فصل چهارم

جستجوی ناآگاهانه

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 بهترین مسیر، h*(n) : ارزان ترین مسیر از گره n تا گره هدف
- تابع ارزیابی، f(n) : هزینه تخمینی ارزان ترین مسیر از طریق n
- **f(n): g(n) + h(n)**
- f*(n) : هزینه ارزان ترین مسیر از طریقn f*(n): g(n) + h*(n)

جستجوی حریصانه

حداقل هزینه تخمین زده شده برای رسیدن به هدف:

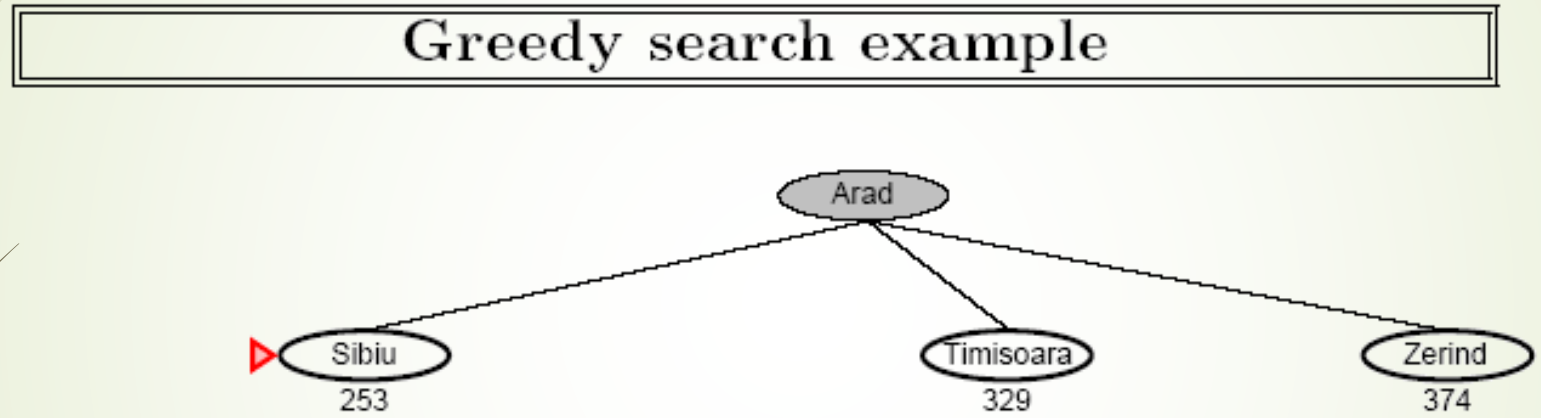
یکی از ساده‌ترین استراتژی‌های جستجوی بهترین، به حداقل رساندن هزینه تخمین زده شده برای رسیدن به هدف است. بدین صورت که حالت گره‌ای که به حالت هدف نزدیک تر است، ابتدا بسط داده می‌شود.

جستجوی حریصانه: جستجوی بهترین که h را به منظور انتخاب گره بعدی برای بسط استفاده می‌کند، جستجوی حریصانه (greedy search) نامیده می‌شود.

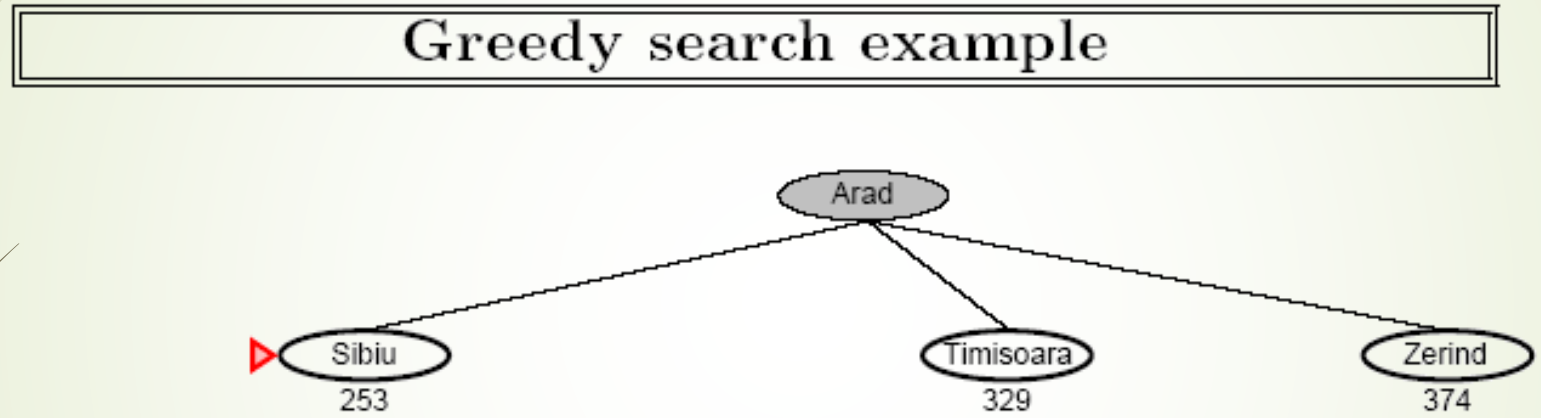
جستجوی حریصانه

Greedy search example

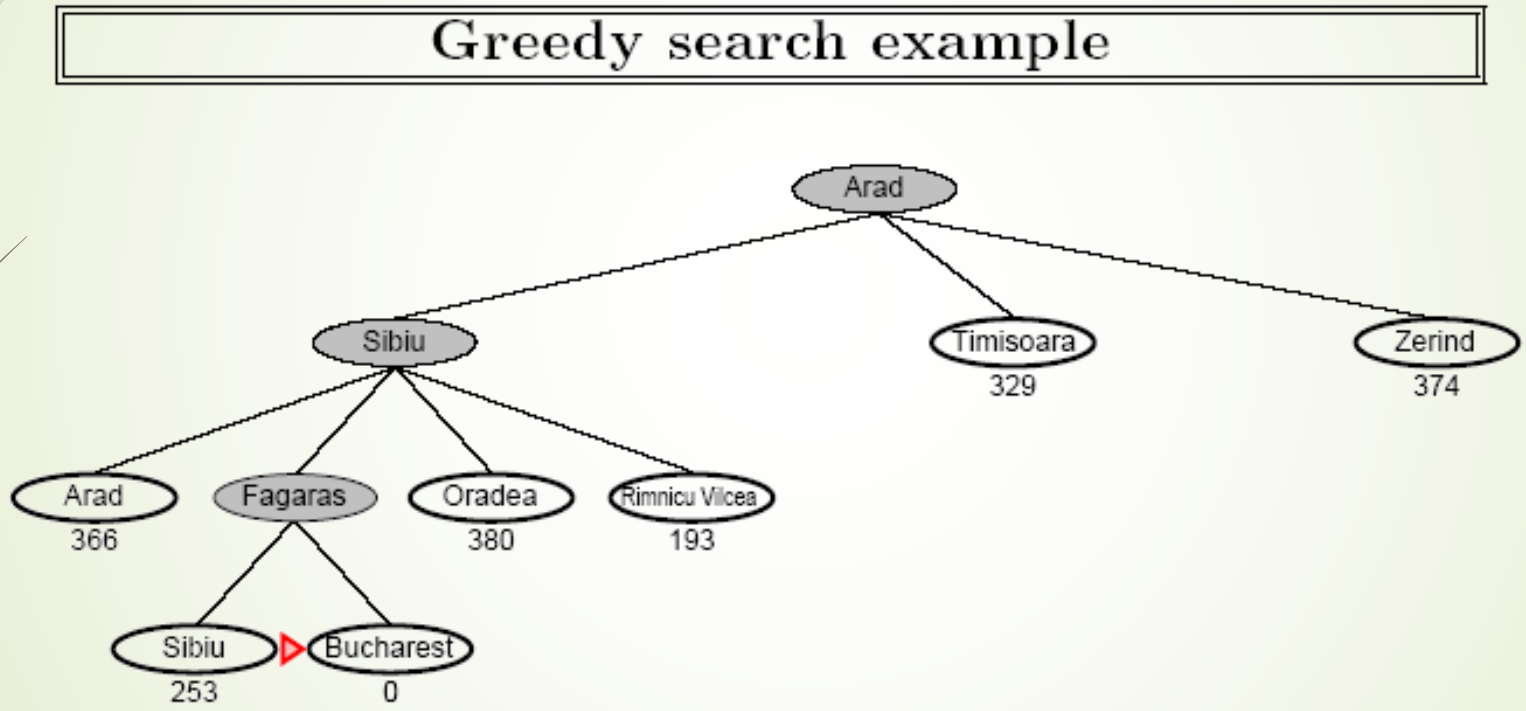
جستجوی حریصانه



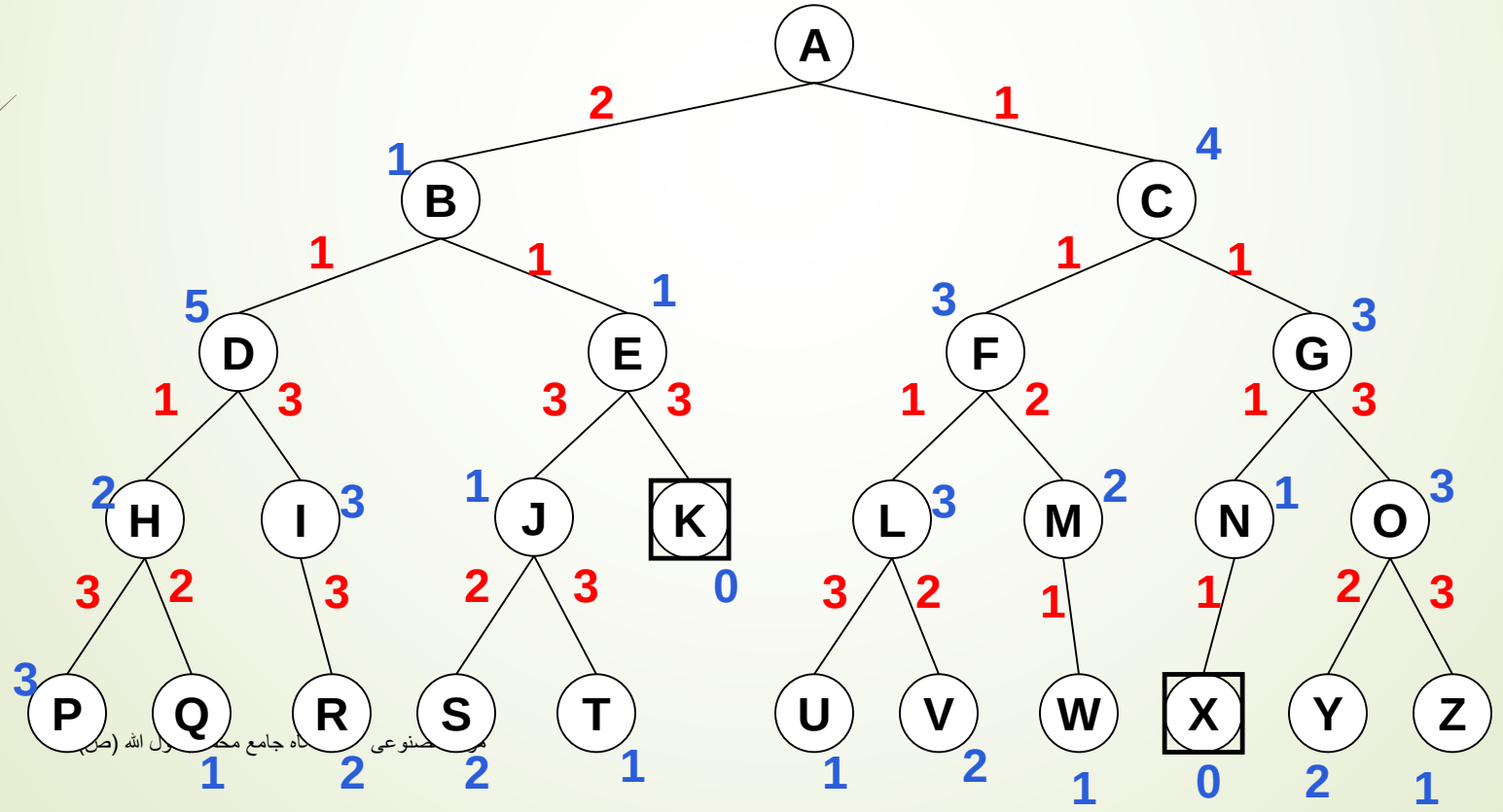
جستجوی حریصانه



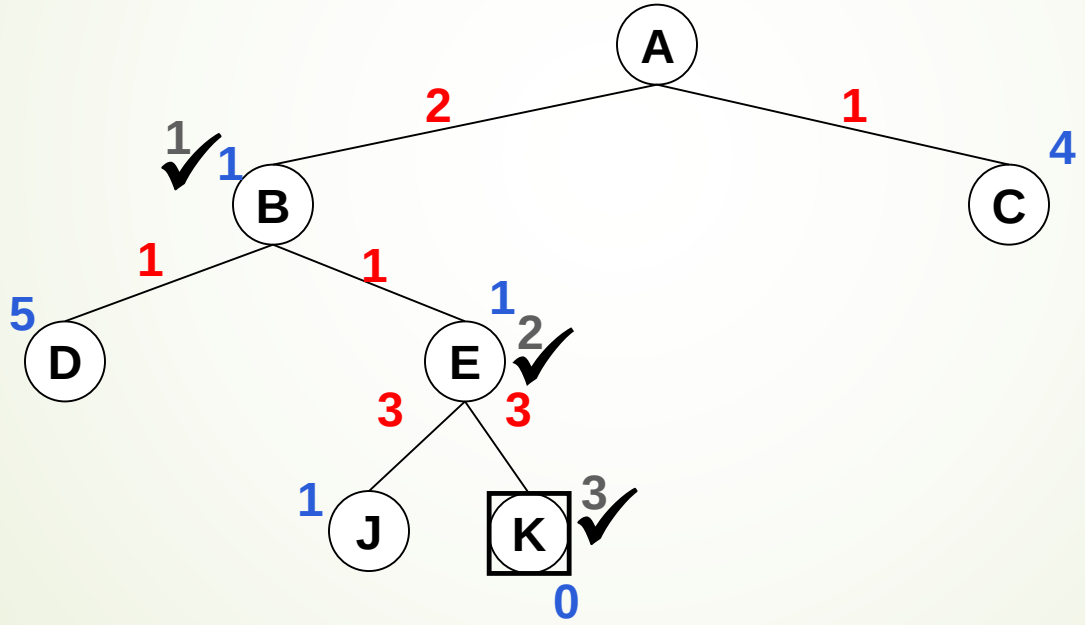
جستجوی حریصانه



جستجوی حریصانه



جستجوی حریصانه



جستجوی حریصانه

- کامل بودن: خیر
  - اما اگر h = h* آنگاه جستجو کامل میشود
- بهینگی: خیر
  - اما اگر h = h* آنگاه جستجو کامل میشود
- پیچیدگی زمانی: $O(b^m)$
  - اما اگر h = h* آنگاه $O(bd)$
- پیچیدگی فضا: $O(b^m)$
  - اما اگر h = h* آنگاه $O(bd)$

حداقل‌سازی مجموع هزینه مسیر: جستجوی A*

جستجو با هزینه یکسان، هزینه مسیر، g(n) را نیز حداقل می‌کند.

با ترکیب دو تابع ارزیابی داریم:

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

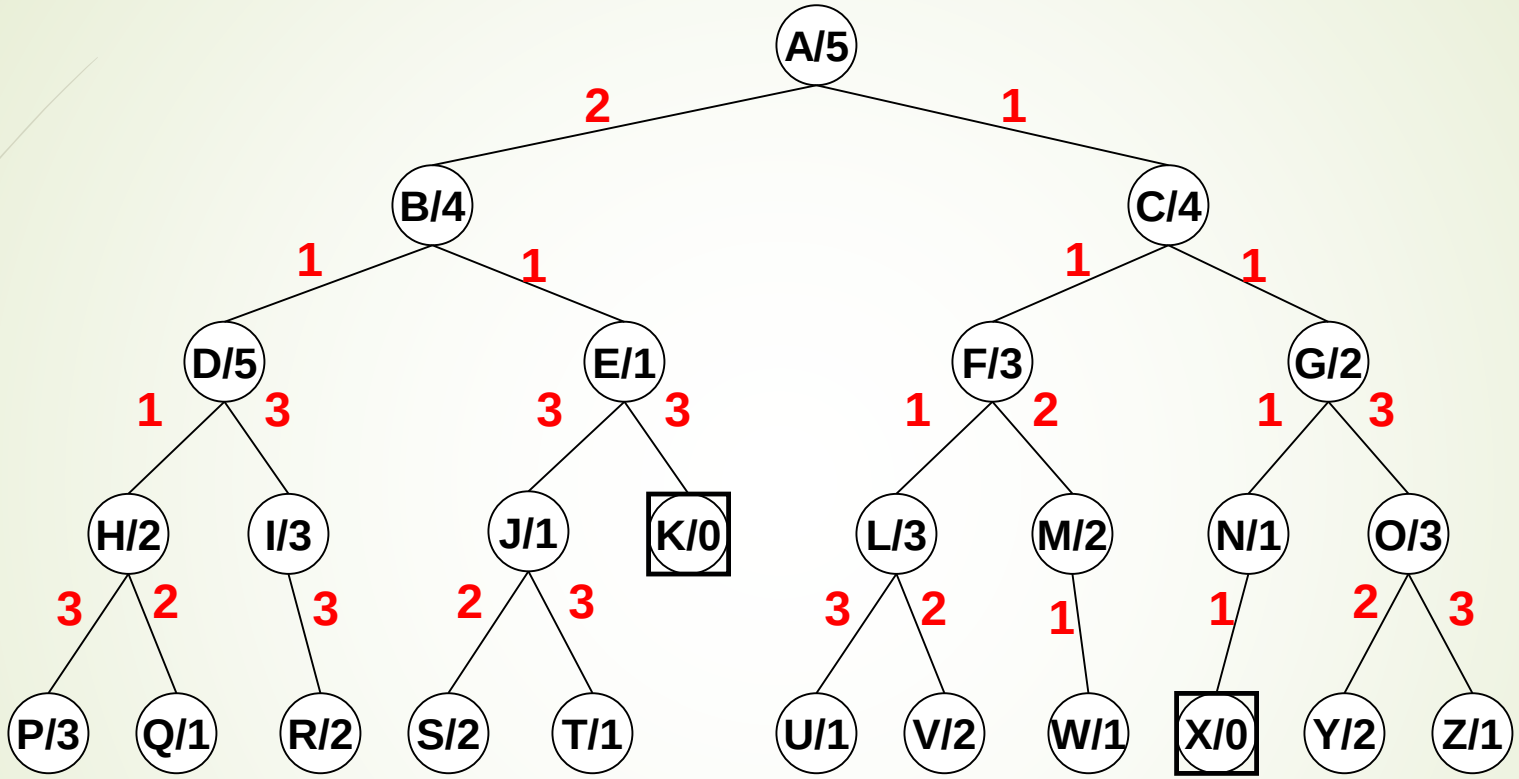
g(n): هزینه مسیر از گره آغازین به گره n را به ما می‌دهد.

h(n): هزینه تخمین زده شده از ارزانترین مسیر از n به هدف است

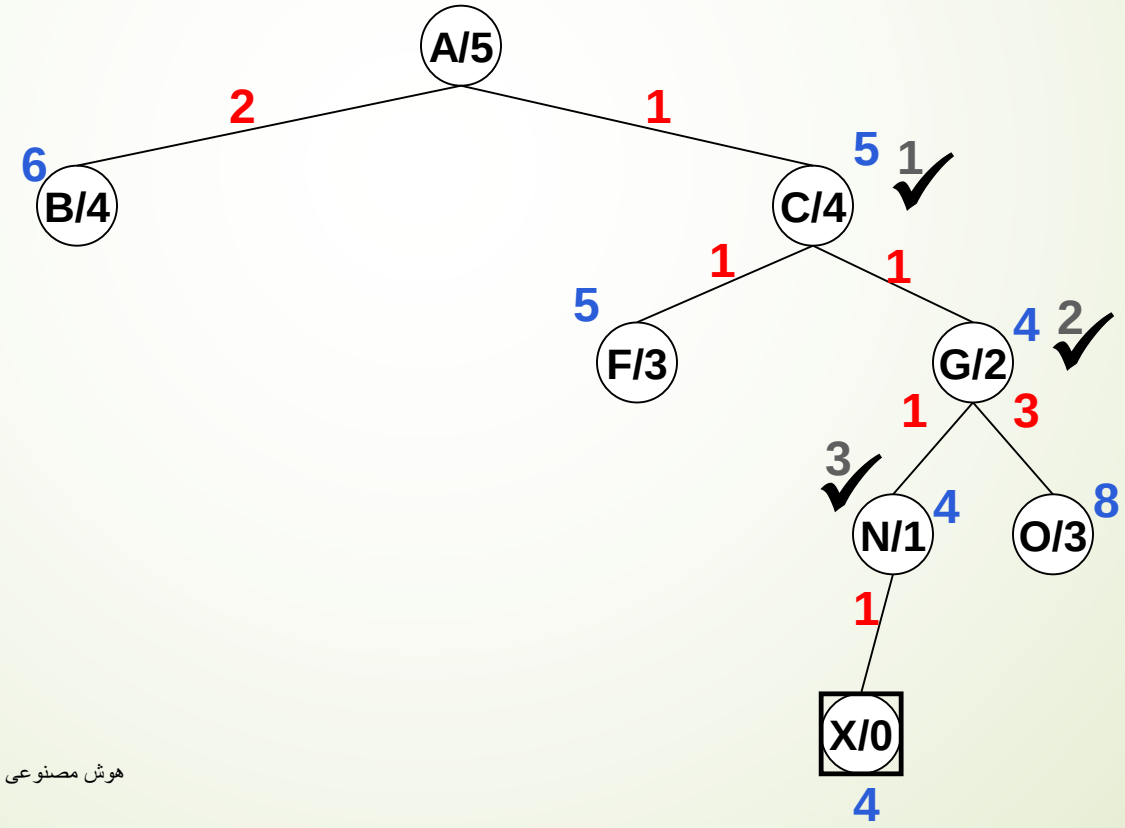
و ما داریم:

هزینه تخمین زده شده ارزانترین راه حل از طریق n = f(n)

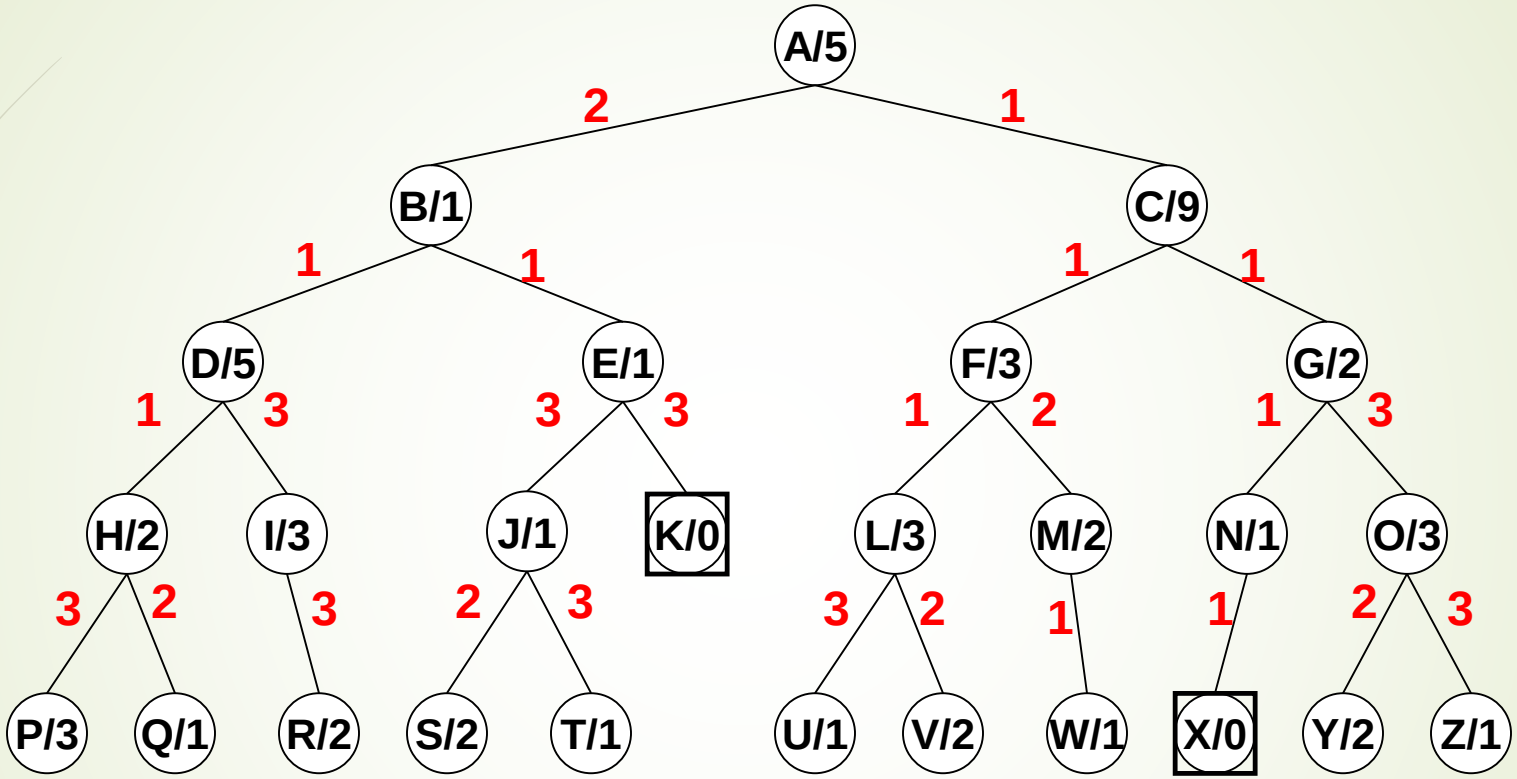
جستجوی A*


A/5
2
1
B/4
C/4
1
1
1
1
D/5
E/1
F/3
G/2
1
3
3
3
1
2
1
3
H/2
I/3
J/1
K/0
L/3
M/2
N/1
O/3
3
2
3
2
3
3
2
1
1
2
3
P/3
Q/1
R/2
S/2
T/1
U/1
V/2
W/1
X/0
Y/2
Z/1

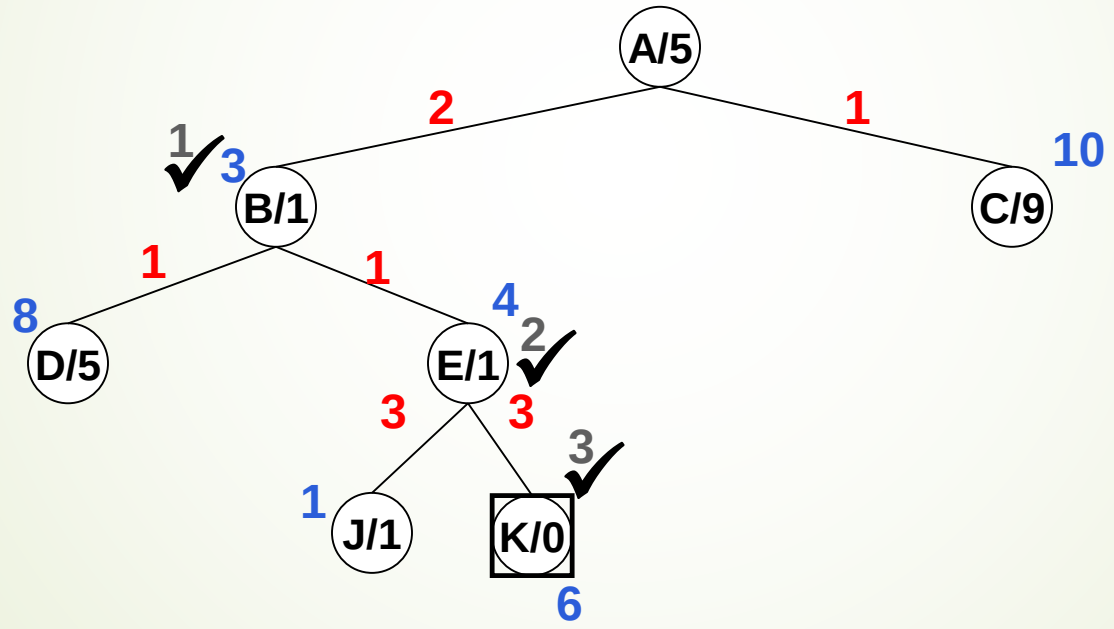
جستجوی A*



جستجوی A*


A/5
2
1
B/1
C/9
1
1
1
1
D/5
E/1
F/3
G/2
1
3
3
3
1
2
1
3
H/2
I/3
J/1
K/0
L/3
M/2
N/1
O/3
3
2
3
2
3
2
1
1
2
3
P/3
Q/1
R/2
S/2
T/1
U/1
V/2
W/1
X/0
Y/2
Z/1

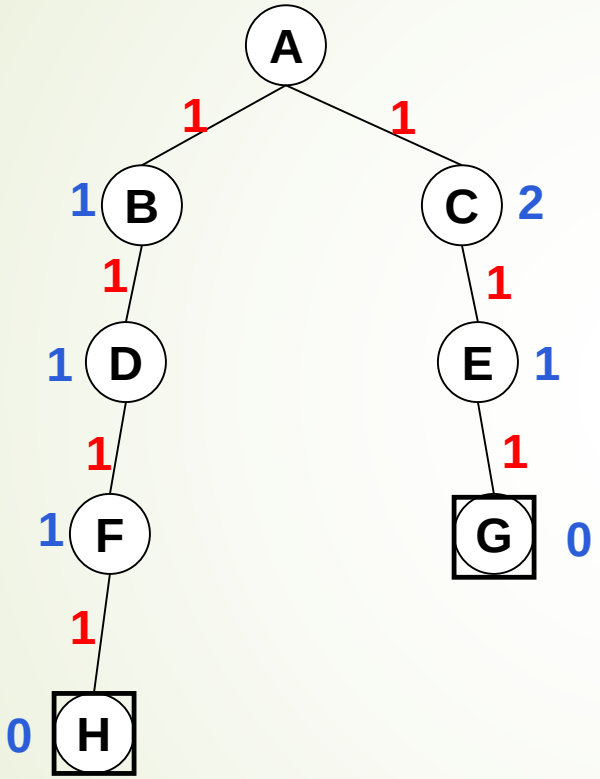
جستجوی A*



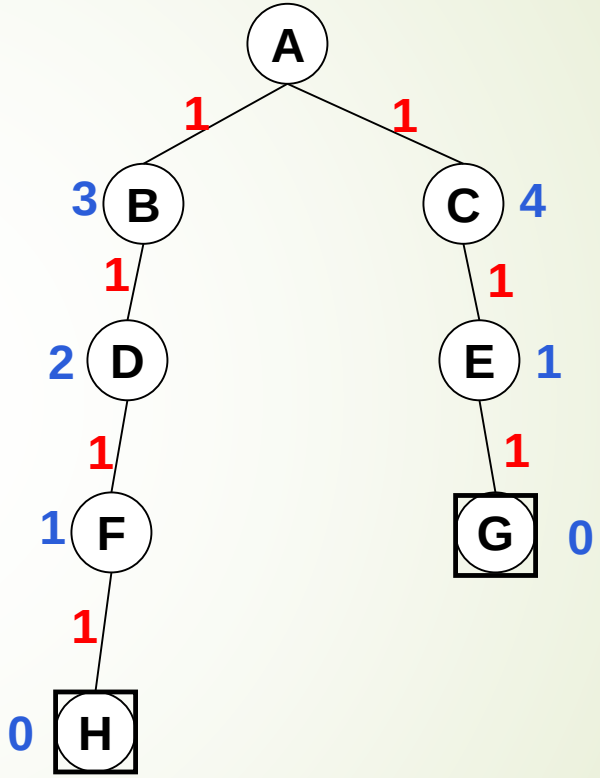
جستجوی A*

- کامل بودن: بله
- بهینگی: بله
- پیچیدگی زمانی: $O(b^m)$
  - اما اگر h = h* آنگاه $O(bd)$
- پیچیدگی فضا: $O(b^m)$
  - اما اگر h = h* آنگاه $O(bd)$

جستجوی A*



$h \leq h^*$



$h \not\leq h^*$

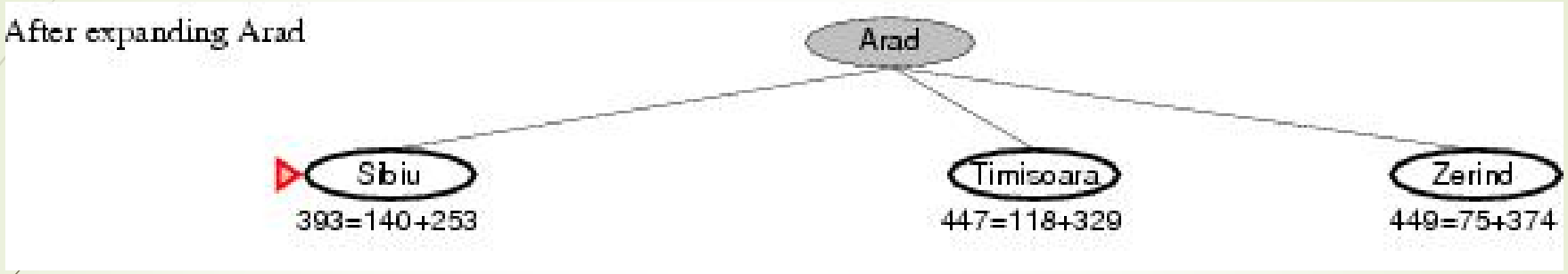
جستجوی A* در نقشه رومانی

(a) The initial state

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:

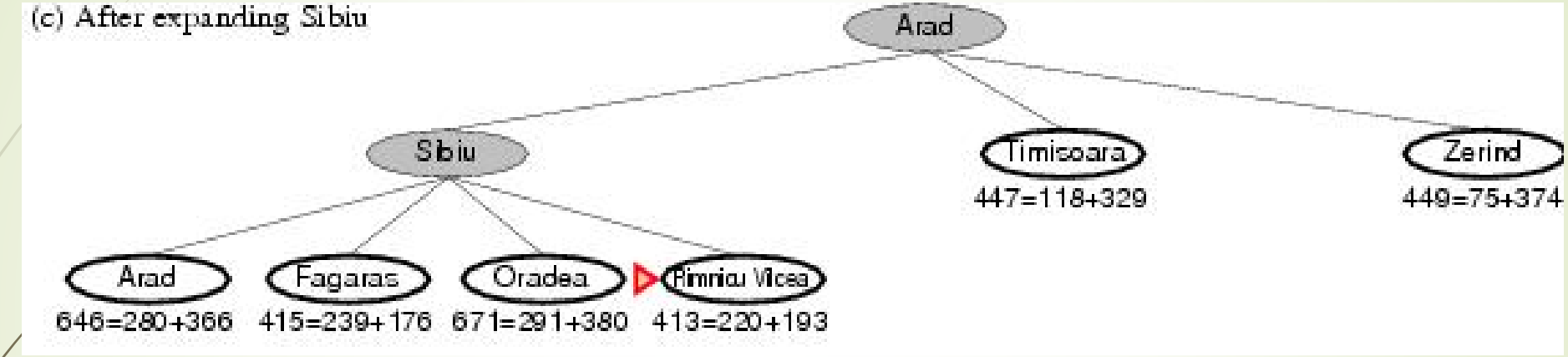
f(Sibiu)=c(Arad,Sibiu)+h(Sibiu)=140+253=393

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

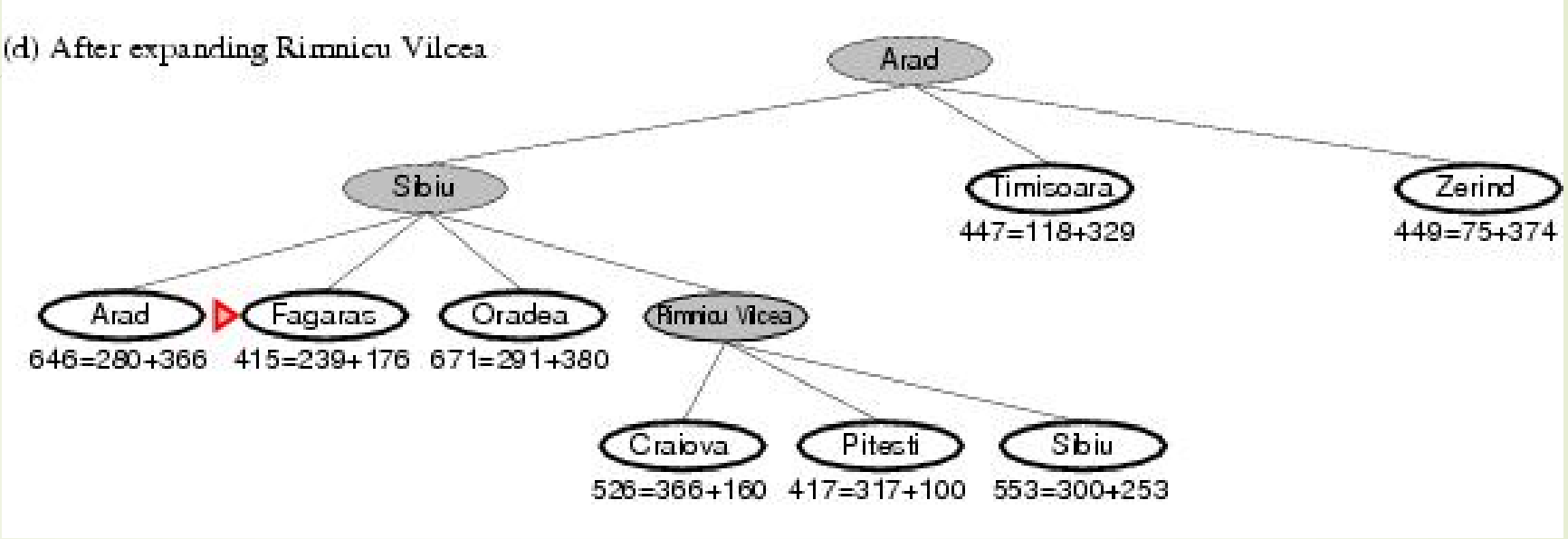
f(Fagaras)=c(Sibiu,Fagaras)+h(Fagaras)=239+179=415

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:

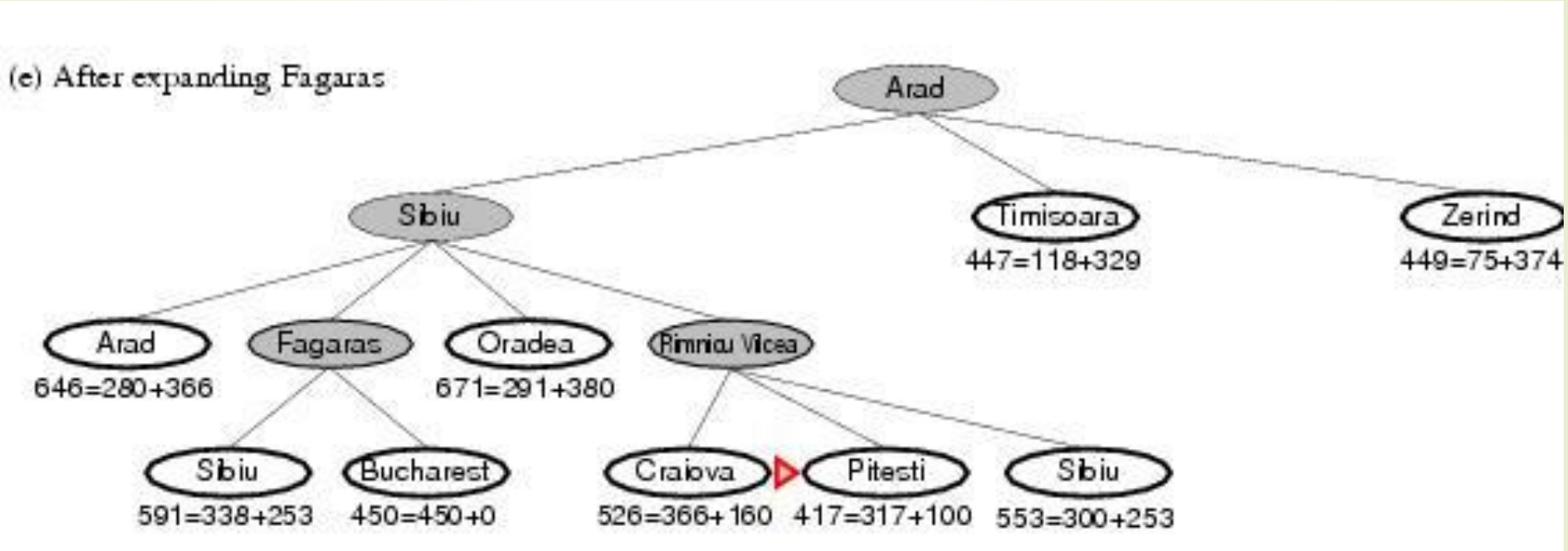
f(Craiova)=c(Rimnicu Vilcea, Craiova)+h(Craiova)=360+160=526

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



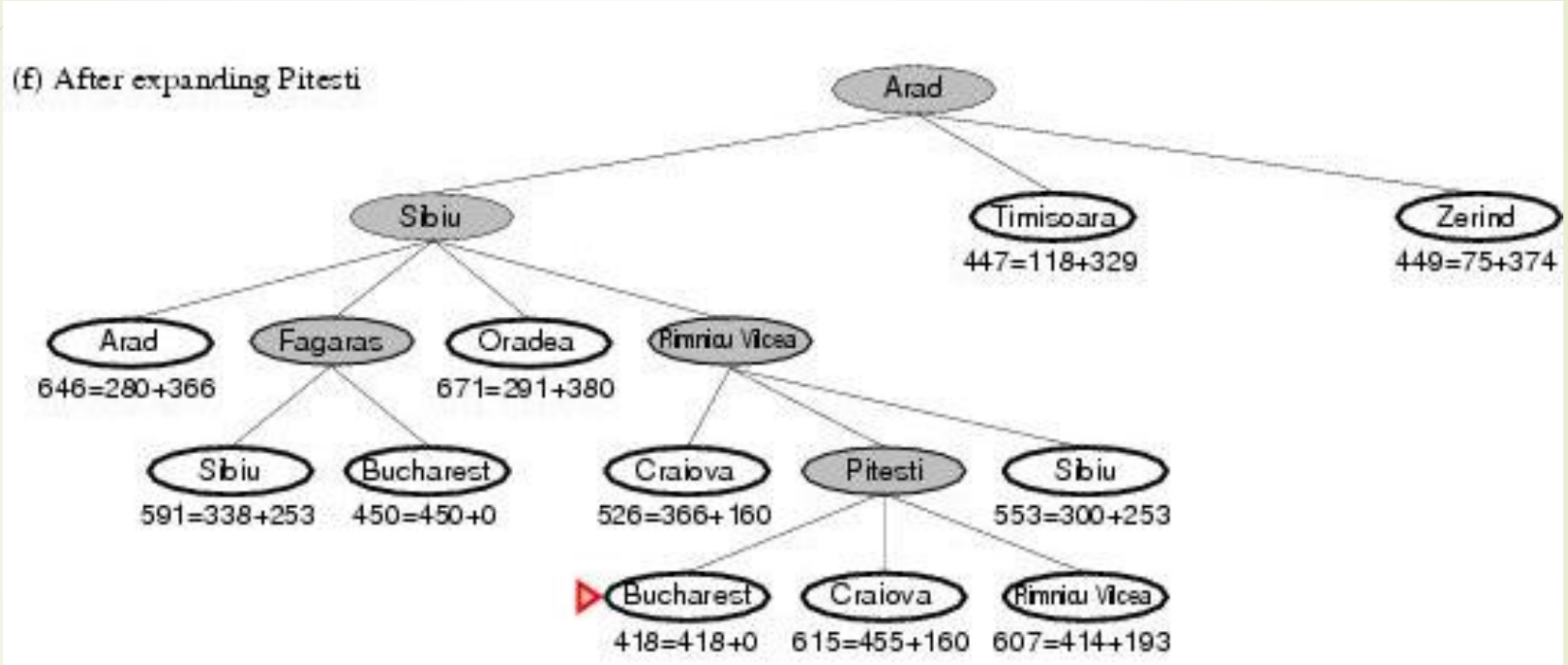
Fagaras را باز کرده و f(n) را برای هر یک از زیربرگها محاسبه میکنیم:

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



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

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.

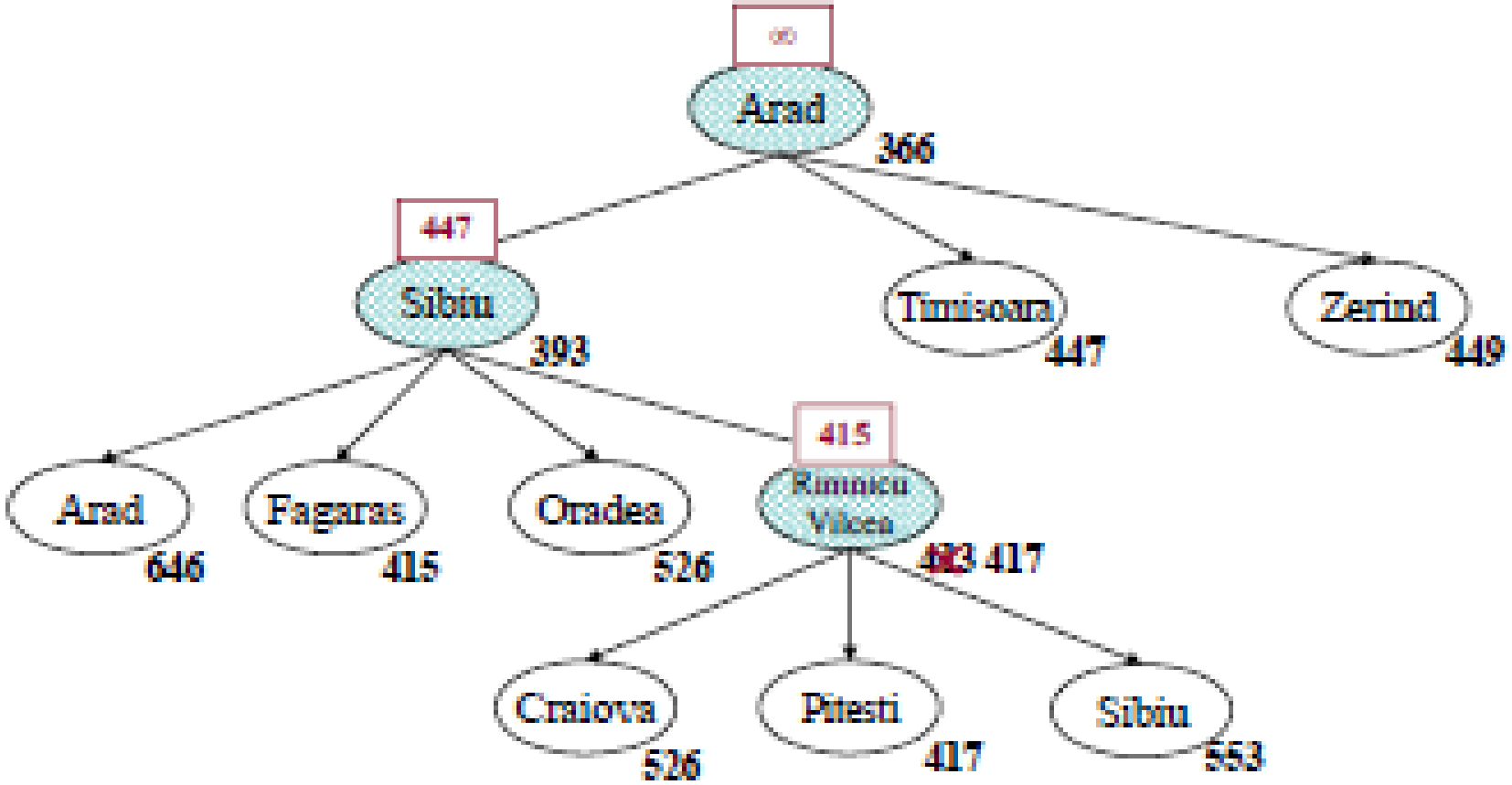
بهترین جستجوی بازگشتی RBFS

- 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.
- 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.

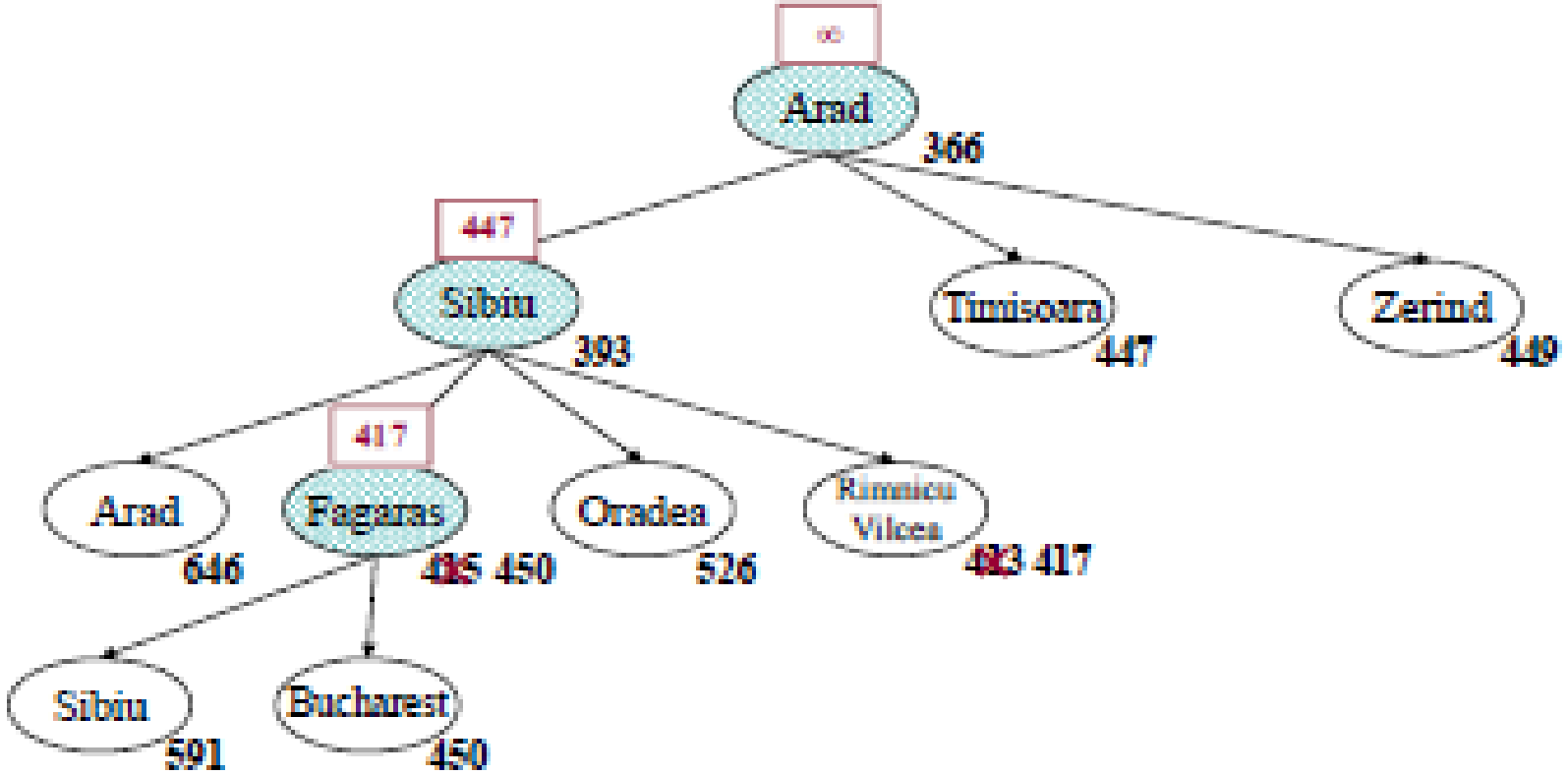
بهترین جستجوی بازگشتی RBFS

- f-valueمقدار مربوط به بهترین مسیر جایگزین موجود را نگهداری می کند.
- اگر f-value فعلی از f-value مسیر جایگزین تجاوز کند، آنگاه به این مسیر جایگزین بازگشت می کند.
- در بازگشت به عقب مقدار f-value مربوط به هر گره موجود در مسیر را با بهترین مقدار f-value فرزندانش جایگزین می کند.
- بنابراین گسترش مجدد نتیجه فعلی هنوز ممکن می باشد.

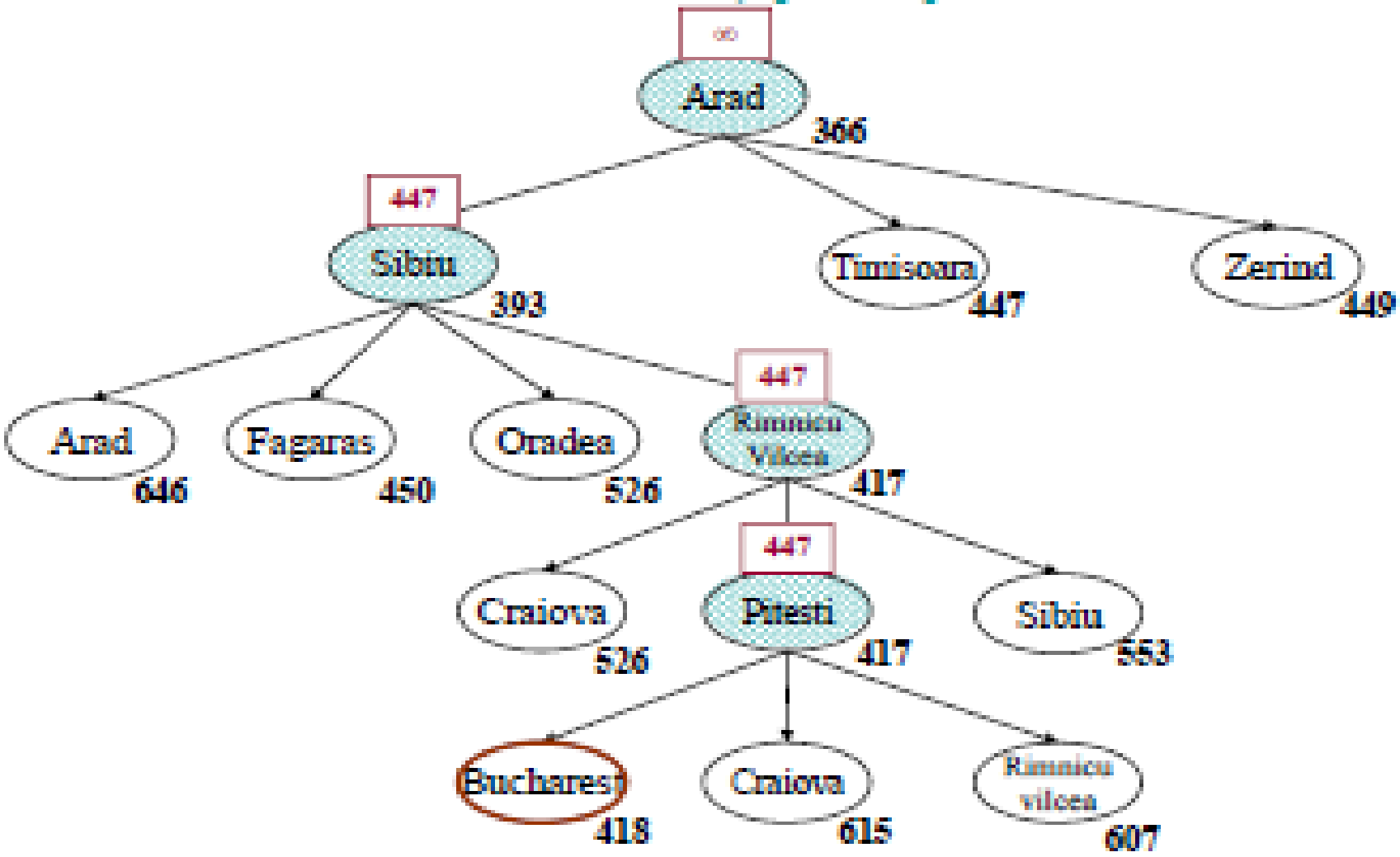
مثال جستجوی RBFS



مثال جستجوی RBFS



مثال جستجوی RBFS



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

جستجوی SMA*

الگوریتم SMA*، حافظه محدود A* ساده شده (Simplified-Memory-BoundedA*) می‌باشد.

این الگوریتم، قادر است تا از تمام حافظه موجود برای اجرای جستجو استفاده کند. استفاده از حافظه بیشتر کارایی جستجو را وسعت می‌بخشد.

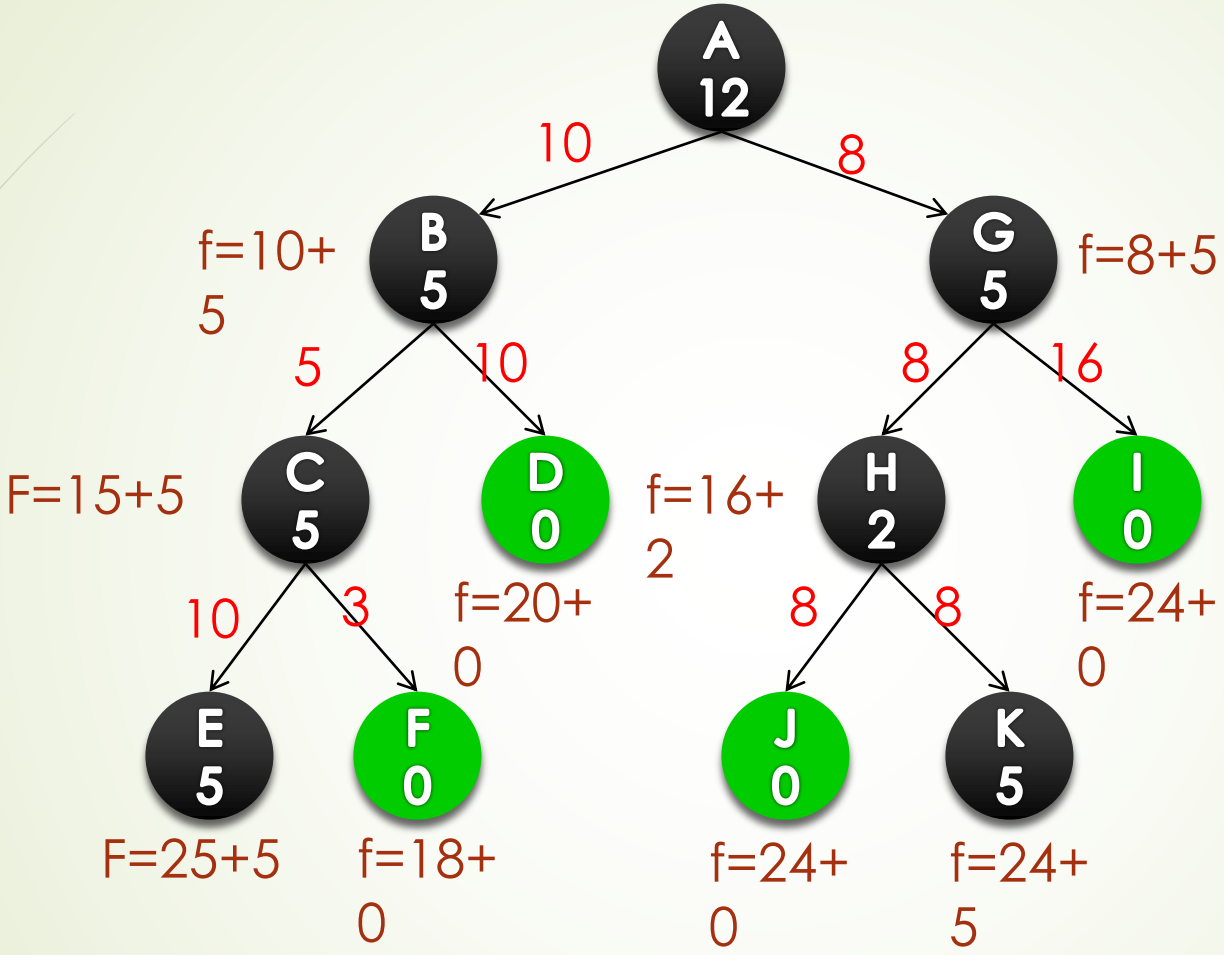
جستجوی SMA*

طراحی SMA* ساده است:

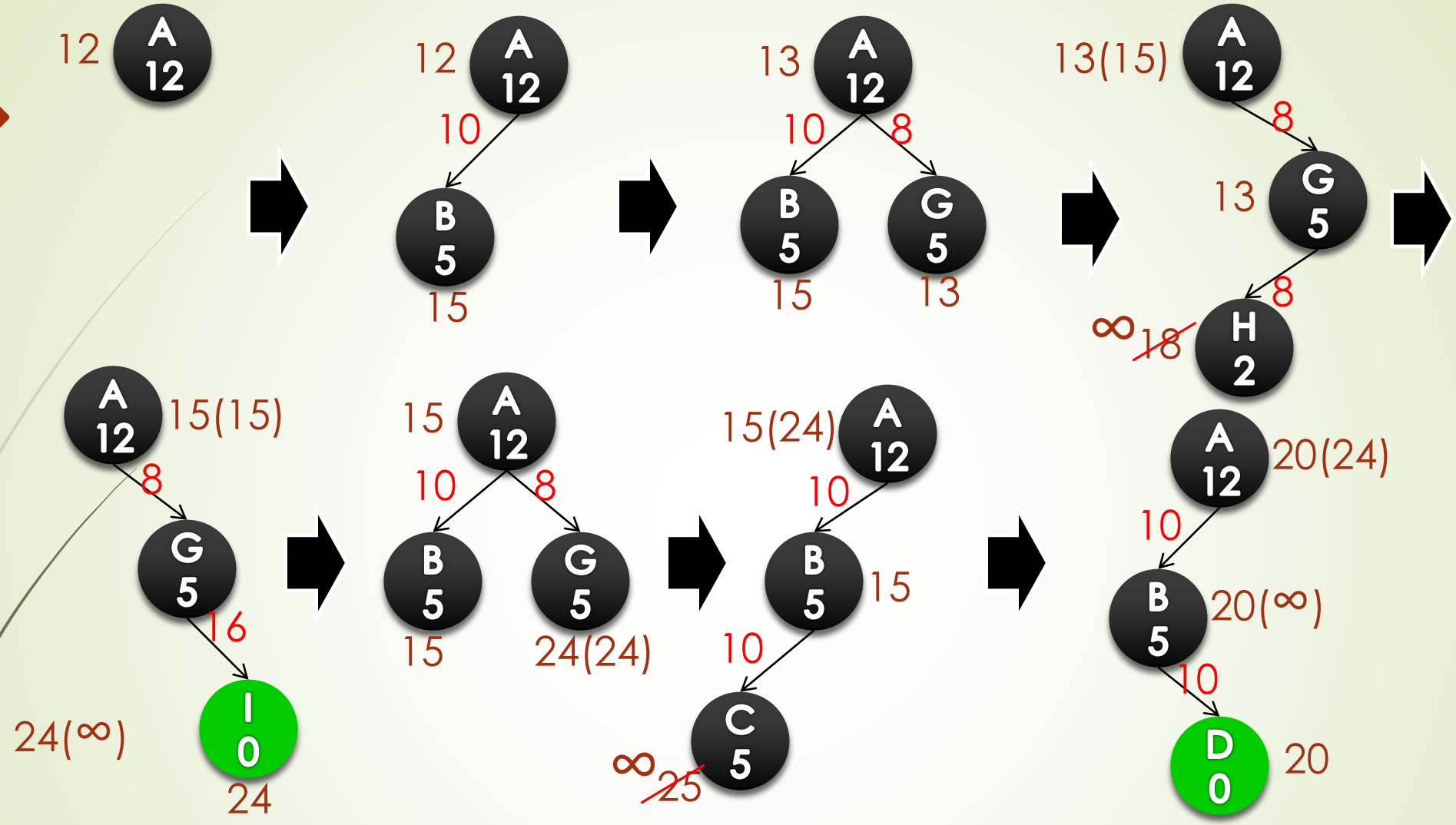
زمانی که نیاز به تولید فرزند داشته باشد ولی حافظه‌ای نداشته باشد، نیاز به ساختن فضا بر روی صف دارد. برای انجام این امر، یک گره را حذف می‌کند. گره‌هایی که به این طریق از صف حذف می‌شوند، گره‌های فراموش‌شده یا (forgotten nodes) نامیده می‌شوند.

برای اجتناب از جستجوی مجدد زیردرخت‌هایی که از حافظه حذف شده‌اند، در گره‌های اجدادی، اطلاعاتی در مورد کیفیت بهترین مسیر در زیر درخت فراموش شده، نگهداری می‌شود.

جستجوی SMA*



- پیشرفت جستجوی SMA* با حافظه ای به اندازه ۳ گره در فضای حالت بالا ...



- مقادير داخل پرانتز مقدار بهترين نسل هاي مابعد فراموش شده را نشان مي دهد
- اكنون حافظه پر است G براي بسط و برگي كه كم عمق (قديم) ترين و بيشترين هزينه دارد حذف مي كنيم.

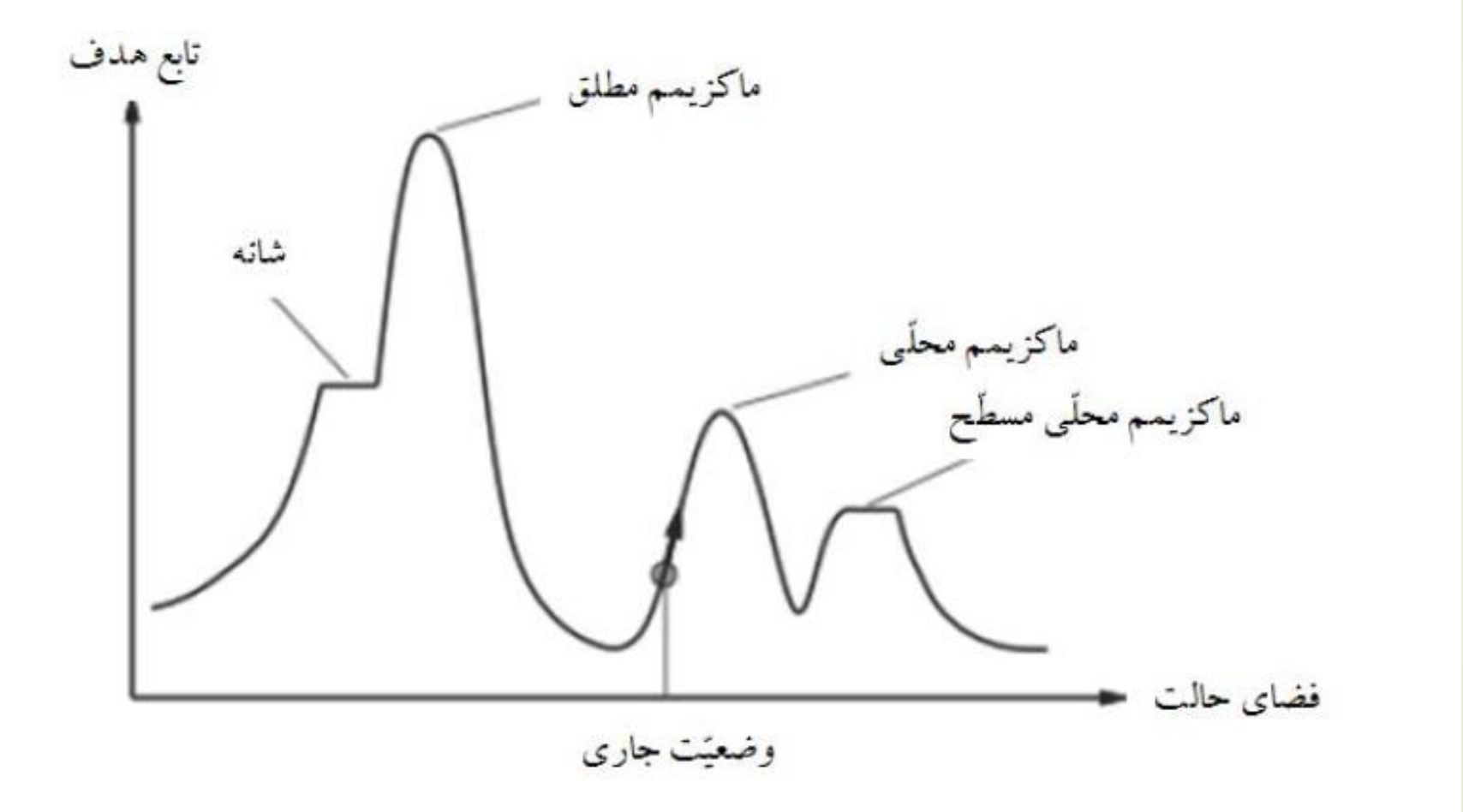
جستجوی SMA*

- این الگوریتم کامل است اگر حافظه برای ذخیره کم عمق ترین مسیر راه حل کافی باشد
- این الگوریتم بهینه است و بهترین راه حل را برمی گرداند که بتواند با حافظه موجود مطابقت داشته باشد
- زمانی که حافظه موجود برای درخت جستجو کامل کافی باشد جستجو کارآ بهینه Optimally efficient است

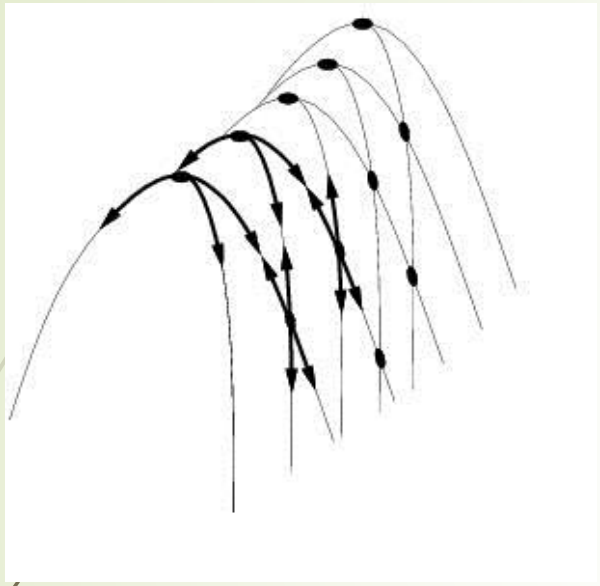
الگوریتم های جست و جوی محلی و بهینه سازی

- الگوریتم های قبلی، فضای جست و جو را به طور سیستماتیک بررسی می کنند (هدف را داریم مهم مسیر رسیدن به هدف است)
  - 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 خود هدف مهم است.
  - 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 جست و جوی محلی و بهینه سازی



جست و جوی تپه نوردی
hill climbing search



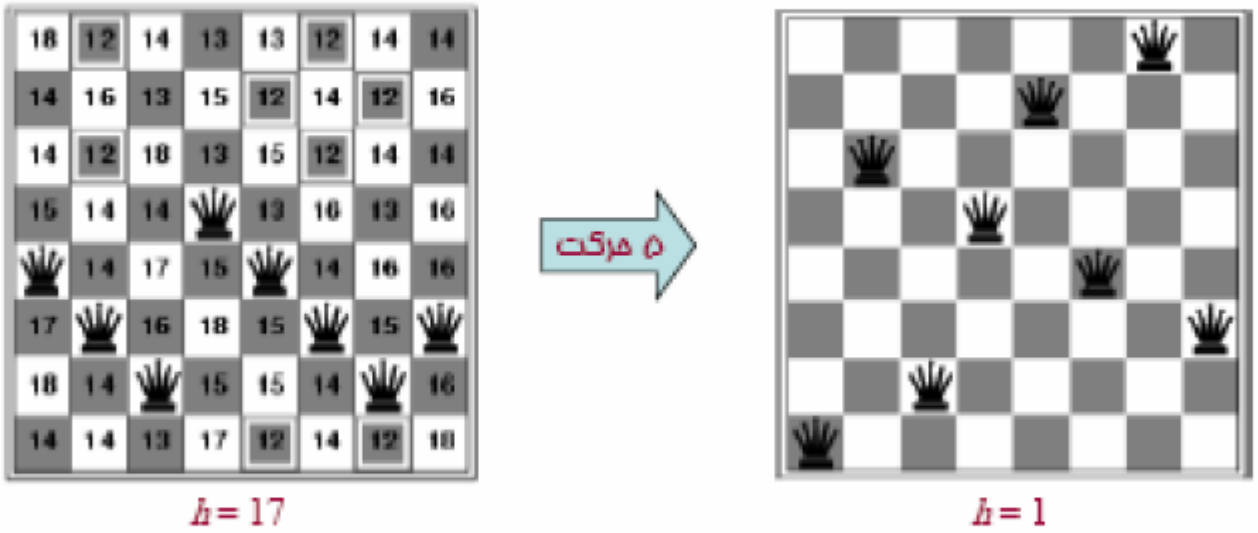
- حلقه ای که در جهت افزایش مقدار حرکت میکند(بطرف بالای تپه)
  - رسیدن به بلندترین قله در همسایگی حالت فعلی، شرط خاتمه است.
- ساختمان داده گره فعلی، فقط حالت و مقدار تابع هدف را نگه میدارد
- جست و جوی محلی حریصانه نیز نام دارد
  - بدون فکر قبلی حالت همسایه خوبی را انتخاب میکند
- تپه نوردی به دلایل زیر میتواند متوقف شود:
  - بیشینه محلی **(در مسئله ۸ وزیر ۸ وزیر بدون برخورد ماکزیمم مطلق است ولی ۷ وزیر بدون برخورد و یکی با برخورد ماکزیمم محلی است.)**
  - برآمدگی ها **(در مسئله ۸ وزیر چند حالت با۷ وزیر بدون برخورد و یکی با برخورد)**
  - فلات شامل شانه ها و ماکزیمم محلی هموار است که ممکن است نتواند راه خود را پیدا کند.

جست و جوی تپه نوردی

- انواع تپه نوردی:
  - تپه نوردی غیرقطعی: از جاهایی می رود که مقدارش کمتر است.
  - تپه نوردی اولین انتخاب: به اولین و بهترین قله که رسید جواب را بر می گرداند.
  - تپه نوردی شروع مجدد تصادفی: قله ها را با هم مقایسه می کند و بیشترین را بر می گرداند.

- مثال: مسئله ۸ وزیر
- مسئله ۸ وزیر با استفاده از فرمولبندی حالت کامل
  - در هر حالت ۸ وزیر در صفحه قرار دارند
- تابع جانشین: انتقال یک وزیر به مربع دیگر در همان ستون
- تابع اکتشاف: جفت وزیرهایی که نسبت به هم گارد دارند

جستوی تپه نوردی : مثال ۸ وزیر



h = تعداد جفت وزیرهایی که بطور مستقیم و یا بطور غیر مستقیم یکدیگر را تهدید می کنند.

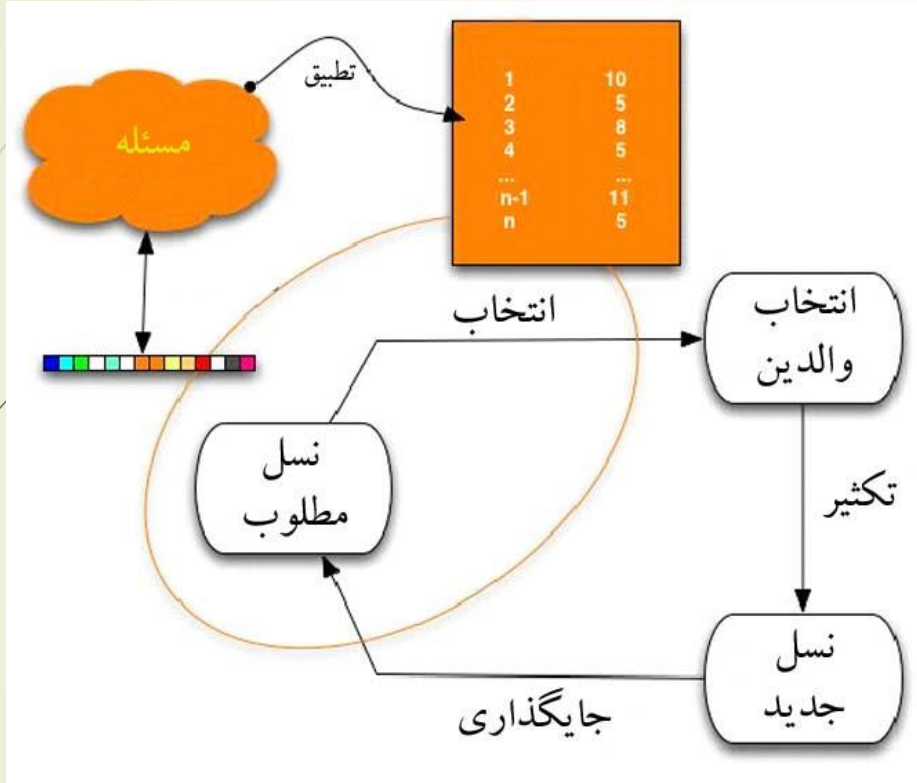
جست و جوی شبیه سازی حرارت
Simulated annealing

- تپه نوردی مرکب با حرکت تصادفی
- شبیه سازی حرارت: حرارت با درجه بالا و به تدریج سرد کردن
- مقایسه با حرکت توپ
  - توپ در فرود از تپه به عمیق ترین شکاف میرود
  - با تکان دادن سطح توپ از بیشینه محلی خارج میشود
  - با تکان شدید شروع(دمای زیاد)
  - بتدریج تکان کاهش(به دمای پایین تر)
- با کاهش زمانبندی دما به تدریج، الگوریتم یک بهینه عمومی را می یابد
- گیر افتادن در ماکزیمم محلی : حرکت به چند گام قبل برای فرار از ماکزیمم محلی

جست و جوی پرتو محلی

- به جای یک حالت، k حالت را نگهداری میکند
  - حالت اولیه: k حالت تصادفی
  - گام بعد: جانشین همه k حالت تولید میشود
  - اگر یکی از جانشین ها هدف بود، تمام میشود
  - وگر نه بهترین جانشین را انتخاب کرده، تکرار میکند
- 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 غیرقطعی
  - به جای انتخاب بهترین k از جانشینها، k جانشین تصادفی را بطوریکه احتمال انتخاب یکی تابعی صعودی از مقدارش باشد، انتخاب میکند

الگوریتم های ژنتیک



شکلی از جست و جوی پرتو غیر قطعی کـه حالتهای جانشین از طریق ترکیب دو حالـت والد تولید میشود

از تکامل ژنتیکی به عنوان یک الگوی حل مسئله استفاده می‌کند.

الگوریتم های ژنتیک

- 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)

ایده کلی

- یک GA برای حل یک مسئله مجموعه بسیار بزرگی از راه حلهای ممکن را تولید میکند.
- هر یک از این راه حلها با استفاده از یک " تابع تناسب "مورد ارزیابی قرار میگیرد.
- آنگاه تعدادی از بهترین راه حلها باعث تولید راه حلهای جدیدی میشوند .که اینکار باعث تکامل راه حلها میگردد.
- بدین ترتیب فضای جستجو در جهتی تکامل پیدا میکند که به راه حل مطلوب برسد
- در صورت انتخاب صحیح پارامترها، این روش میتواند بسیار موثر عمل نماید.

فضای فرضیه

- الگوریتم ژنتیک فرضیه ها ی جدید را با تغییر و ترکیب متوالی اجزا بهترین فرضیه های موجود بدست میاورد.
- در هرمرحله مجموعه ای از فرضیه ها که جمعیت (population) نامیده میشوند از طریق جایگزینی بخشی از جمعیت فعلی با فرزندانی که از بهترین فرضیه های موجود حاصل شده اند بدست میآید.

ویژگیها

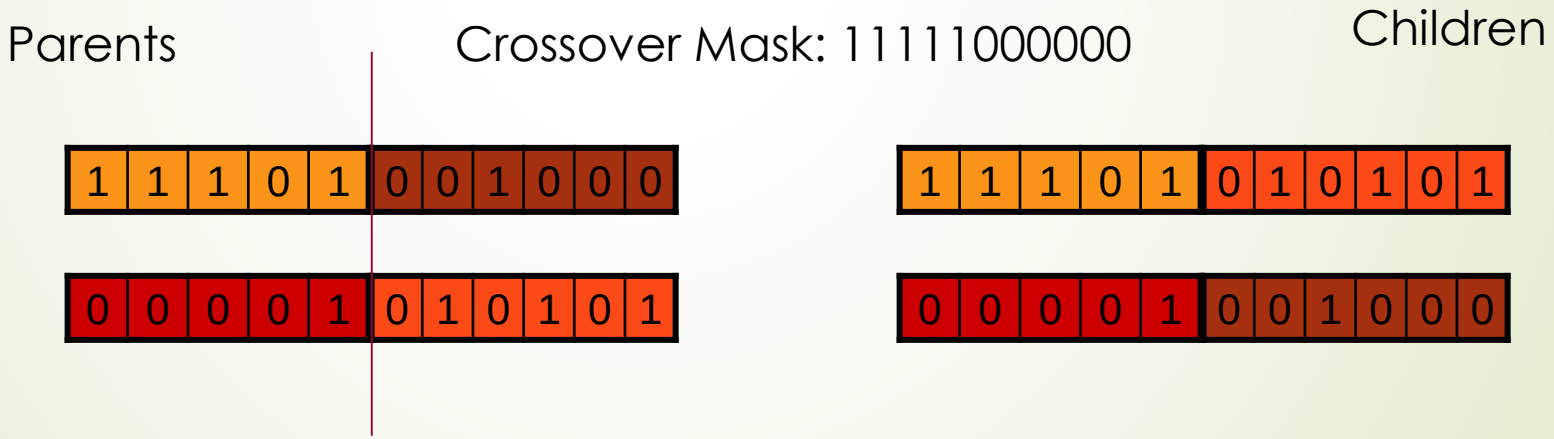
- الگوریتم های ژنتیک در مسائلی که فضای جستجوی بزرگی داشته باشند میتواند بکار گرفته شود.
- همچنین در مسایلی با فضای فرضیه پیچیده که تاثیر اجزا آن در فرضیه کلی ناشناخته باشند میتوان از GA برای جستجو استفاده نمود.
- الگوریتم های ژنتیک را میتوان براحتی بصورت موازی اجرا نمود از اینرو میتوان کامپیوترهای ارزان قیمت تری را بصورت موازی مورد استفاده قرار داد.
- امکان به تله افتادن این الگوریتم در مینیمم محلی کمتر از سایر روشهاست.
- از لحاظ محاسباتی پرهزینه هستند.
- تضمینی برای رسیدن به جواب بهینه وجود ندارد.

اپراتورهای ژنتیکی: Crossover

- اپراتور Crossover با استفاده از دو رشته والد دو رشته فرزند بوجود میآورد .
- برای اینکار قسمتی از بیتهای والدین در بیتهای فرزندان کپی میشود .
- انتخاب بیت هائی که باید از هر یک از والدین کپی شوند به روشهای مختلف انجام میشود
  - single-point crossover
  - Two-point crossover
  - Uniform crossover
- برای تعیین محل بیتهای کپی شونده از یک رشته به نام Crossover Mask استفاده میشود.

Single-point crossover

- یک نقطه تصادفی در طول رشته انتخاب میشود .
- والدین در این نقطه به دوقسمت میشوند.
- هر فرزند با انتخاب تکه اول از یکی از والدین و تکه دوم از والد دیگر بوجود میاید.



روشهای دیگرCrossover

- Two-point crossover

Crossover Mask: 00111110000

Parents

Children

- Uniform crossover

Parents

Crossover Mask: 10011010011

Children

بیتها بصورت یکنواخت از والدین انتخاب میشوند

اپراتورهای ژنتیکی : Mutation

- اپراتور mutation برای بوجود آوردن فرزند فقط از یک والد استفاده میکند .اینکار با انجام تغییرات کوچکی در رشته اولیه بوقوع میپیوندد.
- با استفاده از یک توزیع یکنواخت یک بیت بصورت تصادفی اتنخاب و مقدار آن تغییر پیدا میکند.
- معمولا mutation بعد از انجام crossover اعمال میشود.

Crossover OR mutation?

- این سوال ها سالها مطرح بوده است:
- کدامیک بهتر است؟ کدامیک لازم است؟ کدامیک اصلی است؟
- پاسخی که تاکنون بیشتر از بقیه پاسخها مورد قبول بوده:
  - بستگی به صورت مسئله دارد
  - در حالت کلی بهتر است از هر دو استفاده شود
  - هر کدام نقش مخصوص خود را دارد
  - میتوان الگوریتمی داشت که فقط از mutation استفاده کند ولی الگوریتمی که فقط از crossoverاستفاده کند کار نخواهد کرد

Crossover OR mutation?

- Crossoverخاصیت جستجوگرانه و یا explorative دارد .میتواندبا انجام پرشهای بزرگ به محل هائی دربین والدین رفته و نواحی جدیدی را کشف نماید .
- Mutationخاصیت گسترشی و یا exploitive دارد .میتواند با انجام تغییرات کوچک تصادفی به نواحی کشف شده وسعت ببخشد.
- Crossove اطلاعات والدین را ترکیب میکند درحالیکه mutation میتواند اطلاعات جدیدی اضافه نماید .
- برای رسیدن به یک پاسخ بهینه یک خوش شانسی در mutation لازم است.