هوش مصنوعی

Artificial Intelligence

فصل اول - مقدمه

مراجع

- هوش مصنوعی : رهیافتی نوین (S.Russel & Peter Norvig)
  - مترجم : رامین رهنمون – آناهیتا هماوندی
- برنامه نویسی پرولوگ برای هوش مصنوعی

تعریف

- هوش : عبارت است از توانایی کسب، فهمیدن و بکارگیری دانش و یا توانایی تفکر و استدلال

- هوش مصنوعی عبارت است از مطالعه آنکه چگونه می توان کامپیوترها را وادار به انجام کارهایی کرد که در حال حاضر انسانها آنها را بهتر انجام می دهند.

- سیستمهایی که به طور منطقی (معقول) فکر می کنند.
- سیستمهایی که به طور منطقی (معقول) عمل می کنند.
- سیستمهایی که مثل انسان فکر می کنند.
- سیستمهایی که مثل انسان عمل می کنند.

تعریف هوش مصنوعی

پردازش‌های فکری و استدلالی

سیستم‌هایی که به طور منطقی فکر می‌کنند	سیستم‌هایی که مانند انسان فکر می‌کنند
سیستم‌هایی که به طور منطقی عمل می‌کنند	سیستم‌هایی که مانند انسان عمل می‌کنند

ایده‌آل هوشمندی

ارائه انسانی

تمرکز بر روی پردازش‌های رفتاری

تست تورینگ

- برای بررسی اینکه کامپیوتر هوشمند مثل انسان عمل می کند یا نه ؟
- توسط آلن تورینگ در سال ۱۹۵۰ پیشنهاد شد
- کامپیوتر هوشمند باید توسط یک مصاحبه گر مورد سوال و تحقیق قرار گیرد، در صورتیکه مصاحبه گر نتواند تشخیص دهد که مخاطب او کامپیوتر است یا انسان، آزمون موفقیت آمیز بوده
- کامپیوتر شرکت کننده در آزمون باید تواناییهای زیر را داشته باشد:
  - پردازش زبان طبیعی (NLP)
  - بازنمایی دانش (knowledge Representation) ذخیره سازی اطلاعات
  - استدلال خودکار (Automated reasoning) برای پاسخگویی با استفاده از اطلاعات ذخیره شده
  - یادگیری ماشین (Machine Learning) به منظور وفق پیدا کردن با شرایط جدید

کاربردهای هوش مصنوعی

- کاربردهای هوش مصنوعی
  - بازی ها
  - اثبات تئوری
  - پردازش زبان طبیعی (Natural language Processing)(NLP)
  - یادگیری ماشین (Machine Learning)
    - مثال : شبکه های عصبی (Neural Networks)
  - سیستمهای خبره (Expert Systems)
    - تشخیص پزشکی مانند MYCIN

کاربردهای هوش مصنوعی

- برخی از سیستم‌هایی موجود در جهان که از هوش مصنوعی استفاده می‌کنند

- HITECH
  - اولین برنامه کامپیوتری که موفق به شکست استاد بزرگ شطرنج جهان، آرنولد دنکر شده است.
- PEGASUS
  - یک برنامه درک گفتار که سؤالات کاربر را جواب می‌دهد و تمامی برنامه‌های مسافرتی شخص را با یک برنامه‌ریزی درست، مقرون به صرفه می‌کند.
- MARVEL
  - سیستم خبره‌ای که داده‌های ارسالی از سفینه فضایی را تحلیل نموده و در صورت بروز مشکلات جدی، پیغام هشدار به تحلیلگران می‌دهد.

یک تکنیک هوش مصنوعی چیست ؟

- مسائل بسیار متنوعی در هوش مصنوعی بررسی می شوند که وجه تشابه همه آنها **سخت** بودن آنهاست.
- هوش مصنوعی نیاز به **اطلاعات و دانش** دارد.
- خصایص ناخوش آیند دانش
  - حجیم است
  - به سختی می توان آن را به دقت وصف کرد
  - مرتبا تغییر می کند
- یک تکنیک هوش مصنوعی عبارت است از روشی که با دانش سر و کار دارد و این دانش باید قابل تعمیم، قابل تغییر و اصلاح بوده و در موارد زیادی قابل استفاده باشد، هرچند که این دانش کامل و دقیق نباشد.

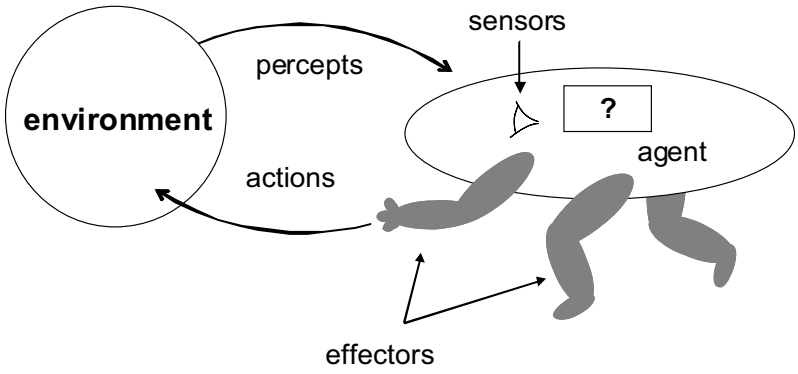
هوش مصنوعی

Artificial Intelligence

فصل دوم – عاملهای هوشمند

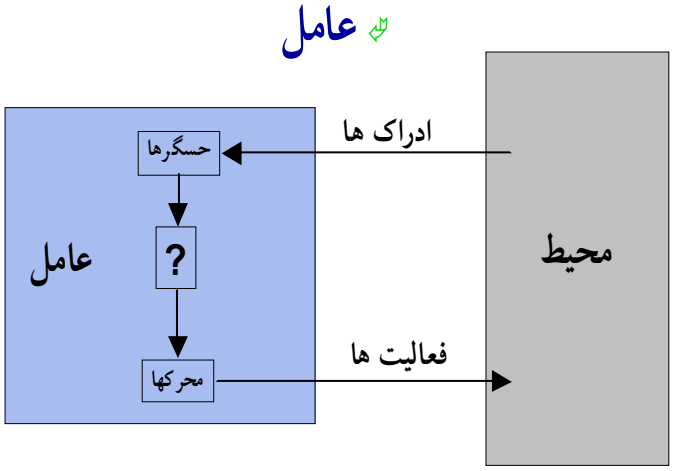
عامل(Agent)

- به هر چیزی اطلاق می‌شود، که قادر به درک محیط پیرامون خود از طریق حس‌گرها(sensor)و اثرگذاری بر روی محیط از طریق اثرکننده‌ها (effector) باشد.



عامل(Agent)

- مثال :
- عوامل انسانی
  - حس کردن: گوش، چشم
  - اثرگذاری: دست، پا
- عوامل روباتیک
  - حس کردن: دوربین، یابنده‌های مادون قرمز
  - اثرگذاری: موتور



مثال

محیط	اهداف	عملیات	ادراکات	نوع عامل
جاده، پیاده‌رو، ترافیک، مشتری	ایمنی، سرعت، قانونمندی، راحتی، افزایش سودمندی	راهنمایی کردن، شتاب‌دهنده، ترمز، صحبت با مسافر	دوربین‌ها، سرعت سنج، GPS، میکروفون	راننده تاکسی

منطقی بودن

- **دنباله ادراکی** (Percept Sequence)
  - تاریخچه مشاهداتی که عامل تا کنون مشاهده و درک کرده است.
- **معیار کارایی**
  - منطقی بودن نیاز به معیار کارایی دارد تا بگوییم یک وظیفه چقدر خوب انجام شده است. (سرعت، تاثیر بر محیط و ...)
- یک عامل منطقی ایده آل، باید برای هر دنباله ادراکی ممکن عملی انجام دهد که باعث حداکثرسازی معیار کارایی شود. این عمل بر اساس موارد زیر صورت می گیرد.
  - دنباله ادراکی
  - دانش درونی
- **تابع عامل**
  - رفتار عامل توسط تابع عامل توصیف میشود که هر دنباله ادراک را به یک فعالیت نقش میکند.

خودمختاری (Autonomy)

- اگر رفتار عامل بدون توجه به دنباله ادراکی و فقط بر اساس دانش درونی باشد، گوییم عامل فاقد خود مختاری است.
- بنابراین اگر رفتار یک عامل بر اساس تصمیمات طراحش (دانش اولیه ذخیره شده) صورت گیرد خود مختار نخواهد بود.
- شرط دوام و بقای عامل ها
  - دانش اولیه وکافی
  - قابلیت یادگیری

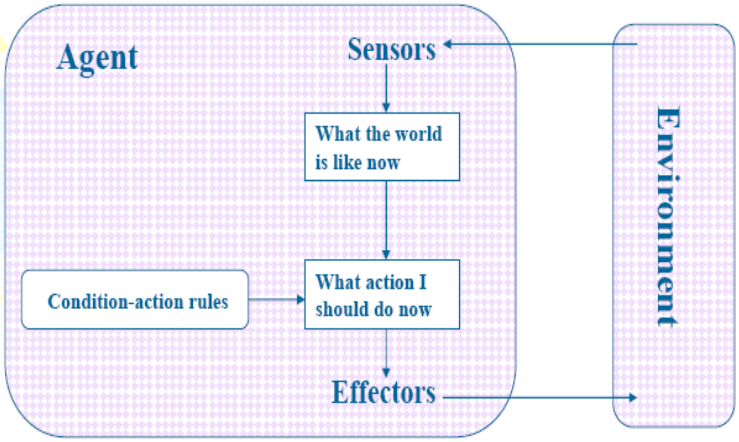
انواع عامل

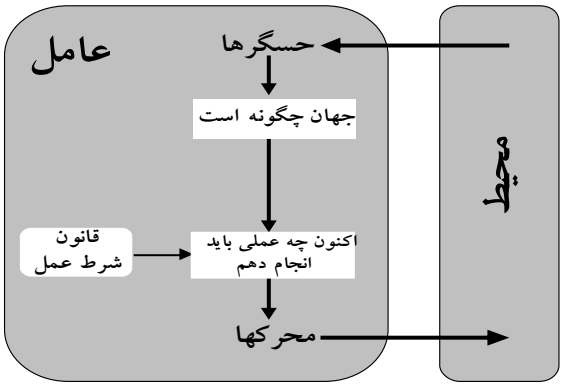
- **عاملهای واکنشی ساده (Simple Reflex Agent)**
- **عاملهایی که وضعیت دنیا را حفظ می کنند(Agents With Memory)**
- **عاملهای هدف گرا(Agents With Goals)**
- **عاملهای سودمند(Utility Based Agents)**

عاملهای واکنشی ساده

- این عاملها فعالیت را بر اساس **درک فعلی** و بدون در نظر گرفتن سابقه ادراک، انتخاب میکند
- به خاطر **حذف سابقه ادراک** برنامه عامل در مقایسه با جدول آن **بسیار کوچک** است
- انتخاب فعالیت بر اساس یکسری **قوانین موقعیت شرطی** انجام میشود
- برای مثال در طراحی سیستم هوشمند راننده تاکسی اگر ماشین جلویی ترمز کند و چراغ های ترمز آن روشن شود، راننده باید اقدام به ترمز کند.
- معایب
  - ایجاد و ذخیره آنها بدلیل حجم بسیار زیاد جدول جستجو (مثلا شطرنج دارای ۱۰ به توان ۱۲۰ حالت می باشد)
  - اگر در محیط تغییری ایجاد شود بایستی جدول جستجو از اول طراحی شود.

عامل واکنشی ساده

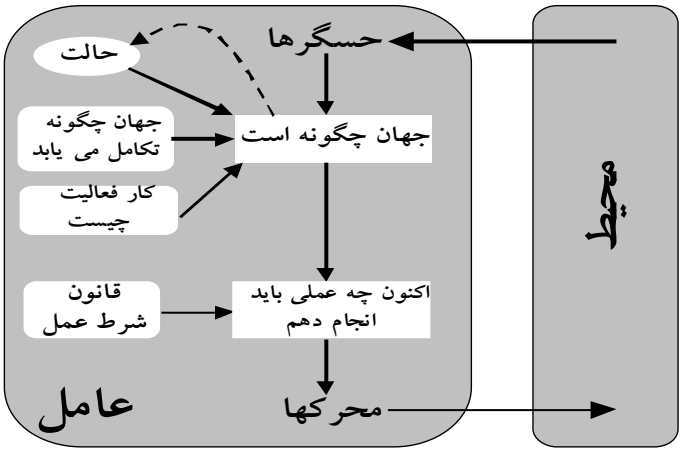


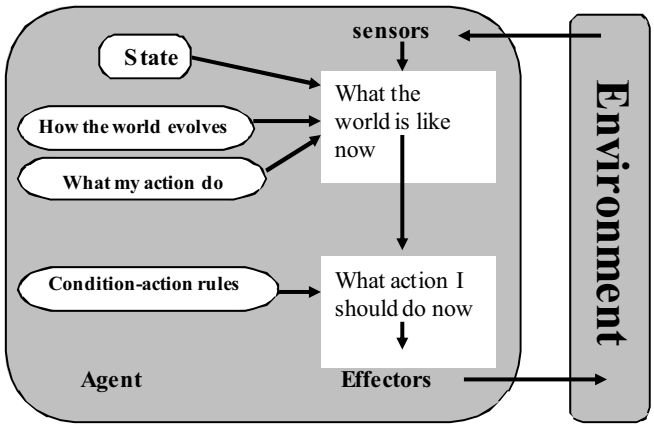


عاملهایی که وضعیت دنیا را حفظ می کنند

- از آنجایی ناشی می‌شود که حسگرها نمی‌توانند دسترسی کامل به وضعیت دنیا را به وجود آورند.
- بعنوان مثال چشمک زدن یک چراغ فقط با یک تصویر قابل تشخیص نمی باشد، بنابراین نیازمند نگهداری برخی از حالتهای قبلی می باشد.
- در چنین شرایطی، عامل ممکن است نیازمند دستکاری برخی اطلاعات وضعیت داخلی باشد تا از طریق آن تمایز بین وضعیت‌های دنیا که در ظاهر ورودی ادراکی یکسانی می‌کنند ولی در واقع معنی کاملاً متفاوتی دارند را میسر سازد.

عاملهایی که وضعیت دنیا را حفظ می کنند





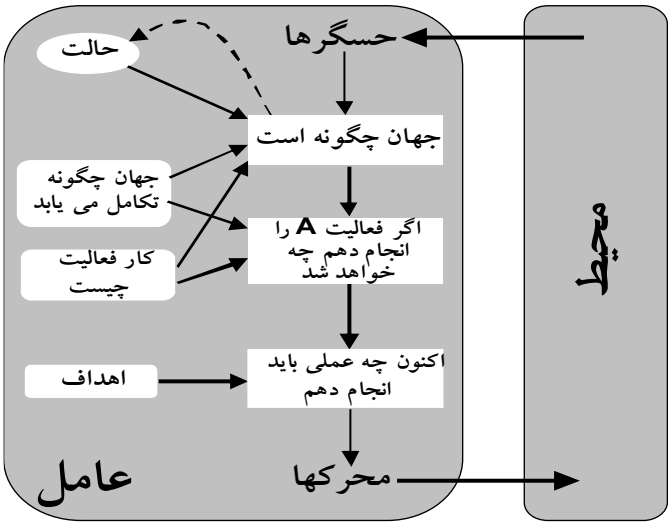
عامل های هدف گرا

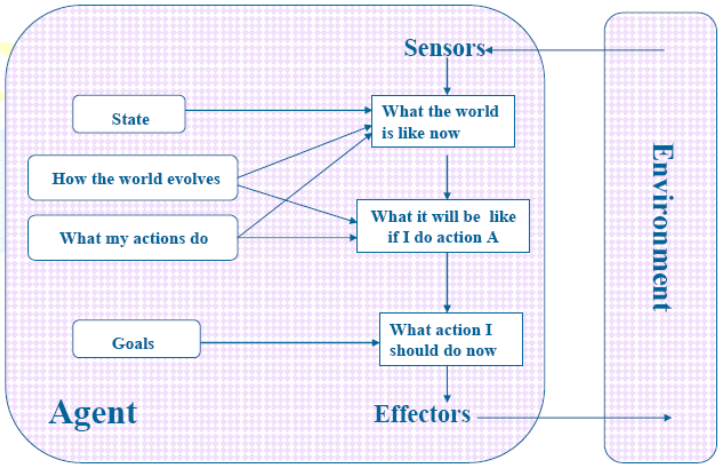
21

- این عامل علاوه بر توصیف حالت فعلی، برای انتخاب موقعیت مطلوب نیازمند اطلاعات هدف نیز میباشد
- اعمال را به گونه ای انتخاب می کنند که به هدف خاصی برسند
- تنها پی گیری حالت فعلی کافی نیست و عامل نیاز به اضافه کردن اهداف، برای تصمیم گیری دارد.
- این نوع تصمیم گیری همواره آینده را در نظر دارد و با قوانین شرط – عمل تفاوت دارد
- بسیار انعطاف پذیر است، بعنوان مثال در مورد مثال راننده تاکسی اگر فقط از عامل واکنشی ساده استفاده شود با تغییر مقصد مسافر، بایستی تعداد زیادی از قوانین شرط – عمل اصلاح شود. در صورتیکه در عامل هدف گرا با تعیین یک هدف تازه، رفتار تازه ای مشاهده خواهد شد.

عامل های هدف گرا

22

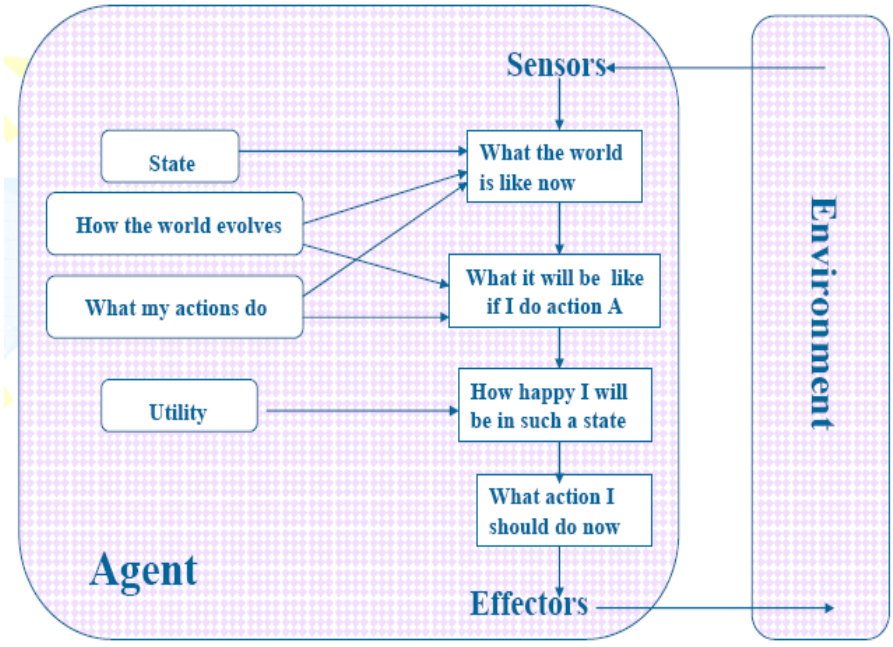




عاملهای سودمند

- وقتی چندین انتخاب ممکن وجود دارد. چگونه تصمیم می گیریم که کدامیک بهترین انتخاب است؟
- یک هدف تفاوت میان وضعیت مطلوب و نامطلوب را به صورت خام مشخص می کند ولی اغلب تابع کارایی عمومی تری نیاز داریم که درجه مطلوب بودن را معین می کند.
- تابع سودمندی $U: States \rightarrow R$
- **تابع سودمندی**، حالت یا دنباله ای از حالتها را به یک عدد **حقیقی** نگاشت میکند که **درجه رضایت** را توصیف مِیکند.
- امکان تصمیم گیری میان اهداف متناقض (مثل سرعت و امنیت) را فراهم می کند
- راهی را انتخاب می کند که بین درجه اهمیت هدف و احتمال رسیدن به هدف توزان برقرار شود. (در صورتیکه چندین هدف وجود داشته باشد و احتمال رسیدن به هیچ یک از آنها قطعی نباشد)

عاملهای سودمند



انواع محیط ها

- قابل دسترسی / غیر قابل دسترسی
  - محیطی که عامل آن توسط ابزار حس‌کننده‌اش امکان دسترسی به وضعیت کامل محیط را داشته باشد. (قابل دسترس)
- قطعی / غیر قطعی
  - یک محیط قطعی است اگر حالت بعدی آن کاملا بوسیله حالت فعلی و عمل عامل مشخص شود.
- اپیزودیک / غیراپیزودیک (تقسیم پذیر / تقسیم ناپذیر)
  - محیط اپیزودیک (episodic)، تجربه عامل به اپیزودهایی تقسیم می‌گردد.
  - اعمالی که در یک بخش انجام می گیرد هیچ تاثیری بر بخشهای دیگر نمی گذارد
  - درچنین محیط هایی عامل نیازمند برنامه ریزی برای آینده نمی باشد.

انواع محیط ها

- ایستا / پویا
  - محیط ایستا در حین تصمیم گیری عامل در مورد عمل خاصی تغییر نمی کند.
  - اگر محیط با گذر زمان تغییر نکندولی معیار کارایی تفاوت کند، محیط نیمه پویا نامیده می شود.
  - **محیط‌های ایستا برای کار ساده هستند زیرا عامل نیاز به نگاه‌کردن به دنیا در حین تصمیم‌گیری عملی نداشته و همچنین در مورد گذر زمان نیز نگران نمی‌باشد.**
- گسسته / پیوسته
  - اگر در محیطی تعداد ادراکات و اعمال مجزا محدود باشد، محیط گسسته است
  - بازی شطرنج گسسته است
  - رانندگی تاکسی پیوسته است
- سخت‌ترین حالت در بین حالات موجود برای محیط:
- **غیر قابل دسترسی، غیرقطعی، غیر اپیزودیک، پویا و پیوسته**

انوع محیط به همراه خصوصیات انها

محیط	قابل دسترسی	قطعی	اپیزودیک	ایستا	گسسته
شطرنج به همراه ساعت					

انوع محیط به همراه خصوصیات انها

محیط	قابل دسترسی	قطعی	اپیزودیک	ایستا	گسسته
شطرنج به همراه ساعت	YES	YES	NO	Semi	YES
شطرنج بدون ساعت					

انوع محیط به همراه خصوصیات انها

29

محیط	قابل دسترسی	قطعی	اپیزودیک	ایستا	گسسته
شطرنج به همراه ساعت	YES	YES	NO	Semi	YES
شطرنج بدون ساعت	YES	YES	NO	YES	YES
راندن تاکسی					

انوع محیط به همراه خصوصیات انها

30

محیط	قابل دسترسی	قطعی	اپیزودیک	ایستا	گسسته
شطرنج به همراه ساعت	YES	YES	NO	Semi	YES
شطرنج بدون ساعت	YES	YES	NO	YES	YES
راندن تاکسی	NO	NO	NO	NO	NO
سیستم تشخیص پزشکی					

انوع محیط به همراه خصوصیات انها

محیط	قابل دسترسی	قطعی	اپیزودیک	ایستا	گسسته
شطرنج به همراه ساعت	YES	YES	NO	Semi	YES
شطرنج بدون ساعت	YES	YES	NO	YES	YES
راندن تاکسی	NO	NO	NO	NO	NO
سیستم تشخیص پزشکی	NO	NO	NO	NO	NO
سیستم تحلیل تصویر					

انوع محیط به همراه خصوصیات انها

محیط	قابل دسترسی	قطعی	اپیزودیک	ایستا	گسسته
شطرنج به همراه ساعت	YES	YES	NO	Semi	YES
شطرنج بدون ساعت	YES	YES	NO	YES	YES
راندن تاکسی	NO	NO	NO	NO	NO
سیستم تشخیص پزشکی	NO	NO	NO	NO	NO
سیستم تحلیل تصویر	YES	YES	YES	Semi	NO
کنترل‌کننده پالایشگاه					

انوع محیط به همراه خصوصیات انها

محیط	قابل دسترسی	قطعی	اپیزودیک	ایستا	گسسته
شطرنج به همراه ساعت	YES	YES	NO	Semi	YES
شطرنج بدون ساعت	YES	YES	NO	YES	YES
راندن تاکسی	NO	NO	NO	NO	NO
سیستم تشخیص پزشکی	NO	NO	NO	NO	NO
سیستم تحلیل تصویر	YES	YES	YES	Semi	NO
کنترل‌کننده پالایشگاه	NO	NO	NO	NO	NO

هوش مصنوعی

Artificial Intelligence

فصل سوم - حل مساله توسط جستجو

فهرست

عامل‌های حل مسئله

- یک نوع عامل هدفگرا، عامل حل مسئله نامیده می‌شود.

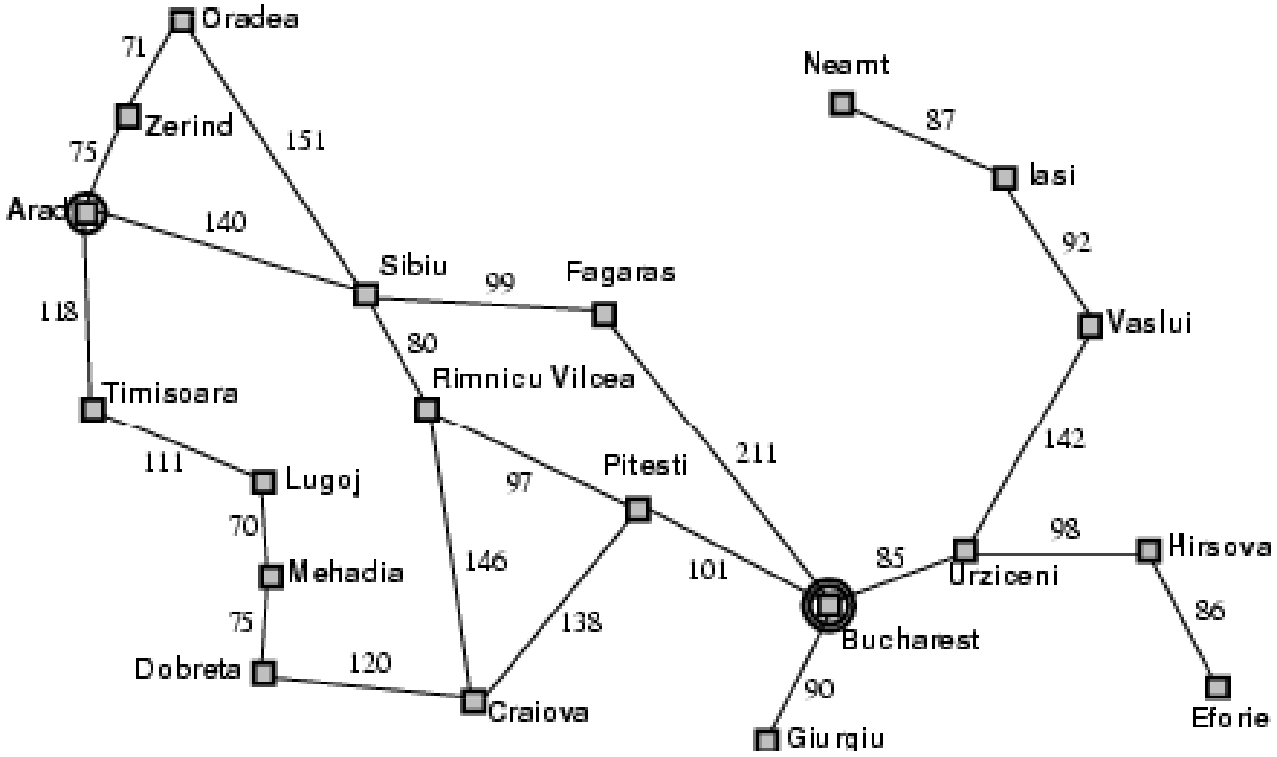
- عامل‌های حل مسئله توسط یافتن ترتیب عملیات تصمیم می‌گیرند که چه انجام دهند تا آنها را به حالت‌های مطلوب سوق دهد.
- الگوریتم جستجو مسئله‌ای را به عنوان ورودی دریافت نموده و راه‌حلی را به صورت دنباله عملیات بر می‌گرداند.

عاملهای حل مسئله

- چهار گام اساسی برای حل مسائل

- فرموله کردن هدف: وضعیتهای مطلوب نهایی کدامند؟
- فرموله کردن مسئله: چه فعالیتها و وضعیتهایی برای رسیدن به هدف موجود است؟
- جستجو: انتخاب بهترین دنباله از فعالیتهایی که منجر به حالاتی با مقدار شناخته شده میشود.
- اجرا: وقتی دنباله فعالیت مطلوب پیدا شد، فعالیتهای پیشنهادی آن میتواند اجرا شود.

مثال: نقشه رومانی



مثال: نقشه رومانی

- صورت مسأله: رفتن از آراد به بخارست
- فرموله کردن هدف: رسیدن به بخارست
- فرموله کردن مسئله:
  - وضعیتها: شهرهای مختلف
  - فعالیتها: حرکت بین شهرها
- جستجو: دنباله ای از شهرها مثل:آراد، سیبیو، فاگارس، بخارست
- این جستجو با توجه به کم هزینه ترین مسیر انتخاب میشود

برای تعریف یک مسئله موارد زیر نیاز داریم:

- حالت اولیه (initial state): حالتی که عامل از آن شروع میکند.
  - در مثال رومانی: شهر آراد (Arad)
- مجموعه‌ای از عملیات ممکن(Action)، که برای عامل قابل دسترسی باشد.
- فضای حالت: مجموعه ای از حالتها که از حالت اولیه میتوان به آنها رسید.
  - در مثال رومانی: کلیه شهرها که با شروع از آراد میتوان به آنها رسید
- آزمون هدف: تعیین میکند که آیا حالت خاصی، حالت هدف است یا خیر
  - هدف صریح: در مثال رومانی، رسیدن به بخارست
  - هدف انتزاعی: در مثال شطرنج، رسیدن به حالت کیش و مات

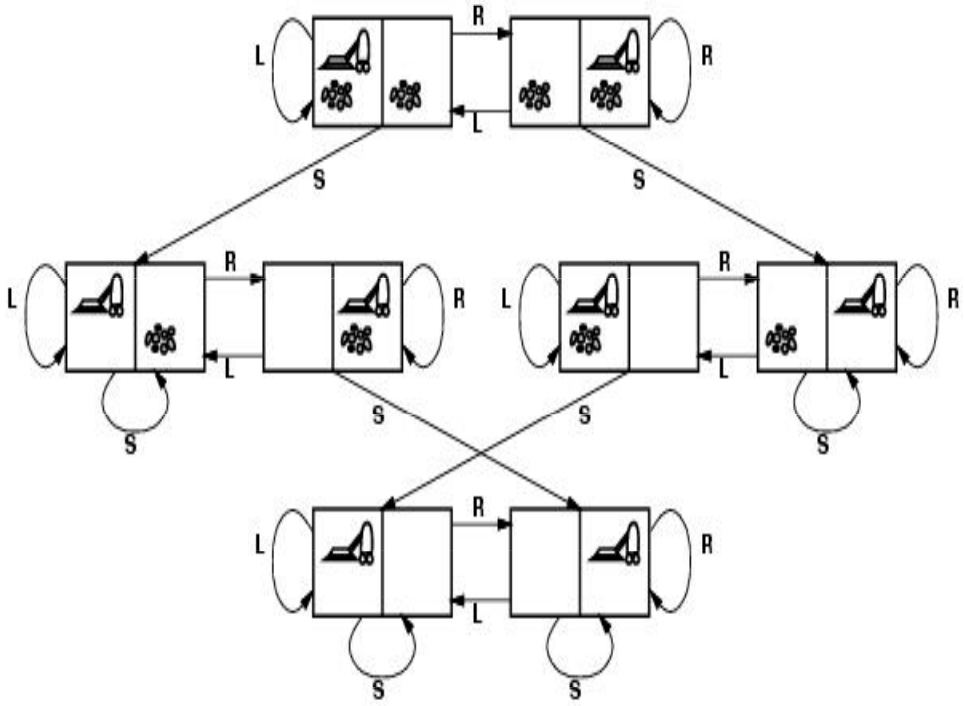
برای تعریف یک مسئله موارد زیر نیاز داریم:

- مسیر
  - در مثال رومانی: Arad, Sibiu, Fagarasیک مسیر است

- هزینه مسیر: برای هر مسیر یک هزینه عددی مثبتی در نظر میگیرد.
  - در مثال رومانی: طول مسیر بین شهرها بر حسب کیلومتر

- راه حل مسئله (solution) مسیری از حالت اولیه به حالت هدف است
  - راه حل بهینه کمترین هزینه مسیر را دارد

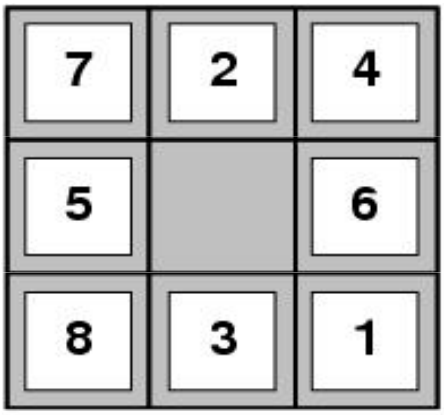
مثال: دنیای جارو برقی



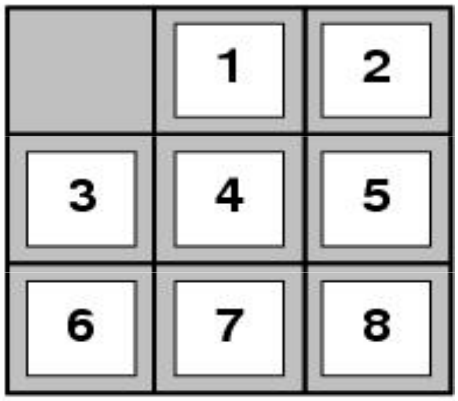
مثال : دنیای جارو برقی

- حالتها:
  - دو مکان که هر یک ممکن است کثیف یا تمیز باشند
  - و در هر مکان ممکن است جارو برقی(عامل) حاضر باشد یا نباشد.
  - لذا ۸ = ۲^۲ *۲حالت در این جهان وجود دارد
- حالت اولیه: هر حالتی میتواند به عنوان حالت اولیه طراحی شود
- سه عملیات: راست، چپ، مکش
- آزمون هدف: تمیزی هر دو مکان
- هزینه مسیر: تعداد مراحل در مسیر

معمای ۸



Start State



Goal State

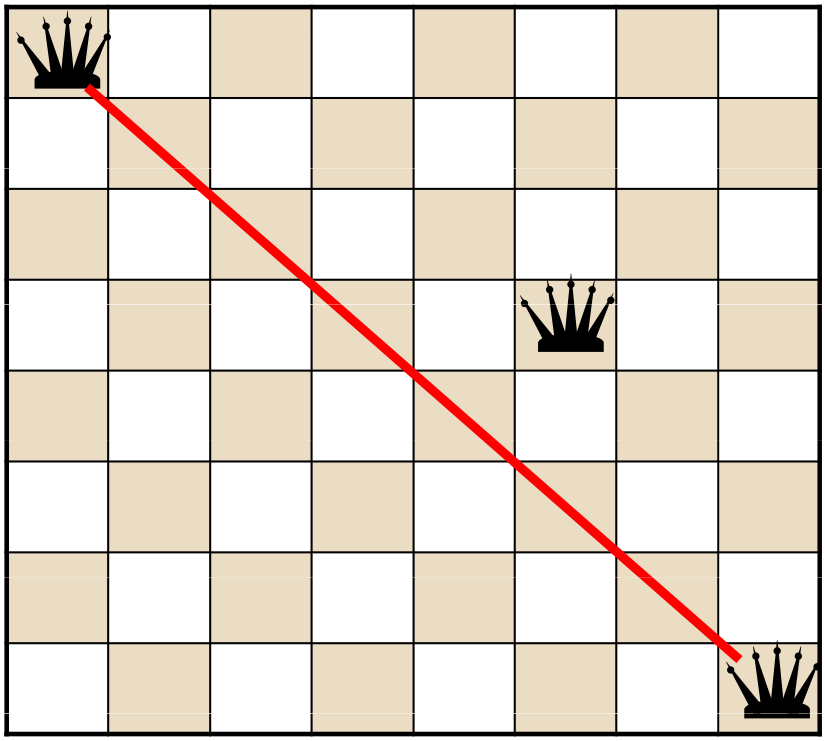
معمای ۸

- **حالت‌ها:** توصیف وضعیت مکان هر ۸ مربع را در یکی از ۹ خانهٔ صفحه مشخص می‌کند. برای کارایی بیشتر، بهتر است که فضاهای خالی نیز ذکر شود.
- **عملگرها:** فضای خالی به چپ، راست، بالا و پائین حرکت کند.
- **آزمون هدف:** وضعیت با ساختار هدف مطابقت می‌کند.
- **هزینه مسیر:** هر قدم ارزش ۱ دارد، بنابراین هزینه مسیر همان طول مسیر است.

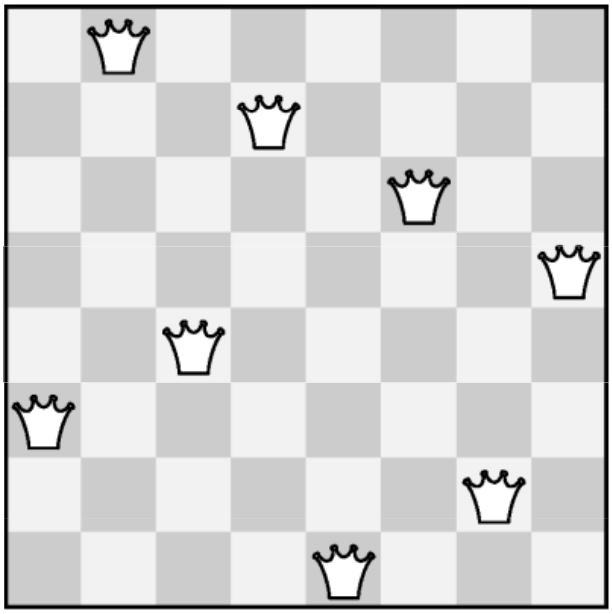
مسئله ۸ وزیر

- هدف از مسئله ۸ وزیر، قرار دادن ۸ وزیر بر روی صفحه شطرنج به صورتی است که هیچ وزیری نتواند به دیگری حمله کند.
- دو نوع بیان ریاضی اصلی وجود دارد بیان افزایشی که با جایگزینی وزیرها، به صورت یکی یکی کار می‌کند و دیگری بیان وضعیت کامل که با تمام ۸ وزیر روی صفحه شروع می‌کند و آنها را حرکت می‌دهد.

مسئله ۸ وزیر



مسئله ۸ وزیر



مسئله کشیش‌ها و آدمخوارها

- سه کشیش و سه آدم خوار در یک طرف رودخانه قرار دارند و هم چنین قایقی که قادر است یک یا دو نفر را حمل کند.
  راهی را بیابید که هر نفر به سمت دیگر رودخانه برود، بدون آنکه تعداد کشیش‌ها در یکجا کمتر از آدم خوارها شود.

مسئله کشیش‌ها و آدمخوارها

- **حالات:** یک حالت شامل یک دنبالهٔ مرتب شده از عدد است که تعداد کشیش‌ها، تعداد آدمخوارها و محل قایق در ساحلی از رودخانه که از آنجا مسئله شروع شده را نمایش می‌دهد.
- **عملگرها:** از هر حالت، عملگرهای ممکن یک کشیش، یک آدمخوار، دو کشیش، دو آدمخوار، یا یکی از هر کدام را در قایق جا می‌دهند.
- **آزمون هدف:** رسیدن به حالت(۰و ۰ و ۰).
- **هزینه مسیر:** تعداد دفعات عبور از رودخانه.

اندازه گیری کارایی حل مسئله

- کامل بودن: آیا الگوریتم تضمین میکند که در صورت وجود راه حل، آن را بیابد؟
- بهینگی: آیا این راهبرد، راه حل بهینه ای را ارائه میکند.
- پیچیدگی زمانی: چقدر طول میکشد تا راه حل را پیدا کند؟
  - تعداد گره های تولید شده در اثنای جستجو
- پیچیدگی فضا: برای جستجو چقدر حافظه نیاز دارد؟
  - حداکثر تعداد گره های ذخیره شده در حافظه

جستجوی ناآگاهانه

- ناآگاهی این است که الگوریتم هیچ اطلاعاتی غیر از تعریف مسئله در اختیار ندارد
- این الگوریتمها فقط میتواند هدف را از غیر هدف تشخیص دهند
- راهبردهایی که تشخیص میدهد یک حالت غیر هدف نسبت به گره غیر هدف دیگر، امید بخش تر است، جست و جوی آگاهانه یا جست و جوی اکتشافی نامیده میشود.

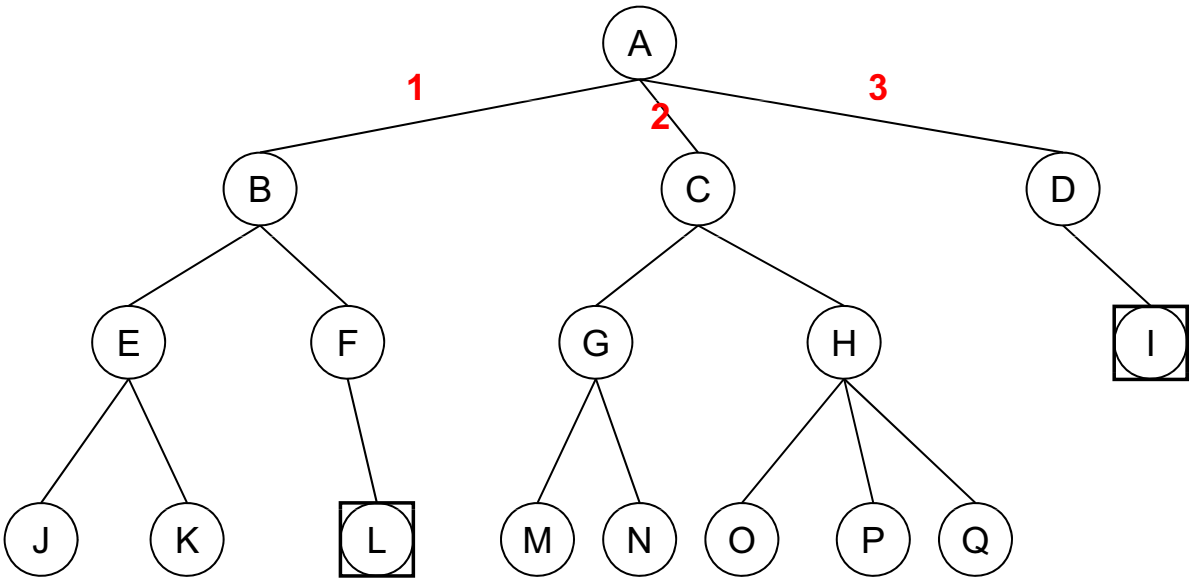
انواع استراتژیهای جستجو

- جست و جوی عرضی (سطحی)
- جست و جوی عمقی
- جست و جوی عمیق کننده تکراری
- جست و جوی هزینه یکنواخت
- جست و جوی عمقی محدود
- جست و جوی دو طرفه

جست وجوی سطحی (عرضی)(BFS)(Breadth First Search)

- در این استراتژی که بسیار ، ابتدا گره ریشه، و سپس تمام گره‌های دیگر گسترش داده می‌شوند.
- به عبارت کلی‌تر، تمام گره‌های عمق d، قبل از گره‌های عمق d+1 گسترش داده می‌شوند.
- گرههای جدید برای هر سطح در انتهای یک صف قرار داده می شوند و به ترتیب گسترش داده خواهند شد.
- اگر چندین راه حل وجود داشته باشد، BFS کم عمق ترین مسیر را پیدا می کند.

جست وجوی سطحی (عرضی)(BFS)(Breadth First Search)



جست وجوی سطحی (عرضی)(BFS)(Breadth First Search)

- **فاکتور انشعاب(Branching Factor) (b)**
  - **حداکثر چند انشعاب می تواند از یک گره به وجود آید.**
- **تعداد گرههای بسط داده شده قبل از یافتن راه حل با عمق d برابر است با :**
- $1 + b + b^2 + b^3 + \ldots + b^d \in O(b^d)$
- **مزایا:**
- جستجوی سطحی، کامل و بهینه می‌باشد زیرا هزینه مسیر، یک تابع غیر نزولی از عمق گره است و این استراتژی اولین جواب در نزدیک ترین سطح را خواهد یافت.
- **معایب:**
- مرتبه زمانی نمایی است. چراکه پیچیدگی زمانی $O(b^d)$
- نیاز به حافظه زیاد چراکه پیچدگی فضایی $O(b^d)$

جست وجوی سطحی (عرضی)(BFS)(Breadth First Search)

- مثال :
- b=10 و فضای مورد نیاز برای هر گره 100 بایت و در هر ثانیه ۱۰۰۰ گره تولید میشود.

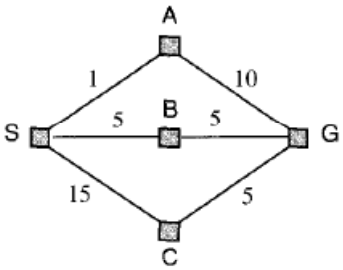
depth	nodes	Time	Memory
0	1	1 ms	100 byte
2	111	0.1 s	11 KB
4	11,111	11 s	1 MB
6	10^6	18 min	111 MB
8	10^8	31 hours	11 GB
…	…	…	…

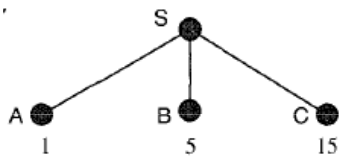
جستجو با هزینه یکسان (Uniform Cost Search)

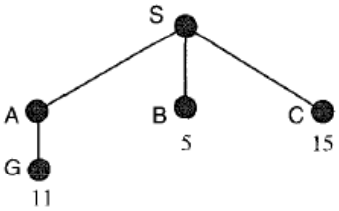
- جستجوی سطحی کم عمق ترین هدف را می باید که لزوما کم هزینه ترین هدف نیست.
- جستجوی با هزینه یکسان روش قبل را با یافتن کم هزینه ترین هدف اصلاح می کند.
- در این روش همواره گرهی بسط داده می شود که کم هزینه ترین مسیر را داشته است.
- در این استراتژی، در شرایط عمومی، اولین راه حل، ارزان‌ترین راه نیز هست.
- اگر هزینه مسیر توسط تابع g(n) اندازه‌گیری شود، اگر حالت زیر برقرار باشدجستجوی سطحی همان جستجوی با هزینه یکنواخت خواهد بود :
- (یعنی اگر هزینه همه قدم ها مساوی باشد)

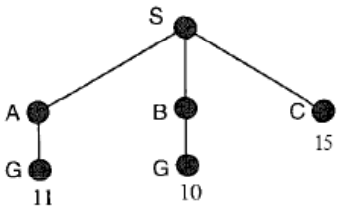
( ) DEPTH( )

جستجو با هزینه یکسان (Uniform Cost Search)









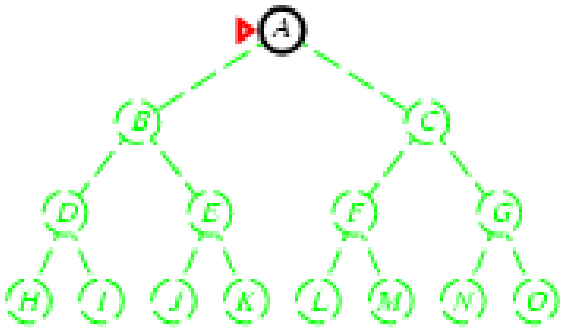
جستجوی عمقی (Depth-first search)

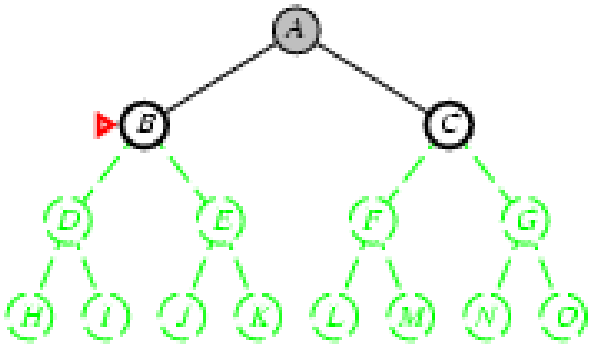
- این استراتژی، یکی از گره‌ها را در پائین‌ترین سطح درخت بسط می‌دهد؛ اما اگر به نتیجه نرسید، به سراغ گره‌هایی در سطوح کم عمق‌تر می‌رود.
- مزایا:
  - این جستجو، نیاز به حافظه نسبتاً کمی فقط برای ذخیره مسیر واحدی از ریشه به یک گره برگی داشته، و گره‌های باقی‌مانده بسط داده نشده نیاز به حافظه ندارد.
- پیچیدگی زمانی $O(b^m)$ می‌باشد. به طوریکه b فاکتور انشعاب فضای حالت، و m حداکثر عمق درخت باشد.
- معایب:
  - جستجوی عمقی نه کامل و نه بهینه است.
  - در درخت‌های با عمق نامحدود و بزرگ این استراتژی کار نمی‌کند.

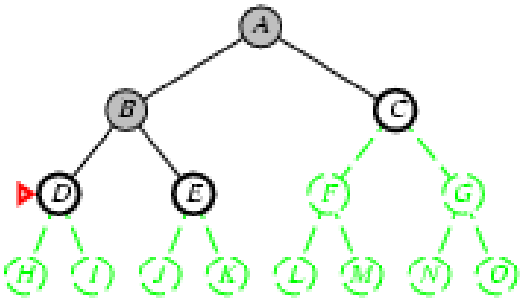
جستجوی عمقی (Depth-first search)

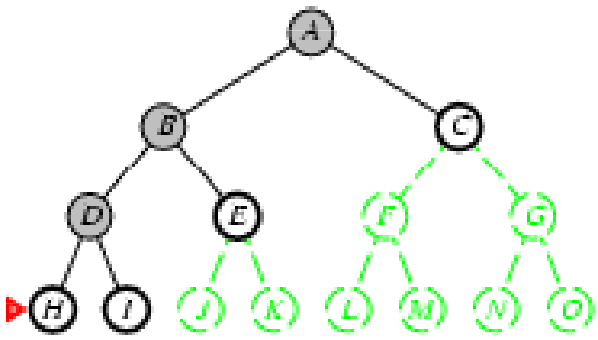
- کامل بودن: خیر
  - اگر زیر درخت چپ عمق نامحدود داشت و فاقد هر گونه راه حل باشد، جستجو هرگز خاتمه نمی یابد.
- بهینگی: خیر
- پیچیدگی زمانی: $O(b^m)$
- پیچیدگی فضا: $O(bm)$
- اگر مقدار m خیلی بزرگتر از d باشد این روش عملکرد بسایر بدی خواهد داشت.
- اگر اهداف با تراکم زیاد در درخت تحت جستجو پراکنده شده باشند، این روش از از جستجوی سطحی بهتر عمل می کند
- پیاده سازی این روش با استفاده از پشته می باشد (LIFO)

جستجوی عمقی (Depth-first search)

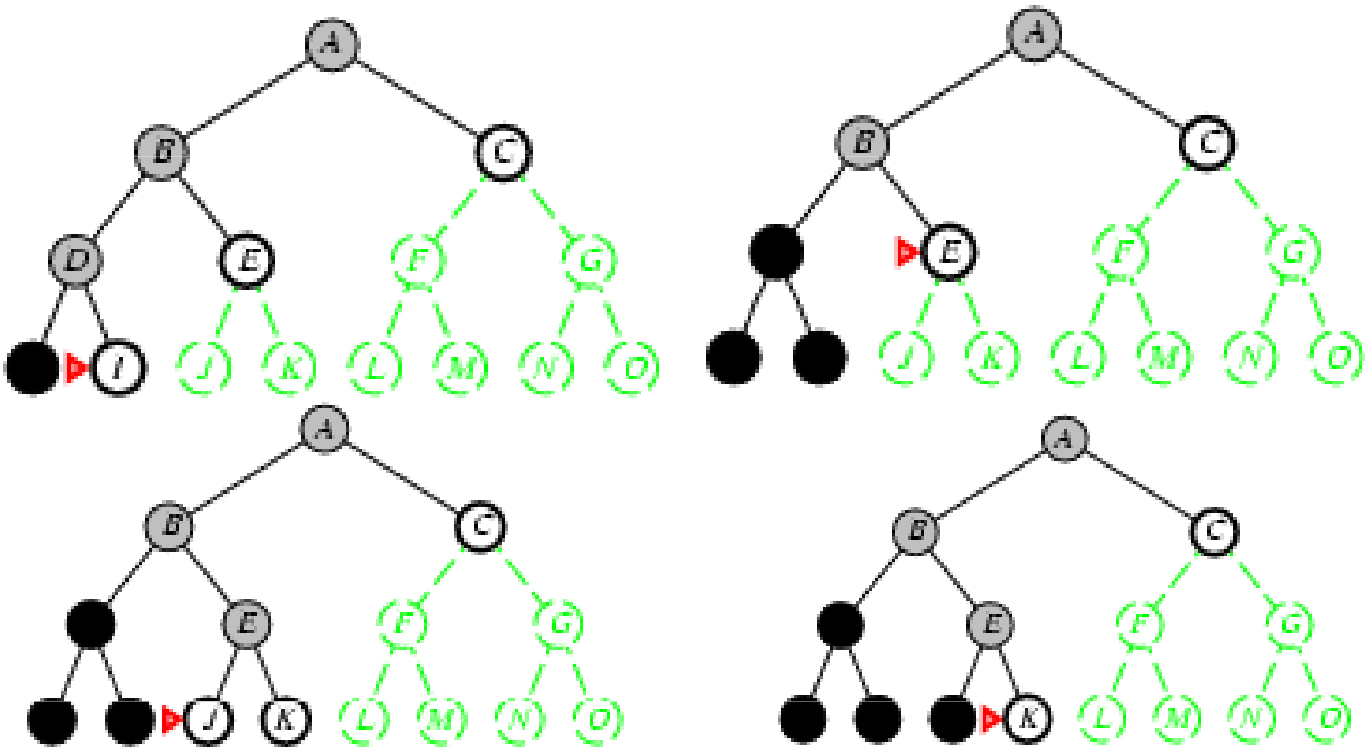




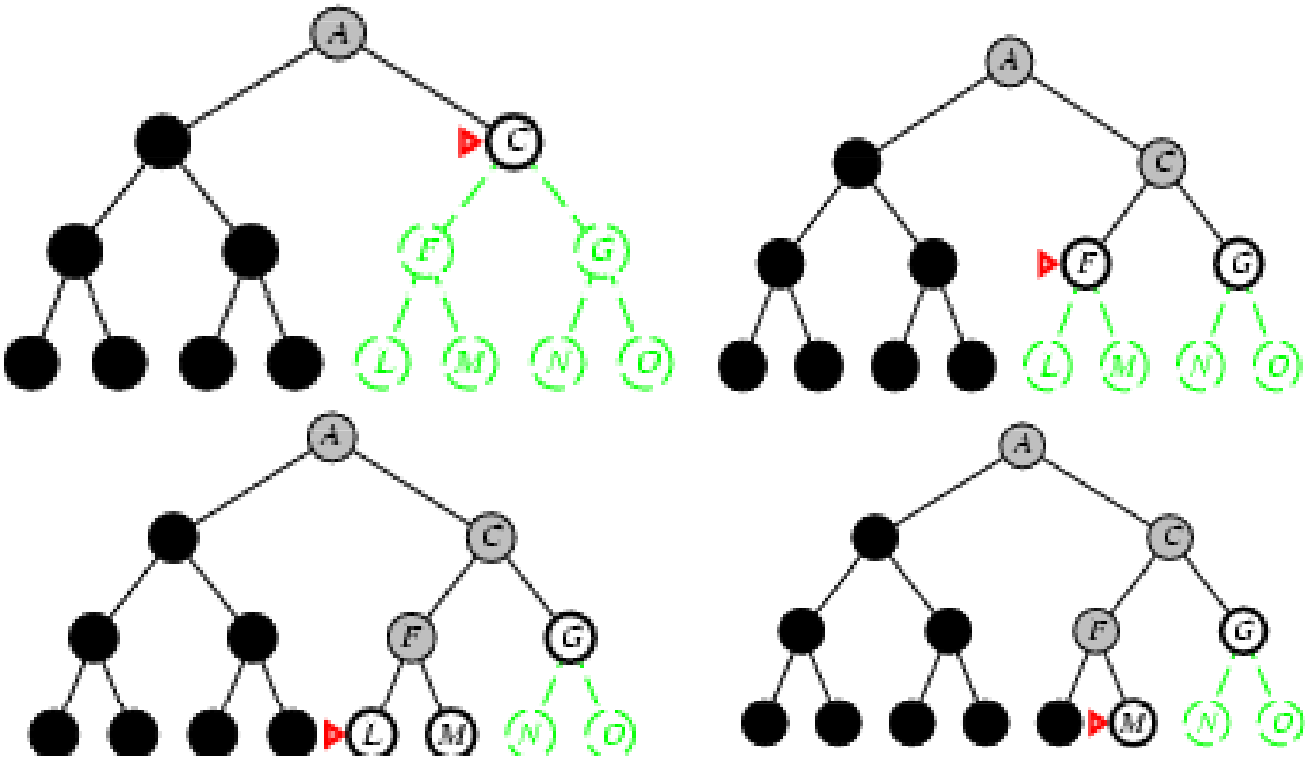




جستجوی عمقی (Depth-first search)



جستجوی عمقی (Depth-first search)



جستجوی عمقی محدود شده

- مسئله درختهای نامحدود میتواند به وسیله جست و جوی عمقی با عمق محدود L بهبود یابد
- بعنوان مثال در مسئله نقشه رومانی چون می دانیم حداکثر ۲۰ شهر وجود دارد؛ پس جواب حداکثر طول ۱۹ خواهد داشت و می توان L را ۱۹ تعیین کرد.
- جستجوی عمقی محدود شده کامل است اما بهینه نیست.
  - اگر $L<d$ و سطحی ترین هدف در خارج از عمق محدود قرار داشته باشد، این راهبرد کامل نخواهد بود.
  - اگر $L>d$ انتخاب شود، این راهبرد بهینه نخواهد بود.
- هزینه زمانی : $O(b^L)$
- هزینه فضایی : $O(bL)$

جستجوی عمیق کننده تکراری

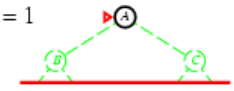
- قسمت دشوار جستجوی عمقی محدود شده، انتخاب یک محدودهٔ خوب است.
- برای بیشتر مسائل، محدودهٔ عمقی مناسب را تا زمانی که مسئله حل نشده است، نمی‌شناسیم.
- جستجوی عمیق‌کنندهٔ تکراری استراتژی است که نظریه انتخاب بهترین محدودهٔ عمقی، توسط امتحان تمام محدودهٔ مسیرهای ممکن را یادآوری می‌کند

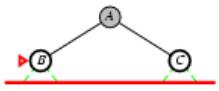
جستجوی عمیق کننده تکراری

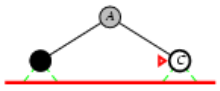
Limit = 0

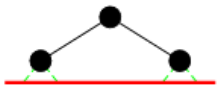
جستجوی عمیق کننده تکراری

Limit = 1



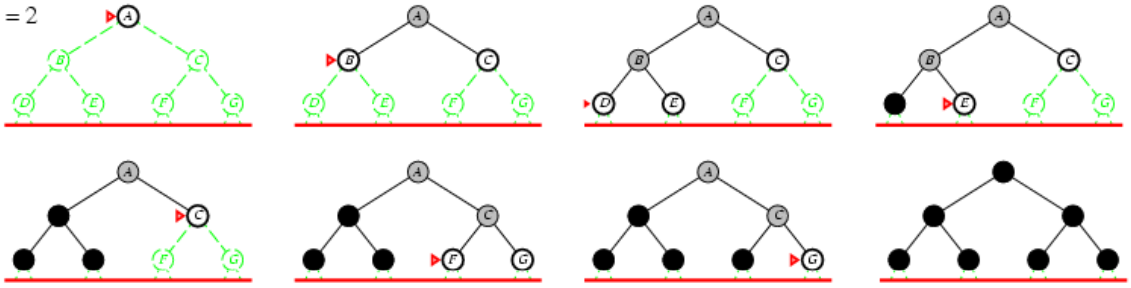






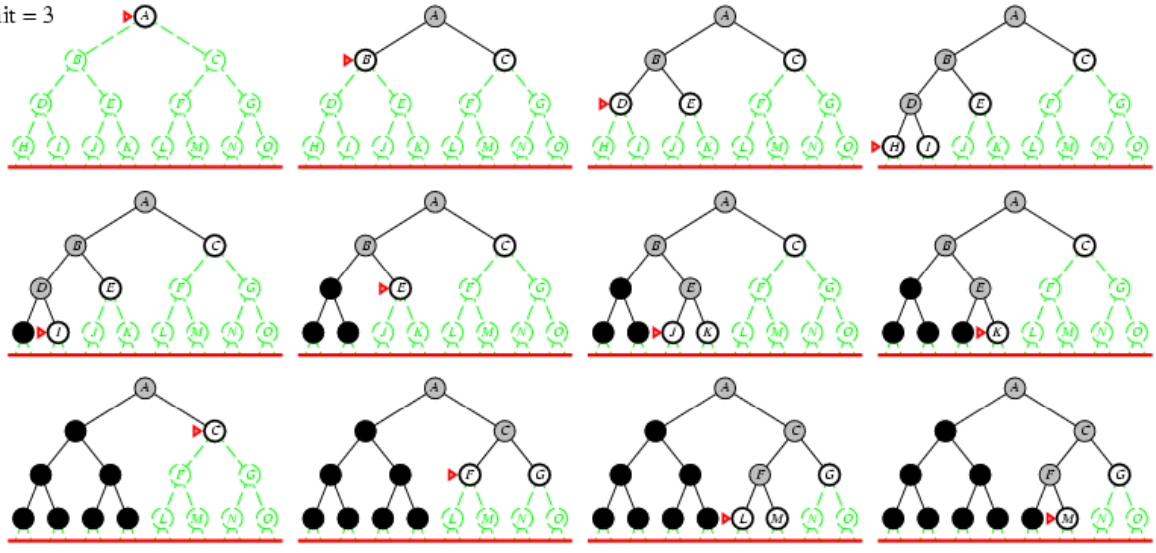
جستجوی عمیق کننده تکراری

Limit = 2



جستجوی عمیق کننده تکراری

Limit = 3



جستجوی عمیق کننده تکراری

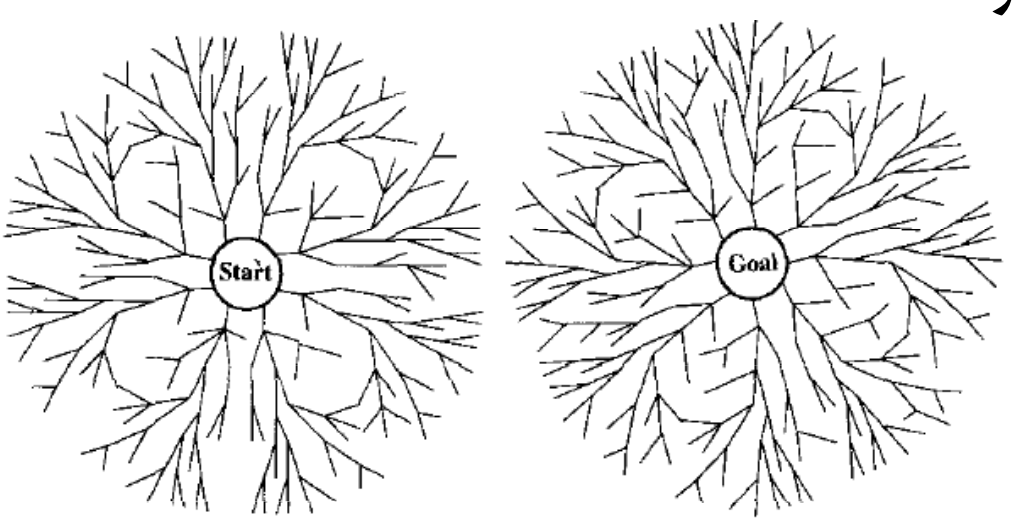
- **مزایا:**
- ترکیبی از مزایای جستجوی سطحی و عمقی را دارد.
- این جستجو مانند جستجوی سطحی کامل و بهینه است، اما فقط مزیت درخواست حافظه اندک را از جستجوی عمقی دارد.
- مرتبه بسط حالات مشابه جستجوی سطحی است، به جز اینکه بعضی حالات چند بار بسط داده می‌شوند.

جستجوی عمیق کننده تکراری

- در جستجوی عمیق‌کننده تکراری، گره‌های سطوح پائینی یک بار بسط داده می‌شوند، آنهایی که یک سطح بالاتر قرار دارند دوبار بسط داده می‌شوند و الی‌آخر تا به ریشه درخت جستجو برسد، که $d+1$ بار بسط داده می‌شوند.
- بنابراین مجموع دفعات بسط در این جستجو عبارتست از:
- $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- پیچیدگی زمانی این جستجو هنوز $O(b^d)$ است، و پیچیدگی فضا $O(bd)$ است.
- در حالت کلی، عمیق‌کننده تکراری، روش جستجوی برتری است؛ زمانی که فضای جستجوی بزرگی وجود دارد و عمق راه حل نیز مجهول است.

جستجوی دو طرفه

- **انجام دو جست و جوی همزمان، یکی از حالت اولیه به هدف و دیگری از هدف به حالت اولیه تا زمانی که دو جست و جو به هم برسند**



جستجوی دو طرفه

1. سؤال اصلی این است که، جستجو از سمت هدف به چه معنی است؟ ماقبل‌های (predeccessors) یک گره n را گره‌هایی درنظر می‌گیریم که n مابعد (successor) آنها باشد. جستجو به سمت عقب بدین معناست که تولید ماقبل‌ها از گرهٔ هدف آغاز شود.
2. چه کار می‌توان کرد زمانی که هدف‌های متفاوتی وجود داشته باشد؟
3. باید یک راه موثر برای کنترل هر گره جدید وجود داشته باشد تا متوجه شویم که آیا این گره قبلاً در درخت جستجو توسط جستجوی طرف دیگر، ظاهر شده است یا خیر.
4. نیاز داریم که تصمیم بگیریم که چه نوع جستجویی در هر نیمه قصد انجام دارد. بعنوان مثال شکل اسلاید قبل انتخاب جستجوی BFS در هر دو طرف را نمایش می دهد.

جستجوی دو طرفه

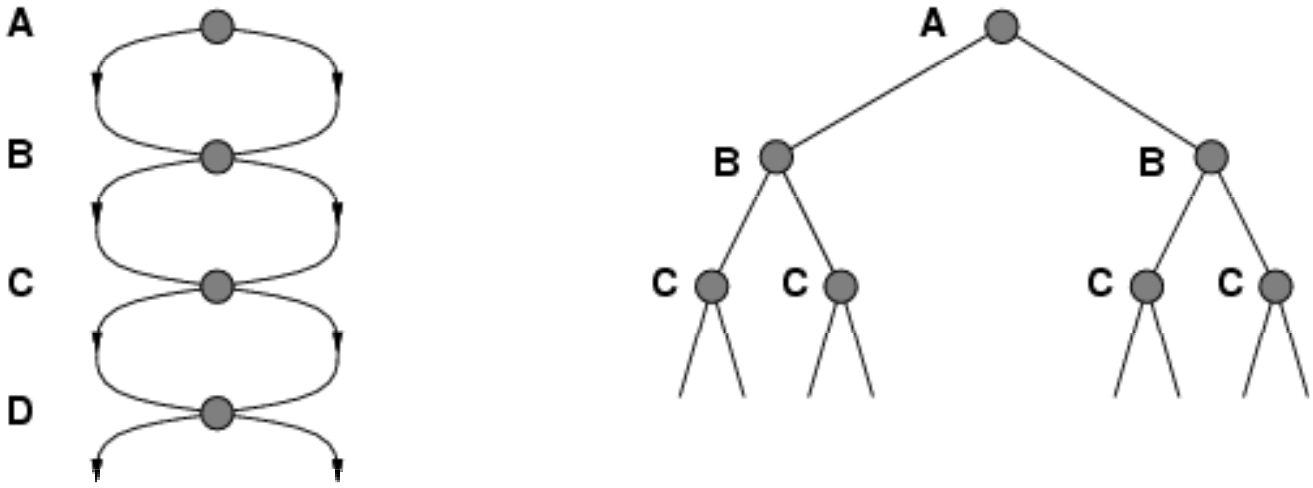
- **کامل بودن: بله**
  - **اگر هر دو جستجو، عرضی باشند و هزینه تمام مراحل یکسان باشد**
- **بهینگی: بله**
  - **اگر هر دو جستجو، عرضی باشند و هزینه تمام مراحل یکسان باشد**
- **پیچیدگی زمانی:** $O(b^{d/2})$
- **پیچیدگی فضا:** $O(b^{d/2})$

مقایسه استراتژی‌های جستجو

- ارزیابی استراتژی‌های جستجو. b فاکتور انشعاب، d عمل پاسخ، m ماکزیمم عمق درخت جستجو، l محدودیت عمق است.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

حالات تکراری



حالات تکراری

- نا توانی در کشف حالات تکراری ممکن است یک مسئله خطی را به یک مسئله نمایی تبدیل کند
- برای مسائل زیادی، حالات تکراری غیرقابل اجتناب هستند. این شامل تمام مسائلی می‌شود که عملگرها قابل وارونه شدن باشند، مانند مسائل مسیریابی و کشیش‌ها و آدمخوارها.
- سه راه برای حل مشکل حالات تکراری برای مقابله با افزایش مرتبه و سرریزی فشار کار کامپیوتر وجود دارد:
  - به حالتی که هم اکنون از آن آمده‌اید، برنگردید.
  - از ایجاد مسیرهای دوار بپرهیزید.
  - حالتی را که قبلاً تولید شده است، مجدداً تولید نکنید. این مسئله باعث می‌شود که هر حالت در حافظه نگهداری شود، پیچیدگی فضایی $O(b^d)$ داشته باشد.

(بهتر است این میزان را $O(s)$ در نظر بگیریم که s تعداد کل حالات در فضای حالت است.)

هوش مصنوعی

Artificial Intelligence

فصل چهارم - روشهای جستجوی آگاهانه

فهرست

مشکلات روشهای جستجوی نا آگاهانه

- معیار انتخاب گره بعدی برای گسترش دادن، تنها به شماره سطح گره بستگی دارد
- از ساختار مسئله بهره نمی برند
- در خت جستجو را با یک ساختار از پیش تعریف شده گسترش می دهند.
  - یعنی قابلیت تطبیق پذیری با آنچه که تا کنون در مسیر جستجو دریافته اند و نیز حرکتی که می تواند خوب باشد را ندارند.

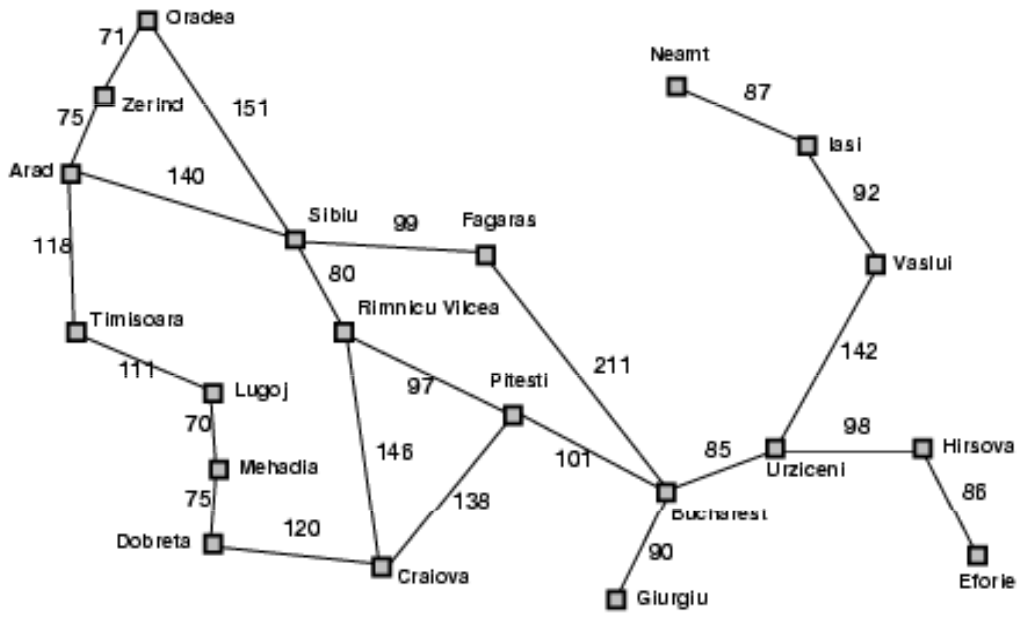
جستجوی آگاهانه (informed) یا اکتشافی (heuristic)

- در جستجوی آگاهانه اطلاعاتی در رابطه با هزینه رسیدن به هدف، در اختیار عامل قرار داده می شود.
- هیوریستیک ها
  - استراتژیهایی برای جستجو که اغلب اوقات ما را به جواب نزدیکتر می کنند.
  - از ساختار مساله نشات می گیرند و هدفشان راهنمایی و هدایت جستجو می باشد.
  - در این روشها هزینه پرداخت شده تا کنون و هزینه تخمینی رسیدن به هدف در نظر می شود.

جستجوی اول بهترین (Best First Search)

- در این الگوریتم ها یک تابع ارزیابی (Evaluation Function) تولید می شود که توضیحاتی در مورد مطلوب بودن یا نبودن بسط یک گره ارائه می دهد.
- گرهی که برای بسط انتخاب خواهد شد بر اساس نتایج تابع ارزیابی بهترین خواهد بود.
- پیاده سازی : از طریق یک صف که گرههای قابل بسط در آن بر اساس مطلوب بودن مرتب شده اند.
- نمونه هایی از این استراتژی دو الگوریتم زیر می باشند :
  - **جستجوی حریصانه (Greedy Search)**
  - **الگوریتم A***

نقشه رومانی به همراه طول جاده و فاصله خط مستقیم



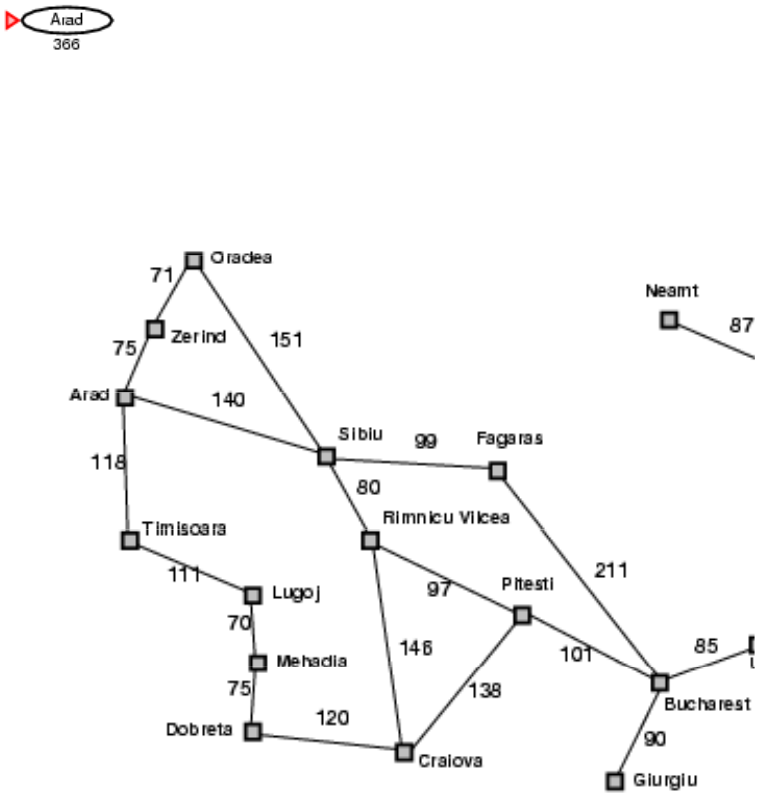
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

جستجوی حریصانه

- تابع ارزیابی h(n) (هیوریستیک)
  - هزینه تخمینی از گره n تا رسیدن به نزدیکترین هدف
- مثال
  - $h_{SLD}(n)$ = *فاصله مستقیم شهر n از بخارست*
- جستجوی حریصانه گرهی را بسط می دهد که به نظر می رسد نزدیکترین گره به هدف (بخارست) می باشد.

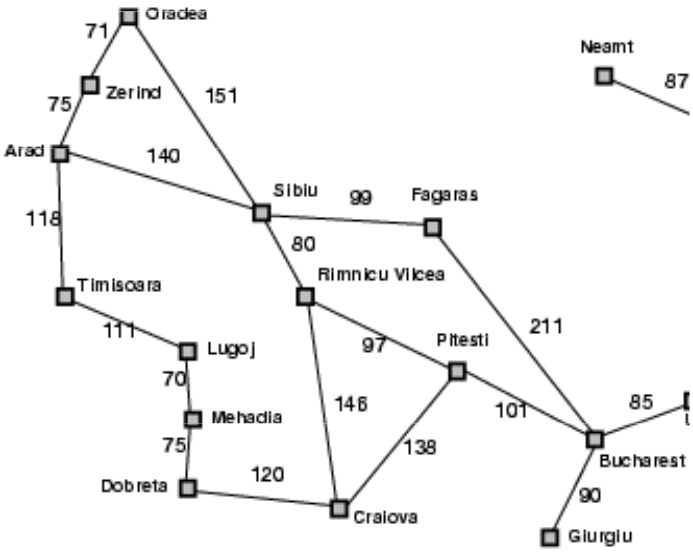
مثال جستجوی حریصانه

Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



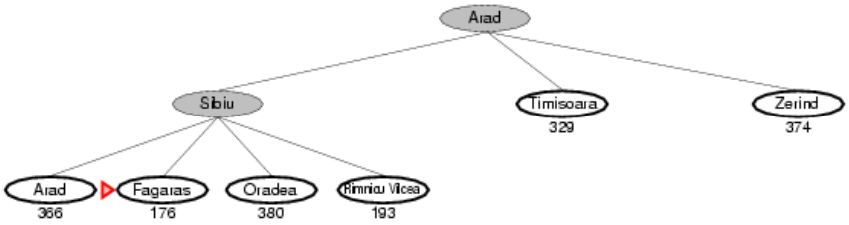
مثال جستجوی حریصانه

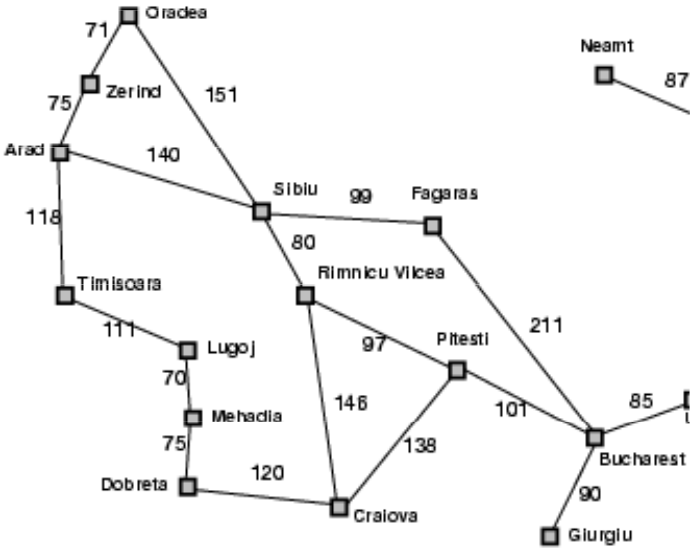
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



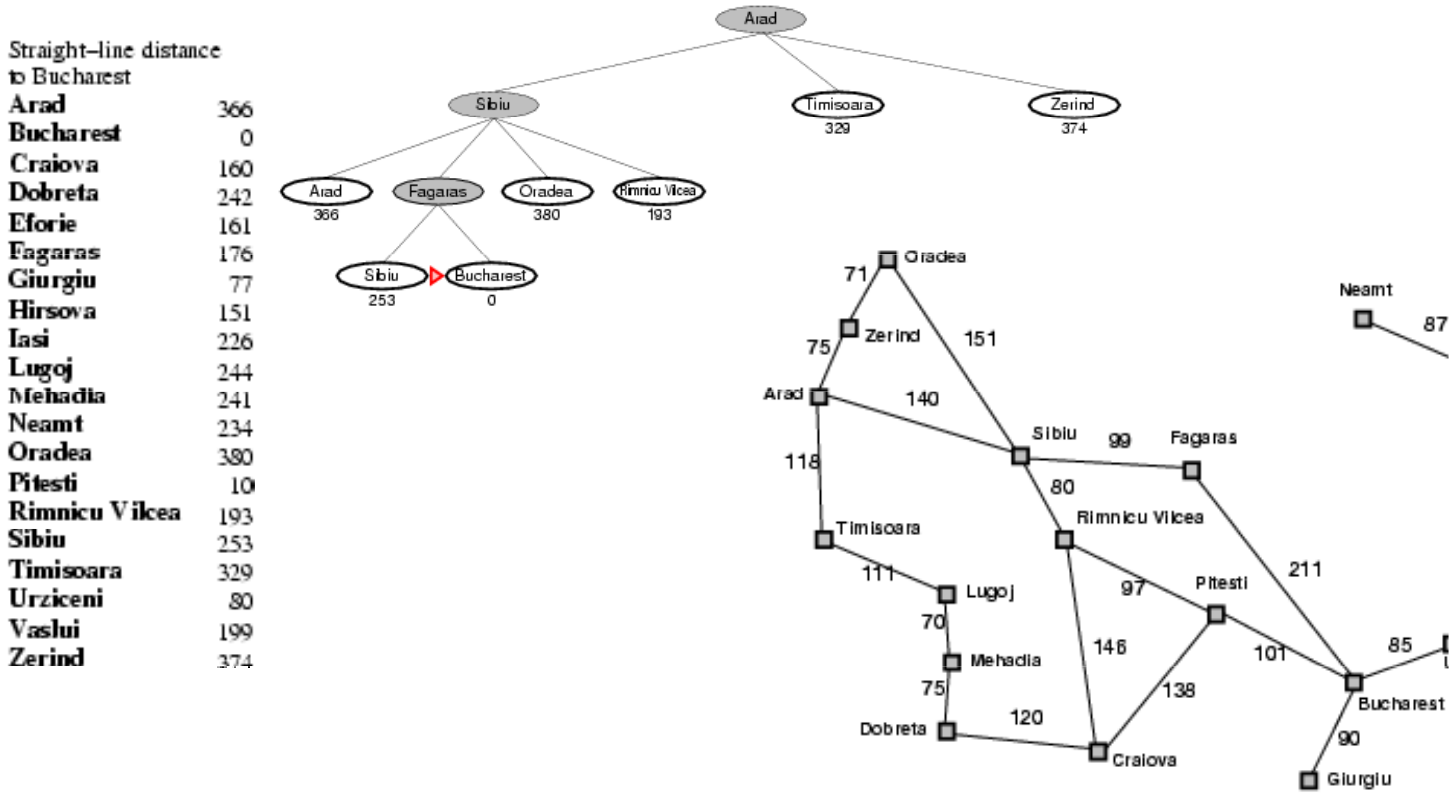
مثال جستجوی حریصانه

Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374





مثال جستجوی حریصانه



ویژگی‌های جستجوی حریصانه

- جستجوی حریصانه از لحاظ دنبال کردن یک مسیر ویژه در تمام طول راه به طرف هدف، مانند جستجوی عمقی است.
- کامل نیست : چراکه ممکن است در یک حلقه گیر کند
  - مثال : Iasi → Neamt → Iasi → Neamt →
  - در فضای حالت محدود با چک کردن گره های تکراری کامل است
- بهینه نیست
  - مثال قبل با هزینه 450km به هدف رسید که بهینه نبود

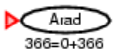
ویژگی‌های جستجوی حریصانه

- پیچیدگی زمانی در بدترین حالت برای جستجوی حریصانه $O(b^m)$، که m حداکثر عمق فضای جستجو است.
- جستجوی حریصانه تمام گره‌ها را در حافظه نگه می‌دارد، بنابراین پیچیدگی فضای آن مشابه پیچیدگی زمانی آن است.
- میزان کاهش پیچیدگی به مسئله و کیفیت تابع h بستگی دارد و در صورتیکه تابع مناسبی انتخاب شود هزینه فوق می تواند بسیار بهبود یابد.

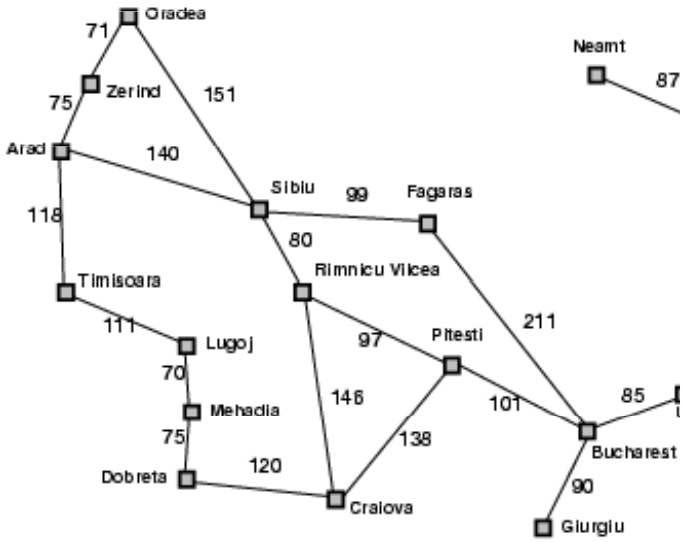
جستجوی A*

- در این جستجو از بسط گرههایی که تا کنون پر هزینه بودن آنها اثبات شده پرهیز می شود.
- در حقیقت جستجوی A* ترکیب مزایای جستجو با هزینه یکسان (UCS) و جستجوی حریصانه می باشد.
- تابع ارزیابی در A* :
- $f(n) = g(n) + h(n)$
- $g(n)$: هزینه مسیر پیموده شده تا n (استفاده شده در UCS)
- $h(n)$: هزینه تخمینی از n تا هدف (استفاده شده در جستجوی حریصانه)
- $f(n)$: هزینه کل تخمینی رسیدن به هدف که از گره n می گذرد

A* مثال جستجوی

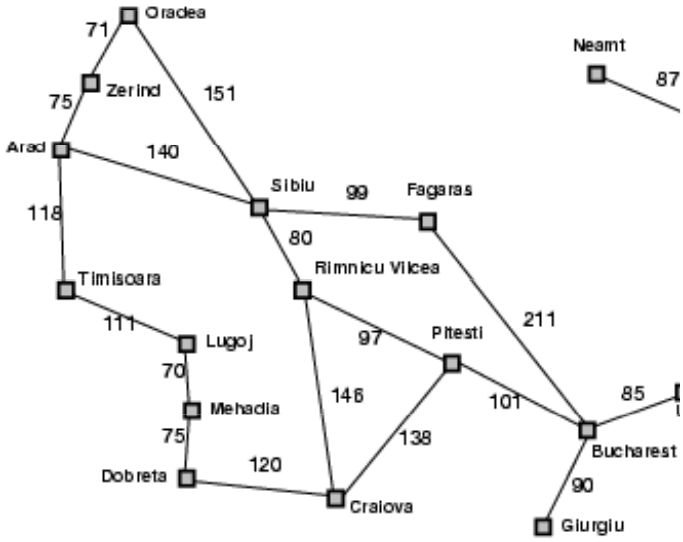


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



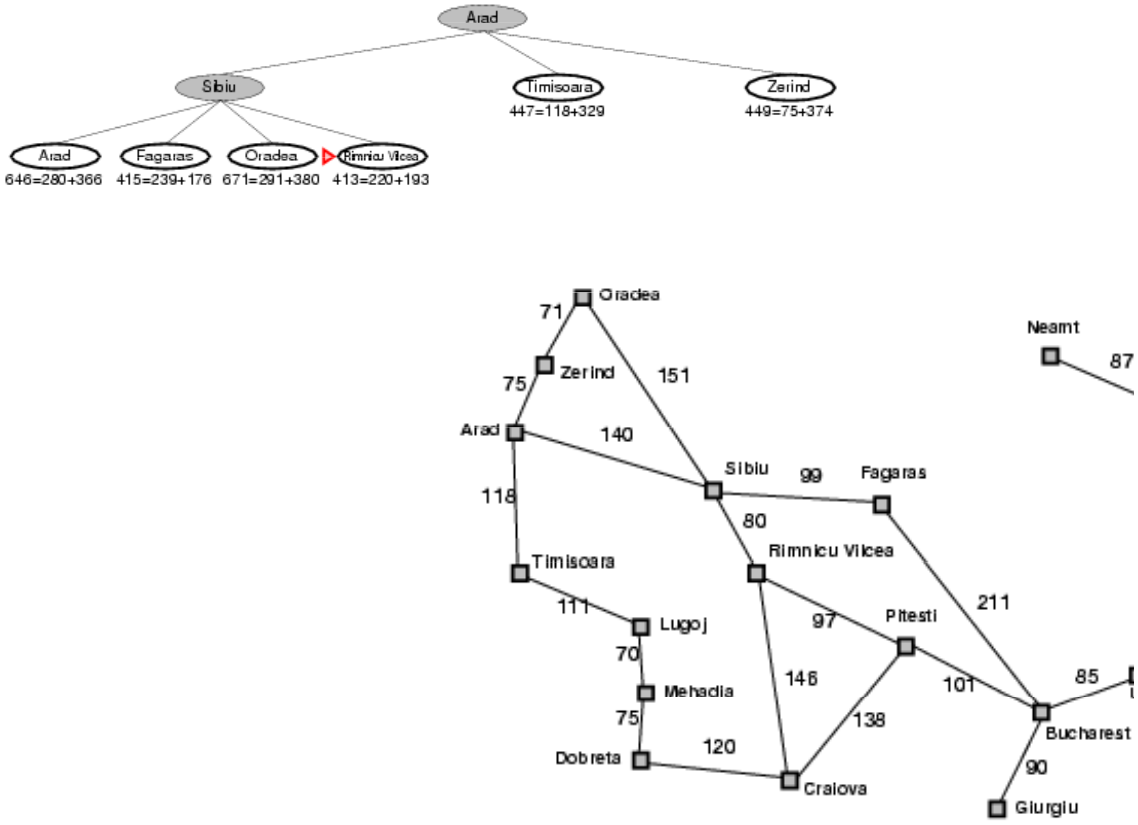
A* مثال جستجوی

Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



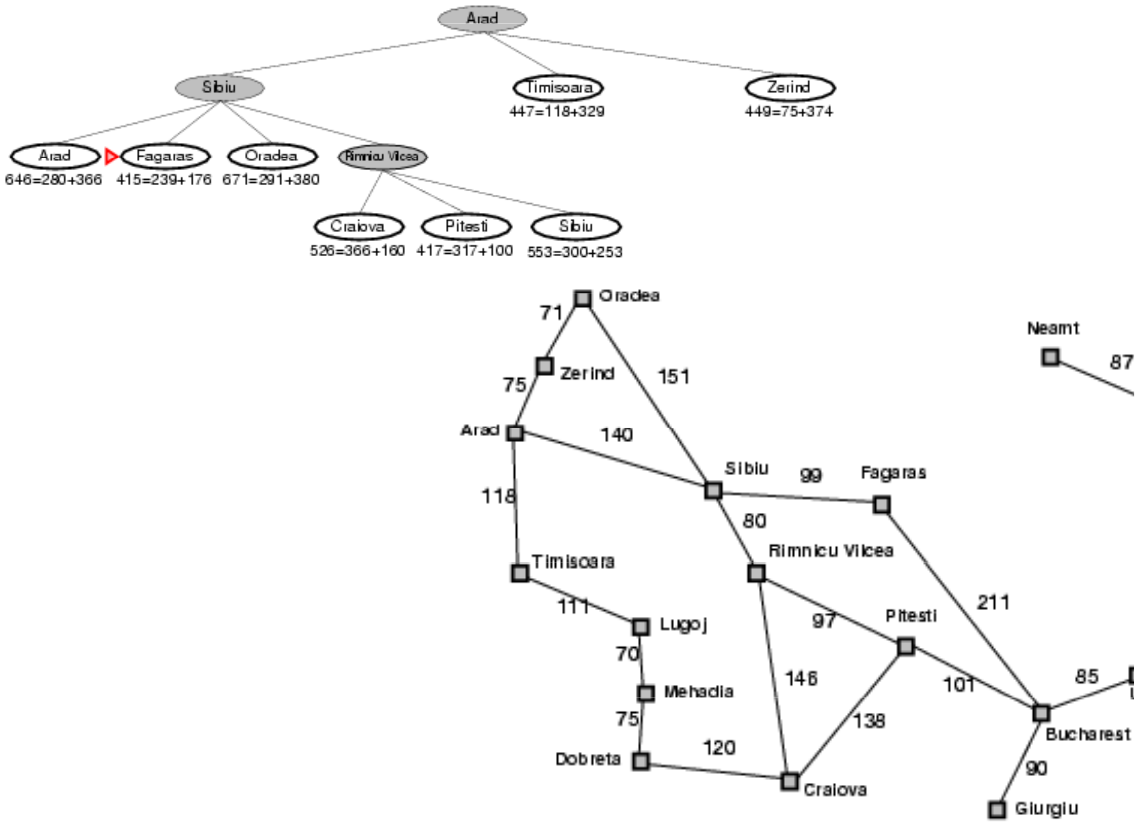
مثال جستجوی A^*

Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



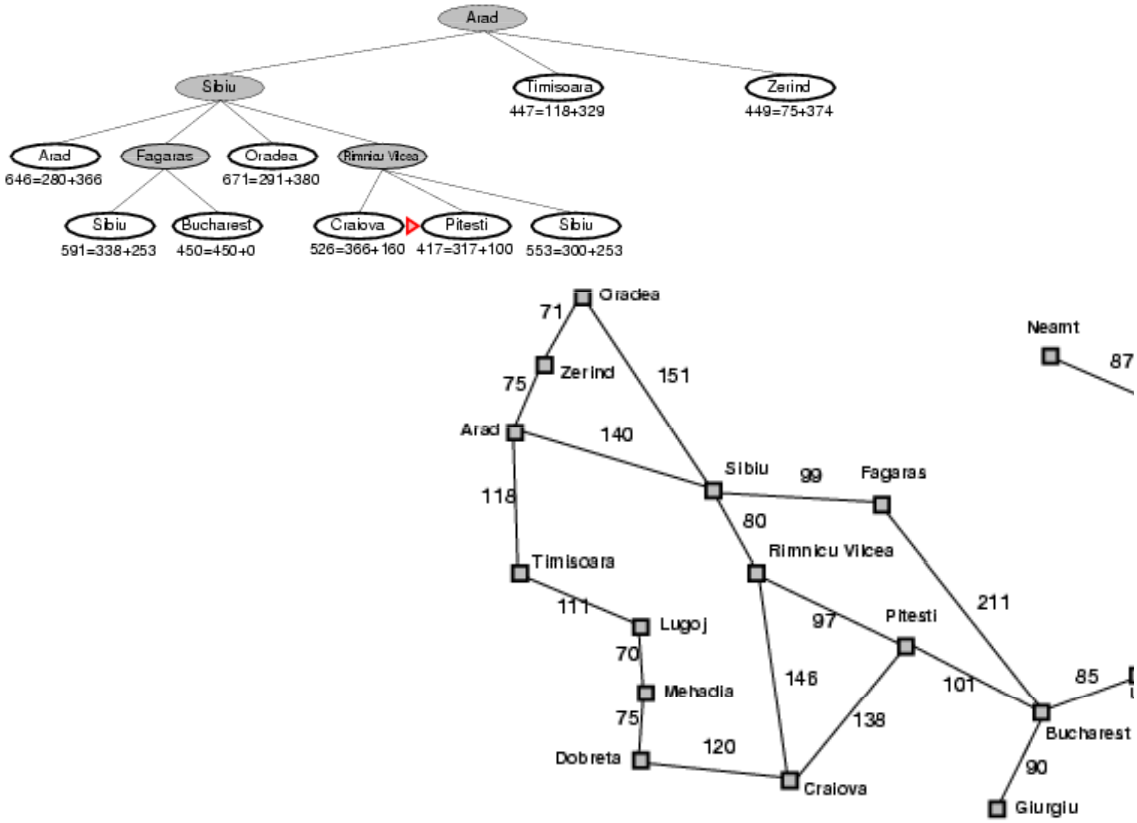
مثال جستجوی A^*

Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



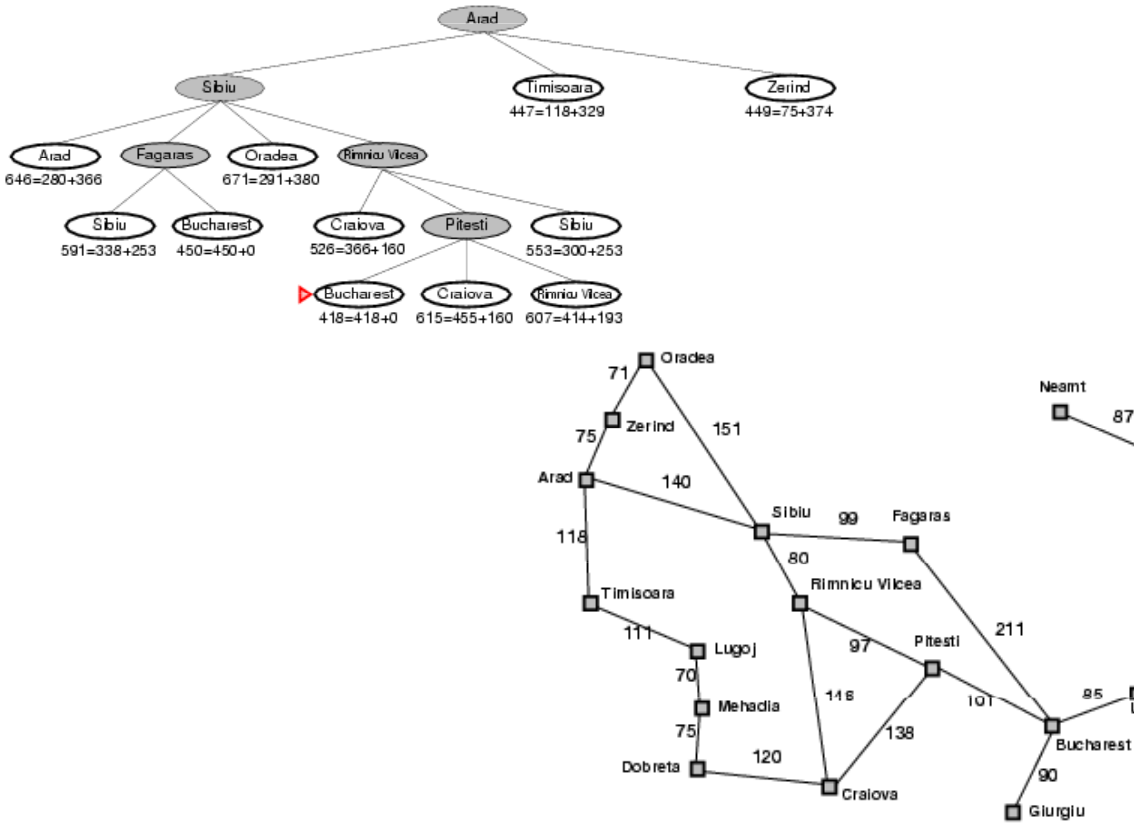
مثال جستجوی A*

Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



مثال جستجوی A*

Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

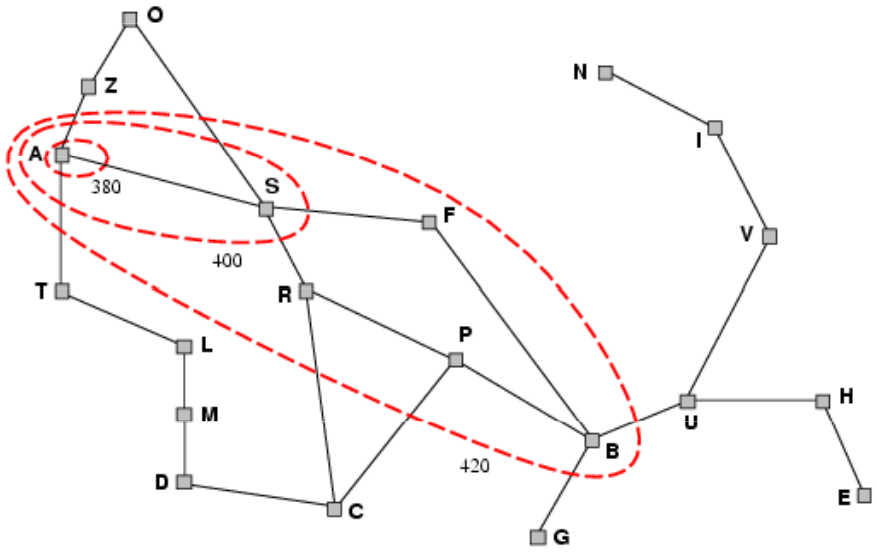


ویژگی‌های جستجوی A*

- **A* بهینه می باشد به شرطیکه تابع هیوریستیک h(n) مورد استفاده در آن یک هیوریستیک قابل قبول (Admissible) باشد**
- **در یک هیوریستیک قابل قبول همواره شرط** $h(n) \leq h^*(n)$ برقرار است
  - $h^*(n)$ هزینه واقعی مسیر از n تا هدف می باشد
  - $h(n) \geq 0$
  - $h(G)=0$ (که G یک هدف است)
  - $0 \leq h(n) \leq h^*(n)$
- مثال : هیوریستیک $h_{SLD}(n)$ قابل قبول می باشد، چراکه هرگز هزینه مسیر واقعی نمی تواند از هزینه خط مستقیم تا هدف کمتر باشد.

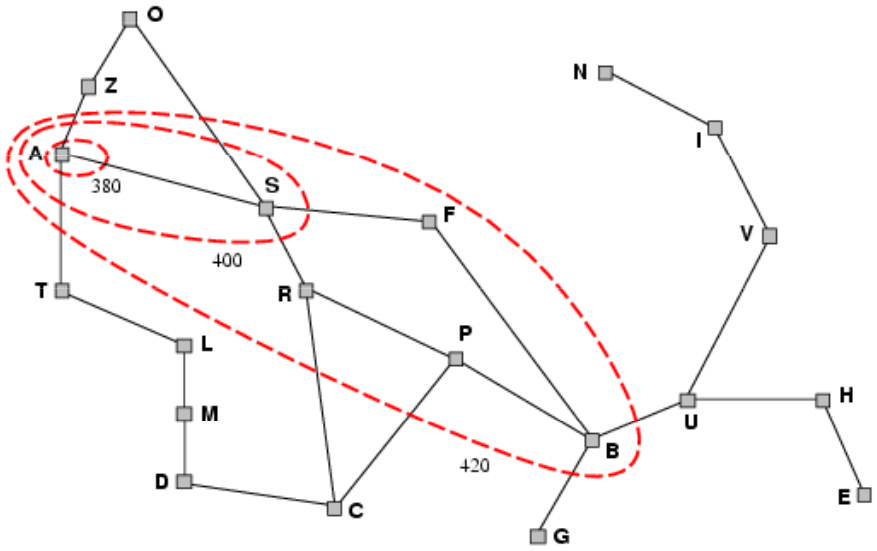
خاصیت یکنوایی (Monotonicity)

- تابع هیوریستیکی یکنوا است که در طول هر مسیری از ریشه، هزینه f حاصل از آن هرگز کاهش نیابد.
- اگر یکنوا نباشد، با ایجاد یک اصلاح جزئی آن را یکنوا می‌کنیم.
- با داشتن شرط فوق می توان کانتورهای (Contour) در فضای حالت کشید.
- بعنوان مثال کانتور با مقدار 400 شامل تمام گره هایی است که مقدار f آنها کمتر از 400 باشد.



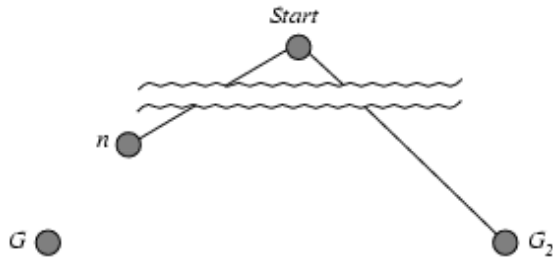
خاصیت یکنوایی (Monotonicity)

- در جستجو با هزینه یکسان (در A* با h=0) این نواحی (کانتورها) به صورت دایره وار در اطراف فضای حالت خواهند بود.
- هر چه تابع هیوریستیک صحیح تر باشد نواحی به سمت هدف کشیده خواهد شد و در اطراف مسیر بهینه باریک تر خواهند شد.

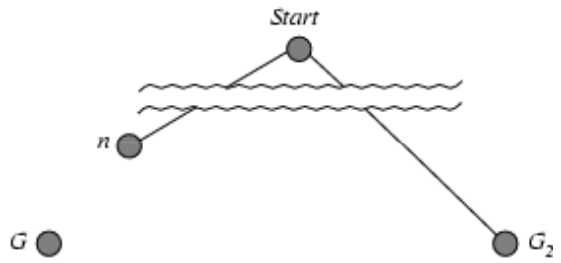


اثبات بهینگی A*

- اگر G حالت هدف بهینه با هزینه مسیر h* باشد.
- فرض کنید G2 یک هدف زیر بهینه (SubOptimal)می باشد. که هزینه مسیر g(G2)>h*
- نشان میدهیم که بسط G2 و پایان الگوریتم با یافتن این هدف غیر ممکن است
- اگر n یک گره بسط داده نشده از صف باشد که بر روی کوتاهترین مسیر به سمت هدف بهینه G باشد.



اثبات بهینگی A*



$f(G_2) = g(G_2)$ چون، $h(G2) = 0$

$> g(G)$ چون، $G2$ زیر بهینه است

$\geq f(n)$ چون، h قابل قبول است

- $f(G_2) > f(n)$ در نتیجه A^* هرگز $G2$ را برای بسط انتخاب نمی کند.

ویژگی‌های جستجوی A*

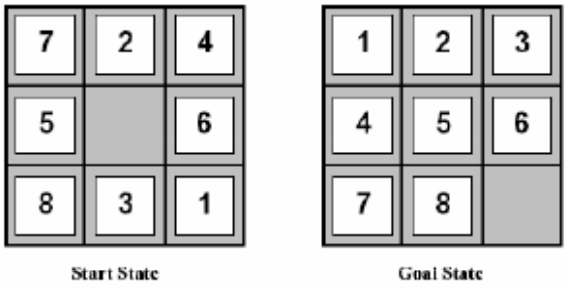
- **A* معمولاً قبل از اینکه دچار کمبود زمان شود، دچار کمبود فضا می‌شود.**
  زیرا این جستجو تمام گره‌های تولید شده را در حافظه ذخیره می‌کند.
- **A* کامل است (در گرافهایی با فاکتور انشعاب محدود)**
- **پیچدگی حافظه** $O(b^d)$ (چراکه گره های تولید شده در حافظه می مانند)
- **پیچیدگی زمانی : نمایی و بر حسب |خطای نسبی * h طول راه حل|**
  - **فقط در شرایطی پیچیدگی زمانی، نمایی نخواهد بود که رشد خطا در تابع هیوریستیک، رشدی سریعتر از لگاریتم هزینه واقعی مسیر نداشته باشد**
  - $|h(n)-h^*(n)| \leq O(\log h^*(n))$

ویژگی‌های جستجوی A*

- اگر f^* هزینه واقعی مسیر بهینه از شروع تا هدف تعریف شود، می توان گفت :
  - A* تمام گره ها با $f(n)<f^*$ را بسط خواهد داد.
  - A* ممکن است تعدادی از گره هایی که دقیقا روی « کانتور هدف» (یعنی $f(n)=f^*$) قرار دارند را قبل از اینکه هدف انتخاب شود بسط دهد.
  - A* هرگز گرهی را با $f(n) > f^*$ هرگز گسترش نمی دهد.
- A* برای هر تابع هیوریستیک قابل قبول، دارای کارایی بهینه می باشد. (Optimally efficient) می باشد. یعنی نسبت به دیگر الگوریتمهای بهینه کمترین تعداد گره را گسترش می دهد.

هیوریستیک های قابل قبول

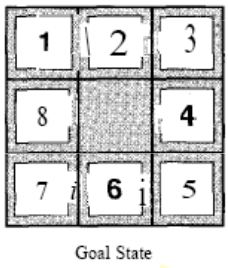
- مثال : معمای ۸
- $h_1(n)$ = تعداد خانه هایی که در محل خود قرار ندارند
- $h_2(n)$ = مجموع فاصله خانه ها از محل های هدف یا نهایی

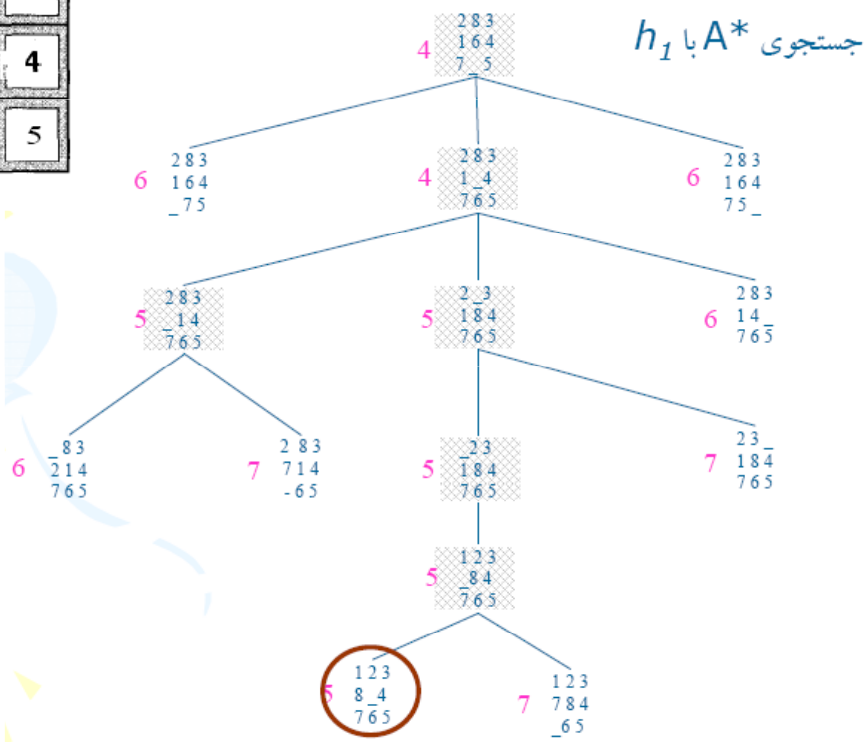


$h_1(n)=6$

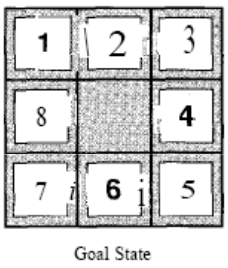
$h_2(n)= 4+ 0 + 3 + 3 + 1 + 0 +2 + 1 =14$

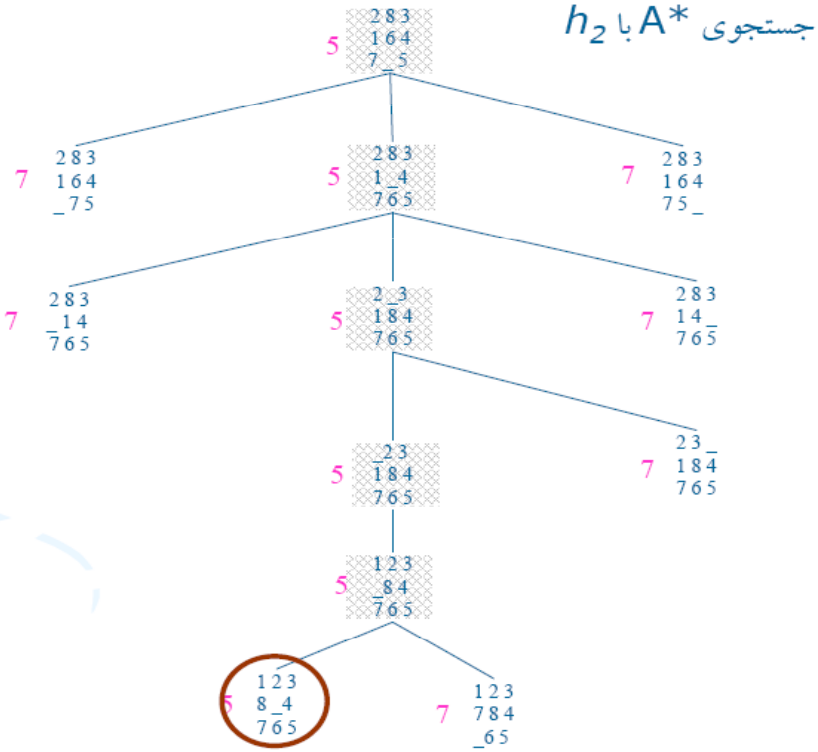
جستجوی A* با h_1





جستجوی A* با h_2





تسلط (Dominance)

- اگر به ازاء هر n داشته باشیم $h_2(n) > h_1(n)$
  - و هر دو هیوریستیک قابل قبول باشند
  - آنگاه h_2 بر h_1 تسلط دارد و برای جستجو بهتر است.

- مثال : مقایسه تعداد گره های بسط داده شده برای رسیدن به هدف در عمق های 12 و 24

- $d=12$ IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes

- $d=24$ IDS = 54,000,000,000
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

فاکتور انشعاب موثر (b^*) (Effective Branching Factor)

- یک راه برای تشخیص کیفیت کشف کنندگی، فاکتور انشعاب مؤثر b^* است
- اگر مجموع تعداد گره‌های بسط داده شده توسط A^* برای یک مسئله ویژه N باشد و عمق راه حل d، پس b^* فاکتور انشعابی است که به صورت زیر می باشد.

$$N=1+ b^*+(b^*)^2\ldots+(b^*)^d$$

- بعنوان مثال اگر A^* در عمق $d=5$ هدفی را با گسترش 25 گره بدست آورد، فاکتور انشعاب موثر از طریق محاسبه $b^*=1.91$ بدست می آید.
- در یک تابع هیوریستیک خوب، b^* نزدیک به یک می باشد و در اینصورت کیفیت کشف کنندگی خوبی دارد.

فاکتور انشعاب موثر (b*) (Effective Branching Factor)

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Figure 4.8 Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A* algorithms with h_1, h_2. Data are averaged over 100 instances of the 8-puzzle, for various solution lengths.

بهبود A* در مصرف حافظه

- جستجوی عمیق کننده تکراری A*
  (Iterative Deepening A*) (IDA*)
  - تعمیم جستجوی عمیق کننده تکراری با استفاده از هیوریستیک ها
- جستجوی ساده شده با حافظه محدود A*
  (Simplified Memory bounded A*) (SMA*)
  - کاهش اندازه حافظه به طوریکه با میزان حافظه موجود مطابقت داشته باشد

جستجوی اکتشافی با حافظه محدود IDA*

- ساده ترین راه برای کاهش حافظه مورد نیاز A* استفاده از عمیق کننده تکرار در زمینه جست و جوی اکتشافی است.
- A* در برخی از مسائل مشکل دچار کمبود حافظه میشود.
- در جستجوی IDA* بجای محدوده عمقی از محدوده f-cost استفاده میشود.
- جستجوی IDA* ترکیب جستجوی عمیق کننده تکراری و A* می باشد تا از مزایای هر دو بهره مند باشد.
- در هر تکرار مقدار f(n) برای گره ها مشخص شده و فقط گره هایی بسط داده می شوند که :
- $f(n) \leq$ f-cost
- در صورتیکه گره هدف در این ناحیه (کانتور) یافت نشود، جستجو با افزایش ناحیه دنبال خواهد شد.
- کامل و بهینه : به شرطیکه کوتاهترین مسیر هدف در حافظه جای بگیرد

جستجوی اکتشافی با حافظه محدود IDA*


```
function IDA*( problem) returns a solution sequence
    inputs: problem, a problem
    local variables: f-limit, the current f-COST limit
                     root, a node

root ← MAKE-NODE( INITIAL-STATE[problem ])
f-limit ← f-COST( root)
loop do
    solution, f-limit ← DFS-CONTOUR(root , f-limit)
    if solution is non-null then return solution
    if f-limit =∞ then return failure
end
```

جستجوی اکتشافی با حافظه محدود IDA*


```
function DFS-CONTOUR(node, f-limit) returns a solution sequence and a new f- COST limit
  inputs: node, a node
          f-limit, the current f- COST limit
  static: next-f, the f- COST limit for the next contour, initially ∞

  if f- COST[node] > f-limit then return null, f- COST[node]
  if GOAL-TEST[problem](STATE[node]) then return node, f-limit
  for each node s in SUCCESSORS(node) do
      solution, new-f ← DFS-CONTOUR(s, f-limit)
      if solution is non-null then return solution, f-limit
      next-f ← MIN(next-f, new-f); end
  return null, next-f
```

جستجوی اکتشافی با حافظه محدود IDA*

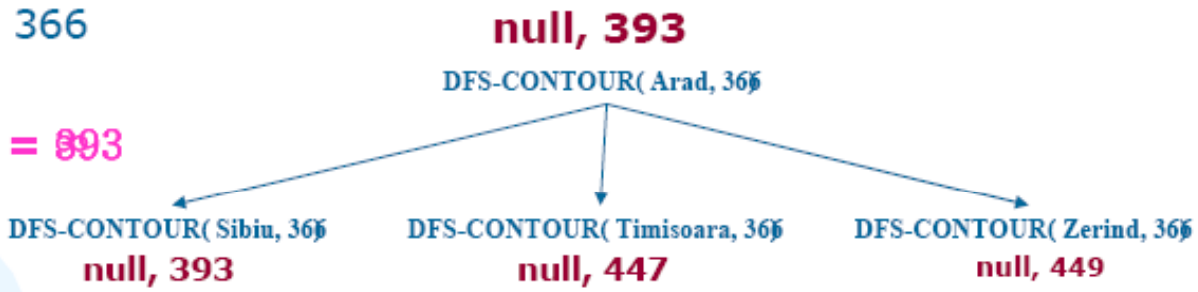
- در هر تکرار IDA* جستجو از ریشه شروع می شود. با این تفاوت که در تابع بازگشتی بجای F-limit مقدار next-f مرحله قبل را جایگذاری می کنیم

جستجوی اکتشافی با حافظه محدود IDA*

1st iteration:

f-limit = 366

next-f = 393

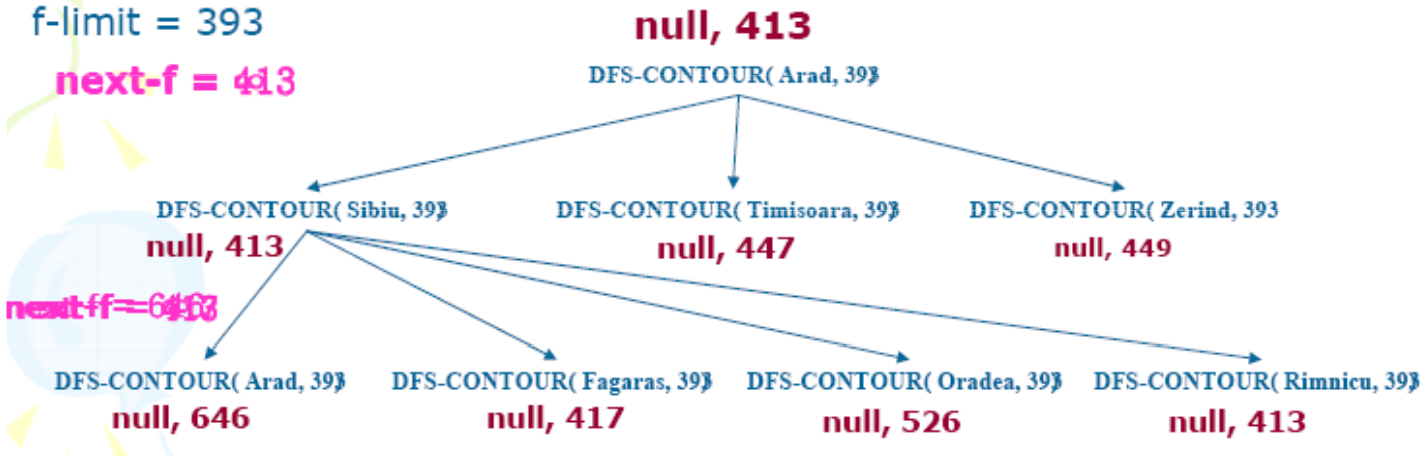


جستجوی اکتشافی با حافظه محدود IDA*

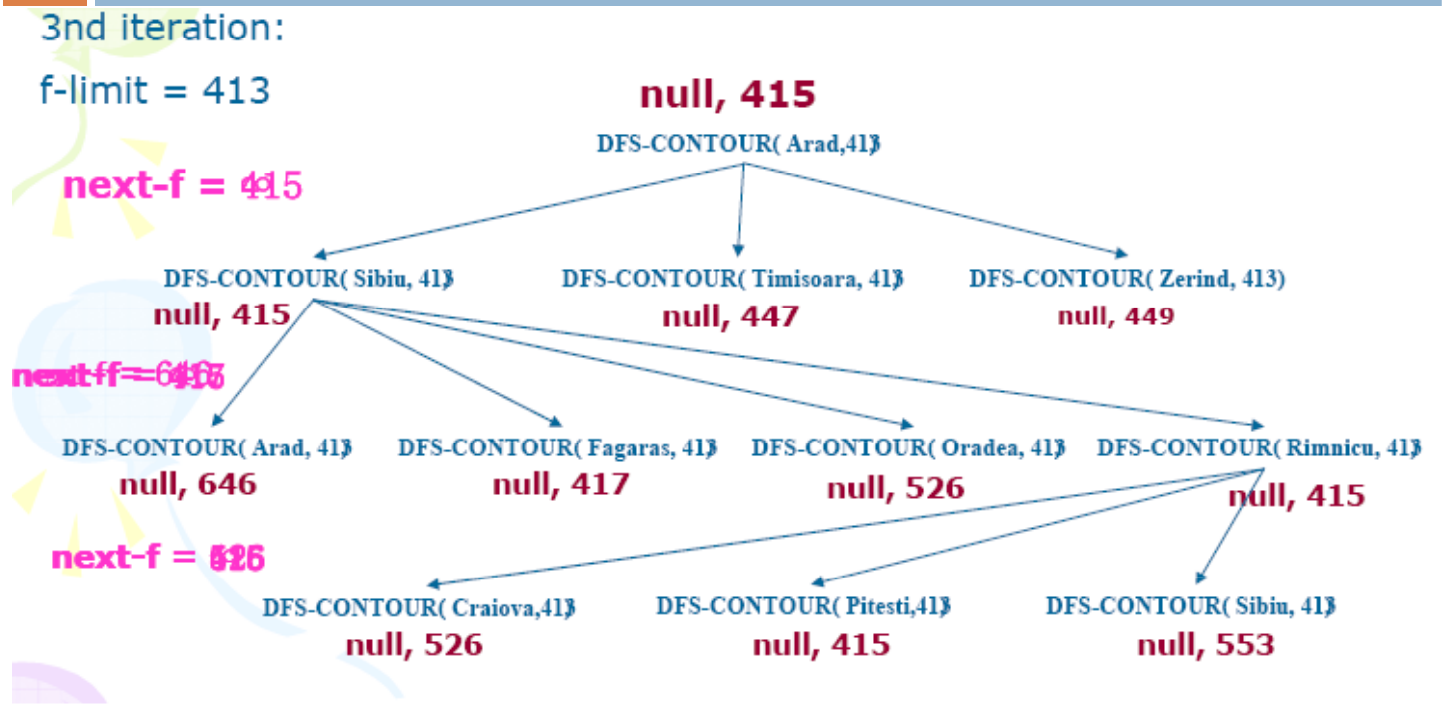
2nd iteration:

f-limit = 393

next-f = 413



جستجوی اکتشافی با حافظه محدود IDA*



فهرست

جستجوی SMA* (Simplified Memory bounded A*)

3

- IDA* حداکثر bd گره در حافظه نگه می دارد.
- بین تکرارهای مختلف تنها چیزی که نگه داشته می شود، تنها یک عدد است و آن F-cost جاری می باشد و قدرت به خاطر سپردن تاریخچه خود را ندارد و از این رو مجبور به تکرار محاسبات می باشد.
- دوباره کاری در بسط گره ها به خصوص در مورد فضای حالاتی که بیشتر شبیه یک گراف هستند تا یک درخت بیشتر نمود پیدا می کند.
- SMA* از تمام حافظه موجود برای اجرای جستجو استفاده می کند.
- مسلما وجود حافظه بیشتر، کارایی جستجو را وسعت می بخشد.
- زمانی که حافظه بیشتری موجود باشد، می توان گره های بیشتری را در حافظه ذخیره کرد و بدین ترتیب از تکرار آنها اجتناب کرد.
- در این الگوریتم در صورت کمبود حافظه، گره هایی که f-cost بالاتری دارند حذف خواهند شد.
- با دادن حافظه منطقی SMA* می تواند مسائل مشکل تری نسبت به A* را حل کند.

جستجوی SMA* (Simplified Memory bounded A*)

4

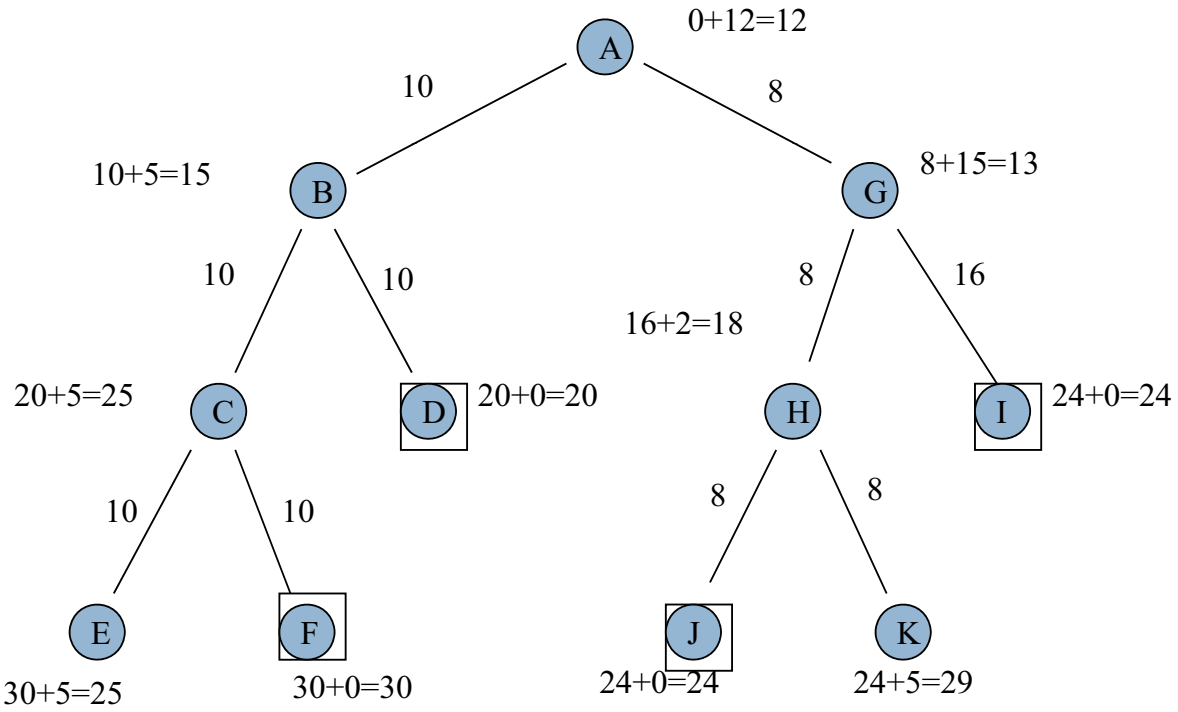
- به گره هایی که در صورت کمبود حافظه، حذف می شوند، گرههای فراموش شده (Forgotten Nodes) گفته می شود.
- برای اجتناب از جستجوی مجدد گره های فراموش شده، اطلاعاتی در مورد هزینه بهترین مسیر زیر درخت فراموش شده، در گره های والدین خود نگهداری می شود.
- همواره هزینه بهترین فرزند فراموش شده در والد ذخیره می شود.
- فقط زمانی زیر درخت فراموش شده دوباره بسط داده می شود که ثابت شود، تمام مسیر های دیگر بدتر از مسیر فراموش شده است.

جستجوی SMA* (Simplified Memory bounded A*)

- هزینه یک گره غیر هدف در حدکثر عمق ممکن، بی نهایت خواهد بود.
- هر بار عمیق ترین گره با کمترین f-cost بسط داده خواهد شد.
- در صورت کمبود حافظه سطحی ترین گره با بیشترین f-cost حذف (فراموش) خواهد شد.

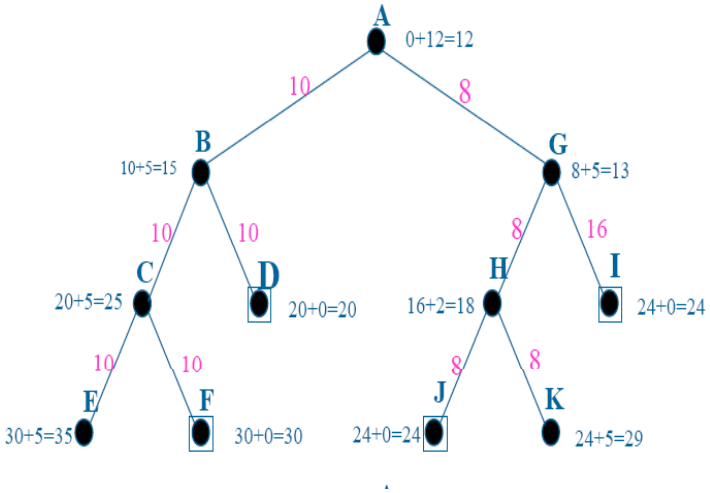
- این الگوریتم کامل است به شرط آنکه حافظه برای ذخیره کم عمق‌ترین مسیر راه حل کافی باشد.
- این الگوریتم بهینه است، اگر حافظه کافی برای ذخیره کم‌عمق‌ترین مسیر راه‌حل کافی باشد. بعلاوه بهترین راه‌حلی را برمی‌گرداند که بتواند با حافظه موجود مطابقت داشته باشد.
- زمانی که حافظه موجود برای کل درخت جستجو کافی باشد، این جستجو بهینگی کارایی (Optimally efficient) خواهد داشت.

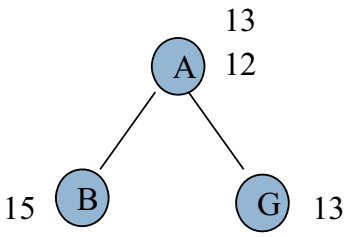
مثال SMA*



مثال SMA* (با حافظه محدود 3)

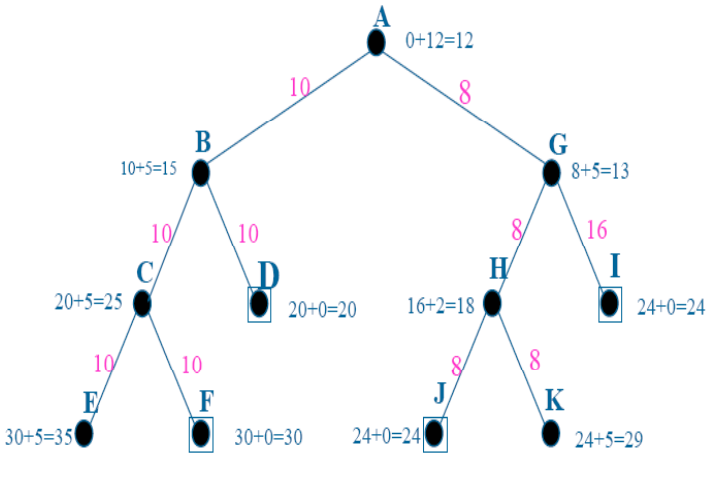
7

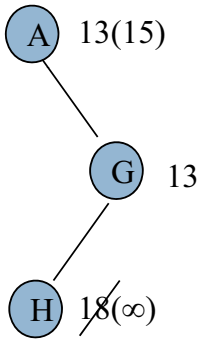




مثال SMA* (با حافظه محدود 3)

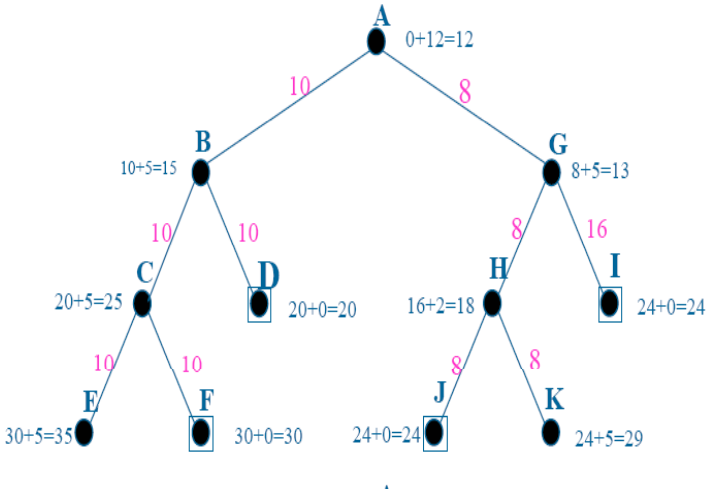
8

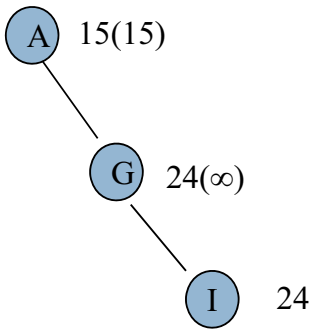




مثال SMA* (با حافظه محدود 3)

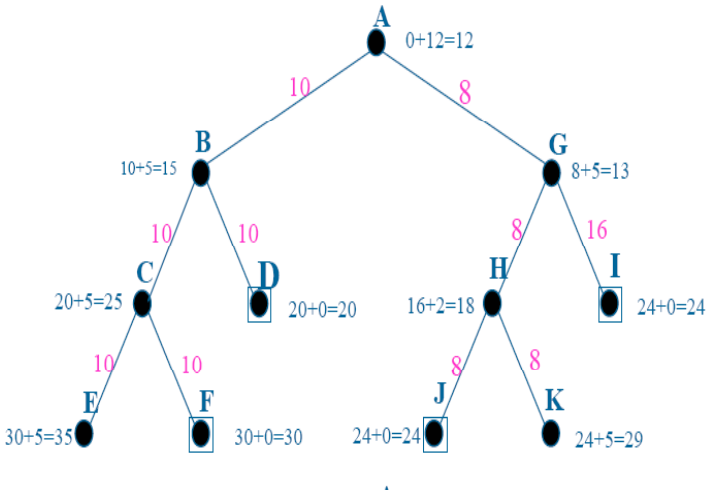
9

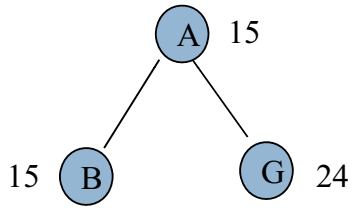




مثال SMA* (با حافظه محدود 3)

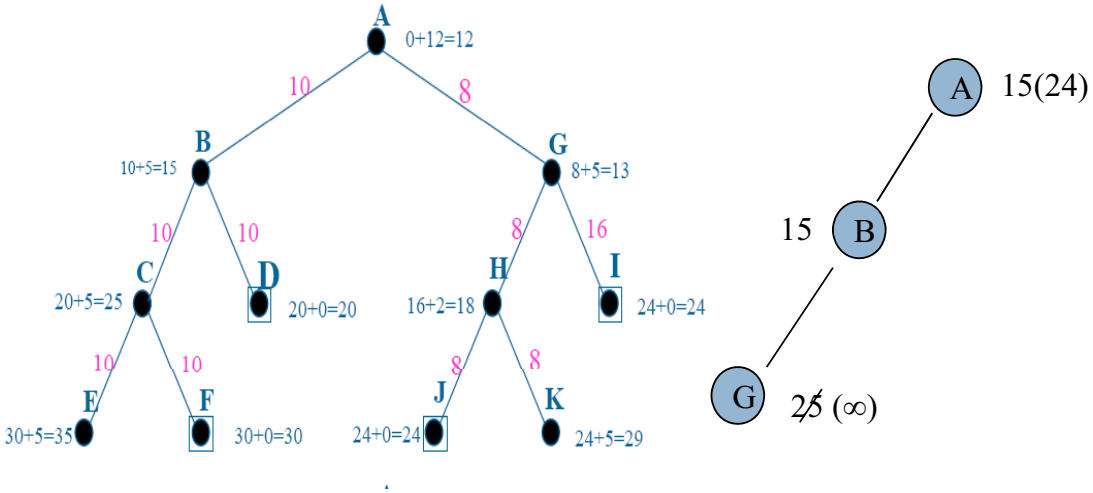
10





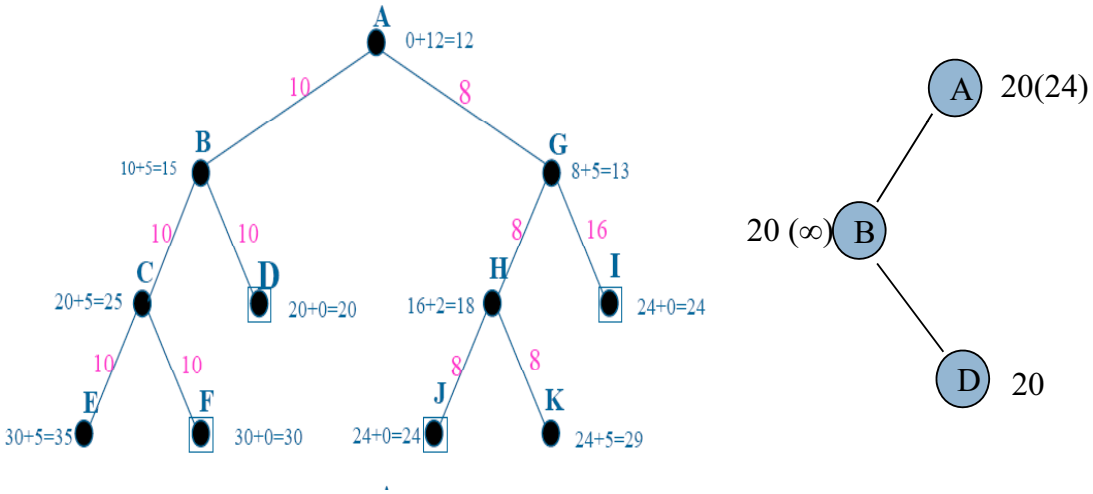
مثال SMA* (با حافظه محدود 3)

11



مثال SMA* (با حافظه محدود 3)

12



نکات الگوریتم SMA*

- در بسط گره s ، اگر s هدف نباشد و در عمق ماکزیمم قرار دارد (یعنی مسیری که به آن می رود تمام حافظه موجود را مصرف کرده) آنگاه :

  $f(s) \rightarrow \infty$ و بدین ترتیب این گره هیچ گاه برای بسط انتخاب نمی شود.

- اگر همه فرزندان گره n تولید شده است، آنگاه f-cost گره n را به کمترین f-cost فرزندانش تغییر بده و این تغییر را تا ریشه اعمال کن

الگوریتم های جست و جوی محلی و بهینه سازی

- الگوریتم های قبلی، فضای جست و جو را به طور سیستماتیک بررسی میکنند
  - تا رسیدن به هدف یک یا چند مسیر نگهداری میشوند
  - مسیر رسیدن به هدف، راه حل مسئله را تشکیل میدهد
- در الگوریتم های محلی مسیر رسیدن به هدف مهم نیست
  - مثال: مسئله ۸ وزیر
- امتیاز عمده جست و جوهای محلی
  - ارائه راه حلهای منطقی در فضاهای بزرگ و نامتناهی

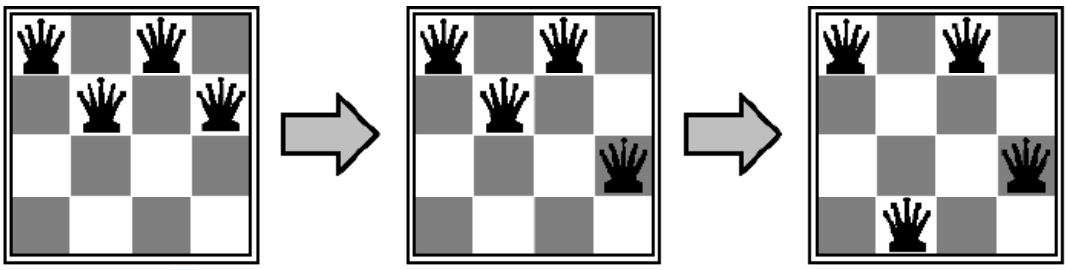
الگوریتمهای اصلاح تکراری (Iterative Improvement Algorithms)

- ایده کلی آن است که با یک ساختار کامل آغاز کرده و تغییراتی به منظور اصلاح کیفیت این ساختار انجام شود.
- در اکثر این مسائل ، مسیر رسیدن به هدف مهم نیست (مسئله ۸ وزیر)
- در مسائلی کاربرد دارد که پاسخ را نمی دانیم و تنها می توانیم حالات بدست آمده را مقایسه کرده و بگوییم کدام بهتر است.
- مسئله ۸ وزیر مثالی از این دسته می باشد که در آن با هر حرکت، ساختار قبلی اصلاح شده و سعی می کنیم به هدف نزدیکتر شویم.
- **الگوریتم های اصلاح تکراری**
  - **تپه نوردی (Hill climbing)**
  - **جستجوی Simulated Annealing**

الگوریتمهای جست و جوی محلی (Local Search Algorithms)

الگوریتمهای اصلاح تکراری (Iterative Improvement Algorithms)

- بعنوان مثال در مسئله n–وزیر سعی می شود با هر حرکت تعدا برخورد ها کاهش یابد.



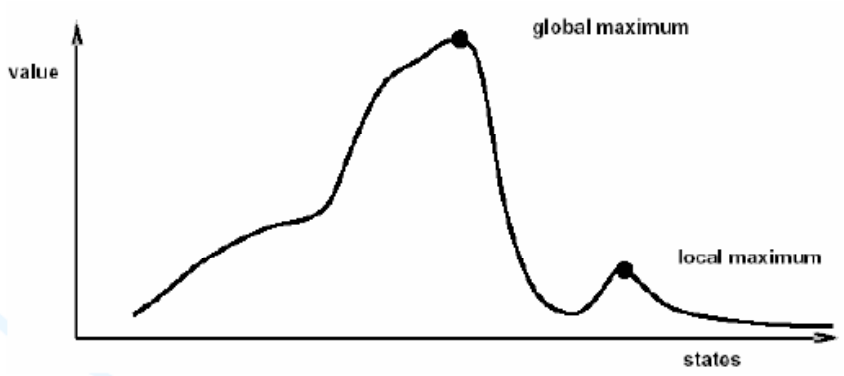
جستجوی تپه نوردی (Hill climbing)

- مشابه یافتن نقطه بلند کوه اورست در یک مه غلیظ
- حلقه ای که در جهت افزایش مقدار حرکت میکند(بطرف بالای تپه)
- رسیدن به بلندترین قله در همسایگی حالت فعلی، شرط خاتمه است

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                   neighbor, a node
  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
     neighbor ← a highest-valued successor of current
     if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
     current ← neighbor
```

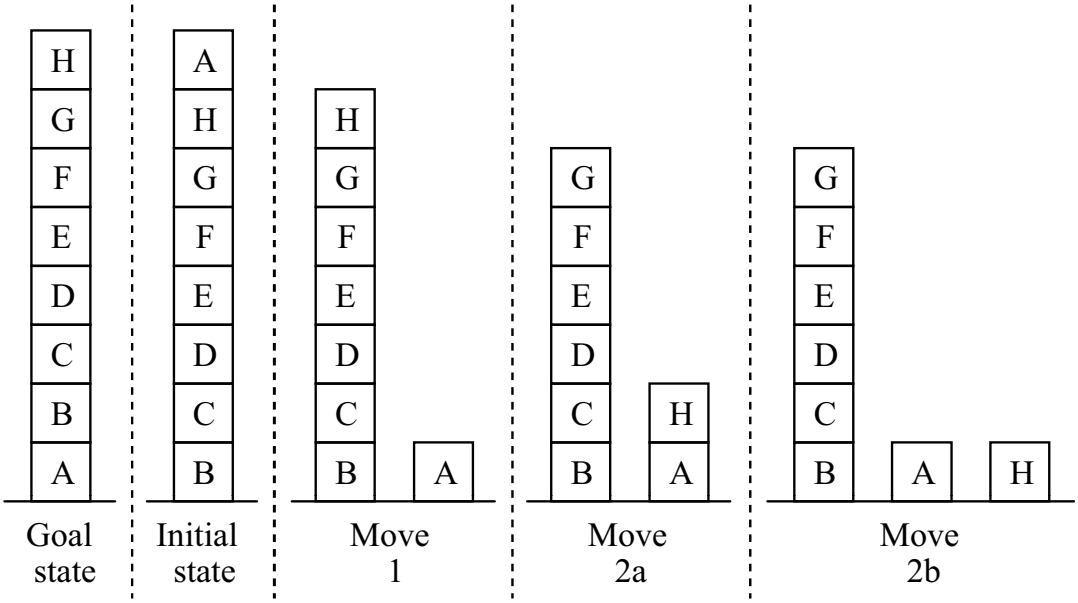
جستجوی تپه نوردی (Hill climbing)

- ممکن است در یک Max محلی گیر کند.

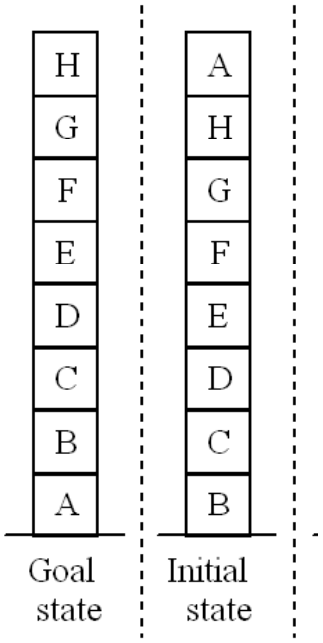


- راه حل : تکرار تپه نوردی با شروع تصادفی

مثال : جستجوی تپه نوردی (Hill climbing)

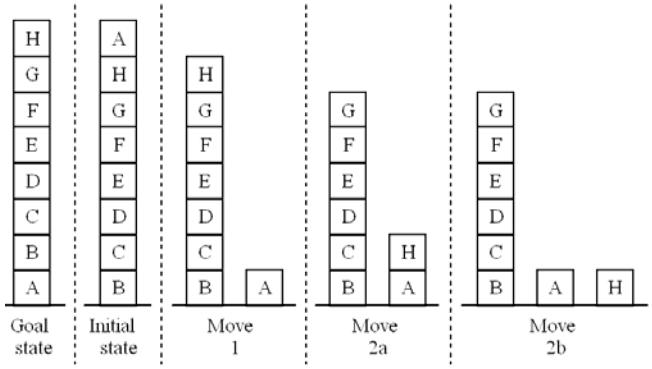


مثال : جستجوی تپه نوردی (Hill climbing)



- تابع هیوریستیک :
  - افزایش شمارنده به ازای تعداد بلوک هایی که بر روی بلوک صحیح نشسته اند . کاهش شمارنده به ازای تعداد بلوک هایی که بر روی بلوک صحیح قرار ندارند.
  - در حالت هدف این عدد حاصل این تابع باید 8 باشد.
  - در حالت اولیه :
  - افزایش شمارنده برای C,D,E,F,G,H برابر با +6
  - کاهش شمارنده به ازای A,B برابر با -2
  - مقدار نهایی تابع هیوریستیک = +4

مثال : جستجوی تپه نوردی (Hill climbing)

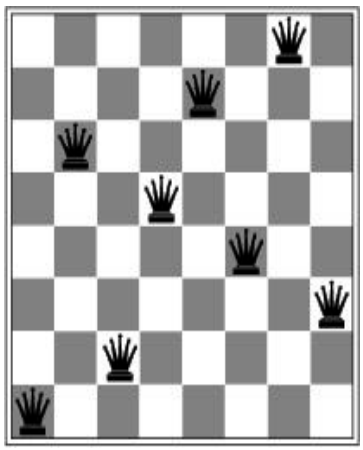


- مقدار نهایی تابع هیوریستیک در حرکت 1 برابر است با +6
- مقدار نهایی تابع هیوریستیک در حرکت 2a برابر است با +4
- مقدار نهایی تابع هیوریستیک در حرکت 2b برابر است با +4
- مقدار +6 بعد از حرکت 1 بعنوان یک Max محلی می باشد. !

مثال 8 وزیر : جستجوی تپه نوردی (Hill climbing)

a)

b)



h = تعداد جفت وزیرهایی که همدیگر را تحدید می کنند

a) وضعیت خاصی که در آن *h*=17 می باشد و مقادیر تابع هیوریستیک برای حرکتهای ممکن بعدی نشان داده شده است

b) یک مینیمم محلی برای تابع هیوریستیک می باشد که در آن *h*=1 می باشد

جستجوی Simulated Annealing

- شبیه جستجوی تپه نوردی می باشد با این تفاوت که در هر مرحله حالت بعدی، بجای انتخاب بهترین حرکت ، به طور تصادفی انتخاب می شود.
- اگر حرکت تصادفی انتخاب شده موجب بهبود وضعیت شود، انجام میشود و در غیر اینصورت فقط با یک احتمال (که به صورت نمایی کاهش می یابد) آن حرکت انجام می شود.
- ایده اصلی : فرار از max های محلی با اجازه دادن به انجام حرکتهای بد (پایین آمدن)
- اما به تدریج اندازه و تعداد حرکتهای بد را کاهش می دهد.
- این الگوریتم اصلاح الگوریتم تپه نوردی می باشد که در آن هنگام رسیدن به max محلی به جای شروع مجدد تصادفی، اجازه پایین رفتن از تپه برای فرار از آن محل، داده می شود.

الگوریتم پرتو محلی (Local Beam Search)

- از k حالت اولیه بصورت تصادفی مسئله شروع می شود.
- تمام این k حالت بسط داده میشود.
- اگر یکی از گره های بسط داده هدف باشد که الگوریتم پایان می پذیرد.
- از گره های بسط داده شده بهترین k گره را انتخاب و الگوریتم تکرار می شود.

هوش مصنوعی

Artificial Intelligence

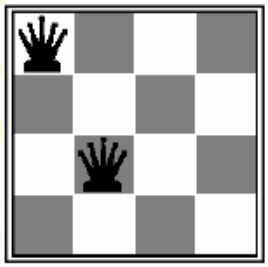
مسائل ارضاء محدودیت (CSP)
(CONSTRAINT SATISFACTION PROBLEM)

مسائل ارضاء محدودیت (CSP) (Constraint Satisfaction Problem)

- در مسائل CSP ، حالات توسط مقادیر مجموعه ای از متغیر ها مانند V_i تعریف می شود.
- آزمون هدف : مجموعه ای از محدودیت ها که مقادیر متغیرها باید از آن پیروی کنند.

- مسائل CSP می توانند توسط الگوریتمهای جستجوی همه منظوره (general purpose) حل شوند ولی به خاطر ساختار خاصشان، با استفاده از الگوریتمهای مخصوص خود کارایی بهتری خواهند داشت.

مثال CSP : مسئله ۴ وزیر



در هر ستون یک وزیر فرض کنید.

متغیرها : Q_1, Q_2, Q_3, Q_4

دامنه ها: $D_i=\{1, 2, 3, 4\}$

محدودیت ها:

$Q_i \neq Q_j$ (دو وزیر نمی توانند در یک سطر قرار بگیرند $Q_1 = 1 \quad Q_2 = 3$

$|Q_i - Q_j| \neq |i - j|$ (یا در یک قطر)

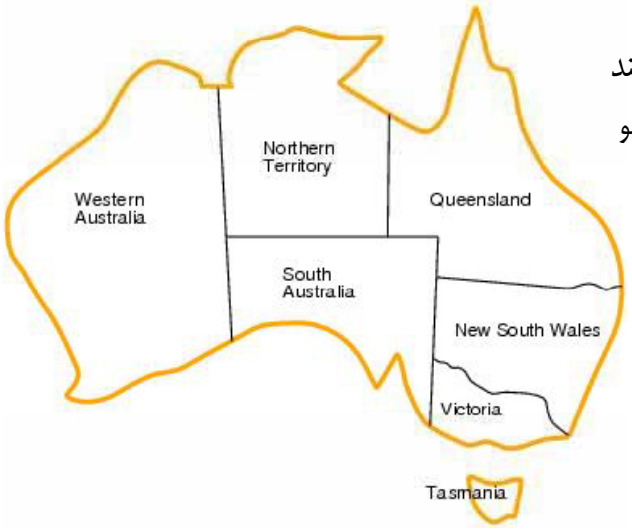
هر محدودیت به مجموعه ای از مقادیر مجاز برای متغیرهایش ترجمه می شود، مثلا:

مقادیر (Q1, Q2) می توانند مانند زیر باشند:

(1, 3), (1, 4), (2, 4), (3, 1), (4, 1), (4, 2)

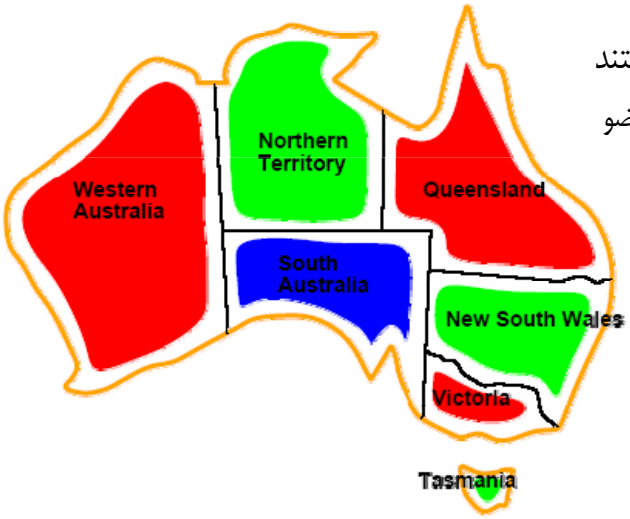
مثال CSP : رنگ آمیزی نقشه

- متغیرها: **WA, NT, Q, NSW, V, SA, T**
- دامنه: {آبی، سبز، قرمز} = **Di**
- محدودیتها: دو منطقه مجاور، همرنگ نیستند
  - مثال: **WA ≠ NT** یعنی **(WA,NT)** عضو
  - {(قرمز،سبز)،(قرمز،آبی)،(سبز،قرمز)، (سبز،آبی)،(آبی،قرمز)،(آبی،سبز)}



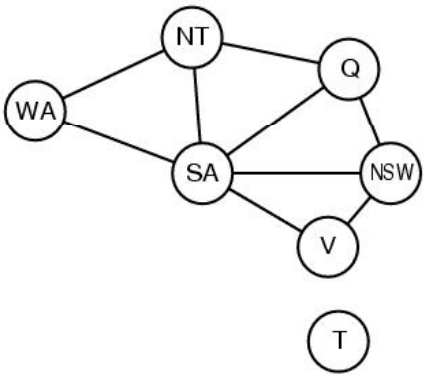
مثال CSP : رنگ آمیزی نقشه

- متغیرها: **WA, NT, Q, NSW, V, SA, T**
- دامنه: **Di** = {آبی، سبز، قرمز}
- محدودیتها: دو منطقه مجاور، همرنگ نیستند
  - مثال: **WA ≠ NT** یعنی **(WA,NT)** عضو
  - {(قرمز،سبز)،(قرمز،آبی)،(سبز،قرمز)، (سبز،آبی)،(آبی،قرمز)،(آبی،سبز)}



گراف محدودیت

- گراف محدودیت
- در گراف محدودیت:
  - گره ها: متغیرها
  - یالها: محدودیتها
- گراف محدودیت برای ساده تر کردن جست و جو بکار میرود



مثال : با استفاده از گراف محدودیت رنگ آمیزی نقشه تبدیل به رنگ آمیزی گراف خواهد شد.

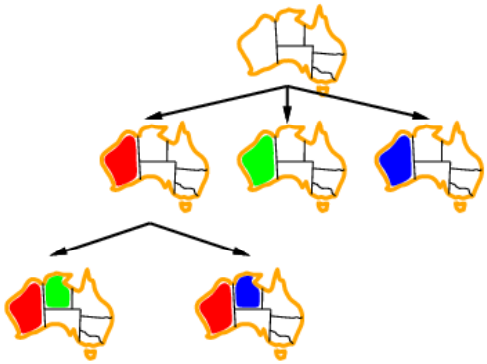
جست و جوی عقبگرد برای CSP

- در حالت آغازین مقادیر هیچ یک از متغیر ها انتساب داده نشده است
- انتساب مقدار یک متغیر در هر قدم و عقبگرد در صورت عدم وجود مقداری معتبر برای انتساب به متغیر
- یک جست و جوی عمقی
- یک الگوریتم ناآگاهانه
  - برای مسئله های بزرگ کارآمد نیست
-

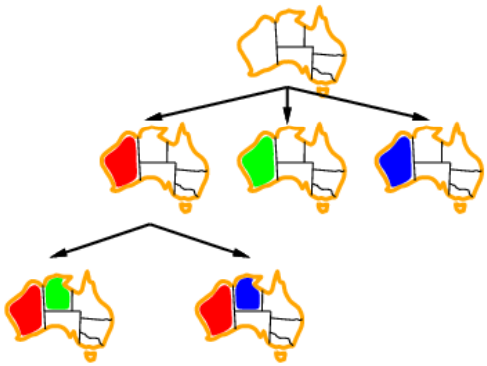
جست و جوی عقبگرد برای CSP

جست و جوی عقبگرد برای CSP

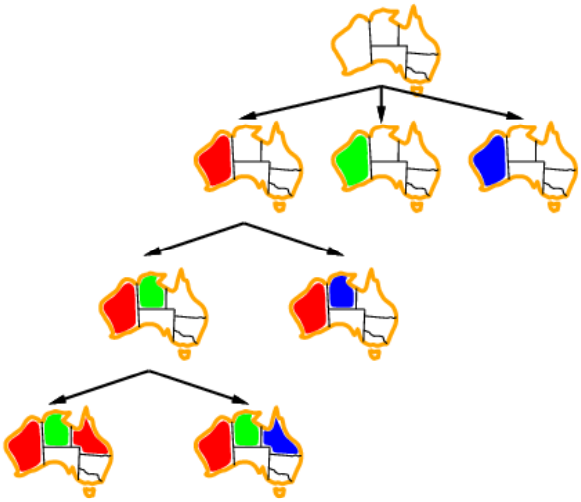
جست و جوی عقبگرد برای CSP



جست و جوی عقبگرد برای CSP



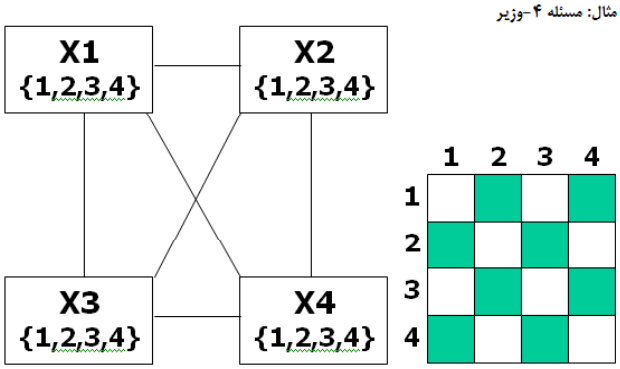
جست و جوی عقبگرد برای CSP



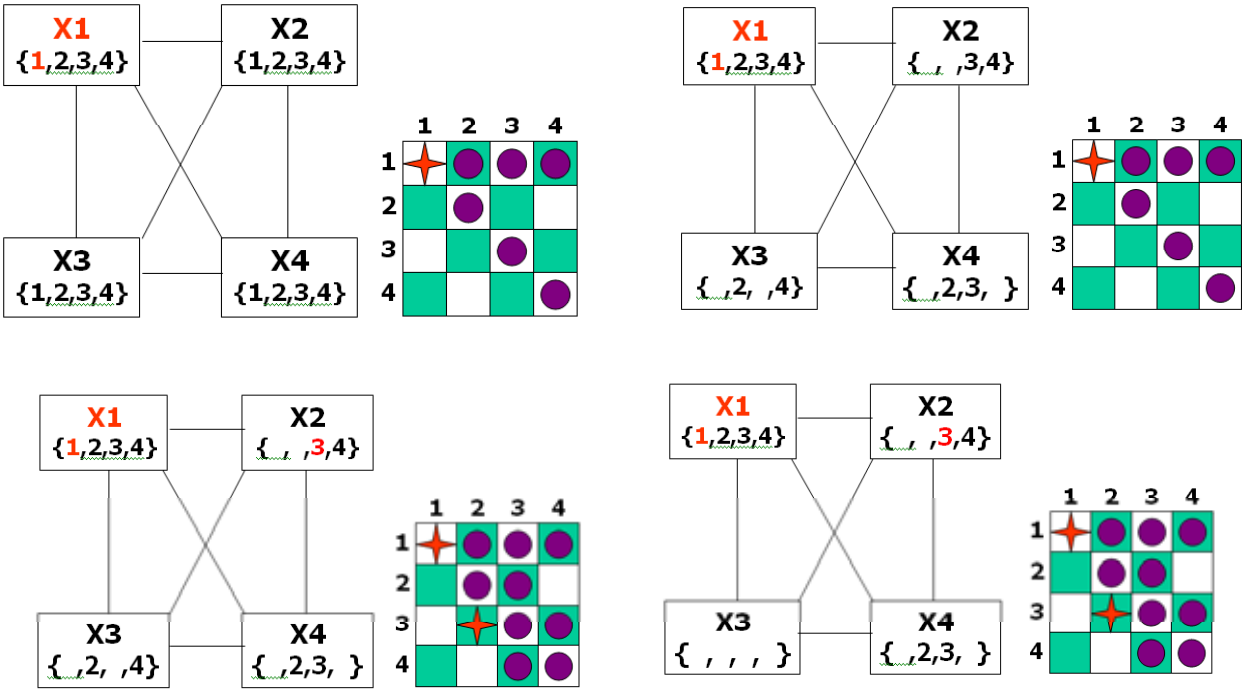
بررسی پیشرو (Forward Checking)

- وقتی انتساب به **X1** صورت میگیرد، فرایند بررسی پیشرو، متغیرهای بدون انتساب، مثل **X2** را در نظر میگیرد که از طریق یک محدودیت به **X1** متصل است و هر مقداری را که با مقدار انتخاب شده برای **X1** برابر است، از دامنه **X2** حذف میکند

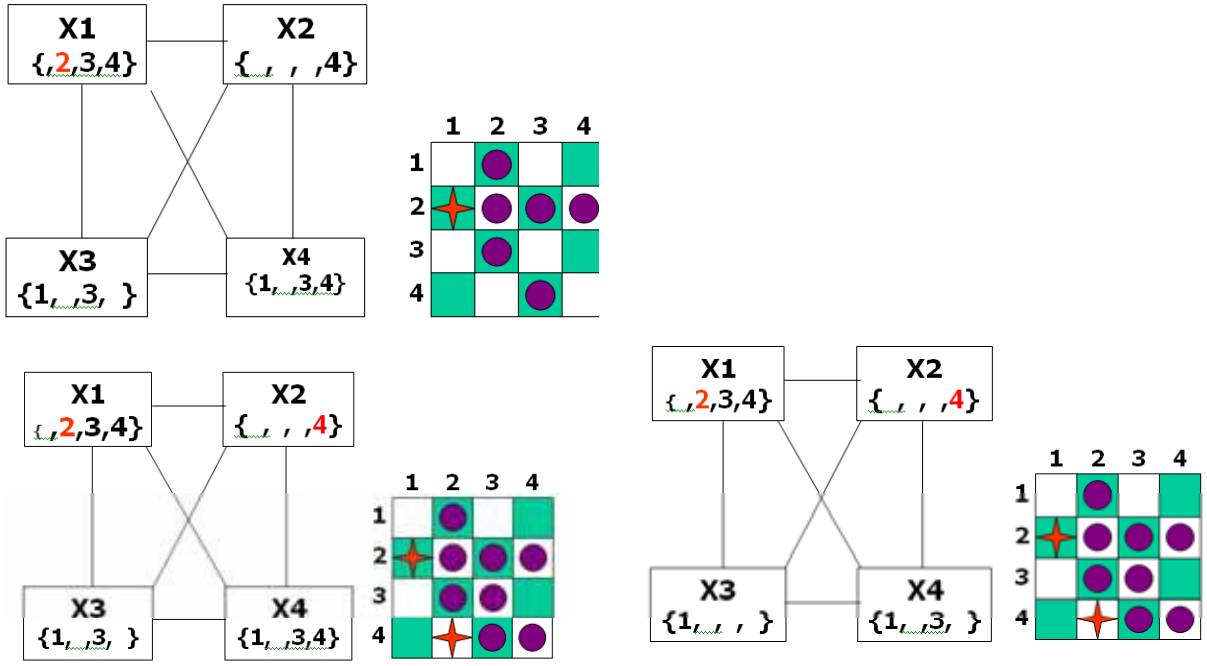
مثال: مسئله ۴-وزیر



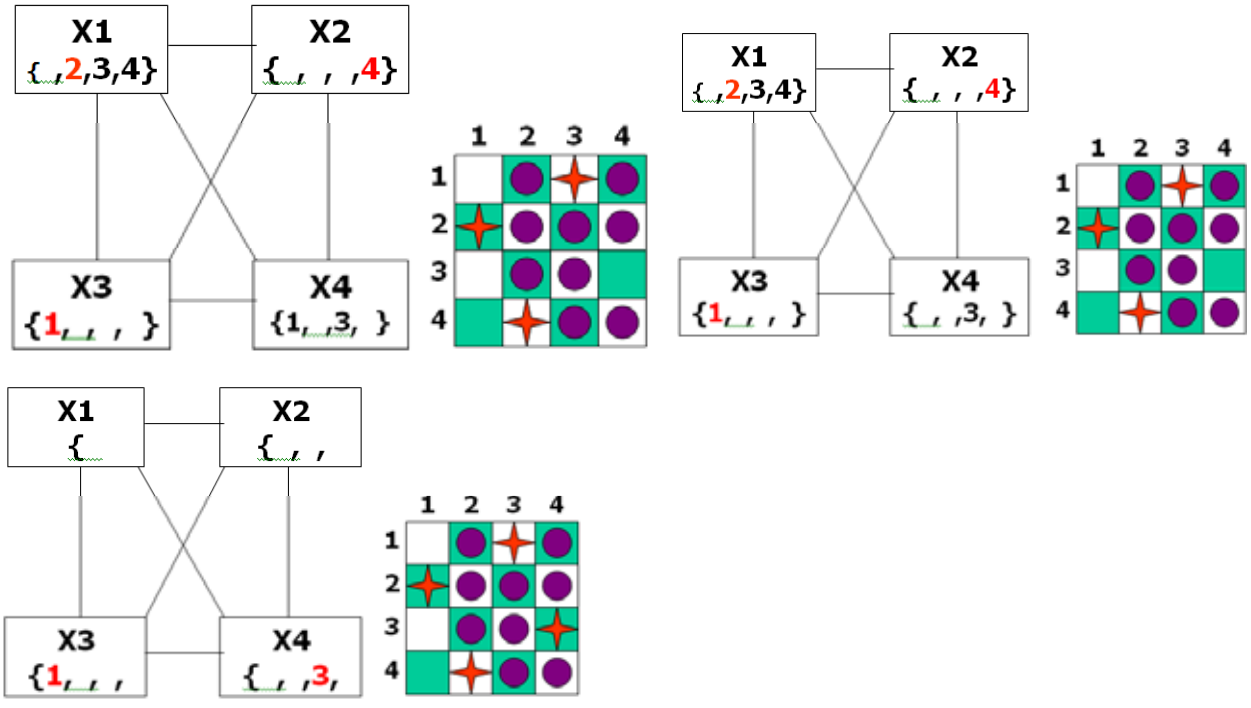
بررسی پیشرو (Forward Checking)



بررسی پیشرو (Forward Checking)



بررسی پیشرو (Forward Checking)



استفاده از الگوریتمهای تکراری در مسائل CSP

- تپه نوردی و SA با حالات کامل کار میکنند، یعنی تمام متغیرها باید دارای مقدار باشند.
- برای اعمال الگوریتمهای تکراری بر روی مسائل CSP :
  - داشتن حالاتی با محدودیتهای نقض شده باید مجاز باشد.
  - عملگرها باید بتوانند به متغیرها مقادیر جدید بدهند.
  - هیوریستیک : تعداد محدودیتهای نقض شده
    - (استفاده از هیوریستیک ها در مسائل CSP موجب اتخاذ تصمیمات هوشمندانه تر خواهد شد)
  - انتخاب متغیر : مقادیری را انتخاب کن که کمترین تعداد محدودیت ها را نقض می کنند.

استفاده از الگوریتمهای تکراری در مسائل CSP

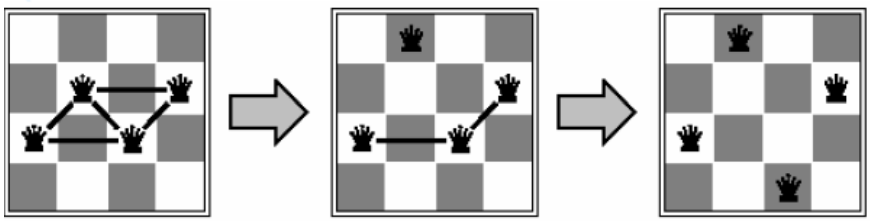
مثال: ۴-وزیر

حالات: ۴ وزیر در ۴ ستون $(4^4=256)$

عملگرها: حرکت وزیرها در یک ستون

تست هدف: هیچ یک از وزیرها همدیگر را تهدید نکنند.

تابع ارزیابی $h(n)$: تعداد تهدیدها



هوش مصنوعی

Artificial Intelligence

فصل پنجم - تئوری بازی

فهرست

- بازیها چیستند و چرا مطالعه میشوند؟
- انواع بازیها
- الگوریتم minimax
- بازیهای چند نفره
- هرس آلفا-بتا
- بازیهای قطعی با اطلاعات ناقص
- بازیهایی که حاوی عنصر شانس هستند

جستجوی خصمانه

بازی ها چیستند و چرا مطالعه میشوند؟

- بازیها حالتی از محیطهای چند عاملی هستند
  - هر عامل نیاز به در نظر گرفتن سایر عاملها و چگونگی تأثیر آنها دارد
  - تمایز بین محیطهای چند عامل رقابتی و همکار
  - محیطهای رقابتی، که در آنها اهداف عاملها با یکدیگر برخورد دارند، منجر به مسئله های خصمانه میشود که به عنوان بازی شناخته میشوند
- چرا مطالعه میشوند؟
  - قابلیتهای هوشمندی انسانها را به کار میگیرند
  - حالت بازی را به راحتی میتوان نمایش داد و عاملها معمولا به مجموعه کوچکی از فعالیتها محدود هستند که نتایج آنها با قوانین دقیقی تعریف شده اند

دلیل بررسی بازیها (ادامه)

- ماهیت جذاب و سرگرم کننده بازیها
- سادگی مدلسازی بازیها در فضای وضعیتها و مجموعه ای از عملیات
- معیارهای عملی و واضح برای ارزیابی میزان موفقیتAI
- مطالعه و بررسی عاملهای رقیب و دشمن
- وجود بازیهای سخت و جالب با فضای حالتهای وسیع
  - بازی شطرنج با درخت جستجو شامل $۳۵^{۱۰۰}$ گره

(فاکتور انشعاب به صورت میانگین ۳۵ بوده و هر بازی معمولا تا ۵۰ حرکت به ازای هر بازیکن ادامه می یابد.)

تئوری بازیها

- بازیهای کامل (Perfect Game)
  - بازیهای دو نفره
  - بازیکنان به نوبت بازی می کنند .
  - قانون مجموع صفر (Zero-sum) : اشتباه یکی به نفع دیگری است
  - هر بازیکن اطلاعات کامل در مورد محیط و حالت بازی دارد و هیچ اطلاعی از دید بازیکن مخفی نیست .(محیط دسترس پذیر)
  - عناصر شانس (استفاده از تاس، ...) دخالت ندارند .
  - حرکات بطور واضح و مشخص تعریف شده اند .
  - هر حرکتی نتیجه مشخصی دارد .(محیط قطعی)
  - مثال: شطرنج، Tic-Tac-Toe
- بازیهای غیر کامل (Imperfect games)
- مثال: فال ورق،

تئوری بازیها

- یک بازی با اجزای زیر قابل تعریف است:
  - حالت شروع شامل وضعیت اولیه به همراه بازیکنی که باید بازی را شروع کند.
  - یک تابع (Succesorfunction)که لیستی شامل زوجهای(*move, state*) متشکل از حرکات قانونی قابل اجرا و نتیجه هر حرکت *را ارائه می دهد* .
  - تابع تست هدف که خاتمه بازی را شناسائی می کند .
  - یک تابع کمکی (Utility/Objective function) که امتیازات دو طرف بازی را ارزیابی می کند. برای هر حالت پایانه یک مقدار عددی را ارائه میکند. مثلا برنده(۱+) و بازنده(۱-)
- حالت اولیه و حرکات معتبر برای هر بازیکن، **درخت بازی** را برای آن بازی ایجاد میکند

چگونگی انجام یک بازی

- تمام حرکات قانونی ممکن را لیست کن .
- حالت بعدی حاصل از انجام هر یک از حرکات فوق را مشخص کن .
- هر یک از حالتهای بعدی ممکن را ارزیابی کرده و بهترین حرکت را انتخاب کن.
- حرکت انتخاب شده را انجام بده .
- صبر کن حریف حرکتش را انجام دهد، مراحل بالا را دوباره تکرار کن .
- مشکلات :
  - نمایش صفحه بازی
  - تولید تمام موقعیت های قانونی برای حرکت بعدی
  - ارزیابی مناسب بودن یک موقعیت
  - پیش بینی

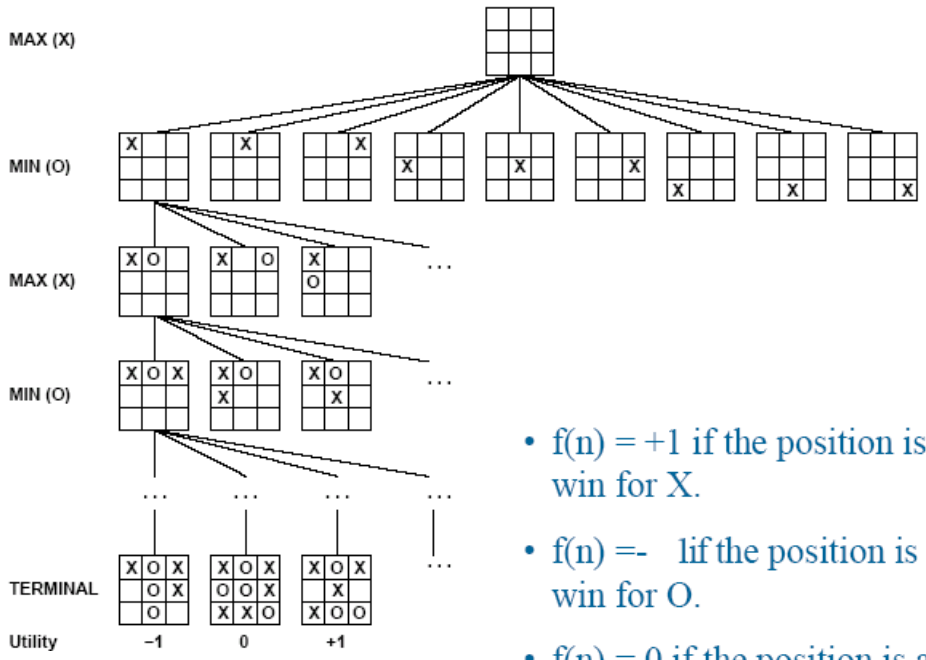
درخت بازی

- استفاده از درخت برای نمایش فضای مسئله برای بازیها اغلب مناسب است.
- گره ریشه شامل حالت شروع بازی می باشد .
- برای هر گره شامل وضعیت جاری، باید تصمیمی برای انتخاب بهترین حرکت بعدی اتخاذ شود .
- هر حرکت قانونی توسط یک شاخه از درخت نشان داده می شو د .
- با استفاده از یک تابع ارزیابی، یک وضعیت از بازی ارزش گذاری می شود.
- گره های برگ، وضعیت های نهائی بازی را نشان می دهند که در اینجا می تواند یکی از مقادیر برد، مساوی و یا باخت باشد .

یک نمونه بازی دو نفره (Tic-Tac-Toe)

- بازی دو نفره: Min و Max
  - اول Max حرکت میکند و سپس به نوبت بازی میکنند تا بازی تمام شود
  - در پایان بازی، برنده جایزه و بازنده جریمه میشود

یک نمونه بازی



- f(n) = +1 if the position is a win for X.
- f(n) =- lif the position is a win for O.
- f(n) = 0 if the position is a draw.

الگوریتم:

- **بازیکن:** انتخاب **بهترین** حالت
- **حریف:** انتخاب بهترین موقعیت برای خودش یا **بدترین وضعیت** برای بازیکن

تابع ارزیابی

- به منظور ارزیابی میزان خوب بودن یک گره، تابع ارزیابی استفاده می شود . (تابع هیوریستیک)
- با استفاده از قانون مجموع صفر (Zero-Sum) می توان از یک تابع برای هر دو بازیکن به منظور ارزیابی موقعیت بازی بهره برد :
- $F(n)>0$ موقعیت n برای من خوب و برای حریف بد است.
- $F(n)<0$ موقعیت n برای منبدو برای حریف خوب است.
- $F(n)=0$ موقعیت nیک وضعیت خنثی است.
- $F(n)>>0$ برد من
- $F(n)<<0$ برد حریف

تابع ارزیابی

- تابع ارزیابی برای بازی Tic-Tac-Toe
- *f(n)=[Number of 3-lengths open for me]*
  – [Number of 3-lengths open for you]

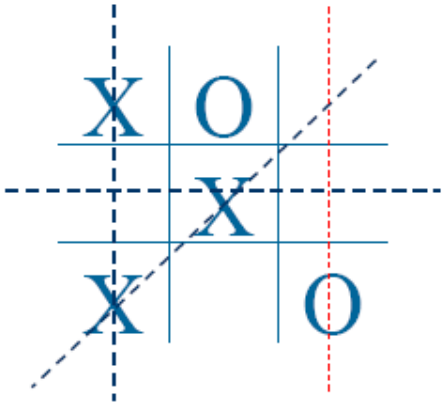
- *تابع ارزیابی در بازی شطرنج اغلب به صورت مجموع وزن دار از امتیازات موقعیت مهره ها بیان می شود :*

امتیازات موقعیت مهره ها مانند تعداد مهره ها، نحوه چیدن مهره ها، تعداد خانه های تحت پوشش و . . . می باشد.

$$f(n) = w_1 * feat_1(n) + w_2 * feat_2(n) + \ldots w_k * feat_k(n)$$

تابع ارزیابی

- *f(n)=[Number of 3-lengths open for me]*
  - *[Number of 3-lengths open for you]*



قانون MiniMax

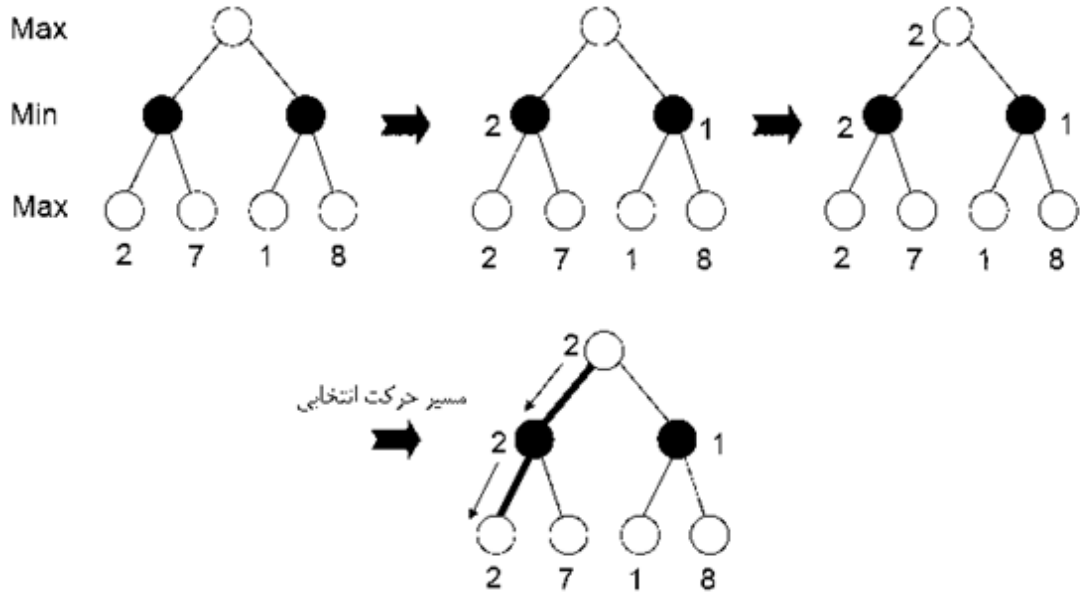
- هدف از جستجو در درخت بازی، تعیین حرکتی(نه یک راه حل) برای بازیکن Max بطوریکه امتیاز او را (بدون توجه به حرکات Min)ماکزیمم کند.
- پیچیدگی زمانی : $O(b^m)$ (چرا که شبیه جستجوی عمقی کامل می باشد)
- پیچیدگی فضایی : از مرتبه خطی ($O(bm)$ یا $O(m)$)
- بازیکن Maxهمواره فرض می کند که بازیکنMin حرکتی را انجام می دهد که امتیازش را ماکزیمم کند و امتیاز Max را مینیمم کند.

- امتیاز هر گره از طریق امتیاز فرزندانش مشخص می شود :
  - مقدار گره Max برابر ماکزیمم مقدار فرزندانش است.
  - مقدار گره Min برابر مینیمم مقدار فرزندانش است.

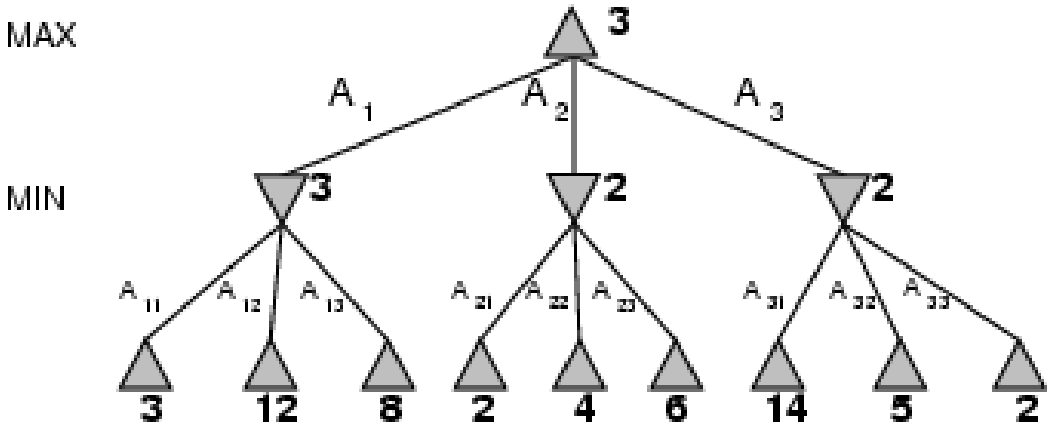
الگوریتم MiniMax

- گره شروع بازی را با برچسب Max ایجاد کن
- گره های سطوح بعدی را تا یک حد معین بسط بده .
- تابع ارزیابی را به ازای هر گره برگ محاسبه کن .
- برای گره های غیر برگ تا رسیدن به گره ریشه، با استفاده از قانون Minimax یک مقدار انتساب بده :
- مقدار گره Max برابر ماکزیمم مقدار فرزندانش است.
- مقدار گره Min برابر مینیمم مقدار فرزندانش است.
- حرکتی را از گره جاری انتخاب کن که به بهترین گره برگ با ماکزیمم مقدار برسد

جستجوی Minimax



جستجوی Minimax

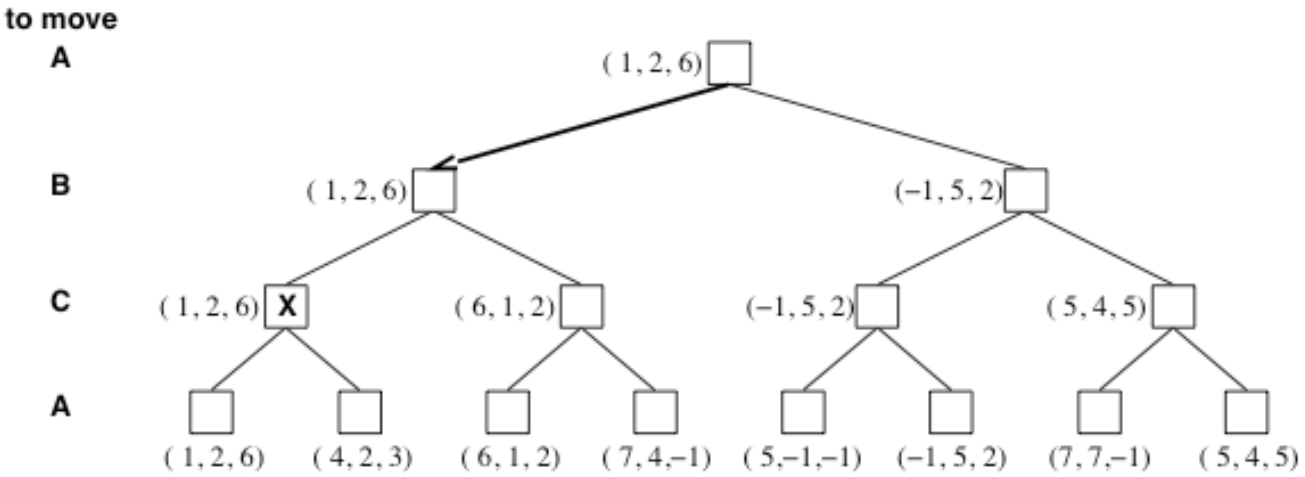


الگوریتم Minimax

- بازیهای چند نفره :
- در بازی دو نفره دو بازیکن وجود داشت و دو مقدار تولید شده توسط تابع سودمندی که با توجه متضاد بودن این نتایج توانستیم آنها را به یک مقدار در هر نوبت تبدیل کنیم.
- استفاده از یک بردار برای نمایش وضعیت بازی به جای یک مقدار
- در بازی چند نفره تابع سودمندی مقادیر مختلفی را در این بردار به ازای هر برگ تولید می کند که هر یک از این مقادیر از دید بازیکن خاصی می باشد.
- بازیکنان بطور رسمی یا غیر رسمی با همدیگر در پیشبرد بازی همکاری می کنند

الگوریتم Minimax

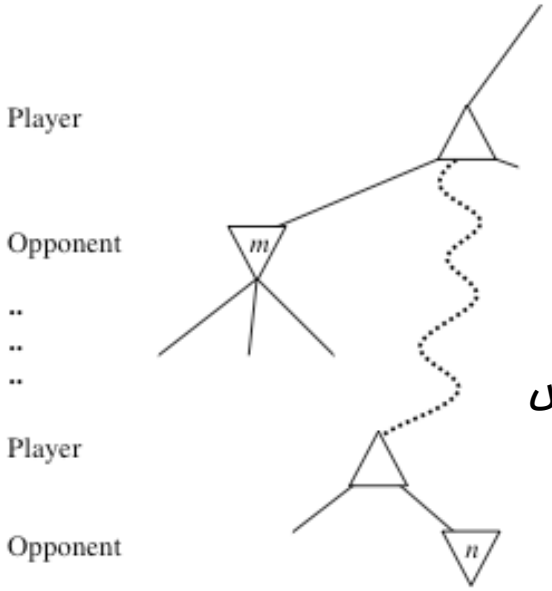
- مثال: درخت Minimax بازی بازی سه نفره



هرس آلفا بتا (Alpha-Beta Pruning)

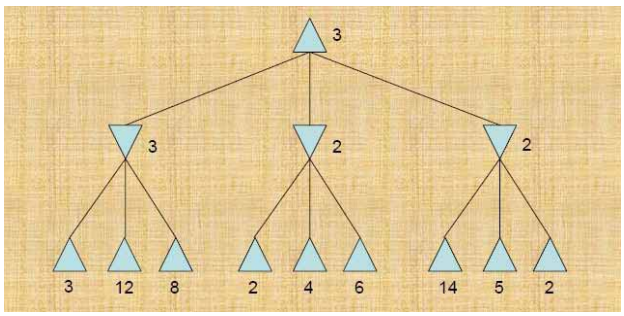
- در جستجوی Minimax تعداد گره های درخت جستجو به صورت نمائی افزایش می یابد.
- •برای حل مشکل، راه حل پیشنهادی عدم بسط گره هایی است که مقدار آنها تاثیری در تصمیم نهائی ندارد .
- برای این منظور، کارآیی الگوریتم Minimax را می توان با استفاده از هرس آلفا – بتا افزایش داد.

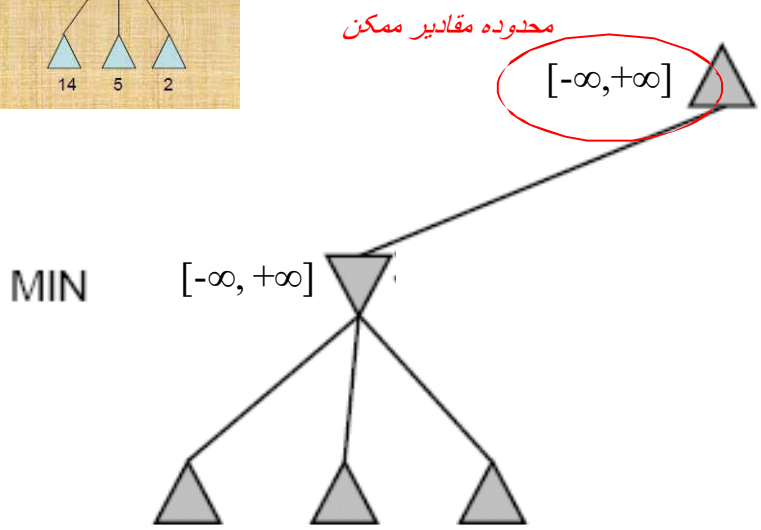
هرس آلفا-بتا



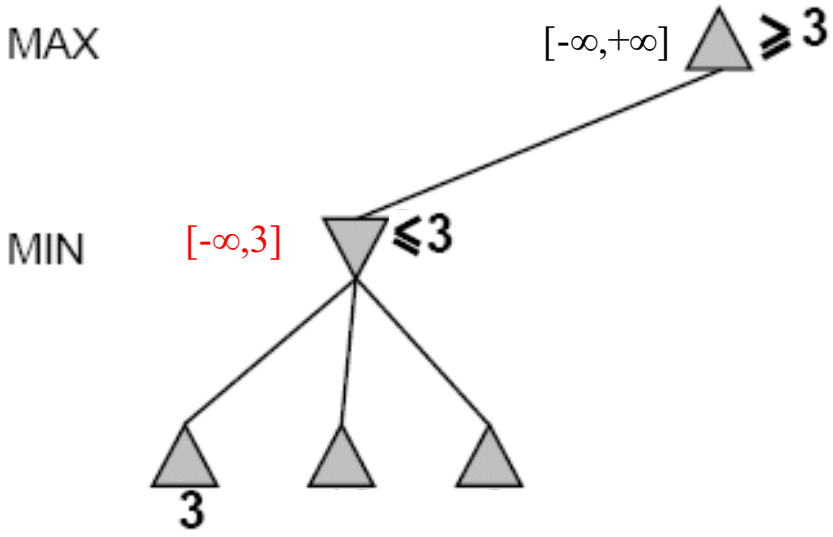
- گره n که هر جای درخت میتواند باشد، بررسی میشود
- اگر بازیکن انتخاب بهتری داشته باشد
  - در گره والد n
  - یا هر انتخاب بهتری تا کنون
- n هیچوقت در بازی واقعی قابل دسترس نخواهد بود
- در نتیجه n هرس میشود

مثال: هرس آلفا-بتا

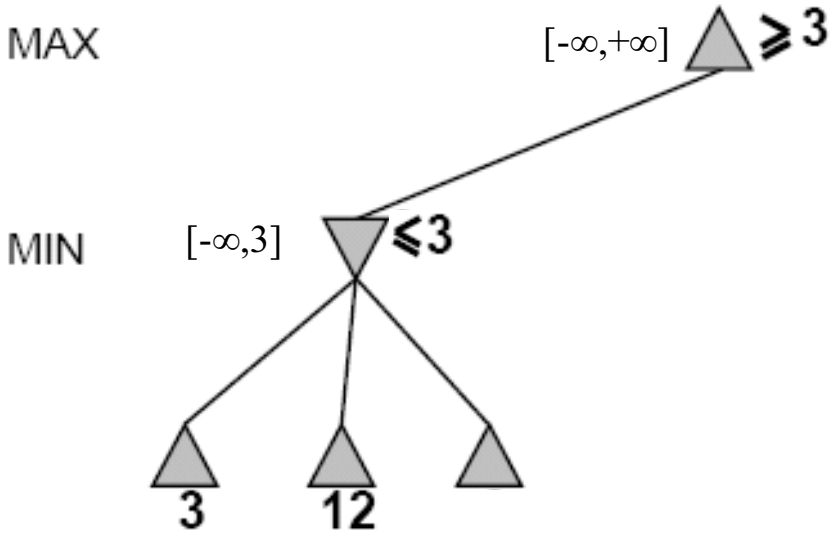




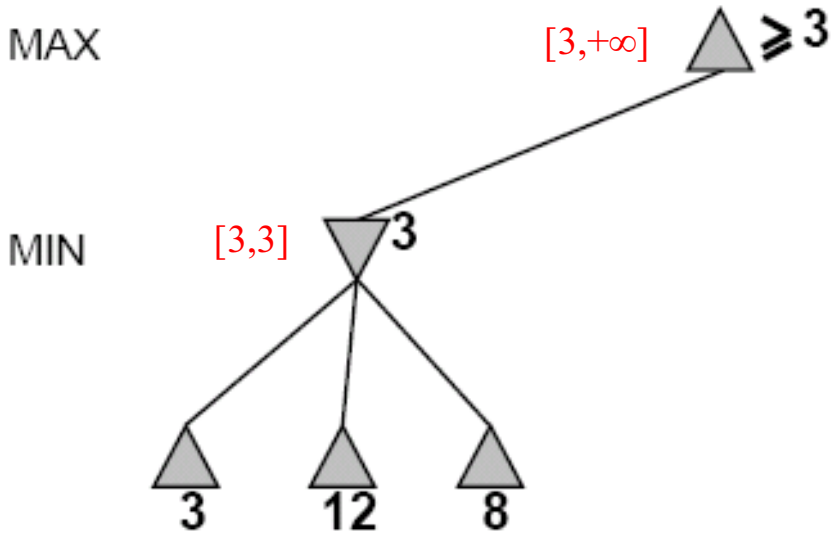
مثال: هرس آلفا-بتا



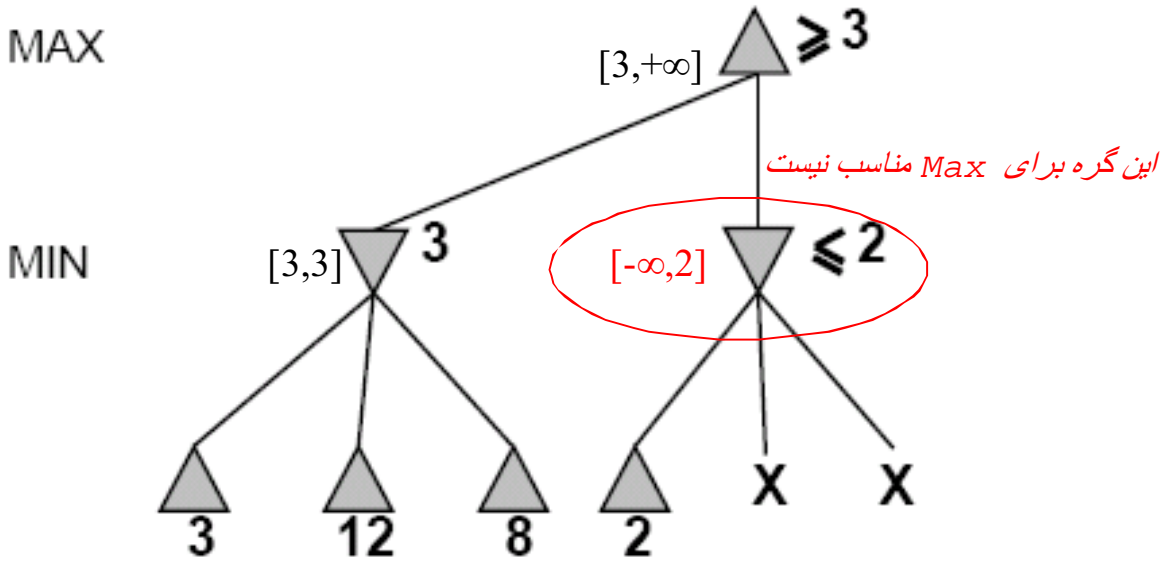
مثال: هرس آلفا-بتا



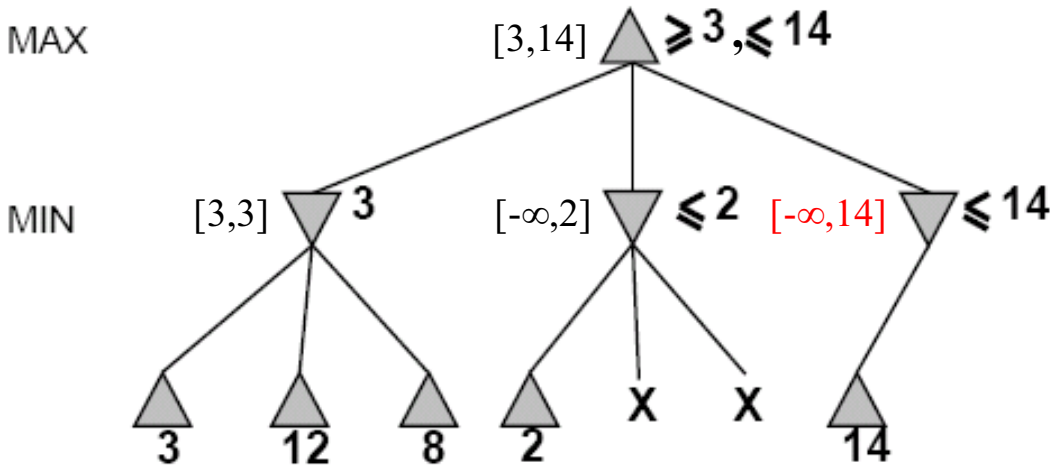
مثال: هرس آلفا-بتا



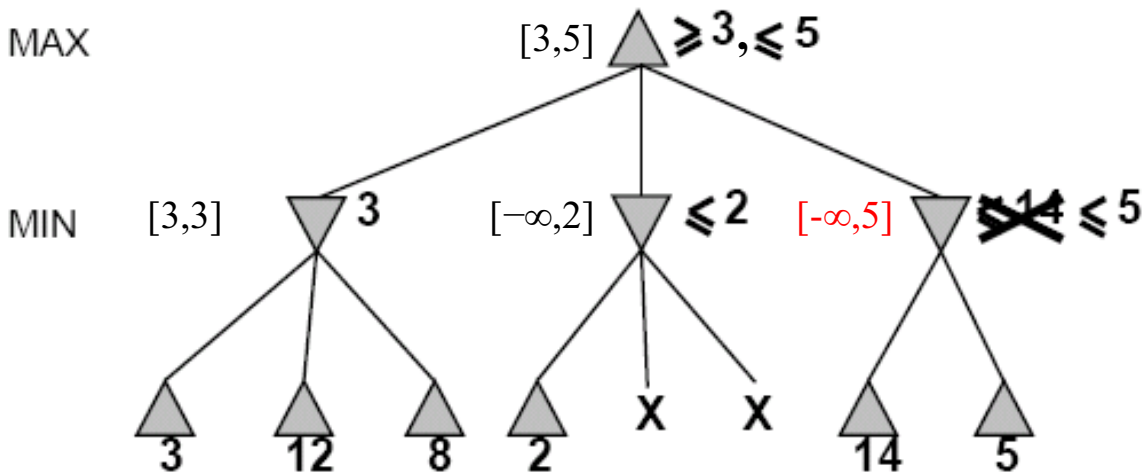
مثال: هرس آلفا-بتا



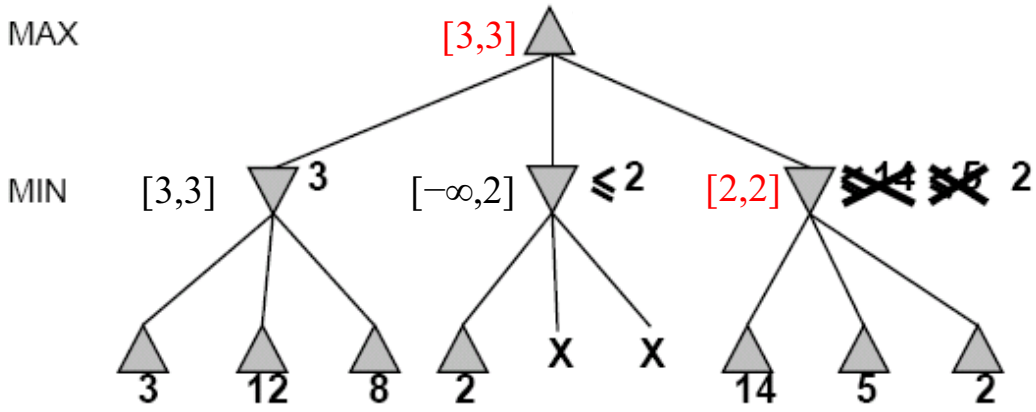
مثال: هرس آلفا-بتا



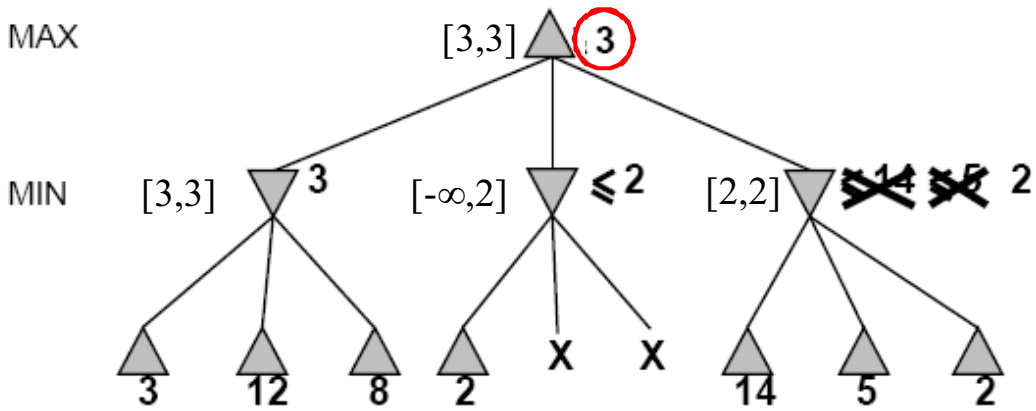
مثال: هرس آلفا-بتا



مثال: هرس آلفا-بتا



مثال: هرس آلفا-بتا



ویژگیهای هرس آلفا

- در بهترین حالت $O(b^{m/2})$ گره مورد بررسی قرار می گیرد.

- بهترین حالت زمانی رخ می دهد که بهترین حرکت هر بازیکن، سمت چپ ترین حرکت در درخت بازی باشد . به عبارت دیگر در گره های Max و در گره های Min فرزند سمت چپ کم ترین مقدار را داشته باشد.

- فاکتور انشعاب موثر به جای b برابر با جذر b خواهد بود.
- اگر حرکتها تصادفی انتخاب شود (به جای بهترین انتخاب)$O(b^{3m/4})$

بازیهای قطعی با اطلاعات ناقص

معایب الگوریتم های پیشین

- الگوریتم minimax کل فضای جست و جوی بازی را تولید میکند
- الگوریتم آلفا-بتا اگرچه بخشی از گره های فضای جستجو را هرس می کند، ولی هنوز لازم است مسیر های منتهی به حالتهای پایانی را برای درخت جستجو طی کند که لازمه این کار صرف زمان زیاد است.
  - این کار عملی نیست، زیرا حرکات باید در زمانی معقول انجام شود

شانون(۱۹۵۰)

برای کمتر شدن زمان جست و جو و اعمال تابع ارزیابی اکتشافی به حالتهای جستجو، بهتر است از گره های غیر پایانه به گره های پایانه پرداخته شود

تصمیم گیری ناقص و بلادرنگ

- برای بهبود دو روش Minimax و Alpha-Beta دو تابع زیر جایگزین می شود.
- تابع Eval جایگزین تابع سودمندی می شود. این تابع تخمینی از وضعیت جاری طرفین بازی را مشابه توابع هیوریستیک که پیش از این بررسی شد ارائه می دهد.
- تابع cutt-of –test جایگزین تابع terminal –test می شود که تشخیص می دهد کی تابع Eval اعمال شود.
- تابع cutt-of –test قبل از رسیدن به برگها ادامه بسط گره ها را خاتمه می دهد. که می تواند مثلا بارسیدن به محدوده عمقی خاصی جواب True برگرداند.
- محدوده عمقی می تواند بر اساس قوانین آن بازی و منطبق بر محدوده زمانی مورد نظر برای هر حرکت تعیین گردد.

تابع ارزیابی

- تفاوت با توابع هیوریستیک :
- توابع هیوریستیک تخمینی از فاصله جاری تا گره هدف را مشخص می کنند .
- تابع ارزیابی تخمینی از میزان سودمندی وضعیت جاری را مشخص می کند .
- تابع ارزیابی خصوصیات مختلفی از وضعیت جاری بازی را مورد بررسی و محاسبه قرار می دهند. برای این منظور ، اغلب مقادیر و امتیازات مختلفی برای هر خصوصیت تعریف و محاسبه شده و در نهایت باهم ترکیب می شوند.
- و محاسبه شده و در نهایت باهم تعریف مختلفی برای هر خصوصیت شوند می ترکیب .
- تابع ارزیابی توانائی تشخیص حالات بعدی را ندارد و تنها می تواند مقداری را محاسبه کند که میزان سودمندی آن وضعیت است .

تابع ارزیابی

- مثال: در بازی شطرنج تابع EVAL به صورت زیر محاسبه می شود:

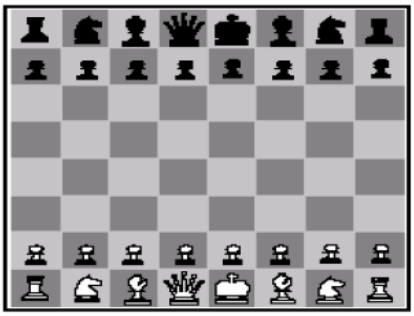
$$EVAL(n) = w_1 * f_1(n) + w_2 * f_2(n) + \ldots + w_k * f_k(n)$$

که در آن :

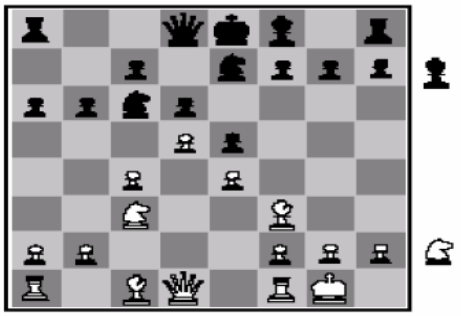
f_i : تعداد هر نوع قطعه در صفحه بازی

w_i : امتیاز آن قطعات (1 برای مهره پیاده، 3 برای مهره اسب و فیل، 5 برای مهره رخ، ...)

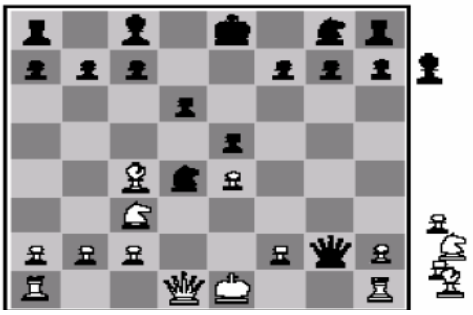
تابع ارزیابی

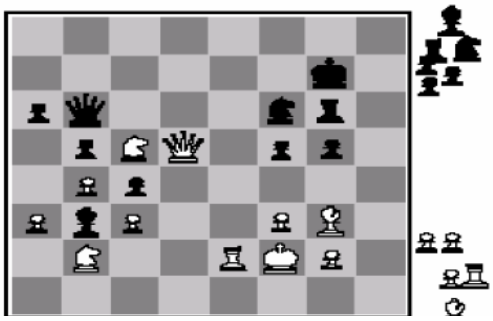


(a) White to move
Fairly even



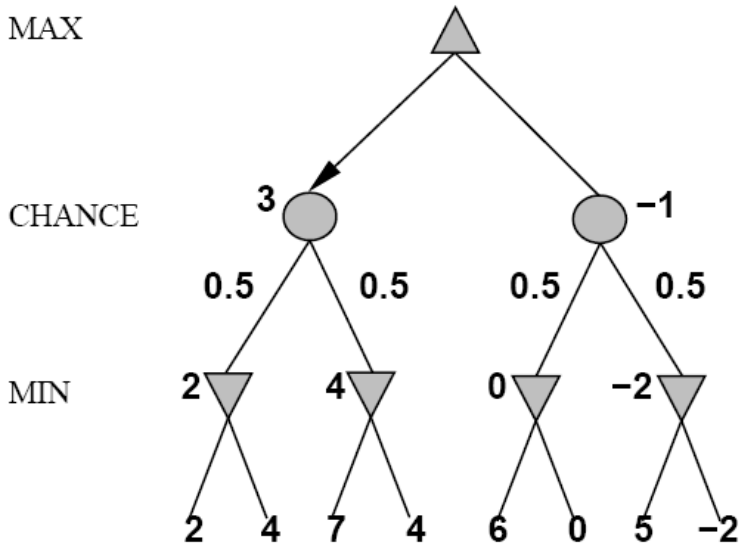
(b) Black to move
White slightly better





بازیهای همراه با شانس

- در بازیهای غیر قطعی ، شانس بوسیله پرتاپ تاس تولید می شود.
  - مثال ساده زیر با استفاده از پرتاپ سکه تولید شده است.



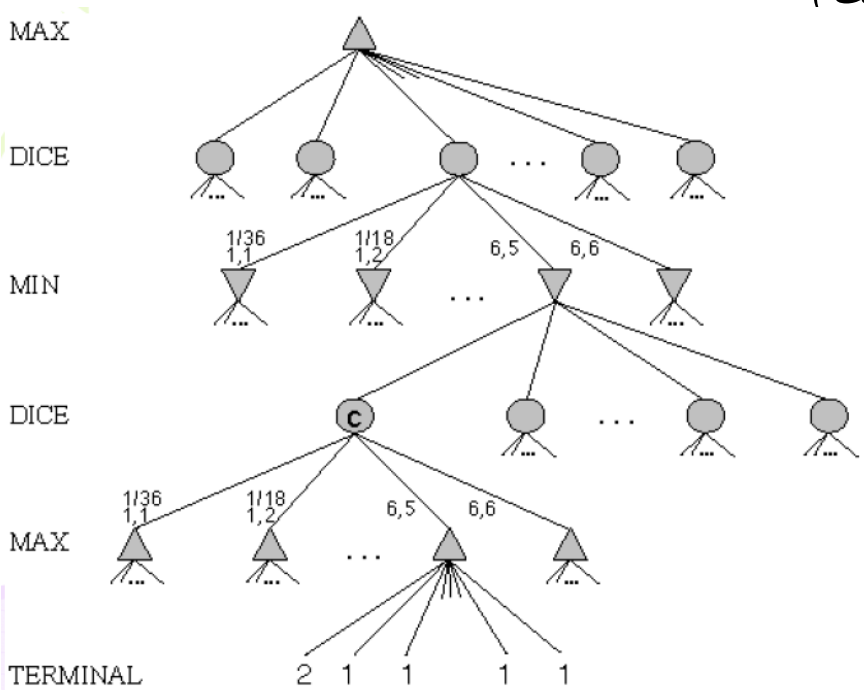
بازیهای همراه با شانس

- بدلیل اینکه در بازیهای همراه با شانس تمام دنباله‌های پرتاب تاس را در نظر می گیرد، زمانی معادل $O(b^m n^m)$ می‌برد، که n تعداد پرتابهای محدود است.

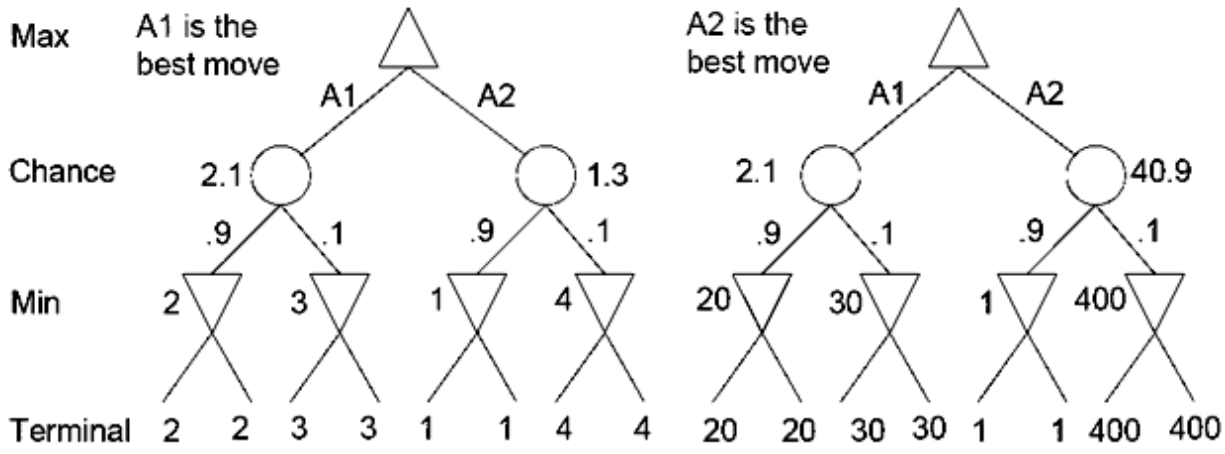
```
...
if state is a MAX node then
        return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a MIN node then
        return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a chance node then
        return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
...
```

بازیهای همراه با شانس

- در مثال زیر هر گره شانس ۲۱ فرزند با احتمال مربوط به خود را دارد
  - (پرتاپ دو تاس همزمان حالت)



بازیهای همراه با عنصر شانس



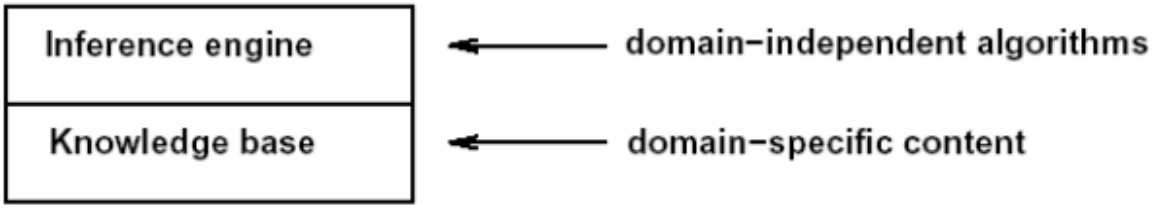
هوش مصنوعی

Artificial Intelligence

فصل ششم - عاملهایی که به طور منطقی استدلال می کنند

عاملهای مبتنی بر دانش

- هدف سیستمهای مبتنی بر دانش آن است که بتوانیم از دانش حل مسئله بطور کامل برای حل آن مسئله بهره جویی کنیم.
- یک سیستم مبتنی بر دانش از دو جزء اصلی تشکیل می شود.
  - پایگاه دانش (knowledge Base) که محل ذخیره سازی دانش مسئله است. که به صورت مجموعه ای از جملات (Sentence) در یک زبان رسمی (Formal) می باشند.
  - موتور استنتاج (Inference Engine) که قادر است از دانش موجود برای حل یک مسئله استفاده کند.



عاملهای مبتنی بر دانش

- عامل مبتنی بر دانش باید بتواند:
  - نمایش حالات و فعالیتها
  - ترکیب ادراکات جدید
  - بروز کردن تصور داخلی خود از جهان
  - استنباط خصوصیات مخفی جهان
  - استنتاج فعالیتهای مناسب
- عاملها در دو سطح متفاوت تعریف میشوند:
  - سطح دانش: عامل چه چیزی میداند و اهداف آن کدامند؟
  - سطح پیاده سازی: ساختمان داده اطلاعات پایگاه دانش و چگونگی دستکاری آنها

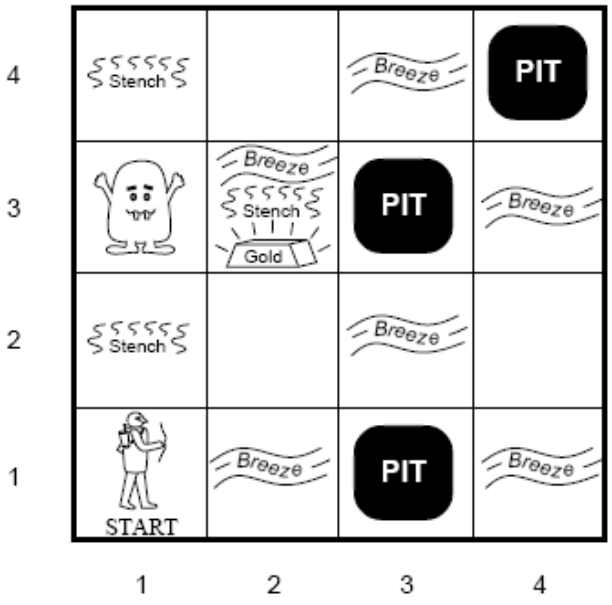
یک عمال ساده مبتنی بر دانش

- TELL : آنچه را عمل نیاز دارد به آن بگو
- ASK : عامل از خود می پرسد که با توجه به استنتاج از دانش موجود چه عملی را باید انجام دهم.

```
function KB-AGENT( percept) returns an action
    static: KB, a knowledge base
            t, a counter, initially 0, indicating time

    TELL( KB, MAKE-PERCEPT-SENTENCE( percept,t ))
    action ← ASK( KB, MAKE-ACTION-QUERY( t))
    TELL( KB, MAKE-ACTION-SENTENCE( action, t))
    t ← t + 1
    return action
```

جهان WUMPUS



معیار کارایی:

- ۱۰۰۰+ انتخاب طلا، ۱۰۰۰- افتادن در گودال یا خورده شدن، ۱- هر مرحله، ۱۰- برای استفاده از تیر

محیط:

- بوی تعفن در مربعهای همجوار WUMPUS
- نسیم در مربعهای همجوار گودال
- درخشش در مربع حاوی طلا
- کشته شدن WUMPUS با شلیک اگر عامل رو به وامپوس باشد
- تیر فقط مستقیم عمل میکند
- برداشتن و انداختن طلا

حسگرها:

- بوی تعفن، نسیم، تابش، ضربه، جیغ زدن

محرکها:

- گردش به چپ، گردش به راست، جلو رفتن، برداشتن، انداختن، شلیک کردن

توصیف جهان WUMPUS

قابل مشاهده کامل: خیر, فقط ادراک محلی

قطعی: بله، نتیجه دقیقا مشخص است

اپیزودیک: خیر، ترتیبی از فعالیتهاست

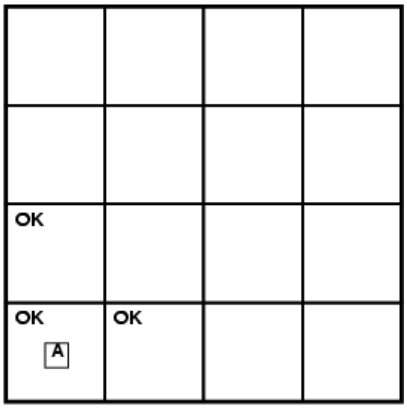
ایستا: بله, WUMPUS و گودالها حرکت ندارند

گسسته: بله

تک عامله: بله، WUMPUS در اصل یک خصوصیت طبیعی است

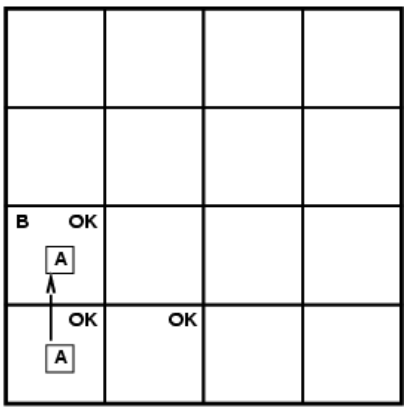
کاوش در جهان WUMPUS

A = عامل
B = نسیم
G = درخشش،طلا
OK = مربع امن
P = گودال
S = تعفن
V = ملاقات شده
W = Wumpus



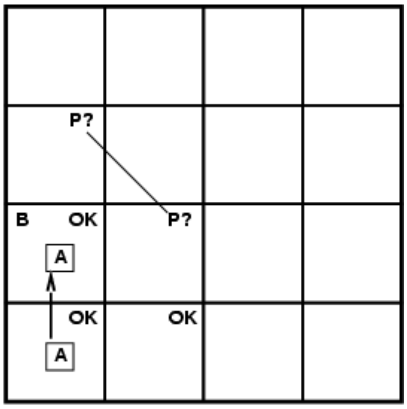
توصیف جهان WUMPUS

A = عامل
B = نسیم
G = درخشش،طلا
OK = مربع امن
P = گودال
S = تعفن
V = ملاقات شده
W = Wumpus



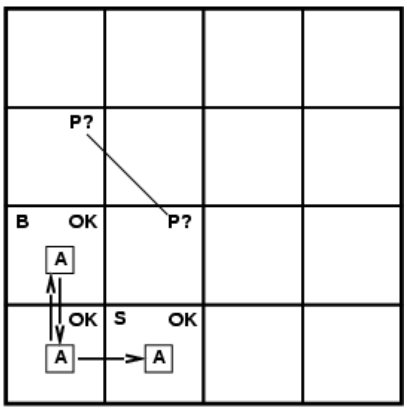
توصیف جهان WUMPUS

A = عامل
B = نسیم
G = درخشش،طلا
OK = مربع امن
P = گودال
S = تعفن
V = ملاقات شده
W = Wumpus



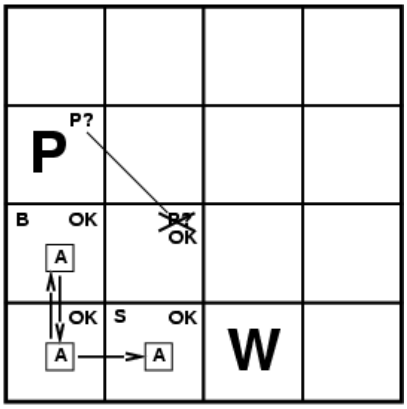
توصیف جهان WUMPUS

A = عامل
B = نسیم
G = درخشش،طلا
OK = مربع امن
P = گودال
S = تعفن
V = ملاقات شده
W = Wumpus



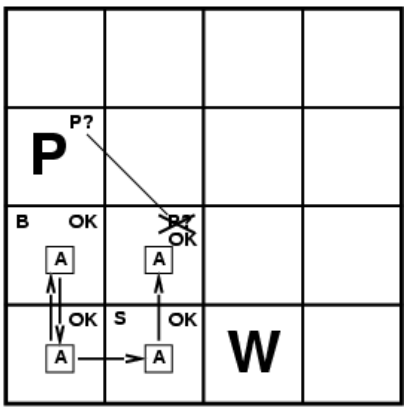
توصیف جهان WUMPUS

A = عامل
B = نسیم
G = درخشش،طلا
OK = مربع امن
P = گودال
S = تعفن
V = ملاقات شده
W = Wumpus



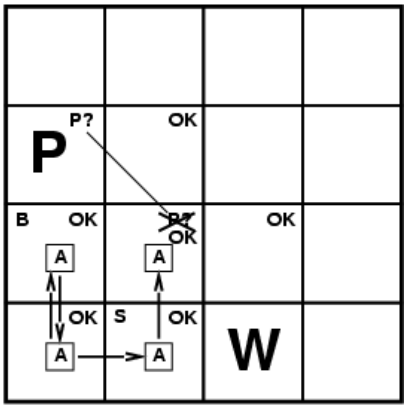
توصیف جهان WUMPUS

A = عامل
B = نسیم
G = درخشش،طلا
OK = مربع امن
P = گودال
S = تعفن
V = ملاقات شده
W = Wumpus



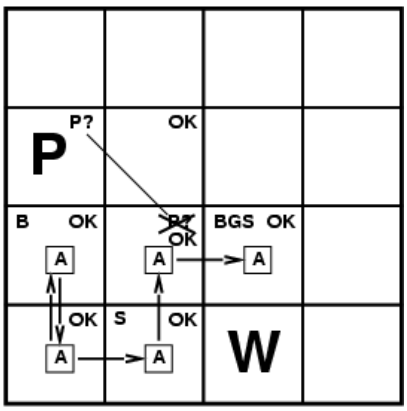
توصیف جهان WUMPUS

A = عامل
B = نسیم
G = درخشش،طلا
OK = مربع امن
P = گودال
S = تعفن
V = ملاقات شده
W = Wumpus

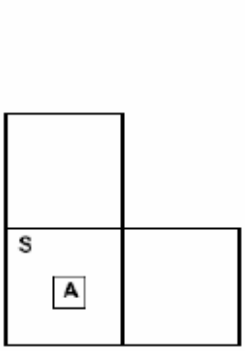


توصیف جهان WUMPUS

A = عامل
B = نسیم
G = درخشش،طلا
OK = مربع امن
P = گودال
S = تعفن
V = ملاقات شده
W = Wumpus



بررسی یک حالت خاص



احساس بو در (1, 1) ← قادر به حرکت نیست

استفاده از استراتژی اجبار:

۱) به خانه روبرو شلیک کن

۲) اگر وامپوس آنجا بوده ← مرده ← امن

۳) اگر وامپوس آنجا نبوده ← امن

منطق

- منطق : یک زبان رسمی برای بازنمایی دانش بطوریکه بتوان از آن نتیجه گیری نمود.
  - ترکیب(نحو)(Syntax): ساختار جملات زبان را معین می کند.
  - معنا (Semantic): یک جمله چه معنایی دارد
  - در منطق، معنای زبان، درستی هر جمله را در برابر هر جهان ممکن تعریف میکند
- مثال، در زبان ریاضیات
  - X+2 >= y یک جمله اما x2+y جمله نیست
  - X+2 >= y در جهان درست است اگر x=7 و y =1
  - X+2 >= y در جهان غلط است اگر x=0 و y =6

استلزام

- استلزام منطقی بین جملات این است که جمله ای بطور منطقی از جمله دیگر پیروی میکند

$KB \models \alpha$ یا $a \models b$

- جمله a استلزام جمله b است
- جمله a جمله b را ایجاد میکند
- اگر و فقط اگر، در هر مدلی که a درست است، b نیز درست است
- اگر a درست باشد، b نیز درست است
- درستی b در درستی a نهفته است

- **مثال: جمله** $x+y=4$ **مستلزم جمله** $4=x+y$ **است**

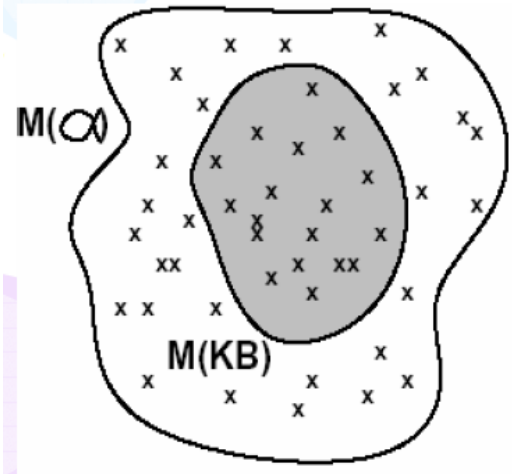
مدل ها

منطق دانان عموماً بر حسب مدل ها فکر می کنند، که بطور رسمی دنیاهای ساخت یافته ای می باشند که درستی را می توان نسبت به آنها ارزیابی کرد.

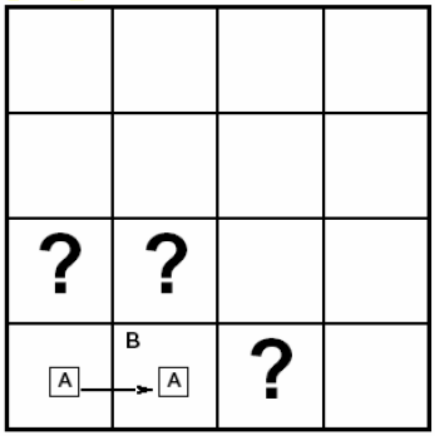
می گوییم m مدلی از جمله α می باشد اگر α در m درست باشد

$M(\alpha)$ مجموعه تمام مدل های α می باشد

بنابراین $KB \models \alpha$ اگر وفقط اگر $M(KB) \subseteq M(\alpha)$



استلزام در دنیای وامپوس

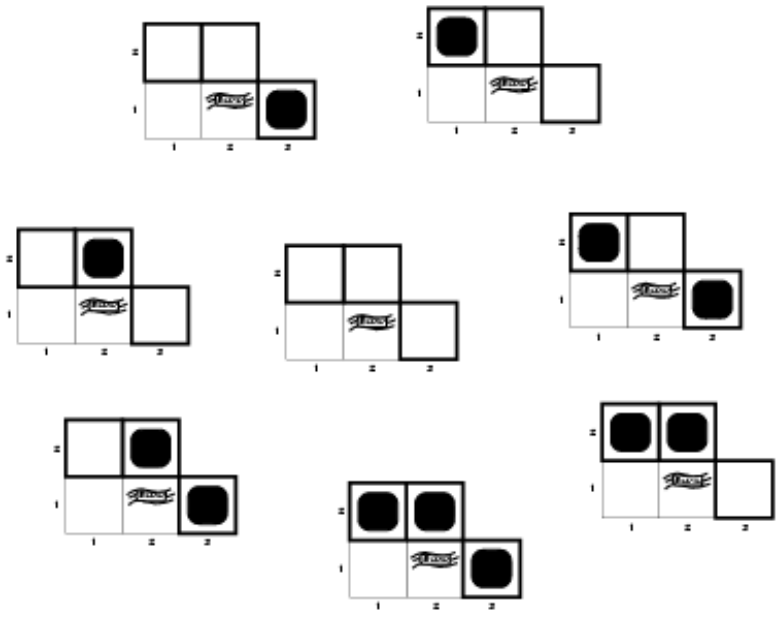


موقعیت پس از (1, 1)، رفتن به راست،
نسیم در (2, 1)

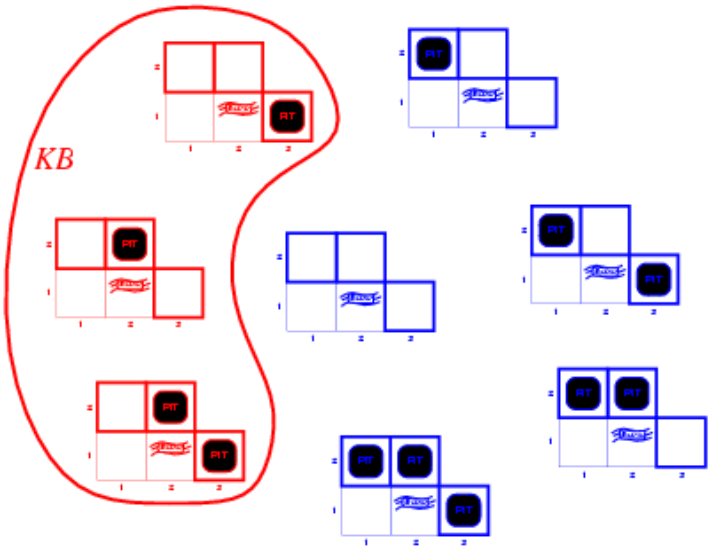
مدلهای ممکن برای ? ها را تنها با
فرض چاله ها در نظر بگیرید

سه انتخاب بولین ← هشت مدل مختلف

مدلهای Wumpus



مدلهای Wumpus



KB

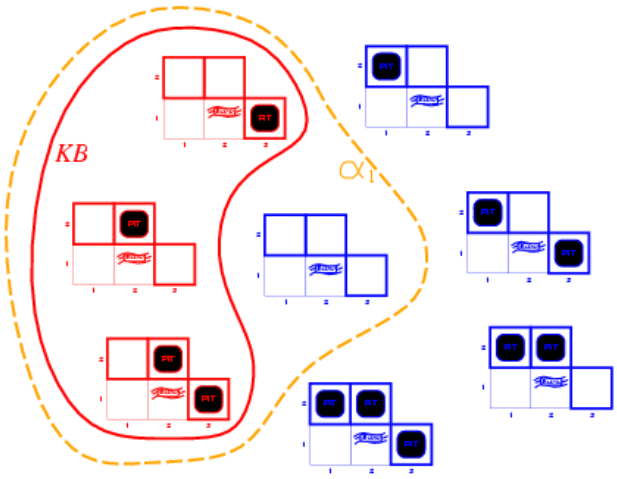
=

قوانین دنیای Wumpus

+

مشاهدات

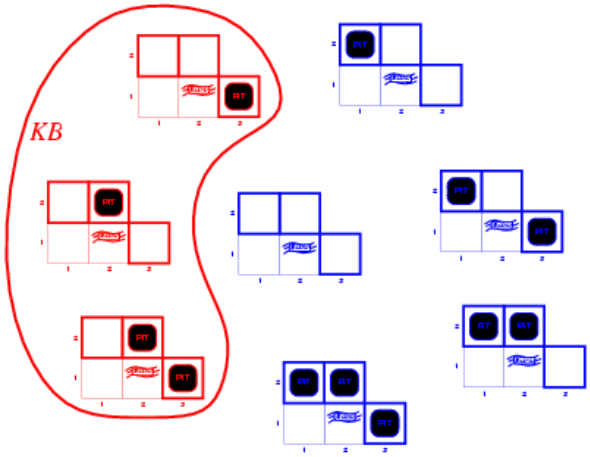
مدلهای Wumpus



KB = wumpus دنیای + مشاهدات

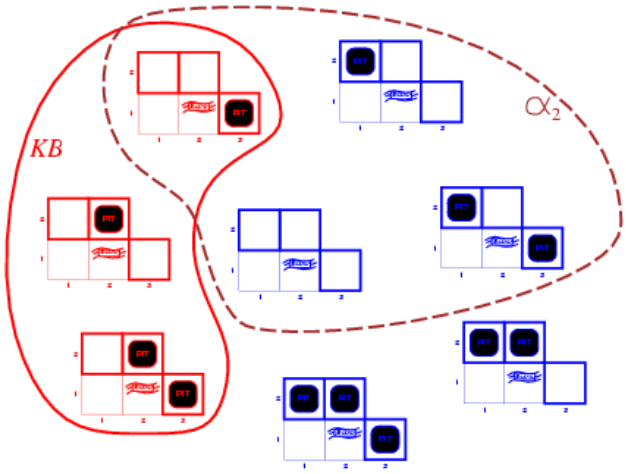
α1 = "[1,2] امن است", $KB \models \alpha 1$

مدلهای Wumpus



KB = wumpus دنیای + مشاهدات

مدلهای Wumpus



KB = wumpus دنیای + مشاهدات

α_2 = "[2,2] امن است", $KB \not\models \alpha_2$

استنتاج (Inference)

$KB \vdash_i \alpha$ = جمله α بوسیله رویه i از KB قابل اشتقاق می باشد.

نتایج KB مانند یک انبار کاه می باشد، و α مانند یک سوزن

استلزام = سوزن در انبار کاه؛ استنتاج = یافتن سوزن

صحت (soundness): رویه i صحیح است اگر

$$KB \vdash_i \alpha \Rightarrow KB \models \alpha$$

کامل بودن (completeness): رویه استنتاج i کامل است اگر

$$KB \models \alpha \Rightarrow KB \vdash_i \alpha$$

مثال: در منطق مرتبه اول (First Order Logic) یک رویه استنتاج کامل و صحیح وجود دارد.

منطق گزاره ای (Syntax)

- منطق گزاره ای ساده ترین نوع منطق است.
- سمبل های گزاره ای P1, P2 و ... هر کدام یک جمله می باشد.
- True و False ثابت های گزاره ای می باشند و هرکدام به تنهایی یک جمله می باشند.

اگر S جمله باشد، آنگاه $\neg S$ نیز یک جمله است (نقیض)

اگر S_1 و S_2 جمله باشند، $S_1 \wedge S_2$ نیز یک جمله است (ترکیب عطفی)

اگر S_1 و S_2 جمله باشند، $S_1 \vee S_2$ نیز یک جمله است (ترکیب فصلی)

اگر S_1 و S_2 جمله باشند، $S_1 \Rightarrow S_2$ نیز یک جمله است (ترکیب شرطی)

اگر S_1 و S_2 جمله باشند، $S_1 \Leftrightarrow S_2$ نیز یک جمله است (ترکیب دوشرطی)

جدول درستی پنج رابطه منطقی

P	Q	┐P	P ^ Q	P v Q	P=>Q	P⇔Q
F	F	T	F	F	T	T
F	T	T	F	T	T	F
T	F	F	F	T	F	F
T	T	F	T	T	T	T

هم ارزی منطقی

- دو جمله هم ارز منطقی هستند اگر و فقط اگر هر دو در یک مدل یکسان درست باشند.

$$\alpha \equiv \beta \quad \text{if and only if} \quad \alpha \models \beta \text{ and } \beta \models \alpha$$

$$\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{de Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}$$

منطق گزاره ای در دنیای Wumpus

اجازه دهید $P_{i,j}$ درست باشد، اگر و فقط اگر در خانه $[i, j]$ چاله باشد.
اجازه دهید $B_{i,j}$ درست باشد، اگر و فقط اگر در خانه $[i, j]$ نسیم باشد.

$\neg P_{1,1}$
$\neg B_{1,1}$
$B_{2,1}$

" چاله ها باعث وزش نسیم در خانه های مجاور می شوند".

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

" در یک خانه نسیم می وزد **اگر و فقط اگر** چاله ای مجاور آن باشد"

اعتبار و صدق پذیری

یک جمله معتبر (valid) است اگر در تمام مدل ها درست باشد

مثال: True، $A \vee \neg A$ ،$A \Rightarrow A$ ،$(A \wedge (A \Rightarrow B) \Rightarrow B)$

ارتباط معتبر بودن با استنتاج:

$KB \models \alpha$ iff $(KB \Rightarrow \alpha)$ is valid

یک جمله صدق پذیر (satisfiable) اگر در بعضی از مدل ها درست باشد

مثال: $A \vee B$

یک جمله صدق ناپذیر است اگر در هیچ مدلی درست نباشد

مثال: $A \wedge \neg A$

ارتباط صدق پذیری با استنتاج:

$KB \models \alpha$ iff $(KB \wedge \neg\alpha)$ is unsatisfiabe

زنجیر پیشرو و عقبگرد

- عبارات هورن: ترکیب فصلی لیترالهایی است که فقط یکی از آنها مثبت است
- مثال : $(\neg L_{1,1} \vee \neg Breeze \vee B_{1,1})$ یک عبارت هورن است

  $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1})$ یک عبارت هورن نیست
  - هر عبارت **هورن** را میتوان به صورت یک استلزام نوشت که مقدمه آن ترکیب عطفی لیترالهای مثبت و تالی آن یک لیترال مثبت است

    $(\neg L_{1,1} \vee \neg Breeze \vee B_{1,1})$ ➔ $(L_{1,1} \wedge Breeze) \Rightarrow B_{1,1}$
  - این نوع عبارات هورن که فقط یک لیترال مثبت دارند، **عبارات معین** نامیده میشوند
  - لیترال مثبت را **رأس** و لیترالهای منفی را **بدنه عبارت** گویند
  - عبارت معینی که فاقد لیترالهای منفی باشد، گزاره ای بنام **حقیقت** نام دارد
  - **عبارات معین** اساس **برنامه نویسی منطقی** را میسازد
- استنتاج با عبارات هورن، از طریق الگوریتم های **زنجیر پیشرو** و **زنجیر عقبگرد** انجام میگیرد

زنجیر پیشرو

الگوریتم زنجیر پیشرو تعیین میکند آیا نماد گزاره ای q(تقاضا)، توسط پایگاه دانش عبارات هورن ایجاب میشود یا خیر

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

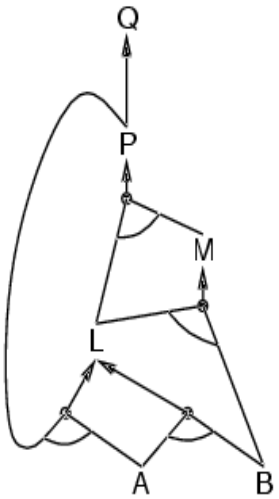
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

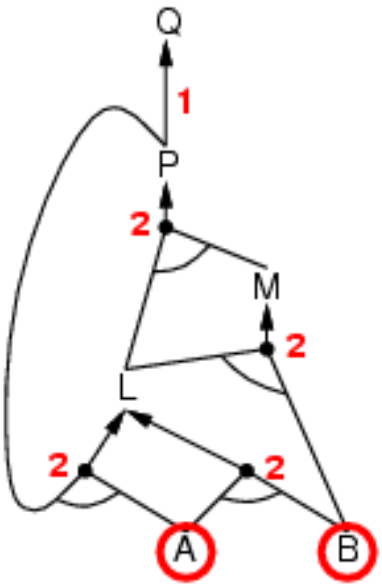
$A \wedge B \Rightarrow L$

A

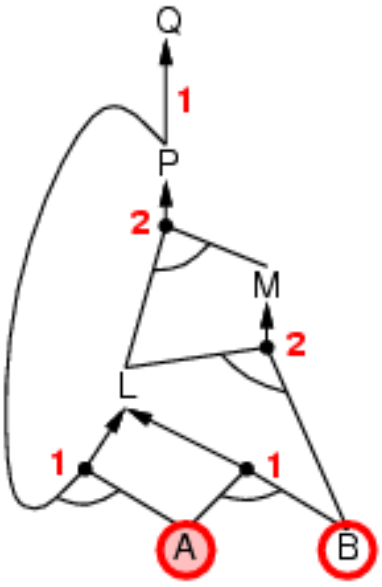
B



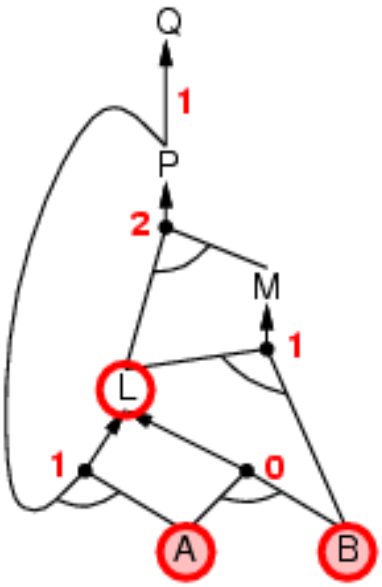
زنجیر پیشرو



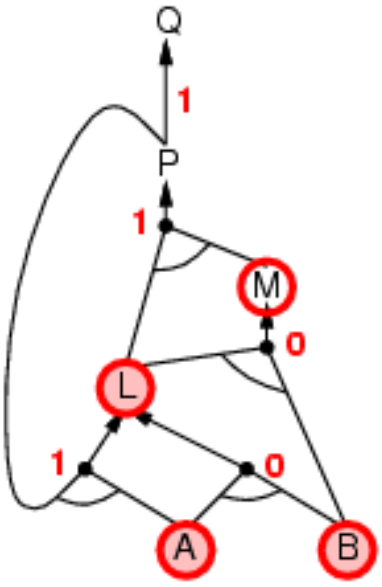
زنجیر پیشرو



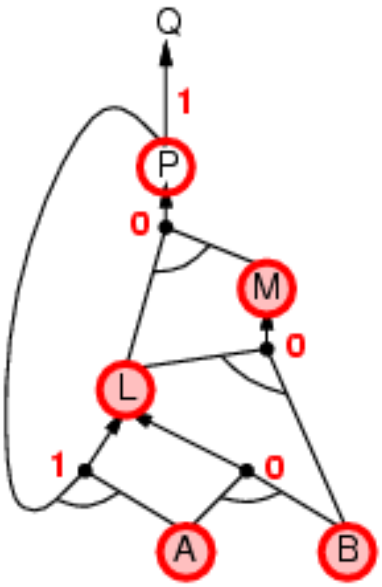
زنجیر پیشرو



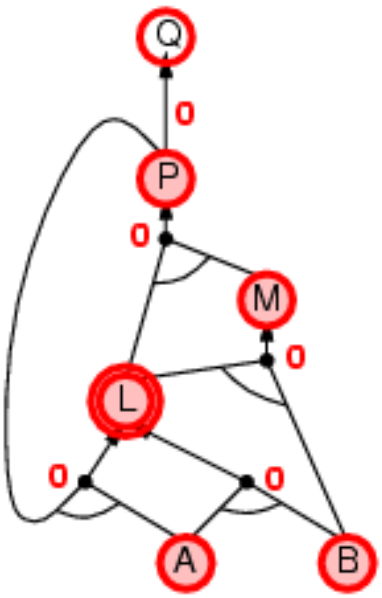
زنجیر پیشرو



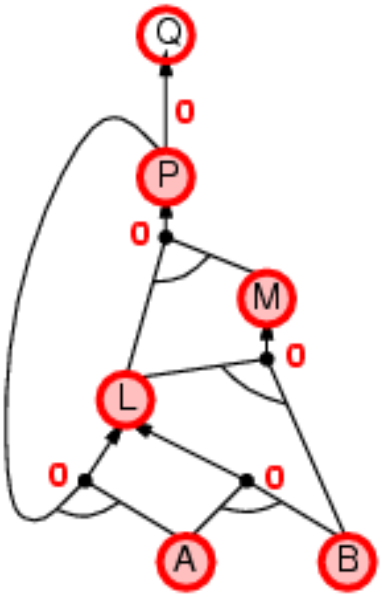
زنجیر پیشرو



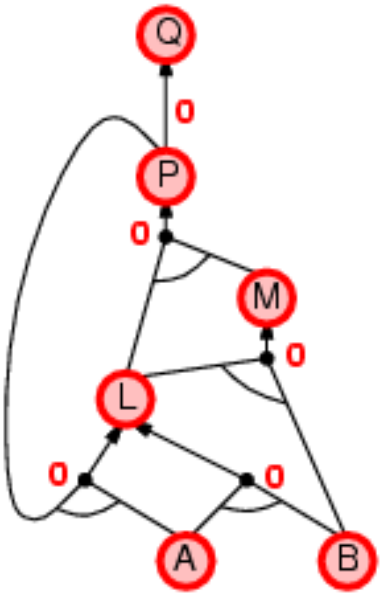
زنجیر پیشرو



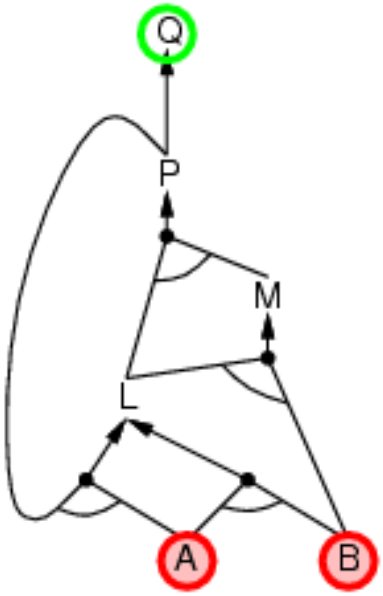
زنجیر پیشرو



زنجیر پیشرو



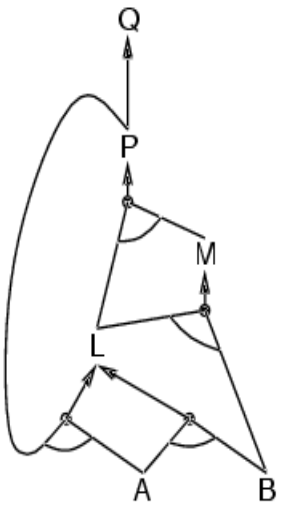
الگوریتم عقبگرد کامل



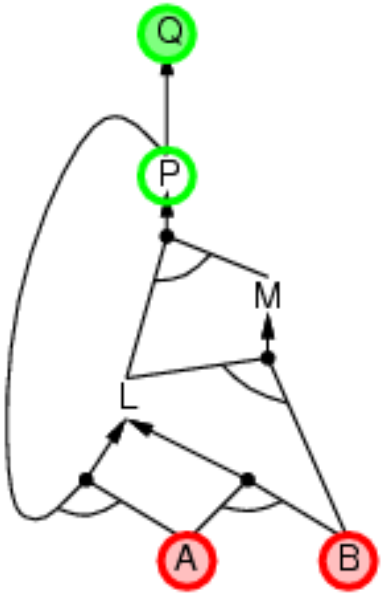
الگوریتم عقبگرد کامل

تغییرات عمده: خاتمه زودرس، اکتشاف نماد محض، اکتشاف عبارت واحد

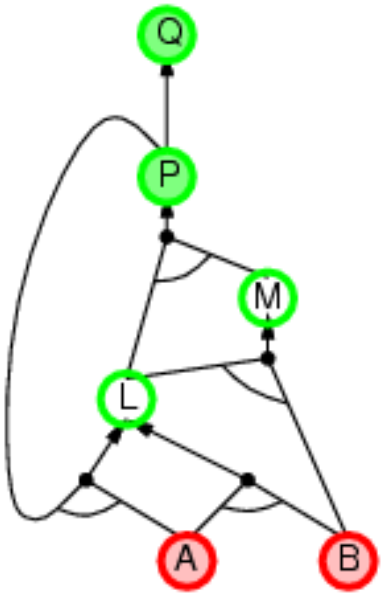
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



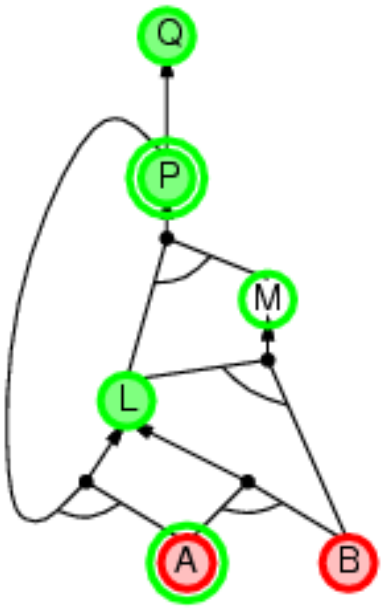
الگوریتم عقبگرد کامل



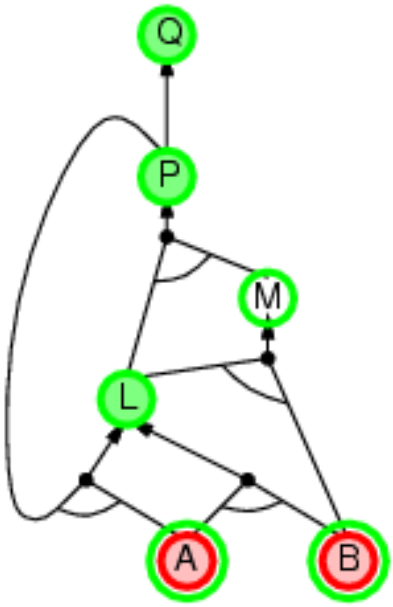
الگوریتم عقبگرد کامل



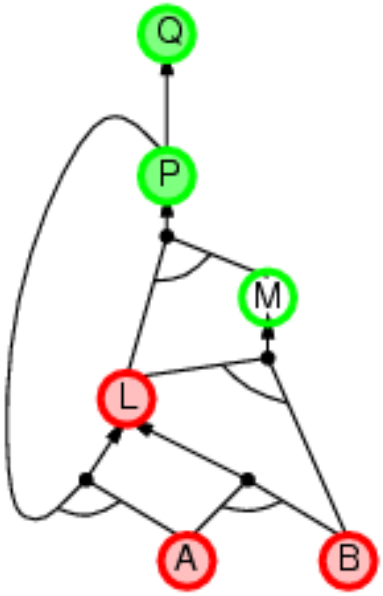
الگوریتم عقبگرد کامل



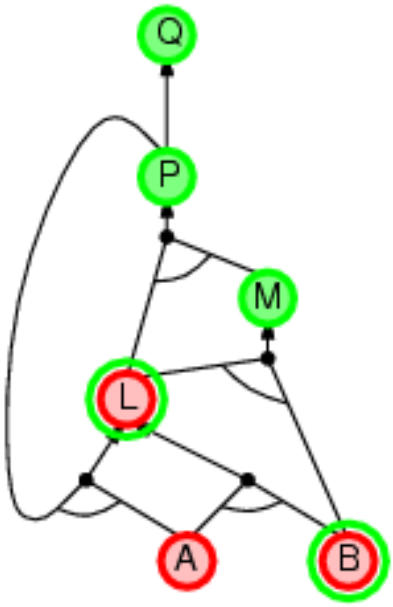
الگوریتم عقبگرد کامل



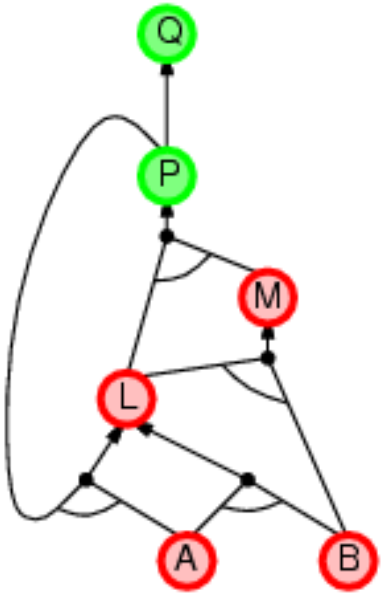
الگوریتم عقبگرد کامل



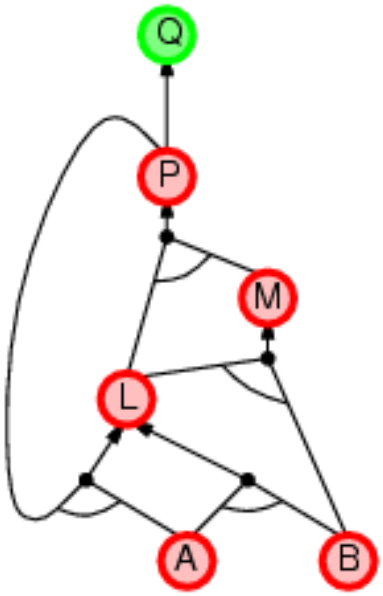
الگوریتم عقبگرد کامل



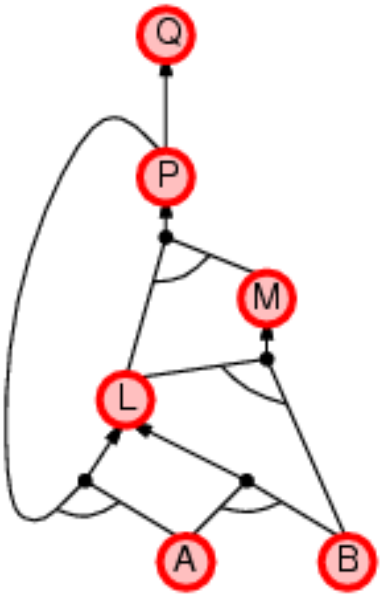
الگوریتم عقبگرد کامل



الگوریتم عقبگرد کامل



الگوریتم عقبگرد کامل



هوش مصنوعی

Artificial Intelligence

فصل هفتم و هشتم- منطق مرتبه اول و استنتاج آن

2

منطق مرتبه اول (منطق مسند)

First Order Logic (Predicate Logic)

فهرست

مروری بر منطق گزاره ای

- ویژگیها
  - ماهیت اعلانی(declarative) یا توصیفی
    - دانش و استنتاج متمایزند و استنتاج کاملاً مستقل از دامنه است
  - قدرت بیان کافی برای اداره کردن اطلاعات جزئی
    - با استفاده از ترکیب فصلی و نقیض
  - قابلیت ترکیب
    - معنای جمله، تابعی از معنای بخشهای آن
- معایب
  - قدرت بیان محدودی دارد
    - بر خلاف زبانهای طبیعی

منطق مرتبه اول

در حالیکه منطق گزاره ای فرض می کند دنیا از حقایق تشکیل شده است، منطق مرتبه اول (برخلاف زبان طبیعی) فرض می کند دنیا شامل موارد زیر است:

- اشیاء (Objects) : دنیا از اشیایی تشکیل شده که بواسطه خصوصیاتشان (Properties) از یکدیگر قابل تمایز می‌باشند
- روابط (Relations): در بین اشیاء روابط مختلفی می تواند وجود داشته باشد که بعضی از این روابط به صورت تابع (Function) می باشند.
- توابع (functions): پدرِ ، برادرِ، یکی بیشتر از و ...

حقایق به صورت ارجاع به اشیاء، خصوصیات و روابط بین اشیاء بیان می شوند.

مثال :

- Sum(1, 2, 3), Even(2), Odd(3), …
- Parent (Bob, Jim), Male(Bob), …
- Add(1, 2), LeftLegof(John), …

انواع منطق

زبان	هستی شناسی (آنچه در جهان هست)	حقیقت شناسی (اعتقادات عامل راجع به حقایق)
منطق گزاره ای	حقایق	درست /نادرست /نامشخص
منطق رتبه اول	حقایق ، اشیا ، رابطه ها	درست /نادرست /نامشخص
منطق زمانی (Temporal Logic)	حقایق ، اشیا ، رابطه ها ، زمان	درست /نادرست /نامشخص
نظریه احتمال	حقایق	درجه ای از اعتقاد متعلق به $[0,1]$

نحو و معنای منطق رتبه اول

- نمادهای ثابت ؛ اشیا را نشان میدهد. مثال: علی ، 2 ، رضا ، ...
- نمادهای مسند ؛ رابطه ها را نشان میدهد. مثال:برادر بودن ، بزرگتر بودن از
- نمادهای تابع ؛ توابع را نشان میدهند. مثال: تابع پای چپ(LeftLeg)
- متغیرها: x , y , a ,b
- روابط منطقی: $\neg, \Rightarrow, \wedge, \vee, \Leftrightarrow$
- تساوی: =
- سورها: $\forall, \exists$

ساختار جملات در منطق مرتبه اول

$$
\begin{aligned}
Sentence &\rightarrow AtomicSentence \\
&\mid (\ Sentence\ Connective\ Sentence\ ) \\
&\mid Quantifier\ Variable, \ldots\ Sentence \\
&\mid \neg\ Sentence \\
\\
AtomicSentence &\rightarrow Predicate(Term, \ldots) \mid Term = Term \\
\\
Term &\rightarrow Function(Term, \ldots) \\
&\mid Constant \\
&\mid Variable
\end{aligned}
$$

ساختار جملات در منطق مرتبه اول

$$
\begin{aligned}
Connective &\rightarrow \Rightarrow \mid \wedge \mid \vee \mid \Leftrightarrow \\
Quantifier &\rightarrow \forall \mid \exists \\
Constant &\rightarrow A \mid X_1 \mid John \mid \cdots \\
Variable &\rightarrow a \mid x \mid s \mid \cdots \\
Predicate &\rightarrow Before \mid HasColor \mid Raining \mid \cdots \\
Function &\rightarrow Mother \mid LeftLeg \mid \cdots
\end{aligned}
$$

Term

- ترم: یک عبارت منطقی که به یک شیء رجوع می کند.
  - ثابت ها (A, B, C, John, ...)
  - متغیرها(a, b, x, …)
  - توابع

MotherOf(Richard),
Length(LeftLegOf(John)),
Cos(30),…

جملات ساده (Atomic Sentence)

- یک جمله اتمی به صورت زیر می باشد:

Predicate($Term_1$,… , $Term_n$)

- Brother(KingJohn,RichardTheLionheart)
- >(Length(LeftLegOf(Richard)),Length(LeftLegOf(KingJohn)))
- Married(FatherOf(Richard), MotherOF(John))

- یک جمله ساده زمانی درست است که رابطه Predicate بین اشیایی که $Term_1$ و … و $Term_n$ به آنها اشاره می کنند برقرار باشد.

جملات پیچیده

با ترکیب جملات اتمیک و روابط منطقی میتوان جملات پیچیده تری ساخت

$\neg S,\ S1 \wedge S2,\ S1 \vee S2,\ S1 \Rightarrow S2,\ S1 \Leftrightarrow S2$

مثال:

$\neg$Brother(LeftLeg(Richard),John)

Brother(Richard,John) $\wedge$ Brother(John,Richard)

King(Richard) $\vee$ King(John)

$\neg$King(Richard) $\Rightarrow$ King(John)

سورها

- کمک میکنند تا به جای شمارش اشیا از طریق نام آنها ، خواص کلکسیون اشیا را بیان کرد
- **سور عمومی ؛** $\forall$ "برای همه"
- **سور وجودی ؛** $\exists$ " وجود دارد حداقل..."

سور عمومی

$\forall$<متغیرها> <جمله>

- $\forall x\ P$ که در آن P یک عبارت منطقي است ، بیان میکند که P برای هر شیء x درست است
- مثال: $\forall x\ King(x) \Rightarrow Person(x)$

سور وجودی

∃ <متغیرها> <جمله>

∃ x P که در آن P یک عبارت منطقي است ، بیان میکند که P حداقل برای یک شیء x درست است

مثال:

Someone at class is smart

$$\exists x\ at(x, class) \wedge Smart(x)$$

خصوصیات سورها

- سور عمومی اغلب با ترکیب شرطی و سور وجودی اغلب با ترکیب عطفی بکار می رود

$$\forall x\ Human(x) \Rightarrow Mortal(x)$$

$$\exists x\ at(x, class) \wedge Smart(x)$$

- مثال : جمله زیر به معنای " هر کسی در کلاس است و هر کسی باهوش است " می باشد

ویا جمله زیر:

$$\forall x\ at(x, class) \wedge Smart(x)$$

$$\exists x\ at(x, class) \Rightarrow Smart(x)$$

خصوصیات سورها

- رابط طبیعي برای کار با $\forall$ و $\wedge$ رابط طبیعي برای کار با $\exists$ میباشد $\Rightarrow$
- $\forall x\ \forall y$برابر است با$\forall y\ \forall x$ و $\exists x\ \exists y$برابر است با $\exists y\ \exists x$
- $\forall y\ \exists x$برابر نیست با$\exists x\ \forall y$

$\exists x\ \forall y\ Loves(x,y)$

- حداقل یک نفر وجود دارد که همه چیز در جهان را دوست دارد

$\forall y\ \exists x\ Loves(x,y)$

- همه در دنیا حداقل یک نفر را دوست دارند

$\forall x \exists y\ mother(y,x)$ $\exists y \forall x\ mother(y,x)$

خصوصیات سورها

- "هر کسی بستنی را دوست دارد" به معنای این است که "هیچ کس وجود ندارد که بستنی را دوست نداشته باشد"

$\forall x\ Likes(x\ ,\ IceCream)$ هم ارز $\neg\exists x\ \neg Likes(x\ ,\ IceCream)$

- $\forall x\ \neg P$ هم ارز $\neg\exists x\ P$
- $\neg\forall x\ P$ هم ارز $\exists x\ \neg P$
- $\forall x\ P$ هم ارز $\neg\exists x\ \neg P$
- $\exists x\ P$ هم ارز $\neg\forall x\ \neg P$

تساوی

- با استفاده از = دو ترم به یک شیء اشاره میکنند
- برای تعیین درستی جمله تساوی باید دید که آیا ارجاع ها به دو ترم ، اشیای یکسانی اند یا خیر
- مثال: ریچارد حداقل دو برادر دارد

$\exists x,y\ Brother(x,Richard) \wedge Brother(y,Richard) \wedge \neg(x=y)$

ادعاها و تقاضاها

- جملات از طریق TELL به پایگاه دانش اضافه میشوند
  - این جملات را **ادعا** گویند
    - TELL (*KB* , King(John))
    - TELL (*KB* , ∀x King(x) => Person(x))
- با استفاده از ASK تقاضاهایی را از پایگاه دانش انجام میدهیم
  - این پرسشها ، **تقاضا** یا **هدف** نام دارد
    - ASK (*KB* , Person(John))
    - ASK(*KB* , ∃x Person(x))
- لیست جانشینی یا انقیاد
  - لیستی از جانشینیها در صورت وجود بیش از یک پاسخ

روابط خانوادگی در منطق مرتبه اول

- Male(John)
- Female(Judy)
- Parent(Jim, John)
- Married(Jim, Julia)
- …

روابط خانوادگی در منطق مرتبه اول

$\forall x, y\ \text{Child}(x, y) \Leftrightarrow \text{Parent}(x, y)$

$\forall x, y\ \text{Sibling}(x, y) \Leftrightarrow x \neq y \wedge \exists z\ \text{Parent}(z, x) \wedge \text{Parent}(z, y)$

$\forall x, y\ \text{Sibling}(x, y) \Leftrightarrow \text{Sibling}(y, x)$

$\forall x, y\ \text{Brother}(x, y) \Leftrightarrow \text{Male}(x) \wedge \text{Sibling}(x, y)$

$\forall x, y\ \text{Sister}(x, y) \Leftrightarrow \text{Female}(x) \wedge \text{Sibling}(x, y)$

$\forall x, y\ \text{Uncle}(x, y) \Leftrightarrow \exists z \text{Parent}(z, x) \wedge \text{Brother}(x, z)$

مثالهایی از دنیای Wumpus

- خصوصیات محل ها:

$$\forall l, t\ At(Agent, l, t) \wedge Stench(l) \Rightarrow Smelly(l)$$

$$\forall l, t\ At(Agent, l, t) \wedge Breeze(l) \Rightarrow Breezy(l)$$

- خانه های مجاور چاله ها دارای نسیم هستند:
  - قوانین تشخیصی (diagnostic) : علت را از روی اثر آن نتیجه می گیرد

$$\forall y\ Breezy(y) \Rightarrow \exists x Pit(x), Adjacent(x, y)$$

  - قوانین سببی (causal): اثر را از روی علت نتیجه می گیرد:

$$\forall x, y\ Pit(x), Adjacent(x, y) \Rightarrow Breezy(y)$$

- تعریف Breezy :

$$\forall y\ Breezy(y) \Leftrightarrow [\exists x Pit(x), Adjacent(x, y)]$$

استنتاج در منطق مرتبه اول

تبدیل یک داستان تاریخی به صورت منطق مسند

- 1.Marcus was a man.
- 1.Man(Marcus)
- 2.Marcus was a Pompeian.
- 2.pompeian (Marcus)
- 3.All Pompeian were Romans
- 3. ∀ x ;Pompeian (x) ⇒ Romans(n)
- 4.caesar was a ruler.
- 4.ruler (Caesar)

تبدیل یک داستان تاریخی به صورت منطق مسند

- 5.All Romans were either loyal to Caesar or hated him
- 5. ∀ x; Roman (x) ⇒ loyal to (x ,Caesar) ∨ hate (x , Caesar)
- 6.Every one is loyal to someone.
- 6. ∀x∃y loyal to (x,y)
- 7.People only try to assassinate ruler's they are not loyal to.
- 7. ∀x∀ y person(x) ∧ ruler(y) ∧ trytoassassinate (x, y)⇒
- ~loyal to (x,y)
- 8.Marcus tried to assassinate Caesar
- 8.trytoassassinate (Marcus , Caesar)

نحوه استنتاج

- سوال زیر با توجه به دانش قبلی پرسیده می شود.

Was Marcus loyal to Cesar ?

Loyal to (Marcus , Cesar) ?

~ loyal to (Marcus , Cesar)

↓ (7 , x | Marcus , y |Caesar)

Person (Marcus) ∧ Ruler (Caesar) ∧ try assassinate (Marcus,Caesar)

↓ (4)

Person (Marcus) ∧ try assassinate (Marcus , Caesar)

↓ (8)

Person (Marcus)

- چون چنین جمله ای در پایگاه دانش وجود ندارد درنتیجه :

T ≡ Loyal to (Marcus ,Cesar) یـا F ≡~ loyal to (Marcus , Cesar)

نحوه استنتاج (ادامه)

- مثال قبل به تناقض رسید

چرا که پایگاه دانش می گوید،

مارکوس به سزار وفادار نبوده است.

- عبارت روبرو را اضافه می کینم.

9.All man Are Person.

$\forall x; man(x) \Rightarrow person(x)$

Person (Marcus)

↓ (9, X | M)

Man(Marcus)

↓

□ تهی

به تهی رسیدن یعنی فرض ما درست بوده است .

F ≡ Loyal to (Marcus ,Cesar) یـا T ≡~ loyal to (Marcus , Cesar)