هوش مصنوعی

فصل چهارم

جستجوی آگاهانه

دانشگاه پیام نور شهرکرد

ایمان مختاری

جستجوی آگاهانه

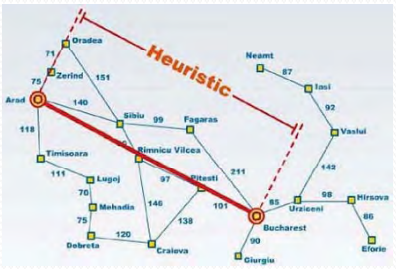
دانش ما از مساله فراتر از تعریف مساله است و توسط اطلاعاتی ما را به هدف نزدیکتر می کنند

تابع هیورستیک یا اکتشاف (Heuristic)

به صورت h(n) نمایش داده می شود و تخمینی از هزینه گره n تا گره ،هدف طبق ارزانترین راه است

اگر n گره هدف باشد در آنصورت h(n) برابر با ۰ است

هرچه هیورستیک به هزینه واقعی نزدیکتر باشد سریعتر به هدف می رسیم



متدهای جستجوی آگاهانه

- جستجو اول بهترین
  - حریصانه
  - A*
  - IDA*
  - RBFS
  - MA* و SMA*
- جستجوی محلی و بهینه سازی
  - تپه نوردی
  - شبیه سازی حرارت
  - پرتو محلی
  - الگوریتمهای ژنتیک

تعاریف

- **تابع هزینه مسیر**، g(n) : هزینه مسیر از گره اولیه تا گره n
- **تابع اکتشافی**، h(n) : هزینه تخمینی ارزان ترین مسیر از گره n به گره هدف
- **تابع ارزیابی**، f(n) : هزینه تخمینی ارزان ترین مسیر از طریق n

روشهای اول بهترین

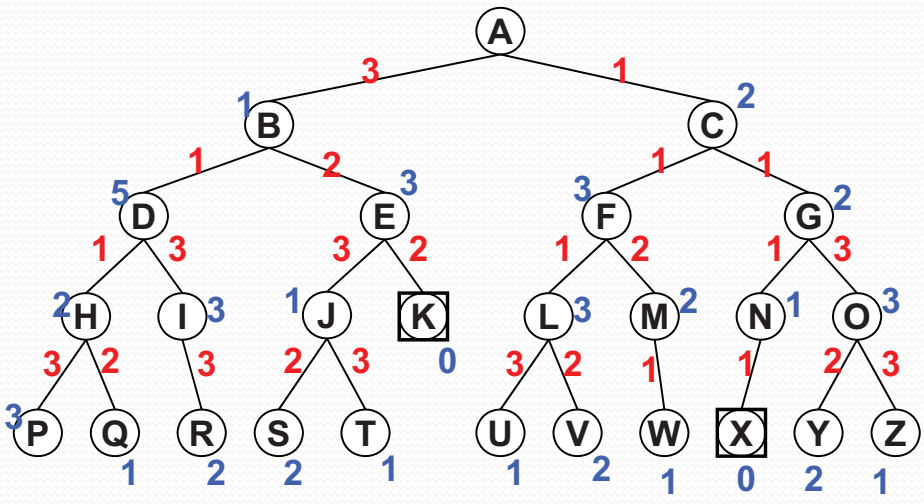
این استراتژی به این صورت بیان می‌شود که در یک درخت، زمانی که گره‌ها مرتب می‌شوند، گره‌ای که بهترین ارزیابی را داشته باشد، قبل از دیگر گره‌ها بسط داده می‌شود.

هدف: یافتن راه‌حل‌های کم‌هزینه است

جستجوی حریصانه:

- گره ای که به نظر به هدف نزدیکتر است ابتدا بسط داده می شود
- اگر در حین بسط حتی زمانیکه به هدف برسیم متوجه شدیم گره ای ادعا دارد با هزینه کمتری ما را به هدف می رساند یا به بن بست رسیدیم بر می گردیم
- چرا خسیسه!!! چون زود میخواد به هدف برسه.

مثال ۱

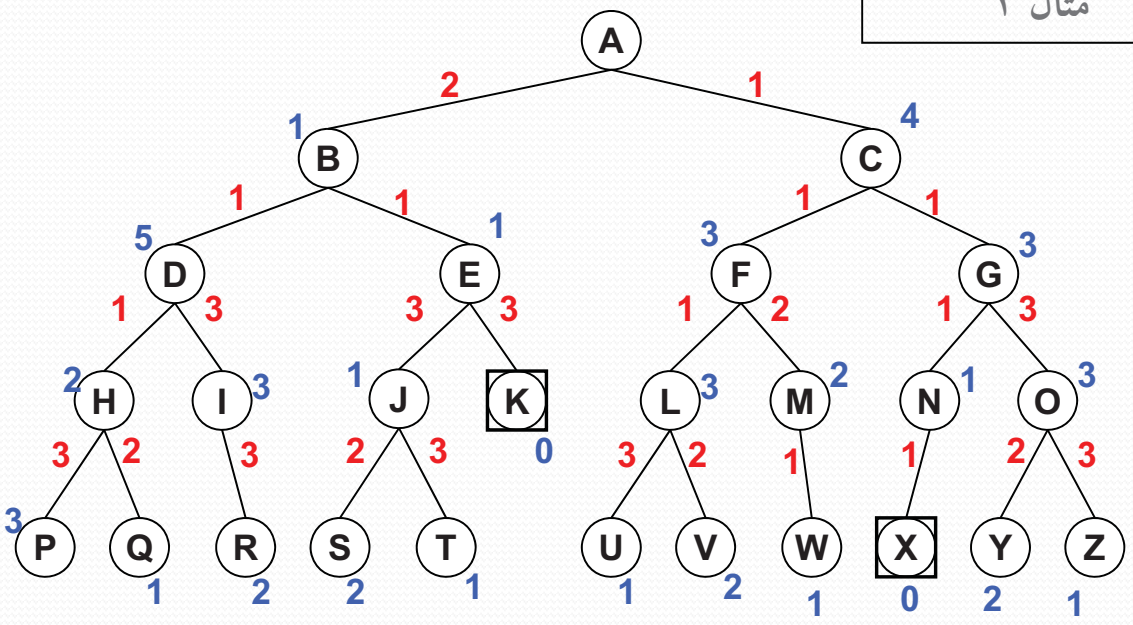


آبي ها :هزينه تخميني تا هدف
قرمزها:هزينه واقعي بين دو گره

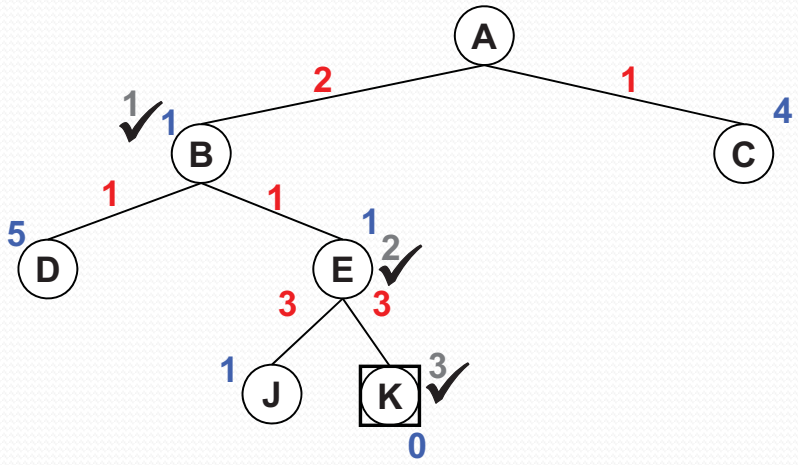
جستجوی حریصانه

جستجوی حریصانه

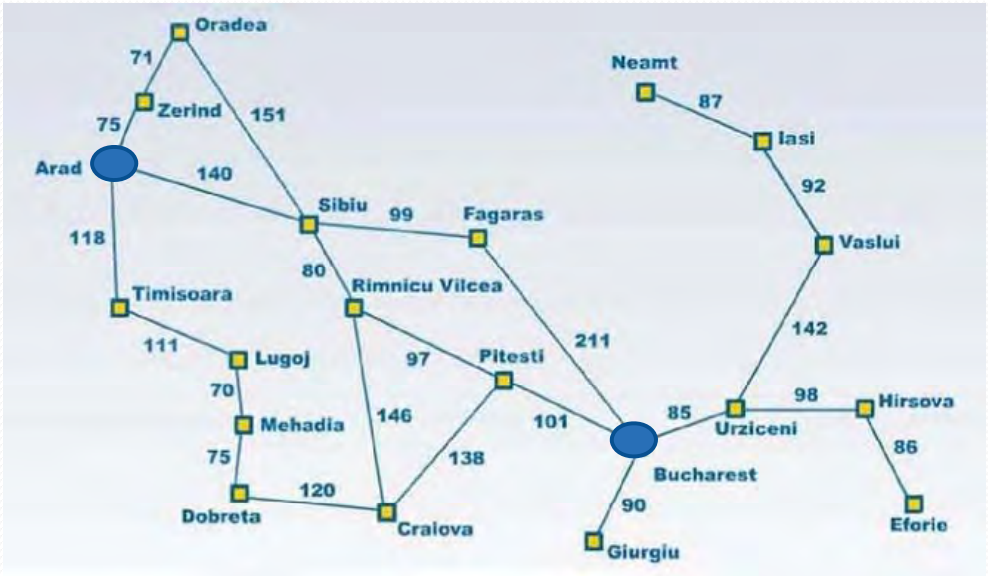
مثال ۲



جستجوی حریصانه



نقشه رومانی و فاصله شهرها تا بخارست



Straight-Line distance To Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

مثال : حرکت از شهر Arad به شهر Bucharest

فاصله مستقیم از شهر n تا Bucharest = $h(n)$

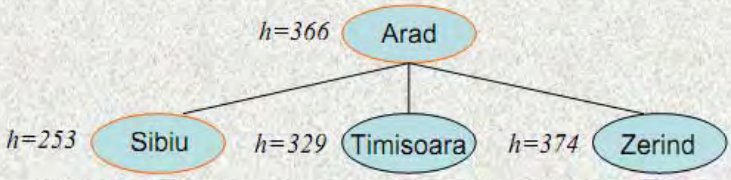
جستجوی حریصانه گرهی را دنبال می کند که به نظر نزدیکترین شهر به گره هدف یعنی شهر Bucharest است.

جستجوی حریصانه

- مثال : حرکت از شهر Arad به شهر Bucharest

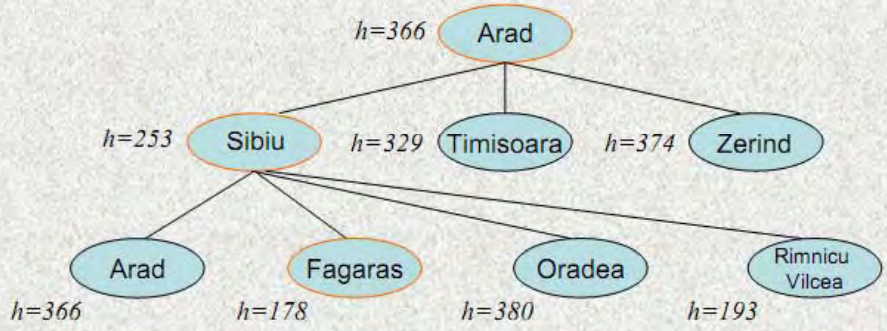
جستجوی حریصانه

- مثال : حرکت از شهر Arad به شهر Bucharest



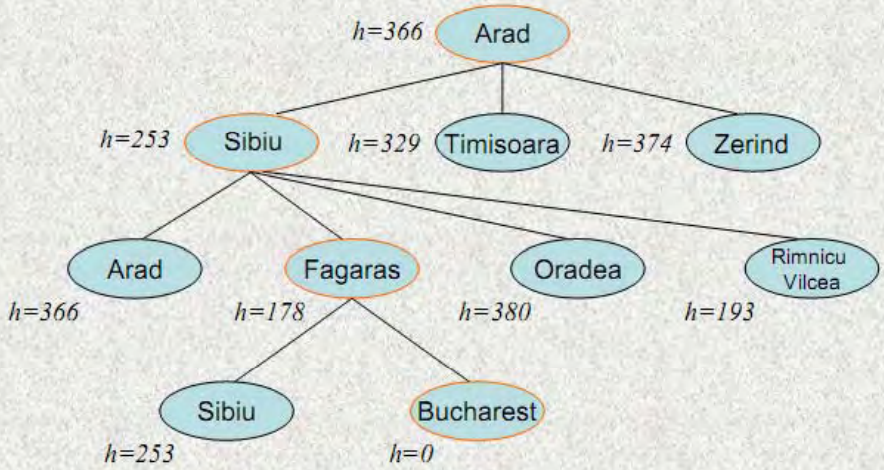
جستجوی حریصانه

- مثال : حرکت از شهر Arad به شهر Bucharest

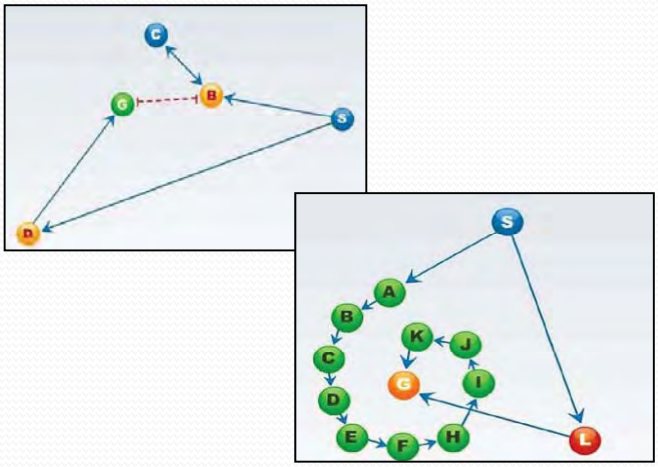


جستجوی حریصانه

- مثال : حرکت از شهر Arad به شهر Bucharest



ارزیابی جستجوی حریصانه



کامل بودن: خیر

چون ممکن است در حلقه بیفتد

بهینگی: خیر

هرچند که در برخی هیورستیکها بهینه است

پیچیدگی زمانی:

در بدترین حالت $O(b^m)$

والی اگر h برابر با هزینه واقعی تا هدف باشد $O(bd)$

پیچیدگی فضا:

در بدترین حالت $O(b^m)$

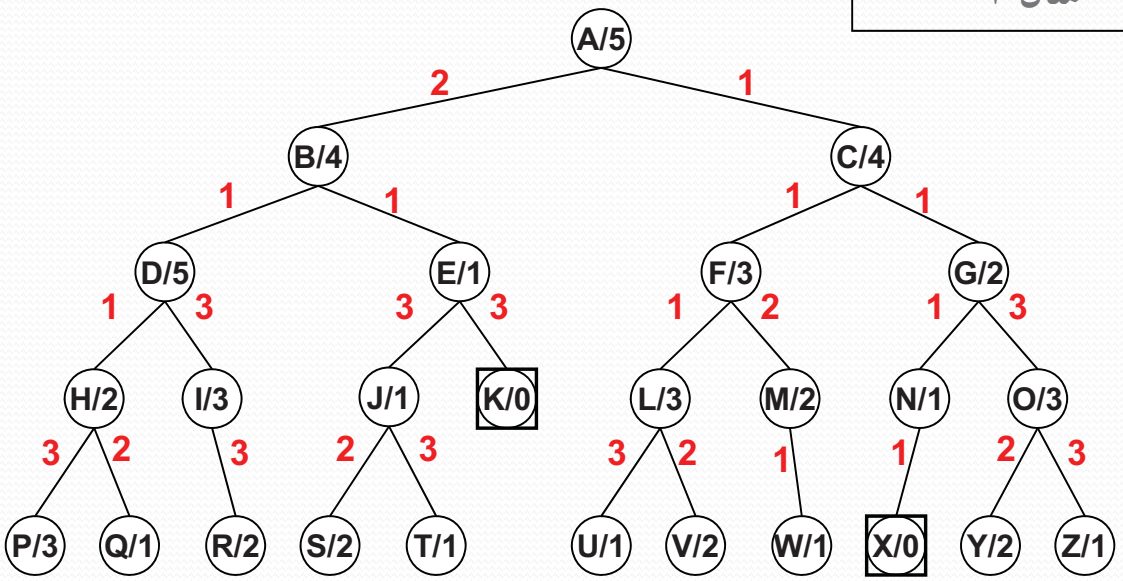
والی اگر h برابر با هزینه واقعی تا هدف باشد $O(bd)$

جستجوی A*

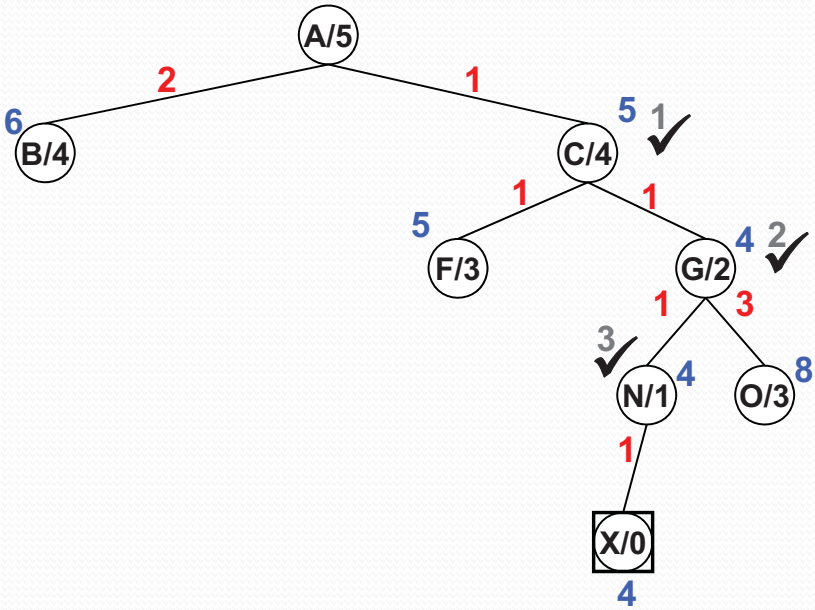
این روش جستجو ، روش کمترین هزینه (ناآگاهانه) و روش حریصانه را ترکیب می کند و میزان حداقل گره را بسط می دهد . به این شکل تابع ارزیابی عبارت است از:

$f(n) = g(n)^* + h(n)$

مثال ۱

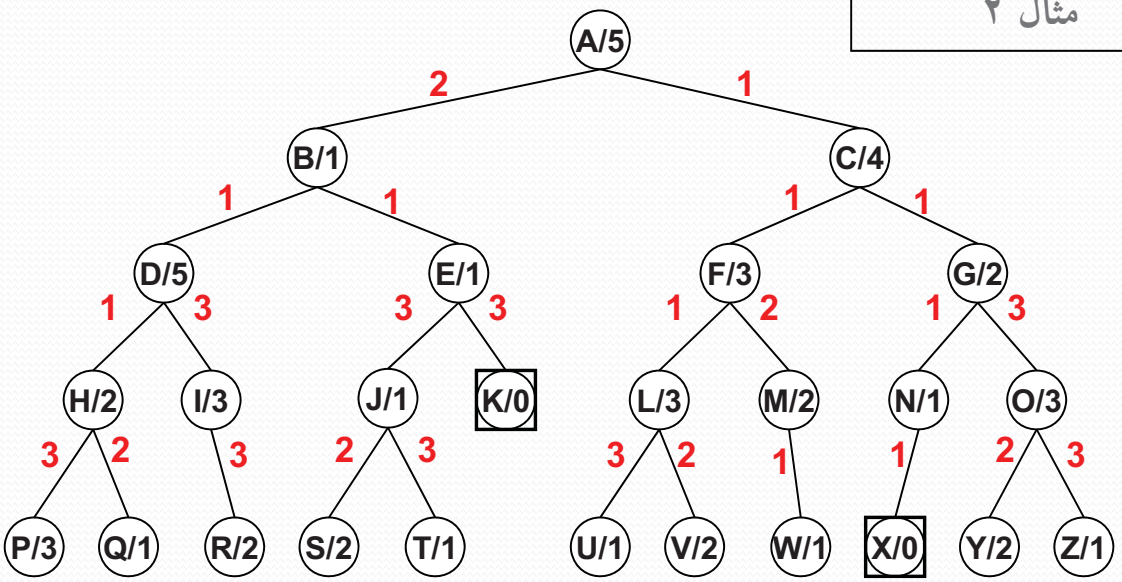


جستجوی A*

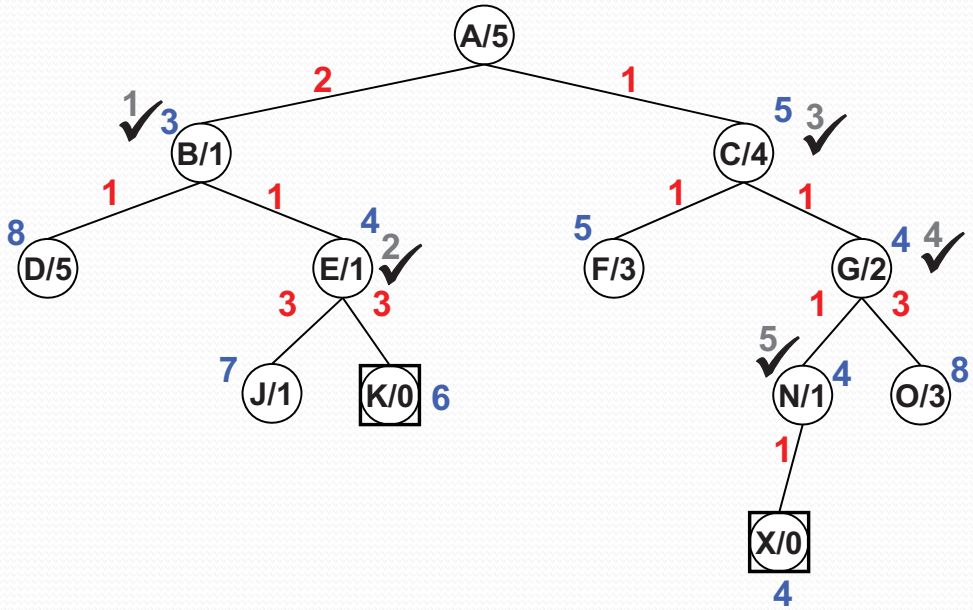


جستجوی A*

مثال ۲

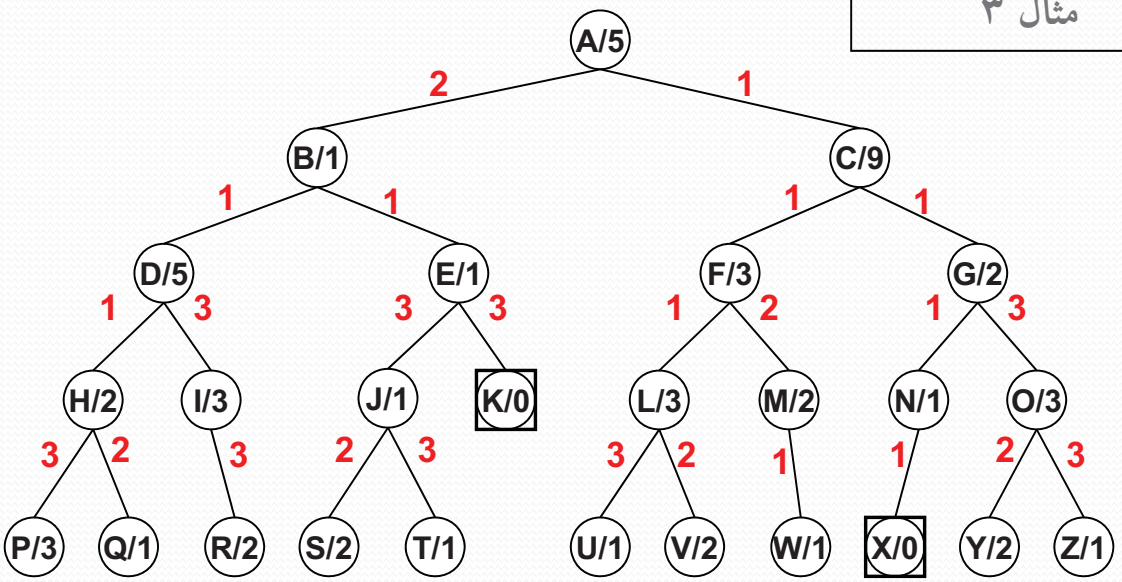


جستجوی A*

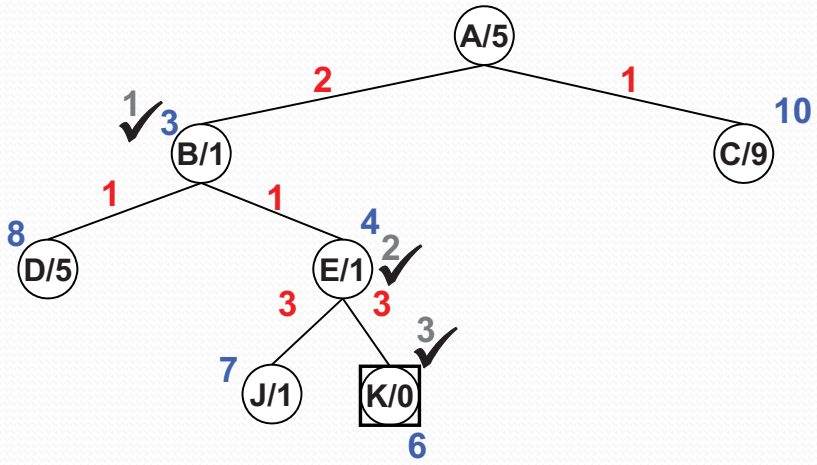


جستجوی A*

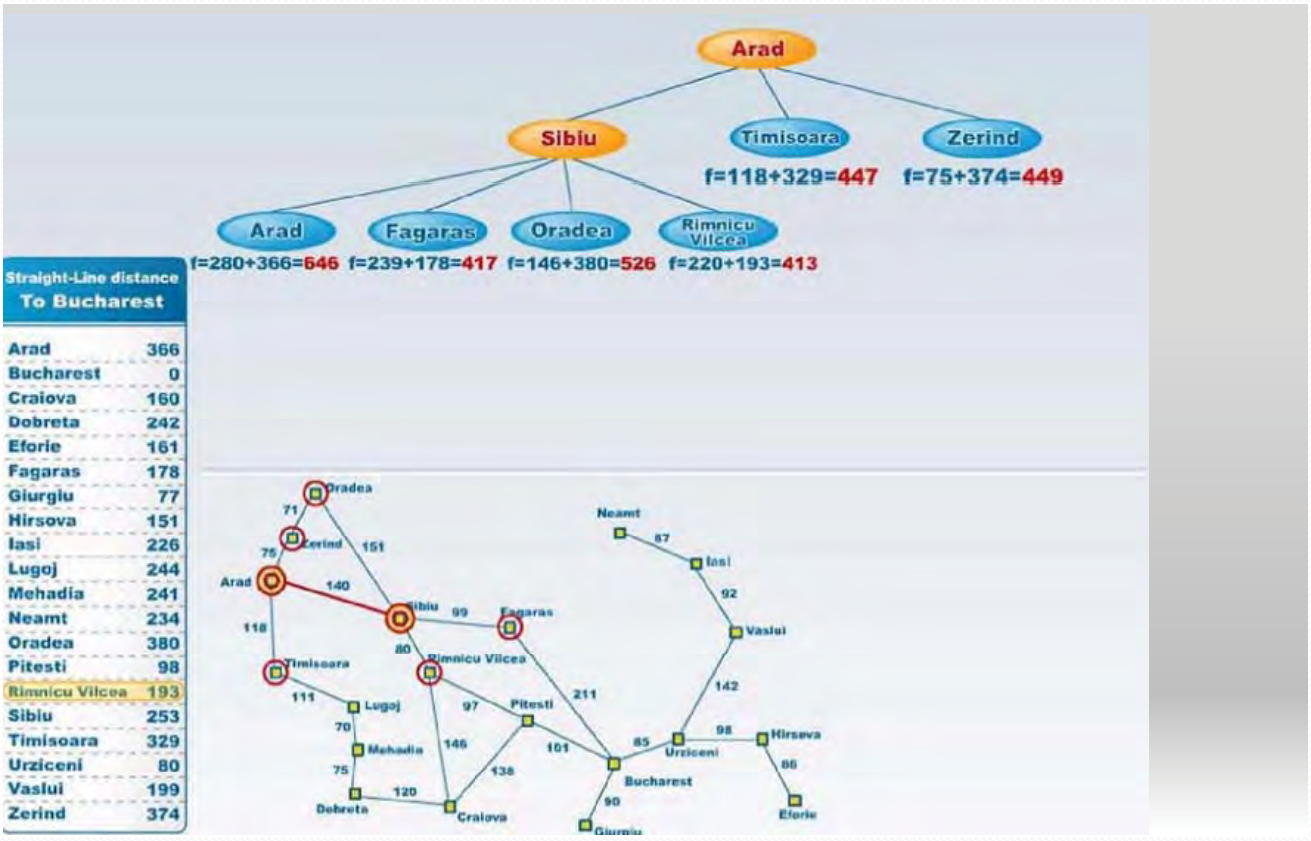
مثال ۳

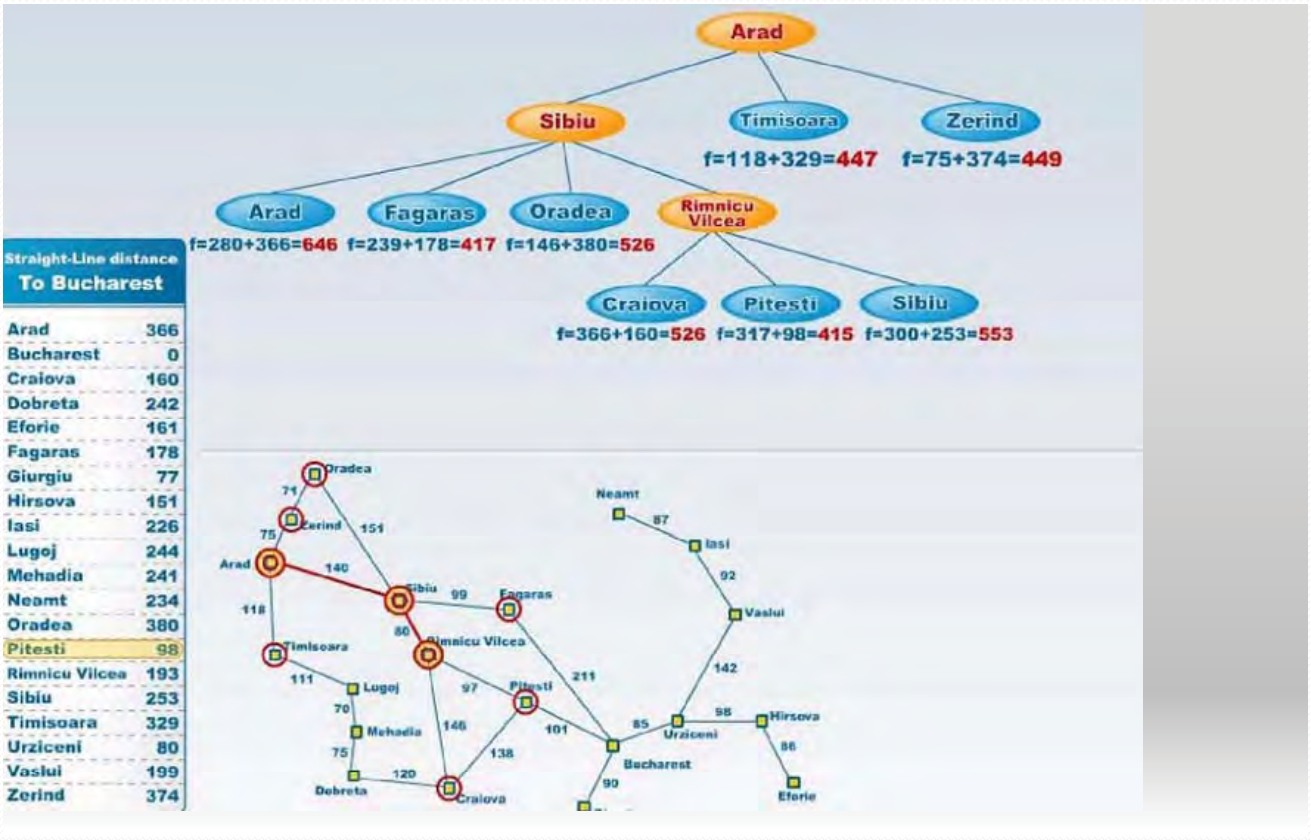


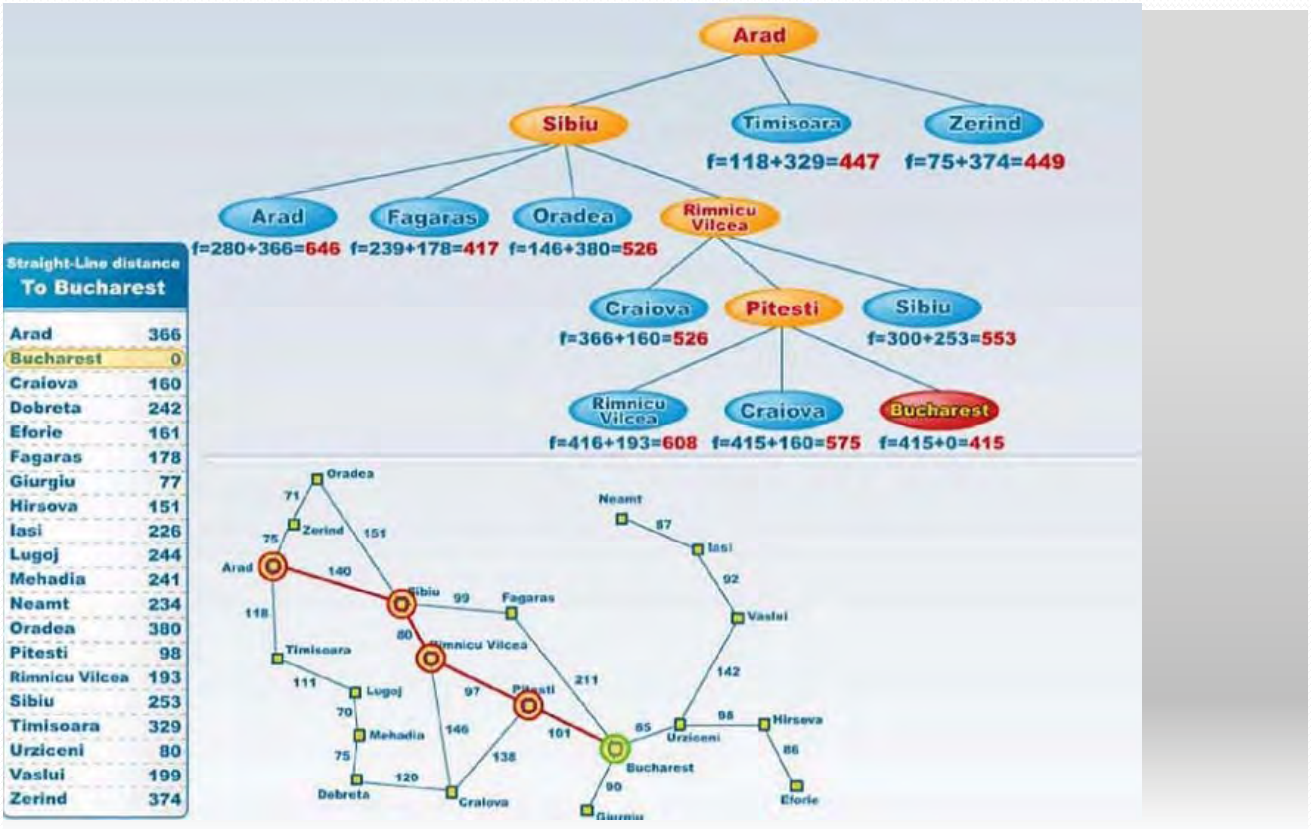
جستجوی A*



Straight-Line distance To Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374


Arad
Sibiu
Timisoara
Zerind
f=118+329=447
f=75+374=449
Arad
Fagaras
Oradea
Rimnicu Vilcea
f=280+366=646
f=239+178=417
f=146+380=526
f=220+193=413
Straight-Line distance
To Bucharest
Arad 366
Bucharest 0
Craiova 160
Dobreta 242
Eforie 161
Fagaras 178
Giurgiu 77
Hirsova 151
Iasi 226
Lugoj 244
Mehadia 241
Neamt 234
Oradea 380
Pitesti 98
Rimnicu Vilcea 193
Sibiu 253
Timisoara 329
Urziceni 80
Vaslui 199
Zerind 374
Oradea
Neamt
Zerind
Iasi
Arad
Sibiu
Fagaras
Vaslui
Rimnicu Vilcea
Timisoara
Lugoj
Pitesti
Hirsova
Mehadia
Urziceni
Bucharest
Dobreta
Craiova
Eforie
Giurgiu


Arad
Sibiu
Timisoara
Zerind
f=118+329=447
f=75+374=449
Arad
Fagaras
Oradea
Rimnicu Vilcea
f=280+366=646
f=239+178=417
f=146+380=526
Craiova
Pitesti
Sibiu
f=366+160=526
f=317+98=415
f=300+253=553
Straight-Line distance
To Bucharest
Arad 366
Bucharest 0
Craiova 160
Dobreta 242
Eforie 161
Fagaras 178
Giurgiu 77
Hirsova 151
Iasi 226
Lugoj 244
Mehadia 241
Neamt 234
Oradea 380
Pitesti 98
Rimnicu Vilcea 193
Sibiu 253
Timisoara 329
Urziceni 80
Vaslui 199
Zerind 374
Oradea
Neamt
Zerind
Iasi
Arad
Sibiu
Fagaras
Vaslui
Rimnicu Vilcea
Timisoara
Lugoj
Pitesti
Hirsova
Mehadia
Urziceni
Bucharest
Dobreta
Craiova
Eforie



کشف‌کنندگی (هیورستیک)قابل قبول:

تابع هیورستیکی را که هزینه‌ای تخمینی اش بیشتر از اندازه واقعی نباشد، یک کشف‌کنندگی قابل قبول (admissible heuristic) گویند.(مثل خط مستقیم در نقشه رومانی)

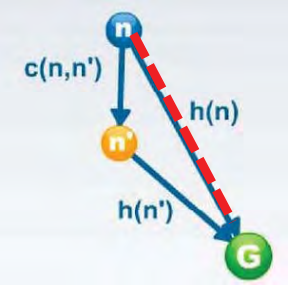
$h^*(n) <= h(n)$

هزینه واقعی

جستجوی درختی Tree-Search

به هر گره رسیدیم بدون توجه به اینکه هدف است شرط بررسی شود و ممکن است با رسیدن به هدف، جستجو همچنان ادامه پیدا کند

❖برای هیورستیک قابل قبول، A* با جستجوی درختی(اگر هدف شرط $h \leq h^*$ را نداشت مجددا جستجو انجام می شود) کامل و بهینه است



کشف‌کنندگی سازگار(شرط یکنوایی)

تابع هیورستیکی را که با نزدیک شدن به هدف ، تخمین دقیقتر می شود
مثلا اگر برای رسیدن به هدف از گره n به گره n' می رویم و c(n,n') هزینه مسیر از n به n' باشد آنگاه باید

h(n)<=h(n')+c(n,n')

تابع هیورستیکی سازگار مسلما ، قابل قبول هم هست

A heuristic is consistent if

$$h(n) \le c(n,a,n') + h(n')$$

If h is consistent, we have

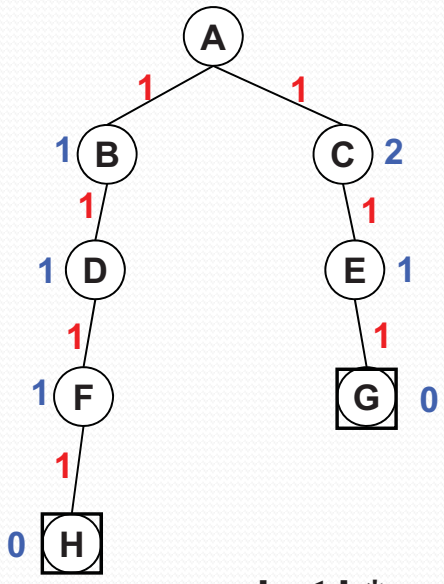
$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\ge g(n) + h(n) \\ &\ge f(n) \end{aligned}$$

جستجوی گرافی graph-Search

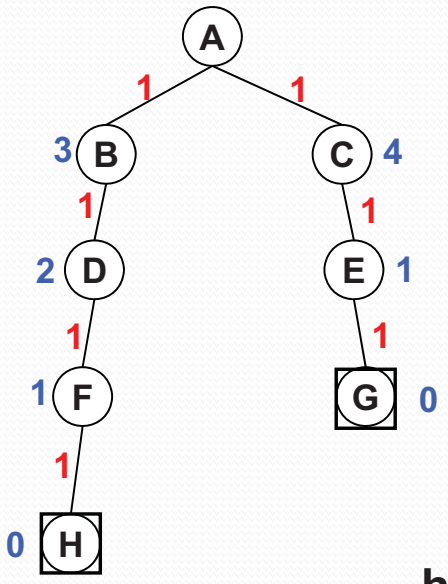
اگر اولین گره هدف پیدا شد دیگر شرط بررسی نمی شود

❖برای شرط هیورستیک سازگار ، A* با جستجوی گرافی هم کامل و بهینه است

جستجوی A* و TreeSerach

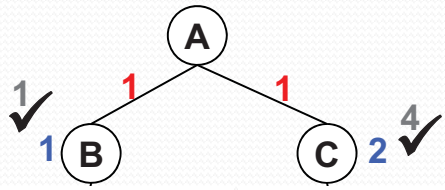


h ≤ h*

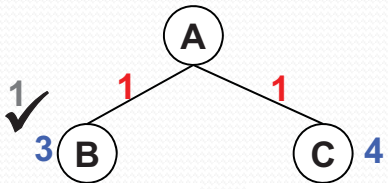


h ≰ h*

جستجوی A* و TreeSerach



$h \leq h^*$



$h \not\leq h^*$

مثال دیگر از جستجوی A*

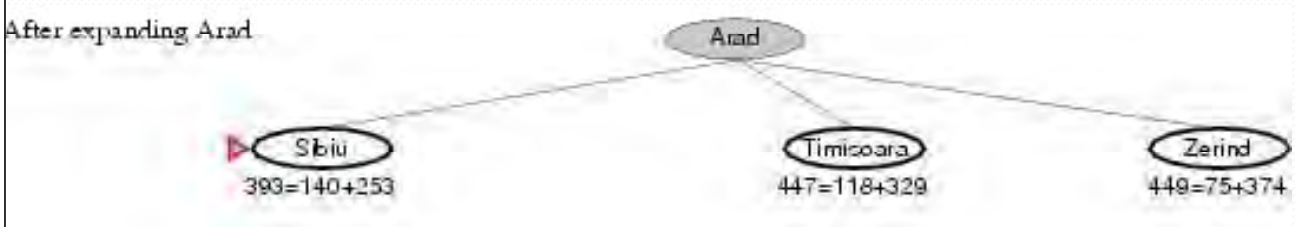
$f(n)=g(n) + h(n)$

جستجوی A* در نقشه رومانی

جستجوی Bucharest با شروع از Arad

f(Arad) = g(Arad)+h(Arad)=0+366=366

جستجوی A* در نقشه رومانی



Arad را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

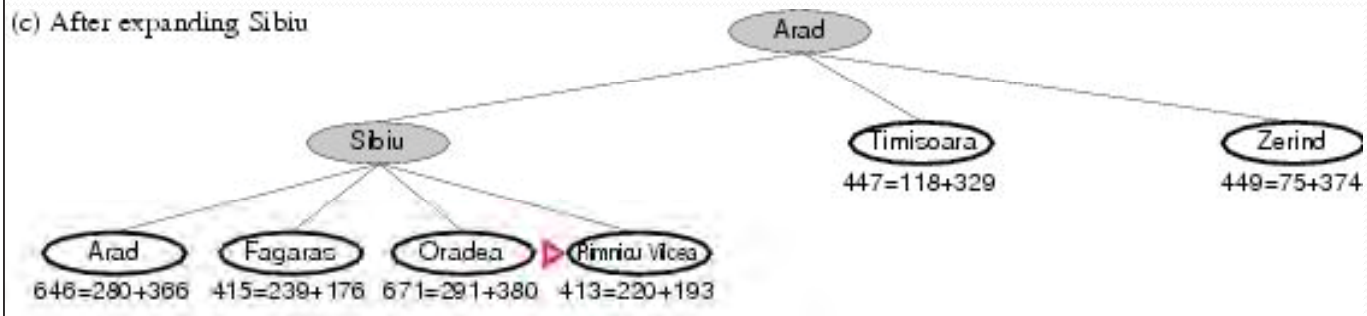
f(Sibiu)=c(Arad,Sibiu)+h(Sibiu)=140+253=393

f(Timisoara)=c(Arad,Timisoara)+h(Timisoara)=118+329=447

f(Zerind)=c(Arad,Zerind)+h(Zerind)=75+374=449

بهترین انتخاب شهر Sibiu است

جستجوی A* در نقشه رومانی

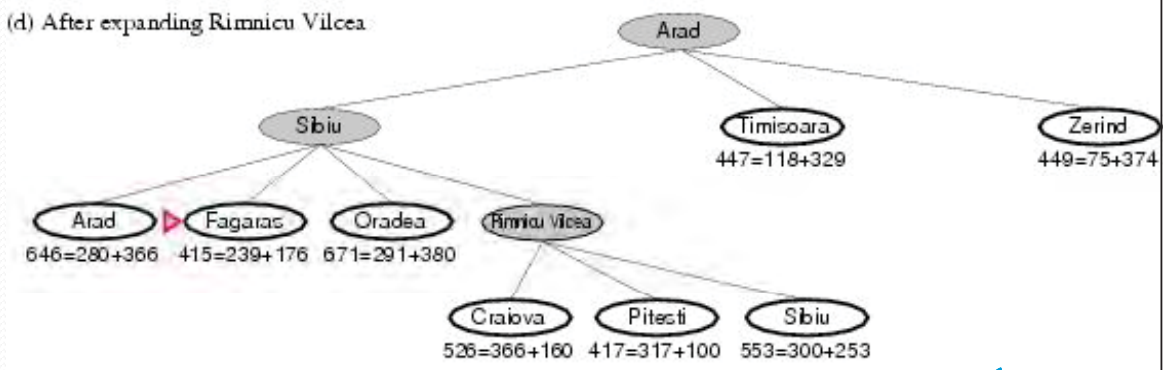


Sibiu را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

f(Arad)=c(Sibiu,Arad)+h(Arad)=280+366=646
f(Fagaras)=c(Sibiu,Fagaras)+h(Fagaras)=239+179=415
f(Oradea)=c(Sibiu,Oradea)+h(Oradea)=291+380=671
f(Rimnicu Vilcea)=c(Sibiu,Rimnicu Vilcea)+ h(Rimnicu Vilcea)=220+192=413

بهترین انتخاب شهر Rimnicu Vilcea است

جستجوی A* در نقشه رومانی



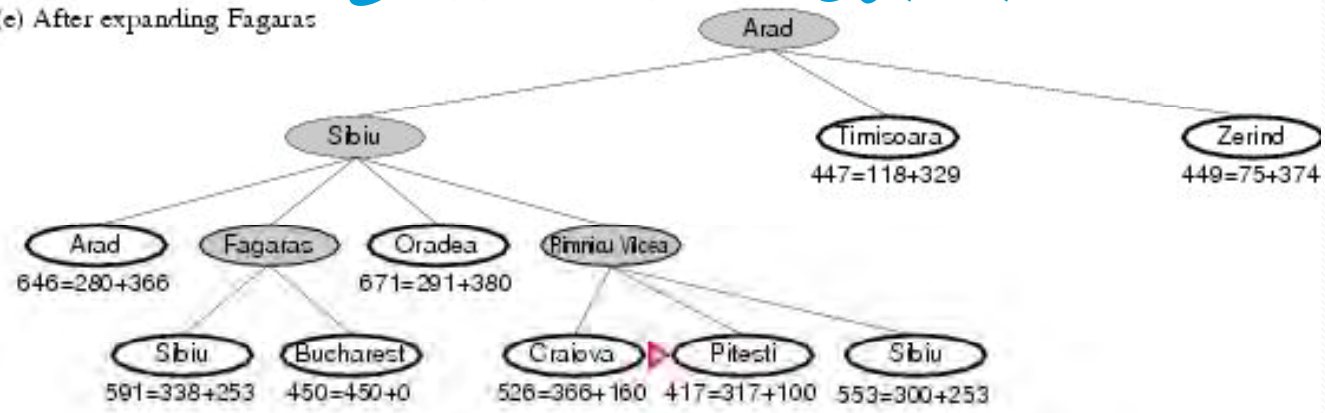
Rimnicu Vilcea را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

f(Craiova)=c(Rimnicu Vilcea, Craiova)+h(Craiova)=360+160=526
f(Pitesti)=c(Rimnicu Vilcea, Pitesti)+h(Pitesti)=317+100=417
f(Sibiu)=c(Rimnicu Vilcea,Sibiu)+h(Sibiu)=300+253=553

بهترین انتخاب شهر Fagaras است

جستجوی A* در نقشه رومانی

(e) After expanding Fagaras



Fagaras را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

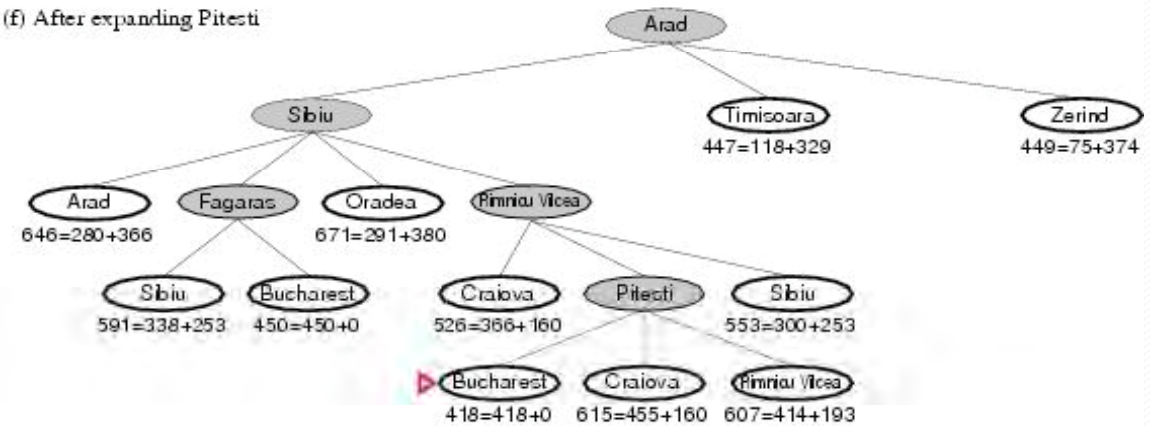
f(Sibiu)=c(Fagaras, Sibiu)+h(Sibiu)=338+253=591

f(Bucharest)=c(Fagaras,Bucharest)+h(Bucharest)=450+0=450

بهترین انتخاب شهر !!! Pitesti است

جستجوی A* در نقشه رومانی

(f) After expanding Pitesti



Pitesti را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

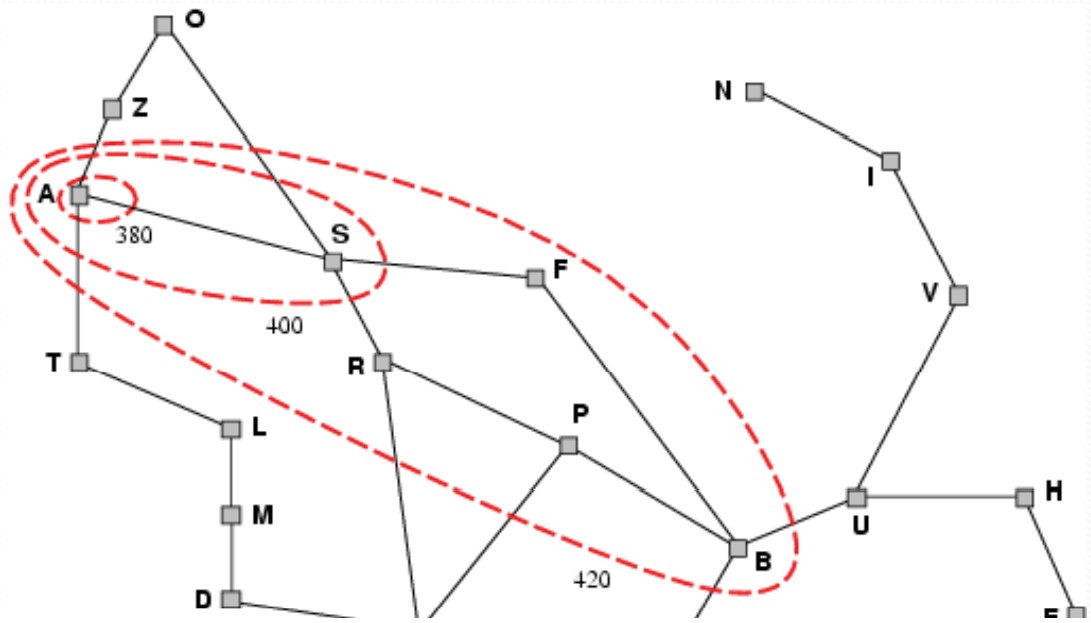
f(Bucharest)=c(Pitesti,Bucharest)+h(Bucharest)=418+0=418

بهترین انتخاب شهر !!! Bucharest است

در اثر جستجوی یکنواخت(سازگاری) اگر C^* هزینه راه حل بهینه باشد در آنصورت:

۱-A^* تمام گره هایی که دارای $f(n)<C^*$ هستند را بسط می دهد

۲-A^* قبل از انتخاب گره هدف ،گره های دقیقا روی کنتور ($f(n)=c^*$) را بسط می دهد



هیورستیک سازگار :جستجوی A^* در نقشه رومانی و کنتورهای بدست آمده در هر کنتور هزینه f گره ها،کوچکتر از محدوده خاص کنتور است

ارزیابی جستجوی A^*

- **کامل بودن:** بله (وابسته به سازگاری یا قابل قبول بودن هیورستیک و روش جستجو گرافی یا درختی)
- **بهینگی:** بله (وابسته به سازگاری یا قابل قبول بودن هیورستیک روش جستجو گرافی یا درختی)
- **پیچیدگی زمانی:** نمایی $O(b^m)$ مگر اینکه خطا در تابع کشف کننده رشدی سریعتر از لگاریتم هزینه مسیر واقعی نداشته باشد، به زبان ریاضی: $| h(n) – h^*(n)| \leq O (\log h^*(n))$
- **پیچیدگی فضا:** تمام گره ها را در حافظه نگه می دارد. $O(b^m)$

❖مشکل اصلی A^* پیچیدگی فضایی که به خصوص در نتیجه GraphSearch و نیاز به نگهداری گره های تولید شده به وجود می آید

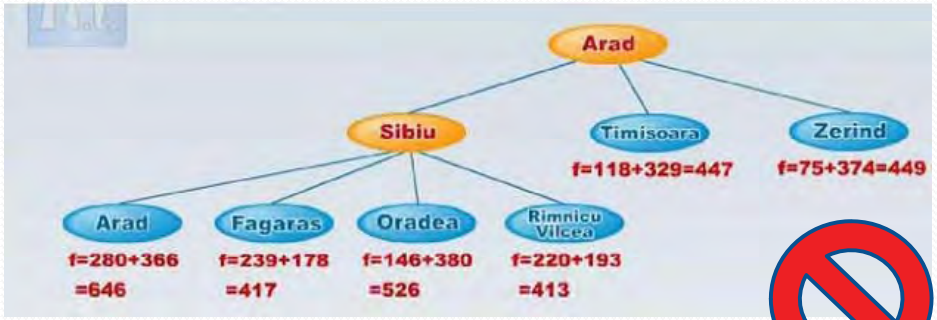
جستجوی هیوریستیک با حافظه محدود

چند راه حل برای مسأله حافظه در A* (با حفظ خصوصیات کامل بودن و بهینگی):

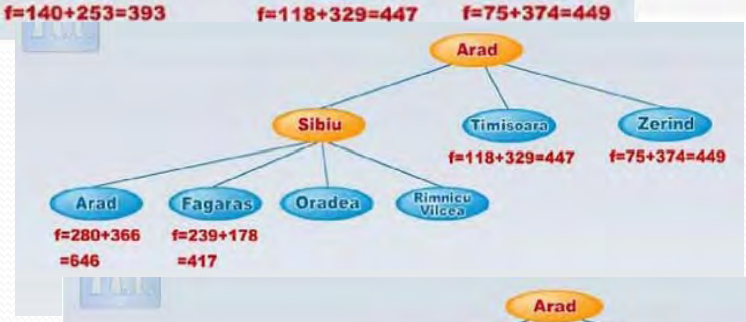
- جستجوی عمیق کننده تکرای A* (IDA*)
  - مانند IDS ولی به جای محدوده عمقی از محدوده *f-cost* ($g+h$) استفاده می شود
- جستجوی اول-بهترین بازگشتی (RBFS)
  - یک الگوریتم بازگشتی با فضای خطی که سعی می کند از جستجوی اول-بهترین استاندارد تقلید کند
- جستجوی (ساده شده) A* با حافظه محدود ((S)MBA*)
  - حذف بدترین گره وقتی که حافظه پر می باشد

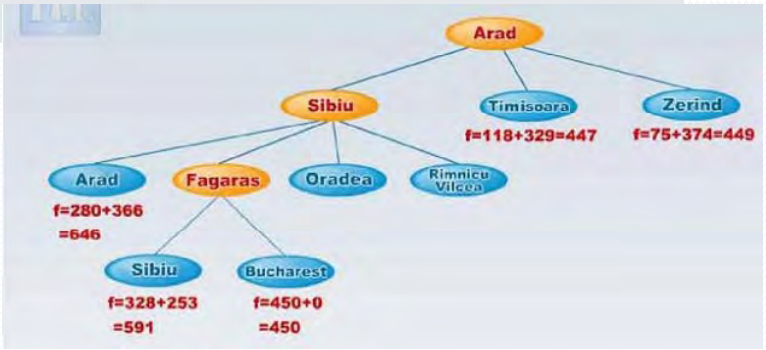
جستجوی اکتشافی با حافظه محدود IDA*

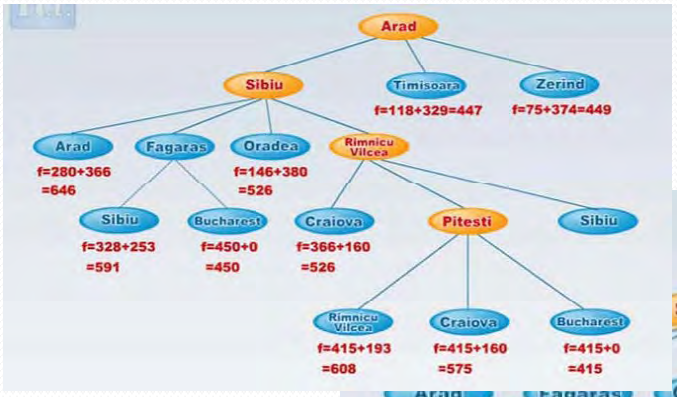
- ساده ترین راه برای کاهش حافظه مورد نیاز A* استفاده از عمیق کننده تکرار در زمینه جست و جوی اکتشافی است.
- الگوریتم عمیق کننده تکرار A* ← IDA*
- در جستجوی IDA* مقدار برش مورد استفاده، عمق نیست بلکه هزینه f(g+h) است.
- IDA* برای اغلب مسئله های با هزینه های مرحله ای، مناسب است و از سربار ناشی از نگهداری صف مرتبی از گره ها اجتناب میکند

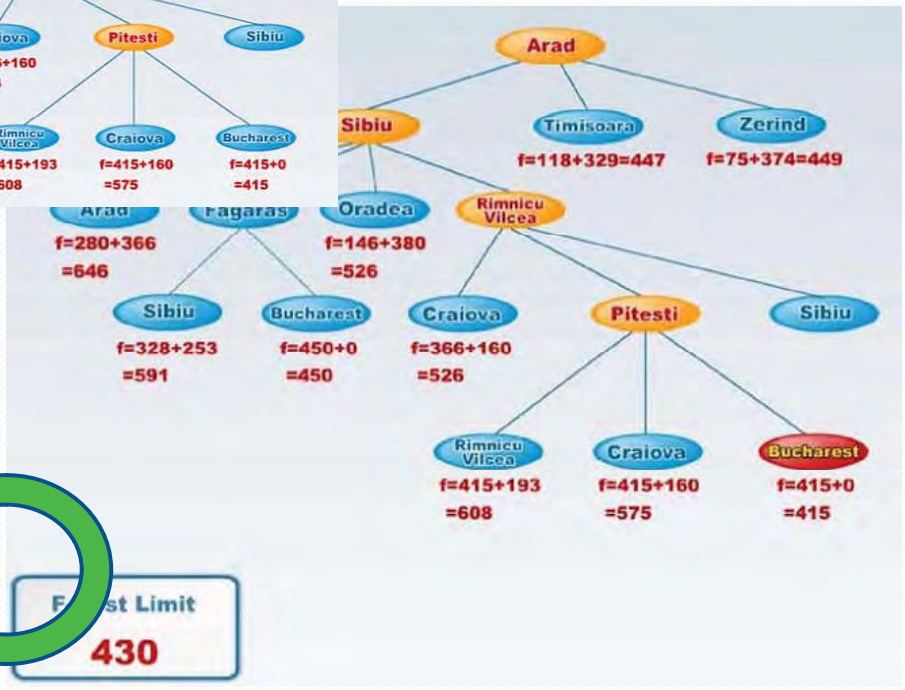


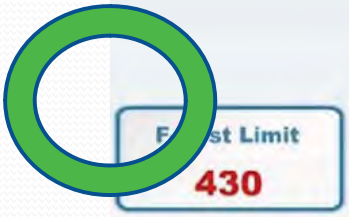
دوباره از صفر با حد 430











ارزیابی IDA*

کامل بودن: بله (اگر A* به جواب برسد ida هم می رسد)

بهینگی: اگر فاصله هزینه پایه، بزرگ انتخاب شود ممکن است بهینه نباشد و اگر پایین انتخاب شود با سرعت کند ،بهینه است

پیچیدگی زمانی: نمایی و به دلیل تکرار از A* کند تر است

پیچیدگی فضا: bd (در A* برابر با b^d بود) (مثل روش عمقی در ناآگاهانه)

بهترین جستجوی بازگشتی RBFS

از IDA* بهتر است، مشکل تولید افراطی گره را دارد .ساختار آن شبیه جست و جوی عمقی بازگشتی است، اما به جای اینکه دائما به طرف پایین مسیر حرکت کند، مقدار f مربوط به بهترین مسیر از هر جد گره فعلی را نگهداری میکند، اگر گره فعلی از این حد تجاوز کند، بازگشتی به عقب برمیگردد تا مسیر دیگری را انتخاب کند.

این جستجو اگر تابع اکتشافی قابل قبولی داشته باشد، بهینه است.

در این جستجو f cost limit برابر هزینه حداقل گره های برادر است.

پیچیدگی فضایی آن O(bd) است

تعیین پیچیدگی زمانی آن به دقت تابع اکتشافی و میزان تغییر بهترین مسیر در اثر بسط گره ها بستگی دارد.

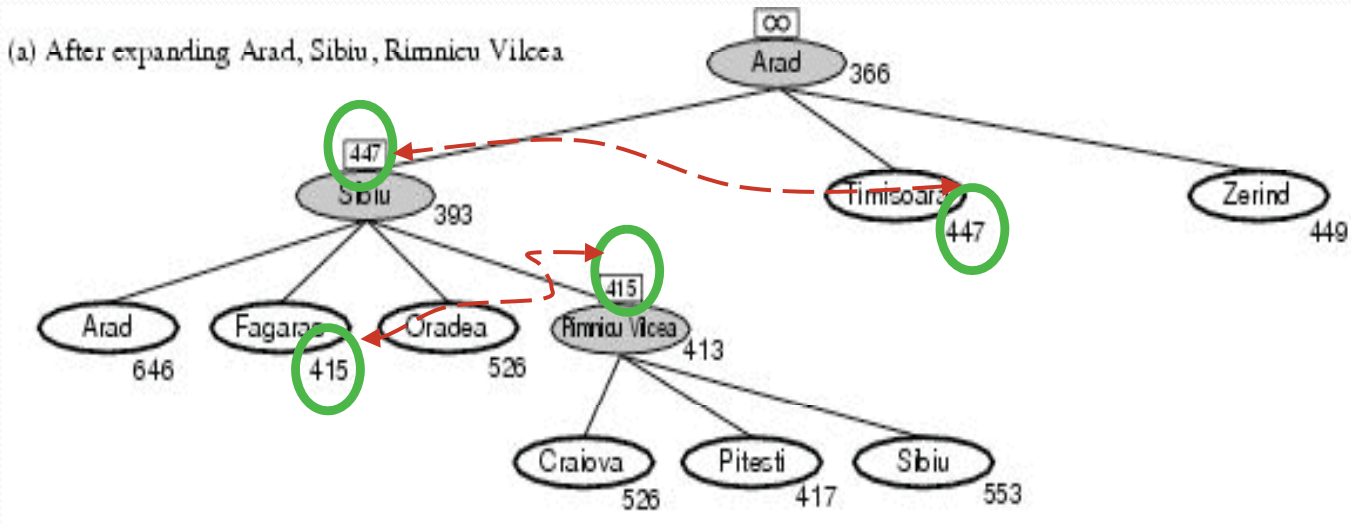
بهترین جستجوی بازگشتی RBFS

- RBFS تا حدی از IDA* کارآمدتر است، اما گره های زیادی تولید میکند.

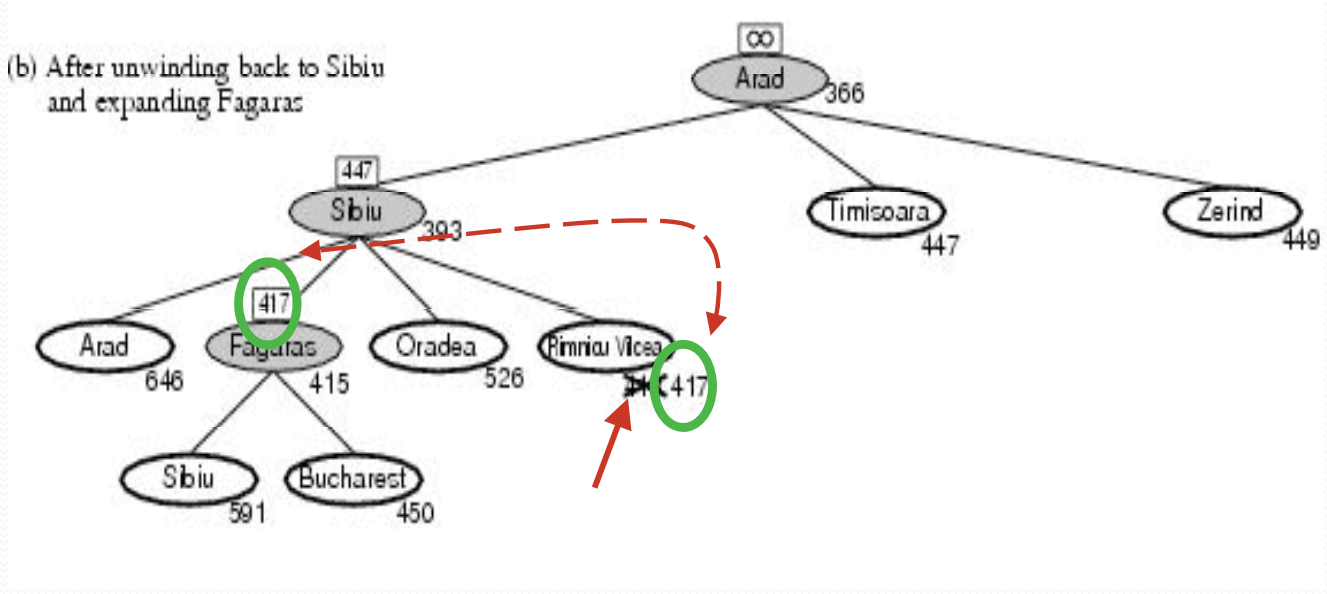
- IDA* و RBFS در معرض افزایش توانی پیچیدگی قرار دارند که در جست و جوی گرافها مرسوم است، زیرا نمیتوانند حالتهای تکراری را در غیر از مسیر فعلی بررسی کنند. لذا، ممکن است یک حالت را چندین بار بررسی کنند.

- IDA* و RBFS از فضای اندکی استفاده میکنند که به آنها آسیب میرساند. IDA* بین هر تکرار فقط یک عدد را نگهداری میکند که فعلی هزینه f است. RBFS اطلاعات بیشتری در حافظه نگهداری میکند

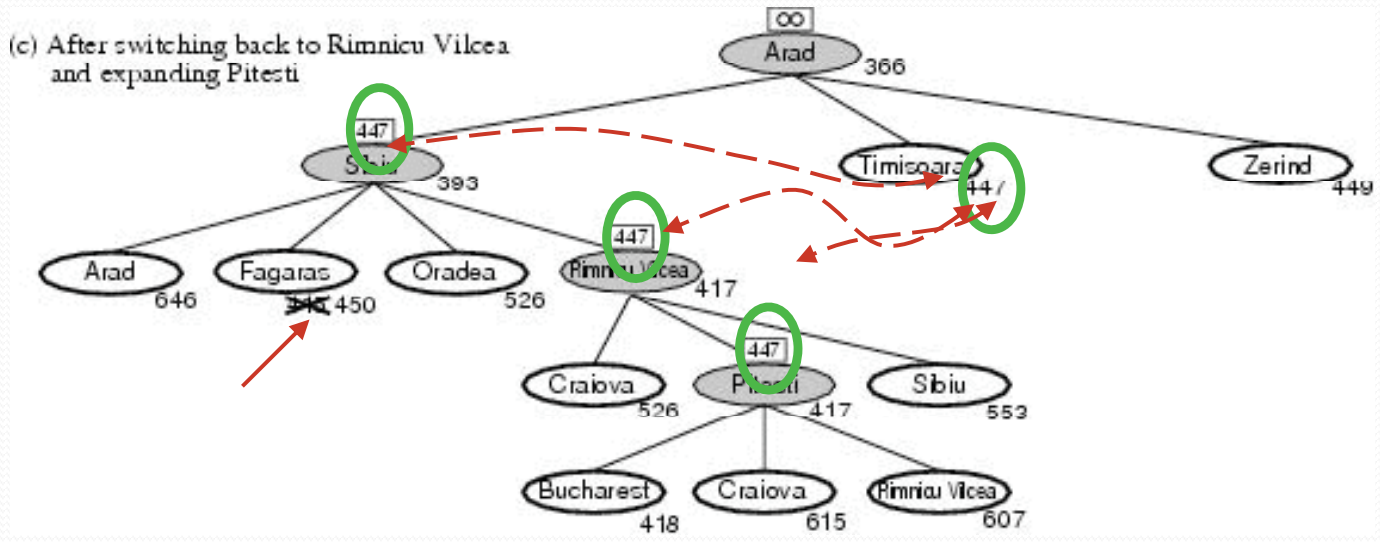
بهترین جستجوی بازگشتی در نقشه رومانی



بهترین جستجوی بازگشتی در نقشه رومانی



بهترین جستجوی بازگشتی در نقشه رومانی



ارزیابی RBFS

- RBFS کمي کارآتر از IDA* مي باشد
  - هنوز گسترش اضافي گره ها وجود دارد(تغییر عقیده)
- RBFS هم مانند A*، اگر $h(n)$ قابل قبول باشد بهینه است
- پیچیدگي حافظه $O(bd)$
  - IDA* فقط یک عدد را نگهداری مي کند (حد فعلي f-cost)
- تعیین پیچیدگي زماني مشکل است
  - به دقت تابع هیوریستیک و میزان تغییر بهترین مسیر در اثر بسط گره ها بستگي دارد.
  - مانند IDA* در معرض افزایش نمایي پیچیدگي زماني قرار دارد
- RBFS و IDA* هر دو از میزان بسیار کم حافظه رنج مي برند.
  - با اینکه حافظه زیادي وجود دارد، نمي توانند از ان استفاده کنند.

جستجوی حافظه محدود ساده SMA*

- SMA* بهترین برگ را بسط میدهد تا حافظه پر شود. در این نقطه بدون از بین بردن گره های قبلی نمیتواند گره جدیدی اضافه کند

- SMA* همیشه بدترین گره برگ را حذف میکند و سپس از طریق گره فراموش شده به والد آن بر میگردد. پس جد زیر درخت فراموش شده، کیفیت بهترین مسیر را در آن زیر درخت میداند

- اگر عمق سطحی ترین گره هدف کمتر از حافظه باشد, کامل است.

- SMA* بهترین الگوریتم همه منظوره برای یافتن حلهای بهینه میباشد

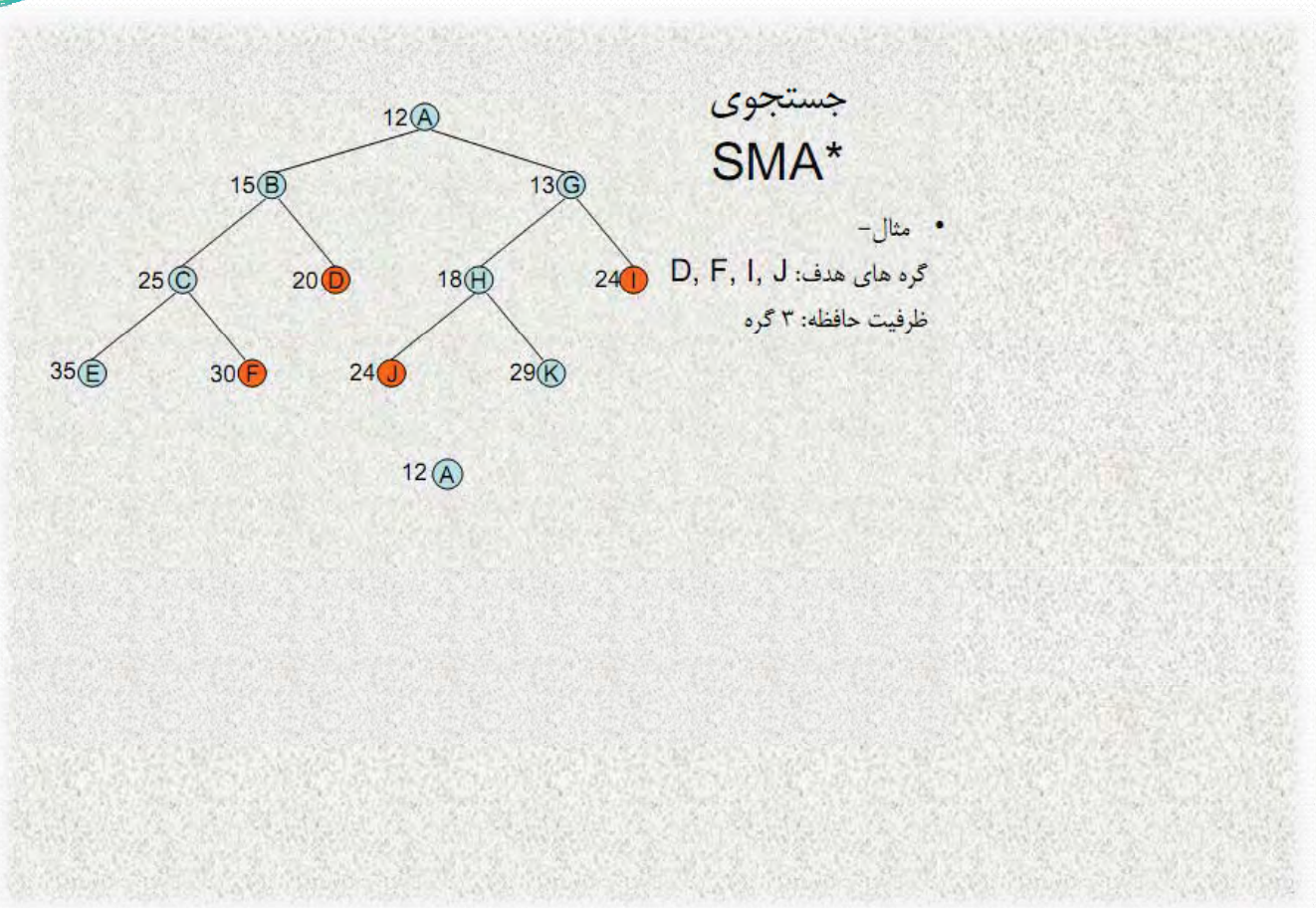
جستجوی حافظه محدود ساده SMA*

اول بدترین حالت از گره ها حذف مي شود

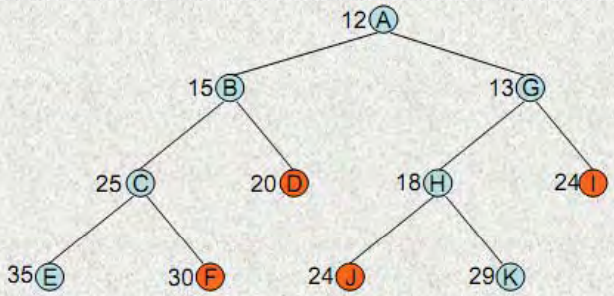
- اگر مقدار f تمام برگها یکسان باشد و الگوریتم یک گره را هم برای بسط و هم برای حذف انتخاب کند، SMA* این مسئله را با بسط بهترین برگ جدید و حذف بهترین برگ قدیمی حل میکند

- ممکن است SMA* مجبور شود دائما بین مجموعه ای از مسیرهای حل کاندید تغییر موضع دهد، در حالی که بخش کوچکی از هر کدام در حافظه جا شود

- محدودیتهای حافظه ممکن است مسئله ها را از نظر زمان محاسباتی، غیر قابل حل کند.



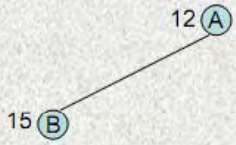
جستجوی SMA*



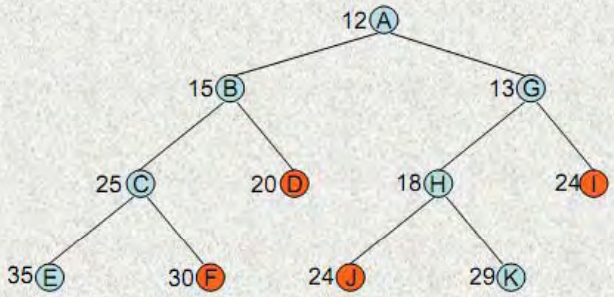
- مثال-

گره های هدف: D, F, I, J

ظرفیت حافظه: ۳ گره



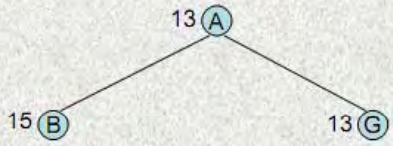
جستجوی SMA*



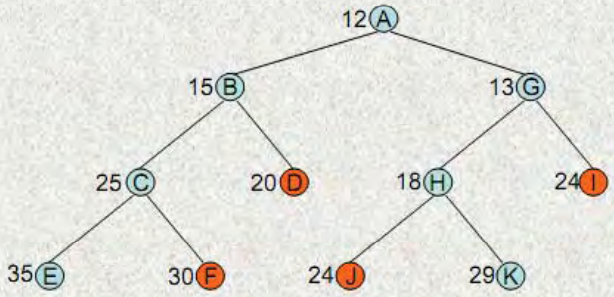
- مثال-

گره های هدف: D, F, I, J

ظرفیت حافظه: ۳ گره



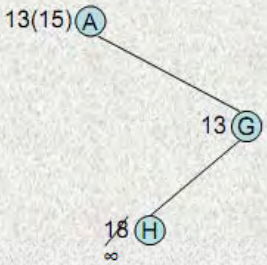
جستجوی
SMA*



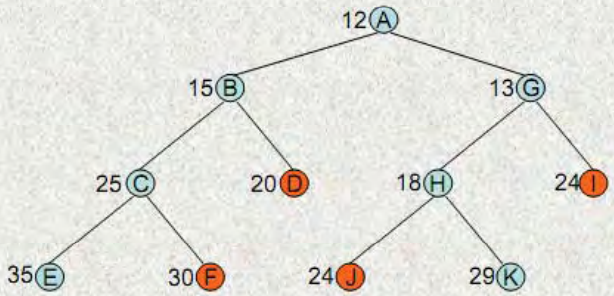
- مثال-

گره های هدف: D, F, I, J

ظرفیت حافظه: ۳ گره



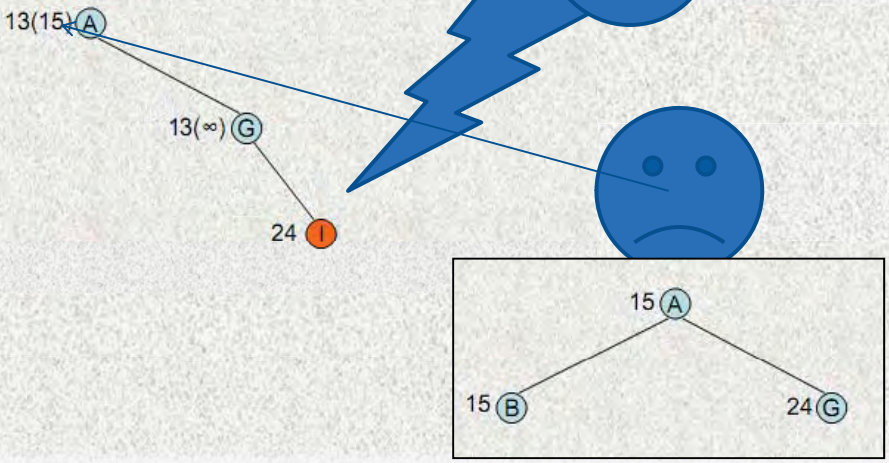
جستجوی
SMA*



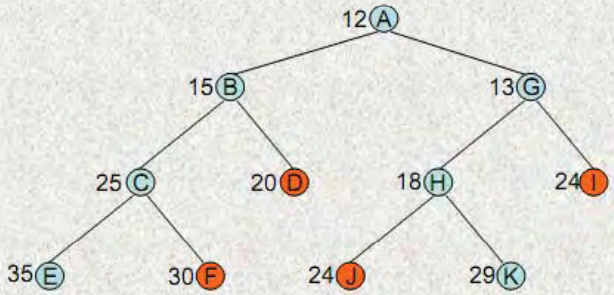
- مثال-

گره های هدف: D, F, I, J

ظرفیت حافظه: ۳ گره



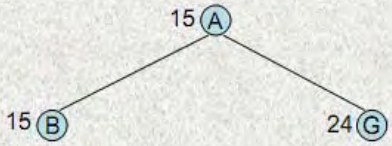
جستجوی
SMA*



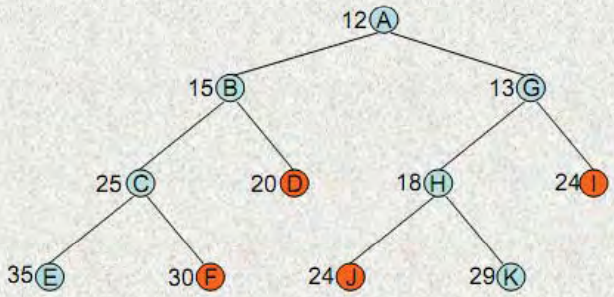
- مثال-

گره های هدف: D, F, I, J

ظرفیت حافظه: ۳ گره



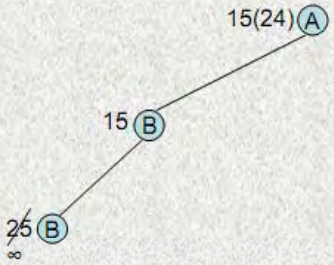
جستجوی
SMA*



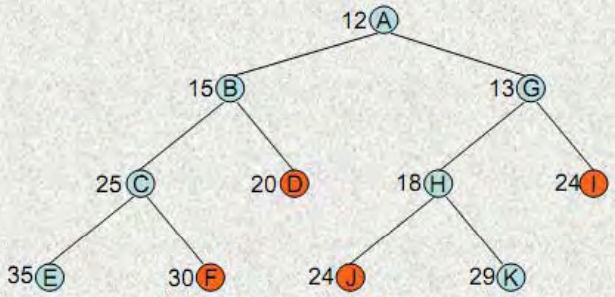
- مثال-

گره های هدف: D, F, I, J

ظرفیت حافظه: ۳ گره



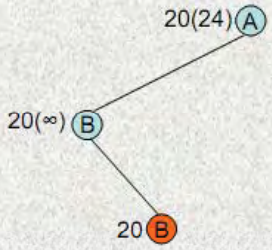
جستجوی SMA*



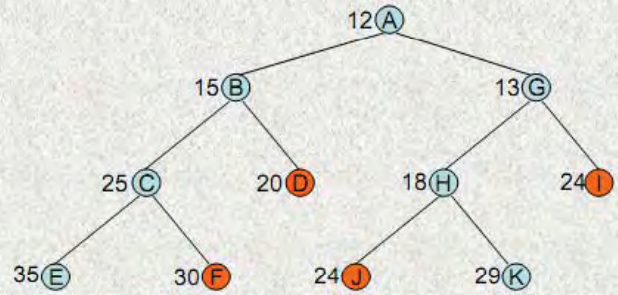
- مثال-

گره های هدف: D, F, I, J

ظرفیت حافظه: ۳ گره



جستجوی SMA*



- مثال-

گره های هدف: D, F, I, J

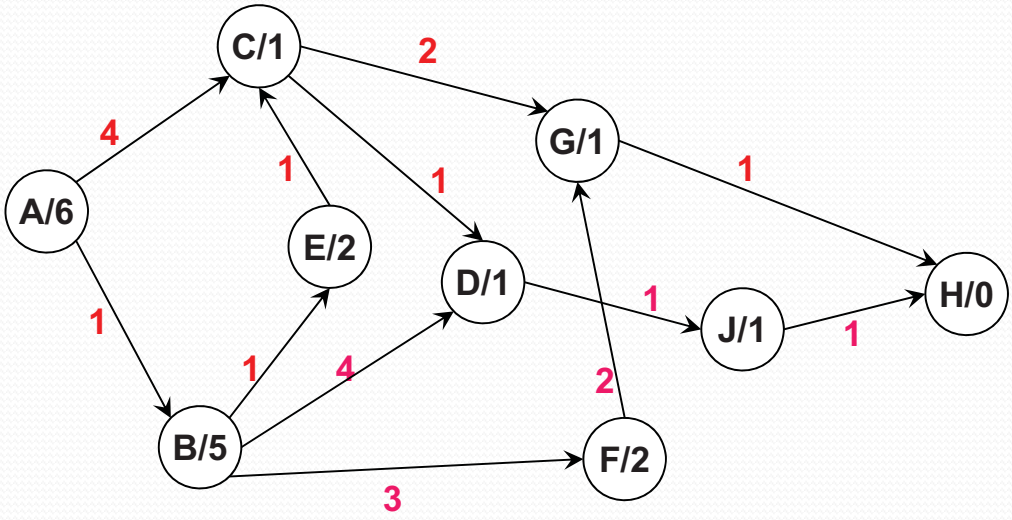
ظرفیت حافظه: ۳ گره

- در مثال فوق، حافظه کافی برای کم عمق ترین مسیر بهینه وجود دارد.
- اگر J هزینه ۱۹ به جای ۲۴ داشت الگوریتم قادر به یافتن آن نبود.
- با داشتن فضای حافظه منطقی، الگوریتم قادر به حل مسائل مشکل تری نسبت به A* می باشد بدون آنکه مشکل سرریزی گره های اضافی مطرح باشد.

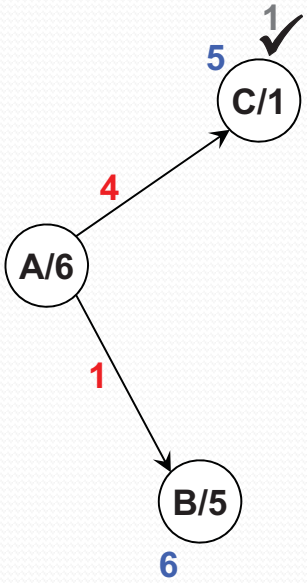
ارزیابی

- کامل بودن: در صورت وجود حافظه کافی برای ذخیره کم عمق ترین مسیر، کامل است.
- بهینه بودن: در صورت وجود حافظه کافی برای ذخیره بهترین مسیر، بهینه است، در غیر اینصورت بهترین مسیر در محدوده حافظه موجود را بر می گرداند.
- از تکرار محاسبات تا جائیکه حافظه اجازه دهد جلوگیری می کند.
- می تواند از تمام حافظه مجاز استفاده کند.
- بنابراین الگوریتم کارا-بهینه است در صورتیکه حافظه کافی برای ذخیره درخت جستجوی کامل در دسترس باشد

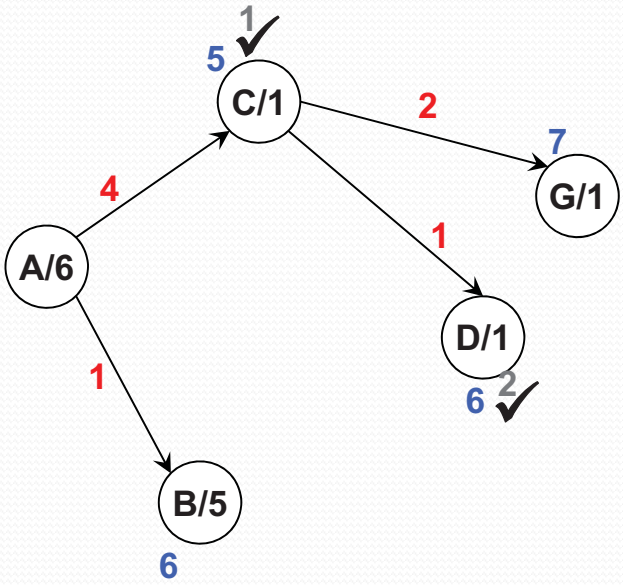
جستجوی گراف با A*



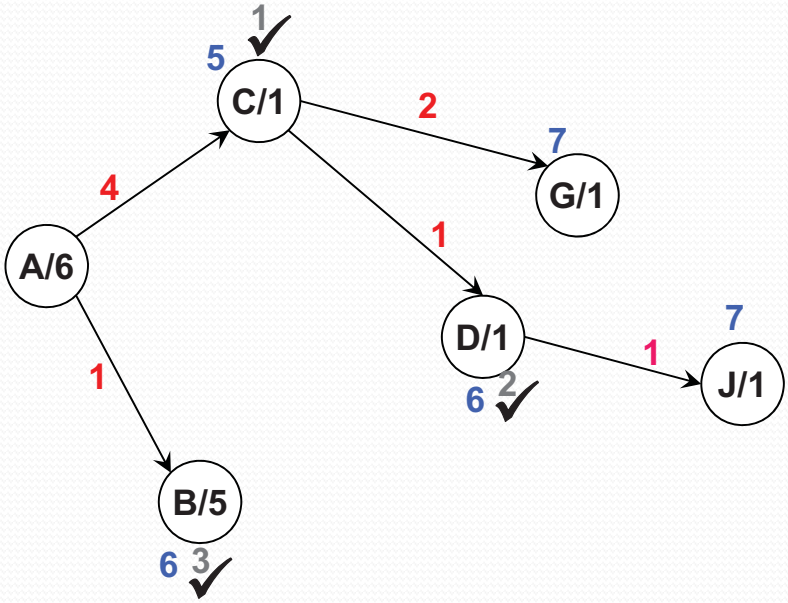
جستجوی گراف با A*



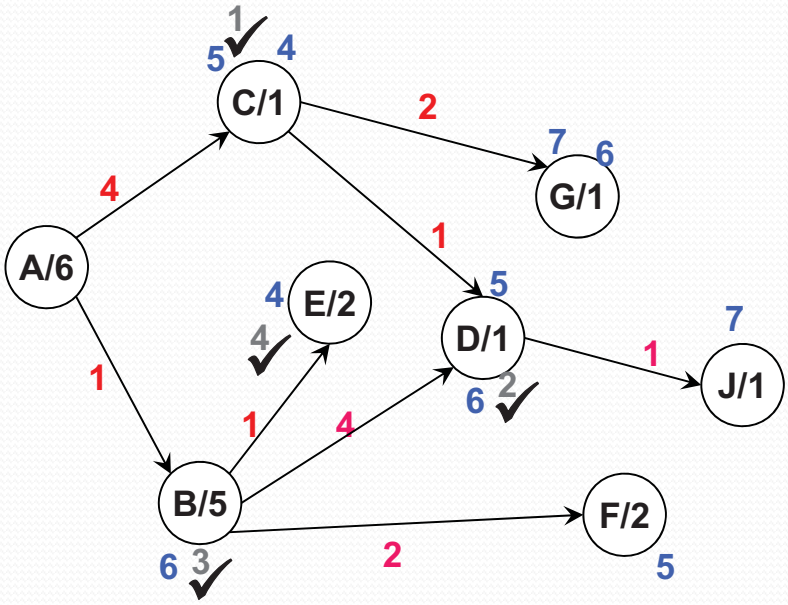
جستجوی گراف با A*



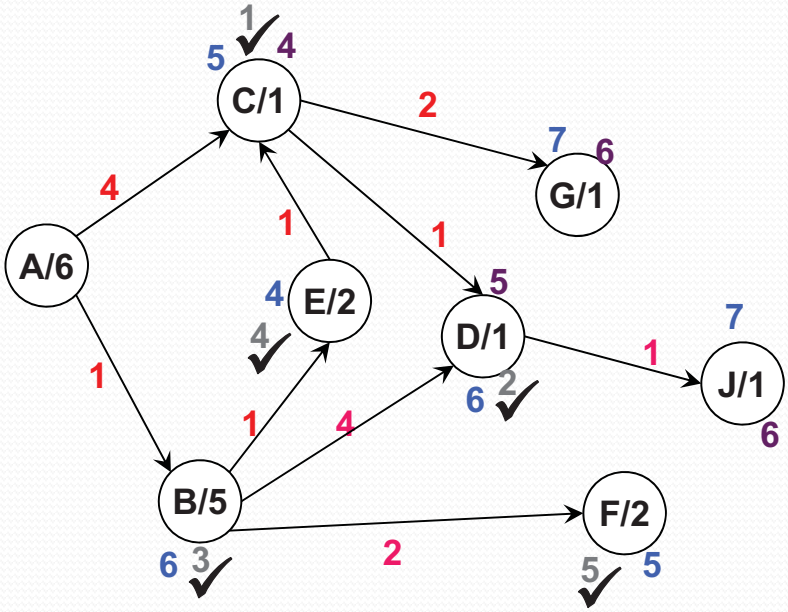
جستجوی گراف با A*



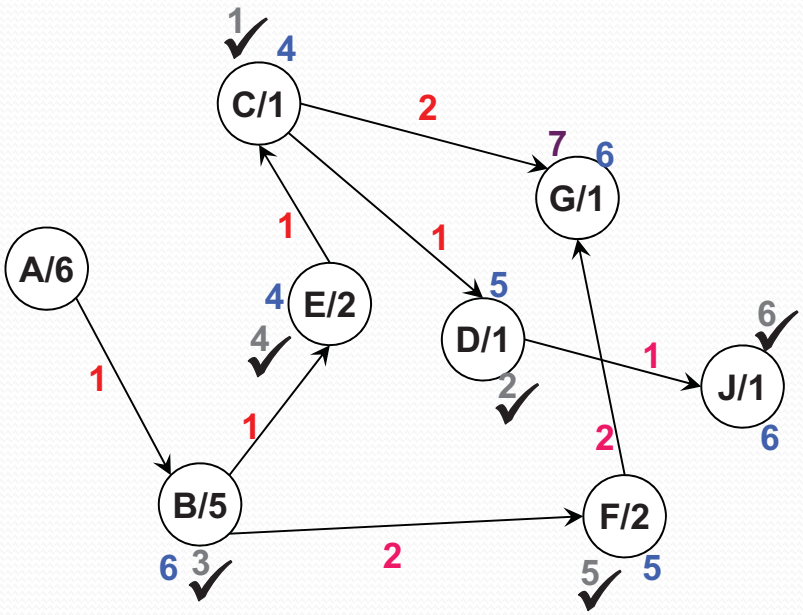
جستجوی گراف با A*



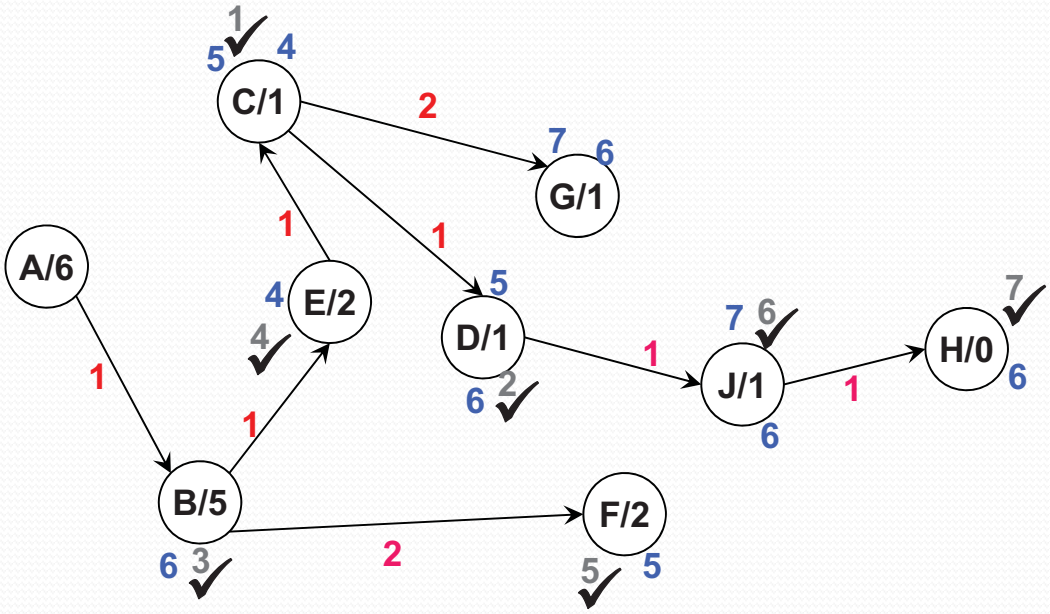
جستجوی گراف با A*



جستجوی گراف با A*



جستجوی گراف با A*



فراگیری برای جست و جوی بهتر

- روشهای جست و جوی قبلی، از **روشهای ثابت** استفاده میکردند.
- عامل با استفاده از **فضای حالت فراسطحی** میتواند یاد بگیرد که بهتر جست و جو کند
- هر حالت در **فضای حالت فرا سطحی**، حالت(محاسباتی) داخلیِ برنامه ای را تسخیر میکند که فضای **حالت سطح شیء**، مثل رومانی را جست و جوَ میکند
- الگوریتم **یادگیری فراسطحی** میتواند چیزهایی را از تجربیات بیاموزد تا زیردرختهای غیر قابل قبول و بدون امید را کاوش نکند.
- هدف یادگیری، **کمینه کردن کل هزینه**، حل مسئله است

توابع هیورستیک(اکتشافی)

- **مثال برای معمای۸**
  - میانگِین هزینه حل تقریبا ۲۲ مرحله و فاکتور انشعاب در حدود ۳ است.
  - جست و جوی جامع تا عمق ۲۲ : $3^{22} \approx 3.1 \times 10^{10}$
  - با انتخاب یک تابع اکتشافی مناسب میتوان مراحل جستجو را کاهش داد

2	8	3
1	6	4
7		5

Start State

1	2	3
8		4
7	6	5

Goal State

  - اگر بخواهیم کوتاهترین راه حل را با **A*** پیدا کنیم به تابع هیورستیک نیازمندیم
  - با انتخاب یک تابع هیورستیک مناسب میتوان مراحل جستجو را کاهش داد

دو روش هیورستیکی متداول برای معمای۸

h1 تعداد کاشیها در مکانهای نادرست

h2 مجموعه فواصل کاشیها از موقعیتهای هدف آنها

h1 اکتشاف قابل قبولی است، زیرا هر کاشی که در جای نامناسبی قرار دارد، حداقل یکبار باید جابجا شود

در **h2** چون کاشیها نمیتوانند در امتداد قطر جا به جا شوند, فاصله ای که محاسبه میکنیم مجموع فواصل افقـی و عمودی است. این فاصله را فاصله بلوک شهر یا فاصله مانهاتان مینامند.

2	8	3
1	6	4
7		5

Start State

1	2	3
8		4
7	6	5

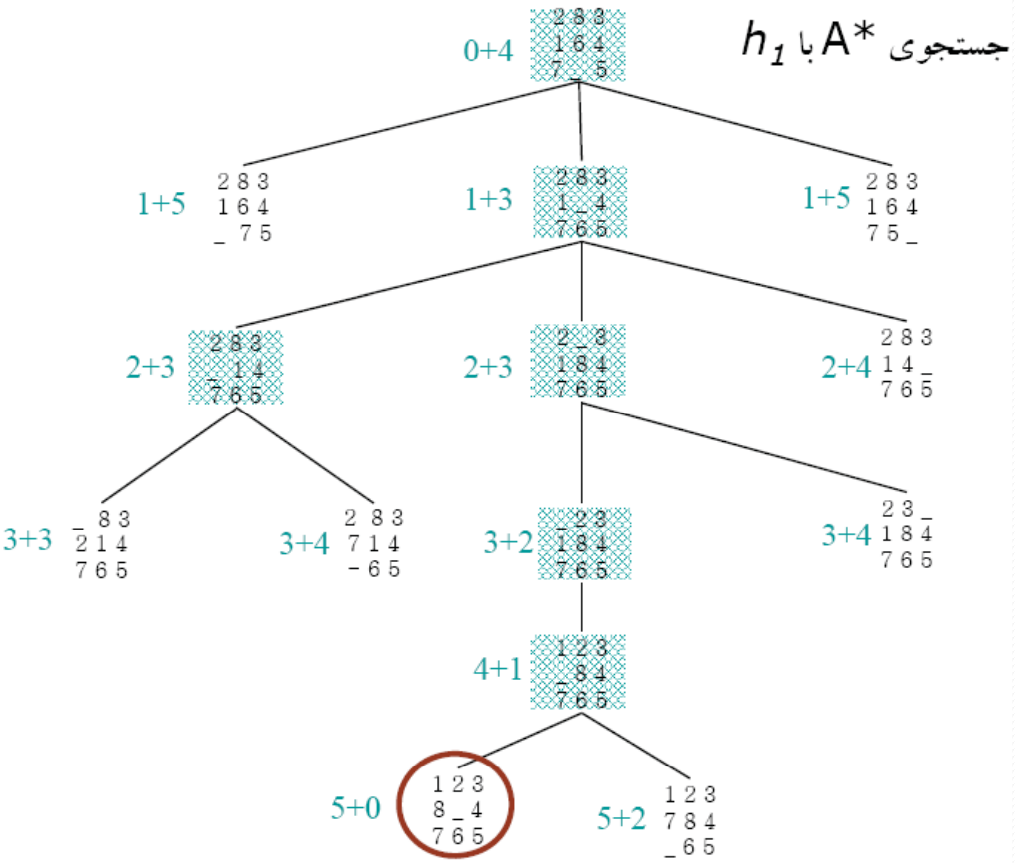
Goal State

h2=1+1+0+0+0+0+1+2=5

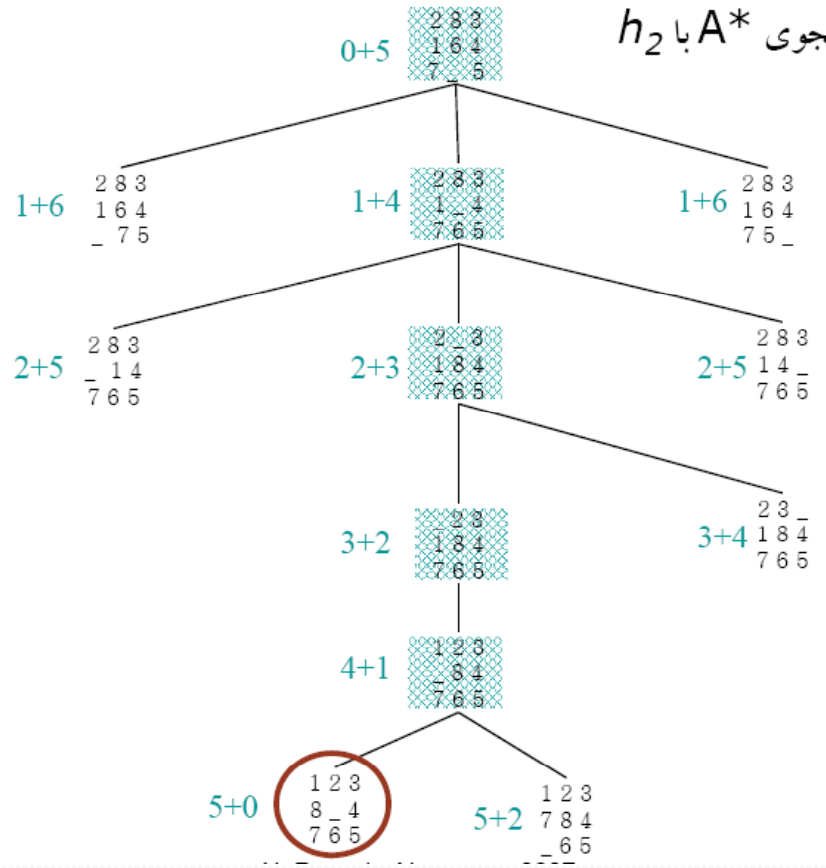
h1=4

- اگر بازاء هر n داشته باشیم، $h_2(n) \geq h_1(n)$
  (و هر دو هیوریستیک قابل قبول باشند)
- آنگاه h_2 بر h_1 تسلط دارد و برای جستجو بهتر می باشد.

جستجوی A* با h_1



مثال: جستجوی A* با h_2



اثر دقت هیورستیک بر کارایی

ضریب موثر انشاب b*

- اگر تعداد گره هایی که برای یک مسئله خاص توسط A* تولید میشود برابر با N و عمق راه حل برابر با d باشد، آن گاه b* **فاکتور انشعابی** است که درخت یکنواختی به عمق d باید داشته باشد تا حاوی N+1 گره باشد

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

- فاکتور انشعاب مؤثر معمولاً برای **مسئله های سخت ثابت** است
- اندازه گیریهای تجربی b* بر روی مجموعه کوچکی از مسئله ها میتواند **راهنمای خوبی برای مفید بودن** اکتشاف باشد
- مقدار b* در اکتشافی با طراحی خوب، نزدیک ۱ است

- **مثال:** اگر A* راه حلی را در عمق ۵ با استفاده از ۵۲ گره پیدا کند، فاکتور انشعاب موثر را محاسبه کنید.

پاسخ:

$$53 = 1 + b^* + (b^*)^2 + \ldots (b^*)^5 \rightarrow b^* = 1.92$$

هزینه جست و جو

فاکتور انشعاب مؤثر

d	Search Cost IDS	Search Cost $A^*(h_1)$	Search Cost $A^*(h_2)$	Effective Branching Factor IDS	Effective Branching Factor $A^*(h_1)$	Effective Branching Factor $A^*(h_2)$
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

میانگین گره های بسط یافته در جستجوی IDS و A* و فاکتور انشعاب مؤثر با استفاده از h1 و h2 مشخص است هیورستیک h2 به کار رفته برتری دارد

برتری در توابع هیورستیک

- اگر برای هر گره n داشته باشیم: h2(n) >= h1(n)
  - h2 بر h1 غالب است
  - غالب بودن مستقیما به کارایی ترجمه میشود
  - تعداد گره هایی که با بکارگیری h2 بسط داده میشود، هرگز بیش از بکارگیری h1 نیست

همیشه بهتر است از تابع هیورستیک با مقادیر بزرگ استفاده کرد، به شرطی که زمان محاسبه اکتشاف، خیلی بزرگ نباشد

ابداع توابع هیورستیک قابل قبول

- می توان هیوریستیکهای قابل قبول را برای یک مسأله، از **هزینه دقیق** راه حل یک نسخه **راحت (relaxed)** از مسأله بدست آورد. تعدیل شده(مثل خط مستقیم در رومانی)
- مثال: قانون معمای هشت. یک کاشی می تواند از خانه A به خانه B برود، اگر A مجاور B باشد و B خالی باشد.
- اگر قوانین معمای هشت به گونه ای راحت شوند که یک کاشی بتواند **به هر خانه ای** حرکت کند، هیوریستیک $h_1(n)$ کوتاهترین راه حل را می دهد.
- اگر قوانین به گونه ای راحت شوند که یک کاشی بتواند **به هر خانه مجاور** حرکت کند، هیوریستیک $h_2(n)$ کوتاهترین راه حل را می دهد.
- **نکته کلیدی:** هزینه راه حل بهینه یک مسأله راحت، بیشتر از هزینه راه حل بهینه در مسأله واقعی نیست.

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
- دو امتیاز عمده جست و جوهای محلی
  - استفاده از حافظه کمکی
  - ارائه راه حلهای منطقی در فضاهای بزرگ و نامتناهی
- این الگوریتمها برای حل مسائل بهینه سازی نیز مفیدند
  - یافتن بهترین حالت بر اساس تابع هدف

الگوریتم های جستجوی محلی و مسائل بهینه سازی

•الگوریتمهای جستجوی محلی از محل و وضعیت جاری استفاده می کنند. و به همسایگان آن حالت تغییر مکان می دهند، مسیر راه حل اهمیت ندارد؛ خود حالت هدف پاسخ مسأله می باشد

هدف یافتن یک پیکره بندی که محدودیت های مسأله را ارضاء کند، مانند مساله n – وزیر در چنین مواردی می توان از الگوریتم های جستجوی محلی بهره گرفت.

یک حالت را به تنهایی در نظر بگیر؛ سعی کن آن را بهبود بخشی.

جستجوی محلی = استفاده از یک حالت فعلی و حرکت به حالت های همسایه

- مزایا:

1. استفاده از حافظه بسیار کم
2. یافتن راه حل های معقول در اغلب موارد در فضاهای حالت بزرگ و یا نامحدود

•مفید برای مسائل بهینه سازی محض و خالص

در رویکردهای قبلی به گشتن در کل فضای حالت می‌پرداختیم و استراتژی جستجو (هیوریستیک) تنها کمک می‌کرد تا ترتیب گشتن مشخص باشد.

در این رویکرد شاید نیازی به گشتن کل فضای حالت نباشد.

دورنمای فضای حالت

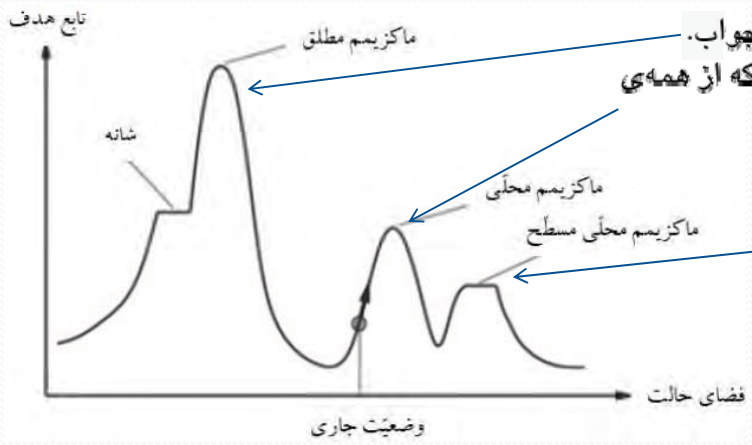
برای درک جستجوی محلی توجه به دورنمای فضای حالت مفید است اگر ارتفاع متناظر با هزینه باشد هدف یافتن عمیقترین دره و اگر ارتفاع متناظر با تابع هدف باشد هدف یافتن بلندترین قله (ماکزیمم مطلق) است

تابع هدف (Objective Function): تابع تعیین کننده‌ی مقدار خوب بودن جواب.

نقطه / نقاط بهینه‌ی کلی (Global Optimum): بهترین جواب.

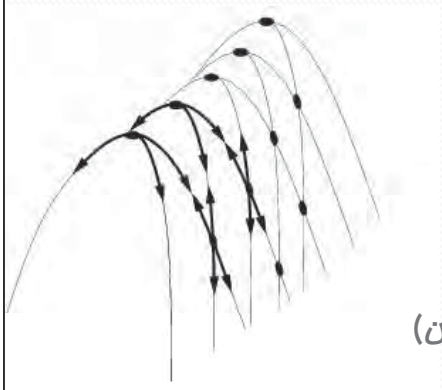
نقطه / نقاط بهینه‌ی محلی (Local Optimum): جوابی که از همه‌ی همسایه‌هایش بهتر است.

فلات تمام نقاط یک ارزش دارند



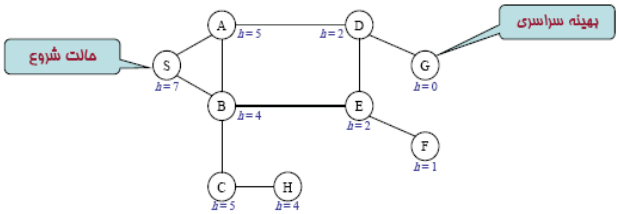
جست و جوی تپه نوردی

- حلقه ای که در جهت افزایش مقدار حرکت میکند(بطرف بالای تپه)
  - رسیدن به بلندترین قله در همسایگی حالت فعلی، شرط خاتمه است.
- ساختمان داده گره فعلی، فقط حالت و مقدار تابع هدف را نگه میدارد
- تپه نوردی گاها جستجوی حریصانه محلی نیز نام دارد
  - بدون فکر قبلی حالت همسایه خوبی را انتخاب میکند
  - تپه نوردی به دلایل زیر میتواند متوقف شود:
    - بیشینه محلی
    - برآمدگی ها (دماغه ها)=
      رشته ای از بیشنه های محلی غیر متصل (باعث حرکت پایین)
    - فلات=تابع ارزیاب ثابت است

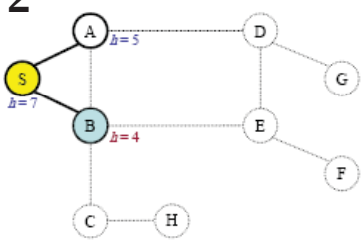


جست و جوی تپه نوردی

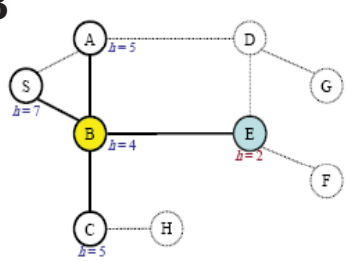
1



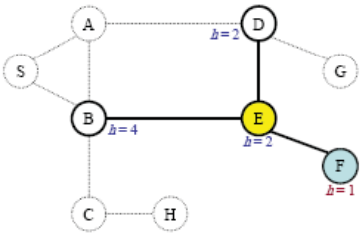
2



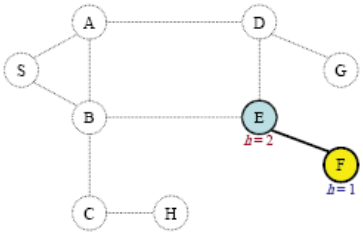
3



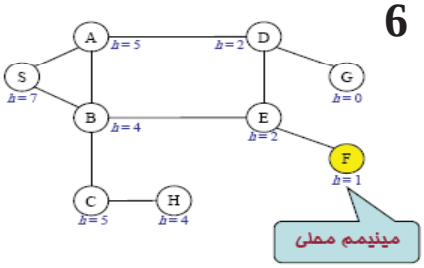
4



5

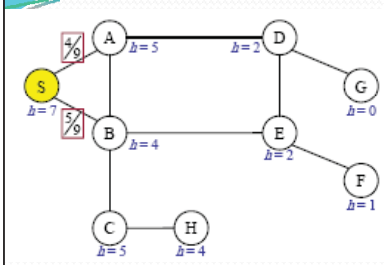
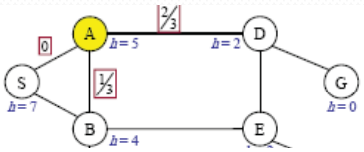
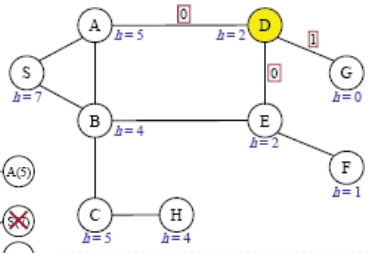
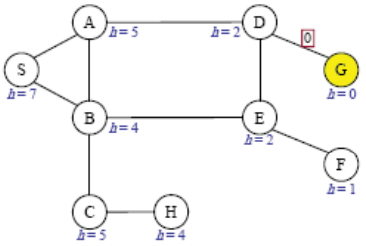
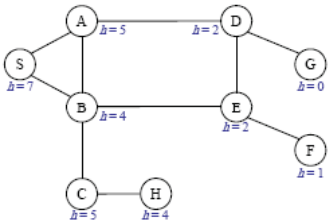
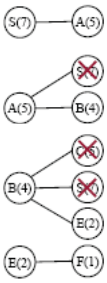


6



انواع تپه نوردی

- تپه نوردی اتفاقی
  - انتخاب تصادفی در میان حرکت های رو به بالا
  - احتمال انتخاب می تواند متناسب با شیب حرکت تغییر کند

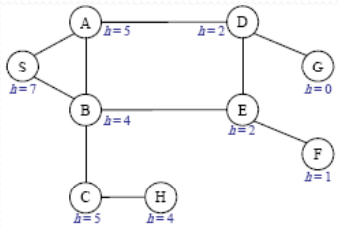







تپه نوردی اولین انتخاب

- همان تپه نوردی اتفاقی که حالت های بعدی را به طور تصادفی تولید می کند تا یکی از آنها بهتر از حالت فعلی باشد

تپه نوردی با شروع مجدد تصادفی (یادآوری: اگر شکست خوردی دوباره سعی کن)

- سعی می کند که از گیر افتادن در ماکزیمم محلی اجتناب کند



$S \rightarrow B \rightarrow E \rightarrow F$ ✗

$C \rightarrow B \rightarrow E \rightarrow F$ ✗

H ✗

$C \rightarrow H$ ✗

$A \rightarrow D \rightarrow G$

در نا امیدی بسی امید است

الگوریتم Simulated Annealing یا سردکردن برنامه‌ریزی شده

جستجوی سخت سازی شبیه سازی شده

اگر به الگوریتم اجازه دهیم تا گاهی اوقات به جای پیداکردن جواب بهتر، نقاط بدتری را پیدا کند، این شانس را داریم که از بعضی قله‌ها پایین آمده و به قله‌ی بالاتر برویم.

کاربردها:

- حل مسائل VLSI
- برنامه ریزی
- اعمال بهینه سازی
- زمانبندی خطوط هوایی

جست و جوی پرتو محلی

- مثل الگوریتم تپه نوردی فقط به جای یک حالت، k حالت را نگهداری میکند
- به جای شروع از حالت جاری از چند حالت شروع می کند

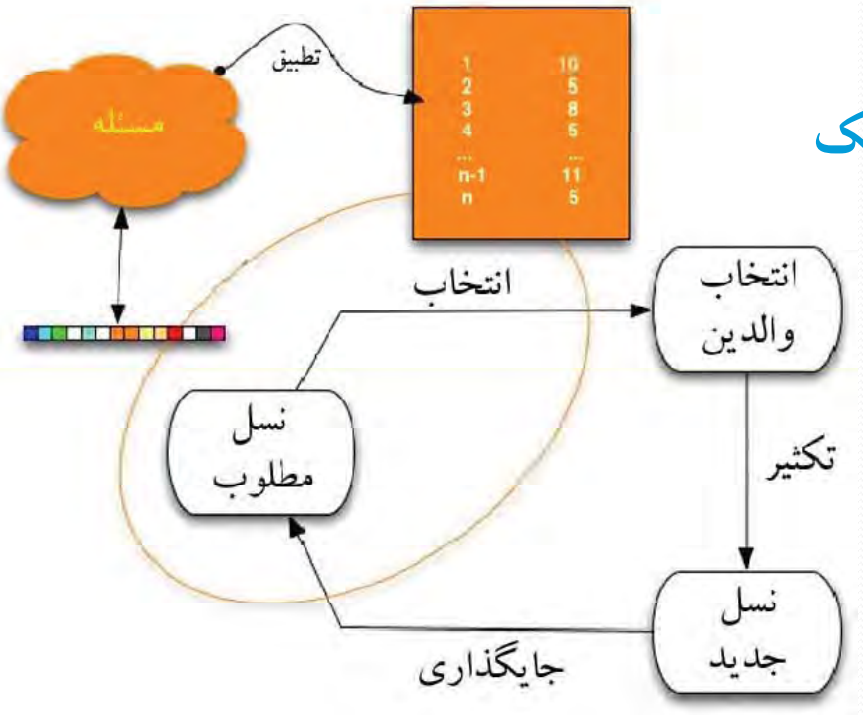
شروع با k حالت که به طور تصادفی ایجاد شده اند.

در هر تکرار، تمام فرزندان برای هر k حالت تولید می شوند.

اگر یکی از آنها حالت هدف بود جستجو متوقف می شود و در غیر این صورت از میان لیست کامل فرزندان k تا از بهترین ها انتخاب می شوند و مرحله بالا تکرار می شود.

- تفاوت عمده با جستجوی شروع مجدد تصادفی
  - در جست و جوی شروع مجدد تصادفی، هر فرایند مستقل از بقیه اجرا میشود
  - در جست و جوی پرتو محلی، اطلاعات مفیدی بین k فرایند موازی مبادله میشود
- جست و جوی پرتو اتفاقی(غیر قطعی)
  - به جای انتخاب بهترین k از جانشینها، k جانشین تصادفی را بطوریکه احتمال انتخاب یکی تابعی صعودی از مقدارش باشد، انتخاب میکند(مثل تپه نوردی اتفاقی)

الگوریتم های ژنتیک



شکلی از جست و جوی پرتو غیر قطعی که حالتهای جانشین از طریق ترکیب دو حالت والد تولید میشود این کار تا تولید بهترین حالت ادامه می یابد

الگوریتم های ژنتیک(GA)

یک حالت بعدی با ترکیب دو حالت پدر ایجاد می شود.

شروع با k حالت تصادفی (جمعیت)

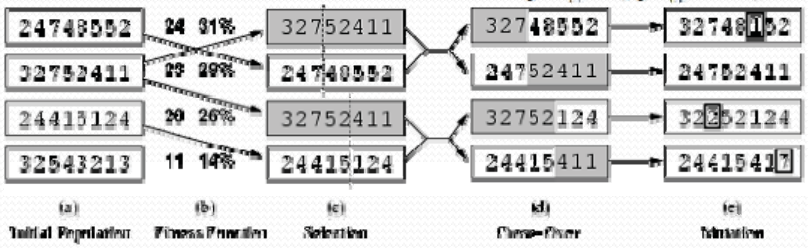
یک حالت با یک رشته بر روی یک مجموعه محدود بازنمایی می شود

(اغلب رشته ای از صفر و یک) – (کروموزوم)

تابع ارزیابی (تابع برازندگی – Fitness function) : مقادیر بالاتری را برای حالات بهتر ایجاد می کند.

نسل بعدی جمعیت با انجام اعمال زیر روی جمعیت فعلی تولید می شود:

- انتخاب (Selection)
- آمیزش (Crossover)
- جهش (Mutation)



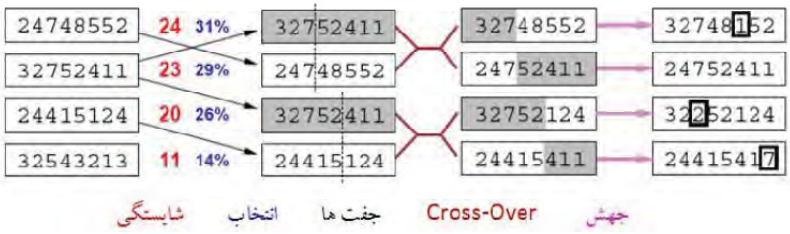
- تابع برازندگی : تعداد جفت وزیر هایی که یکدیگر را تهدید نمی کنند (مینیمم: صفر، ماکزیمم: $8 * 7/2 = 28$)
- $24 / (24+23+20+11) = 31\%$
- $23 / (24+23+20+11) = 29\%$

تابع برازندگي ممکن طبق آستانه خاصي بعضي نسلها را انتخاب مي کند

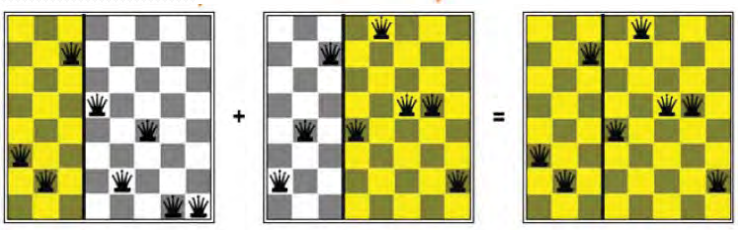
نقطه پيوند تصادفي براي آميزش هر جفت تعيين مي شود

احتمال جهش تصادفي در فرزندها

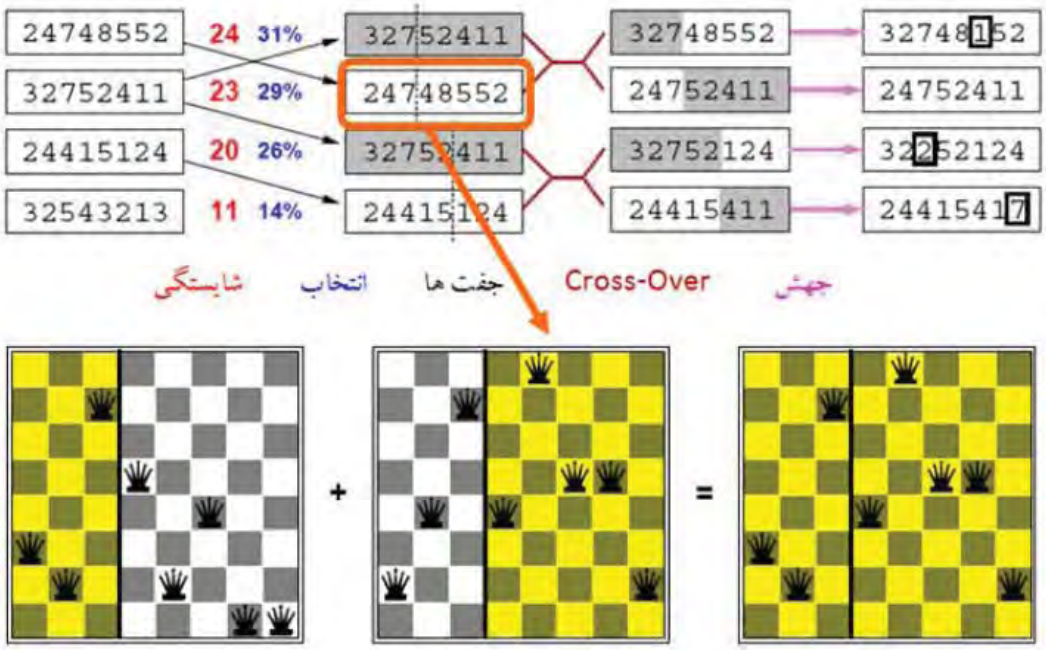
مثال : ۸ – وزیر



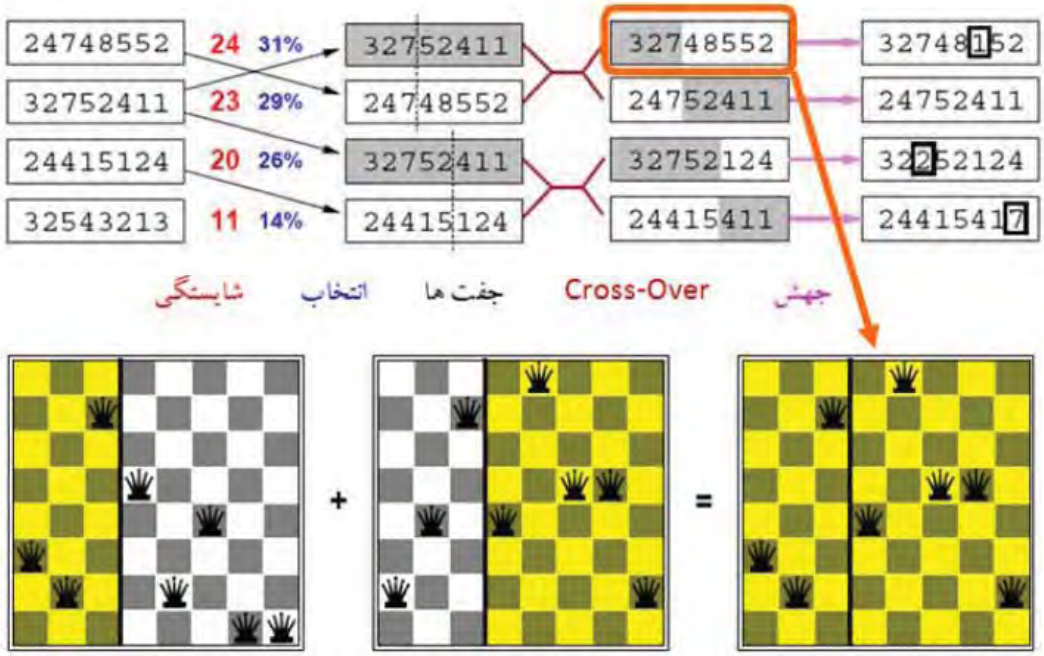
گام بعدی :



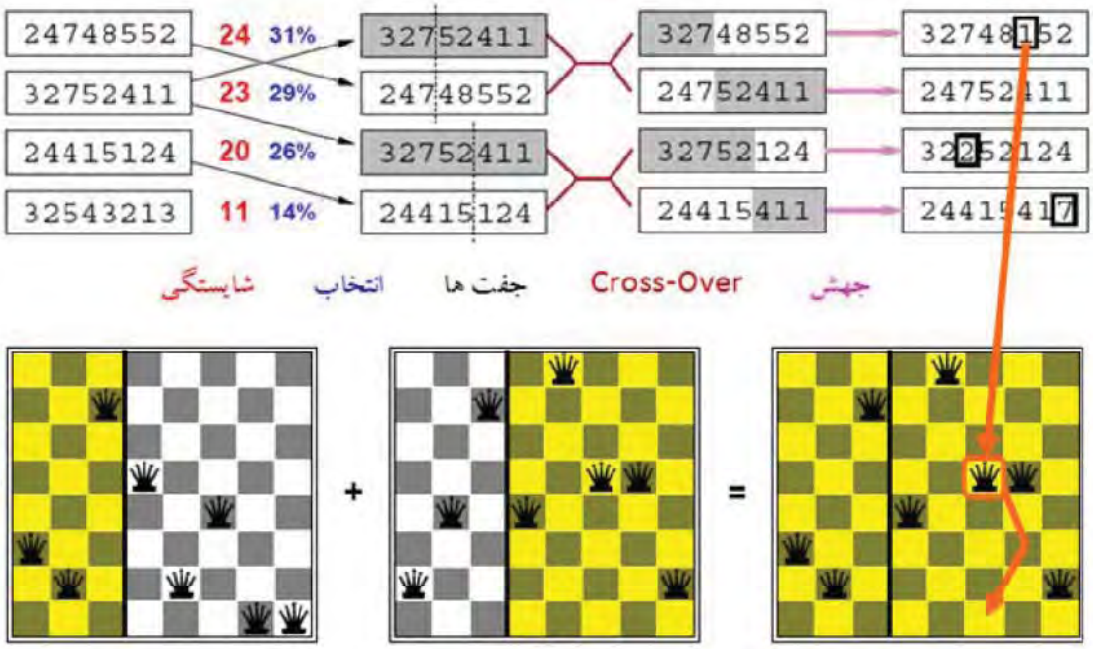
گام بعدی :



گام بعدی :



گام بعدی :



اگر الگوریتمهای ژنتیک مزیتی داشته باشند، مزیت اصلی آنها مربوط به عمل پیوند است.

جست و جوی محلی در فضاهای پیوسته

فضاهای حالت پیوسته – تصوّر کنید که می خواهیم جای سه فرودگاه را در کشور رومانی مشخص نماییم: فضای حالت شش بعدی توسط $(x_1, y_2), (x_2, y_2), (x_3, y_3)$ تعریف می شود . تابع هدف : $f(x_1, y_2, x_2, y_2, x_3, y_3)$ = مجموعه ای از فواصل مربّعی از هر شهر به نزدیک ترین فرودگاه می باشد . متدهای مجزا [1]، فاصله های پیوسته را به فاصله های مجزا تبدیل می نمایند . گرادیان تجربی با تغییرات $\pm \delta$ در هر وضعیّت متناسب است . گرادیان از فرمول زیر محاسبه می شود:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

جست و جوی محلی در فضاهای پیوسته

- گسسته در مقابل محیط های پیوسته
  - در فضاهای پیوسته، تابع جانشین در اغلب موارد، حالتهای نامتناهی را بر میگرداند
- حل مسئله:
  - گسسته کردن همسایه هر حالت
  - استفاده از شیب منظره

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f \quad where\ \nabla f = \left\{ \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots \right\}$$

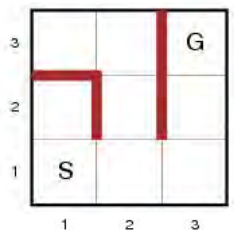
  - روش نیوتن رافسون

عاملهای جست و جوی (onlineبرخط) و محیطهای ناشناخته

- تا به حال همه الگوریتمها برون خطی بودند
  - برون خطی(Offline):راه حل قبل از اجرا مشخص است
  - درون خطی(onlineبرخط) :با یک در میان کردن محاسبات و فعالیت عمل میکند یعنی اول یک اقدام انجام می دهند سپس محیط را مشاهده می کنند و اقدام بعدی را محاسبه می کنند
- جستجوی درون خطی در محیطهای پویا و نیمه پویا مفید است
  - آنچه را که باید واقعا اتفاق بیفتد، در نظر گرفته نمیشود
- جست و جوی درون خطی ایده ضروری برای **مسئله اکتشاف** است
  - فعالیتها و حالتها برای عامل مشخص نیستند
  - مثال:قرار گرفتن روبات در محیطی جدید, نوزاد تازه بدنیا آمده

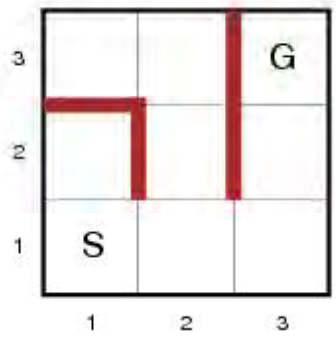
مسئله های جست و جوی (برخطonline)

- اطلاعات عامل
  - Actions(s): لیستی از فعالیتهای مجاز در حالت s
  - تابع هزینه مرحله ای c(s,a,s'): استفاده وقتی که بداند s' نتیجه است و اگر نداند s' نتیجه است قابل استفاده نیست
  - Goal-Test(s): آزمون هدف
- عامل حالت ملاقات شده قبلی را تشخیص میدهد
- فعالیتها قطعی اند
- دسترسی به تابع اکتشافی قابل قبول h(s) و فقط عدم اطلاع از موانع

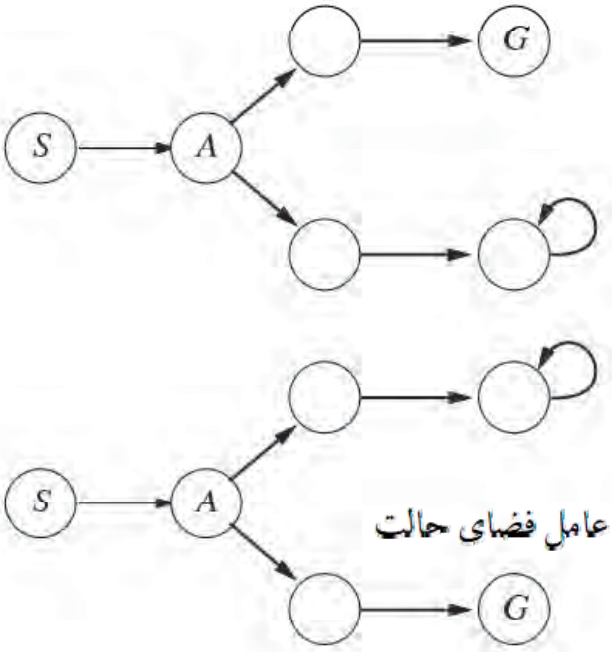


مسئله های جست و جوی (برخطonline)

- هدف: رسیدن به G با کمترین هزینه
- هزینه: مجموع هزینه های مراحل مسیری است که عامل طی میکند
- **نسبت رقابتی: مقایسه هزینه با هزینه مسیری که اگر عامل فضای حالت را از قبل میشناخت، طی میکرد**
  - در بعض موارد، بهترین نسبت رقابتی نامتناهی است
  - ممکن است جستجو به یک حالت بن بست برسد که نتوان از طریق آن به هدف رسید



مسئله های جست و جوی (برخطonline)



مثال دعوای متخاصمانه(دشمنانه)

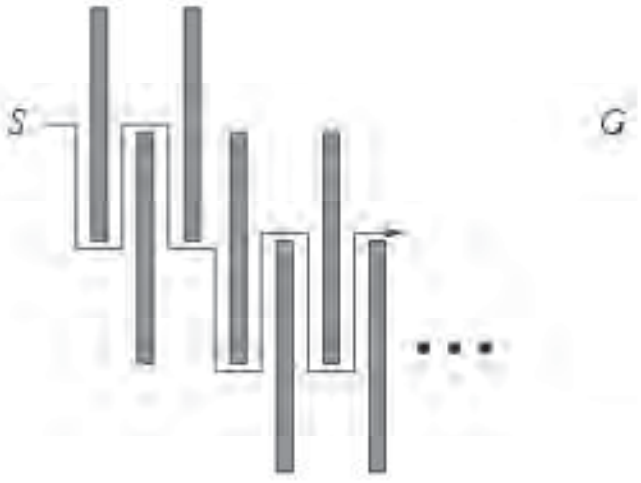
دو فضای حالت که عامل جست و جوی (برخطonline) را به بن بست میرسانند. هر عاملی حداقل در یکی از این دو فضا شکست می خورد

یک دشمن را تصور کنید که قادر است در حین کاوش عامل فضای حالت را بسازد

حالت های ملاقات شده S و A. حالت بعدی؟

- در یکی از فضاهای حالت شکست می خورد
- هیچ الگوریتمی نمی تواند از بن بست ها در تمامی فضاهای حالت اجتناب کند

مسئله های جست و جوی (برخطonline)



یک محیط دو بعدی که موجب میشود عامل جست و جوی(برخطonline) مسیر دلخواه ناکارآمدی و طولانی را برای رسیدن به هدف حل کند جرا که حریف مداوم دیوار می کشد.

عامل های جست و جوی (online برخط)

عامل یک نقشه از محیط نگهداری می کند

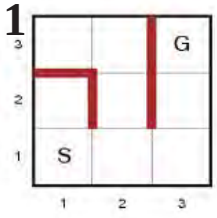
- نقشه بر اساس ورودی ادراکی بهنگام می شود
- از این نقشه برای انتخاب عمل بعدی استفاده می شود

به تفاوت مثلا با A* دقت کنید

- یک نسخه online تنها می تواند گرهی را گسترش دهد که به طور فیزیکی در آن قرار داشته باشد

مساله مسیر پر پیچ و خم بر روی یک صفحه ۳*۳

1



حالت اولیه: $s'=(1,1)$

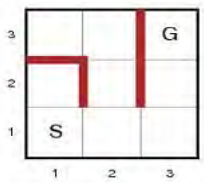
Result و Unexplored(UX)

Unbacktracked(UB) و...

تهی هستند

S و a نیز تهی هستند

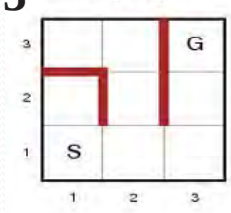
2



- GOAL-TEST((1, 1))?
  - S not = G thus false
- (1,1) a new state?
  - True
  - ACTION((1,1)) -> UX[(1,1)]
    - {RIGHT,UP}
- s is null?
  - True (initially)
- UX[(1,1)] empty?
  - False
- POP(UX[(1,1)]) -> a
  - a = UP
- s = (1,1)
- Return a

3

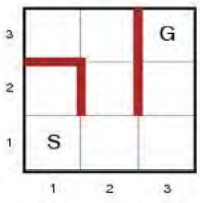
S'=(1,1)



- GOAL-TEST((1,1))?
  - S not = G thus false
- (1,1) a new state?
  - false
- s is null?
  - false (s = (2, 1))
  - result [DOWN, (2, 1)] <- (1,1
  - UB[(1,1)] = {(2,1)}
- UX[(1,1)] empty?
  - False
- a = RIGHT, s = (1,1)
- return a

4

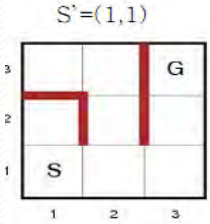
S'=(1,2)



- GOAL-TEST((1,2))?
  - S not = G thus false
- (1,2) a new state?
  - True,
    UX[(1,2)]={RIGHT,UP,LEFT}
- s is null?
  - false (s = (1,1))
  - result [RIGHT, (1,1)] <- (1, 2)
  - UB[(1,2)] = {(1,1)}
- UX[(1, 2)] empty?
  - False
- a = LEFT, s =(1, 2)
- return a

5

S'=(1,1)



- GOAL-TEST((1, 1))?
  - S not = G thus false
- (1, 1) a new state?
  - false
- s is null?
  - false (s = (1, 2))
  - result [LEFT, (1, 2)] <- (1, 1)
  - UB[(1, 1)]={(1, 2), (2, 1)}
- UX[(1, 1)] empty?
  - True
  - UB[(1, 1)] empty? False
- a = b for b in result [b, (1,1)] = (1, 2)
  - b = RIGHT
- a = RIGHT, s = (1, 1) ...

6

بدترین حالت: هر گره دو بار ملاقات شده است

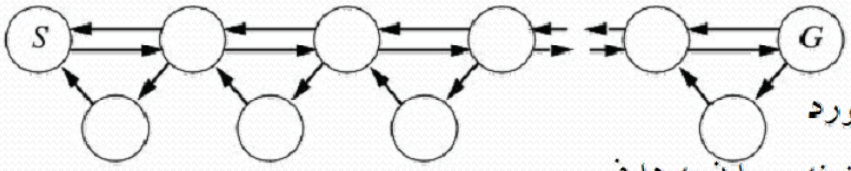
یک عامل ممکن است در حالی که به پاسخ نزدیک است، یک راه طولانی را بپیماید

یک روش عمیق کننده تکرای online می تواند این مشکل را حل کند

جستجوی عمقی online فقط وقتی کار می کند که اعمال قابل برگشت باشند

جست جوی محلی و آنلاین (برخط)

- تپه نوردی online می باشد
  - یک حالت ذخیره شده است
- عملکرد بد به دلیل وجود ماکزیمم های محلی
  - شروع مجدد تصادفی غیر ممکن است
- راه حل: حرکت تصادفی باعث کاوش می شود (می تواند حالات بسیاری را به صورت نمایی تولید کند)



راه حل ۲: اضافه نمودن حافظه به تپه نورد

ذخیره بهترین تخمین فعلی $H(s)$ از هزینه رسیدن به هدف

$H(s)$ در ابتدا تخمین هیوریستیک $h(s)$ می باشد

پس از آن بر اساس تجربه بهنگام می شود (شکل پایین)

به روز کردن هیورستیک ها = LRTA* تمایل به ادامه مسیر جاری طبق عنصر یادگیری