
جستجوی آگاهانه (اکتشافی)
دانشگاه صنعتی قوچان
محمدحسین سیگاری
هوش مصنوعی و سیستم خبره


فهرست
مقایسه جستجوی ناآگاهانه و آگاهانه
جستجوی آگاهانه
A*، IDA*، RBFS، SMA*
بحث روی تابع اکتشاف
جستجوی محلی و بهینه سازی
جستجوی آنلاین
۲
هوش مصنوعی و سیستم خبره

# جستجوی ناآگاهانه و آگاهانه

- جستجوی ناآگاهانه (Uninformed)
  - جستجوی کور (Blind)
  - الگوریتم هیچ اطلاعاتی غیر از تعریف مسئله در اختیار ندارد

- جستجوی آگاهانه (Informed)
  - جستجوی اکتشافی (Heuristic)
  - الگوریتم اطلاعاتی غیر از اطلاعات مسئله در اختیار دارد که معمولا در غالب اطلاعات اکتشافی (heuristic) و غیردقیق است. اما می تواند برای حل سریعتر مسئله مورد استفاده قرار گیرد.

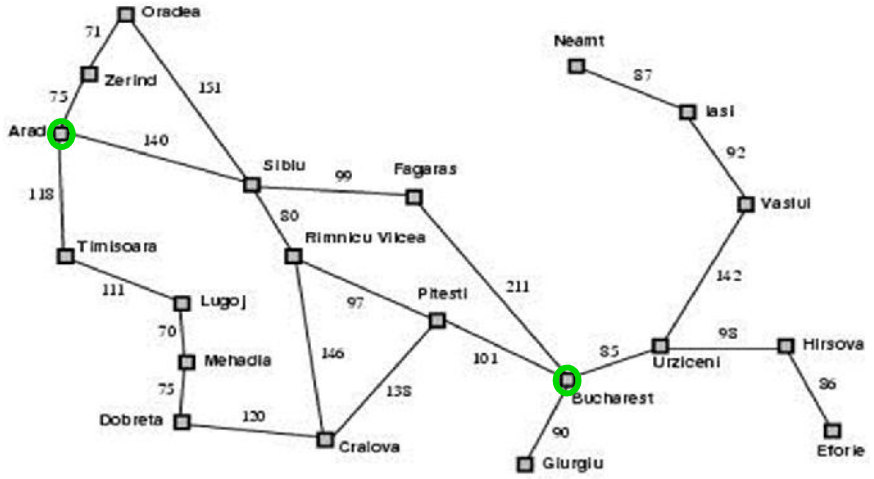


# جستجوی ناآگاهانه و آگاهانه

- اطلاعات اصلی مسئله: g(n)
  - هزینه صرف شده برای رسیدن به حالت n از حالت اولیه
  - g([initial state])=0 و g(n)>0
- اطلاعات اضافی (هیوریستیک): h(n)
  - هزینه تخمینی لازم برای اینکه از حالت n به حالت هدف برسیم
  - h([goal state])=0 و h(n)>0

- [dictionary]
  - Heuristic: “A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood.”

# مثال: سفر در کشور رومانی






# مثال: نقشه کشور رومانی

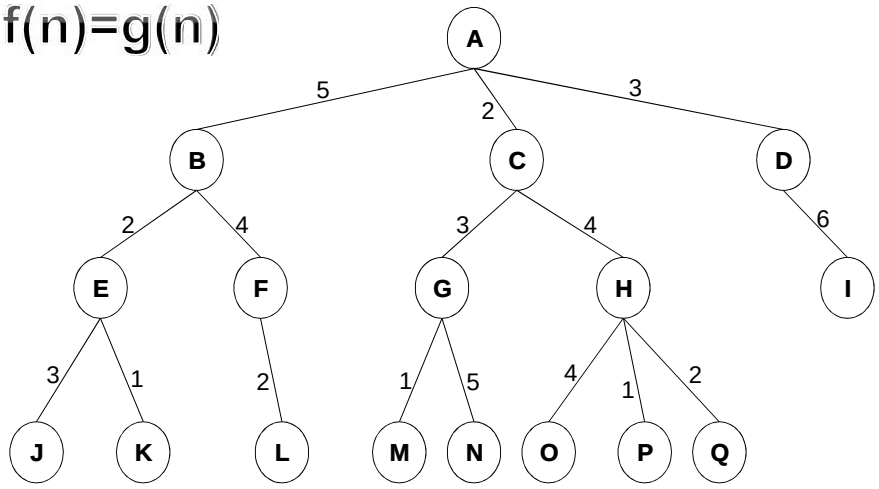
- اطلاعات اصلی مسئله: g(n)
  - هزینه (فاصله طی شده) از Arad تا شهر فعلی (n)
- اطلاعات اضافی (هیوریستیک): h(n)
  - هزینه (فاصله) تخمینی از شهر فعلی (n) تا شهر Bucharest

$H_{SLD}(X,Bucharest)$

| | | | |
|---|---|---|---|
| Arad | 366 | Mehadia | 241 |
| Bucharest | 0 | Neamt | 234 |
| Craiova | 160 | Oradea | 380 |
| Dobreta | 242 | Pitesti | 100 |
| Eforie | 161 | Rimnicu Vilcea | 193 |
| Fagaras | 176 | Sibiu | 253 |
| Giurgiu | 77 | Timisoara | 329 |
| Hirsova | 151 | Urziceni | 80 |
| Iasi | 226 | Vaslui | 199 |
| Lugoj | 244 | Zerind | 374 |

## جستجوی هزینه یکنواخت (UCS)

f(n)=g(n)

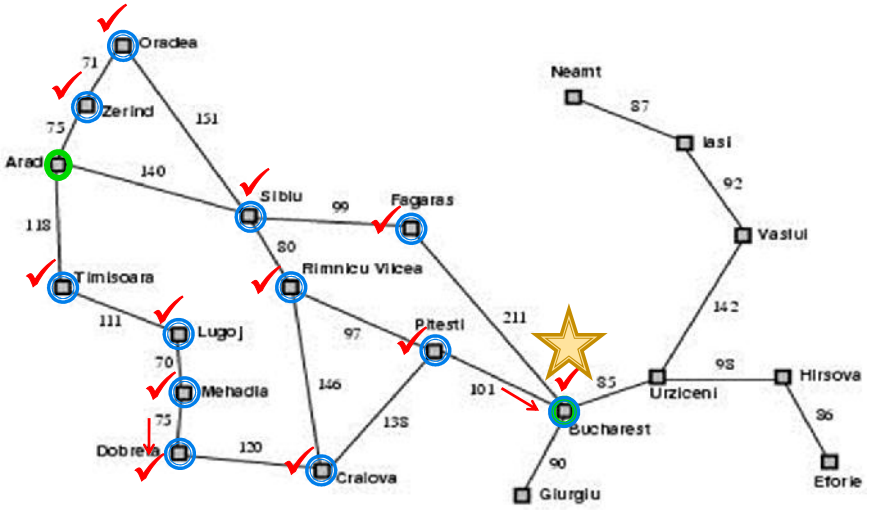





## جستجوی هزینه یکنواخت (UCS)

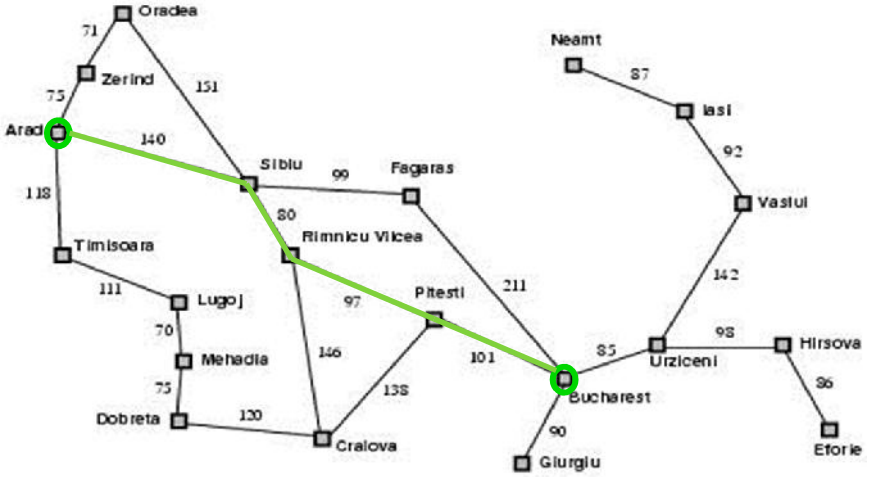
- کامل بودن:
  - کامل است، اگر هزینه مسیر غیرنزولی باشد. به عبارت دیگر هزینه هر یال غیرصفر باشد
  - اگر هزینه مسیر نزولی باشد، هدف در عمق بی نهایت قرار دارد
- بهینه بودن:
  - بهینه است
- پیچیدگی زمانی:
  - $O(b^{[C^*/e]})$ که e کمترین هزینه هر یال و $C^*$ کمترین هزینه بهترین هدف است
- پیچیدگی فضا:
  - $O(b^{[C^*/e]})$ که e کمترین هزینه هر یال و $C^*$ کمترین هزینه بهترین هدف است

## جستجوی UCS در مثال نقشه کشور رومانی



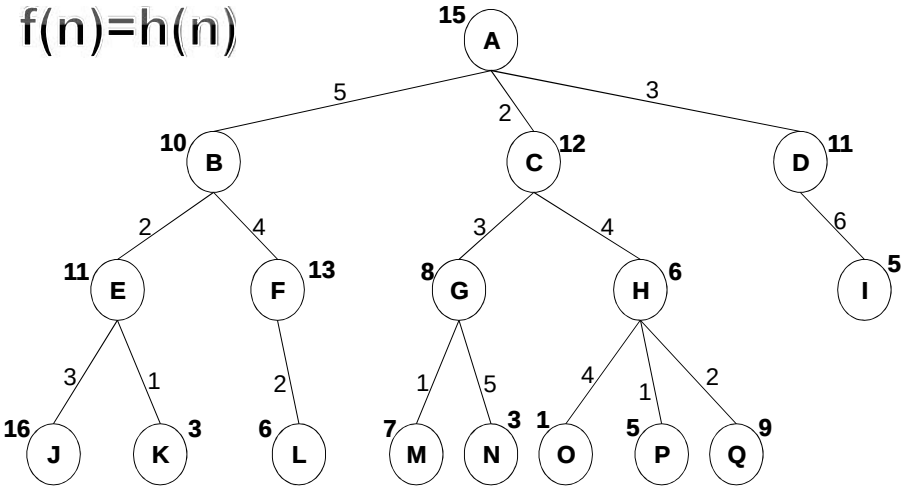



## جستجوی UCS در مثال نقشه کشور رومانی



## جستجوی حریصانه (Greedy)

f(n)=h(n)

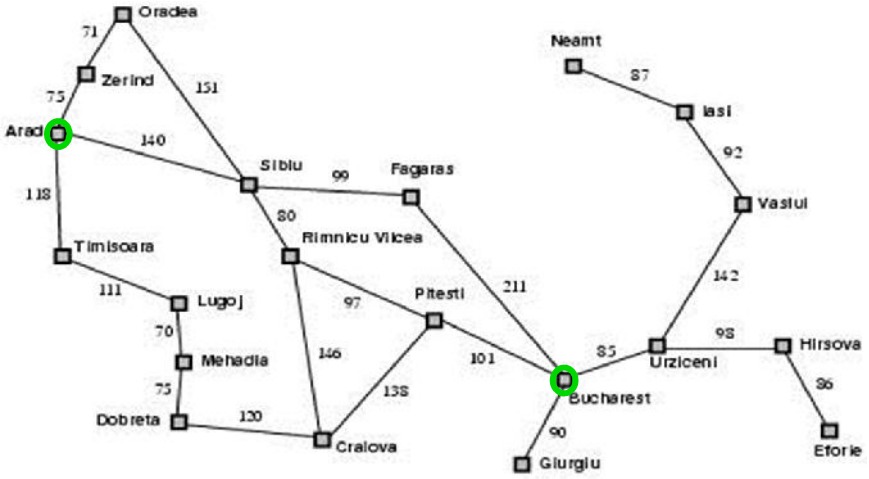





## جستجوی حریصانه (Greedy)

- کامل بودن:
  - کامل است، اگر حداکثر عمق درخت (m) بی نهایت نباشد
- بهینه بودن:
  - بهینه نیست، مگر اینکه تابع هیوریستیک به جای مقدار تخمینی، مقدار دقیق فاصله تا هدف را بیان کند. یعنی $h(n)=h^*(n)$
- پیچیدگی زمانی:
  - $O(b^m)$
    - اگر شرط بهینه بودن را داشته باشد $O(bd)$ خواهد شد
- پیچیدگی فضا:
  - $O(b^m)$
    - اگر شرط بهینه بودن را داشته باشد $O(bd)$ خواهد شد

# جستجوی حریصانه در مثال نقشه کشور رومانی

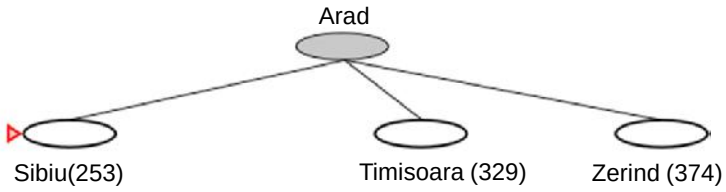





# جستجوی حریصانه در مثال نقشه کشور رومانی

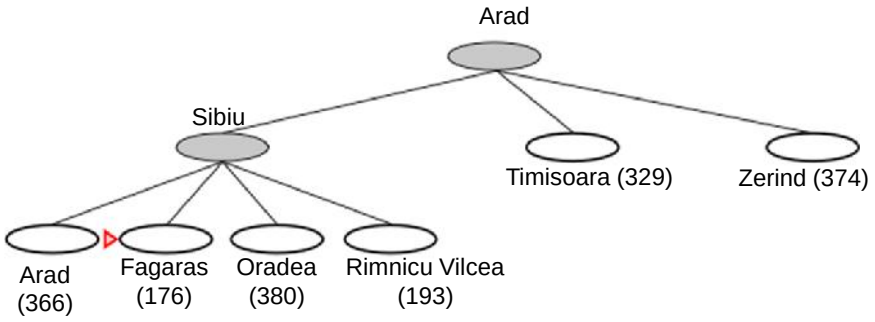
Arad (366)

## جستجوی حریصانه در مثال نقشه کشور رومانی

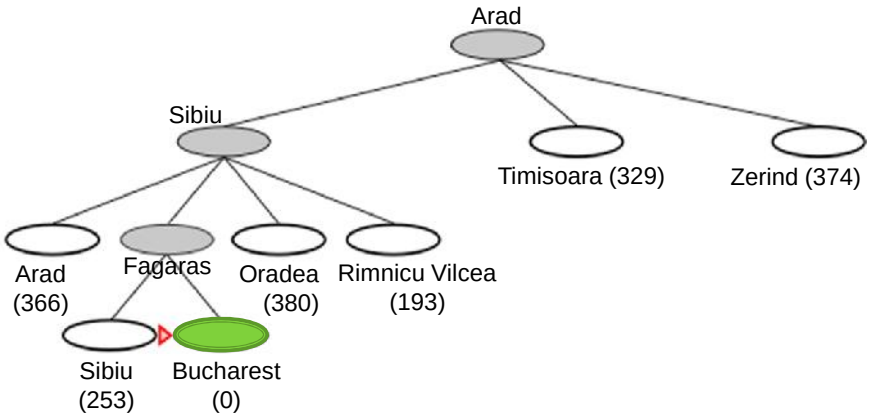





## جستجوی حریصانه در مثال نقشه کشور رومانی

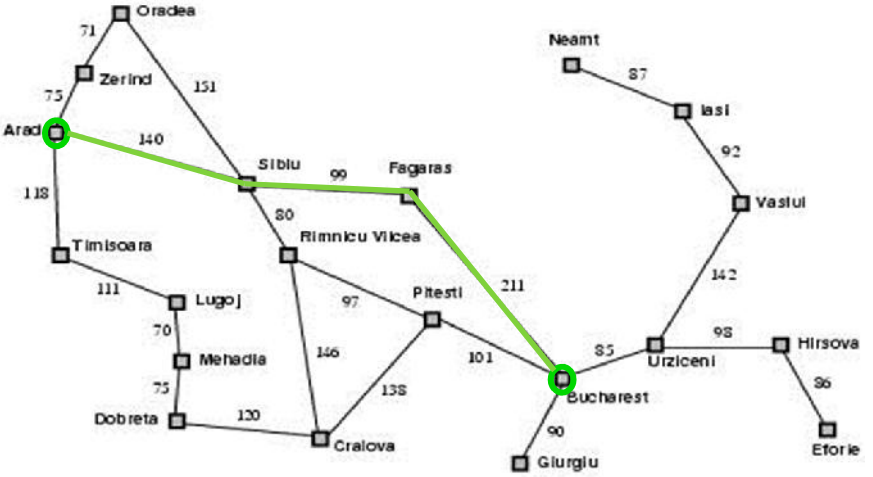


## جستجوی حریصانه در مثال نقشه کشور رومانی






## جستجوی حریصانه در مثال نقشه کشور رومانی



# فهرست

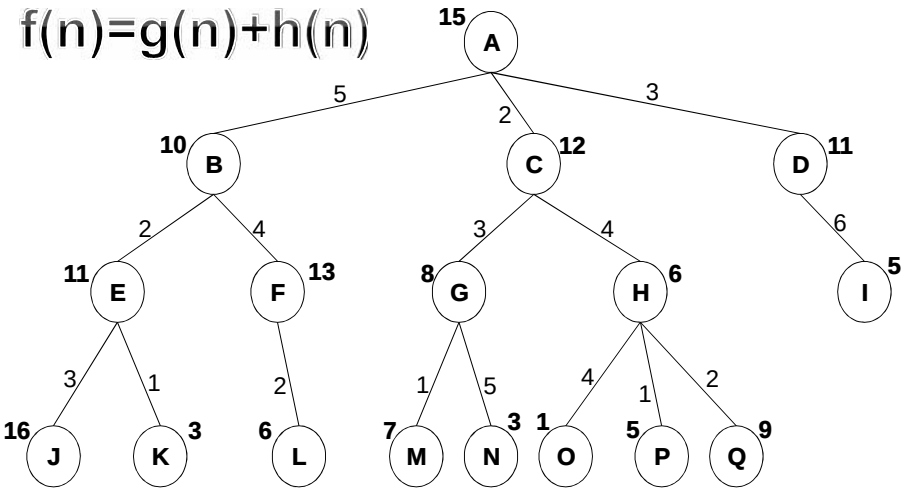
- مقایسه جستجوی ناآگاهانه و آگاهانه
- جستجوی آگاهانه
  - A*، IDA*، RBFS، SMA*
- بحث روی تابع اکتشاف
- جستجوی محلی و بهینه سازی
- جستجوی آنلاین



# جستجوی A*

- برای بسط یک گره، همانند جستجوی UCS به هزینه ای که تاکنون پرداخت شده توجه دارد
- برای بسط یک گره، همانند جستجوی Greedy به هزینه تخمینی از گره فعلی تا گره هدف توجه دارد
- $f(n)=g(n)+h(n)$
  - هزینه تخمینی مسیر از حالت اولیه تا حالت هدف به طوری که این مسیر از حالت n بگذرد

## جستجوی A*






## جستجوی A*

- کامل بودن:
  - کامل است
- بهینه بودن:
  - بهینه است اگر h(n) خاصیت admissible داشته باشد
- پیچیدگی زمانی:
  - $O(b^m)$
    - اگر h(n) مقدار دقیق هزینه گره n تا هدف را بیان کند، O(bd) خواهد شد
- پیچیدگی فضا:
  - $O(b^m)$
    - اگر h(n) مقدار دقیق هزینه گره n تا هدف را بیان کند، O(bd) خواهد شد

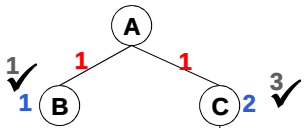
## خواص تابع اکتشاف (heuristic)

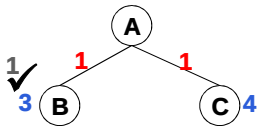
- خاصیت قابل پذیرش بودن (Admissible)
- اگر $h^*(n)$ مقدار دقیق فاصله گره n تا هدف را نشان دهد، $h(n)$ را قابل پذیرش می گوییم اگر به ازای تمام nها: $h(n) \leq h^*(n)$
- به عبارت دیگر، اگر $h(n)$ بیشتر از حد واقعی هزینه را تخمین نزند (over estimate)، قابل پذیرش است.
- اگر $h(n)$ قابل پذیرش باشد، می توان تضمین کرد که تمام گره هایی که $f(n) < f^*(n)$ بوده، مورد بررسی قرار گرفته و جواب بدست آمده بهینه است



## خواص تابع اکتشاف (heuristic)

### جستجوی A*

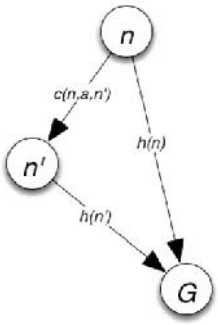



$h \leq h^*$ $h \nleq h^*$

## خواص تابع اکتشاف (heuristic)

- خاصیت سازگاری (consistency) یا یکنوایی (monotonicity)
- اگر h(n) در طول همه مسیرها غیرصعودی باشد، می گوییم h(n) سازگار است




- به عبارت دیگر:
- $h(n) \leq c(n,a,n') + h(n')$
- این خاصیت معادل با نامساوی مثلثی است



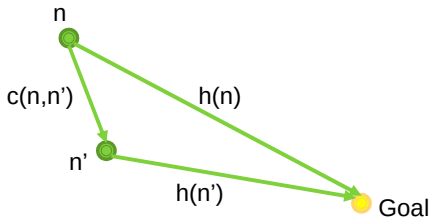
## بحث روی الگوریتم A*

- اگر فرض کنیم $C^*$ هزینه بهترین هدف (که کمترین هزینه را دارد) باشد و h(n) قابل پذیرش باشد
- الگوریتم A* تمام گره هایی که $f(n) < C^*$ را بسط می دهد
- الگوریتم A* بعضی از گره هایی که $f(n) = C^*$ را بسط می دهد
- الگوریتم A* هیچ گره از گره هایی که $f(n) > C^*$ را بسط نمی دهد

# جستجوی A* در مثال نقشه کشور رومانی

- به عنوان مثال در نقشه شهر رومانی، $h_{SLD}(n)$ خواص زیر را دارد:

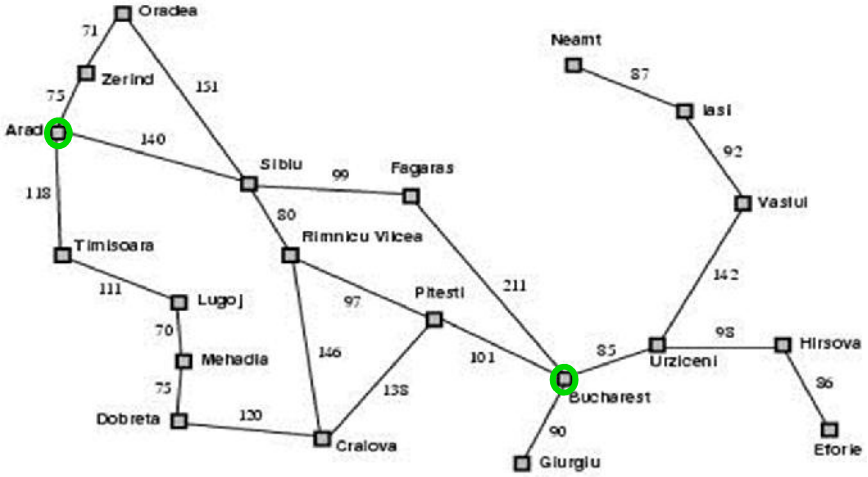
- خاصیت قابل پذیرش بودن:
  - بلی
  - فاصله مستقیم هر شهر تا شهر دیگر کوچکتر یا مساوی فاصله آن دو شهر بر اساس راه های زمینی است



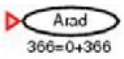

- خاصیت سازگاری (یکنوایی):
  - بلی
  - $h_{SLD}(n) \leq d(n,n') + h_{SLD}(n')$



# جستجوی A* در مثال نقشه کشور رومانی

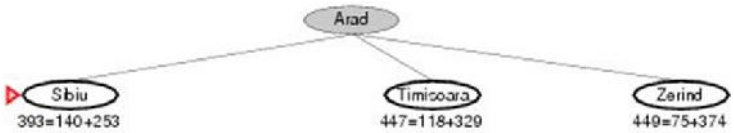


## جستجوی A* در مثال نقشه کشور رومانی

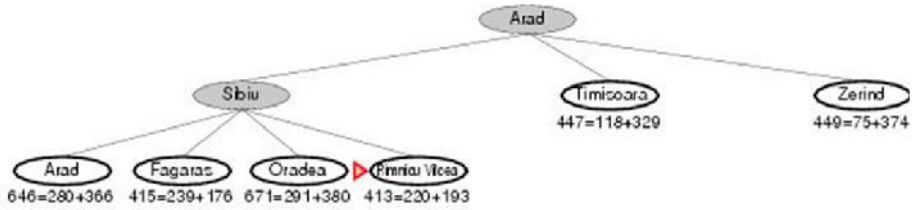





## جستجوی A* در مثال نقشه کشور رومانی

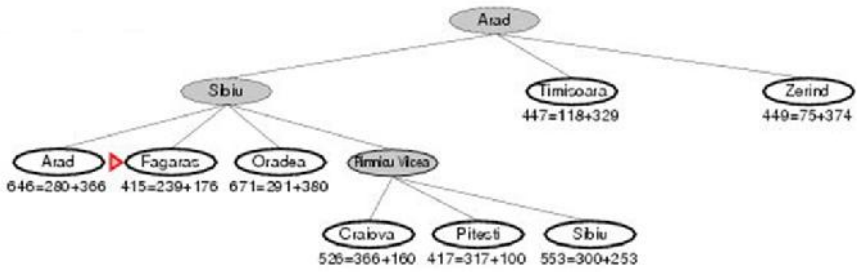


## جستجوی A* در مثال نقشه کشور رومانی

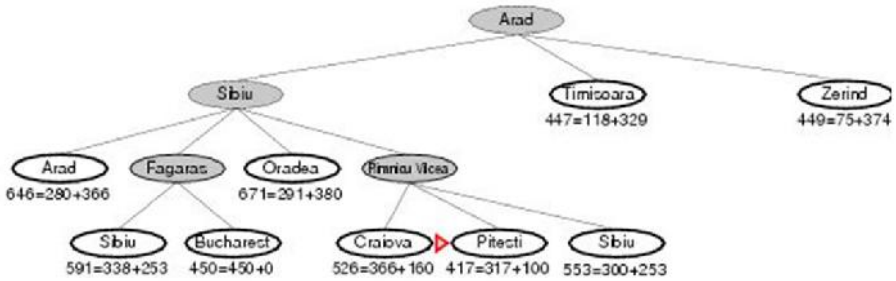





## جستجوی A* در مثال نقشه کشور رومانی

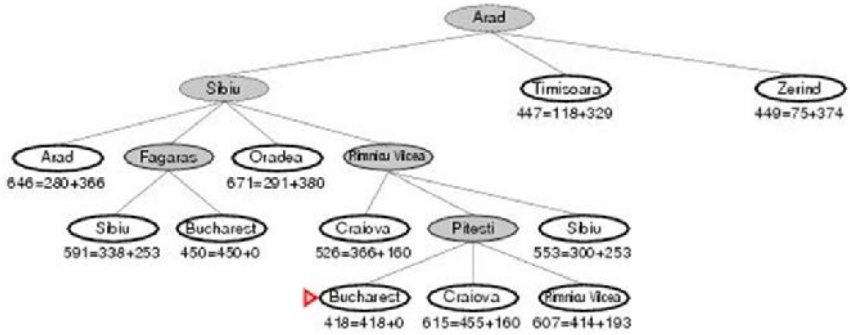


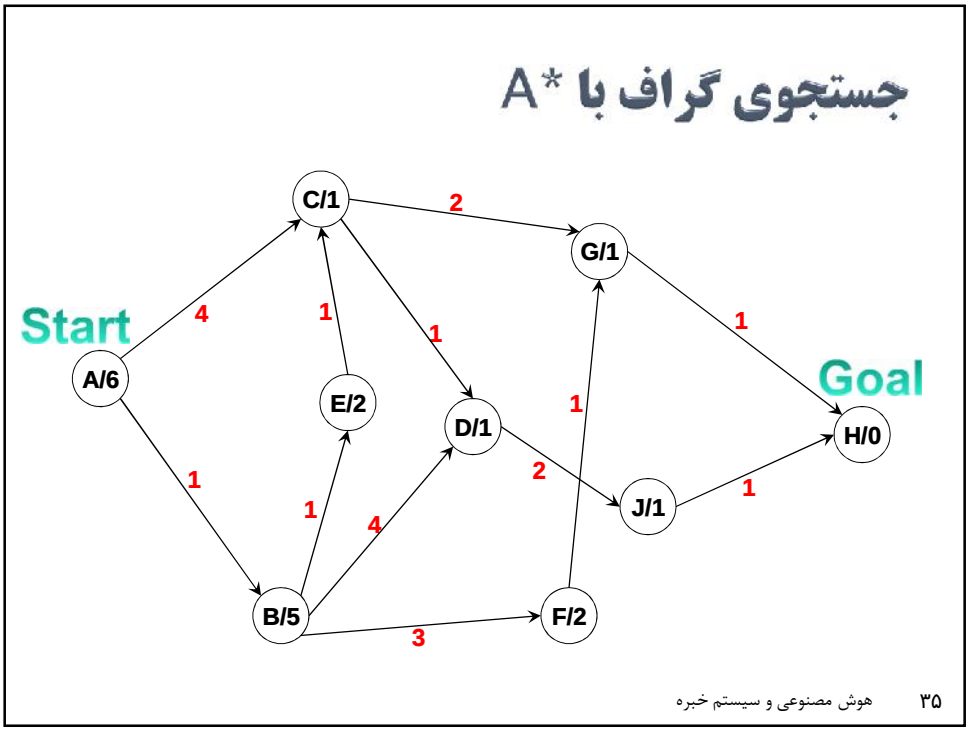
## جستجوی A* در مثال نقشه کشور رومانی

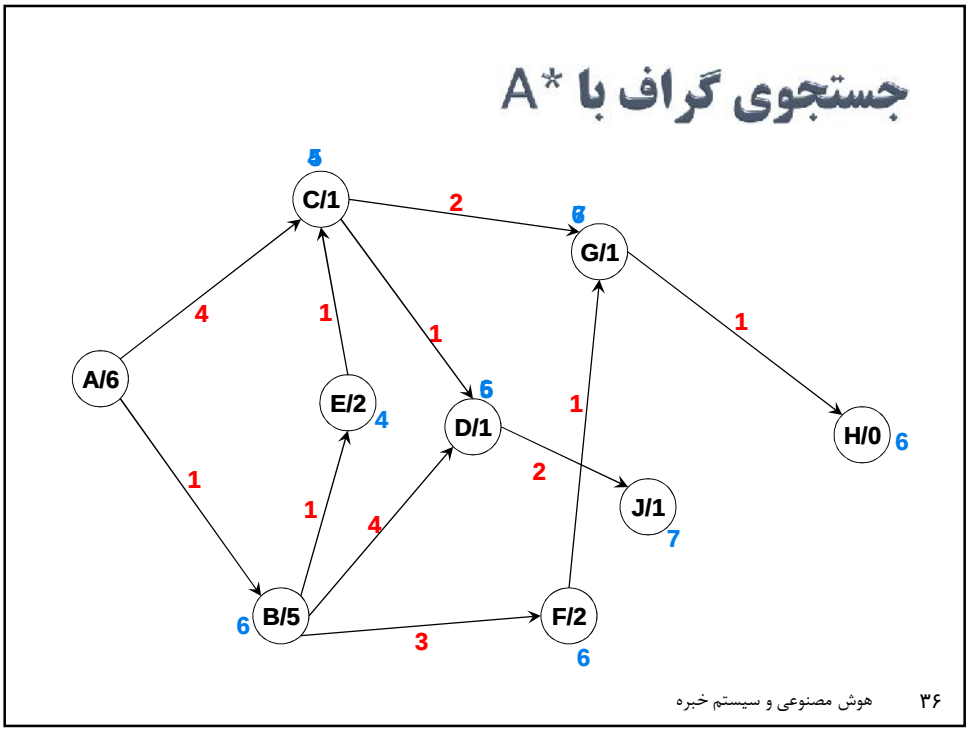




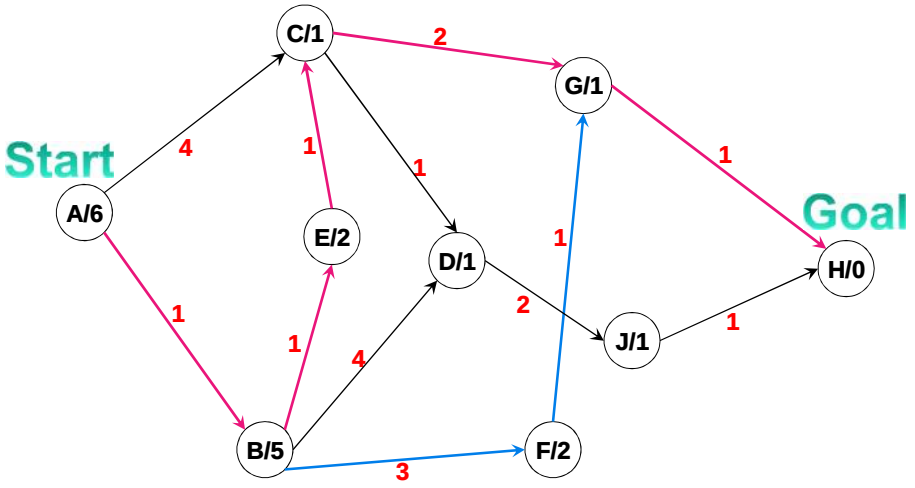

## جستجوی A* در مثال نقشه کشور رومانی




جستجوی گراف با A*
Start
Goal
C/1
G/1
A/6
E/2
D/1
H/0
J/1
B/5
F/2
4
1
2
1
1
1
1
2
1
1
4
3
هوش مصنوعی و سیستم خبره
۳۵


جستجوی گراف با A*
C/1
G/1
A/6
E/2
D/1
H/0
J/1
B/5
F/2
4
1
2
1
1
1
1
2
1
1
4
3
6
هوش مصنوعی و سیستم خبره
۳۶

## جستجوی گراف با A*






## بررسی الگوریتم A*

- مزایای الگوریتم A*
  - کامل است
  - در صورتی که h(n) قابل پذیرش باشد، الگوریتم A* بهینه است
- معایب الگوریتم A*
  - اگر h(n) تابع اکتشافی خوبی نباشد، A* از نظر پیچیدگی زمانی ضعیف خواهد بود.
  - A* تمام گره هایی را که بسط می دهد، در حافظه نگه می دارد. بنابراین از نظر پیچیدگی فضا ضعیف است. بنابراین اگر h(n) تابع اکتشافی خوبی نباشد، از نظر پیچیدگی فضا نیز ضعیف خواهد بود.

**پیچیدگی فضا مهمتر از پیچیدگی زمانی**

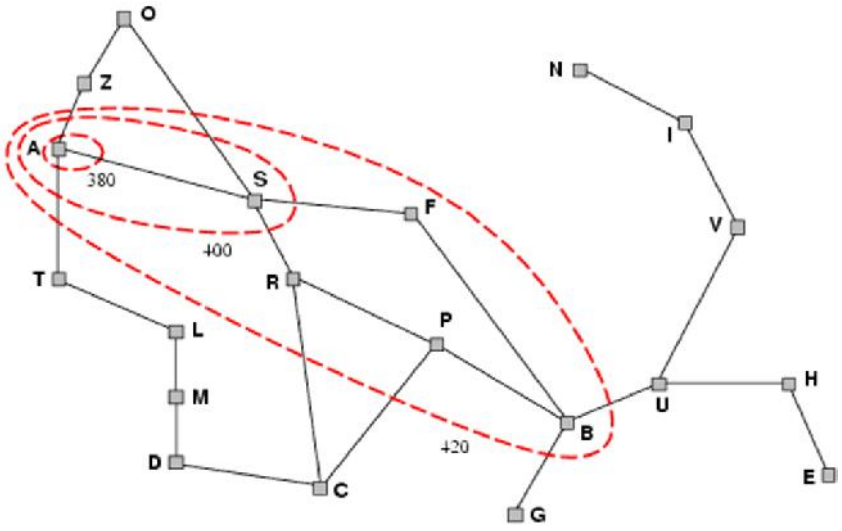
- هر چه h(n) به h*(n) نزدیک تر باشد، بهتر است

# جستجوی IDA*

- IDA*=Iterative Deepening A*
- استفاده از الگوریتم DFS
- IDA* شبیه به IDS است، با این تفاوت که به جای محدود کردن عمق، مقدار f را محدود کرده و در هر بار تکرار، حداکثر مقدار f را افزایش می دهد
- برای مسائلی که هزینه مراحل به صورت پله ای افزایش می یابد، مناسب است.



# جستجوی IDA*



## جستجوی IDA*

- کامل بودن:
  - کامل است
- بهینه بودن:
  - بهینه است اگر
    - h(n) خاصیت admissible داشته باشد
    - مقادیر پله ای برای افزایش f مناسب انتخاب گردد
- پیچیدگی زمانی:
  - $O(b^m)$
- پیچیدگی فضا:
  - O(bm)



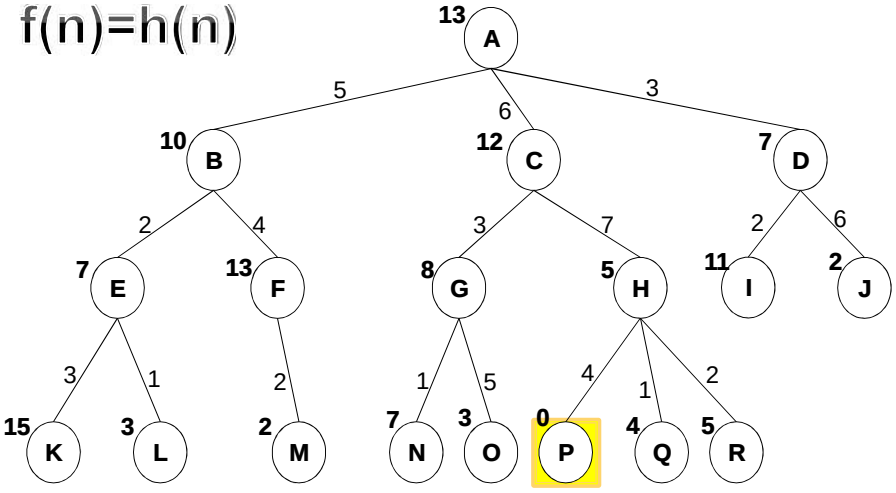
## جستجوی RBFS

- RBFS=Recursive Best First Search

- شبیه به الگوریتم DFS عمل می کند، با این تفاوت که سعی می کند اطلاعات مختصری از نوادگان (Successor) خود که هنوز آنها را بسط نداده و به عقب بازگشت (back track) کرده، نگه دارد.

## جستجوی RBFS

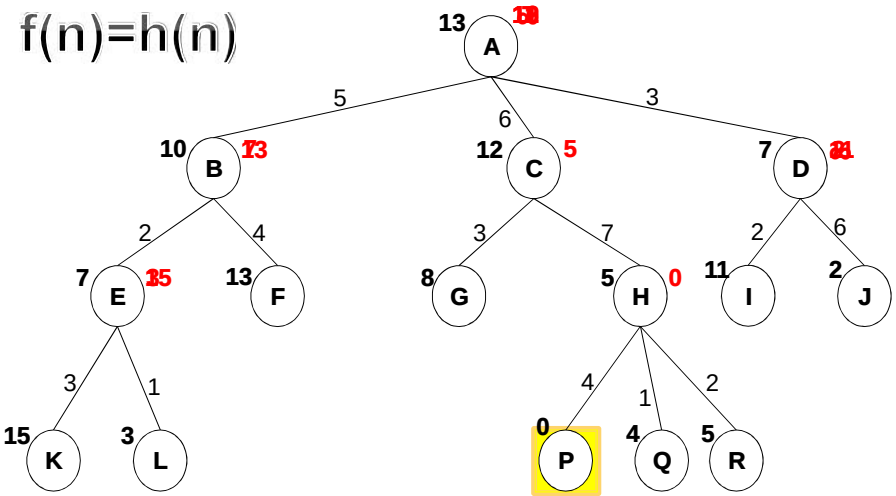
f(n)=h(n)

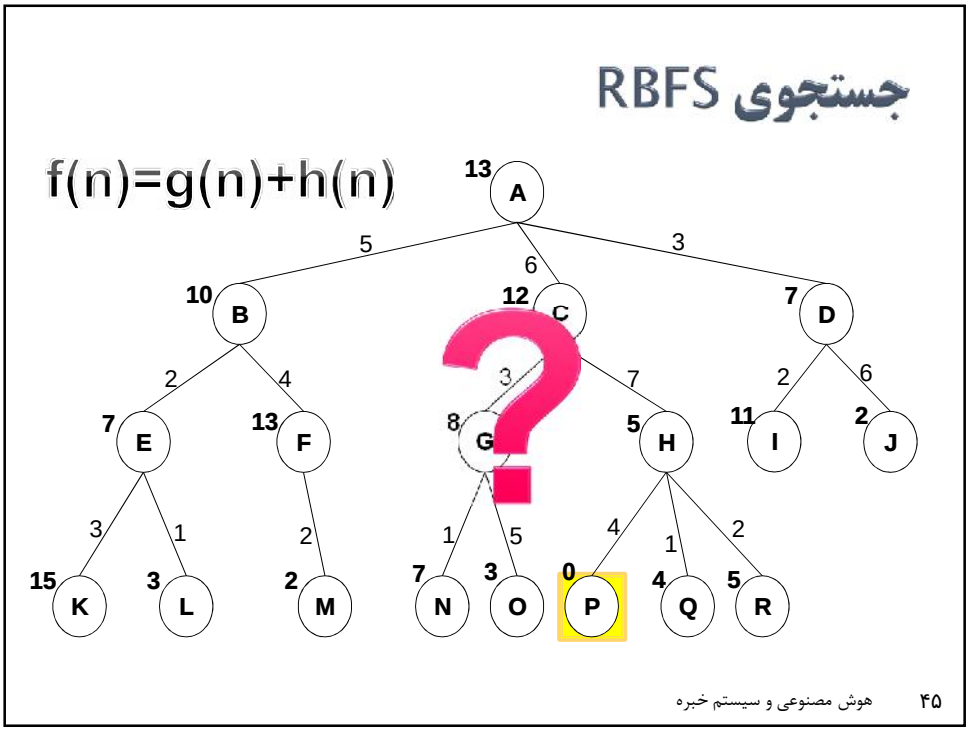


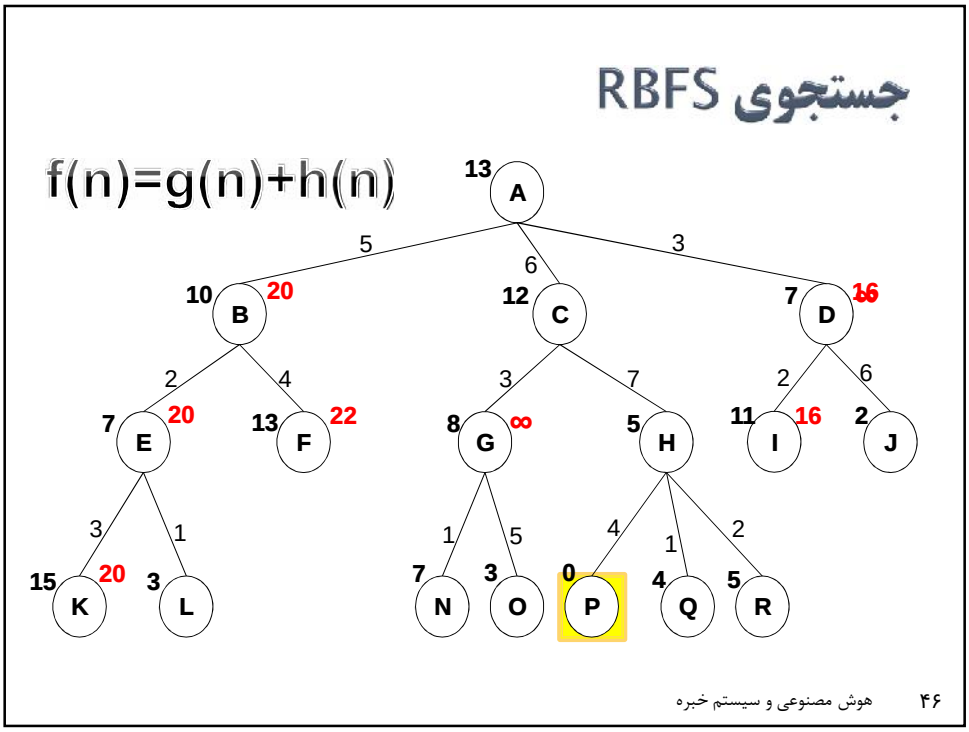



## جستجوی RBFS

f(n)=h(n)




جستجوی RBFS
f(n)=g(n)+h(n)
A 13
B 10
C 12
D 7
E 7
F 13
G 8
H 5
I 11
J 2
K 15
L 3
M 2
N 7
O 3
P 0
Q 4
R 5
5 3 6 2 4 3 7 2 6 3 1 2 1 5 4 1 2
هوش مصنوعی و سیستم خبره ۴۵


جستجوی RBFS
f(n)=g(n)+h(n)
A 13
B 10 20
C 12
D 7 16
E 7 20
F 13 22
G 8 ∞
H 5
I 11 16
J 2
K 15 20
L 3
N 7
O 3
P 0
Q 4
R 5
5 3 6 2 4 3 7 2 6 3 1 1 5 4 1 2
هوش مصنوعی و سیستم خبره ۴۶

## جستجوی RBFS

- کامل بودن:
  - کامل است
- بهینه بودن:
  - بهینه است اگر
    - $f(n)=g(n)+h(n)$
    - $h(n)$ خاصیت admissible داشته باشد
- پیچیدگی زمانی:
  - $O(b^m)$
- پیچیدگی فضا:
  - $O(bm)$



## جستجوی SMA*

- SMA*=Simple Memory-bounded A*
- SMA* بهترین برگ را بسط می دهد تا زمانی که حافظه پر شود. در این حالت، بدون از بین بردن گره های قبلی نمی توان گره جدیدی اضافه کرد
- SMA* پس از پر شدن حافظه، بدترین گره برگ و زیر درخت متعلق به آن را حذف می کند و سپس از طریق گره فراموش شده به والد آن بر می گردد. بنابراین همواره جد زیردرخت فراموش شده، کیفیت بهترین مسیر را در آن زیر درخت می داند. اگر ارزش تمام گره های موجود در حافظه یکسان باشد، گره قدیمی تر حذف خواهد شد

## جستجوی SMA*

- کامل بودن:
  - کامل است، اگر تعداد گره های قابل تولید در حافظه، بیشتر از **کمترین عمق هدف** **(لزوما این هدف، بهترین هدف نیست)** باشد
- بهینه بودن:
  - بهینه است اگر
    - h(n) خاصیت admissible داشته باشد
    - تعداد گره های قابل تولید در حافظه، بیشتر از **عمق بهترین هدف** باشد
- پیچیدگی زمانی:
  - ؟؟؟؟؟ (محدودیت های حافظه ممکن است مسئله را از نظر زمان محاسباتی، غیر قابل حل کند)
- پیچیدگی فضا:
  - برابر با تعداد گره های قابل تولید در حجم حافظه محدود



## مقایسه A*، IDA*، RBFS و SMA*

| | A* | IDA* | RBFS | SMA* |
|---|---|---|---|---|
| Time Complexity | ★★★★ | ★ | ★★ | ★★★ |
| Space Complexity | ★ | ★★ | ★★★★ | ★★★ |

# فهرست

- مقایسه جستجوی ناآگاهانه و آگاهانه
- جستجوی آگاهانه
  - A*، IDA*، RBFS، SMA*
- بحث روی تابع اکتشاف
- جستجوی محلی و بهینه سازی
- جستجوی آنلاین



# بررسی کیفیت تابع اکتشاف

- هزینه جستجو
  - تعداد گره های تولید شده توسط الگوریتم A*
- فاکتور انشعاب موثر (Effective Branch Factor)
  - اگر تعداد N گره توسط الگوریتم A* تولید شود و عمق راه حل مسئله d باشد، b* را فاکتور انشعاب موثر می گویند که
  - $N+1=1+b^*+(b^*)^2+\ldots+(b^*)^d$
- برای یک تابع اکتشافی خوب، هزینه جستجو باید به سمت d و فاکتور انشعاب موثر باید به سمت ۱ میل کند.

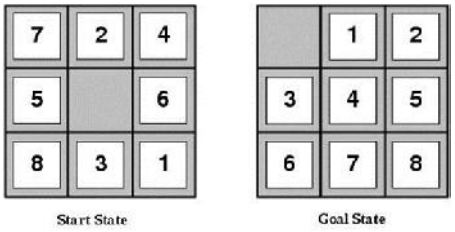
# مثالی از پیچیدگی مسائل واقعی

- بازی شطرنج:
  - فاکتور انشعاب موثر=حدود ۳۵
  - متوسط تعداد حرکت هر بازیکن تا پایان بازی=۵۰ حرکت
  - متوسط تعداد کل حرکت بازیکنان تا پایان بازی=۱۰۰ حرکت

- متوسط تعداد گره های بسط داده شده توسط الگوریتم A* برای بازی شطرنج
  - $N+1=1+35+(35)^2+\ldots+(35)^{100}>35^{100}>2.5\times10^{154}$
- اگر فرض کنیم هر گره فقط ۱ بایت باشد، نیاز به بیش از $2^{500}$ بایت حافظه است. این حافظه تقریبا برابر $10^{145}$ گیگابایت می باشد.

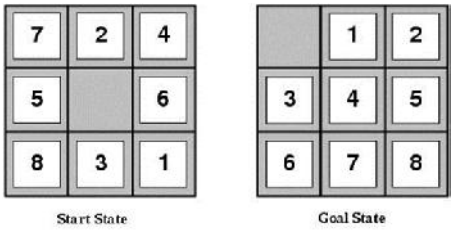


# تابع اکتشاف برای مسئله 8-Puzzle




- $h_1(n)=$ تعداد کاشی هایی که در جای اصلی خود قرار ندارند
- $h_1(n)=8$
- $h_1(n)$ قابل پذیرش (admissible) است
- $h_1(n)$ سازگار (consistent) است

## تابع اکتشاف برای مسئله 8-Puzzle




- $h_2(n)$=مجموع فواصل کاشی ها از موقعیت های هدف آنها
- $h_2(n)=18$
- $h_2(n)$ قابل پذیرش (admissible) است
- $h_2(n)$ سازگار (consistent) است

**Manhattan Distance = City Block Distance**



## مقایسه توابع اکتشاف در مسئله 8-Puzzle

- برای ۱۲۰۰ حالت مختلف از مسئله 8-Puzzle که عمق هدف بین ۲ تا ۲۴ متغیر بود، از سه روش جستجو برای حل مسئله استفاده گردید
- جستجوی IDS
- جستجوی A* با تابع اکتشاف $h_1(n)$
- جستجوی A* با تابع اکتشاف $h_2(n)$

# مقایسه توابع اکتشاف در مسئله 8-Puzzle

| | 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 | 3644035 | 227 | 73 | 2.78 | 1.42 | 1.24 |
| 14 | – | 539 | 113 | – | 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 |



# مقایسه توابع اکتشاف

- اگر دو تابع اکتشاف $h_1(n)$ و $h_2(n)$ قابل پذیرش باشد و به ازای هر n رابطه $h_1(n) \leq h_2(n)$ برقرار باشد، $h_2(n)$ تابع اکتشاف بهتری می باشد و می گوییم $h_2(n)$ بر $h_1(n)$ غالب (dominant) است.
- در این حالت، تعداد گره هایی که بر اساس $h_1(n)$ بسط داده خواهد شد، همواره بیشتر یا مساوی تعداد گره هایی که بر اساس $h_2(n)$ بسط داده می شود

- اگر $h_i(n)$ ها توابع اکتشاف قابل پذیرش باشند:
- $h_1(n) \leq h_2(n) \leq \ldots \leq h_i(n) \leq \ldots \leq h_m(n) \leq h^*(n)$
- تابع $h_m(n)$ بهترین تابع اکتشاف است

## ارائه تابع اکتشافی قابل پذیرش برای مسائل

هزینه یک روش ساده و آسان (relax) برای حل مسئله

استفاده از هزینه حل زیرمسائل (sub-problem)

تجربه و استفاده از دانش فرد خبره در حل مسئله



## فهرست

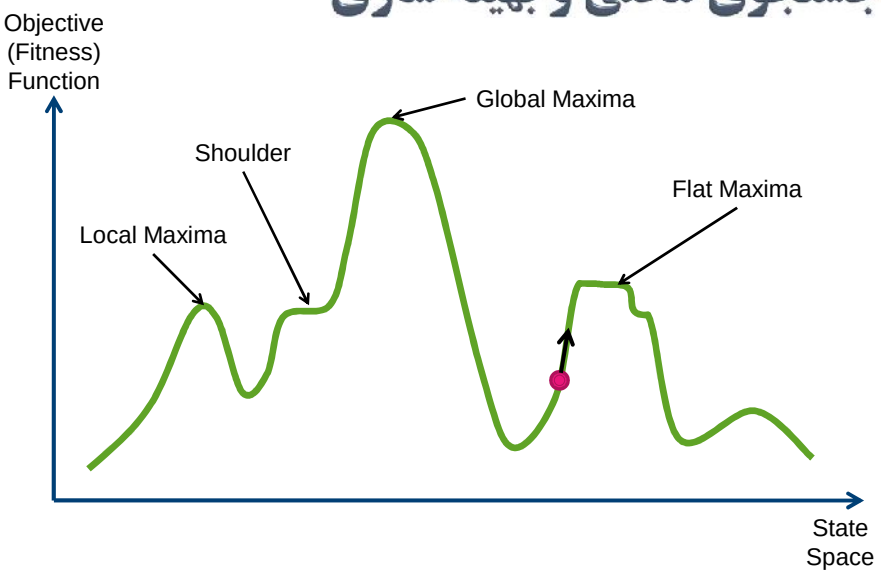
- مقایسه جستجوی ناآگاهانه و آگاهانه
- جستجوی آگاهانه
  - A*، IDA*، RBFS، SMA*
- بحث روی تابع اکتشاف
- جستجوی محلی و بهینه سازی
- جستجوی آنلاین

## تفاوت جستجوی معمولی و جستجوی محلی

| معیار | جستجوی معمولی | جستجوی محلی |
|---|---|---|
| کاربرد | - مسائلی که مسیر رسیدن به راه حل اهمیت دارد | - مسائلی که مسیر رسیدن به راه حل اهمیت ندارد<br>- مسائل بهینه سازی |
| فضای حالت | متناهی و محدود | نامتناهی یا بسیار بزرگ |
| حجم حافظه | زیاد / بسیار زیاد | بسیار کم / کم |
| حجم محاسبات | زیاد | کم |
| بهینه بودن | می توان الگوریتم را طوری انتخاب کرد که بهینه باشد | معمولا به جواب sub-optimal دست می یابد |



## جستجوی محلی و بهینه سازی



## الگوریتم تپه نوردی (Hill Climbing)

- در حالت فعلی، بر اساس حالت های همسایه (حالت هایی که می توان با انجام یک فعالیت به آنها رفت)، حالتی را انتخاب می کند که مقدار تابع برازش (Fitness Function) را بیشینه کند.
- تپه نوردی تا زمانی ادامه دارد که هیچ همسایه ای، بهتر از حالت فعلی نباشد.

- تپه نوردی = **جستجوی محلی حریصانه**

- معایب:
  - الگوریتم تپه نوردی ممکن است در ماکزیمم محلی گرفتار شود
  - الگوریتم تپه نوردی در نقاط flat یا shoulder بدون نتیجه خاتمه می یابد



## الگوریتم تپه نوردی در مسئله 8-Queens

- مسئله در حالت complete state را در نظر می گیریم، به طوری که هر وزیر در یک ستون قرار داشته باشد

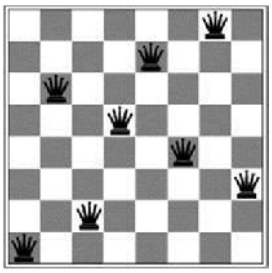
- تابع جانشین (successor): جابجایی یک وزیر در ستون خود

- f(n) برابر است با تعداد جفت وزیرانی که در گارد هم قرار دارند

Fitness Function

## الگوریتم تپه نوردی در مسئله 8-Queens

- شکل سمت چپ: حالتی با هزینه f=17 که مقدار f را برای هر جانشین نشان می دهد
- شکل سمت راست: کمینه محلی در فضای حالت ۸ وزیر؛ f=1



## انواع الگوریتم تپه نوردی

- تپه نوردی ساده (معمولی)
  - Hill Climbing
  - در هر لحظه فقط بهترین حالت همسایه را انتخاب می کند
- تپه نوردی غیرقطعی
  - Stochastic Hill Climbing
  - هر یک از حالت های همسایه را ممکن است به طور تصادفی انتخاب کند
  - میزان احتمال حرکت به سمت هر حالت همسایه، متناسب با میزان برازش آن خواهد بود

# انواع الگوریتم تپه نوردی

- تپه نوردی اولین انتخاب
  - First Choice Hill Climbing
  - حالت های همسایه را به طور تصادفی تولید کرده و اولین حالتی که بهتر از حالت فعلی باشد، انتخاب می کند
- تپه نوردی با شروع مجدد تصادفی
  - Random Restart Hill Climbing
  - همانند تپه نوردی ساده است، با این تفاوت الگوریتم چند بار اجرا می گردد و هر بار حالت اولیه آن به صورت تصادفی تعیین می گردد، در نهایت، بهترین حالت از چند اجرا به عنوان جواب خروجی خواهد بود



# الگوریتم Simulated Annealing

- مشابه روش Stochastic Hill Climbing
- ایده اصلی از نحوه ذوب و تهیه آلیاژ در صنعت متالوژی
  - هنگامی که فلزات مختلف در دمای بالا ذوب شده و به تدریج سرد می شوند، آلیاژ بدست آمده از آنها، دارای بیشترین پایداری (کمترین انرژی درونی) می باشد
- مثالی برای درک بهتر Simulated Annealing
  - یک توپ در یک فضای ناهموار (توپ=حالت کنونی، فضای ناهموار=مقدار تابع برازش در فضای حالت)
  - حرکت توپ به سمت نقطه ای که کمترین انرژی محلی را دارد
  - تکان دادن فضای ناهموار برای جابجایی موقت توپ برای اینکه به کمترین انرژی ممکن برسد

## Simulated Annealing الگوریتم

Energy Minimization



## Simulated Annealing الگوریتم

- در Simulated Annealing پارامتری به نام T وجود دارد که میزان دمای سیستم را نشان می دهد.
- در Simulated Annealing افزایش دما، به معنی افزایش انرژی سیستم و در نتیجه افزایش بی نظمی خواهد شد. افزایش بی نظمی با حرکات تصادفی بیشتر مدل می گردد.
- در ابتدای اجرای الگوریتم، حرکات تصادفی بیشتر است. اما با گذشت زمان، حرکات تصادفی کاهش یافته و در نهایت، Simulated Annealing مشابه Hill Climbing عمل خواهد کرد.

# الگوریتم جستجوی پرتو محلی

- جستجوی پرتو محلی (Local Beam Search)
- همانند الگوریتم تپه نوردی است، با این تفاوت که به جای بررسی یک نقطه از فضای حالت، به طور همزمان چند نقطه (k) مورد ارزیابی قرار می گیرند. نقاط اولیه، به صورت تصادفی تعیین می گردد.
- تفاوت جستجوی پرتو محلی با Random Restart Hill Climbing
  - در جستجوی پرتو محلی، چند نقطه ای که در فضای حالت جستجو می کنند، از وضعیت یکدیگر اطلاع دارند و اگر یک نقطه به نقطه بهینه برسد، الگوریتم متوقف می گردد، اما در تپه نوردی با شروع مجدد تصادفی، الگوریتم تپه نوردی با یک نقطه جستجو را انجام می دهد و این الگوریتم چند بار تکرار می گردد.
- جستجوی پرتو محلی غیرقطعی
  - همانند الگوریتم تپه نوردی غیرقطعی، حرکت k نقطه، لزوما به سمت بهترین همسایه نیست و حرکت آنها تصادفی تعیین می گردد

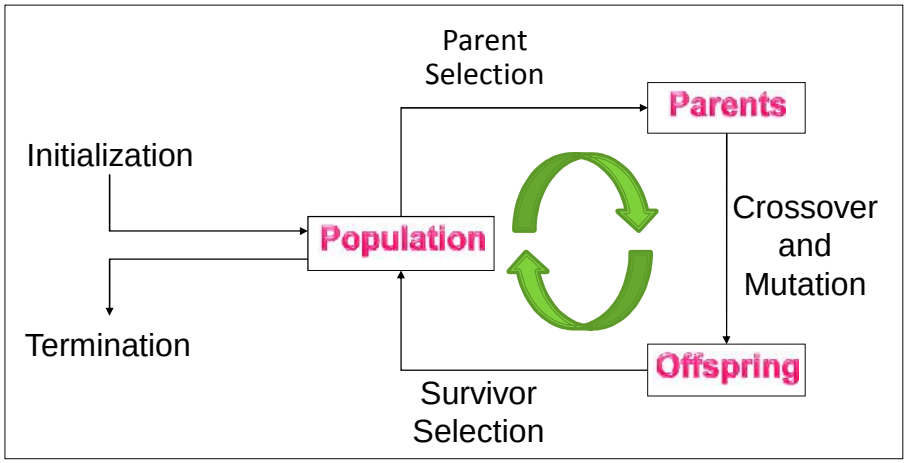


# الگوریتم ژنتیک

- الگوریتم ژنتیک=Genetic Algorithm (GA)

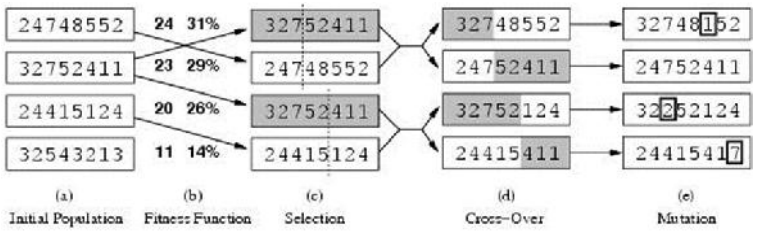
- مشابه الگوریتم پرتو محلی است، با این تفاوت که به جای اینکه k نقطه به صورت محلی جابجا شوند، k نقطه جدید بر اساس ترکیبی از k نقطه مرحله قبل تولید می شوند.

# الگوریتم ژنتیک



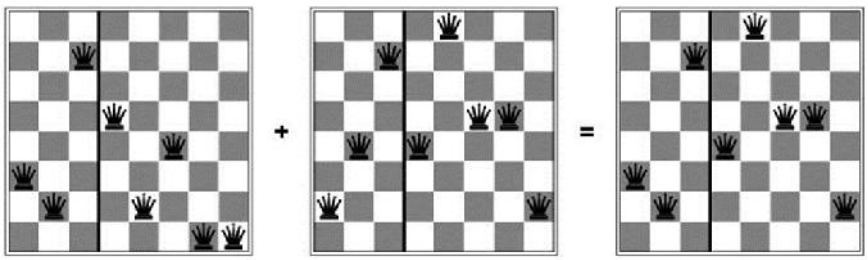



# الگوریتم ژنتیک در مسئله 8-Queens



## الگوریتم ژنتیک در مسئله 8-Queens

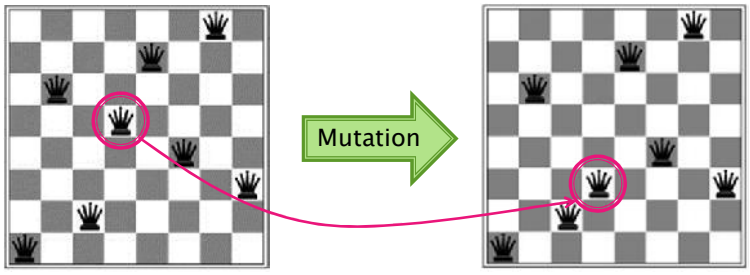
### Recombination





## الگوریتم ژنتیک در مسئله 8-Queens

### Mutation



# جستجوی محلی در فضای پیوسته

- در فضاهای پیوسته، تابع جانشین در اغلب موارد، حالت های نامتناهی را بر می گرداند
- روش های جستجوی محلی در فضای پیوسته
  - گسسته کردن فضای پیوسته
  - استفاده از روش کاهش گرادیان (Gradient Decent) برای مسئله کمینه کردن

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



# فهرست

- مقایسه جستجوی ناآگاهانه و آگاهانه
- جستجوی آگاهانه
  - A*، IDA*، RBFS، SMA*
- بحث روی تابع اکتشاف
- جستجوی محلی و بهینه سازی
- جستجوی آنلاین

## جستجوی Offline

- اکثر روش های قبلی، جستجوی offline بودند.
- در جستجوی offline باید:
  - محیط کاملا قابل مشاهده (fully observable) باشد
  - محیط ایستا (static) باشد
- در جستجوی offline ابتدا محاسبات انجام می شد، سپس مسیر راه حل بدست آمده و در نهایت راه حل بدست آمده اجرا می شد

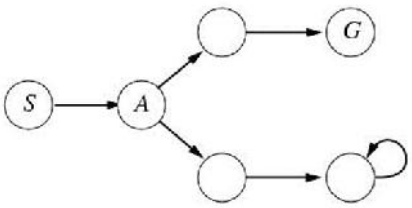


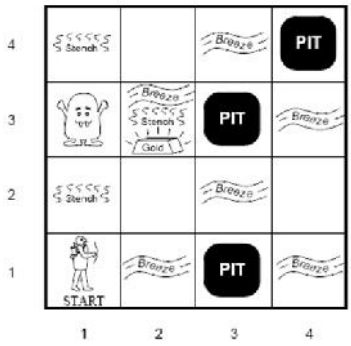
## جستجوی Online

- در جستجوی online عامل تنها موارد زیر را می داند
  - A(s): مجموعه actionهای مجاز در حالت s
  - C(s,a,s'): هزینه مرحله ای عامل هنگامیکه در حالت s فعالیت a را انجام دهد و به حالت s' برود. در اغلب موارد C(s,a,s') بعد از انجام a در حالت s و رفتن به حالت s' معلوم می شود.
  - Goal-Test(s): آزمون هدف برای بررسی حالت s
- در جستجوی online، عامل پس از انجام هر عملی (طی بخشی از مسیر راه حل)، باید محاسبات خود را تکرار کند و سپس فعالیت بهینه را انتخاب نماید.

# جستجوی Online

- جستجوی online لزوما همواره با موفقیت همراه نیست
  - گرفتار شدن در یک حلقه بسته (Closed Loop)
  - از بین رفتن عامل به خاطر انجام عملی که موجب نابودی عامل می گردد
    - چون عامل کاملا همه محیط را نمی شناسد و نتیجه کامل اعمال خود را نمی داند






# جستجوی محلی Online

- الگوریتم تپه نوردی می تواند به عنوان یک الگوریتم محلی Online مورد استفاده قرار گیرد
- مهمترین مشکل این الگوریتم، گرفتار شدن در ماکزیمم های محلی است
- به خاطر محلی بودن الگوریتم، از روش شروع مجدد تصادفی (Random Restart) نمی توان استفاده کرد. چون نمی توان عامل را در یک لحظه از نقطه به نقطه دیگر از فضای حالت منتقل کرد
- راه حل: افزودن فعالیت های تصادفی برای تضمین جستجوی تمام فضای حالت

## الگوریتم LRTA*

- LRTA*=Learning Real-Time A*
- برای حل مشکل الگوریتم تپه نوردی در جستجوی محلی که ممکن است در ماکزیمم محلی گرفتار شود، می توان از الگوریتم LRTA* استفاده نمود.
- در LRTA* با افزودن حافظه، عامل را از ماکزیمم های محلی می رهاند



## الگوریتم LRTA*

- در ابتدا، مقدار تخمینی میزان فاصله هر حالت از حالت هدف، با مقدار تابع اکتشاف h(s)، مقداردهی می گردد. H(s)=h(s)
- بعد از انجام هر فعالیت، مقدار H(s) به روز می شود.