. | قانون دوّم، روبوت باید دستوراتی را که به وسیله‌ی انسان به آن داده شده و قانون اوّل را نقض نمی‌کند، اجرا نماید. | قانون سوّم، روبوت باید از خودش مواظبت کند و این مواظبت نباید باعث نقض قانون اوّل یا نقض قانون دوّم شود.

روبوت‌ها اغلب به اطّلاعاتی که ماورای پنج حسّ انسان هستند، نیاز دارند؛ حسگرهای روبوت، پیکربندی یا وضعیّت و محیطش را ارزیابی می‌کنند و اطّلاعات این چنینی را به کنترلر روبوت، به صورت علامت‌های الکترونیکی می‌فرستند.

عمل‌کننده‌های رایج روبوتیک از ترکیب ابزارهای الکترومکانیکی مختلف استفاده می‌کنند.

کنترل کننده، هوش لازم برای کنترل روبوت را به وجود می‌آورد؛ اطّلاعات حس شده را پردازش می‌کند و دستورهای کنترلی را برای عمل‌کننده‌ها محاسبه می‌کند تا کارهای خاصّی را انجام دهند.

واحدهای رابط، سخت‌افزاری برای ارتباط بین محیط و کنترل کننده که به صورت دیجیتالی می‌باشد، هستند ([به بیانی دیگر، واسط بین] حسگرها و عمل‌کننده‌ها [می‌باشند]).

یادآوری یا تکمیل مطلب‌های فصل بیست و سوّم

مطلب- *روبوت‌ها*، عامل‌هایی فیزیکی هستند که کارها را با دستکاری جهان فیزیکی انجام می‌دهند. برای انجام این کار، آنها به اندام‌های مُجری[۱]، مثل پاها، چرخ‌ها، مَفصَل‌ها و تسمه‌ها مجهّز شده‌اند. روبوت‌ها همچنین به حسگرها مجهّز شده‌اند که به آنها اجازه می‌دهند محیطشان را درک کنند. امروزه در روبوتیک از مجموعه‌ای از حسگرهای گوناگون، شامل دوربین‌ها و اشعه‌ی لیزر، برای اندازه‌گیری محیط، و ژیروسکوپ[۲]ها و شتاب‌سنج‌ها، برای اندازه‌گیری حرکت روبوت استفاده می‌شود.[۳]

تست- حسگر، نمونه‌ای از یک وسیله‌ی ..... برای یک روبوت است.

۱) خروجی

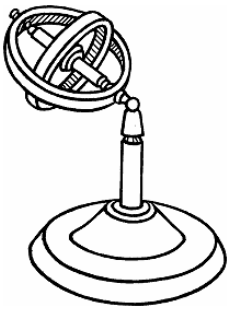
۲) ورودی

۳) ذخیره‌سازی

---

۱ - effectors

۲ - gyroscope، وسیله‌ای برای اندازه‌گیری یا دستکاری جهت می‌باشد. (http://en.wikipedia.org/wiki/Gyroscope) تصویری از یک ژیروسکوپ:



(Babylon> Merriam-Webster Collegiate® Dictionary)

۳ - کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم، فصل بیست و پنجم، علم روبوتیک(Robotics)، صفحه‌ی ۹۷۱

۴) حافظه

جواب- گزینه‌ی «۲» درست است.[۱]

سؤال- روبوت‌ها از اندام‌های مجری(effectors) و عمل کننده‌ها(actuators) برای دستکاری محیط استفاده می‌کنند؛ تفاوت میان این دو چیست؟

جواب- اندام‌های مجری، هر وسیله‌ای(ابزاری) هستند که بر روی محیط اثر می‌گذارند؛ مثلاً بازوها، پاها، چرخ‌ها، انگشتان، قلّاب‌ها، بال‌ها و .... عمل کننده‌ها، دستگاه(مکانیزم)هایی هستند که اندام‌های مجری را قادر به اجرای عملیّات می‌کنند؛ مثلاً موتورهای الکتریکی، موتور(سیلندر)های روغنی و موتور(سیلندر)های گازی.[۲]

سؤال- قانون‌های سه گانه‌ی روبوتیک اسحاق آسیموف را بیان کنید.

جواب- به متن درس مراجعه کنید.[۳]

سؤال- چرا روبوت Unimate در تاریخ روبوتیک مهم است؟

جواب- چون اوّلین روبوت صنعتی است. همچنین اوّلین روبوت موفّقی است که به طور گسترده‌ای در مکانیزاسیون کارخانه استفاده شده است.[۴]

---

۱ - تست‌های مربوط به «هوش مصنوعی» موجود در پایگاه اینترنتی http://quegrande.org، که این پایگاه اینترنتی، مربوط به جامعه‌ای از زبان‌آموزان کشور اسپانیا می‌باشد، سال ۲۰۰۸ میلادی

۲ - آزمون شماره‌ی ۲ درس «سمینار دانشجوی سال اوّل روبوتیک(Robotics Freshman Seminar)» استاد، «والری هِنْدِرسُنْ سامِتْ(Valerie Henderson Summet)»، دانشکده‌ی علوم کامپیوتر و ریّاضیّات دانشگاه اِمُری(EMORY)ی کشور آمریکا، پاییز سال ۲۰۱۲ میلادی

۳ - آزمون شماره‌ی ۳ درس «سمینار دانشجوی سال اوّل روبوتیک» استاد، «والری هندرسن سامت»، دانشکده‌ی علوم کامپیوتر و ریّاضیّات دانشگاه امری کشور آمریکا، پاییز سال ۲۰۱۲ میلادی

۴ - آزمون شماره‌ی ۱ درس «سمینار دانشجوی سال اوّل روبوتیک» استاد، «والری هندرسن سامت»، دانشکده‌ی علوم کامپیوتر و ریّاضیّات دانشگاه امری کشور آمریکا، پاییز سال ۲۰۱۲ میلادی

فصل بیست و چهارم

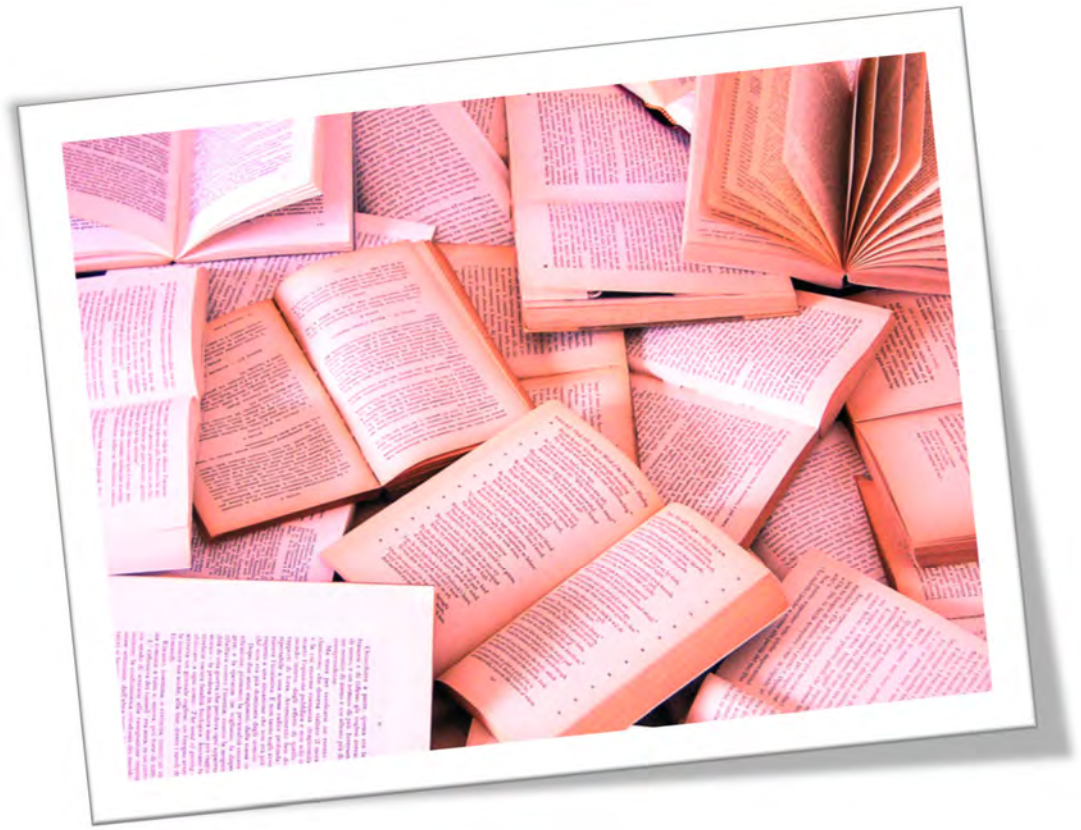
۱

آشنایی با زبان برنامه‌نویسی پُرولُوگ[۲]

---

۱ – تصویر، آلِنْ کُلْمِراوِرْ(Alain Colmerauer)؛ متولّد ۱۹۴۱ میلادی، دانشمند علوم کامپیوتری و پدیدآورنده‌ی زبان برنامه‌نویسی پرولوگ، از کشور فرانسه است.
(http://alain.colmerauer.free.fr و http://en.wikipedia.org/wiki/Alain_Colmerauer)

۲ – Prolog Programming



فهرست برخی از عنوان‌های نوشته‌ها

محاسبه‌های موفّق

محاسبه‌های یگانه

محاسبه‌های چندگانه

یکسان‌سازی(یکی‌سازی)

پردازش لیست

توابع موجود برای لیست‌ها

برّرسی نوع

مقایسه‌ی عبارت‌ها

محاسبه

مقایسه‌ی عبارت‌های محاسباتی

بازگشت

ورودی و خروجی

استفاده از کد اَسکی

توجّه:

در ابتدای مطالعه‌ی این فصل، کدها را در محیط پرولوگ وارد نکنید، چون ممکن است که به خطا برخورد کنید!؛ صبر کنید تا در ادامه‌ی همین فصل و مخصوصاً در قسمت‌های «مثال به وجود آوردن یک پایگاه دانش در SWI-Prolog» و «مثال اجرای یک برنامه در اعلان SWI-Prolog» با طریقه‌ی کد نویسی در محیط SWI-Prolog آشنا شوید!

توجّه:

این فصل براساس SWI-Prolog نسخه‌ی 6.2.0 در ویندوز ۷(سِوِن) نوشته شده است.

پرولوگ که یکی از اوّلین زبان‌های برنامه‌نویسی منطقی است[۱]، یک زبان برنامه‌نویسی سطح بالا[۲] می‌باشد و زبانی است که برای برنامه‌نویسی هوش مصنوعی به وجود آمده است.[۳] پرولوگ به معنی *«برنامه‌نویسی در منطق»*[۴] می‌باشد. با این زبان به راحتی می‌توان دانش را بیان کرد. برای جستجوی رشته‌ها یا الگوها در داده‌های ورودی هم مناسب می‌باشد(پشتیبانی از تطبیق الگو[۵])[۶].
آرم[۷] این زبان هم به صورت زیر است:


آلِن کُلمِراوِر

ایده‌ی این زبان اوّلین بار به وسیله‌ی گروهی دُوْر(پیرامون) آلِن کُلمِراوِر در ابتدای دهه‌ی ۱۹۷۰ میلادی بیان شد و نخستین سیستم پرولوگ در سال ۱۹۷۲ میلادی به وسیله‌ی آلِن کُلمِراور و فیلیپ راسِل[۸] به وجود آمد.

دانلود SWI-Prolog

یکی از انواع پرولوگ و در ضمن رایج‌ترین نوع آن، SWI-Prolog می‌باشد که با استفاده از آدرس اینترنتی http://www.swi-prolog.org/download/stable می‌توانید نسخه‌ی مورد نظر خود را دانلود کرده و نصب نمایید.

۱- http://en.wikipedia.org/wiki/Prolog

۲- High Level Language؛ **یادآوری**- در زبان‌های برنامه‌نویسی سطح بالا، دستورها به زبانی ساده که شبیه زبان انسان است، نوشته می‌شوند؛ زبان‌های برنامه‌نویسی سطح بالا دورتر به زبان ماشین ترجمه می‌شوند. (Babylon> Babylon English-English)

۳- Babylon> Concise Oxford English Dictionary

۴- *PROLOG* = *PRO*gramming in *LOG*ic

۵- pattern-matching

۶- Babylon> FOLDOC

۷- logo

۸- Phillippe Roussel

تعریف رَوال(زیر برنامه)[1]ها

تعریف– برنامه‌ها از تعریف روال‌ها تشکیل شده‌اند؛ یک روال، منبعی برای ارزیابی چیزی می‌باشد.

مثال– «a:-b,c.» *(یعنی، اگر b و c(هر دو) درست باشند، آنگاه a درست خواهد بود.)*، به صورت روالی، یک روال برای ارزیابی a به وسیله‌ی ارزیابی b,c است. در اینجا «ارزیابی» چیزی، به معنی تشخیص این است که آیا با توجّه به برنامه، در کل، درست است یا نه؟. روال «a:-b,c»، در منطق می‌تواند به صورت «a←b∧c» نوشته شود و سپس به این صورت بیان شود، «اگر b و c (هر دو) درست باشند، آنگاه a درست خواهد بود» یا «a درست است، در صورتی که b درست باشد و c هم درست باشد (هر دو درست باشند)».

نکته:

در پرولوگ، «:-»، به معنای «if(اگر)»؛ «,»، به معنای «and»؛ و «;»، به معنای «or» می‌باشد.

فراخوانی روال‌ها

اجرا شامل ارزیابی فراخوانی‌ها می‌باشد و با یک پرس‌وجوی *اوّلیه شروع* می‌شود.

مثال‌ها:

```
?-a,d,e.

?-likes[2](chris[3],X).

?-likes(soh,prolog),likes(soh,ai).
```

پرس‌وجوهای بالا در حال پرسیدن این هستند که: آیا فراخوانی‌های درون آنها با توجّه به روال‌های ارائه شده در برنامه درستند یا نه؟. *پرولوگ فراخوانی‌های در یک پرس‌وجو را به صورت ترتیبی و با همان ترتیبی که از چپ به راست نوشته شده‌اند، ارزیابی می‌کند.*

---

۱ – procedure(رَوال یا زیرْبرنامه)؛ **یادآوری**– [یک برنامه‌ی کامپیوتری، مجموعه‌ای از یک یا چند زیربرنامه است و هر زیربرنامه، مجموعه‌ای از دستورهای کامپیوتری است؛ هر زیربرنامه، دارای یک نام است و به وسیله‌ی این نام می‌تواند صدا زده(فراخوانی) شود و دستورهای درون آن اجرا شود. (Babylon> Merriam-Webster Collegiate® Dictionary)

۲ – در زبان انگلیسی یعنی: «دوست می‌دارد»

۳ – کِریسْ، در زبان انگلیسی نامی برای یک زن یا یک مرد است؛ کوتاه‌نویسی‌ای برای Christian، برای مرد یا Christina، برای زن است. (Babylon> Babylon English-English)

مثال:

?-a,d,e.

در مثال بالا اوّل a ارزیابی می‌شود، بعد d ارزیابی می‌شود و سپس e ارزیابی می‌شود.

چند تعریف

متغیّر: برخی از عناصر کلّی سیستم را ارائه می‌کند.

ثابت[۱]: یک عضو مشخّص و شناخته شده‌ی سیستم را بیان می‌کند.

گزاره[۲]: ارتباط یا خصوصیّتی را در سیستم بیان می‌کند.

مطلب مهم:

قرارداد– *واژگان شروع شده با یک حرف بزرگ یا خطّ زیرین(_)، متغیّر هستند.*

مثال:

?-likes(chris,X).

در اینجا X، *[نام]* یک متغیّر می‌باشد.

نکته:

پرس‌وجوها و روال‌ها هر دو به کلاس عبارت‌های منطقی تعلّق دارند.

نکته:

گزاره‌ها و ثابت‌ها همیشه با حروف کوچک یا عدد شروع می‌شوند.

---

۱ – constant

۲ – predicate

مطلب مهم:

هر خط در یک برنامه‌ی پرولوگ، یک عبارت(کِلُزْ)[۱] نام دارد. در پرولوگ دو نوع عبارت داریم؛ یکی، قانون[۲]ها که *دارای علامت «:-» هستند؛ مثل «happy(john):-happy(mary).» ؛ یعنی، «اگر مری خوشحال باشد، آنگاه جان خوشحال است.»* و دیگری، واقعیّت‌ها[۳] که *دارای علامت «:-» نمی‌باشند؛ مثل «happy(mary).» ؛ یعنی، «مری خوشحال است.» هر واقعیّت فقط دارای یک گزاره است. هر عبارت در پرولوگ با نقطه(.) خاتمه می‌یابد.* در یک قانون، گزاره‌ی قبل از علامت «:-»، سَر[۴] قانون نام دارد و گزاره‌ای که بعد از این علامت می‌آید، بدنه[۵]ی قانون نام دارد. به قانون زیر توجّه کنید:

likes(sara,X)	:-	toy[۶](X),plays(sara,X)	.
سر		بدنه	

تعریف یک گزاره در پرولوگ تقریباً با تعریف یک زیربرنامه برابر است. در ضمن، گزاره‌هایی که در بدنه‌ی شرط می‌آیند، تقریباً با فراخوانی زیربرنامه برابر هستند.

نکته:

ثابت‌ها و متغیّرها به طور مستقیم در عبارت ظاهر نمی‌شوند و فقط به صورت آرگومان در گزاره‌ها ظاهر می‌شوند.

نکته:

گزاره‌ها تقریباً هیچوقت به صورت آرگومان‌های دیگر گزاره‌ها قرار نمی‌گیرند.

توضیح[۷]

توضیح‌ها به وسیله‌ی پرولوگ پردازش نمی‌شوند؛ به عبارت دیگر، کاری بر روی آنها انجام نمی‌شود. اگر توضیح، یک خطّی باشد، باید در ابتدای خط از علامت درصد(٪) استفاده کنیم و اگر توضیح، چند خطّی باشد، باید در ابتدای آن از «/» و در انتهای آن از «*/» استفاده نماییم. البتّه برای توضیح یک خطّی هم می‌توان از «/*»، در ابتدای توضیح و «*/»، در انتهای توضیح استفاده کرد.*

۱- clause

۲- rule

۳- facts

۴- head

۵- body

۶- در زبان انگلیسی یعنی: «اسبابْ بازی»

۷- comment(remark)؛ **یادآوری**- در برنامه‌نویسی، متن توضیحی‌ای است که در کد برنامه قرار داده می‌شود تا خوانندگان دیگر بتوانند آن را بفهمند. (Babylon> FOLDOC)

مثال- توضیح یک خطّی:

```
% Soh Jel
```

مثال- توضیح یک خطّی:

```
/* Soh Jel */
```

مثال- توضیح چند خطّی:

```
/* Soh
Jel */
```

محاسبه‌ها

یک *محاسبه*، *زنجیره‌ای* از *پرس‌وجوها* می‌باشد.

مثال:

?-a,d,e. پرس‌وجوی اوّلیّه

a:-b,c. شرط برنامه، با سر(ابتدای) a و بدنه‌ی b,c

با شروع از پرس‌وجوی اوّل، اوّلین فراخوانی در آن با ابتدای شرط نشان داده شده منطبق می‌شود و بنابراین، پرس‌وجوی به وجود آمده به صورت «?-b,c,d,e.» می‌باشد. سپس اجرا در مورد پرس‌وجوی به وجود آمده به روشی مشابه عمل می‌کند.

در مورد متغیّرها

مطلب مهم:

متغّیرهایی که درون پرس‌وجوها هستند، به صورت سور وجودی(∃) رفتار می‌نمایند.

مثال:

```
?-likes(X,prolog).
```

می‌گوید که «آیا (∃X) likes(X,prolog)(یعنی، *وجود دارد* Xای که X، prolog را دوست داشته باشد؟)»، یا «Xای را پیدا نمایید که به ازای آن، likes(X,prolog) درست باشد».

مطلب مهم:

متغّیرهای درون شرط‌های برنامه به صورت سورهای عمومی(∀) عمل می‌کنند.

مثال:

likes(chris,X):-likes(X,prolog).

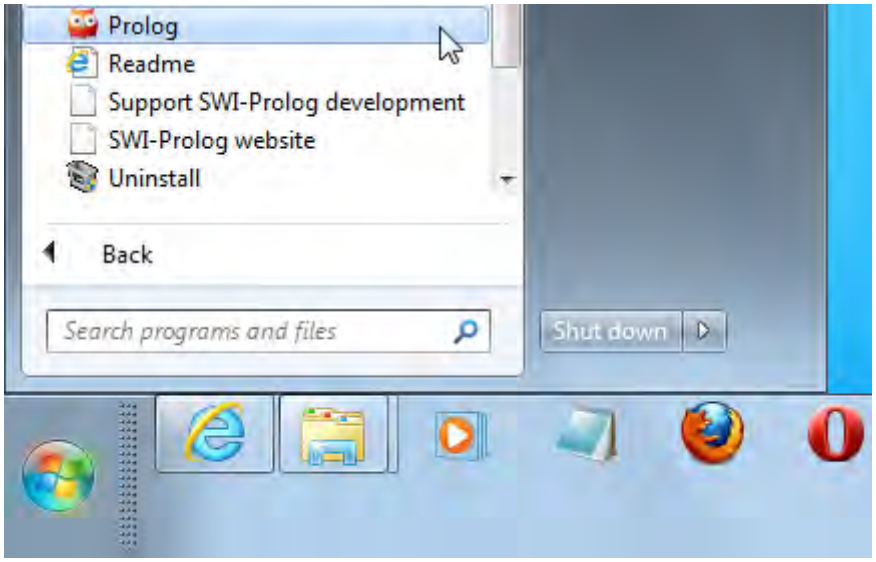
این عبارت را بیان می‌نماید:

$$(\forall X)\ (likes(chris,X) \leftarrow likes(X,prolog))$$

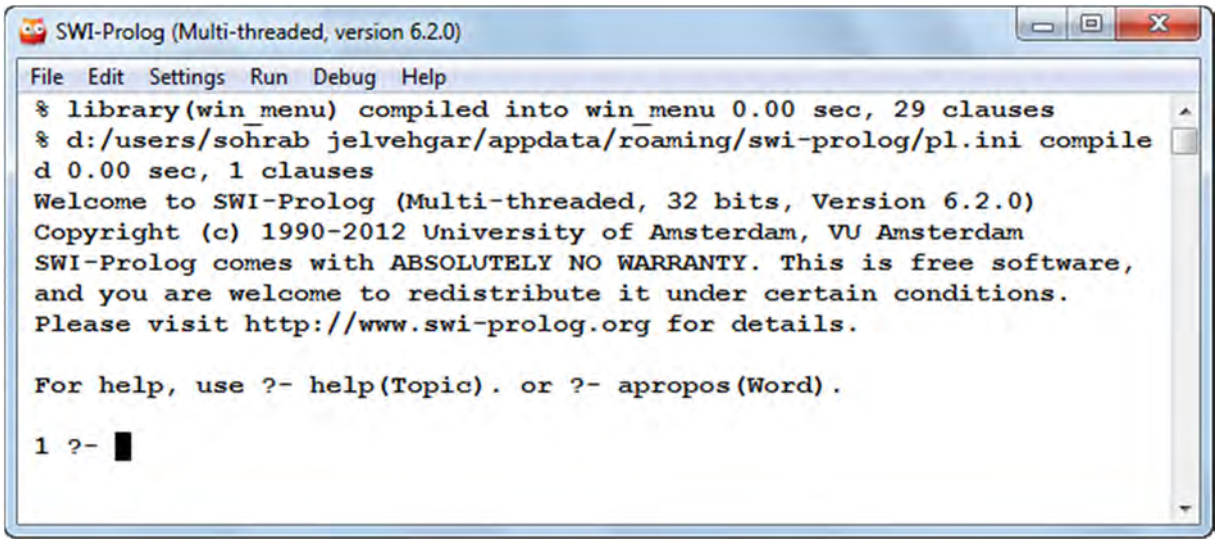
و می‌گوید: «به ازای هر X، اگر X، prolog را دوست داشته باشد، آنگاه chris، X را دوست خواهد داشت.»

مثال به وجود آوردن یک پایگاه دانش در SWI-Prolog:

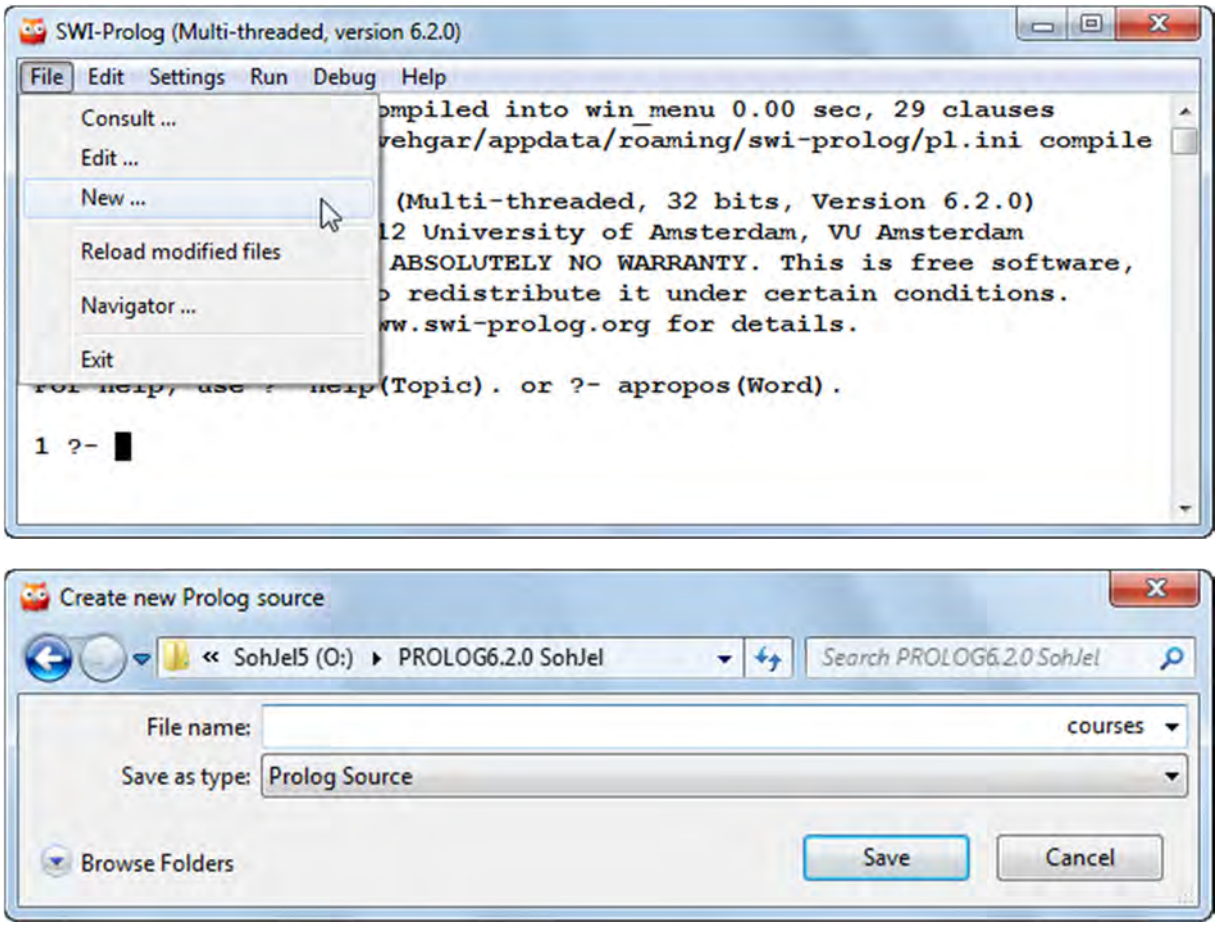
ابتدا برنامه‌ی SWI-Prolog را که در گذشته بر روی کامپیوتر خود نصب کرده‌ایم، اجرا می‌کنیم:



بعد از اجرای مسیر بالا(Start> All Programs> SWI-Prolog> Prolog) در ویندوز ۷، محیط برنامه به صورت زیر باز می‌شود:

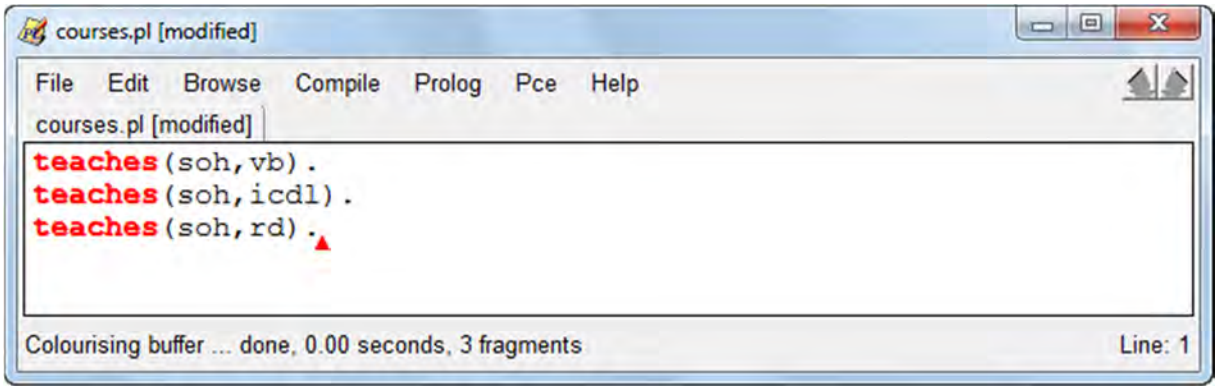


در پنجره‌ی بالا، از منوی File، گزینه‌ی New را انتخاب کرده و در پنجره‌ای که باز می‌شود، در مسیری دلخواه قرار گرفته و فایلی را به وجود می‌آوریم؛ مثلاً نام آن را courses[1] می‌گذاریم:

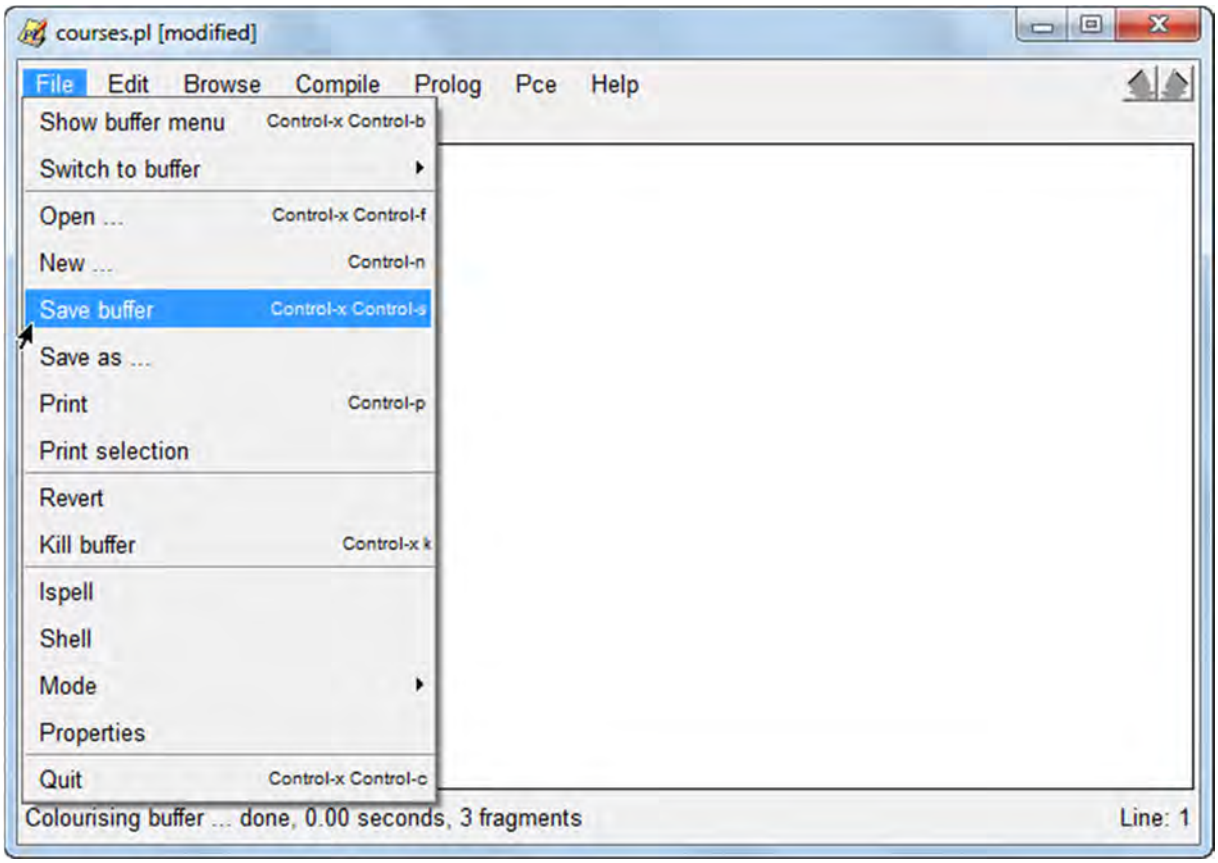


۱ – در لغت به معنی «درس‌ها» یا «دُوْره‌ها» می‌باشد.

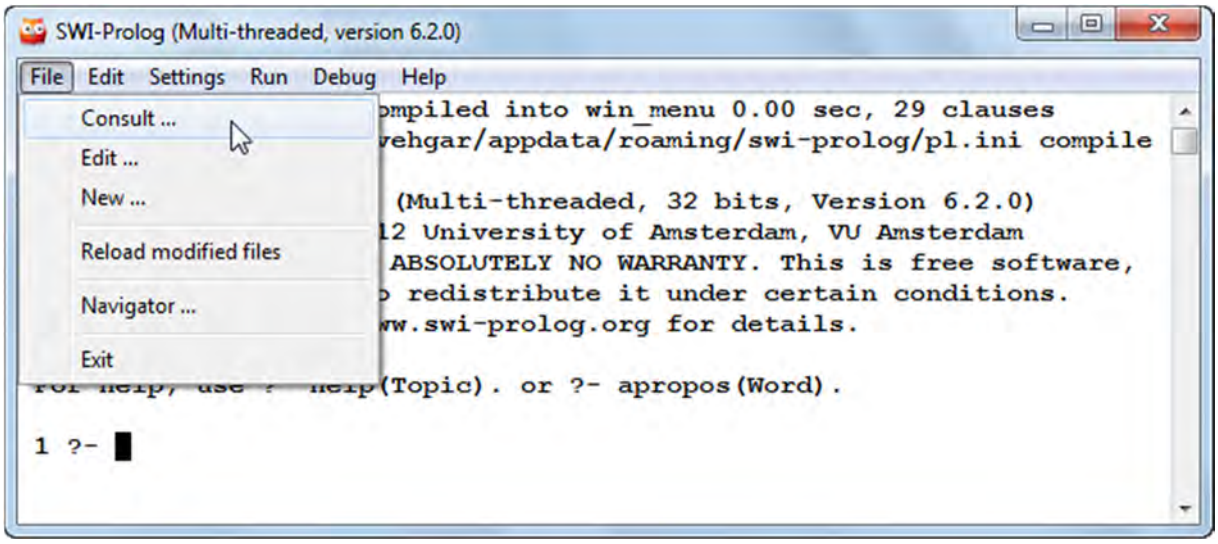
بعد از کلیک بر روی دکمه‌ی Save، پنجره‌ای برای این فایل باز می‌شود که در این پنجره موردهای زیر را می‌نویسیم:



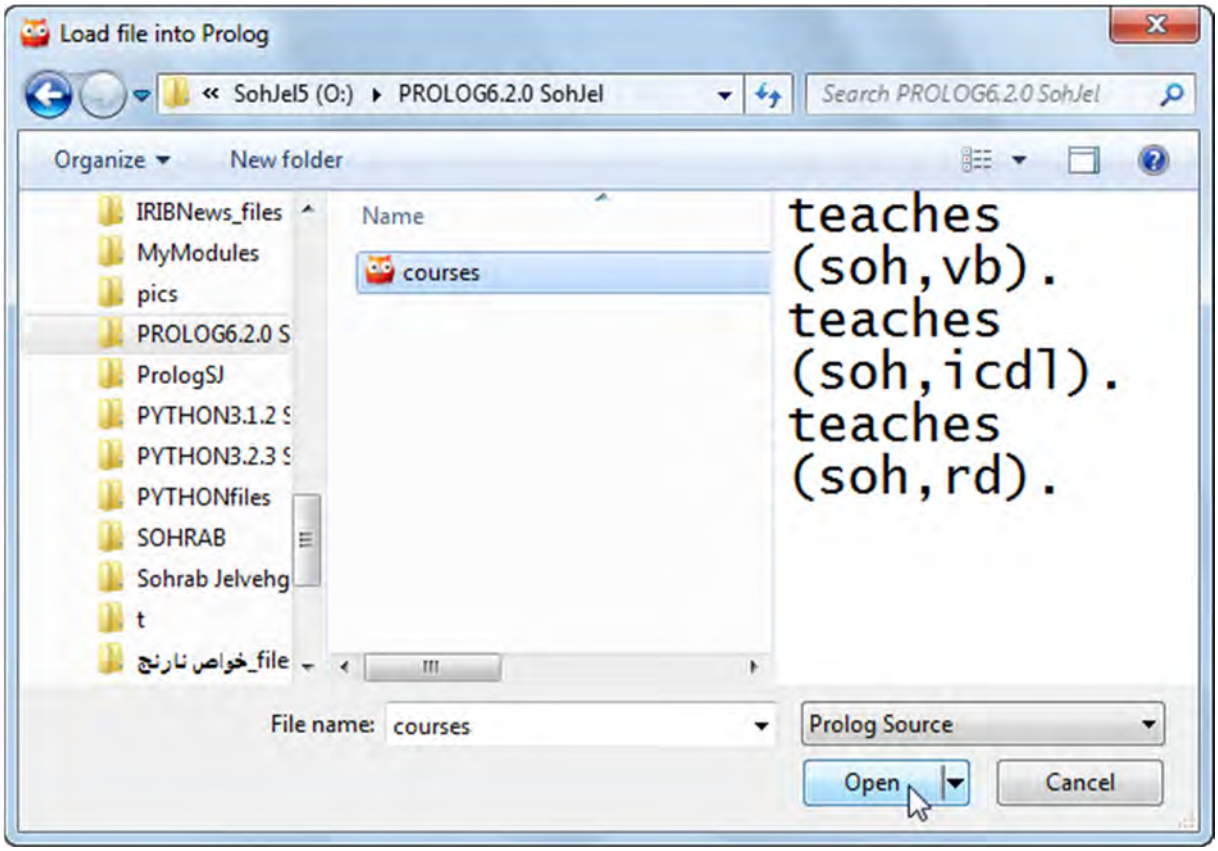
سپس از منوی File، گزینه‌ی Save buffer را انتخاب می‌کنیم:

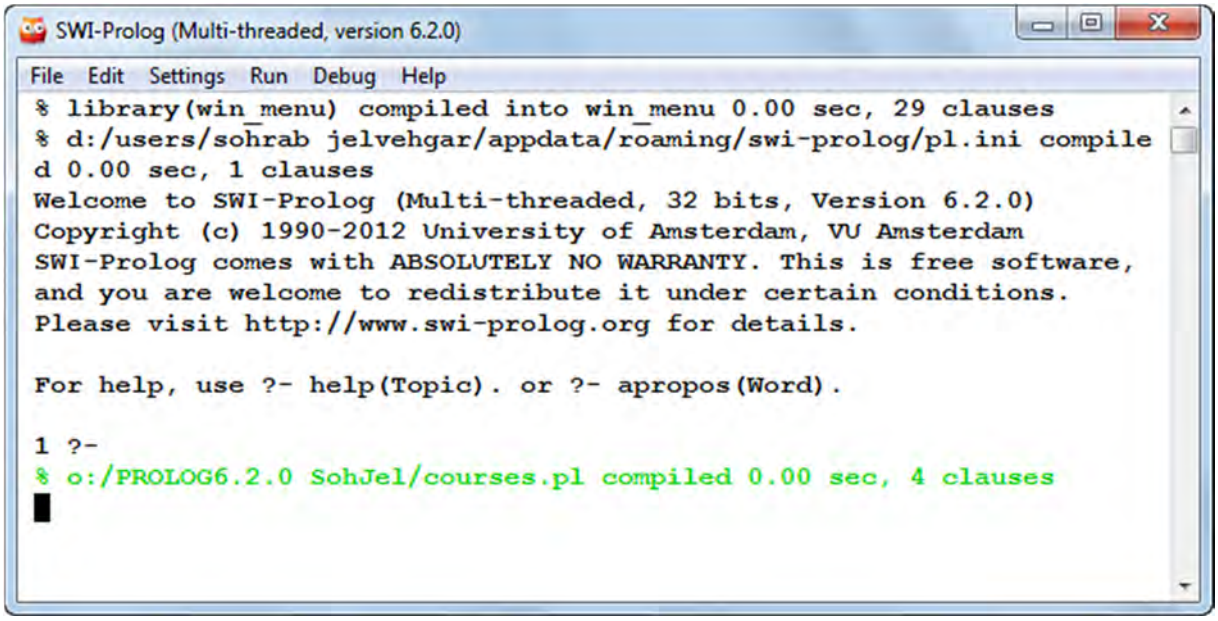


این پنجره(courses.pl) را می‌بندیم و به محیط SWI-Prolog برمی‌گردیم؛ از منوی File، Consult را انتخاب کرده:

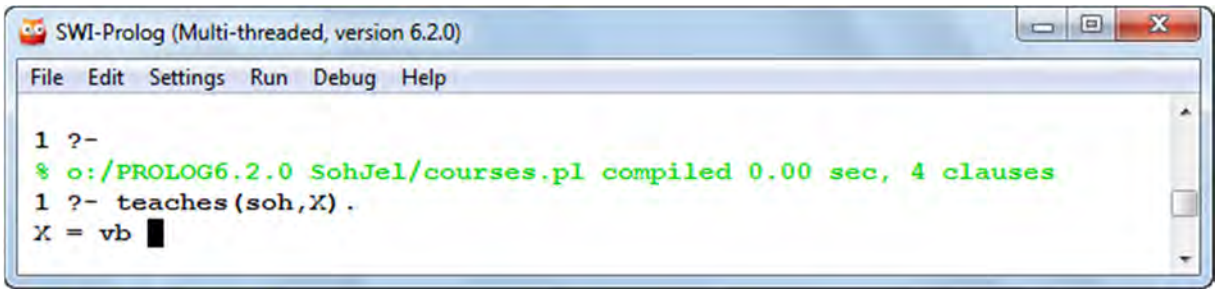


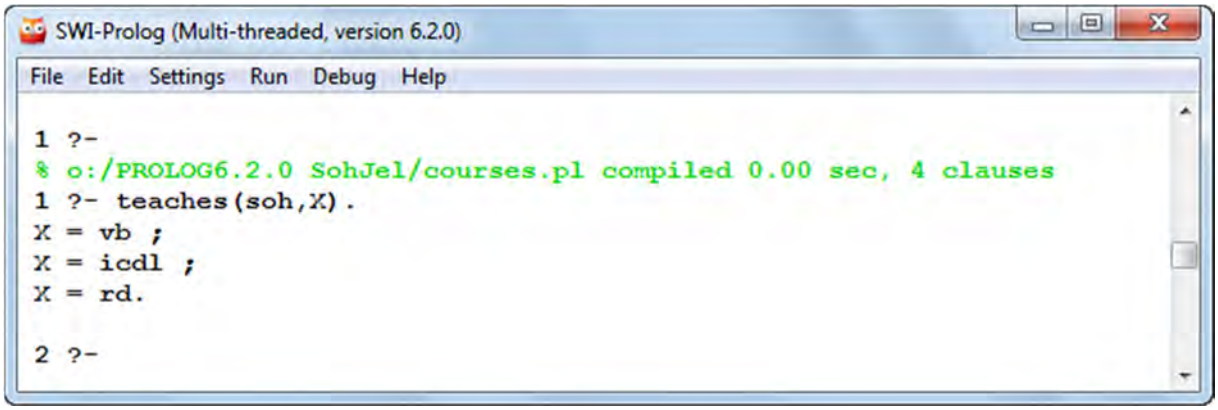
و فایلی را که به وجود آوردیم(courses.pl) را باز می‌کنیم:



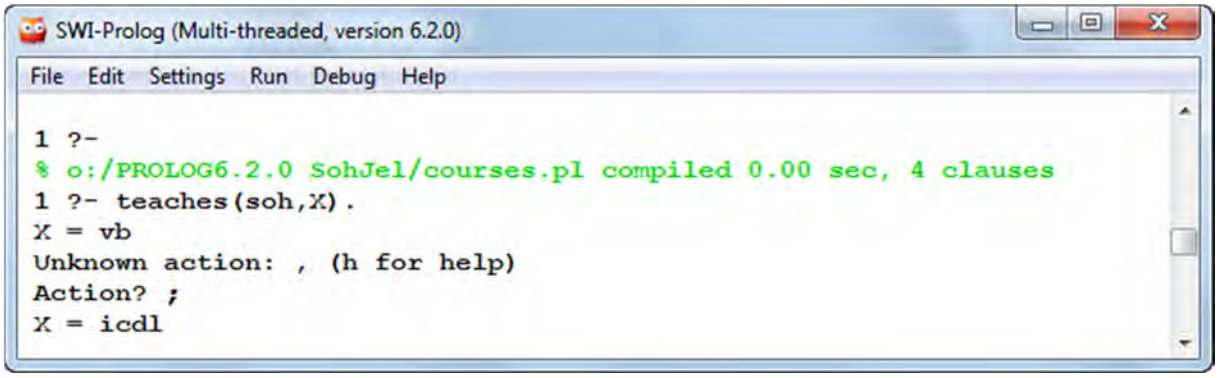


حال می‌توانیم پرسش‌هایی مثل موارد زیر را بیان کرده و پاسخ خود را دریافت نماییم:

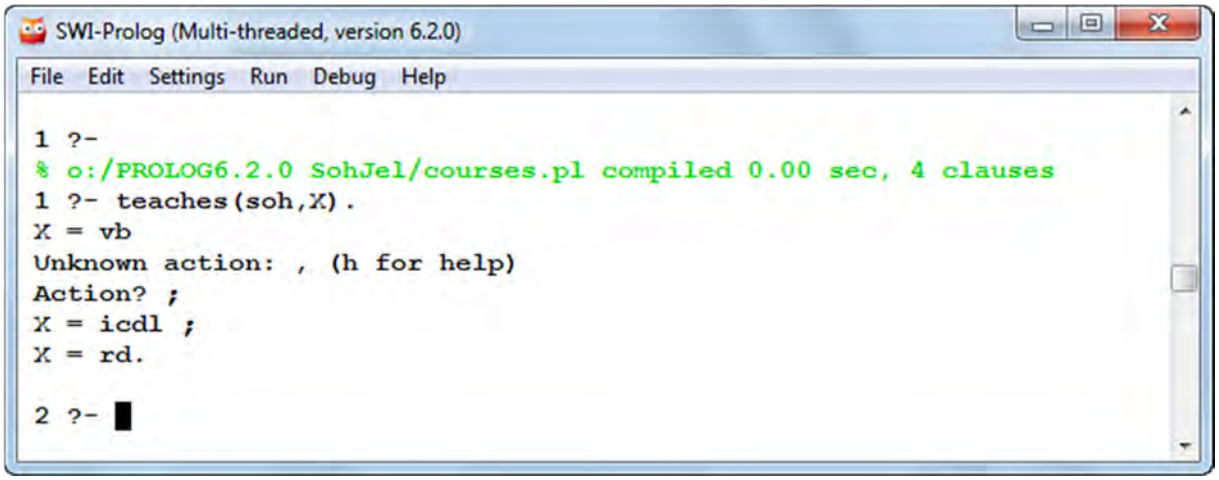




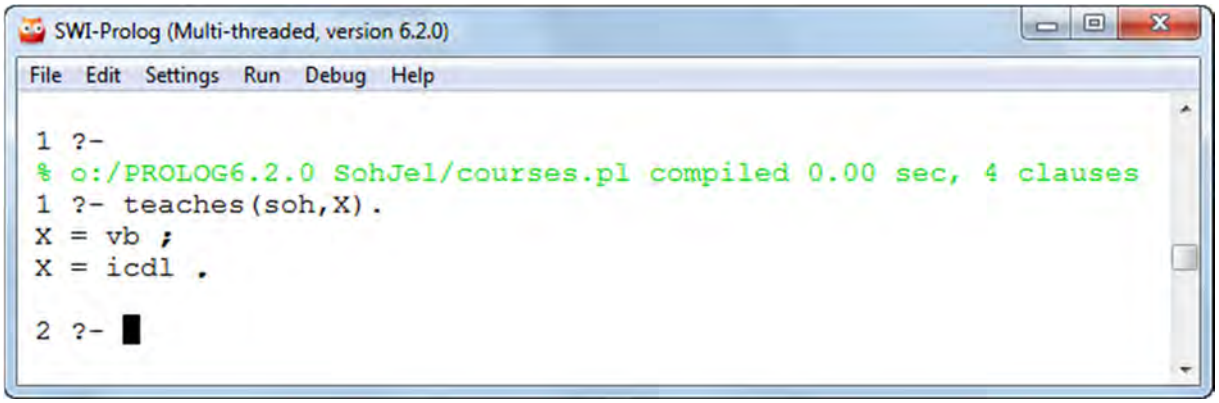
همان طور که در بالا دیدیم، بعد از گرفتن هر جواب، می‌توانیم برای دریافت جواب بعدی، از یکی از کلیدهای «;» یا «n» یا «r» یا «Spacebar» یا «Tab» روی صفحه‌کلید استفاده نماییم. *توجّه کنید که اگر برای دریافت جواب بعدی از کاما(,) استفاده کنیم، با پیام «Unknown action» برمی‌خوریم:*



برای ادامه (همان طور که در شکل بالا می‌بینیم) می‌توانیم از علامت «;» استفاده کنیم:



برای متوقّف کردن گرفتن جواب‌ها هم می‌توانیم از کلید «اینتر[1]» روی صفحه کلید استفاده نماییم:

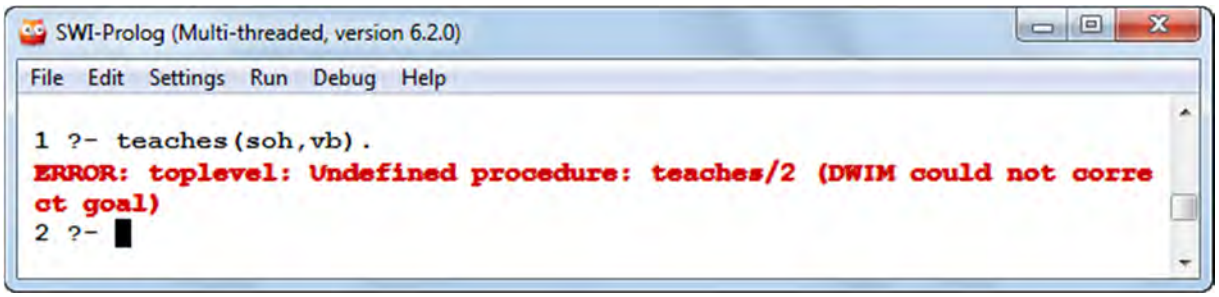


۱- Enter

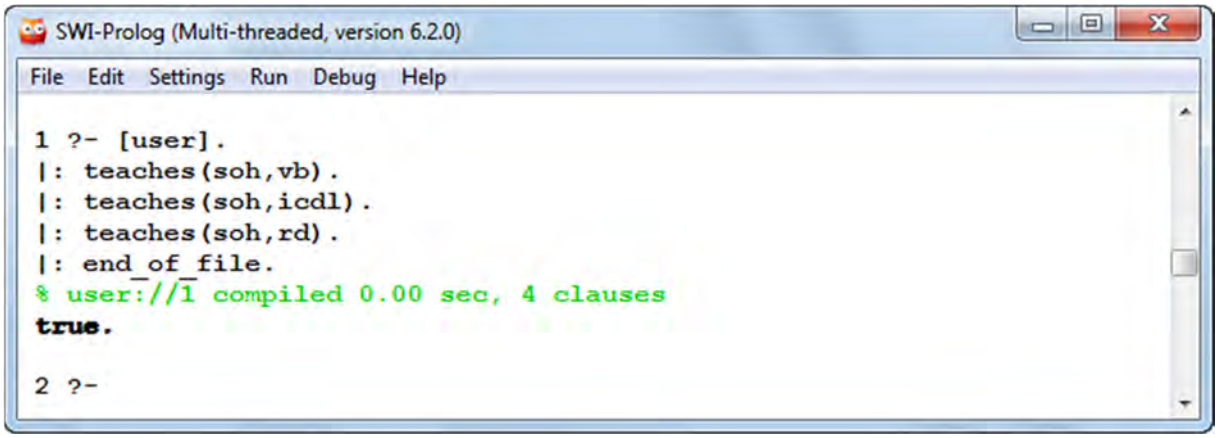
مثال اجرای یک برنامه در اعلان SWI-Prolog:

توجّه:

اگر بخواهیم یک برنامه، مثلاً پایگاه دانش مثال قبل را در اعلان SWI-Prolog بنویسیم، با خطای زیر برخورد می‌کنیم:

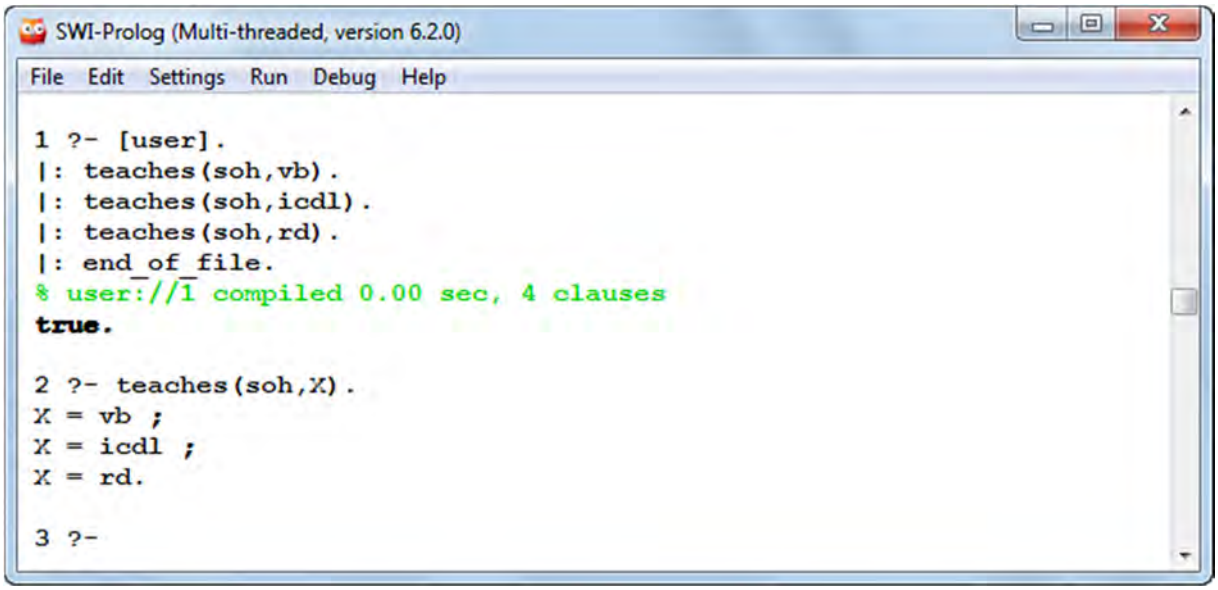


برای اینکه این کار را انجام دهیم، می‌توانیم به صورتی که در زیر آمده است، اقدام نمائیم و برنامه‌ی خود را بنویسیم:

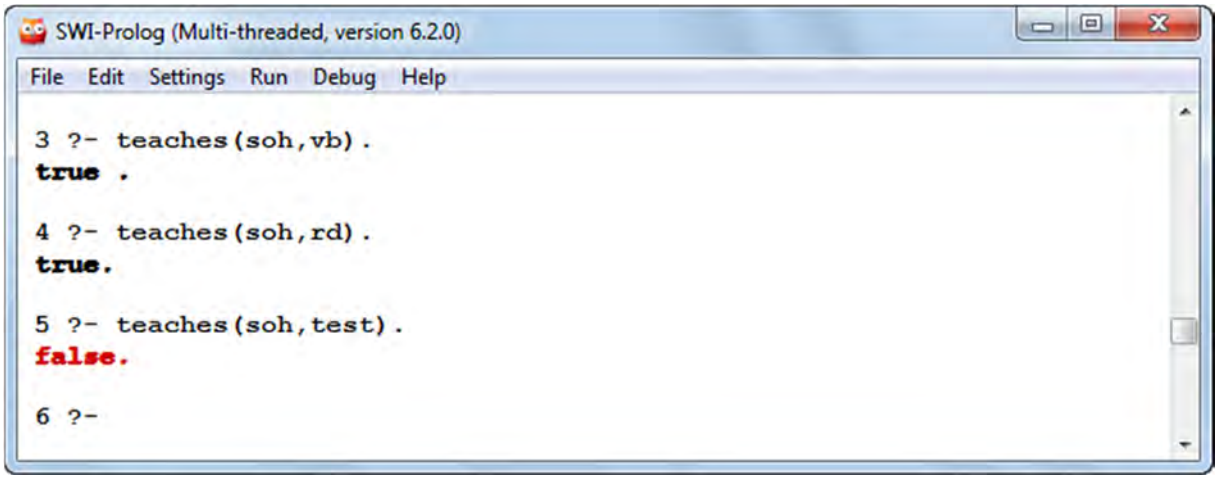


در شکل قبل، به جای نوشتن end_of_file می‌توان از ترکیب کلیدی CTRL+D استفاده کرد.

حال می‌توانیم سؤالات خود را همانند آنچه که در مثال قبل دیدیم، بیان کنیم:



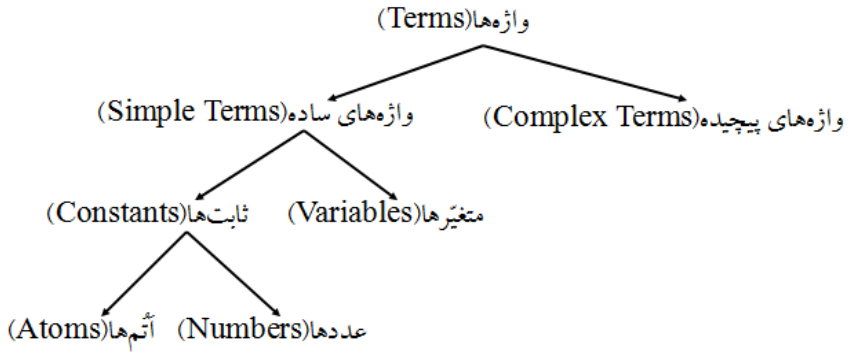
همچنین چه برنامه‌ی خود را در فایل بنویسیم و چه در اعلان SWI-Prolog، می‌توان سؤالات بیش‌تری را هم مثل آنچه که در شکل زیر می‌بینیم، بپرسیم:



نوشتن برنامه در اعلان SWI-Prolog، ایده‌ی خوبی است، ولی بیش‌تر عادت کنید که برنامه‌ی خود را در فایل بنویسید، چون هر آنچه شما در اعلان پرولوگ نوشته‌اید، به محض خارج شدن از Prolog، از بین خواهد رفت!.

واژه(تِرم)های پرولوگ[1]

تعریف– واژه‌ها، عناصری هستند که می‌توانند به صورت آرگومان‌های گزاره‌ها ظاهر شوند. واژه‌هایی که شامل هیچ متغیّری نمی‌باشند، *زَمینه*[2] نام دارند. یک ویژگی خاصّ پرولوگ این است که می‌تواند هم داده‌های زمینه و هم داده‌های غیرزمینه را پردازش نماید. یک برنامه‌ی پرولوگ می‌تواند کارهای مفید را با یک ساختمان داده حتّا در زمانی که ساختمان داده‌ی آن تاحدودی ناشناخته می‌باشد، انجام دهد.



واژه‌های ساده

متغیّرها– مثل:

_, _35, Person, Left_Subtree, Chris, Y31, X

ثابت‌ها–

اعداد– مثل:

-6.31, -10, 5.6, 3

اتم‌ها– اتم‌ها از موارد زیر می‌توانند باشند:

یک رشته از کاراکترهای با حرف‌های بزرگ، کوچک، عددها، کاراکتر زیر خط(_) است که *با یک حرف کوچک آغاز می‌شوند.*

یک رشته‌ی دلخواه از کاراکترها که بین دو کاراکتر آپاستارُف(کوتیشن تکی) قرار گرفته‌اند، مثل:

'Ali'، 'reZA 20'، '654321'، '!@#'

یک رشته از کاراکترهای ویژه مثل:

---

۱ – prolog terms

۲ – ground

:- . ; , :

مثال- موردهای زیر اتم هستند.

[ ], 'Hello there', x2, z, tom, apple

واژه‌های پیچیده(مرکّب)

اتم‌ها، عددها و متغیّرها اجزای سازنده‌ی واژه‌های پیچیده هستند. واژه‌های پیچیده، مثل:

fatherOf(ahmad,hasan)

plays(john,football)

addof(X,Y,Z)

واژه‌های پیچیده می‌توانند به صورت تودرتو در واژه‌های پیچیده‌ی دیگر به کار روند:

plays(father(father(father(javad))),pingpong)

محاسبه‌های موفّق

به عنوان مثال، در صورتی که در [کد] برنامه داشته باشیم: «likes(bob[1],prolog).»، در این صورت پاسخ سؤال زیر True خواهد بود و یک محاسبه‌ی موفّق داریم:

«?-likes(bob,prolog).»

محاسبه‌های موفّق می‌توانند یگانه یا چندگانه باشند؛ در زیر با محاسبه‌های یگانه و محاسبه‌های چندگانه آشنا می‌شویم:

محاسبه‌های یگانه

تعریف- در موقع فراخوانی، اگر فقط **یک** محاسبه تولید شود (چه موفّقیّت‌آمیز باشد و چه نباشد)، محاسبه، یگانه نامیده می‌شود؛ در این صورت درخت جستجو از یک شاخه‌ی تک تشکیل شده است.

---

۱ - بابْ: در زبان انگلیسی نامی برای یک مرد است و شکلی از کلمه‌ی رُبِرت(Robert) می‌باشد. (Babylon> Babylon English)

مثال- با استفاده از «append[1](input,input,input)» موجود در پرولوگ می‌توان رشته‌ها را به هم متّصل نمود؛ پرس‌وجوی

?-append([a,b],[c,d],[a,b,c,d]).

فقط دارای یک محاسبه می‌باشد که موفّق هم می‌شود. اگر در محیط پرولوگ عبارت بالا را بنویسیم، جواب «true.» را دریافت می‌کنیم.

در صورتی که در پرولوگ عبارت زیر را بنویسیم، با جواب «A = [a, b, c].» روبه‌رو می‌شویم:

append([a],[b,c],A).

محاسبه‌های چندگانه

تعریف- در حالت کلّی، در پرولوگ، محاسبه‌ها به صورت چندگانه می‌باشد؛ وقتی که در فراخوانی، چند محاسبه داشته باشیم، محاسبه‌های چندگانه داریم؛ در این زمان، درخت جستجو دارای چند شاخه می‌باشد. محاسبه‌های موفّق چندگانه ممکن است جواب‌های متمایزی را نتیجه بدهند یا جواب‌های متمایزی را نتیجه ندهند. هر جواب، یک نتیجه‌ی منطقی از برنامه می‌باشد.

در صورتی که چند محاسبه داشته باشیم(درخت جستجو دارای چند شاخه باشد)، پرولوگ در هر لحظه فقط یک شاخه را تولید می‌نماید. هر کدام از محاسباتی که در حال حاضر در حال ایجاد می‌باشند، پرولوگ به آنها می‌پردازد تا اینکه یا موفّق شوند و یا به صورت محدود، دارای عدم موفّقیّت شوند. این روش، *جستجوی اوّلْ عمق* است که یک روش *ناعادلانه*[2] می‌باشد و در آن ضمانتی برای تولید همه‌ی محاسبه‌ها وجود ندارد، مگر اینکه همه محدود باشند. در زمانی که یک محاسبه خاتمه می‌یابد، پرولوگ به شاخه‌هایی که زودتر ارائه شده‌اند و تاکنون به آنها نرفته است، برگشت به عقب[3] می‌نماید. ارزیابی فقط در زمانی که هیچ مورد این جوری باقی نمانده باشد، خاتمه می‌یابد. ترتیبی که در آن، شاخه‌ها امتحان می‌شوند، وابسته به ترتیب متنی شرط‌های انتساب داده شده در برنامه می‌باشد؛ این، *قانون جستجوی*[4] پرولوگ نام دارد *که شاخه‌ها را در درخت جستجو اولوّیت‌بندی می‌نماید.*

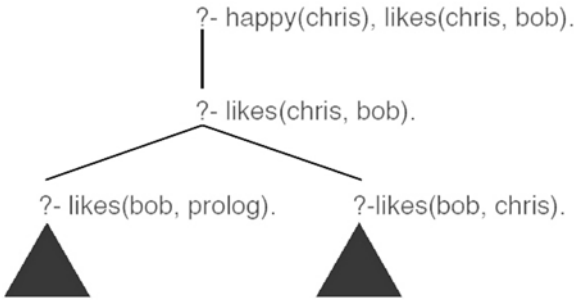
مثال- پرس‌وجوی «?-happy(chris),likes(chris,bob).» در صورتی که در [کد] برنامه داشته باشیم:

happy(chris).
likes(chris,bob):-likes(bob,prolog).
likes(chris,bob):-likes(bob,chris).

و...، دارای یک درخت جستجو است، که در آن، هر شاخه، یک محاسبه‌ی مجزّا می‌باشد:

---

۱ - در لغت یعنی: «افزودن»
۲ - unfair
۳ - backtrack
۴ - search rule



این زیردرخت‌ها بسته به اینکه دیگر چه چیزهایی در [کد] برنامه باشد، ممکن است موفّق یا ناموفّق باشند.

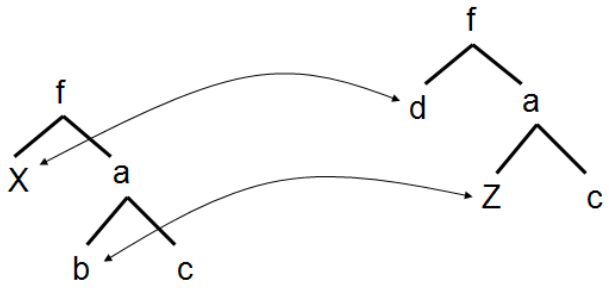
یکسان‌سازی(یکی‌سازی)[۱]

تعریف– دو عبارت در پرولوگ یکسان هستند، اگر برای هر متغیّر موجود در آنها(دو عبارت) موردی وجود داشته باشد که آنها(دو عبارت) را برابر نماید.

مثال– اگر داشته باشیم «likes(bob,Y):-understand[۲]s(bob,Y).»، پرس‌وجوی «?-likes(U,chris).»، با قرار دادن U/bob و Y/chris می‌تواند حل(یکسان) شود. این پِروسه، *یکسان‌سازی* نام دارد و پرس‌وجوی به وجود آمده به صورت زیر می‌باشد:

?-understands(bob,chris).

مثال– دو عبارت f(d,a(Z,c)) و f(X,a(b,c)) یکسان هستند:



اگر X را برابر با d قرار دهیم(X/d) و Z را برابر با b قرار دهیم(Z/b)، عبارت‌ها برابر می‌شوند. اگر در محیط پرولوگ بنویسید:

?- f(X,a(b,c))=f(d,a(Z,c)).

خروجی زیر را خواهید دید:

---

۱ – unification

۲ – در لغت یعنی: «فهمیدن»

X=d,

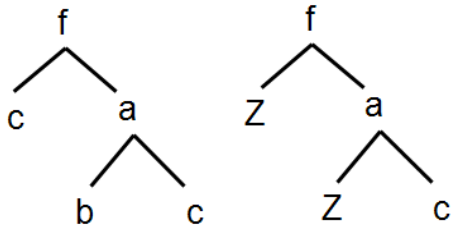
Z=b.

نکته‌ی مهم:

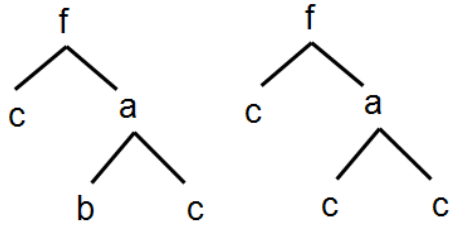
از عملگر مساوی(=) برای برّرسی یکسان بودن دو عبارت استفاده می‌نماییم.

مثال- آیا دو عبارت f(Z,a(Z,c)) و f(c,a(b,c)) با هم برابر هستند؟

در ابتدا- برای این دو عبارت، شکل زیر را خواهیم داشت:



گام بعد- حال اگر Z را با c جایگزین نماییم، داریم:



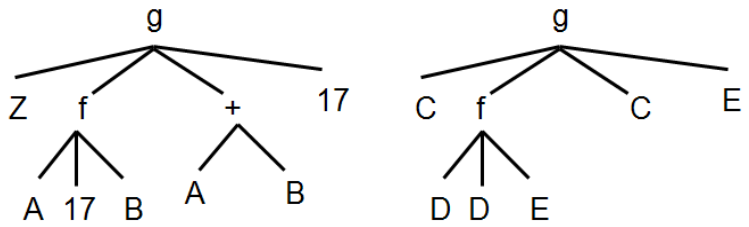
پس همان طور که می‌بینید، دو عبارت با هم برابر نمی‌باشند. اگر در محیط پرولوگ عبارت زیر را بنویسید، با پاسخ false از سوی پرولوگ مواجه می‌شوید:

f(c,a(b,c))=f(Z,a(Z,c)).

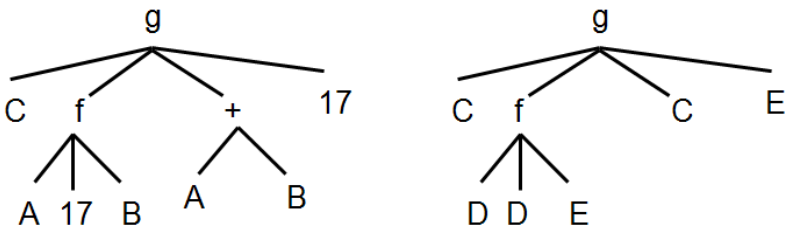
false.

مثال- آیا دو عبارت g(C,f(D,D,E),C,E) و g(Z,f(A,17,B),A+B,17) با هم برابرند؟

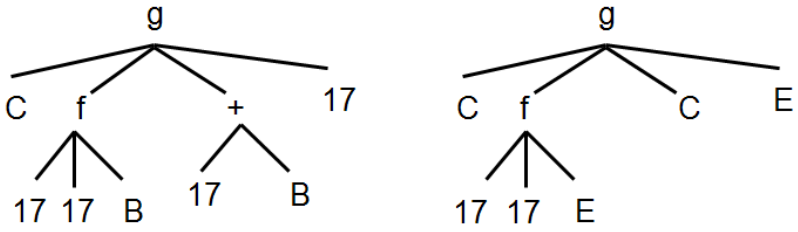
در ابتدا- برای این دو عبارت، شکل زیر را خواهیم داشت:



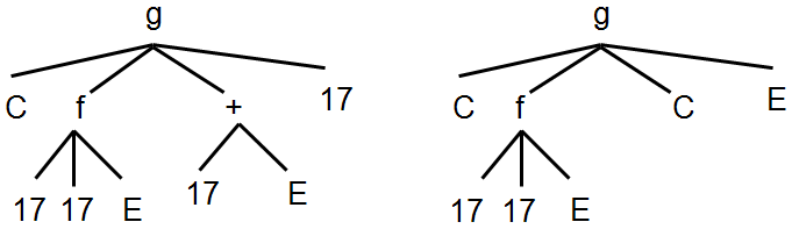
گام نخست- به جای Z، C را قرار می‌دهیم:



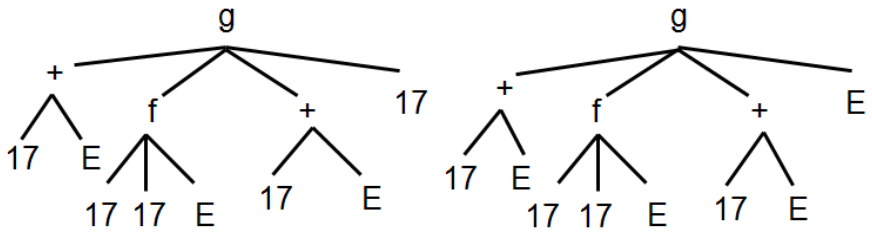
گام بعدی- به جای A، D را قرار می‌دهیم و سپس به جای D، ۱۷ را قرار می‌دهیم:



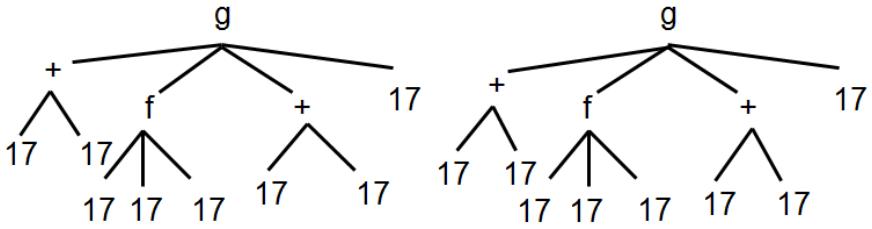
گام بعدی- به جای B، E را قرار می‌دهیم:



گام بعدی– به جای C، E + ۱۷ را قرار می‌دهیم:



در انتـها– به جای E، ۱۷ را قرار می‌دهیم:



در نتیجه می‌بینید که دو عبارت باهم برابر هستند. اگر در پرولوگ این عبارت را بنویسیم، با جواب زیر مواجه می‌شویم:

g(Z,f(A,17,B),A+B,17)=g(C,f(D,D,E),C,E).

Z = 17+17,

A = 17,

B = 17,

C = 17+17,

D = 17,

E = 17.

اگر عبارت‌های بالا را جایگزین نماییم، به لیست برابر خواهیم رسید.[۱]

پردازش لیست

تعریف– لیست، رشته‌ای محدود از عنصرها است.

۱ – http://www.cl.cam.ac.uk

مثال- [ ]، یک لیست تهی می‌باشد؛ [۳، ۲، ۱]، یک لیست دارای سه عدد می‌باشد؛ [bob، ted[۱]، alice[۲]]، لیستی دارای سه عبارت(ثابت) می‌باشد و لیست [a،[۳، ۲، ۱]،[ ]]، لیستی از لیست‌ها (و یک عبارت) می‌باشد.

لیست، از دو قسمت سَر[۳](H) و دنباله یا دُم[۴](T) تشکیل می‌شود؛ اگر n، طول لیست باشد، قسمت اوّل لیست، سر نـام دارد و بقیّـه‌ی لیست (که *لیستی* با طول n-1 است)، دم یا دنباله نام دارد.

عملگر «|»[۵]

پرولوگ دارای عملگری به صورت «|» است، که از آن می‌توان برای تجزیّه‌ی لیست استفاده نمود.

مثال- اگر در محیط پرولوگ عبارت زیر را بنویسیم:

```
?- [S|J]=[s,o,h,j,e,l].
```

این جواب را دریاقت خواهیم کرد:

```
S = s,
J = [o, h, j, e, l].
```

مثال- اگر در محیط پرولوگ عبارت زیر را بنویسیم:

```
?- [H|T]=[[ ],[1,2,3],a].
```

این جواب را دریاقت خواهیم کرد:

```
H=[ ]
T=[[1,2,3],a]
```

نکته:

[[ ]|X]، به صورت [X] نوشته می‌شود.

مثال- در لیست‌های زیر سر و دم(دنباله) را مشخّص نمایید.

در لیست [a,b,c]، a، سر و [b,c]، دنباله می‌باشد.

---

۱ - تِدْ، در زبان انگلیسی نامی برای یک مرد می‌باشد و شکلی از نام اِدوارد(Edward) یا تِئودُور(Theodore) است. (Babylon> Babylon English)

۲ - آلیسْ، در زبان انگلیسی نامی برای یک زن است. (Babylon> Babylon English)

۳ - Head

۴ - Tail

۵ - برای تایپ این کاراکتر(|) باید ترکیب کلیدهای «Shift» و «\» روی صفحه‌کلید کامپیوتر را فشار دهیم.

در لیست [a]، سر، a و دنباله، برابر با تهی([]) می‌باشد.

[](تهی)، لیست نمی‌باشد و بنابراین، دارای سر و دنباله نیست.

در لیست [[the,cat],sat]،
[the,cat]، سر و [sat]، دنباله است.

در لیست [[the, cardinal], [pulled, [off]], [each, [plum, coloured], shoe]]، سر و دنباله، به صورت زیر است:
H=[the,cardinal] , T=[[pulled, [off]], [each, [plum, coloured], shoe]]

توابع موجود برای لیست‌ها

پرولوگ دارای کتابخانه‌ای از تابع‌های لیست می‌باشد، نظیر:

تابع append( )

به صورت کلّی append(X,Y,Z) می‌باشد و Y را به X اضافه می‌کند و Z را برمی‌گرداند.

مثال- اگر در پرولوگ عبارت زیر را بنویسیم، پاسخ «A = [a,b,c].» را دریافت خواهیم نمود:

append([a,b],[c],A).

تابع reverse[1]( )

به صورت reverse(X,Y) می‌باشد و Y، معکوس X می‌باشد.

مثال- با نوشتن «reverse([a,b,c],A).» در محیط پرولوگ، جواب «A = [c,b,a].» را دریافت می‌نماییم.

تابع length[2]( )

به صورت length(X,N) می‌باشد و N، طول X است.

مثال- با نوشتن «length([s,o,h],J).» در محیط پرولوگ، جواب «J = 3.» را دریافت می‌نماییم.

---

۱ - در لغت یعنی: «واژگون کردن، معکوس کردن»

۲ - در لغت یعنی: «درازا، طول»

تابع member[1]( )

این تابع به صورت کلّی member(U,X) می‌باشد و اگر عنصر U، در لیست X باشد، true را برمی‌گرداند و اگر U در X نباشد، false را برمی‌گرداند.

مثال‌ها:

```
?- member(a,[a,b,c]).
true.

?- member([a],[a,b,c]).
false.
```

تابع sort[2]( )

این تابع که دارای شکل کلّی sort(X,Y) می‌باشد، X را مرتّب می‌نماید و در Y قرار می‌دهد.

مثال:

```
?- sort([a,c,b],M).
M = [a,b,c].
```

مثال:

```
?- sort([4,s,o,2,1,h],_SJ).
_SJ = [1,2,4,h,o,s].
```

برّرسی نوع[3]

برای برّرسی انواع آرگومان‌ها می‌توانید از موردهای زیر استفاده نمایید:

---

۱ - در لغت یعنی: «عضو»

۲ - در لغت یعنی: «مرتّب کردن، منظّم کردن»

۳ - Type-checking

atom(X)

اگر X اتم باشد، true را برمی‌گرداند و اگر نباشد، false را برمی‌گرداند.

مثال‌ها:

```
?- atom(S).
false.

?- atom(t).
true.

?- atom(:).
true.

?- atom(Sj).
false.

?- atom(_xyz).
false.

?- atom(x_y_z).
true.

?- atom(s,j).
ERROR: Undefined procedure: atom/2
ERROR:     However, there are definitions for:
ERROR:         atom/1
false.
```

در اینجا(مورد بالا) به دلیل اینکه از دو آرگومان برای تابع استفاده کرده‌ایم، با خطا برخورد کرده‌ایم (تابع atom( ) فقط یک آرگومان دارد).

```
?- atom([1,2,3]).
false.

?- atom([a,b]).
```

false.

?- atom('SJ').
true.

?- atom("SJ").
false.

?- atom(A::B).
ERROR: Syntax error: Operator expected
ERROR: atom(A
ERROR: ** here **
ERROR: ::B) .

?- atom(:-).
true.

?- atom('!@#$%^&*').
true.

number(X)

اگر X عدد باشد، true را به ما می‌دهد.

مثال‌ها:

?- number(a).
false.

?- number(12).
true.

?- number(1000+10).
false.

?- number('100').
false.

?- number(100).
true.

integer(X)

برّرسی می‌کند که آیا X عدد صحیح است یا نه.

مثال‌ها:

?- integer(a).
false.

?- integer(192).
true.

?- integer(-32768).
true.

?- integer(-32769).
true.

?- integer(-40000).
true.

?- integer(40000).
true.

?- integer(3.14).
false.

?- integer(1000000).

true.

var(X)

اگر X یک متغیّر باشد، true را برمی‌گرداند.

مثال‌ها:

```
?- var(j).
false.

?- var(J).
true.

?- var(Soh).
true.

?- var('Soh').
false.

?- var([l,e,j]).
false.
```

nonvar(X)

اگر X متغیّر نباشد، true را برمی‌گرداند.

مثال‌ها:

```
?- nonvar(j).
true.

?- nonvar(J).
false.
```

مقایسه‌ی عبارت‌ها

پرولوگ دارای عملگرهایی برای مقایسه‌ی عبارت‌ها می‌باشد:

عملگر «=»

مطلب مهم:

همان طور که در گذشته هم دیدیم، این عملگر برای برّرسی *یکسان بودن* دو عبارت می‌باشد. مثلاً در عبارت X=a، اگر به جای X، a را قرار دهیم، عبارت یکسان خواهد شد.

عملگر «==»(دو مساوی پشت سر هم)

اگر بخواهیم برّرسی کنیم که آیا دو عبارت با هم دقیقاً برابر هستند یا نه از این عملگر استفاده می‌نماییم.

مثال‌ها:

```
?- 3==3.
true.

?- 3==X.
false.

?- 5==6.
false.

?- [a,b]==[a,b].
true.

?- [a,b]=[b,a].
false.

?- [a,b]==[a,B].
false.
```

عملگر «=\»

برای برّرسی اینکه آیا دو عبارت نامساوی هستند یا نه، از این عملگر استفاده می‌نماییم؛ اگر دو عبارت نامساوی داشته باشیم، حاصل true می‌باشد و اگر دو عبارت با هم مساوی باشند، حاصل false خواهد بود.

مثال‌ها:

```
?- 3\=3.
false.

?- 3\=4.
true.
```

محاسبه

برای محاسبه، از عملگرهای «+»، برای «انجام عمل جمع»؛ «-»، برای «انجام عمل تفریق»؛ «*»، برای «انجام عمل ضرب»؛ «/»، برای «انجام عمل تقسیم»؛ «**»، برای «انجام عمل به توان رساندن»؛ «//»، برای «به دست آوردن خارج قسمت تقسیم صحیح» و از «mod»، برای «به دست آوردن باقیمانده‌ی تقسیم صحیح» استفاده می‌نماییم.

نکته‌ی مهم:

اگر عبارت زیر را در محیط پرولوگ بنویسیم،

```
?- S01=2+6.
```

جواب زیر را با فشردن دکمه‌ی «اینتر» روی صفحه‌کلید، دریافت خواهیم کرد:

```
S01 = 2+6.
```

و حاصل جمع را دریافت نمی‌کنیم؛ امّا با استفاده از عملگر «is»[1]، حاصل جمع را دریافت خواهیم کرد:

```
?- S01 is 2+6.
S01 = 8.
```

مثال‌ها:

```
?- S02 is 100-1000.
S02 = -900.
```

---

۱ - این عملگر(is) را در ادامه‌ی همین فصل برّرسی خواهیم نمود.

?- S03 is 1000*1000.
S03 = 1000000.

?- S04 is 10/3.
S04 = 3.3333333333333335.

?- S05 is 10**3.
S05 = 1000.

?- S06 is 20//3.
S06 = 6.

?- S07 is 20 mod 3.
S07 = 2.

عملگر is

مطلب مهم:

عملگر is موجود در پرولوگ، دارای دو آرگومان می‌باشد؛ به دوّمین آرگومان به چشم یک عبارت محاسباتی نگاه می‌کند و آن را با آرگومان اوّل برابر قرار می‌دهد؛ *برای وادار کردن پرولوگ به محاسبه‌ی عبارت‌های محاسباتی، مثل جمع، ضرب، تقسیم و غیره، همان طور که در گذشته دیدیم، می‌توان از عملگر is استفاده کرد.*

مثال‌ها:

توجّه:

عددی که در این مثال ظاهراً به صورت توان می‌باشد، توان نیست و معرّف پاورقی است؛ البتّه چنانچه قبلاً گفتیم، برای به توان رساندن در پرولوگ از عملگر «**» استفاده می‌نماییم.

?- A is 2+3*4[1].

A = 14.

?- 10 is 2*5.

true.

?- A is 2+3-1.

A = 4.

?- X+4 is 12.

false.

?- X is 12,Y is X+2.

X = 12,

Y = 14.

مثال‌های بیش‌تر:

عبارت «?- X is (2+2).» درست است و X/4 خواهد بود.

«?- 4 is (2+2).» هم درست است و خروجی True را به ما نشان خواهد داد.

نکته:

«(۲+۲)» را می‌توان به صورت پیشوندی؛ یعنی، «+(۲،۲)» هم نوشت.

مثال:

?- 4 is +(2,2).

true.

---

۱ - **توضیح**- در عبارت محاسباتی 2+3*4، به دلیل اینکه اولویّت ضرب(*) بیش‌تر از جمع(+) است، اوّل ۳*۴ محاسبه می‌شود که برابر با ۱۲ می‌باشد؛ سپس ۲ با ۱۲ جمع می‌شود که در نتیجه حاصل برابر با ۱۴ می‌شود.

نکته‌ی مهم:

توجّه کنید که «is» را با «=» اشتباه نگیرید. X=Y دارای معنی «[آیا] X ممکن است با Y ***یکسان*** شود» می‌باشد.

مثال‌ها- عبارت «?- X=(2+2).» درست است و X/(2+2) خواهد بود؛ «?- 4=(2+2).» خطا نمی‌دهد ولی دارای عدم موفّقیّت می‌باشد(False را به ما نشان می‌دهد)؛ «?- X=(Y+2).» درست است و X/(Y+2) می‌باشد:

```
?- X=(2+2).
X = 2+2.

?- 4=(2+2).
false.

?- X=(Y+2).
X = Y+2.
```

مقایسه‌ی عبارت‌های محاسباتی

عملگر «=:=»

E1=:=E2 برّرسی می کند که آیا مقادیر E1 و E2 با هم برابرند یا نه؟.

مثال‌ها:

توجّه:

عددهایی که در این مثال ظاهراً به صورت توان می‌باشند، توان نیستند و معرّف پاورقی هستند.

```
?- 2+3*4[1] =:= (2+3)*4[2].
```

---

۱ - **توضیح**- در این عبارت، در ابتدا عمل ضرب انجام می‌شود و سپس عمل جمع انجام می‌شود:

2+3*4 = 2+12 = 14

۲ - **توضیح**- در این عبارت، به دلیل اینکه اولویّت عبارت درون پَرانتز بیش‌تر است، در ابتدا عمل جمع درون پرانتز انجام می‌شود و سپس عمل ضرب انجام می‌شود:

false.

?- 2*3+4 =:= (2*3)+4.
true.

عملگر «=\=»

E1=\=E2 برّرسی می‌کند که آیا مقادیر E1 و E2 نامساویند؟.

مثال‌ها:

?- 2+3*4 =\= (2+3)*4.
true.

?- 2*3+4 =\= (2*3)+4.
false.

عملگر «<»

E1<E2 برّرسی می‌کند که آیا مقدار E1 کوچک‌تر از E2 می‌باشد یا نه؟.

مثال‌ها:

?- 5<4.
false.

?- 4<5.
true.

همچنین عملگرهای «<» (برای برّرسی بزرگ‌تر بودن)، «=<» (برای برّرسی بزرگ‌تر یا مساوی بودن) و «>=» *(برای برّرسی مساوی یا کوچک‌تر بودن)* را هم داریم.

مثال:

?- 3=<5.
true.

(2+3)*4 = 5*4 = 20

نکته:

در پرولوگ عملگر «=>» نداریم و در صورت استفاده با خطا مواجه خواهیم شد.

مثال:

```
?- 3<=5.
ERROR: Syntax error: Operator expected
ERROR: 3
ERROR: ** here **
ERROR: <=5.
```

بازگشت[1]

یادآوری- بازگشت، در هر زبانی، تابعی است که می‌تواند تا زمانی که به هدف خود برسد، خودش را فراخوانی نماید. در پرولوگ و هر زبان دیگر، بازگشت دارای دست‌کم دو مورد زیر است:

۱- یک واقعیّت که شرط پایان می‌باشد.

۲- دستوری که خودش را صدا می‌زند.

در هر مرحله، اوّلین واقعیّت برّرسی می‌شود؛ اگر درست بود، بازگشت خاتمه می‌یابد و در صورتی که درست نبود، بازگشت ادامه می‌یابد. دستور بازگشتی نباید خودش را با آرگومان یکسان فراخوانی کند؛ در صورتی که این رویداد رخ دهد، برنامه هیچوقت پایان نمی‌یابد.

مثال:

```
parent(john,paul[2]).

parent(paul,tom[3]).

parent(tom,mary).


ancestor[1](X,Y):- parent(X,Y).
```

---

۱ - recursion

۲ - پُل، در زبان انگلیسی هم نام می‌تواند باشد و هم نام خانوادگی (Babylon> Babylon English)؛ در اینجا نام یک مرد است.

۳ - تامْ، در زبان انگلیسی نامی برای یک مرد است و شکلی از کلمه‌ی توماس(Thomas) می‌باشد. (Babylon> Babylon English)

ancestor(X,Y):- parent(X,Z), ancestor(Z,Y).

این برنامه اجداد را با استفاده از پایگاه داده‌ای که در ابتدای برنامه آمده است، با استفاده از دستور بازگشتی ancestor پیدا می‌کند. اگر حالا از پرولوگ بپرسیم:

?- ancestor(john,tom).

پرولوگ در ابتدا سعی می‌کند «parent(john,tom).» را پیدا نماید، که موفّق نمی‌شود. حال شرط دوّم را برّرسی می‌کند، پرسش جدید، «parent(john,Z).» می‌شود. پرولوگ «Z=paul» را پیدا می‌کند و سعی می‌کند «ancestor(paul,tom)» را پیدا نماید و به «parent(paul,tom).» می‌رسد، که درست است؛ بنابراین، «ancestor(paul,tom)» هم درست است و در نتیجه «ancestor(john,tom)» هم درست است و خروجی true را از پرولوگ دریافت خواهیم کرد. اجرا به صورت زیر خواهد بود:

فراخوانی «ancestor(john,tom).»

فراخوانی «parent(john,tom).»

شکست «parent(john,tom).»

فراخوانی «parent(john,Z).»

تلاش برای Z=paul

فراخوانی «ancestor(paul,tom).»

فراخوانی «parent(paul,tom).»

«parent(paul,tom).» موفّق می‌شود

«ancestor(paul,tom).» موفّق می‌شود

Z=paul موفّق می‌شود

ancestor(john,tom) موفّق می‌شود

نقیض[۲](+\)

می‌توان برای معکوس کردن درستی یا غلطی یک گزاره، از نقیض استفاده کرد؛ به عنوان مثال، پایگاه داده‌ی پرولوگ زیر را در نظر بگیرید:

p(a).

---

۱ – در لغت به معنی «اجداد» یا «جد» می‌باشد.

۲ – negation

p(b).

حال اگر سؤال «p(a). +\» را در پرولوگ مطرح کنیم، جواب «no.» را دریافت خواهیم کرد. به طور معکوس، جواب «p(c).+\»، «yes.» می‌باشد.[۱]

بُرِش[۲](!)

برای انجام عمل برش، از علامت «!» استفاده می‌نماییم.

مثال- پایگاه دانش فایل «courses.pl» را که در گذشته به وجود آوردیم، به یاد بیاورید؛ در این پایگاه دانش، اگر پرس‌وجو را به صورت زیر بیان کنیم، فقط اوّلین مورد(vb) به ما نشان داده می‌شود و بقیّه‌ی موارد(icdl وrd) به ما نشان داده نخواهند شد:

```
1 ?- teaches(soh,X),!.
X = vb.

2 ?-
```

ورودی و خروجی

پرولوگ می‌تواند در یک فایل بنویسد و یا از آن بخواند. فایلی که پرولوگ از آن می‌خواند، «ورودی» نام دارد و فایلی که پرولوگ در آن می‌نویسد، «خروجی» نام دارد. وقتی که شما پرولوگ را اجرا می‌نمایید، خروجی پیش‌فرض، صفحه‌ی نمایش شما است و ورودی پیش‌فرض، صفحه‌کلید شما می‌باشد. برخی وقت‌ها یک برنامه می‌خواهد عبارتی را از یک فایل یا صفحه‌کلید بخواند و عبارتی را در یک فایل یا بر روی صفحه بنویسد. برای خواندن عبارت از ورودی فعّال، از دستور «read» استفاده می‌کنیم؛ مثلاً دستور «read(X).»، عبارت را از ورودی فعّال می‌خواند و آن را در [متغیّر] X قرار می‌دهد. دستور «write»، عبارت را در خروجی فعّال می‌نویسد؛ مثلاً دستور «write(Y).»، عبارت Y را بر روی خروجی فعّال می‌نویسد.

مثال:

```
?- write('hello').
```

خروجی:

---

۱ - **توضیح**- به دلیل اینکه p(c) در پایگاه داده‌ی این مثال وجود ندارد، به طور پیش‌فرض برابر با false می‌باشد و درنتیجه نقیض آن برابر با true خواهد بود.

۲ - cut

hello
true.

مثال:

?- write('hello'),write('world').

خروجی:

helloworld
true.

در مثال دوّم، اگر بخواهیم کلمه‌ی world در خطّ جدید نوشته شود، از nl(اِنْ اِلْ) استفاده می‌نماییم.

مثال:

?- write('hello'), nl,write('world').

خروجی:

hello
world
true.

مثال:

?- read(X),write(X).

در این هنگام پرولوگ منتظر وارد کردن مقدار X می‌ماند و در ابتدای خط، علامت «:|» را مشاهده می‌نمایید؛ اگر بخواهیم به X مقدار ۱۲ را بدهیم، باید بعد از نوشتن عدد ۱۲، از کاراکتر نقطه(.) استفاده نمایید و سپس کلید «Enter» را فشار دهیم، که در این صورت خروجی زیر را خواهیم دید:

|: 12.
12
X = 12.

استفاده از کد اَسکی[1]

می‌توانید در پرولوگ از کد اسکی هم استفاده نمایید؛ به عنوان مثال، اگر در پرولوگ عبارت زیر را بنویسید، خروجی «ABC.» و true را خواهید داشت[1]:

---

۱ - *ASCII*(American Standard Code for Information Interchange) code؛ در کامپیوتر، هر کاراکتر، دارای یک کد است، که به این کد، «کد اسکی» می‌گویند. مثلاً کد اسکی کاراکتر A، ۶۵ است.

```
?- put(65),put(66),put(67).
```

چکیده‌ی مطلب‌های فصل بیست و چهارم

برنامه‌ها از تعریف روال‌ها تشکیل شده‌اند؛ یک روال، منبعی برای ارزیابی چیزی می‌باشد.

در پرولوگ، «:-»، به معنای «if(اگر)»؛ «,»، به معنای «and»؛ و «;»، به معنای «or» می‌باشد.

پرولوگ فراخوانی‌های در یک پرس‌وجو را به صورت ترتیبی و با همان ترتیبی که از چپ به راست نوشته شده‌اند، ارزیابی می‌کند.

واژگان شروع شده با یک حرف بزرگ یا خطّ زیرین(_)، متغیّر هستند.

گزاره‌ها و ثابت‌ها همیشه با حروف کوچک یا عدد شروع می‌شوند.

در پرولوگ دو نوع عبارت داریم؛ یکی، قانون‌ها که دارای علامت «:-» هستند؛ و دیگری، واقعیّت‌ها که دارای علامت «:-» نمی‌باشند.

قانون «a:-b,c.»؛ یعنی، «اگر b و c (هر دو) درست باشند، آنگاه a درست خواهد بود.»

هر واقعیّت، مثل «happy(mary).» فقط دارای یک گزاره است.

هر عبارت در پرولوگ با نقطه(.) خاتمه می‌یابد.

تعریف یک گزاره در پرولوگ تقریباً با تعریف یک زیربرنامه برابر است و گزاره‌هایی که در بدنه‌ی شرط می‌آیند، تقریباً با فراخوانی زیربرنامه برابر هستند.

توضیح‌ها به وسیله‌ی پرولوگ پردازش نمی‌شوند؛ به عبارت دیگر، کاری بر روی آنها انجام نمی‌شود.

متغیّرهایی که درون پرس‌وجوها هستند، به صورت سور وجودی(∃) رفتار می‌نمایند.

متغیّرهای درون شرط‌های برنامه به صورت سورهای عمومی(∀) عمل می‌کنند.

واژه‌ها عناصری هستند که می‌توانند به صورت آرگومان‌های گزاره‌ها ظاهر شوند.

اتم‌ها، عددها و متغیّرها اجزای سازنده‌ی واژه‌های پیچیده هستند.

دو عبارت در پرولوگ یکسان هستند، اگر برای هر متغیّر موجود در آنها(دو عبارت) موردی وجود داشته باشد که آنها(دو عبارت) را برابر نماید.

از عملگر مساوی(=) برای برّرسی یکسان بودن دو عبارت استفاده می‌نماییم.

لیست، رشته‌ای محدود از عنصرها است.

لیست، از دو قسمت سر(H) و دنباله یا دم(T) تشکیل می‌شود؛ اگر n، طول لیست باشد، قسمت اوّل لیست، سر نام دارد و بقیّه‌ی لیست (که لیستی با طول n-1 است)، دم یا دنباله نام دارد؛ به عنوان مثال، در لیست [a,b,c]، a، سر و [b,c]، دنباله می‌باشد.

در پرولوگ، از عملگرهای «/»، برای «انجام عمل تقسیم»؛ «**»، برای «انجام عمل به توان رساندن»؛ «//»، برای «به دست آوردن خارج قسمت تقسیم صحیح» و از «mod»، برای «به دست آوردن باقیمانده‌ی تقسیم صحیح» استفاده می‌نماییم.

برای وادار کردن پرولوگ به محاسبه‌ی عبارت‌های محاسباتی، مثل جمع، ضرب، تقسیم و غیره می‌توان از عملگر is استفاده کرد.

در پرولوگ برای برّرسی مساوی یا کوچک‌تر بودن از عملگر «=<» استفاده می‌کنیم و عملگر «<=» نداریم.

بازگشت، در هر زبانی، تابعی است که می‌تواند تا زمانی که به هدف خود برسد، خودش را فراخوانی نماید.

یادآوری یا تکمیل مطلب‌های فصل بیست و چهارم

مطلب- ☑**کنکور سراسری مهندسی کامپیوتر سال ۸۴-** زنجیره‌ی پَسرُو(معکوس)، مکانیزم اساسی اجرای برنامه‌های پرولوگ است.[۱]

سؤال- نوع داده‌ای L_ چیست؟

جواب- متغیّر.[۲]

سؤال- نوع داده‌ای x_ چیست؟

جواب- متغیّر.[۳]

سؤال- نوع داده‌ای 50.1 چیست؟

جواب- عدد.[۱]

---

۱ - مطالب درس «مفاهیم زبان‌های برنامه‌نویسی(Programming Language Concepts)» استاد، «دِیوید سْتاتْس(David Stotts)»، دانشگاه ایالت کارُلینای شمالی(North Carolina)ی کشور آمریکا، بهار سال ۲۰۰۴ میلادی

۲ - آزمون پایان‌ترم درس «آزمایشگاه مفاهیم پرولوگ(Prolog Concepts Laboratory)» استاد، «دیانا اینکِپن(Diana Inkpen)»، دانشکده‌ی فنّـآوری اطّلاعات و مهندسی دانشگاه شهر اُتاوا(Ottawa)ی کشور کانادا، ۲۷ آوریل سال ۲۰۰۶ میلادی

۳ - آزمون پایان‌ترم درس «آزمایشگاه مفاهیم پرولوگ» استاد، «دیانا اینکپن»، دانشکده‌ی فنّـآوری اطّلاعات و مهندسی دانشگاه شهر اتاوای کشور کانادا، ۲۰ دسامبر سال ۲۰۰۵ میلادی

سؤال- نوع داده‌ای var1 چیست؟

جواب- اتم.[۲]

درست یا غلط- در پرولوگ، بهترین راه برای بیان ارتباط برادری میان سهراب و فرهاد، استفاده از لیست [sohrab,farhad] است.

جواب- «**غلط**» **است؛** لیست، یک مجموعه را نشان می‌دهد و نه یک ارتباط را. در واقع بهترین راه برای نمایش ارتباط برادری، استفاده از گزاره‌ای شبیه «brother(sohrab,farhad).» است.[۳]

درست یا غلط- در دستور [A|B]، B آخرین عنصر لیست را می‌دهد.

جواب- «**غلط**» **است؛** A، سَرِ لیست و B، دُم(دنباله‌ی) لیست است.[۴]

سؤال- حاصل عبارت زیر چیست؟

```
?- [X|Y]=[1,2,3].
```

جواب-

```
X = 1,
Y = [2, 3].
```[۵]

سؤال- حاصل عبارت زیر چیست؟

```
?- f[1,2,3]=f[X|Y].
```

جواب-

```
X = 1,
Y = [2, 3].
```[۱]

---

۱ - آزمون پایان‌ترم درس «آزمایشگاه مفاهیم پرولوگ» استاد، «دیانا اینکپن»، دانشکده‌ی فنّ‌آوری اطّلاعات و مهندسی دانشگاه شهر اتاوای کشور کانادا، ۲۰ دسامبر سال ۲۰۰۵ میلادی

۲ - آزمون پایان‌ترم درس «آزمایشگاه مفاهیم پرولوگ» استاد، «دیانا اینکپن»، دانشکده‌ی فنّ‌آوری اطّلاعات و مهندسی دانشگاه شهر اتاوای کشور کانادا، ۲۷ آوریل سال ۲۰۰۶ میلادی

۳ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۱۵ فوریّه‌ی سال ۲۰۱۲ میلادی

۴ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۱۵ فوریّه‌ی سال ۲۰۱۲ میلادی

۵ - آزمون پایان‌ترم درس «آزمایشگاه مفاهیم پرولوگ» استاد، «دیانا اینکپن»، دانشکده‌ی فنّ‌آوری اطّلاعات و مهندسی دانشگاه شهر اتاوای کشور کانادا، ۲۷ آوریل سال ۲۰۰۶ میلادی

سؤال- حاصل عبارت زیر چیست؟

?- [X|Y]=[f(a),f(b)].

جواب-

X = f(a),
Y = [f(b)].[۲]

سؤال- حاصل عبارت زیر چیست؟

?- [X|Y]=[f(a)].

جواب-

X = f(a),
Y = [ ].[۳]

سؤال- حاصل عبارت زیر چیست؟

?- [X|[Y|Z]]=[1,2,3].

جواب-

X = 1,
Y = 2,
Z = [3].[۴]

برای سادگی می‌توان [Y|Z] را W فرض کرد:

[X|W]=[1,2,3].
X=1
W=[2,3]
[Y|Z]=[2,3]
Y=2
Z=[3]

سؤال- کلز(عبارت)ی برای نسبت خانوادگی «خواهر» در پرولوگ بنویسید.

---

۱ - آزمون پایان‌ترم درس «آزمایشگاه مفاهیم پرولوگ» استاد، «دیانا اینکپن»، دانشکده‌ی فنّـآوری اطّلاعات و مهندسی دانشگاه شهر اتاوای کشور کانادا، ۲۰ دسامبر سال ۲۰۰۵ میلادی

۲ - آزمون پایان‌ترم درس «آزمایشگاه مفاهیم پرولوگ» استاد، «دیانا اینکپن»، دانشکده‌ی فنّـآوری اطّلاعات و مهندسی دانشگاه شهر اتاوای کشور کانادا، ۲۷ آوریل سال ۲۰۰۶ میلادی

۳ - آزمون پایان‌ترم درس «آزمایشگاه مفاهیم پرولوگ» استاد، «دیانا اینکپن»، دانشکده‌ی فنّـآوری اطّلاعات و مهندسی دانشگاه شهر اتاوای کشور کانادا، ۲۰ دسامبر سال ۲۰۰۵ میلادی

۴ - آزمون پایان‌ترم درس «آزمایشگاه مفاهیم پرولوگ» استاد، «دیانا اینکپن»، دانشکده‌ی فنّـآوری اطّلاعات و مهندسی دانشگاه شهر اتاوای کشور کانادا، ۲۷ آوریل سال ۲۰۰۶ میلادی

جواب- X، خواهر Y است، اگر:

sister(X, Y):- female(X), sibling(X, Y).[۱]

سؤال- کلزی برای نسبت خانوادگی «برادر» در پرولوگ بنویسید.

جواب-

brother(X,Y):- male(X), sibling(X,Y).[۲]

سؤال- کلزی برای نسبت خانوادگی «uncle(دایی یا عمو)» در پرولوگ بنویسید.

جواب-

uncle(X,Y):- brother(X,Z), parent(Z,Y).[۳]

سؤال- کلزی برای نسبت خانوادگی «aunt(عمّه یا خاله)» در پرولوگ بنویسید.

جواب-

aunt(X,Y):- sister(X,Z), parent(Z,Y).[۴]

درست یا غلط- در پرولوگ عبارت «X=<Y->Z is Y;Z is X» از ساختار «if-then-else» موجود در پرولوگ استفاده می‌کند.

جواب- «درست» است؛ در پرولوگ از ساختار «A->B;C» برای «if A then B else C» استفاده می‌شود. بنابراین، عبارت بالا می‌تواند به صورت «if(X=<Y) then Z=Y else Z=X» تفسیر شود.[۵]

سؤال- آیا دو گزاره‌ی ancestor(john,mary) و ancestor(X,Y,[ ]) قابل یکسان‌سازی هستند؟

جواب- «نه»؛[۱] چون تعداد آرگومان(عملوند)های آنها با هم متفاوت است؛ تعداد آرگومان‌های ancestor(*john,mary*)، ۲ تاست، امّا تعداد آرگومان‌های ancestor(*X,Y,[ ]*)، ۳ تاست.

---

۱ - نکته‌های درس «آزمایشگاه مفاهیم پرولوگ» استاد، «دیانا اینکپن»، دانشکده‌ی فنّـآوری اطّلاعات و مهندسی دانشگاه شهر اتاوای کشور کانادا

۲ - نکته‌های درس «آزمایشگاه مفاهیم پرولوگ» استاد، «دیانا اینکپن»، دانشکده‌ی فنّـآوری اطّلاعات و مهندسی دانشگاه شهر اتاوای کشور کانادا

۳ - نکته‌های درس «آزمایشگاه مفاهیم پرولوگ» استاد، «دیانا اینکپن»، دانشکده‌ی فنّـآوری اطّلاعات و مهندسی دانشگاه شهر اتاوای کشور کانادا

۴ - نکته‌های درس «آزمایشگاه مفاهیم پرولوگ» استاد، «دیانا اینکپن»، دانشکده‌ی فنّـآوری اطّلاعات و مهندسی دانشگاه شهر اتاوای کشور کانادا

۵ - آزمون پایان‌ترم درس «هوش مصنوعی ۱» دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبرتی دانشگاه ساپینزای کشور ایتالیا، ۷ ژوئن سال ۲۰۱۲ میلادی

سؤال- حاصل عبارت زیر چیست؟

?- [X,Y]=[f(a), f(b), c].

جواب- چون دو عبارت، به این دلیل که تعداد آرگومان(عملوند)های آنها با هم متفاوت است، قابل یکسان‌سازی نیستند، حاصل false خواهد بود.[۲]

سؤال- حاصل عبارت زیر چیست؟

?- [X,Y]=[f(a), f(b)].

جواب- دو عبارت، قابل یکسان‌سازی هستند، پس حاصل به صورت زیر خواهد بود:

X = f(a),
Y = f(b).[۳]

سؤال- آیا دو عبارت wife[۴](mary,john) و husband[۵](X,Y) قابل یکسان‌سازی هستند؟

جواب- **«نه»؛**[۶] چون نام‌های آنها(wife, husband) با هم متفاوت است.

سؤال- ☑ **کنکور سراسری مهندسی کامپیوتر سال ۸۳**- معادل منطقی برنامه‌ی پرولوگ زیر چیست؟

p:- a,!,b.
p:- c.

جواب-

P ← (a∧b)∨(¬a∧c)[۷]

---

۱ - آزمون میان‌ترم درس «مبانی هوش مصنوعی» استاد، «تیم فینین»، دانشکده‌ی مهندسی برق و علوم کامپیوتر دانشگاه شهر بالتیمور ایالت مریلند کشور آمریکا، ۱۶ اکتبر سال ۲۰۰۶ میلادی

۲ - آزمون پایان‌ترم درس «آزمایشگاه مفاهیم پرولوگ» استاد، «دیانا اینکپن»، دانشکده‌ی فنّ‌آوری اطّلاعات و مهندسی دانشگاه شهر اتاوای کشور کانادا، ۲۷ آوریل سال ۲۰۰۶ میلادی

۳ - آزمون پایان‌ترم درس «آزمایشگاه مفاهیم پرولوگ» استاد، «دیانا اینکپن»، دانشکده‌ی فنّ‌آوری اطّلاعات و مهندسی دانشگاه شهر اتاوای کشور کانادا، ۲۷ آوریل سال ۲۰۰۶ میلادی

۴ - در زبان انگلیسی در لغت به معنی «همسر» است.

۵ - در زبان انگلیسی در لغت به معنی «شوهر» است.

۶ - آزمون میان‌ترم درس «مبانی هوش مصنوعی» استاد، «تیم فینین»، دانشکده‌ی مهندسی برق و علوم کامپیوتر دانشگاه شهر بالتیمور ایالت مریلند کشور آمریکا، ۱۶ اکتبر سال ۲۰۰۶ میلادی

۷ - مطالب موجود در پایگاه اینترنتی دانشکده‌ی انفورماتیک دانشگاه ماساریک(Masaryk) کشور چک.

فصل بیست و پنجم

آشنایی با زبان برنامه‌نویسی پایتون[۲]

۱ – تصویر، گوییدُ وَنْ راسُمْ(Guido van Rossum)، متولّد ۱۹۵۶ میلادی، برنامه‌نویس کامپیوتری هُلَندی و به وجود آورنده‌ی زبان برنامه‌نویسی پایتون می‌باشد. (http://en.wikipedia.org/wiki/Guido_van_Rossum)

۲ – Python

فهرست برخی از عنوان‌های نوشته‌ها

اندیس‌های مثبت و منفی

برگرداندن کپی یک زیر مجموعه

کپی گرفتن از کلّ لیست

تاپل‌ها تغییرناپذیرند

لیست‌ها تغییرپذیرند

کارهایی که فقط بر روی لیست‌ها قابل انجام هستند

مقایسه‌ی تاپل‌ها و لیست‌ها

تبدیل لیست به تاپل و تبدیل تاپل به لیست

فهمیدن معناهای آدرس

تابع‌ها در پایتون

نوع‌ها در پایتون

صدازدن(فراخوانی) یک تابع

بهترین شیءهای پایتون تابع‌ها هستند

عبارت‌های منطقی

عملگرهای مقایسه‌ای

عبارت‌های منطقی بولی

ویژگی‌های ویژه‌ی and و or

عبارت‌های شرطی

کنترل جریان [برنامه]

عبارت‌های شرطی

حلقه‌های while

کلمه‌های کلیدی break و continue

عبارت‌های [برّرسی کننده‌ی] وضعیّت(assert)

ساخت لیست‌های جدید

حلقه‌های for

برخی موردهای جالب در مورد تابع‌ها

تعیین مقدارهای پیش‌ْفرض برای آرگومان‌ها

ترتیب آرگومان‌ها

انتساب چندگانه (یادآوری)

ظرف(متغیّر)های خالی

عملیّات بر روی رشته‌ها

تبدیل رشته‌ها

وارد کردن و نمونه(ماژول)ها

توجّه:

این فصل براساس Python نسخه‌ی 3.2.3 در ویندوز ۷(سِوِن) نوشته شده است.


جوانی گوییدُ وَن راسُم

پایتون زبانی است که شما می‌توانید به کد آن دست پیدا نمایید[۱] و آن را ویرایش نمایید. در اواخر دهه‌ی ۱۹۹۰ میلادی به وسیله‌ی گوییدُ وَن راسُم به وجود آمد. این زبان، خلاصه نویس‌تر از زبان برنامه‌نویسی جاوا می‌باشد. دارای انواع داده‌ای[۲] برای رشته‌ها، لیست‌ها و موردهای دیگر می‌باشد. دارای توانایی زیاد پردازش روی اعداد است و برای احتمال و برنامه‌نویسی یادگیری ماشینی مناسب است. به طور گسترده‌ای برای پردازش زبان طبیعی مورد استفاده قرار می‌گیرد و در ابتدا هم برای پردازش زبان طبیعی به وجود آمد. پایتون را می‌توانید از آدرس‌اینترنتی http://www.python.org دانلود نمایید.

این زبان به افتخار گروه کُمِدی انگلیسی «مُنتی پایتون[۳]»، پایتون نام گرفته است:

۱ - ویژگی open source
۲ - datatypes
۳ - Monty Python

شکل بالا– تصویری از گروه کمدی پایتون در سال ۱۹۶۹ میلادی[۱]

آرم این زبان هم به صورت زیر است:

مفسّر[۲] پایتون

پایتون دارای مفسّر است؛ یعنی، برنامه‌ها را خط به خط مورد برّرسی قرار می‌دهد و اجرا می‌کند؛ در ضمن، اعلان آن هم به صورت «>>>» می‌باشد؛ به عنوان مثال، اگر در محیط پایتون عبارت زیر را بنویسید و سپس دکمه‌ی اینتِر روی صفحه کلید را فشار دهید، خروجی ۲۷ را خواهید داشت:

```
>>> 3*(7+2)
```

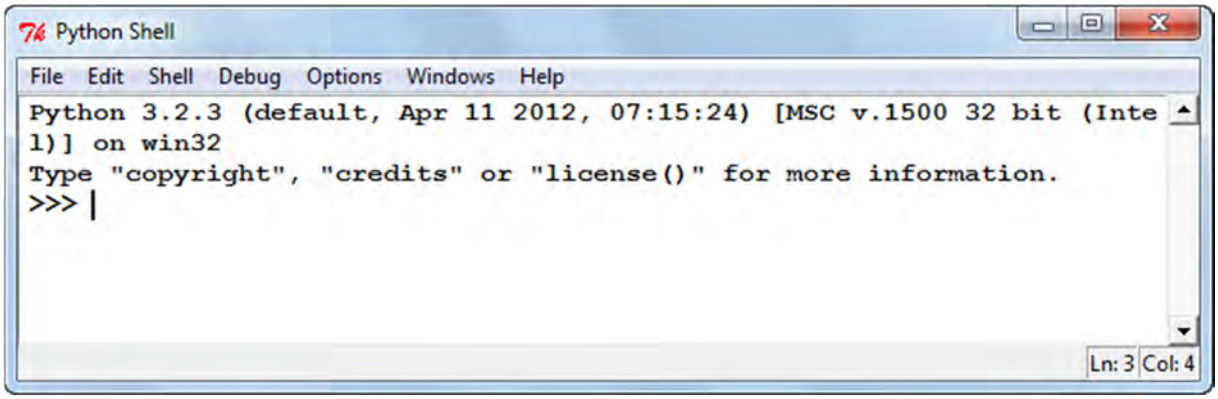
---

۱ – http://en.wikipedia.org/wiki/Monty_Python

۲ – interpreter

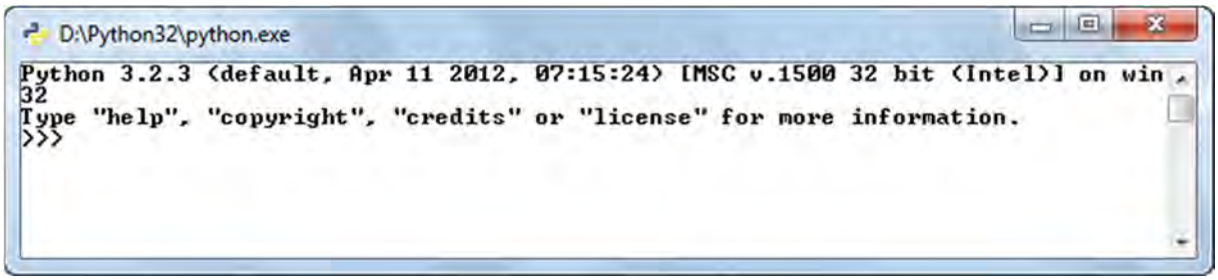
محیط پایتون

دارای محیط گرافیکی IDLE است که در زیر تصویری از آن را مشاهده می‌نمایید:



در این محیط که یک ویرایشگر متنی[۱] با کدهای رنگی[۲] و دارای تورفتگی‌های هوشمند[۳] است، می‌توانید فایل‌های پایتون را به وجود آورید. همچنین می‌توانید از دستورهای موجود در منوها برای تغییر تنظیم‌های سیستم و اجرای فایل‌ها استفاده نمایید.

در ضمن، پایتون دارای محیط خطّ فرمان[۴] هم می‌باشد، که در زیر تصویری از آن را مشاهده می‌نمایید:



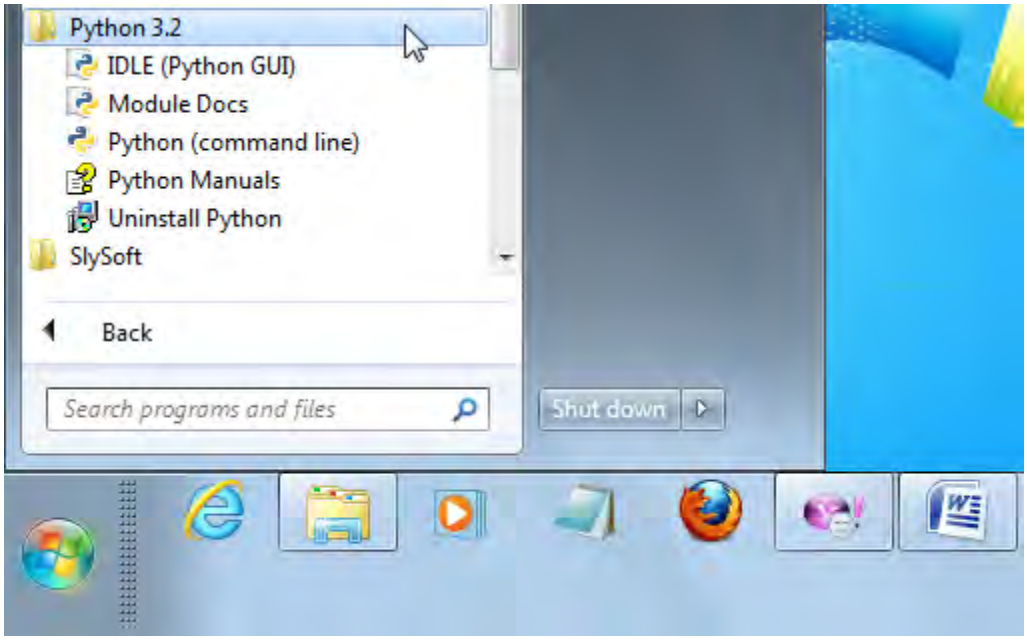
نکته:

برای تغییر رنگ محیط خطّ فرمان، بر روی نوار عنوان پنجره‌ی نشان داده شده در شکل بالا کلیک راست کرده و گزینه‌ی Defaults را از منوی ظاهر شده انتخاب کنید و سپس از سربرگ[۵] colors استفاده کنید. توجّه کنید که تنظیم‌های رنگی که انجام می‌دهید، بار بعدی‌ای که محیط خطّ فرمان را اجرا می‌کنید، دیده می‌شوند.

۱- text editor
۲- color-coding
۳- smart indenting
۴- command line
۵- tab

نحوه‌ی اجرا و خروج از پایتون در سیستم عامل ویندوز ۷

برای اجرای پایتون در ویندوز، از مسیر «Start > All Programs > Python x.y» استفاده نمایید؛ سپس در صورتی که می‌خواهید از محیط گرافیکی IDLE استفاده کنید، «IDLE (Python GUI)» را انتخاب کنید و اگر می‌خواهید از محیط خطّ فرمان استفاده کنید، «Python (Command line)» را برگزینید:



برای خروج از پایتون در حالت غیر IDLE، ابتدا کلیدهای «CONTROL+Z» را فشرده و سپس کلید اینتِر را فشار دهید.

نکته:

فایل‌های پایتون معمولاً به صورت «py.*» می‌باشند؛ به عبارت دیگر، پسوند فایل‌های پایتون معمولاً py است.

مبانی پایتون


پایتون به بزرگ و کوچک بودن حروف، حسّاس می‌باشد[۱]. برای توضیح[۲]نویسی از کاراکتر نامبِر(#) استفاده می‌کنیم.

مثال– اجرای یک برنامه‌ی نمونه

فرض کنید می‌خواهیم برنامه‌ی زیر را اجرا کنیم:

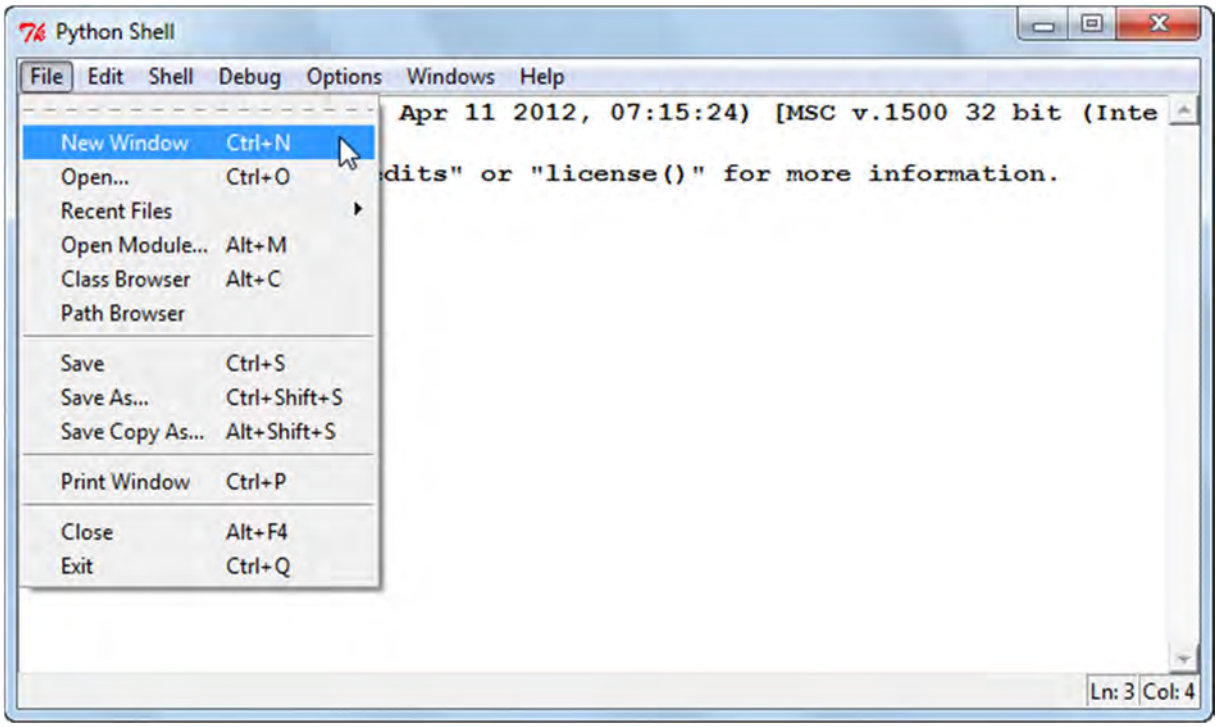
```
x = 34 - 23          # A comment.
y = 'Hello'          # Another one.
z = 3.45
if z == 3.45 or y == 'Hello':
    x = x + 1
    y = y + ' World'   # String concat.
print(x)
print(y)
```


ابتدا در محیط IDLE پایتون، از منوی File، گزینه‌ی New window را انتخاب کنیم:

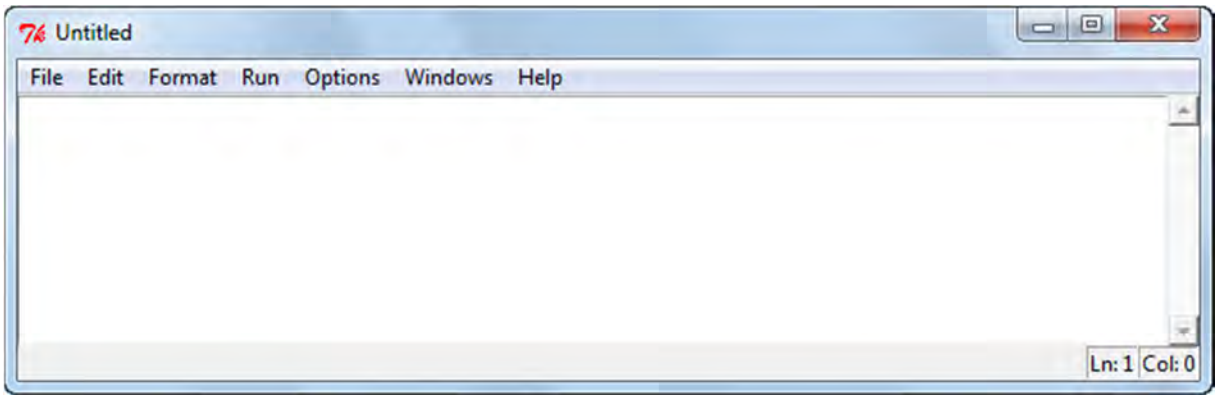
---

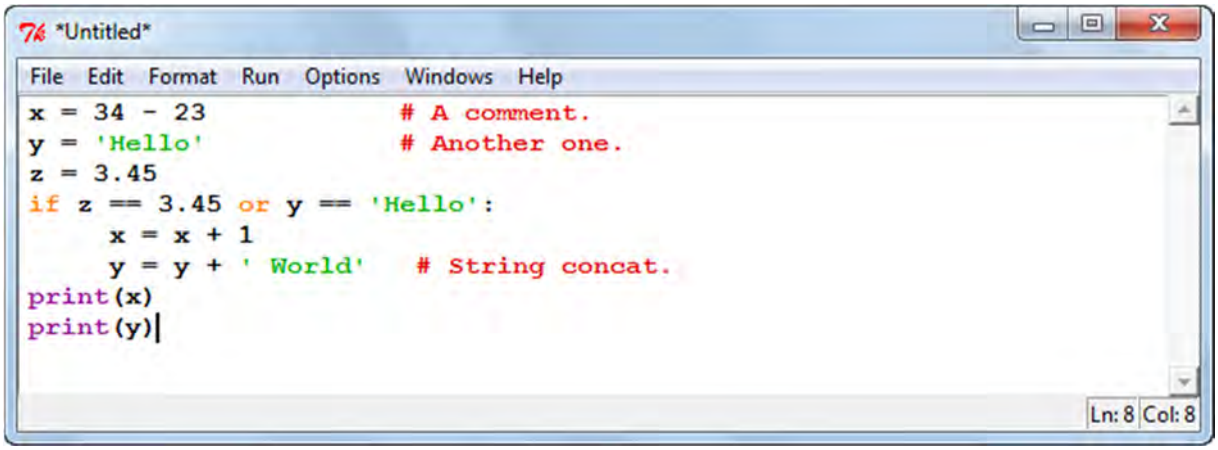
۱ – case sensitive است.

۲ – comment(remark)

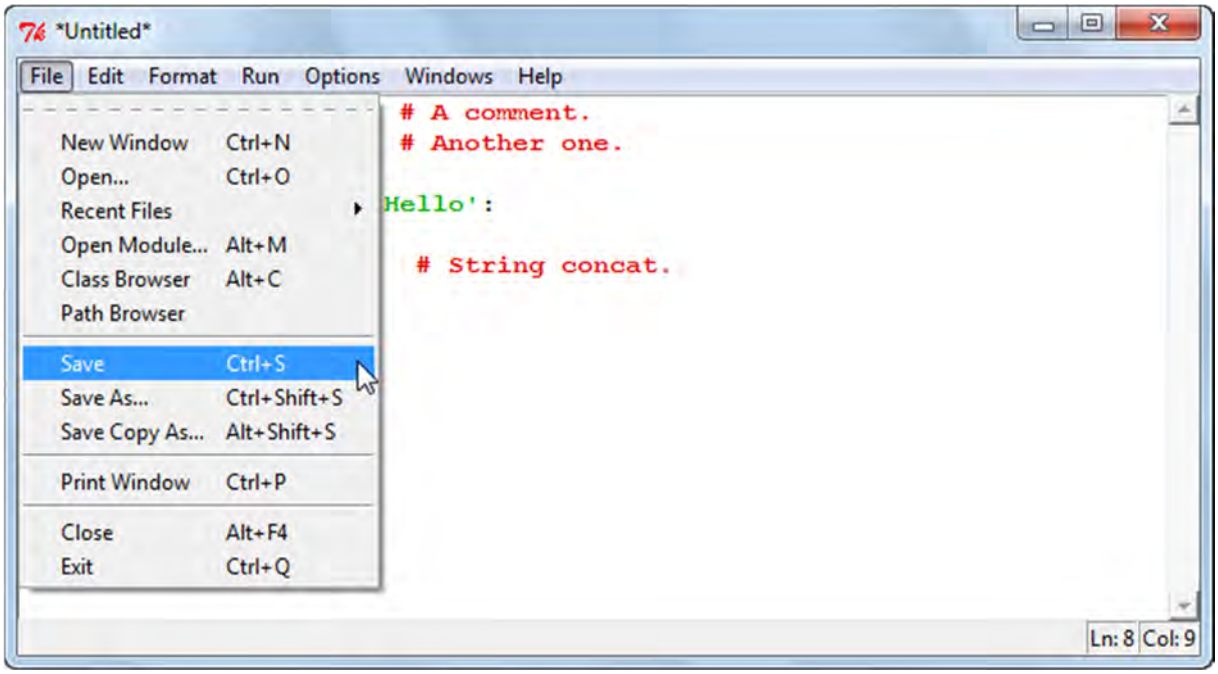


در پنجره‌ای که به وجود می‌آید، شروع به تایپ کد مورد نظر خود می‌نماییم:

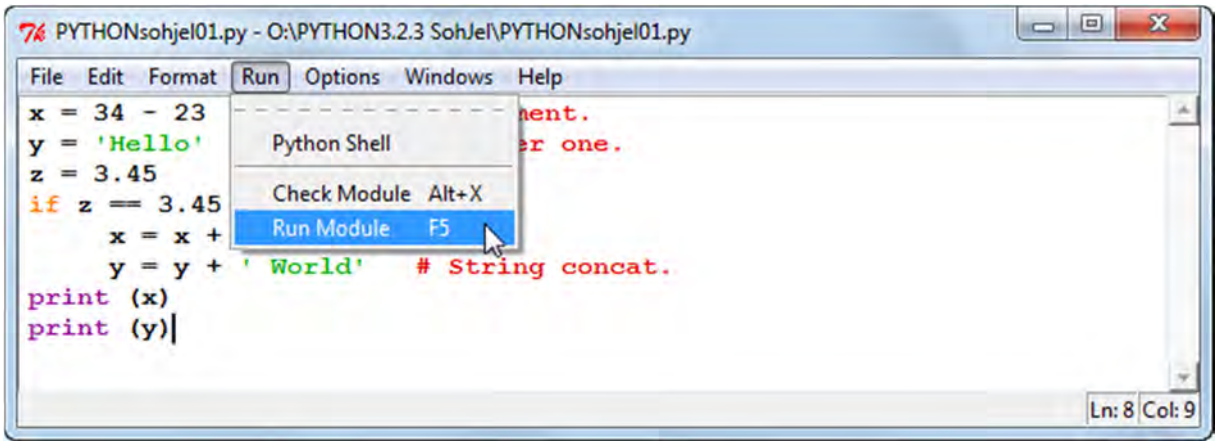




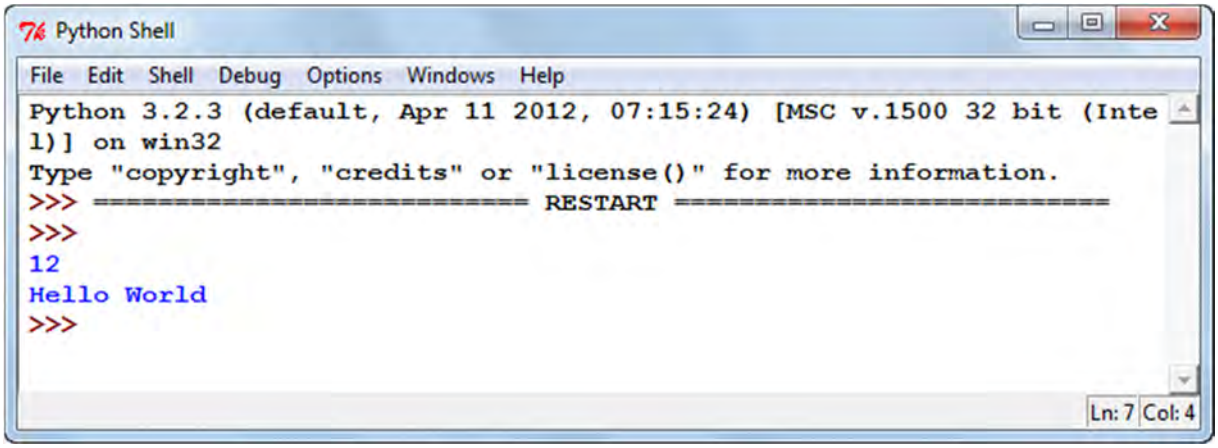
سپس برای ذخیره‌ی کدی که نوشتیم، از منوی File، گزینه‌ی Save را انتخاب می‌نماییم:



بعد از ذخیره کردن فایل، برای اجرا، از منوی Run، گزینه‌ی Run module را انتخاب می‌کنیم و یا از کلید F5 استفاده می‌نماییم:



بعد از این کار، محیط IDLE (Python shell) باز می‌شود و نتیجه‌ی اجرای برنامه‌ای که نوشته بودیم را می‌توانیم در آن ببینیم:



توضیح‌هایی در مورد کد مثال قبل

همان طور که در کد بالا می‌بینید، از علامت مساوی(=) برای انتساب استفاده می‌کنیم؛ برای برّرسی مساوی بودن از دو مساوی پشت سر هم(==) استفاده می‌کنیم. برای اعداد، از + (برای انجام عمل جمع)، - (برای انجام عمل تفریق)، * (برای ضرب)، / (برای تقسیم) و از ٪ (برای به‌دست‌آوردن باقیمانده‌ی تقسیم) استفاده می‌کنیم.

مثال‌ها:

```
>>> 20+3
23

>>> 20-3
17

>>> 20*3
60

>>> 20/3
6.666666666666667

>>> 20%3
2
```

نکته:

از عملگر + برای چسباندن رشته‌ها به هم نیز استفاده می‌نماییم؛ به عنوان مثال:

```
>>> 'Soh'+'Jel'+'@yahoo.com'
'SohJel@yahoo.com'
```

عملگرهای منطقی استفاده شده در این مثال، and، or و not هستند. دستور اصلی‌ای که از آن برای چاپ استفاده می‌کنیم، تابع print است. *در پایتون لازم نیست که نوع‌های داده‌ای را مشخّص نماییم، چرا که پایتون خودش نوع‌های داده‌ای را تشخیص می‌دهد.*

انواع داده‌ای اساسی

اعداد صحیح، که به صورت پیش‌فرض برای عددها در نظر گرفته می‌شوند؛ مثلاً z=5+2 که جواب آن ۷ است.

اعداد اعشاری، مثل: x=3.456

رشته‌ها، برای مقدارهای رشته‌ای می‌توانیم از کوتیشنِ تکی(' ') یا کوتیشن دوتایی(" ") استفاده کنیم.

مثال‌ها:

```
>>> "Soh"+"Jel"
'SohJel'

>>> 'Soh'+'Jel'
'SohJel'

>>> "Soh"+'Jel'
'SohJel'
```

مثال:

```
>>> "Soh "Jel"
SyntaxError: invalid syntax
```

برای اینکه با این خطا مواجه نشویم، می‌توانیم از سه کوتیشن دوتایی پشت سرهم(""" """) استفاده کنیم:

```
>>> """Soh "Jel"""
'Soh "Jel'
```

فضای خالی[۱]

فضای خالی در پایتون معنادار است، مخصوصاً تورفتگی و قرار دادن خط‌های جدید. برای مشخّص کردن انتهای یک خط از برنامه، به خطّ بعدی بروید. زمانی که می‌خواهید ادامه‌ی یک خط از کد را در خطّ بعدی بنویسید از بَکْ اِسْلَشْ(\) استفاده نمایید.

مثال:

```
>>> "Soh" \
    +"Jel"
'SohJel'
```

در زبان‌هایی مانند سی و جاوا برای مشخّص کردن بلوک‌های کد از بِراکِت({ }) استفاده می‌کنیم، ولی در پایتون برای این منظور از براکت استفاده نمی‌کنیم. *اوّلین خط با تورفتگی کم‌تر، خارج از بلوک است. اوّلین خط با تورفتگی بیش‌تر، یک بلوک تودرتو[۲] را به وجود می‌آورد. اغلب دو نقطه‌ی بیانی(:)[۳] در شروع یک بلوک جدید ظاهر می‌شود؛* به عنوان مثال، برای تعریف تابع و کلاس.

مثال- خروجی برنامه‌ی زیر Not SohJel است؛ توجّه کنید که تابع print اوّل، در دستور if است و تابع print دوّم، خارج از if می‌باشد و به دلیل برقرار نبودن شرط، تابع print دوّم اجرا می‌شود.

```
t="Soh"
u="123"
if u=="Jel":
    print("SohJel")
print("Not SohJel")
```

۱- Whitespace

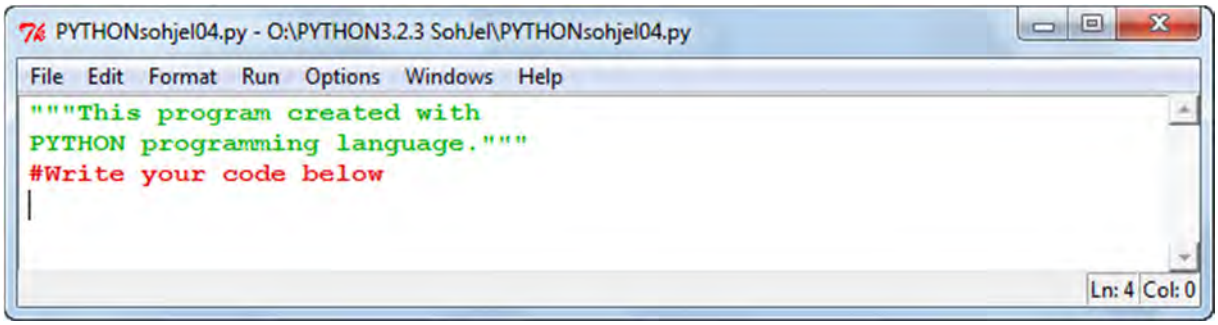
۲- nested block؛ **یادآوری**- بلوکی که در بلوک دیگر برنامه قرار دارد.

۳- colon

اضافه نمودن رشته‌ی مستندات[1] و توضیح

در پایتون برای اضافه نمودن رشته‌ی مستندات می‌توانید از سه کوتیشن دوتایی(""" """) استفاده نمایید و برای اضافه کردن توضیح می‌توانید از کاراکتر نامبر(#) استفاده کنید.

مثال:



دستیابی به نام‌هایی که وجود ندارند

اگر شما بخواهید برای متغیّرها از نامی استفاده نمایید که در گذشته با استفاده از مقداردهی آن را به وجود نیاورده‌اید، با خطا مواجه خواهید شد.

مثال– در مثال زیر چونکه در گذشته y را با استفاده از مقداردهی به وجود نیاورده‌ایم، با خطا مواجه می‌شویم:

```
>>> y
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    y
NameError: name 'y' is not defined
```

ولی اگر قبل از بیان y آن را مقداردهی کرده باشیم، با خطا مواجه نخواهیم شد:

```
>>> y=1000
>>> y
```

---

۱ – documentation string؛ **تعریف**– روشی دیگر برای توضیحْ نوشتن در پایتون است و رشته‌ای است که به تنهایی و بدون انتساب، در اوّلین خطّ یک نمونه(ماژول(module))، کلاس، مِتُد و یا تابع قرار می‌گیرد.
(http://en.wikipedia.org/wiki/Python_syntax_and_semantics)

1000

انتساب چندگانه

شما می‌توانید در یک زمان چند نام را مقداردهی نمایید.

مثال:

```
>>> x,y=14,"SJ"
>>> x
14
>>> y
'SJ'
```

قانون‌های نامگذاری متغیّرها

نام‌ها نسبت به حروف بزرگ و کوچک حسّاس هستند؛ به عنوان مثال، نام SJ (با حرف‌های بزرگ) با نام sj (با حرف‌های کوچک) فرق دارد. نام متغیّر می‌تواند شامل حرف‌ها، عددها و کاراکتر زیر خط(_) باشد؛ نام متغیّر نمی‌تواند با عدد شروع شود؛ همچنین استفاده از کاراکتر فضای خالی[۱] در بین حروف نام متغیّر مُجاز نمی‌باشد.

مثال- نام‌های

t2، t_11، xyz و _2W

مجاز هستند؛ ولی نام‌های

2t، 11_t، 123، 1_2 و t 20

غیرمجاز می‌باشند.

در ضمن، از *کلمه‌های ذخیره شده(رزِروْ شده)*[۲] هم نمی‌توانیم استفاده کنیم، در زیر برخی از این کلمه‌ها آورده شده است:

False, class, finally, is, return, None, continue, for, lambda, try, True, def, from, nonlocal, while, and, del, global, not, with, as, elif, if, or, yield, assert, else, import, pass, break, except, in, raise

---

۱ - space یا blank، که با استفاده از دکمه‌ی Spacebar روی صفحه‌کلید نوشته می‌شود.

۲ - reserved words؛ **یادآوری**- در کامپیوتر، کلمه‌ای است که در یک زبان برنامه‌نویسی دارای معنی ثابتی است و نمی‌تواند به وسیله‌ی برنامه‌نویس دوباره تعریف شود. (Babylon> Concise Oxford English Dictionary)

تاپِل‌ها[1]، لیست‌ها و رشته‌ها

تاپل‌ها

رشته‌ای ساده از عنصرهای *تغییرناپذیر*[2] هستند و عنصرها می‌توانند دارای نوع‌های مختلف باشند. تاپل‌ها با استفاده از پرانتز تعریف می‌شوند و بین هر عنصر، یک کاما(,) قرار می‌گیرد.

مثال:

```
>>> tu = (12345, 54321, 'hello!')
```

رشته‌ها

تغییرناپذیر هستند و خیلی شبیه به تاپل‌ها می‌باشند. رشته‌ها با استفاده از کوتِیشِن(' ') یا کوتیشن دوتایی(" ") و یا سه کوتیشن دوتایی پشت سرهم(""" """) تعریف می‌شوند.

نکته:

اگر از سه کوتیشن دوتایی پشت سرهم استفاده کنیم، می‌توانیم رشته‌های چندخطّی داشته باشیم.

مثال‌ها:

```
>>> SJ='Soh Jel'

>>> SJ="Soh Jel"

>>> SJ="""Soh
Jel"""
```

۱- tuples

۲- immutable

لیست‌ها

تغییرپذیر[1] هستند و شامل نوع‌های مختلف می‌باشند و با استفاده از کِروشه([ ]) تعریف می‌شوند و بین هر عنصر از کاما استفاده می‌نماییم.

مثال:

```
>>> z=[3,"xyz",14,"12"]
>>> z
[3, 'xyz', 14, '12']
```

تمام سه نوع تاپل، لیست و رشته از لحاظ ظاهر و کارکرد شبیه هم هستند؛ فرق اصلی آنها در این است که *تاپل‌ها و رشته‌ها تغییرناپذیر هستند؛ ولی لیست‌ها تغییرپذیر می‌باشند.*

مطلب مهم:

می‌توانیم به هر عنصر تاپل، لیست و رشته با استفاده از *آرایه* دسترسی پیدا کنیم؛ *توجّه کنید که اندیس آرایه از عدد صفر شروع می‌شود.*

مثال– در تاپل زیر عنصر دوّم، 'abc' است که با استفاده از دستور tu[1] می‌توان به آن دست یافت.

```
>>> tu = (12345, 54321, 'hello!')
>>> tu[1]    # Second item in the tuple.
54321
```

مثال– در لیست زیر عنصر سوّم را مورد دسترسی قرار داده‌ایم:

```
>>> a=[3,"3.14",'12',8]
>>> a[2]
'12'
```

مثال– در زیر اوّلین عنصر رشته را به دست آورده‌ایم:

```
>>> d="SohJel"
>>> d[0]
'S'
```

---

۱ – mutable

مثال- چنانچه گفتیم، لیست‌ها *تغییرپذیر هستند*؛ در مثال زیر در ابتدا چون خواسته‌ایم عنصری خارج از محدوده را تغییر دهیم، با خطا مواجه شده‌ایم؛ ولی بار دوّم، عنصر سوّم لیست را از h به j تغییر داده‌ایم:

```
>>> t_1=['s','o','h']
>>> t_1[3]="jel"
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    t_1[3]="jel"
IndexError: list assignment index out of range
>>> t_1[2]='j'
>>> t_1
['s', 'o', 'j']
```

مثال- همان طور که گفتیم، تاپل‌ها و رشته‌ها *تغییر ناپذیر هستند*؛ در زیر به دلیل اینکه خواسته‌ایم عنصر چهارم رشته را تغییر دهیم با خطا مواجه شده‌ایم.

```
>>> S_J_="Sohjel"
>>> S_J_[3]="J"
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    S_J_[3]="J"
TypeError: 'str' object does not support item assignment
```

اندیس‌های مثبت و منفی

در اندیس مثبت، شمارش از سمت چپ شروع می‌شود و شروع اندیس از عدد صفر است، ولی در اندیس منفی، شمارش از سمت راست است و اندیس از عدد منفی یک شروع می‌شود.

مثال:

```
>>> _230="PYTHON"
>>> _230[2]
'T'
>>> _230[-2]
'O'
```

برگرداندن کپی یک زیر مجموعه

مثال– در تاپل زیر می‌خواهیم از عنصر دوّم تا چهارم را برگردانیم، داریم:

```
>>> sj=('s','o','h',20)
>>> sj[1:3]
('o', 'h')
```

توجّه کنید که از اندیس منفی هم می‌توان استفاده کرد:

```
>>> sj[1:-1]
('o', 'h')
```

برای اینکه بُرِش را از اوّل شروع کنیم، می‌توانیم اندیس اوّل را ذکر نکنیم، همچنین برای اینکه برش را تا آخر ادامه دهیم، می‌توانیم اندیس آخر را ذکر ننماییم.

مثال‌ها:

```
>>> Soh=[1,2,3,4,5]
>>> Soh[:3]
[1, 2, 3]
>>> Soh[1:]
[2, 3, 4, 5]
>>> Soh[:]
[1, 2, 3, 4, 5]
```

کپی گرفتن از کلّ لیست

می‌توانیم از کلّ یک لیست کپی بگیریم و در یک لیست جدید قرار دهیم، به مثال زیر توجّه کنید:

```
>>> s1=[1,2,3,4]
>>> s2=s1
>>> s2
[1, 2, 3, 4]
>>> s2[1]=5
>>> s2
[1, 5, 3, 4]
```

```
>>> s1
[1, 5, 3, 4]
```

مطلب مهم:

در مثال بالا اگر s1 را تغییر دهیم، s2 هم تغییر خواهد کرد و اگر s2 را تغییر دهیم، s1 هم تغییر می‌کند، به عبارت دیگر، در این حالت دو کپی با یک آدرس به وجود می‌آیند و اگر یکی تغییر نماید، دیگری هم تغییر پیدا خواهد کرد. ولی در مثال زیر اگر j1 را تغییر دهیم، j2تغییر نخواهد کرد، همچنین اگر j2 را تغییر دهیم، j1 تغییر نمی‌کند، به عبارت دیگر، در این حالت دو کپی با دو آدرس متفاوت به وجود می‌آیند و اگر یکی تغییر کند، دیگری تغییری نمی‌کند:

```
>>> j1=[5,4,3,2,1,0]
>>> j2=j1[:]
>>> j1
[5, 4, 3, 2, 1, 0]
>>> j2
[5, 4, 3, 2, 1, 0]
>>> j2[-1]=6
>>> j2
[5, 4, 3, 2, 1, 6]
>>> j1
[5, 4, 3, 2, 1, 0]
```

عملگر in

برای برّرسی اینکه آیا یک مقدار، درون یک ظرف(متغیّر) وجود دارد یا نه از این عملگر استفاده می‌نماییم؛ اگر مقدار مورد نظر، در ظرف وجود داشت، نتیجه True خواهد بود و اگر وجود نداشته باشد، نتیجه False خواهد بود.

مثال:

```
>>> F=(10,100,1000,10000)
>>> 10 in f
Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
    10 in f
NameError: name 'f' is not defined
```

همان طور که گفتیم، پایتون بین حروف بزرگ و کوچک فرق می‌گذارد، به همین دلیل در بالا چون از حرف کوچک f استفاده کرده‌ایم، از ما خطا گرفته است.

```
>>> 10 in F
```

True

>>> -10 in F

False

این عملگر در مورد رشته‌ها برّرسی می‌کند که آیا زیررشته‌ی ذکر شده وجود دارد یا نه؟.

مثال:

```
>>> test="This is a test."
>>> H in test
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    H in test
NameError: name 'H' is not defined
```

در بالا چونکه H بدون کوتیشن یا کوتیشن دوتایی بیان شده است، در نتیجه مقدار رشته‌ای نمی‌باشد - نام متغیّری است که هنوز تعریف نکرده‌ایم- و با خطا روبه‌رو شده‌ایم:

```
>>> 'H' in test
False
>>> 'h' in test
True
>>> 'SohJel' not in test
True
>>> "test" not in test
False
```

نکته:

in جزو کلمه‌های کلیدی پایتون است و در حلقه‌های for و لیست‌ها هم مورد استفاده دارد.

عملگر +

از این عملگر برای جمع عددها؛ کنار هم قرار دادن رشته‌ها؛ تاپل‌ها؛ و لیست‌ها استفاده می‌شود.

مثال‌ها:

```
>>> 1+5+3+5
14
```

```
>>> "SJ likes"+" "+"PYTHON programming language!"
'SJ likes PYTHON programming language!'

>>> s=(1,2,3)
>>> j=(3,2,1)
>>> s+j
(1, 2, 3, 3, 2, 1)

>>> u=["123",1,"22"]
>>> v=["abc",32]
>>> v+u
['abc', 32, '123', 1, '22']

>>> s+v
Traceback (most recent call last):
  File "<pyshell#56>", line 1, in <module>
    s+v
TypeError: can only concatenate tuple (not "list") to tuple
```

در بالا چون خواسته‌ایم دو نوع داده‌ای متفاوت تاپل و لیست را در کنار هم قرار دهیم، با خطا مواجه شده‌ایم.

عملگر ❋

این عملگر، تاپل، لیست یا رشته‌ای جدید را با تکرار محتوای اوّلیّه به وجود می‌آورد؛ در ضمن، برای ضرب اعداد هم از این عملگر استفاده می‌نماییم.

مثال‌ها:

```
>>> "SohJel"*5
'SohJelSohJelSohJelSohJelSohJel'
>>> ('S','o','h','Jel')*3
('S', 'o', 'h', 'Jel', 'S', 'o', 'h', 'Jel', 'S', 'o', 'h', 'Jel')
>>> [1,2,3]*4
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

تاپل‌ها تغییرناپذیرند

چنان که در گذشته هم دیدیم، نمی‌توان مقدارهای موجود در یک تاپل را تغییر داد، *ولی می‌توانیم یک تاپل جدید به وجود آوریم و به نام تاپل قبلی نسبت دهیم.*

مثال:

```
>>> f=(1,2,3)
>>> f[1]=12
Traceback (most recent call last):
  File "<pyshell#61>", line 1, in <module>
    f[1]=12
TypeError: 'tuple' object does not support item assignment
>>> f=(1,12,3)
```

لیست‌ها تغییرپذیرند

همان طور که در گذشته هم گفتیم، می‌توان عناصر موجود در لیست را تغییر داد.

مثال:

```
>>> g=[1,2,3]
>>> g[2]=1
>>> g
[1, 2, 1]
```

کارهایی که فقط بر روی لیست‌ها قابل انجام هستند

مِتُدِ[1] append( )

با این دستور می‌توان *فقط* یک عنصر را به انتهای یک لیست اضافه نمود.

مثال:

```
>>> SJli01=[1,2,3]
```

---

۱ - method، **یادآوری**- زیربرنامه‌ای مشخّص برای اجرا یا دستیابی به چیزی است. (Babylon> Concise Oxford English Dictionary)

```
>>> SJli01.append('j','k')
Traceback (most recent call last):
  File "<pyshell#67>", line 1, in <module>
    SJli01.append('j','k')
TypeError: append() takes exactly one argument (2 given)
```

همان طور که گفتیم، با دستور append( ) فقط می‌توان یک عنصر را به لیست اضافه کرد؛ در بالا چون خواسته‌ایم با این دستور، دو عنصر را به لیست اضافه کنیم، با خطا مواجه شده‌ایم.

```
>>> SJli01.append('j')
>>> SJli01
[1, 2, 3, 'j']
```

متد insert[1](x,y)

این دستور عنصر y را در محلّ x لیست درج می‌کند.

مثال:

```
>>> SJli02=['a','b','c','t']
>>> SJli02.insert(2,'SSJJ')
>>> SJli02
['a', 'b', 'SSJJ', 'c', 't']
```

تفاوت متد extend[2]( )، با عملگر +

عملگر + یک لیست جدید را با آدرس جدید به وجود می‌آورد؛ ولی متد extend بر روی خود لیست و در همان آدرس قبلی لیست کار می‌کند.

مثال:

```
>>> SJli03=[3,2,1]
>>> SJli03.extend('1','2','3')
Traceback (most recent call last):
  File "<pyshell#74>", line 1, in <module>
    SJli03.extend('1','2','3')
TypeError: extend() takes exactly one argument (3 given)
```

---

۱ – در لغت یعنی: «درج کردن»

۲ – در لغت یعنی: «توسعه‌دادن»

در بالا چون خواسته‌ایم سه عنصر را به لیست اضافه کنیم با خطا مواجه شده‌ایم؛ با متد extend فقط می‌توان یک عنصر و یا یک لیست را به لیست اصلی اضافه کرد.

```
>>> SJli03.extend(['1','2','3'])
>>> SJli03
[3, 2, 1, '1', '2', '3']
>>> SJli03.extend('w')
>>> SJli03
[3, 2, 1, '1', '2', '3', 'w']
```

نکته‌ی گیج کننده:

اگر با استفاده از متد append بخواهیم یک *لیست* را به لیست اصلی اضافه کنیم، خود *لیست*، به لیست اصلی اضافه می‌شود؛ ولی اگر در این مورد از متد extend استفاده کنیم، *عنصرهای موجود در لیست*، به لیست اصلی اضافه خواهند شد. برای روشن شدن مطلب به مثال زیر توجّه نمایید.

مثال:

```
>>> SJli04=[1,2]
>>> SJli04.append([3,4])
>>> SJli04
[1, 2, [3, 4]]
>>> SJli04.extend([5,6])
>>> SJli04
[1, 2, [3, 4], 5, 6]
```

متد index[1](x)

مکان اوّلین رویداد x ذکر شده را برمی‌گرداند.

مثال‌ها:

```
>>> SJli05=['S','o','h']
>>> SJli05.index('s')
Traceback (most recent call last):
  File "<pyshell#85>", line 1, in <module>
    SJli05.index('s')
```

---

۱ – در اینجا یعنی: «نشان دادن»

ValueError: 's' is not in list

در بالا چون s کوچک در لیست وجود ندارد، با خطا برخورد کرده‌ایم.

```
>>> SJli05.index('S')
0
>>> SJli05.index("h")
2
```

متد count[1](x)

این متد تعداد رویداد x بیان شده در لیست را به ما می‌دهد.

مثال‌ها:

```
>>> SJli06=[1,2,2,3,3,3,4,4,4,4]
>>> SJli06.count(2)
2
>>> SJli06.count('2')
0
>>> SJli06.count(4)
4
>>> SJli06.count(12)
0
```

متد remove[2](x)

این متد *اوّلین* رویداد x ذکر شده را از لیست حذف می‌نماید.

مثال‌ها:

```
>>> SJli07=['s1','s2','s3','s4']
>>> SJli07.remove('s2')
>>> SJli07
['s1', 's3', 's4']
```

---

۱ - در لغت یعنی: «شمردن»

۲ - در لغت یعنی: «حذف کردن»

```
>>> SJli08=['A','B','C','A','B']
>>> SJli08.remove("B")
>>> SJli08
['A', 'C', 'A', 'B']
```

متد reverse( )

عناصر لیست را معکوس می‌نماید.

مثال‌ها:

```
>>> SJli09=[1,2,3,4,5]
>>> SJli09.reverse( )
>>> SJli09
[5, 4, 3, 2, 1]

>>> SJli10=["Ali","Ahmad",'Javad']
>>> SJli10.reverse
<built-in method reverse of list object at 0x01219EB8>
```

توجّه کنید که باید پرانتزهای متدها را بگذارید حتّا اگر آرگومانی نداشته باشند.

```
>>> SJli10.reverse( )
>>> SJli10
['Javad', 'Ahmad', 'Ali']
```

متد sort( )

عنصرهای یک لیست را مرتّب می کند.

مثال‌ها:

```
>>> SJli11=[5,4,12,3,11,20,1]
>>> SJli11.sort( )
>>> SJli11
[1, 3, 4, 5, 11, 12, 20]

>>> SJli12=['hasan','javad','majid','ebrahim']
>>> SJli12.sort( )
```

```
>>> SJli12
['ebrahim', 'hasan', 'javad', 'majid']
```

مقایسه‌ی تاپل‌ها و لیست‌ها

لیست‌ها نسبت به تاپل‌ها کندتر هستند، ولی دارای توانایی‌های بیش‌تری هستند؛ لیست‌ها می‌توانند تغییر، داده شوند و ابزارهای آماده‌ی زیادی دارند که می‌توانیم به وسیله‌ی آنها لیست‌ها را تغییر دهیم. تاپل‌ها تغییرناپذیر هستند و دارای ابزارهای کم‌تری می‌باشند.

تبدیل لیست به تاپل و تبدیل تاپل به لیست

برای تبدیل لیست به تاپل از tuple(LIST) استفاده می‌کنیم و برای تبدیل تاپل به لیست از list(TUPLE) استفاده می‌کنیم.

مثال‌ها:

```
>>> SJli14=[1,2,3,4]
>>> SJtu01=tuple(SJli14)
>>> SJtu01
(1, 2, 3, 4)

>>> SJli15=list(SJtu01)
>>> SJli15
[1, 2, 3, 4]
```

فهمیدن معناهای آدرس

در پایتون انتساب برای انواع داده‌ای ساده مثل عددهای صحیح، اعشاری و رشته‌ها درست کار می‌کند؛ یعنی اینکه اگر در آینده یکی را تغییر دهیم، دیگری تغییر نخواهد کرد.

مثال:

```
>>> SJ01=11
>>> SJ02=SJ01
>>> SJ02=12
>>> SJ01
11
```

ولی برای نوع‌های دیگر داده‌ای، مثل لیست‌ها، دیکشنری(فرهنگ)ها[1] و انواعی که کاربر تعریف می‌کند، انتساب به این صورت کار نمی‌کند؛ یعنی، اگر در آینده یک تغییری در یکی بدهیم، دیگری هم تغییر خواهد کرد.

مثال:

```
>>> SJli18=[1,2,3]
>>> SJli19=SJli18
>>> SJli19.append(4)
>>> SJli19
[1, 2, 3, 4]
>>> SJli18
[1, 2, 3, 4]
```

نکته‌ی مهم:

اگر انتساب به صورت زیر باشد، درست کار می‌کند.

```
>>> SJli16=[1,2,3]
>>> SJli17=SJli16
>>> SJli17=[4,5,6,7]
>>> SJli16
[1, 2, 3]
```

۱ - dictionaries؛ در ادامه‌ی همین فصل با این نوع داده‌ای آشنا می‌شویم.

دیکشِنِری(فرهنگ)ها: یک نوع نِگاشت کننده

دیکشنری‌ها، یک نگاشت[۱] را میان یک مجموعه از کلید[۲]ها و یک مجموعه از مقدار[۳]ها نگهداری می‌کنند؛ کلیدها می‌توانند هر نوع تغییرناپذیر باشند؛ مقدارها می‌توانند از هر نوعی باشند؛ یک دیکشنری تکی می‌تواند مقدارهایی از نوع‌های مختلف را نگهداری نماید. می‌توانید زوج‌های کلید–مقدار را تعریف، ویرایش، دیدن، جستجو و حذف نمایید.

ساخت و دستیابی به دیکشنری‌ها

```
>>> SJdic01={'Username':'SohJel','Password':12345}
>>> SJdic01['Username']
'SohJel'
>>> SJdic01['Password']
12345

>>> SJdic01['SohJel']
Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    SJdic01['SohJel']
KeyError: 'SohJel'
```

در مثال بالا چون برای دستیابی از کلید استفاده نکرده‌ایم و از مقدار استفاده کرده‌ایم، با خطا مواجه شده‌ایم.

به‌روز رسانی دیکشنری‌ها

```
>>> SJdic01['Username']='Sohrab'
>>> SJdic01
{'Username': 'Sohrab', 'Password': 12345}
```

---

۱ - mapping
۲ - key
۳ - value

کلیدها باید منحصربه‌فرد باشند. اگر به یک کلید که دارای مقدار است، مقداری بدهیم، مقدار جدید، جایگزین مقدار قبلی خواهد شد.

در زیر یک کلید و مقدار جدید را به دیکشنری مثال قبل اضافه نموده‌ایم:

```
>>> SJdic01['id']=654321
>>> SJdic01
{'Username': 'Sohrab', 'Password': 12345, 'id': 654321}
>>> SJdic01['tel']=54321
>>> SJdic01
{'Username': 'Sohrab', 'Password': 12345, 'tel': 54321, 'id': 654321}
```

دیکشنری‌ها به صورت نامرتّب هستند؛ فقره‌ی(قلم) جدید ممکن است در هر جایی در خروجی ظاهر شود.

حذف(برداشتن) فقره(قلم)های موجود در دیکشنری

برای برداشتن یکی از فقره‌های موجود در دیکشنری، از دستور del استفاده می‌کنیم و برای حذف کامل فقره‌های موجود در دیکشنری، از متد ( )clear استفاده می‌نماییم.

مثال:

```
>>> del SJdic01['tel']
>>> SJdic01
{'Username': 'Sohrab', 'Password': 12345, 'id': 654321}

>>> SJdic01.clear ( )
>>> SJdic01
{}
```

متدهای مفید دستیابی در دیکشنری‌ها

برای این قسمت، دیکشنری زیر را در نظر می‌گیریم:

```
>>> SJdic02={'Fname':''Soh'','Lname':'Jel','Tel':12345}
>>> SJdic02
{'Lname': 'Jel', 'Tel': 12345, 'Fname': 'Soh'}
```

متد ( )keys

این متد کلیدهای موجود در دیکشنری را به صورت یک *لیست* برمی‌گرداند.

مثال:

```
>>> SJdic02.keys( )
dict_keys(['Lname', 'Tel', 'Fname'])
```

متد ( )values

لیست مقدارها را به صورت یک *لیست* برمی‌گرداند.

مثال:

```
>>> SJdic02.values( )
dict_values(['Jel', 12345, 'Soh'])
```

متد ( )items

لیستی از عنصرهای موجود در دیکشنری را به صورت تاپل‌هایی برمی‌گرداند.

مثال:

```
>>> SJdic02.items( )
dict_items([('Lname', 'Jel'), ('Tel', 12345), ('Fname', 'Soh')])
```

تابع‌ها در پایتون

تعریف تابع

تعریف تابع با بیان کلمه‌ی کلیدی def آغاز می‌شود؛ سپس نام تابع ذکر می‌شود و سپس آرگومان‌ها در داخل پرانتز ذکر می‌شوند؛ سپس دو نقطه‌ی بیانی(:) می‌گذاریم؛ سپس در خطّ بعد، با یک تورفتگی - رعایت تورفتگی هم مهم است– شروع به

نوشتن کدهای برنامه می‌کنیم؛ برای برگرداندن مقدار به نقطه‌ای که فراخوانی از آنجا صورت گرفته است، اوّل از کلمه‌ی کلیدی return[1] استفاده می‌نماییم و سپس مقداری که می‌خواهیم برگردانده شود را ذکر می‌کنیم. برای تعریف تابع از ساختار زیر استفاده می‌نماییم:

```
def نام تابع(لیست آرگومان‌ها):
    «رشته‌ی مستندها»
    خطّ اوّل
    خطّ دوّم
    ...
    return مقداری که می‌خواهیم برگردانده شود
```

نوع‌ها در پایتون

تشخیص نوع پویا: همان‌طور که در گذشته هم گفتیم، پایتون نوع‌های داده‌ای را به صورت خودکار برای متغیّرهای موجود در برنامه تشخیص می‌دهد؛ *امّا چنانچه در گذشته هم دیدیم، این مطلب به معنی این نیست که پایتون در مورد نوع داده‌ها حسّاس نمی‌باشد؛* مثلاً شما نمی‌توانید یک نوع داده‌ای صحیح را با یک نوع رشته‌ای جمع نمایید.

مثال:

```
>>> "SJ"+12
Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    "SJ"+12
TypeError: Can't convert 'int' object to str implicitly
```

در اینجا(بالا) چون خواسته‌ایم یک نوع رشته‌ای را با یک نوع صحیح جمع کنیم، خطا دریافت کرده‌ایم.

صدازدن(فراخوانی) یک تابع

باید ابتدا نام تابع را ذکر کنیم و سپس آرگومان‌ها را بیان کنیم.

مثال- تابع زیر دو عدد را گرفته و حاصلضرب آنها را نشان می‌دهد.

```
>>> def SJfunc01(x,y):
```

---

۱ - در اینجا در لغت یعنی: «برگردان»

```
        return x*y

>>> SJfunc01(10,5)
50
```

مثال- تابع زیر عددی را گرفته و قرینه‌ی آن را برمی‌گرداند.

```
>>> def SJfunc02(a):
        return -1*a

>>> SJfunc02(-10)
10
>>> SJfunc02(10)
-10
```

مثال- (شبیه‌سازی تابع علامت[۱]) تابع زیر عددی را گرفته، اگر منفی بود، ۱- را برمی‌گرداند، صفر بود، صفر را برمی‌گرداند و اگر مثبت بود، ۱ را برمی‌گرداند.

```
>>> def SJfunc03(b):
        if b<0:
                return -1
        if b==0:
                return 0
        if b>0:
                return 1


>>> SJfunc03(-12)
-1
>>> SJfunc03(0)
0
>>> SJfunc03(12)
1
```

۱- sgn(signum)

مثال- اگر به تابع زیر سه عدد بدهیم، حاصل‌جمع آنها را به ما می‌دهد و اگر سه رشته بدهیم، این سه رشته را کنار هم قرار می‌دهد.

```
>>> def SJfunc04(A,B,C):
        return A+B+C

>>> SJfunc04(1,5,8)
14
>>> SJfunc04("Soh"+"Jel"+"@yahoo.com")
Traceback (most recent call last):
  File "<pyshell#52>", line 1, in <module>
    SJfunc04("Soh"+"Jel"+"@yahoo.com")
TypeError: SJfunc04() takes exactly 3 arguments (1 given)
```

در اینجا(بالا) چون در موقع فراخوانی تابع بین آرگومان‌های آن از کاما استفاده نکرده‌ایم و به اشتباه از «+» استفاده کرده‌ایم، با خطا مواجه شده‌ایم.

```
>>> SJfunc04("Soh","Jel","@yahoo.com")
'SohJel@yahoo.com'
```

مثال- تابع زیر عددی را گرفته، اگر زوج بود، even[1] را نشان می‌دهد و اگر فرد بود، odd[2] را نشان می‌دهد.

نکته:

برای حلّ این مثال از این نکته استفاده می‌کنیم که اگر باقیمانده‌ی تقسیم عددی بر عدد دو برابر با عدد صفر باشد، آنگاه آن عدد زوج است و در غیراین‌صورت آن عدد فرد خواهد بود.

```
>>> def SJfunc05(n):
        if n%2==0:
            return "even"
        elif[3] n%2!=0:
            return "odd"
```

---

۱ - یعنی: «زوج»

۲ - یعنی: «فرد»

۳ - در زبان برنامه‌نویسی پایتون، کوتاه شده‌ی کلمه‌ی «else if»، به معنی لغوی «در غیر این صورت، اگر» است.

```
>>> SJfunc05(11)
'odd'
>>> SJfunc05(12)
'even'
```

بهترین شیءهای پایتون تابع‌ها هستند

تابع‌ها می‌توانند به صورت دیگر نوع‌های داده‌ای مورد استفاده قرار گیرند؛ تابع‌ها می‌توانند آرگومان تابع باشند؛ مقدارهای تابع‌ها را برگردانند؛ به متغیّرها نسبت داده شوند؛ قسمتی از تاپل‌ها، لیست‌ها و غیره باشند.

مثال-

```
>>> def SJfunc10(x):
        return x*3

>>> def SJfunc11(q,x):
        return q(x)

>>> SJfunc11(SJfunc10,15)
45
```

عبارت‌های منطقی

True و False

True و False جزء ثابت‌های موجود در پایتون می‌باشند. صفر، None و شیء خالی و متغیّر خالی، دارای ارزش False هستند. عددهای غیر صفر و شیءهایی که خالی نیستند، دارای ارزش True هستند.

عملگرهای مقایسه‌ای

عبارتند از: ==(دو مساوی پشت سر هم؛ برای برّرسی برابر بودن)، =!(برای برّرسی نابرابر بودن)، >(کوچک‌تر)، <(بزرگ‌تر)، کوچک‌تر یا مساوی(=>) و بزرگ‌تر یا مساوی(=<).

مثال‌ها:

```
>>> 3==3
True

>>> 3==4
False

>>> 3!=3
False

>>> 3!=4
True

>>> 3<3
False

>>> 3<4
True
```

```
>>> 3<=3
True

>>> 3<=2
False

>>> 3>4
False

>>> 5>4
True

>>> 5>5
False

>>> 5>=5
True

>>> 5>=4
True

>>> 5>=10
False
```

عبارت‌های منطقی بولی

می‌توانیم عبارت‌های بولی را با هم ترکیب نماییم:

a and b در صورتی درست است که a و b *هر دو درست* باشند.

a or b در صورتی درست است که دست‌کم *یا* a درست باشد و *یا* b.

not a، اگر a درست باشد، not آن، غلط خواهد بود و اگر a نادرست باشد، not آن درست خواهد بود.

در صورت نیاز می‌توانید از پرانتز برای رفع ابهام از عبارت‌های بولی طولانی استفاده نمایید.

ویژگی‌های ویژه‌ی and و or

and و or دقیقاً True یا False را برنمی‌گردانند، حتّا ممکن است مقداری غیر بولی را برگردانند؛ مثلاً در عبارت X and Y and Z اگر همه‌ی عملوندها (X,Y,Z) درست باشند، مقدار Z برگردانده می‌شود و در غیر این صورت مقدار اوّلین زیر عبارت غلط برگردانده می‌شود؛ یا در عبارت X or Y or Z اگر همه‌ی عملوندها غلط باشند، مقدار Z برگردانده می‌شود و در غیر این صورت مقدار اوّلین زیر عبارتی که True است، برگردانده خواهد شد.

حُقّه‌ای در and و or

به عبارت «result = test and expr1 or expr2» توجّه کنید؛ در این عبارت اگر test درست باشد، result، expr1 است؛ ولی اگر test نادرست باشد result برابر با expr2 است. این عبارت تقریباً شبیه عبارت زیر که در C++ موجود است، عمل می‌کند:

«test ? expr1: expr2»

مثال- خروجی برنامه‌ی زیر چون a زوج است، even خواهد بود.

```
>>> a=10
>>> result=a%2==0 and "even" or "odd"
>>> print(result)
even
```

ولی در زیر چون a فرد است، خروجی odd است.

```
>>> a=11
>>> result=a%2==0 and "even" or "odd"
>>> print(result)
odd
```

نکته‌ی مهم:

از به کار بردن این حقّه خودداری کنید، چون گاهی اوقات ممکن است درست کار نکند؛ ولی ممکن است آن را در برنامه‌ها ببینید. استفاده از این حقّه با وجود عبارت‌های شرطی[1] که در پایتون نسخه‌ی ۲٫۶ وجود دارد، غیرلازم است.

۱- conditional expressions

عبارت‌های شرطی

موردی جدید است که در پایتون نسخه‌ی 2.6 به بعد وجود دارد و دارای ساختار کلّی زیر می‌باشد:

x= true_value if condition else false_value

در این ساختار ابتدا condition(شرط) مورد ارزیابی قرار می‌گیرد، اگر condition درست باشد، true_value ارزیابی می‌شود و برگردانده می‌شود و در صورتی که condition نادرست باشد، false_value مورد ارزیابی قرار گرفته و برگردانده می‌شود. توصیّه می‌کنیم که این ساختار را با پرانتز و به صورت زیر به کار ببرید:

x= (true_value if condition else false_value)

مثال:

```
>>> S1=20
>>> S2=("even" if S1%2==0 else "odd")
>>> S2
'even'

>>> S1=21
>>> S2=("even" if S1%2==0 else "odd")
>>> S2
'odd'
```

مثال:

```
>>> S3=10
>>> S4=("good" if S3>=0 else "bad")
>>> S4
'good'

>>> S3=-10
>>> S4=("good" if S3>=0 else "bad")
>>> S4
'bad'
```

کنترل جریان [برنامه]

در پایتون چند عبارت وجود دارد که جریان یک برنامه را کنترل می‌کند؛ همه‌ی آنها از برّرسی شرط‌های بولی استفاده می‌نمایند و عبارتند از:

- عبارت‌های شرطی(if)
- حلقه‌های while
- عبارت‌های [برّرسی کننده‌ی] وضعیّت(assert)

عبارت‌های شرطی

مثال- در برنامه‌ی زیر پس از اجرا ابتدا پیغام «Please enter an integer number:» به شما نشان داده می‌شود و برحسب اینکه عددی که وارد کرده‌اید، مثبت، صفر یا منفی باشد، پیام مناسبی به شما نشان داده خواهد شد.

```
SJ20=int(input("Please enter an integer number: "))
if SJ20<0:
   print("negative")
elif SJ20==0:
   print("zero")
elif SJ20>0:
   print("positive")
```

اجرا:

```
Please enter an integer number: -15
negative
```

مثال- برنامه‌ای بنویسید که در ابتدا با نشان دادن پیام «Enter your city name:» نام شهری از شما پرسیده شود، سپس اگر شما Shiraz را وارد کردید، پیغام Fars به شما نشان داده شود، Mashhad را وارد کردید، Khorasan e Razavi به شما نشان داده شود، Tehran را وارد کردید، Tehran به شما نشان داده شود، اگر هیچ چیزی وارد نکردید، Nothing به شما نشان داده شود و در غیر این صورت پیام Unknown city به شما نشان داده شود.

```
SJ21=input("Enter your city name: ")
if SJ21=="Shiraz":
   print("Fars")
elif SJ21=="Mashhad":
   print("Khorasan e Razavi")
elif SJ21=="Tehran":
   print("Tehran")
elif SJ21=="'':
   print("Nothing")
else:
   print("Unknown city")
```

در زیر نتیجه‌ی سه بار اجرای این برنامه را مشاهده می‌نمایید:

```
>>> ================== RESTART ==================
>>>
Enter a string: Shiraz
Fars

>>> ================== RESTART ==================
>>>
Enter a string:
Nothing

>>> ================== RESTART ==================
>>>
Enter a string: Abadan
Unknown city
```

---

۱ - دو کوتیشن دوتایی پشت سر هم

ممکن است در برنامه‌ی شما elifهای بیش‌تری وجود داشته باشد و یا اینکه هیچ elifی وجود نداشته باشد. کلمه‌ی کلیدی elif، کوته نوشتی برای else if (به معنای لغوی «در غیر این صورت اگر») است. رشته عبارت ... elif ... elif ... if جانشینی برای switch یا case که در زبان‌های دیگر موجودند، می‌باشد. توجّه کنید که برای به وجود آوردن بلوک‌ها، از تورفتگی استفاده می‌شود و بعد از عبارت‌های بولی از دو نقطه‌ی بیانی(:) استفاده می‌شود.

حلقه‌های while

مثال- برنامه‌ی زیر اعداد فرد کوچک‌تر از ۱۰ را چاپ می‌کند.

```
a=1
print(a)
while a<9:
   a=a+2
   print(a)
print("از حلقه خارج شدیم")
```

خروجی این برنامه به صورت زیر است:

```
1
3
5
7
9
از حلقه خارج شدیم
```

کلمه‌های کلیدی break و continue

break مانند زبان برنامه‌نویسی سی ما را از درونی‌ترین حلقه‌ی for یا while خارج می‌کند. continue هم مثل زبان برنامه‌نویسی سی ما را به ابتدای حلقه منتقل می‌نماید.

مثال- برنامه‌ی زیر اعداد فرد از ۱ تا ۹ را چاپ می‌کند؛ هر بار که وارد حلقه‌ی while می‌شویم، یک واحد به متغیّر a افزوده می‌شود، اگر a زوج باشد، دستور if اجرا می‌شود و به وسیله‌ی دستور continue به ابتدای حلقه منتقل می‌شویم و a چاپ نمی‌شود؛ ولی اگر a فرد باشد، دستور if اجرا نمی‌شود و در نتیجه a چاپ خواهد شد.

```
a=1
print(a)
while a<9:
   a=a+1
   if a%2==0:
      continue
   print(a)
```

مثال- خروجی برنامه‌ی زیر ۲، ۴ و ۶ می‌باشد؛ وقتی که SJ30 = 6 می‌شود، دستور if اجرا شده و به همین دلیل از حلقه‌ی while خارج می‌شویم.

```
SJ30=2
print(SJ30)
while SJ30<10:
   SJ30=SJ30+2
   print(SJ30)
   if SJ30==6:
      break
```

عبارت‌های [برّرسی کننده‌ی] وضعیّت(assert)

assert درست بودن یک شرط را در موقع اجرای برنامه برّرسی می‌کند و اگر شرط برقرار نبود، برنامه را با دادن خطا متوقّف می‌کند.

مثال- برنامه‌ی زیر تا وقتی که a<5 باشد، اجرا می‌شود و وقتی که a≥5 باشد؛ یعنی، شرط برقرار نباشد، با دادن خطا متوقّف می‌شود.

```
a=0
print(a)
while a<10:
```

```
a=a+1
print(a)
assert(a<5)
```

با اجرای برنامه‌ی بالا خروجی زیر را خواهیم داشت:

```
>>> ================ RESTART ================
>>>
0
1
2
3
4
5
Traceback (most recent call last):
  File "O:/PYTHON3.2.3 SohJel/PYTHONsohjel32.py", line 6, in <module>
    assert(a<5)
AssertionError
>>>
```

ساخت لیست‌های جدید

پایتون دارای خصیصه‌ای قدرتمند برای تولید یک لیست جدید با به کارگیری یک عمل برای هر عضو لیست اصلی است. برنامه‌نویس‌های پایتون از این خصوصیّت به طور گسترده‌ای استفاده می‌نمایند. شما در برنامه‌های پایتون موردهای زیادی را از آن خواهید دید. در این ساختار گاهی حقّه‌هایی را هم مشاهده خواهید کرد و از for، in و if، در این ساختار استفاده می‌کنیم. شکل کلّی این ساختار به صورت زیر است:

[expression for name in list]

که در آن expression عبارتی محاسباتی یا عملی روی متغیّر name است.

مثال– در برنامه‌ی زیر با استفاده از این توانایی ذکر شده در پایتون، هر عنصر لیست در عدد ۲ ضرب می‌شود.

```
>>> SJli10=[1,2,3,4,5]
>>> [SJ*2 for SJ in SJli10]
[2, 4, 6, 8, 10]
```

مثال– در برنامه‌ی زیر به هر عنصر موجود در لیست، رشته‌ی «@yahoo.com» اضافه خواهد شد.

```
>>> SJli11=["sohjel","sohrabejelvehgar","sohrab_jelvehgar"]
>>> [elem+"@yahoo.com" for elem in SJli11]
['sohjel@yahoo.com', 'sohrabejelvehgar@yahoo.com', 'sohrab_jelvehgar@yahoo.com']
```

مثال:

```
>>> SJli12=['S',1,'o',2,'h',3]
>>> [sj*10 for sj in SJli12]
['SSSSSSSSSS', 10, 'oooooooooo', 20, 'hhhhhhhhhh', 30]
```

مثال– در مثال زیر چون برای رشته‌های موجود در لیست نمی‌توانیم از عملگر منها(–) استفاده کنیم، با خطا مواجه شده‌ایم.

```
>>> SJli14=["J",2,"e",4,"l",6]
>>> [j-2 for j in SJli14]
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    [j-2 for j in SJli14]
  File "<pyshell#7>", line 1, in <listcomp>
    [j-2 for j in SJli14]
TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

مثال:

```
>>> SJli15=[("Soh",1),("Jel",2)]
>>> [n*100 for (x,n) in SJli15]
[100, 200]
>>> SJli15
[('Soh', 1), ('Jel', 2)]

>>> SJli16=[n*100 for (x,n) in SJli15]
>>> SJli16
[100, 200]
```

مثال:

```
>>> SJli17=[("Soh",1),("Jel",2)]
>>> SJli18=[x,n*100 for (x,n) in SJli17]
SyntaxError: invalid syntax
```

در اینجا(بالا) به دلیل این که در قسمت expression خواسته‌ایم از دو عبارت محاسباتی استفاده کنیم، با خطا برخورد نموده‌ایم.

```
>>> SJli18=[(x,n*100) for (x,n) in SJli17]
>>> SJli18
[('Soh', 100), ('Jel', 200)]
```

نکته:

expression می‌تواند شامل تابع‌هایی که کاربر تعریف کرده است، باشد.

مثال– در برنامه‌ی زیر قرینه‌ی اعداد موجود درلیست با استفاده از تابع محاسبه می‌شود.

```
>>> SJli19=[-2,-1,0,1,2,3]
>>> def SJfunc11(x):
        return -1*x

>>> SJli20=[SJfunc11(y) for y in SJli19]
>>> SJli20
[2, 1, 0, -1, -2, -3]
```

مثال– در برنامه‌ی زیر لیستی از زوج‌ها را داریم که با استفاده از تابع جای عنصرهای موجود در هر زوج با هم تعویض می‌شود.

```
>>> def SJfuncSWAP(a,b):
        return b,a

>>> SJli21=[(-1,1),(1,-1),("j",'s')]
>>> SJli22=[SJfuncSWAP(x,y) for (x,y) in SJli21]
>>> SJli22
[(1, -1), (-1, 1), ('s', 'j')]
```

مثال– در برنامه‌ی زیر دوتایی‌های موجود در هر عنصر لیست به وسیله‌ی تابع با هم جمع می‌شوند.

```
>>> SJli23=[(1,2),(3,4),(5,6)]
```

```
>>> def SJfuncADD2(x,y):
        return x+y

>>> [SJfuncADD2(a,b) for (a,b) in SJli23]
[3, 7, 11]
```

مثال– در برنامه‌ی زیر سه‌تایی‌های موجود در هر عنصر لیست با استفاده از تابع با هم جمع می‌شوند.

```
>>> SJli24=[("S","o","h"),('J','e','l'),(200,20,7)]
>>> def SJfuncADD3(f,g,h):
        return f+g+h

>>> [SJfuncADD3(x,y,z) for (x,y,z) in SJli24]
['Soh', 'Jel', 227]
```

نکته:

در این ساختار می‌توانیم با استفاده از if شرطی را هم تعیین کنیم، تا عملیّات بر روی عنصرهایی خاص از لیست انجام شوند.

مثال– در این برنامه فقط عنصرهای زوج موجود در لیست قرینه می‌شوند.

```
>>> SJli25=[1,2,3,4,5,6,7,8,9,10]
>>> [u*(-1) for u in SJli25 if u%2==0]
[-2, -4, -6, -8, -10]
```

مثال– در این برنامه فقط لیست توان ۳(مکعّب) اعداد کوچک‌تر از عدد ۶ موجود در لیست برگردانده می‌شود.

```
>>> SJli26=[1,2,3,4,5,6,7,8,9]
>>> [e*e*e for e in SJli26 if e<6]
[1, 8, 27, 64, 125]
```

نکته:

از آنجایی که در این ساختار یک لیست به صورت ورودی دریافت می‌شود و حاصل هم یک لیست است، می‌توانیم به راحتی ساختارهای تودرتو به وجود بیاوریم.

مثال– در ساختار تودرتوی زیر در ابتدا عنصرهای موجود در لیست با عدد سه جمع می‌شوند و سپس عنصرهای لیست به وجودآمده با عدد دو جمع می‌شوند.

```
>>> SJli27=[1,2,3,4,5]
```

```
>>> [f+2 for f in [g+3 for g in SJli27]]
[6, 7, 8, 9, 10]
```

حلقه‌های for

ساختاری که در بخش قبل بیان کردیم:

(expression for name in list if filter)

و عملیّات از هم جدا کردن[۱] یا به هم متّصل کردن[۲] معمولاً در زبان‌های دیگر به حلقه‌ی for نیاز دارند؛ بنابراین، برنامه‌های پایتون از تعداد کم‌تری حلقه‌های for استفاده می‌کنند. با این وجود هنوز هم یادگیری حلقه‌های for لازم است.

نکته:

کلمه‌های کلیدی for و in در ساختار

expression for name in list if filter

هم استفاده می‌شوند، امّا در مجموع با حلقه‌ی for تفاوت دارند.

مثال:

```
>>> SJli28=['.blogfa.com','.persianblog.ir','.loxblog.ir']
>>> for x in SJli28:
        print('sohjel'+x)
```

این عبارت به این معنی است که: «برای xهای موجود در لیست SJli28، 'sohjel'+x را چاپ کن.»

خروجی این برنامه به صورت زیر خواهد بود:

۱- split

۲- join

sohjel.blogfa.com
sohjel.persianblog.ir
sohjel.loxblog.ir

مثال- برنامه‌ی زیر عناصر موجود در تاپل را چاپ می‌نماید.

```
>>> SJtu10=('S','o','h')
>>> for u in SJtu10:
        print(u)
```

```
S
o
h
```

مثال- برنامه زیر هر عنصر موجود در رشته را دو بار چاپ می‌نماید.

```
>>> SJstr10="SohJel"
>>> for w in SJstr10:
        print(w+w)
```

```
SS
oo
hh
JJ
ee
ll
```

تابع range[1]( )

این تابع لیست‌هایی را با استفاده از تصاعد حسابی[2] به وجود می‌آورد.

---

۱ - در لغت یعنی: «محدوده»

۲ - arithmetic progression؛ عملی ریاضی است که در آن هر عضو دارای رابطه‌ای با عضو قبلی است. (Babylon> English-English)

مثال:

```
>>> list(range(1,9,2))
[1, 3, 5, 7]
```

همان طور که مشاهده می‌کنید، پارامتر اوّل این تابع، عدد شروع است؛ پارامتر دوّم، عددی است که تا آن عدد می‌خواهیم لیست ما به وجود آید؛ و پارامتر سوّم، مقدار افزایش یا کاهش هر عدد نسبت به عدد قبلی است. حاصل این تابع هم لیستی از اعداد صحیح می‌باشد.

مثال:

```
>>> range(1,9,2)
range(1, 9, 2)
```

مثال:

```
>>> SJt=range(1,9,2)
>>> SJt
range(1, 9, 2)
>>> 3 in SJt
True
>>> 2 in SJt
False
```

نکته:

در لیستی که به وسیله‌ی این تابع به وجود می‌آید، خود پارامتر دوّم وجود ندارد.

مثال:

```
>>> list(range(9,1,-2))
[9, 7, 5, 3]
```

نکته:

مقدار افزایش یا کاهش می‌تواند مثبت یا منفی باشد، ولی به صورت اعشاری نمی‌تواند باشد.

مثال:

```
>>> list(range(2,4,0.5))
Traceback (most recent call last):
  File "<pyshell#61>", line 1, in <module>
```

```
list(range(2,4,0.5))
TypeError: 'float' object cannot be interpreted as an integer
```

مثال‌های بیش‌تر:

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> list(range(-10))
[ ]

>>> list(range(10,2))
[ ]

>>> list(range(2,10))
[2, 3, 4, 5, 6, 7, 8, 9]
```

مثال- برنامه‌ی زیر توان دوّم(مربّع) عددهای از یک تا ده را چاپ می‌نماید.

```
>>> for SJ in range(1,10):
        print(SJ*SJ)


1
4
9
16
25
36
49
64
81
```

مثال- خروجی برنامه‌ی زیر عددهای فرد یک رقمی است.

```
>>> for SJ in range(1,10,2):
        print(SJ)
```

1
3
5
7
9

نکته:

در صورتی که ساختار حلقه‌ی for به صورت زیر باشد، <item> می‌تواند پیچیده‌تر از نام یک متغیّر تک باشد:

```
for <item> in <collection>:
        <statements>
```

مثال:

```
>>> for (x,y,z) in [('S',1,2),('o',2,3),('h',3,4)]:
        print(x,z)
```

S 2
o 3
h 4

برخی موردهای جالب در مورد تابع‌ها

تعیین مقدارهای پیش‌فرض برای آرگومان‌ها

می‌توانیم مقدارهایی را به صورت پیش‌فرض برای آرگومان‌های تابع تعیین نماییم؛ این آرگومان‌ها در زمان فراخوانی تابع به صورت اختیاری هستند.

مثال- در تعریف تابع زیر آرگومان‌های y و z به صورت پیش‌فرض دارای مقدار هستند؛ در موقع فراخوانی تابع اگر این آرگومان‌ها ذکر نشوند، از مقدارهای پیش‌فرض استفاده می‌شود.

```
>>> def SJfunc30(x,y="yth",z="on"):
        return x+y+z

>>> SJfunc30("Soh","Jel",".py")
'SohJel.py'
```

در اینجا(بالا) به دلیل اینکه هر سه آرگومان تابع ذکر شده است، از مقدارهای پیش‌فرض استفاده نشده است.

```
>>> SJfunc30("Soh","Jel")
'SohJelon'
```

در اینجا(بالا) آرگومان‌های x و y ذکر شده‌اند و به دلیل اینکه آرگومان z بیان نشده است، از مقدار پیش‌فرض آن؛ یعنی، on استفاده شده است.

```
>>> SJfunc30("P")
'Python'
```

در اینجا(بالا) فقط آرگومان x ذکر شده است و برای آرگومان‌های y و z از مقدارهای پیش‌فرض که در تعریف تابع تعیین شده‌اند، استفاده شده است.

ترتیب آرگومان‌ها

می‌توانیم آرگومان‌های تابع‌ها را بدون رعایت ترتیبی که در تعریف تابع مشخّص کرده‌ایم، فراخوانی نماییم؛ به این آرگومان‌ها که بدون در نظر گرفتن ترتیب فراخوانی می‌شوند، *آرگومان‌های کلیدی*[۱] می‌گویند.

مثال:

```
>>> def SJfunc32(x,y,z):
        return x+y+z

>>> SJfunc32("S","o","h")
'Soh'

>>> SJfunc32("S",z="h",y="o")
'Soh'

>>> SJfunc32(z="h",x="s",y="o")
'soh'
```

نکته‌ی مهم:

آرگومان‌های غیرکلیدی نمی‌توانند بعد از آرگومان‌های کلیدی قرار گیرند.

مثال:

```
>>> SJfunc32(z="h","S","o")
SyntaxError: non-keyword arg after keyword arg
```

۱- keyword arguments

انتساب[1] چندگانه (یادآوری) و ظرف(متغیّر)های خالی

انتساب چندگانه (یادآوری)

در گذشته انتساب چندگانه را به صورت زیر دیدیم:

```
>>> X,Y="Soh",123
```

که در این صورت داریم:

```
>>> X
'Soh'
>>> Y
123
```

با رعایت نوع می‌توانید انتساب‌های به صورت زیر هم داشته باشید:

```
>>> (x, y, (w, z)) = (2, 3, (4, 5))
```

که در این صورت داریم:

```
>>> x
2
>>> y
3
>>> w
4
>>> z
```

---

۱ - assignment(نسبت دادن یا گمارش)

5

```
>>> [u, v] = [4, 5]
```

که در این صورت داریم:

```
>>> u
4
>>> v
5
```

انتساب یک نام را در صورتی که در گذشته وجود نداشته باشد، به وجود می‌آورد؛ مثلاً x=3 نام x را با نوع صحیح به وجود می‌آورد.

ظرف(متغیّر)های خالی

می‌توانیم ظرف(متغیّر)های خالی را هم به وجود بیاوریم:

مثال- در زیر به ترتیب، یک لیست خالی، یک تاپل خالی و یک فرهنگ(دیکشنری) خالی را ساخته‌ایم.

```
>>> SJli30=[ ]
>>> SJtu30=( )
>>> SJdi30={ }
```

نکته:

همان طور که در گذشته هم گفتیم، یک ظرف خالی، به صورت منطقی با False برابر است (درست مانند None).

نکته:

همان طور که در گذشته هم گفتیم، اگر قبل از ایجاد یک لیست، از متد ( )append استفاده نماییم، با خطا مواجه می‌شویم؛ برای انجام این کار می‌توانیم اوّل این لیست را به صورت خالی به وجود بیاوریم و سپس با دستور ( )append عنصری را که می‌خواهیم، به آن اضافه نماییم.

مثال:

```
>>> SJli40.append("fg")
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    SJli40.append("fg")
```

NameError: name 'SJli40' is not defined

در اینجا(بالا) به دلیل اینکه در گذشته لیست را تعریف نکرده‌ایم، با خطا مواجه شده‌ایم.

```
>>> SJli41=[ ]
>>> SJli41.append("fg")
>>> SJli41
['fg']
```

در کد بالا در ابتدا لیست را به صورت بدون محتوا(خالی) به وجود آورده‌ایم و سپس با استفاده از متد append() عنصری را به آن اضافه کرده‌ایم.

عملیّات بر روی رشته‌ها

عملگر قالب‌بندی رشته(%)

این عملگر به ما اجازه‌ی کنترل کردن چگونگی نمایش رشته‌ی خروجی را می‌دهد. همچنین اگر ما بخواهیم تعداد رقم‌های بعد از اعشار یک عدد را کنترل کنیم، می‌توانیم از این عملگر استفاده نماییم. این عملگر خیلی شبیه دستور sprint موجود در زبان برنامه‌نویسی سی است.

مثال:

```
>>> S="X"
>>> J=3.14
>>> "%s = %d" %(S,J)
'X = 3'
```

در موقع اجرا به جای %s، رشته‌ی S قرار داده می‌شود و به جای %d، - به دلیل اینکه از %d برای قالب‌بندی عددهای صحیح استفاده می‌شود– عدد سه(۳) قرار می‌گیرد (مقدار J؛ یعنی، 3.14، به نوع صحیح؛ یعنی، ۳ تبدیل می‌شود.)

```
>>> "%s = %f" %(S,J)
'X = 3.140000'
```

همان طور که دیدیم، می‌توانیم از تابع print برای نمایش یک رشته بر روی صفحه‌ی نمایش استفاده نماییم و با استفاده از عملگر ٪ هم می‌توانیم رشته‌ی خروجی را قالب‌بندی نماییم.

مثال:

```
>>> print("%s - %d" %("SJ",20))
SJ - 20
```

نکته:

اگر بین دو رشته، در تابع print، از کاما استفاده کنیم، بین آنها یک فضای خالی قرار می‌گیرد.

مثال:

```
>>> print("Soh","Jel")
Soh Jel
```

تبدیل رشته‌ها

تبدیل لیست به یک رشته (متد ( )join)

این متد لیستی از رشته‌ها را به یک رشته تبدیل می‌کند.

مثال:

```
>>> t=".".join(["sohjel","blogfa","com"])
>>> t
'sohjel.blogfa.com'
```

همان طور که در مثال بالا می‌بینید، می‌توان رشته‌ای (کاراکتری) را تعیین کرد تا به عنوان جداکننده بین رشته‌های موجود در لیست قرار گیرد؛ در مثال بالا از کاراکتر نقطه(.) به عنوان جداکننده استفاده شده است.

```
>>> u="".join(["S","o","h","J","e","l"])
>>> u
'SohJel'
```

در مثال بالا از دو دابل کوتیشن پشت سر هم("") به عنوان جداکننده استفاده کرده‌ایم، پس در خروجی('SohJel') بین کاراکترها هیچ فاصله‌ای وجود ندارد.

مثال:

```
>>> v="723".join(["S","o","h","J","e","l"])
>>> v
'S723o723h723J723e723l'
```

تبدیل یک رشته به لیستی از رشته‌ها (متد ( )split)

با استفاده از این متد می‌توان یک رشته را براساس یک جداکننده به لیستی از رشته‌ها تبدیل نمود.

مثال- همان طور که در زیر می‌بینید، رشته‌ی "S,o,h,J,e,l" براساس جداکننده‌ی کاما به لیستی از رشته‌ها تبدیل شده است.

```
>>> "S,o,h,J,e,l".split(",")
['S', 'o', 'h', 'J', 'e', 'l']
```

تبدیل نوع‌های مختلف داده‌ای به یک رشته

در پایتون تابعی به نام str( ) وجود دارد که با استفاده از آن می‌توان هر نوع داده‌ای را به نوع رشته‌ای تبدیل نمود.

مثال:

```
>>> "SJ"+str(20)
'SJ20'
```

مثال:

```
>>> SJ+str(20)
Traceback (most recent call last):
```

```
  File "<pyshell#0>", line 1, in <module>
    SJ+str(20)
NameError: name 'SJ' is not defined
```

در مثال بالا چون متغّیر SJ در گذشته تعریف نشده است، با خطا مواجه شده‌ایم؛ باید قبلاً آن را از نوع رشته‌ای تعریف می‌کردیم.

```
>>> SJ=10
>>> SJ+str(20)
Traceback (most recent call last):
  File "<pyshell#128>", line 1, in <module>
    SJ+str(20)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> SJ='sj'
>>> SJ+str(20)
'sj20'
```

مثال:

```
>>> SJstr60="1,2,3"
>>> SJstr60+str(4)
'1,2,34'
```

وارد کردن[۱] و نمونه(ماژول)ها[۲]

با این کار می‌توانید از کلاس‌ها و تابع‌هایی که در فایل‌های دیگر تعریف شده‌اند، استفاده نمایید. یک نمونه یا ماژول، در پایتون، فایلی با پسوند py است و شبیه دستور import موجود در جاوا و دستور include موجود در C++ می‌باشد.

---

۱ - importing

۲ - modules

مثال- فرض کنید می‌خواهیم یک نمونه(ماژول) بنویسیم که با وارد کردن و اجرای تابع نوشته شده در آن، عبارت " This is a test." نوشته شود؛ برای این کار ابتدا وارد محیط IDLE پایتون شده و از منوی File، گزینه‌ی New window را انتخاب می‌کنیم و فایلی به نام SJpy01.py را با محتوای زیر به وجود می‌آوریم و *در همان درایوی که پایتون نصب شده، در پوشه‌ی Python32 ذخیره می‌کنیم.*

```
def SJfunc80( ):
   print('This is a test.')
```

حال در محیط IDLE پایتون، دستور زیر را نوشته و با این کار، فایل SJpy01 را وارد می‌کنیم:

```
>>> import SJpy01
```

حال برای اجرای تابع ( )SJfunc80، عبارت زیر را می‌نویسیم؛ *توجّه کنید که اوّل باید نام نمونه را نوشته و سپس یک نقطه گذاشته و سپس نام تابعی را که می‌خواهیم اجرا کنیم، بنویسیم:*

```
>>> SJpy01.SJfunc80( )
This is a test.
```

اضافه نمودن دایرکتوری‌های دیگر برای نمونه‌ها

برای اینکه دایرکتوری‌های دیگری را برای نمونه‌ها تعیین کنید، می‌توانید از فرمان زیر استفاده نمایید:

sys.path.append(مسیر دایرکتوری‌ای که نمونه(ها) در آن قرار دارد)

مثال- فایل نمونه‌ای را به نام SJmodule2.py در مسیر "d:\MyModules\" با محتوای زیر به وجود می‌آوریم.

```
def SJmodule02( ):
   print('This is my second module')
```

سپس دستور زیر را در محیط GUI (IDLE) پایتون می‌نویسیم:

```
>>> import SJmodule2
```

در این هنگام چونکه پایتون هنوز مسیر "d:\MyModules\" را نمی‌شناسد، با خطای زیر مواجه می‌شویم:

```
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    import SJmodule2
ImportError: No module named SJmodule2
```

برای شناساندن این مسیر باید ابتدا نمونه‌ی sys را که در خود پایتون وجود دارد، وارد نماییم:

```
>>> import sys
```

سپس با استفاده از دستور زیر مسیر خود را به پایتون می‌شناسانیم:

```
>>> sys.path.append('d:\MyModules')
```

نکته:

اگر دستور import sys را به کار نبرده باشیم، با اجرای دستور بالا با خطای زیر مواجه می‌شویم.

```
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    sys.path.append('d:\MyModules')
NameError: name 'sys' is not defined
```

حال مطابق مثال قبل می‌توانیم نمونه‌ای را که خودمان نوشته‌ایم را وارد کرده و استفاده کنیم:

```
>>> import SJmodule2
>>> SJmodule2.SJmodule02( )
This is my second module
```

چکیده‌ی مطلب‌های فصل بیست و پنجم

پایتون به بزرگ و کوچک بودن حروف حسّاس می‌باشد.

اوّلین خط با تورفتگی کم‌تر، خارج از بلوک است. اوّلین خط با تورفتگی بیش‌تر، یک بلوک تودرتو را به وجود می‌آورد.

پایتون نوع‌های داده‌ای را به صورت خودکار برای متغیّرهای موجود در برنامه تشخیص می‌دهد؛ امّا این مطلب به معنی این نیست که پایتون در مورد نوع داده‌ها حسّاس نمی‌باشد.

اگر شما بخواهید برای متغیّرها از نامی استفاده نمایید که در گذشته با استفاده از مقداردهی آن را به وجود نیاورده‌اید، با خطا مواجه خواهید شد.

تمام سه نوع تاپل، لیست و رشته از لحاظ ظاهر و کارکرد شبیه هم هستند؛ فرق اصلی آنها در این است که تاپل‌ها و رشته‌ها تغییرناپذیر هستند؛ ولی لیست‌ها تغییرپذیر می‌باشند.

می‌توانیم به هر عنصر تاپل، لیست و رشته با استفاده از آرایه دسترسی پیدا کنیم؛ توجّه کنید که اندیس آرایه از عدد صفر شروع می‌شود.

در اندیس مثبت، شمارش از سمت چپ شروع می‌شود و شروع اندیس از عدد صفر است، ولی در اندیس منفی، شمارش از سمت راست است و اندیس از عدد منفی یک شروع می‌شود.

اگر با استفاده از متد append بخواهیم یک لیست را به لیست اصلی اضافه کنیم، خود لیست به لیست اصلی اضافه می‌شود، ولی اگر در این مورد از متد extend استفاده کنیم، عنصرهای موجود در لیست به لیست اصلی اضافه خواهند شد.

دیکشنری‌ها یک نگاشت را میان یک مجموعه از کلیدها و یک مجموعه از مقدارها نگهداری می‌کنند.

برای برداشتن یکی از فقره‌های موجود در دیکشنری از دستور del استفاده می‌کنیم و برای حذف کامل فقره‌های موجود در دیکشنری از متد ( )clear استفاده می‌نماییم.

کلمه‌ی کلیدی break ما را از درونی‌ترین حلقه‌ی for یا while خارج می‌کند.

کلمه‌ی کلیدی continue ما را به ابتدای حلقه منتقل می‌نماید.

پایتون دارای خصیصه‌ای قدرتمند برای تولید یک لیست جدید با به کارگیری یک عمل برای هر عضو لیست اصلی است.

یادآوری یا تکمیل مطلب‌های فصل بیست و پنجم

تعریف– کلمه‌ی کلیدی[۱] را تعریف کنید.

جواب– کامپایلر از کلمه‌ی کلیدی برای تجزیّه کردن[۲] برنامه استفاده می‌کند؛ از کلمه‌های کلیدی، مثل if, def و while نمی‌توانید به عنوان نام متغیّر استفاده کنید.[۳]

سؤال– جواب مفسّر پایتون به عبارت زیر چیست؟؛ نوع داده‌ای حاصل چه خواهد بود؟

```
>>> "Hello" + "World" + "!"
```

جواب–

```
'HelloWorld!'
```

str.[۴]

تعریف– نوع داده‌ای int را تعریف کنید.

جواب– نوع داده‌ای‌ای عددی در پایتون است که اعداد بدون اعشار، مثل ۳–، ۵، ۰، ۱۲–، ۵۲۵ و غیره را نگه می‌دارد.[۵]

سؤال– جواب مفسّر پایتون به عبارت زیر چیست؟؛ نوع داده‌ای حاصل چه خواهد بود؟

---

۱ – keyword

۲ – parse

۳ – آزمون درس «معرّفی کامپیوتر(Intro to Computing)»، دانشکده‌ی کامپیوتر دانشگاه ایالت جُورْجیا(Georgia)ی کشور آمریکا، پاییز سال ۲۰۱۰ میلادی

۴ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، تابستان سال ۲۰۰۹ میلادی

۵ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

```
>>> 3 + 2
```

جواب-

```
5
```

int.[۱]

سؤال- جواب مفسّر پایتون به عبارت زیر چیست؟؛ نوع داده‌ای حاصل چه خواهد بود؟

```
>>> 1 + 2 * 3
```

جواب-

```
7
```

int.[۲]

تعریف- نوع داده‌ای float را تعریف کنید.

جواب- نوع داده‌ای‌ در پایتون است که اعداد اعشاری را نگه می‌دارد.[۳]

سؤال- جواب مفسّر پایتون به عبارت زیر چیست؟؛ نوع داده‌ای حاصل چه خواهد بود؟

```
>>> 1.5+3
```

جواب-

```
4.5
```

float.[۴]

سؤال- جواب مفسّر پایتون به عبارت زیر چیست؟؛ نوع داده‌ای حاصل چه خواهد بود؟

```
>>> "cs1301" * 3
```

جواب-

```
'cs1301cs1301cs1301'
```

str.[۵]

سؤال- جواب مفسّر پایتون به عبارت زیر چیست؟؛ نوع داده‌ای حاصل چه خواهد بود؟

```
>>> "a" + "b" * 2
```

جواب-

```
'abb'
```

---

۱ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، تابستان سال ۲۰۰۹ میلادی
۲ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی
۳ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی
۴ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۱۰ میلادی
۵ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، تابستان سال ۲۰۰۹ میلادی

str.[۱]

تست– کدام گزینه مُجاز است؟

۱) "A" + 5

۲) "A" - 5

۳) "A" * 5

۴) "A" / 5

جواب– گزینه‌ی «۳» درست است.[۲]

```
>>> "A" + 5
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    "A" + 5
TypeError: Can't convert 'int' object to str implicitly
>>> "A" - 5
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    "A" - 5
TypeError: unsupported operand type(s) for -: 'str' and 'int'
>>> "A" * 5
'AAAAA'
>>> "A" / 5
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    "A" / 5
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

سؤال– جواب مفسّر پایتون به عبارت زیر چیست؟؛ نوع داده‌ای حاصل چه خواهد بود؟

```
>>> (6.0-1)**2+3
```

جواب–

```
28.0
```

float.[۳]

سؤال– جواب مفسّر پایتون به عبارت زیر چیست؟؛ نوع داده‌ای حاصل چه خواهد بود؟ (int, float, str, bool)

```
>>> (6-1)**2+3
```

جواب–

---

۱ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۲ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۰ میلادی

۳ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، تابستان سال ۲۰۰۹ میلادی

28

.int

سؤال– جواب مفسّر پایتون به عبارت زیر چیست؟؛ نوع داده‌ای حاصل چه خواهد بود؟

```
>>> (3000 * 3 + 1)
```

جواب–

9001

[۱].int

سؤال– جواب مفسّر پایتون به عبارت زیر چیست؟؛ نوع داده‌ای حاصل چه خواهد بود؟

```
>>> "Thirty" + str(34) + "Four"
```

جواب–

'Thirty34Four'

[۲].str

سؤال– جواب مفسّر پایتون به عبارت زیر چیست؟

```
>>> "Thirty" + 34 + "Four"
```

جواب–

```
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    "Thirty" + 34 + "Four"
TypeError: Can't convert 'int' object to str implicitly
```

تست– کدام گزینه مُجاز است؟

۱) "A"+"B"

۲) "A"-"B"

۳) "A"*"B"

۴) "A"/"B"

جواب– گزینه‌ی «۱» درست است.[۳]

```
>>> "A"+"B"
'AB'
>>> "A"-"B"
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
```

---

۱ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۱۰ میلادی

۲ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، تابستان سال ۲۰۰۹ میلادی

۳ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۰ میلادی

```
   "A"-"B"
TypeError: unsupported operand type(s) for -: 'str' and 'str'
>>> "A"*"B"
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    "A"*"B"
TypeError: can't multiply sequence by non-int of type 'str'
>>> "A"/"B"
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    "A"/"B"
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

سؤال- جواب مفسّر پایتون به عبارت زیر چیست؟؛ نوع داده‌ای حاصل چه خواهد بود؟

```
>>> (True or False) and (not False)
```

جواب-

True

bool.[1]

تست- حاصل کدام گزینه True خواهد بود؟

۱) True or False and False

۲) True and False or False

۳) True and True and False

۴) not True or False

جواب- گزینه‌ی «۱» درست است.[2]

```
>>> True or False and False
True
>>> True and False or False
False
>>> True and True and False
False
>>> not True or False
False
```

سؤال- جواب مفسّر پایتون به عبارت زیر چیست؟؛ نوع داده‌ای حاصل چه خواهد بود؟

```
>>> True and (3!=2)
```

جواب-

True

---

۱ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۱۰ میلادی

۲ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۰ میلادی

bool.[۱]

سؤال– جواب مفسّر پایتون به عبارت زیر چیست؟؛ نوع داده‌ای حاصل چه خواهد بود؟

```
>>> 7.0>5.0
```

جواب–

```
True
```

bool.[۲]

سؤال– جواب مفسّر پایتون به عبارت زیر چیست؟؛ نوع داده‌ای حاصل چه خواهد بود؟

```
>>> "a"<"c"
```

جواب–

```
True
```

bool.[۳]

سؤال– جواب مفسّر پایتون به عبارت زیر چیست؟

```
>>> print("%.2f" % (4/2))
```

جواب–

2.00[۴]

در ضمن،

```
>>> print("%.0f" % (4/2))
2
>>> print("%.f" % (4/2))
2
>>> print("%d" % (4/2))
2
>>> print("%f" % (4/2))
2.000000
>>> print("%.1f" % (4/2))
2.0
>>> print("%.5f" % (4/2))
2.00000
>>> print("%.5f" (4/2))
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    print("%.5f" (4/2))
```

---

۱ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، تابستان سال ۲۰۰۹ میلادی

۲ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، تابستان سال ۲۰۰۹ میلادی

۳ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۱۰ میلادی

۴ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

TypeError: 'str' object is not callable

سؤال– جواب مفسّر پایتون به عبارت زیر چیست؟؛ نوع داده‌ای حاصل چه خواهد بود؟

```
>>> print("Pumpkin[1] %.3f" % 3.1459)
```

جواب–

Pumpkin 3.146

str.[2]

تست– اگر x = input("Please enter a number.")، نوع داده‌ای x چیست؟

۱) bool

۲) float

۳) int

۴) str

۵) list

جواب– گزینه‌ی «۴» درست است.[3]

```
>>> x = input("Please enter a number. ")
Please enter a number. 3
>>> type(x)
<class 'str'>
```

در پایتون برای تشخیص نوع‌های داده‌ای از تابع type( ) استفاده می‌کنیم؛ مثال‌ها:

```
>>> type("2")
<class 'str'>
>>> type("2")==str
True
>>> type("2")==int
False
>>> type(12)
<class 'int'>
>>> type(3.141592)
<class 'float'>
>>> type(99999999)
<class 'int'>
>>> type(-99999999)
<class 'int'>
```

سؤال– در پایتون از کدام عملگر(اپراتور) برای برّرسی مساوی بودن استفاده می‌شود؟

---

۱ – در زبان انگلیسی در لغت به معنی «کدو تنبل» می‌باشد.

۲ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، تابستان سال ۲۰۰۹ میلادی

۳ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۰ میلادی

جواب– ==(دو مساوی پشت سر هم).[۱]

سؤال– در پایتون از کدام عملگر برای انتساب استفاده می‌شود؟

جواب– =.[۲]

سؤال– در پایتون از کدام کلمه‌ی کلیدی برای ساخت جمله‌ی شرطی استفاده می‌شود؟

جواب– if.[۳]

سؤال– در پایتون از کلمه‌های کلیدی ..... و .....، برای ساخت حلقه‌ها استفاده می‌شود.

جواب– for - while.[۴]

سؤال– پایتون روش‌هایی برای اجرای مکرّر یک بلوک از برنامه دارد. از حلقه‌ی ..... برای تکرار یک زنجیره(دنباله، رشته) یا تکرار بلوکی از برنامه برای تعداد مشخّصی دفعه استفاده می‌شود، در حالی که از حلقه‌ی ..... برای تکرار بلوکی از برنامه تا زمانی که عبارتی بولّین درست باشد، استفاده می‌شود. تابعی که از ..... استفاده می‌کند، هم می‌تواند بلوک‌هایی از برنامه را تکرار نماید، امّا برای انجام این کار باید خودش را فراخوانی کند(صدا بزند).

جواب– به ترتیب، از راست به چپ(←)، for – while – بازگشت[۵].[۶]

سؤال– در پایتون کد زیر چه چیزی را چاپ می‌کند؟

```
if (0 + 1 == True):
   print("false")
if (5/2 > 2):
   print("banana")
else:
   print("pear[۷]")
if (type("2") == int):
   print("apple")
elif (23 % 7 <= 2):
   print("peach[۱]")
```

---

۱ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، تابستان سال ۲۰۰۹ میلادی
۲ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، تابستان سال ۲۰۰۹ میلادی
۳ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، تابستان سال ۲۰۰۹ میلادی
۴ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، تابستان سال ۲۰۰۹ میلادی
۵ – recursion
۶ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۱۰ میلادی
۷ – در زبان انگلیسی در لغت به معنی «گلابی» است.

```
if (int(3.1) == float(3.0)):
   print("watermelon[۲]")
```

جواب-

```
false
banana
peach
watermelon[۳]
```

تابع int(x)، x را به نوع integer(عدد صحیح) تبدیل می‌کند. تابع float(x)، x را به نوع floating point(ممیّز شناور، عدد اعشاری) تبدیل می‌کند؛ مثال‌ها:

```
>>> int(3.1)
3
>>> int(3.9)
3
>>> int(3)
3
>>> float(3.0)
3.0
>>> float(3.1)
3.1
>>> float(3.9)
3.9
>>> float(3)
3.0
>>> int(3.1)==float(3.0)
```

سؤال- خروجی برنامه‌ی زیر چیست؟

```
x = 3
while x >= 0:
   if x%2 == 0:
      print("chocolate")
   elif x-3<0:
      print("flowers")
   else:
      print("good")
   x = x - 1
```

جواب-

```
good
chocolate
```

---

۱ - در زبان انگلیسی در لغت به معنی «هلو» است.

۲ - در زبان انگلیسی در لغت به معنی «هندوانه» است.

۳ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۱۰ میلادی

flowers
chocolate[۱]

سؤال- خروجی برنامه‌ی زیر چیست؟

```
def o(n):
   print(n*3)
   return n*2

print("Begin[۲]")
y = o(7)

if (y > 15):
   print("Breakfast[۳]")
elif (y > 10):
   print("Lunch[۴]")
if (y > 5):
   print("Dinner")
else:
   print("Dessert[۵]")

if ('a' < 'b'):
   print("Apple")
elif ('a' < 'A'):
   print("Peanut[۶]")
else:
   print("Butter[۷]")
print("Finished[۸]")
```

جواب-

```
Begin
21
Lunch
```

---

۱ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۲ - در زبان انگلیسی در لغت به معنی «شروع» است.

۳ - در زبان انگلیسی در لغت به معنی «صبحانه» است.

۴ - در زبان انگلیسی در لغت به معنی «ناهار» است.

۵ - در زبان انگلیسی در لغت به معنی «دِسِر» است.

۶ - در زبان انگلیسی در لغت به معنی «بادام زمینی» است.

۷ - در زبان انگلیسی در لغت به معنی «کَره» است.

۸ - در زبان انگلیسی در لغت به معنی «تمام» است.

Dinner
Apple
Finished[1]

سؤال– خطای برنامه‌ی زیر را پیدا کنید.

```
e = "2.718"
pi = 3.14
pie = str(pi) + e
print(int(e))
print(int(pi))
print(pie)
```

جواب– خطّ چهارم(print(int(e)) دارای خطا است؛ شما نمی‌توانید یک رشته که به صورت عدد اعشاری است را به نوع صحیح(int) تبدیل کنید.[2]؛ حاصل اجرای کد:

```
e = "2"
pi = 3.14
pie = str(pi) + e
print(int(e))
print(int(pi))
print(pie)
```

به صورت زیر است:

```
2
3
3.142
```

سؤال– در پایتون کد زیر چه چیزی را چاپ می‌کند؟

```
if 5<5:
   print("A")
if 5+5:
   print("B")
elif 5==5:
   print("C")
if "D":
   print("D")
if 0:
   print("E")
else:
   print("F")
```

جواب–

```
B
D
```

---

۱ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی

۲ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۰ میلادی

F[۱]

سؤال- کد زیر چه چیزی را چاپ می‌کند؟

```
>>> list(range(3,7))
```

جواب-

[3, 4, 5, 6][۲]

سؤال- کد زیر چه چیزی را چاپ می‌کند؟

```
>>> list(range(3,10,2))
```

جواب-

[3, 5, 7, 9][۳]

سؤال- کد زیر چه چیزی را چاپ می‌کند؟

```
>>> list(range(3,20,4))
```

جواب-

[3, 7, 11, 15, 19][۴]

سؤال- در پایتون کد زیر چه چیزی را چاپ می‌کند؟

```
for myInt in range(2,15,3):
   if (myInt % 2 == 0):
      print(myInt * 2)
   else:
      print(myInt)
```

جواب-

4
5
16
11
28[۵]

سؤال- در پایتون کد زیر چه چیزی را چاپ می‌کند؟

```
l = ["open", "close", "in", "out", "up", "down" ]
```

---

۱ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۱۰ میلادی
۲ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۰۹ میلادی
۳ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۰۹ میلادی
۴ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۱ میلادی
۵ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، تابستان سال ۲۰۰۹ میلادی

```
for i in range(0,6,2):
   print(l[i])
```

جواب-

```
open
in
up[1]
```

سؤال- پس از اجرای خطوط زیر چه چیزی چاپ خواهد شد؟

```
>>> s = "look, contents!"[2]
>>> print(s[ :7:2])
```

جواب-

```
>>> s = "look, contents!"
>>> print(s[ :7:2])
lo,c[3]
>>> print(s[ :7:1])
look, c
>>> print(s[ :7:3])
lkc
>>> print(s[ :7:4])
l,
>>> print(s[1:7:1])
ook, c
>>> print(s[1:7:2])
ok
>>> print(s[1:7:3])
o,
```

سؤال- تابعی به نام iN بنویسید که عددی صحیح را گرفته و مقداری بولّین را برگرداند. اگر عدد گرفته شده منفی بود، تابع باید مقدار بولّین True را برگرداند. اگر عدد گرفته شده صفر یا مثبت بود، تابع باید مقدار بولّین False را برگرداند.
به عنوان مثال، موارد آزمون می‌توانند به صورت زیر باشند:

```
>>> iN(10)
False
>>> iN(0)
False
>>> iN(-10)
True
```

جواب-

---

۱ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۱۰ میلادی
۲ - در زبان انگلیسی به معنی «به محتویّات(مندرجات) نگاه کن!» است.
۳ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۰ میلادی

```
def iN( aNum):
   if aNum < 0:
      return True
   else:
      return False
```[۱]

سؤال- تابعی به نام h را طوری بنویسید که عددی صحیح و مثبت را گرفته و از آن عدد تا عدد صفر را با استفاده از حلقه‌ی while به صورت نزولی چاپ کند؛ مثلاً:

```
>>> h(5)
5
4
3
2
1
0
```

جواب-

```
def h(n):
   while n >= 0:
      print(n)
      n = n - 1
```[۲]

سؤال- تابعی به نام factorial را طوری بنویسید که عددی صحیح و مثبت را گرفته و فاکتوریل آن را چاپ کند.

جواب-

راه حلّ ۱:

```
def factorial( aNum):
   if (aNum == 0):
      return 1
   return aNum * factorial(aNum-1)
```

راه حلّ ۲:

```
def factorial( aNum):
   product = 1
   index = 1
   while(index <= aNum):
      product = product * index;
      index = index + 1
   return( product )
```[۳]

نمونه‌ای از اجرا:

---

۱ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۱۰ میلادی

۲ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۰ میلادی

۳ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۰ میلادی

```
>>> factorial(5)
120
```

سؤال– تابعی به نام K بنویسید که رشته‌ای را گرفته و تعداد کاراکترهای جِیِ بزرگ انگلیسی(J) موجود در این رشته را شمرده و چاپ کند.

جواب–

```
def K(str):
   c = 0
   for l in str:
      if( l == "J"):
         c = c + 1
   return( c)'
```

نمونه‌هایی از اجرای این تابع:

```
>>> K("")
0
>>> K(SJ)
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    K(SJ)
NameError: name 'SJ' is not defined
>>> K("SJ")
1
>>> K("sj")
0
>>> K("SSjJJJmJ)

SyntaxError: EOL while scanning string literal
>>> K("SSjJJJmJ")
4
```

تست– در پایتون کدامیک از تعریف‌های تابع زیر درست است؟

```
1)
def myFunc( ):
        print("Hello!")
2)
define myFunc( ):
        print("Hello!")
3)
def myFunc( )
        print("Hello!")
```

۴) هیچکدام

---

۱ – آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۰۹ میلادی

جواب- گزینه‌ی «۱» درست است.[۱]

سؤال- در پایتون کد زیر چه چیزی را چاپ می‌کند؟

```
MyVar=10
def aFunc1(MyVar):
   print(MyVar*3)
   return(5)
   print("goodbye!")
MyVar = MyVar + aFunc1("Go")
print(MyVar)
```

جواب-

```
GoGoGo
15
```
[۲]

سؤال- الف) در پایتون کد زیر چه چیزی را چاپ می‌کند؟

```
def g(x):
   print(x)
   x = int(x)
   print(x + 4)
   y = str(x)
   print(y + "5")
   return y
z = g(6.5)
```

جواب-

```
6.5
10
65
```

ب) مقدار متغیّر z را به همراه نوع آن تعیین کنید.

جواب- (رشته‌ی) "6"[۳]

سؤال- جاهای خالی تابع زیر را با کلمات مناسب پر کنید.

```
---------- oddOrEven(number):
   if number ---------- 2 == 0:
      ---------- "even"
   ----------:
      ---------- "odd"
```

---

۱ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، تابستان سال ۲۰۰۹ میلادی

۲ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، تابستان سال ۲۰۰۹ میلادی

۳ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۱۰ میلادی

جواب-

```
def oddOrEven(number):
   if number % 2 == 0:
      return "even"
   else:
      return "odd"
```
[1]

سؤال- اگر a=1 و b=5 باشد، خروجی تابع زیر چیست؟

```
def c(a,b):
   a = a + 1
   while a < b:
      print(a)
      a = a+1
```

جواب-

```
>>> c(1,5)
2
3
4
```

در ضمن:

```
>>> c(5,1)
>>>
```
[2]

سؤال- پس از اجرای خطوط زیر چه چیزی چاپ خواهد شد؟

```
>>> def f(x):
        if x == 0:
                return 1
        else:
                return f(x-1 ) + 1


>>> print(f(5))
```

جواب- ۶[3]

سؤال- حاصل test(1) چیست؟

```
def test(n):
   if n == 0:
      print("bad!")
   if n == 1:
```

---

۱ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۱۰ میلادی

۲ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۱۰ میلادی

۳ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۰ میلادی

```
    print("good!")
  if n == 2:
    print("better!")
  else:
    print("best!")
```

جواب-

```
good!
best!
```

توجّه کنید که else درون این تابع در صورتی که n برابر با ۲ نباشد اجرا می‌شود!.[۱]

سؤال- خروجی برنامه‌ی زیر چیست؟

```
def fun1(x):
  print("Fun1 x:", x)
  return x * 2

print("Start")

y = fun1(10)

if (5 > y ):
  print(y)
elif ( 15 > y ):
  print(y + 10)
elif (25 > y ):
  print(y + 100)
elif (35 > y):
  print(y + 1000)
else:
  print(y + 10000)

print("End")
```

جواب-

```
Start
Fun1 x: 10
120
End
```

۱ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، بهار سال ۲۰۱۰ میلادی

۲ - آزمون درس «معرّفی کامپیوتر»، دانشکده‌ی کامپیوتر دانشگاه ایالت جورجیای کشور آمریکا، پاییز سال ۲۰۰۹ میلادی

فهرست برخی از منبع‌ها و مأخذها

این کتاب با استفاده از منبع‌های به زبان انگلیسی درون اینترنت ترجمه شده است؛ بیش‌ترِ مطلب‌های استفاده شده برای ترجمه‌ی این کتاب، مطلب‌های موجود در پایگاه‌های اینترنتی دانشگاه‌های کشور ایالات متّحده‌ی آمریکا می‌باشد؛ در زیر فهرست برخی از منبع‌هایی که در ترجمه‌ی این کتاب از آنها استفاده شده را مشاهده می‌نمایید:[1]

۱- پایگاه اینترنتی دانشگاه شهر سانفِرانسیسکوی کشور ایالات متّحده‌ی آمریکا، به نشانی اینترنتی
www.cs.usfca.edu

۲- مطالب اِسْتوارت راسِل و پیتر نورویگ و سایر استادان که در پایگاه اینترنتی دانشکده‌ی شهر بِرکْلی(Berkeley) دانشگاه ایالت کالیفُرنیای کشور ایالات متّحده‌ی آمریکا، با نشانی اینترنتی زیر وجود دارد:
http://www.cs.berkeley.edu

۳- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر دانشگاه دولتی ایالت تِگزاس کشور ایالات متّحده‌ی آمریکا، به نشانی اینترنتی
www.cs.txstate.edu

۴- مطلب‌های دکتر، جان مِک‌کارتی که در پایگاه اینترنتی زیر وجود دارد:
http://www-formal.stanford.edu

---

۱ - توجّه کنید که یک مطلب ممکن است در منبع‌های متعدّدی ذکر شده باشد.

۵- پایگاه اینترنتی دانشگاه ایالت تگزاس کشور ایالات متّحده‌ی آمریکا، به نشانی اینترنتی
www.cs.utexas.edu

۶- پایگاه اینترنتی دانشگاه شهر پیتسبورگ کشور ایالات متّحده‌ی آمریکا، به نشانی اینترنتی
www.cs.pitt.edu

۷- پایگاه اینترنتی دانشگاه مریلَند کشور ایالات متّحده‌ی آمریکا، با نشانی اینترنتی
http://www.cs.umd.edu

۸- پایگاه اینترنتی مدرسه‌ی علوم کامپیوتر و مهندسی، به نشانی اینترنتی
www.cse.unsw.edu.au

۹- پایگاه اینترنتی جامعه‌ی هوش محاسباتی، به نشانی اینترنتی
http://ebrains.la.asu.edu/~jennie/tutorial/index.htm

۱۰- پایگاه اینترنتی مهندسان سیاهپوست، به نشانی اینترنتی
www.nsbe.org

۱۱- پایگاه اینترنتی دانشگاه آزاد شهر بوزِن کشور ایتالیا، به نشانی اینترنتی
http://www.inf.unibz.it

۱۲- پایگاه اینترنتی دانشگاه کوبلِنز کشور آلمان، به نشانی اینترنتی
http://www.uni-koblenz.de

۱۳- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر دانشگاه کورک کشور ایرلند، به نشانی اینترنتی
http://www.cs.ucc.ie

۱۴- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر و اطّلاعات کشور نُروژ، به نشانی اینترنتی
www.idi.ntnu.no

۱۵- پایگاه اینترنتی دانشگاه ملّی کشور تایوان، به نشانی اینترنتی
www.csie.ntu.edu.tw

۱۶- پایگاه اینترنتی دانشگاه لیورپول کشور انگلستان، به نشانی اینترنتی
http://www.csc.liv.ac.uk

۱۷- پایگاه اینترنتی دانشگاه خصوصی سَبِنسای کشور ترکیّه، به نشانی اینترنتی
http://people.sabanciuniv.edu

۱۸- پایگاه اینترنتی دانشگاه شمال غربی کشور ایالات متّحده‌ی آمریکا، به نشانی اینترنتی
http://www.eecs.northwestern.edu

۱۹- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر دانشگاه بولوگنای کشور ایتالیا، به نشانی اینترنتی
http://www.cs.unibo.it

۲۰- پایگاه اینترنتی آزمایشگاه زبان دانشگاه کَمبریج کشور انگلستان، به نشانی اینترنتی
http://www.cl.cam.ac.uk

۲۱- پایگاه اینترنتی دانشگاه شاه خَلید کشور عربستان سعودی، به نشانی اینترنتی
http://www.shawkani.net

۲۲- پایگاه اینترنتی دانشگاه گُلداِسمیتز دانشگاه لندن کشور انگلیس، به نشانی اینترنتی
http://www.gold.ac.uk

۲۳- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر دانشگاه ایالت کِنت کشور ایالات متّحده‌ی آمریکا، به نشانی اینترنتی
http://www.cs.kent.edu

۲۴- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر و اطّلاعات دانشگاه ایالت پِنسیلوانیای کشور ایالات متّحده‌ی آمریکا، به نشانی اینترنتی
http://www.cis.upenn.edu

۲۵- برنامه‌ی Help زبان برنامه‌نویسی پایتون و همچنین مطلب‌های در مورد زبان برنامه‌نویسی پایتون که در پایگاه اینترنتی زیر وجود دارد:
http://docs.python.org

۲۶- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر دانشگاه ویکتوریای کشور کانادا، به نشانی اینترنتی
http://webhome.cs.uvic.ca

۲۷- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر دانشگاه شهر بالتیمُور ایالت مِریلَند کشور ایالات متّحده‌ی آمریکا، با نشانی اینترنتی
http://www.cs.umbc.edu

۲۸- پایگاه اینترنتی دانشگاه استنفورد کشور ایالات متّحده‌ی آمریکا، به نشانی اینترنتی
http://www.stanford.edu

۲۹- پایگاه اینترنتی دانشگاه فنّی میشیگان کشور ایالات متّحده‌ی آمریکا، به نشانی اینترنتی
http://www.mtu.edu

۳۰- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر دانشگاه یونیون ایالت نیویورک کشور ایالات متّحده‌ی آمریکا، به نشانی اینترنتی
http://cs.union.edu

۳۱- پایگاه اینترنتی دانشکده‌ی شهر اِروایِن دانشگاه ایالت کالیفورنیای کشور ایالات متّحده‌ی آمریکا، به نشانی اینترنتی
http://www.ics.uci.edu

۳۲- پایگاه اینترنتی دانشکده‌ی مهندسی کامپیوتر، کنترل و مدیریّت آنتونیو روبِرتی دانشگاه ساپینزای کشور ایتالیا، به نشانی اینترنتی
http://www.dis.uniroma1.it

۳۳- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر و مهندسی دانشگاه ایالت تگزاس کشور آمریکا، به نشانی اینترنتی
http://robotics.cs.tamu.edu

۳۴- پایگاه اینترنتی دانشگاه McGill کشور کانادا، به نشانی اینترنتی
http://www.cs.mcgill.ca

۳۵- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر دانشگاه نیویورک کشور آمریکا، به نشانی اینترنتی
http://www.cs.nyu.edu

۳۶- برنامه‌ی Help زبان برنامه نویسی پرولوگ و همچنین «سؤالات رایج پرسیده شده و جواب داده شده»[۱]ی پرولوگ، به نشانی اینترنتی
http://www.swi-prolog.org/FAQ

۳۷- دانشکده‌ی نوآوری(ابداع)، طرّاحی و مهندسی دانشگاه مالاردالِن(Malardalen) کشور سوئد، به نشانی اینترنتی
http://www.mdh.se

۳۸- پایگاه اینترنتی دانشگاه ایالت کالیفرنیا، شهر برکْلی کشور آمریکا، به نشانی اینترنتی
https://hkn.eecs.berkeley.edu

۳۹- پایگاه اینترنتی دانشکده‌ی مهندسی نرم‌افزار کامپیوتر دانشگاه پادشاه سعودی کشور عربستان سعودی، به نشانی اینترنتی
http://faculty.ksu.edu.sa

۴۰- پایگاه اینترنتی آزمایشگاه هوش مصنوعی استنفورد، به نشانی اینترنتی
http://ai.stanford.edu

۴۱- کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش سوّم

۴۲- پایگاه اینترنتی دانشگاه ویسکانسین-مَدیسُون کشور آمریکا، به نشانی اینترنتی
http://www.wisc.edu

۴۳- پایگاه اینترنتی دانشگاه Western Australia کشور استرالیا، به نشانی اینترنتی
http://web.csse.uwa.edu.au

۴۴- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر دانشگاه کلمبیای کشور آمریکا، به نشانی اینترنتی
http://www.cs.columbia.edu

۱- *F*requently *A*sked *Q*uestions(*FAQ*)

۴۵- پایگاه اینترنتی جامعه‌ای از زبان‌آموزان کشور اسپانیا، به نشانی اینترنتی
http://quegrande.org

۴۶- پایگاه اینترنتی دانشکده‌ی فنّـآوری اطّلاعات و مهندسی دانشگاه شهر اُتاوای کشور کانادا، به نشانی اینترنتی
http://www.site.uottawa.ca

۴۷- پایگاه اینترنتی دانشکده‌ی کامپیوتر دانشگاه ایالت جُورْجیای کشور آمریکا، به نشانی اینترنتی
http://www.cc.gatech.edu

۴۸- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر و ریّاضیّات دانشگاه اِمُری کشور آمریکا، به نشانی اینترنتی
http://www.mathcs.emory.edu

۴۹- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر دانشگاه ایالت آیُوای کشور آمریکا، به نشانی اینترنتی
http://www.cs.iastate.edu

۵۰- پایگاه اینترنتی دانشکده‌ی انفورماتیک دانشگاه ماساریک کشور چک، به نشانی اینترنتی
http://www.fi.muni.cz

۵۱- پایگاه اینترنتی دانشگاه ایالت ویومینگ کشور آمریکا، به نشانی اینترنتی
http://jeffclune.com

۵۲- پایگاه اینترنتی دانشکده‌ی انفورماتیک و مخابرات دانشگاه شهر آتن کشور یونان، به نشانی اینترنتی
http://old.di.uoa.gr

۵۳- پایگاه اینترنتی دانشگاه ایالت کارُلینای شمالی کشور آمریکا، به نشانی اینترنتی
http://www.cs.unc.edu

۵۴- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر و مهندسی دانشگاه واشینگتن کشور آمریکا، به نشانی اینترنتی
http://courses.cs.washington.edu

۵۵- پایگاه اینترنتی دانشگاه کارلْتون کشور کانادا، به نشانی اینترنتی
http://www.carleton.edu

۵۶- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر دانشگاه ایالت ایلینویز کشور آمریکا، به نشانی اینترنتی
http://cs.illinois.edu

۵۷- پایگاه اینترنتی دانشگاه ایالت پُرتلَند کشور آمریکا، به نشانی اینترنتی
http://www.pdx.edu

۵۸- پایگاه اینترنتی دانشگاه شهر بابل کشور عراق، به نشانی اینترنتی
http://www.uobabylon.edu.iq

۵۹- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر دانشگاه ایالت ویرجینیای کشور ایالات متّحده‌ی آمریکا، به نشانی اینترنتی http://www.cs.virginia.edu

۶۰- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر و اطّلاعات دانشگاه پادشاه فَهَد کشور عربستان سعودی، به نشانی اینترنتی http://opencourseware.kfupm.edu.sa

۶۱- پایگاه اینترنتی دانشکده‌ی علوم کامپیوتر دانشگاه اُتاگوی کشور نیو زیلند، به نشانی اینترنتی http://www.cs.otago.ac.nz

۶۲- کتاب هوش مصنوعی آقایان، استوارت راسل و پیتر نورویگ، ویرایش دوّم

23th Edition - Year 2020

eBook

Artificial Intelligence

(In Persian) (Free)

Translated By:

Sohrab Jelvehgar

, Computer Software Engineer

Using:

English Sources On The Internet

sohjel