

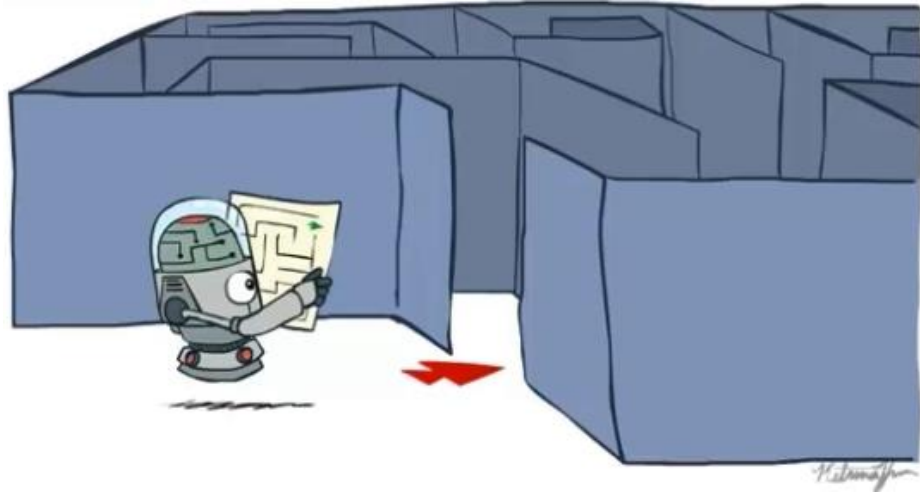
جستجوی آگاهانه

سید ناصر رضوی www.snrazavi.ir

۱۳۹۷

یادآوری: جستجو

۲



□ مسائل جستجو.

- حالت‌ها (بیانگر چیدمان محیط)
- عمل‌ها و هزینه‌ها
- تابع جانشین (بیانگر دینامیک محیط)
- حالت شروع و حالت هدف

□ درخت جستجو.

- گره‌ها: بیانگر دنباله عملیات لازم برای رسیدن به حالت‌ها

□ استراتژی‌های جستجو.

- ایجاد درخت جستجو به یک شیوه منظم
- اعمال یک ترتیب بر روی گره‌های برگ

یادآوری: الگوریتم عمومی جستجوی درختی

۳

```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of the problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
  end
```

□ ایده‌های مهم.

□ استراتژی گسترش دادن

□ ساختمان داده fringe

□ گسترش دادن گره‌ها

هر بار کدام گره درخت باید گسترش داده شود؟

یادآوری: جستجوی ناآگاهانه

۴

□ جستجوی ناآگاهانه.

□ کاوش درخت جستجو به یک روش از پیش تعریف شده و منظم بدون توجه به هدف.

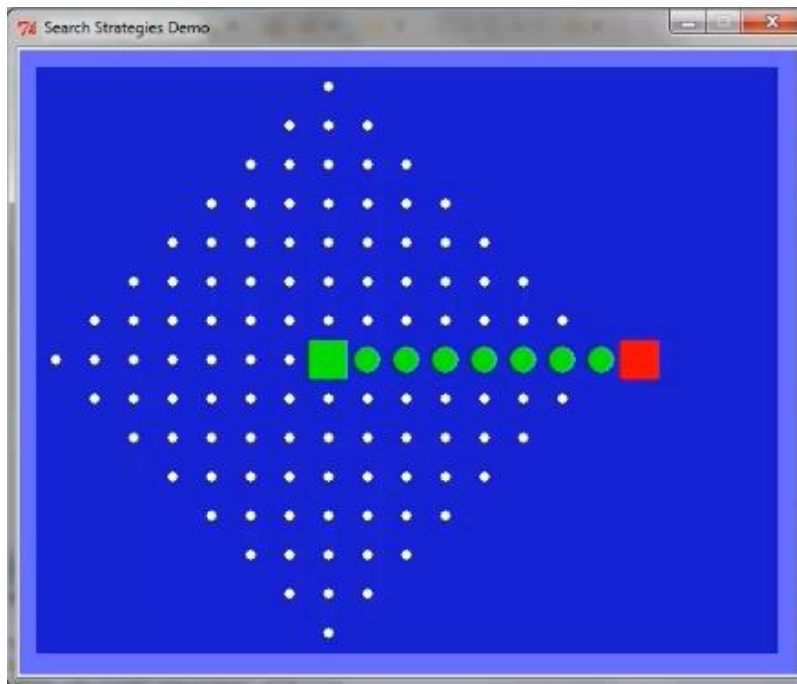


یادآوری: جستجوی ناآگاهانه

۵

□ جستجوی هزینه یکنواخت.

□ بررسی حالت‌ها در تمام جهت‌ها برای پیدا کردن راه‌حل بهینه.



یادآوری: پیچیدگی زمانی استراتژی‌های ناآگاهانه

۶

□ بهترین استراتژی ناآگاهانه (عمیق کننده تکراری)، دارای پیچیدگی زمانی **نمایی** است!!!

□ کامل؟ بله

[مانند جستجوی سطحی]

□ بهینه؟ بله

[مانند جستجوی سطحی]

ترکیب مزایای استراتژی‌های
سطحی و عمقی!

□ پیچیدگی زمانی؟ **نمایی**

[مانند جستجوی عمقی]

$$db^1 + (d-1)b^2 + (d-2)b^3 + \dots + 1b^d \in O(b^d)$$

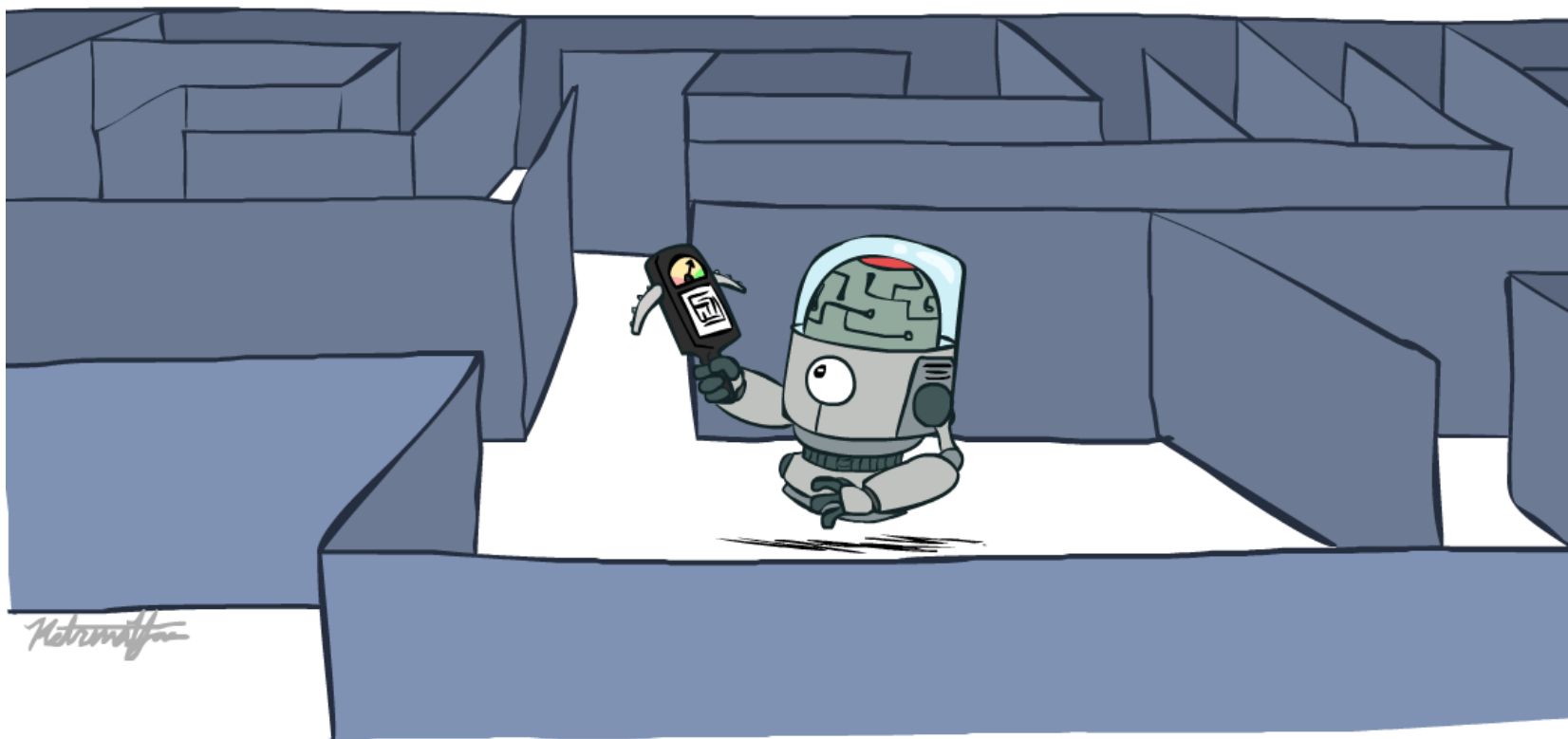
$$O(bd)$$

□ پیچیدگی حافظه؟ **خطی**

[مانند جستجوی عمقی]

جستجوی آگاهانه

۷



□ استراتژی‌های جستجوی آگاهانه.



□ هیوریستیک‌ها

□ جستجوی حریصانه

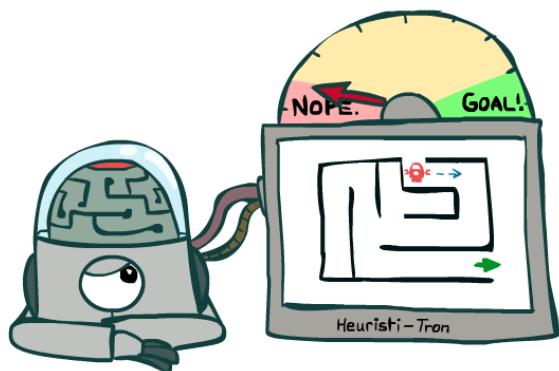
□ جستجوی A^*

□ کاهش مصرف حافظه A^*

□ ابداع توابع هیوریستیک

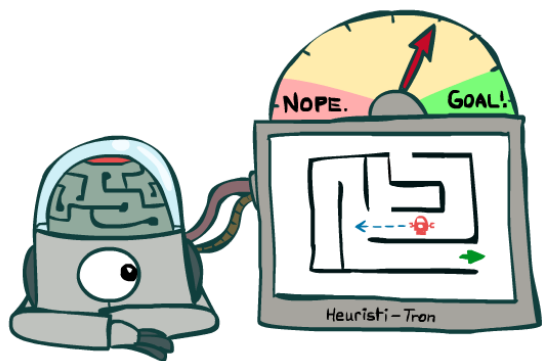
جستجوی آگاهانه

۹



□ جستجوی ناآگاهانه.

- تنها از اطلاعات موجود در تعریف مسئله استفاده می کند.
- تنها می تواند حالت های هدف را از حالت های غیرهدف تشخیص دهد.

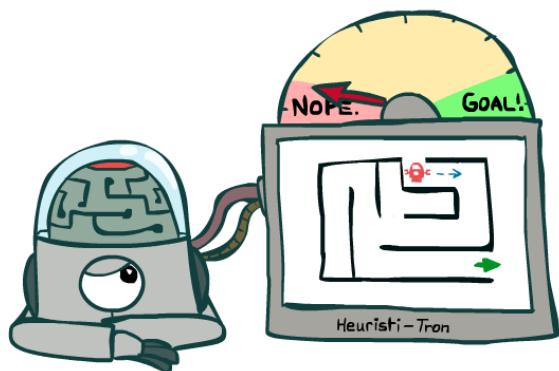


□ جستجوی آگاهانه.

- علاوه بر اطلاعات موجود در تعریف مسئله، از اطلاعات خاص مسئله بهره می برد.
- بین حالت های غیرهدف (بسته به فاصله آنها تا هدف) تمایز قائل می شود.

توابع هیوریستیک

۱۰

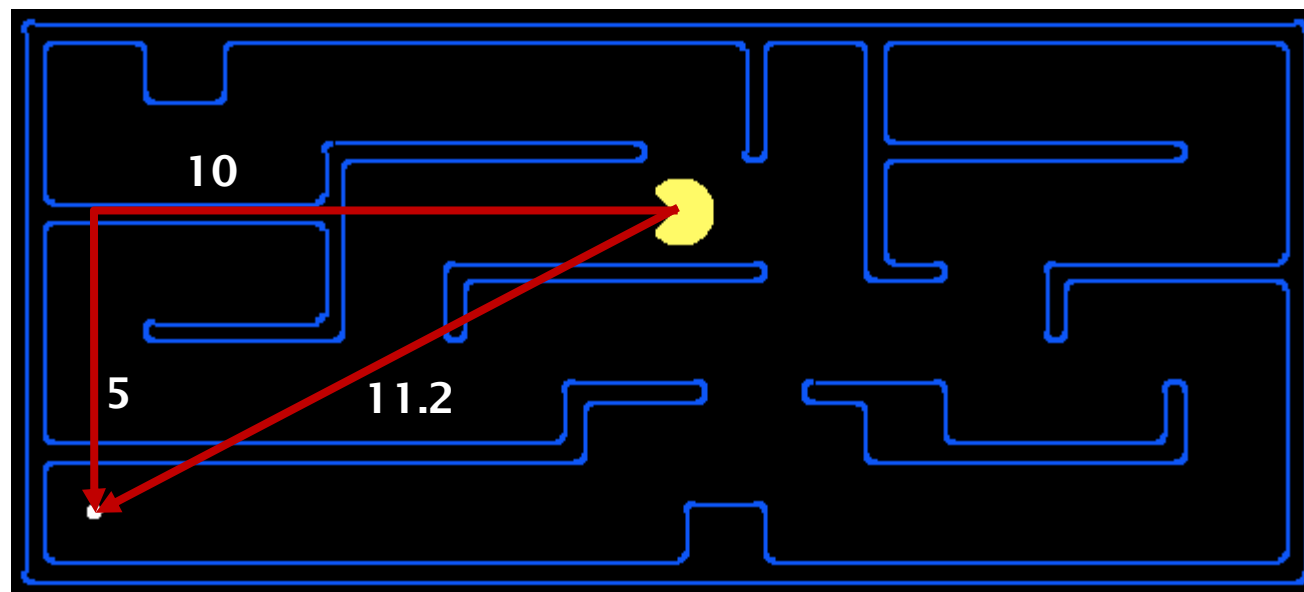


□ هیوریستیک.

□ یک تابع که فاصله هر حالت را تا هدف **تخمین** می‌زند.

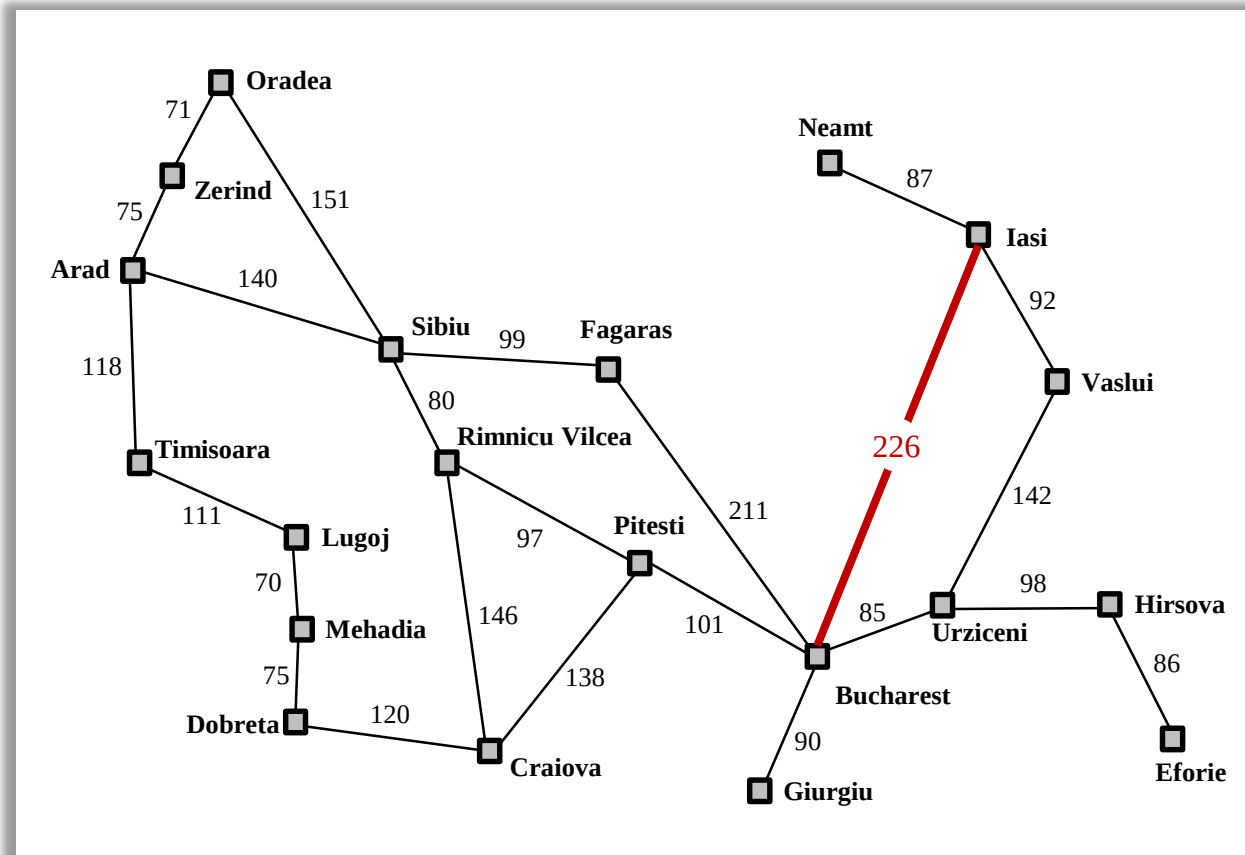
□ وابسته به مسئله است.

□ مثال: فاصله مانهاتانی و فاصله اقلیدسی در مسائل مسیریابی



توابع هیوریستیک: مثال

۱۱



فاصله مستقیم تا بخارست

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

جستجوی اول بهترین

۱۲

□ ایده.

ارزیابی؛ بیانگر میزان مطلوب بودن یک گره برای گسترش دادن

□ هر گره‌ای که تولید می‌شود را ارزیابی کن. [تابع ارزیابی]

□ هر بار بهترین گره گسترش نیافته را انتخاب و گسترش بده.

□ پیاده‌سازی. با استفاده از یک صف اولویت که در آن گره‌ها بر اساس میزان مطلوب بودنشان مرتب شده‌اند.

تابع ارزیابی



$$f(n) = h(n)$$

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

□ انواع خاص از جستجوی اول بهترین.

□ جستجوی حریصانه

□ جستجوی A^*

جستجوی مریضانه

۱۳



جستجوی حریصانه

۱۴

□ تابع ارزیابی.



$$f(n) = h(n)$$

↑
تابع ارزیابی

↑
تابع هیوریستیک

□ تابع هیوریستیک.

□ $h(n)$ = فاصله تخمینی از گره n تا هدف.

□ جستجوی حریصانه.

□ اول گره‌ای را گسترش بده که به نظر می‌رسد به هدف نزدیک‌تر است.

جستجوی حریصانه

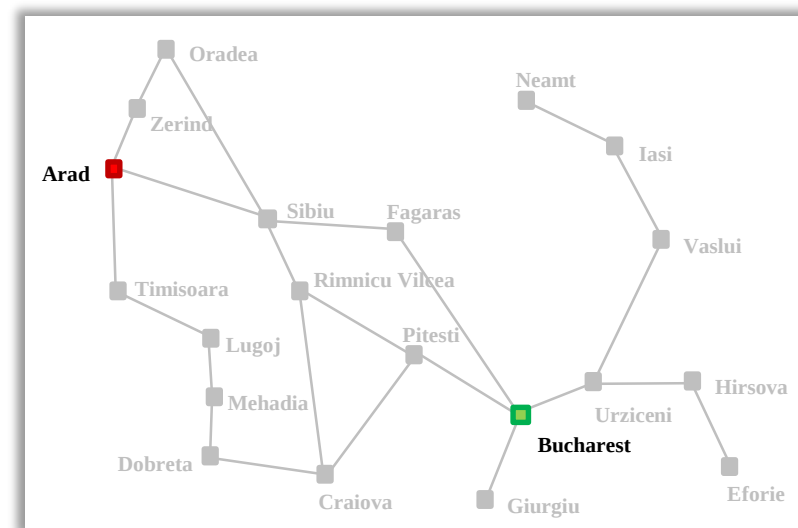
۱۵

□ جستجوی حریصانه. گره‌ای را گسترش بده که به نظر می‌رسد به هدف نزدیک‌تر است.



frontier:

[Arad (366)]



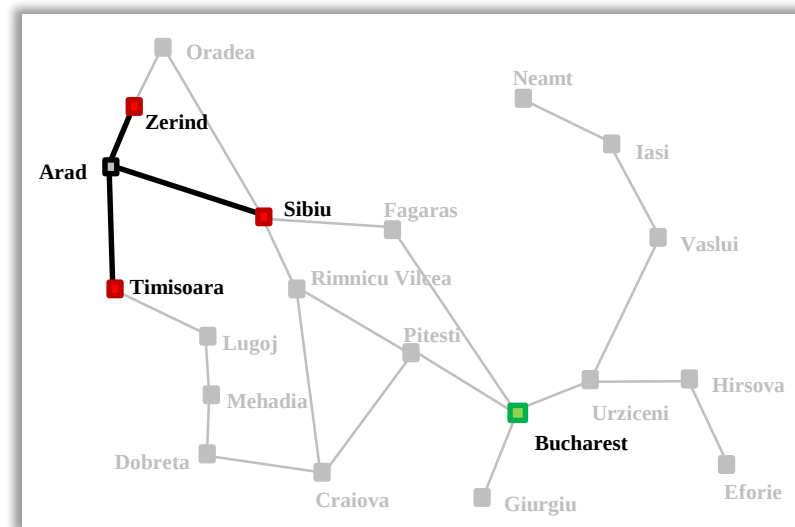
16

```

graph TD
    Arad([Arad]) --> Sibiu([Sibiu])
    Arad --> Timisoara([Timisoara])
    Arad --> Zerind([Zerind])
    Sibiu --- fSibiu[253]
    Timisoara --- fTimisoara[329]
    Zerind --- fZerind[374]

```

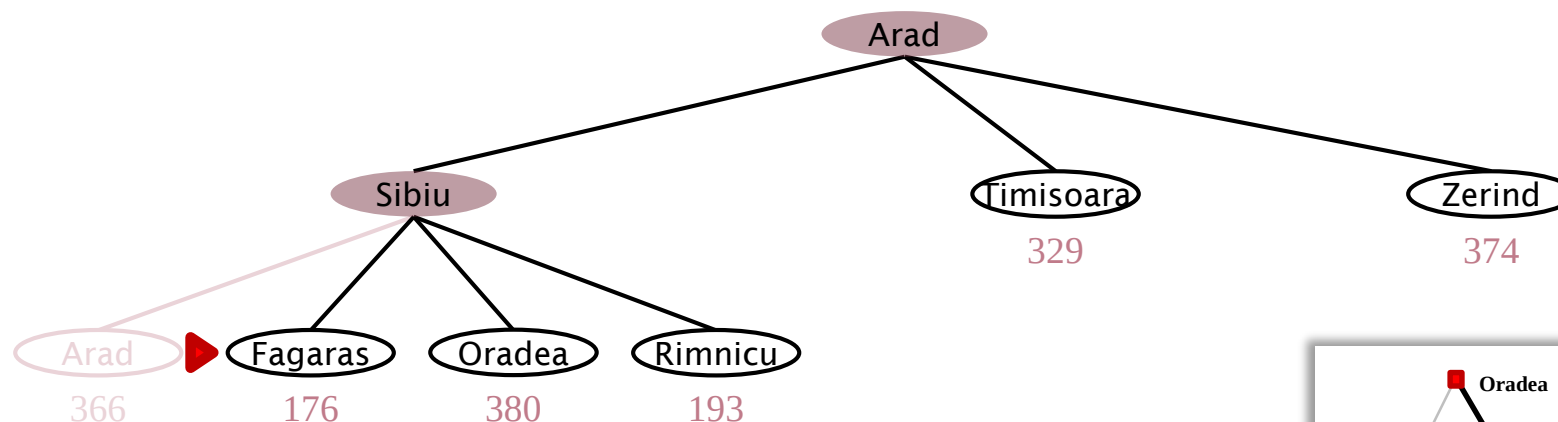
[Sibiu (253), Timisoara (329), Zerind (374)]



جستجوی حریصانه

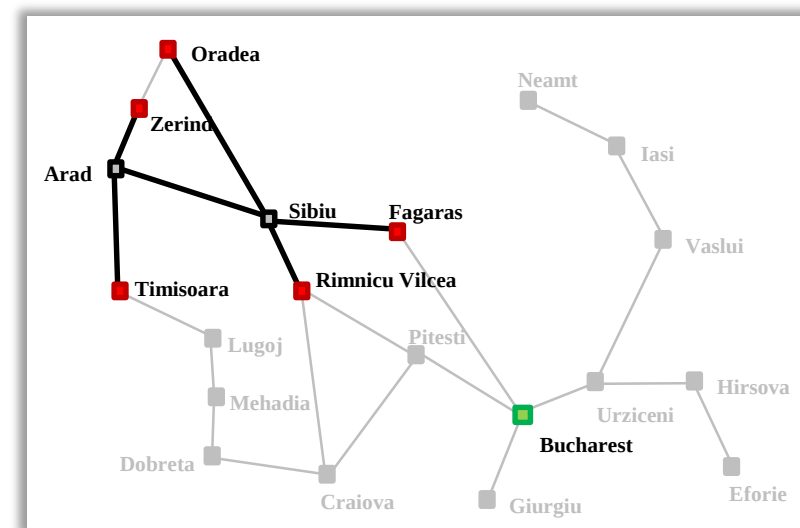
۱۷

□ جستجوی حریصانه. گره‌ای را گسترش بده که به نظر می‌رسد به هدف نزدیک‌تر است.



frontier:

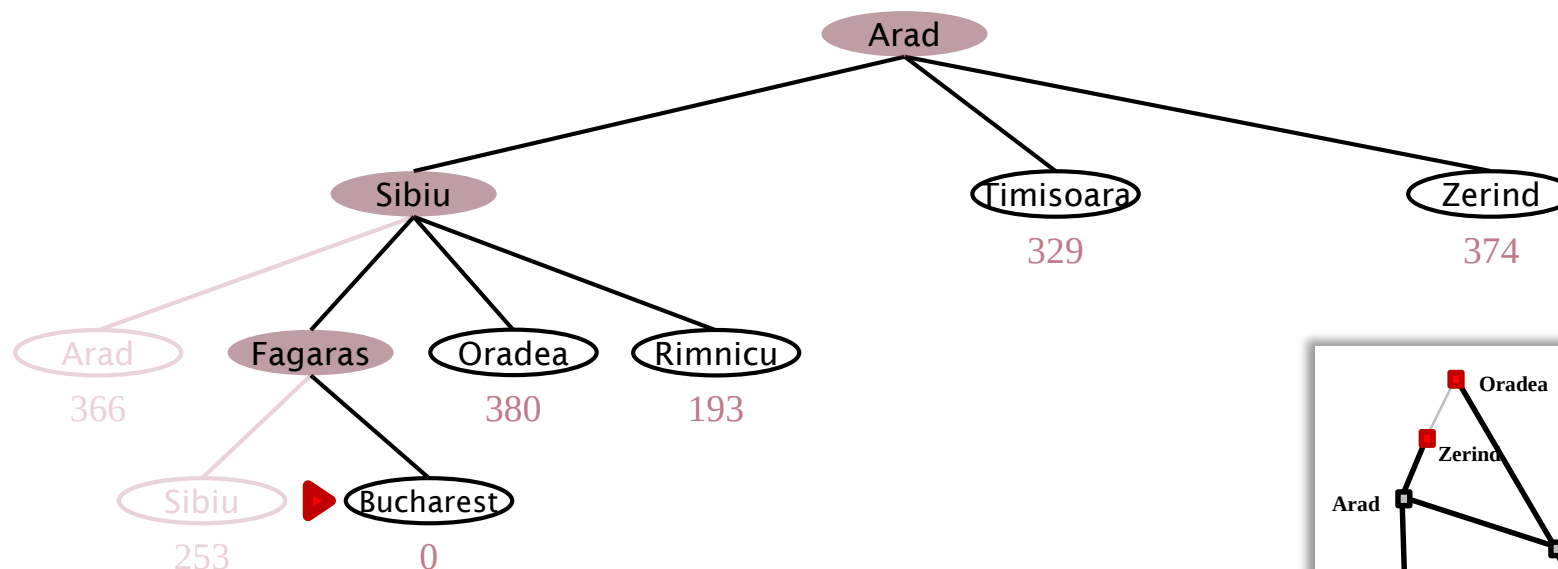
[Fagaras (176), Rimnicu (193), Timisoara (329), Zerind (374), Oradea (380)]



جستجوی حریصانه

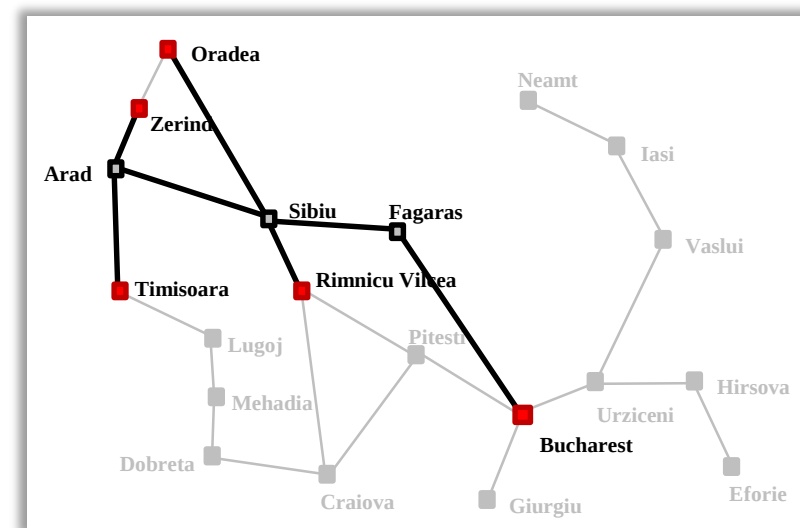
۱۸

□ جستجوی حریصانه. گره‌ای را گسترش بده که به نظر می‌رسد به هدف نزدیک‌تر است.



frontier:

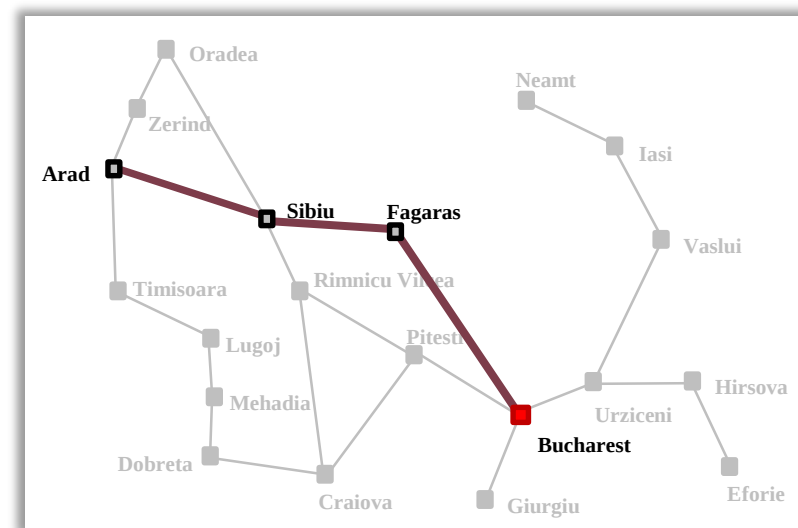
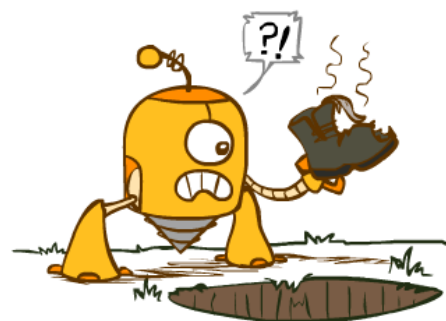
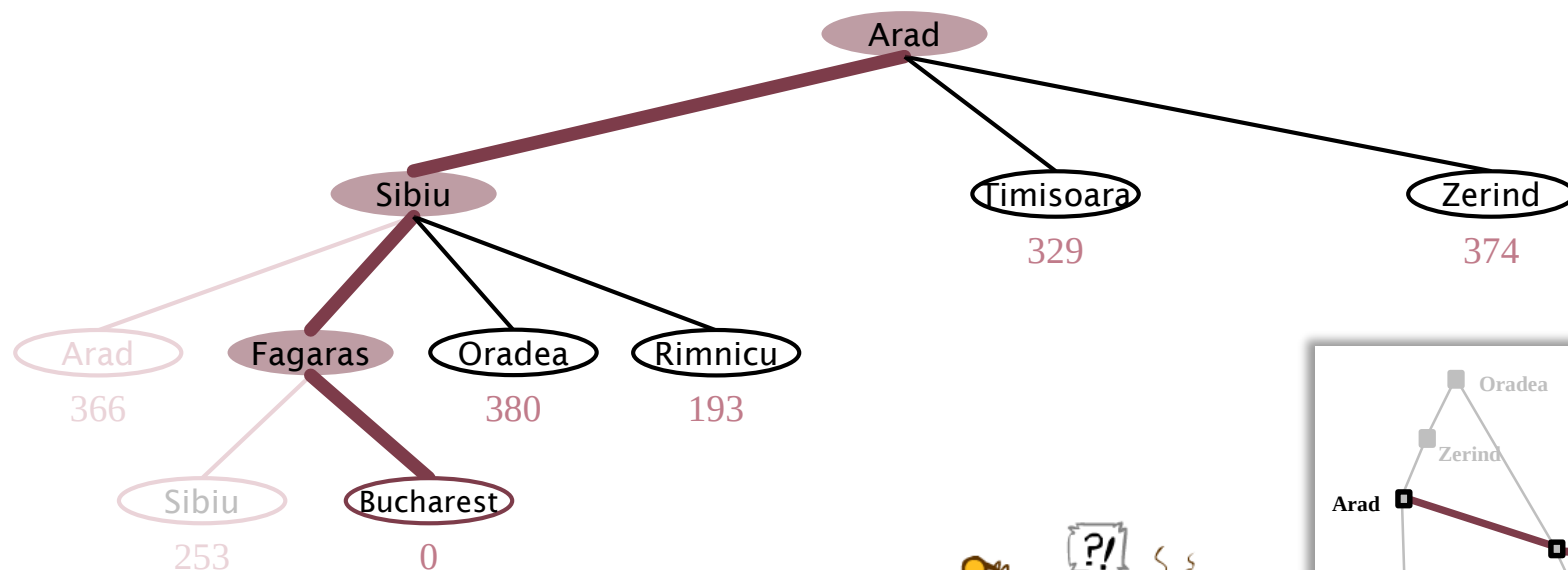
[Bucharest (0), Rimnicu (193), Timisoara (329), Zerind (374), Oradea (380)]



جستجوی حریصانه

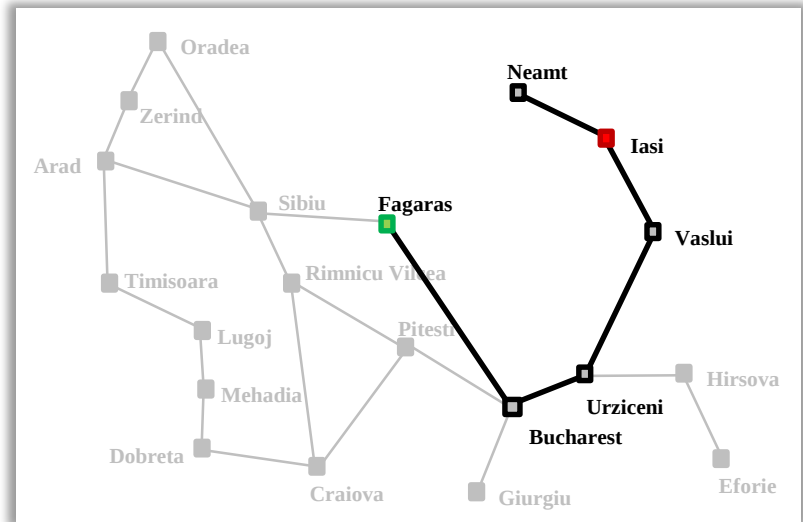
۱۹

□ جستجوی حریصانه. گره‌ای را گسترش بده که به نظر می‌رسد به هدف نزدیک‌تر است.



ارزیابی جستجوی حریصانه

۲۰



□ کامل؟ خیر [ممکن است در یک حلقه بی نهایت گیر کند]

□ مثال: جستجو از یاش به فگره

□ بهینه؟ خیر [مثال رومانی]

$$b^1 + b^2 + b^3 + \dots + b^m \in O(b^m)$$

$$O(b^m)$$

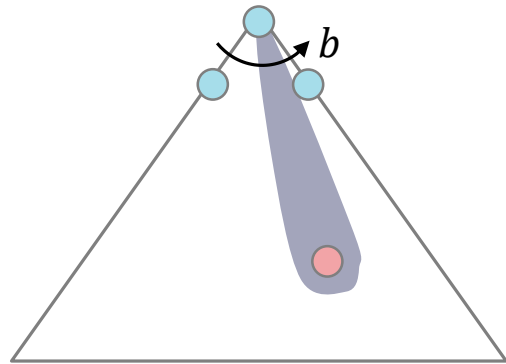
□ پیچیدگی زمانی؟ نمایی [مانند جستجوی عمقی]

□ پیچیدگی حافظه؟ نمایی

جستجوی حریصانه: تحلیل کارایی

۲۱

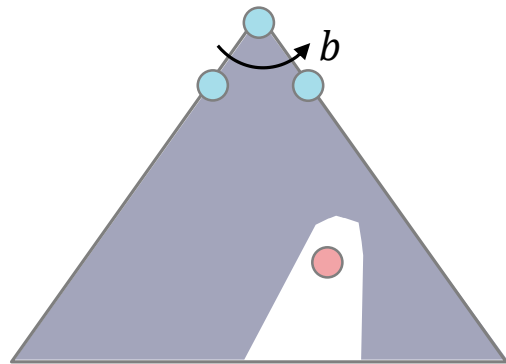
□ جستجوی حریصانه. گره‌ای را گسترش بده که به نظر می‌رسد به هدف نزدیک‌تر است.



□ تحلیل بهترین حالت. [زمان و حافظه خطی]

□ استراتژی حریصانه در بهترین حالت، مسیر جستجو را مستقیماً به سمت یک حالت هدف (که معمولاً بهینه نیست) هدایت می‌کند.

□ در بهترین حالت، اولین مسیر بررسی شده شامل گره هدف است.

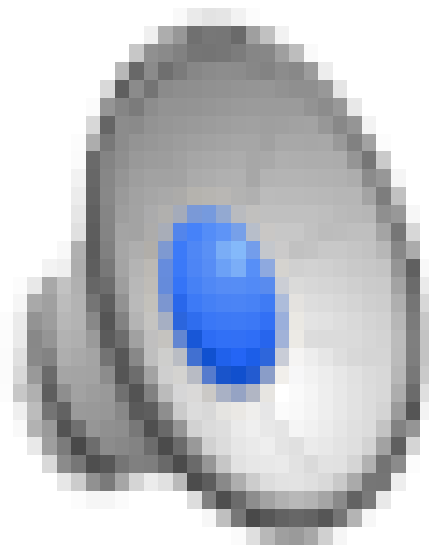


□ تحلیل بدترین حالت. [زمان و حافظه نمایی]

□ استراتژی حریصانه در بدترین حالت همانند یک جستجوی عمقی که خیلی بد هدایت شده است، عمل می‌کند.

□ در بدترین حالت، آخرین مسیر بررسی شده شامل گره هدف است.

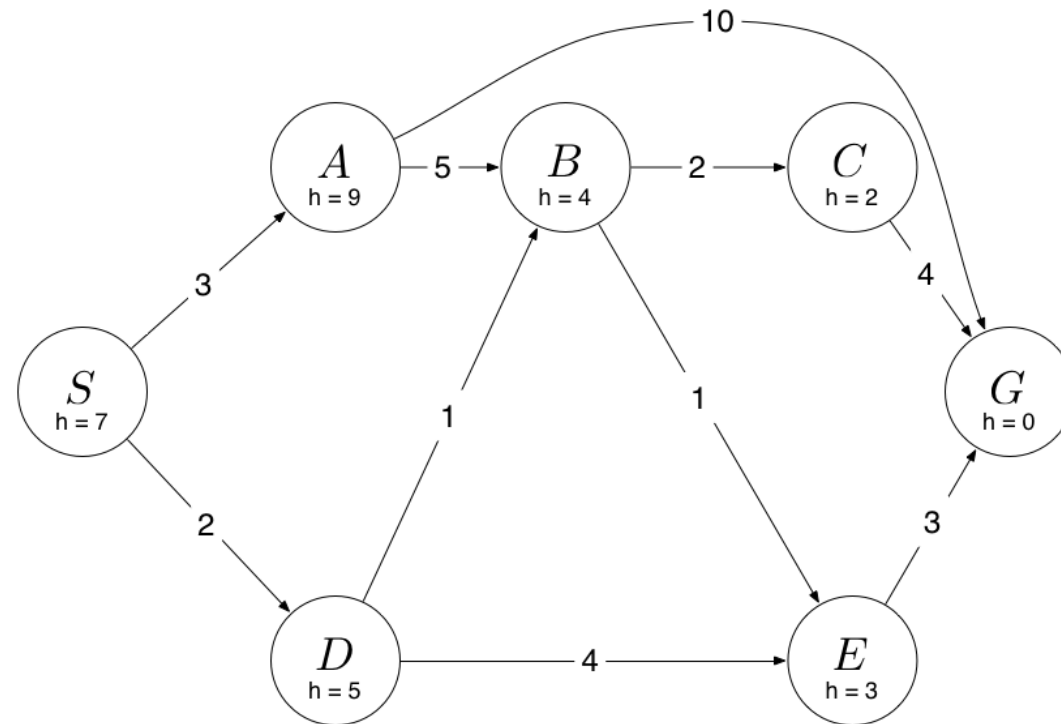
جستجوی مریضانه: اجرای نمایشی



پرسش کلاسی: اجرای جستجوی حریصانه

۲۳

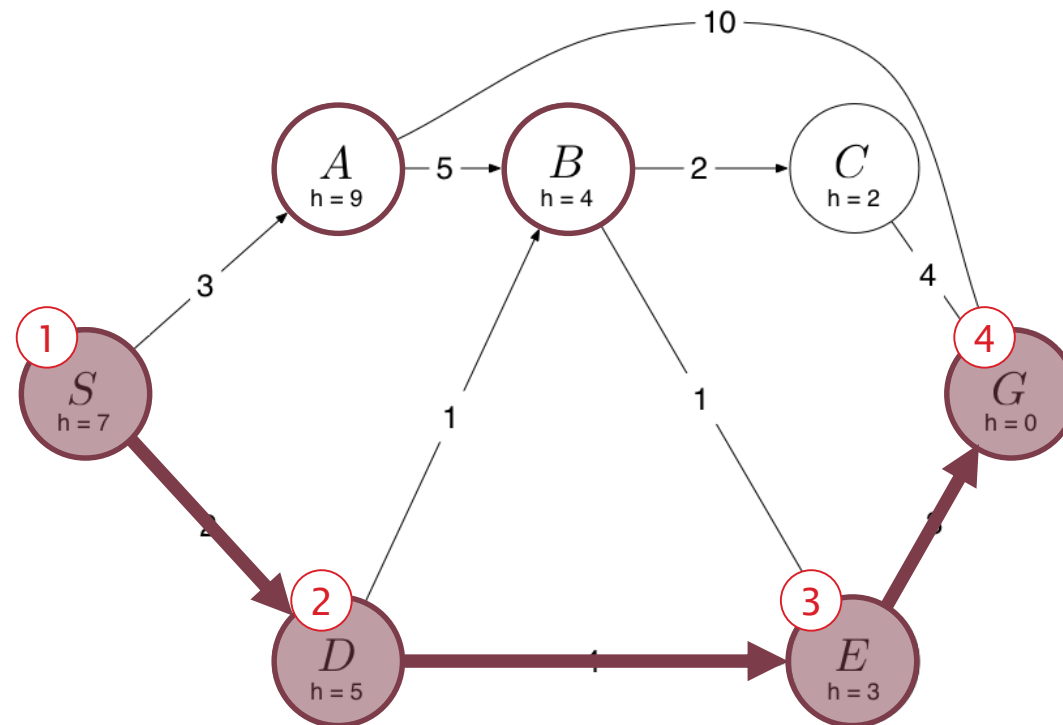
□ س. اگر بر روی گراف زیر از جستجوی حریصانه استفاده شود، مسیر یافته شده کدام است؟



پرسش کلاسی: اجرای جستجوی حریصانه

۲۴

□ س. اگر بر روی گراف زیر از جستجوی حریصانه استفاده شود، مسیر یافته شده کدام است؟



جستجوی A^*

۲۵

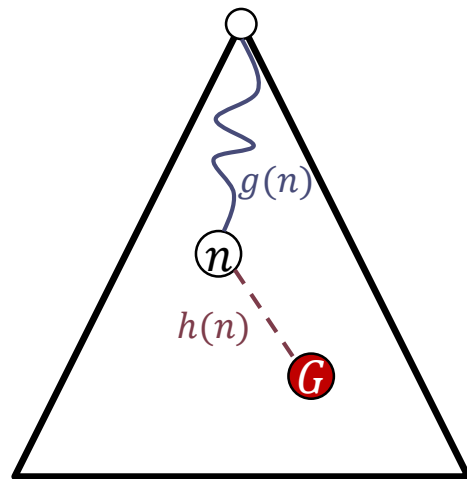


□ ایده. از ادامه دادن مسیرهایی که می‌دانی پرهزینه هستند، اجتناب کن.

□ جستجوی A^* . ترکیب مزایای جستجوی هزینه یکنواخت و جستجوی حریصانه.

□ جستجوی هزینه یکنواخت: کامل و بهینه است، اما می‌تواند بسیار زمان‌بر و حافظه‌بر باشد.

□ جستجوی حریصانه: نه کامل است و نه بهینه، اما می‌تواند بسیار سریع و کم حافظه باشد.



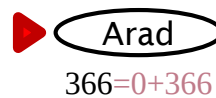
□ تابع ارزیابی.

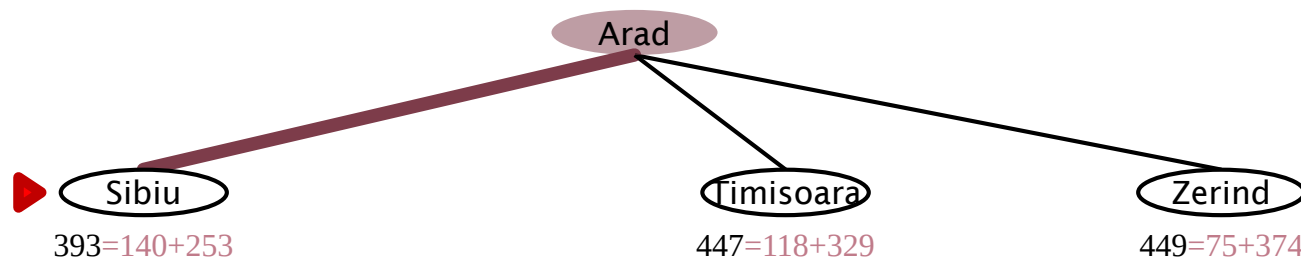
□ $g(n)$: هزینه واقعی مسیر پیموده شده از ریشه تا گره n .

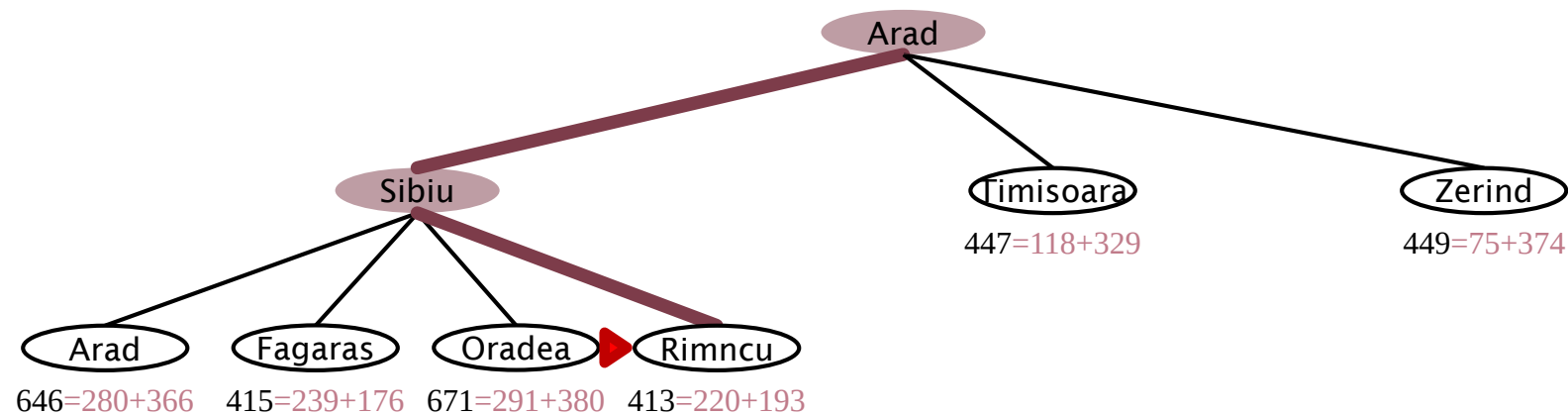
□ $h(n)$: هزینه تخمینی کوتاه‌ترین مسیر از گره n تا هدف.

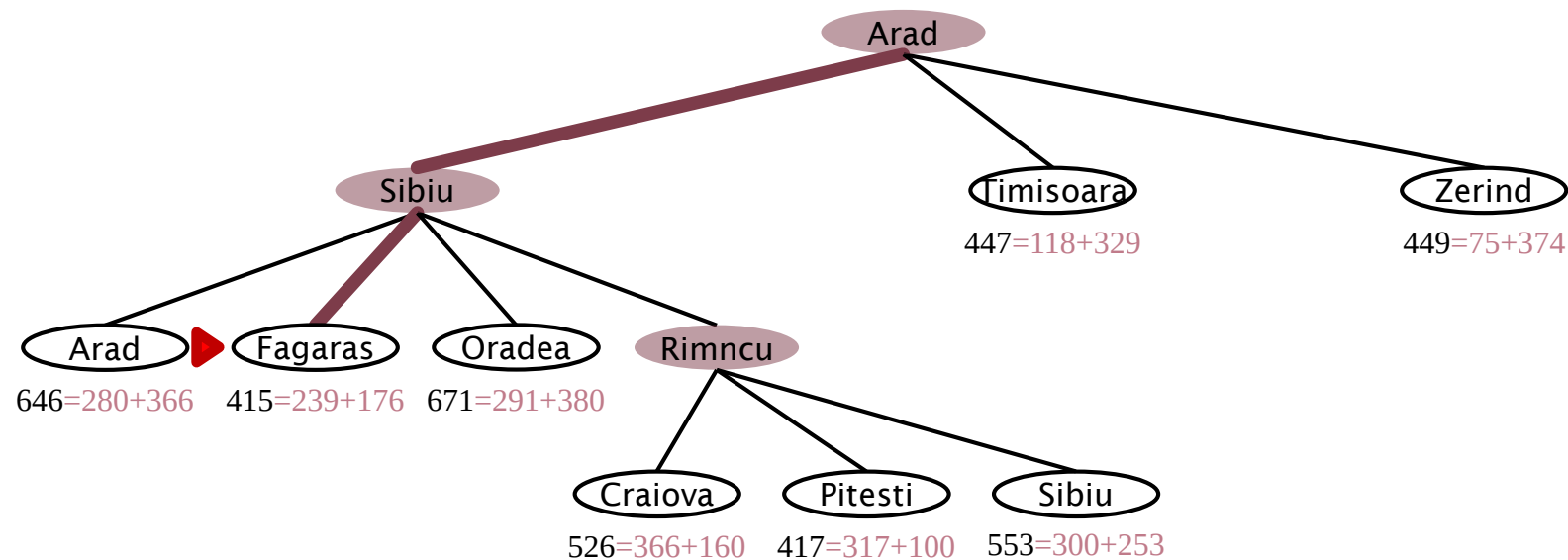
□ $f(n)$: هزینه تخمینی کوتاه‌ترین مسیر راه‌حلی که از گره n می‌گذرد.

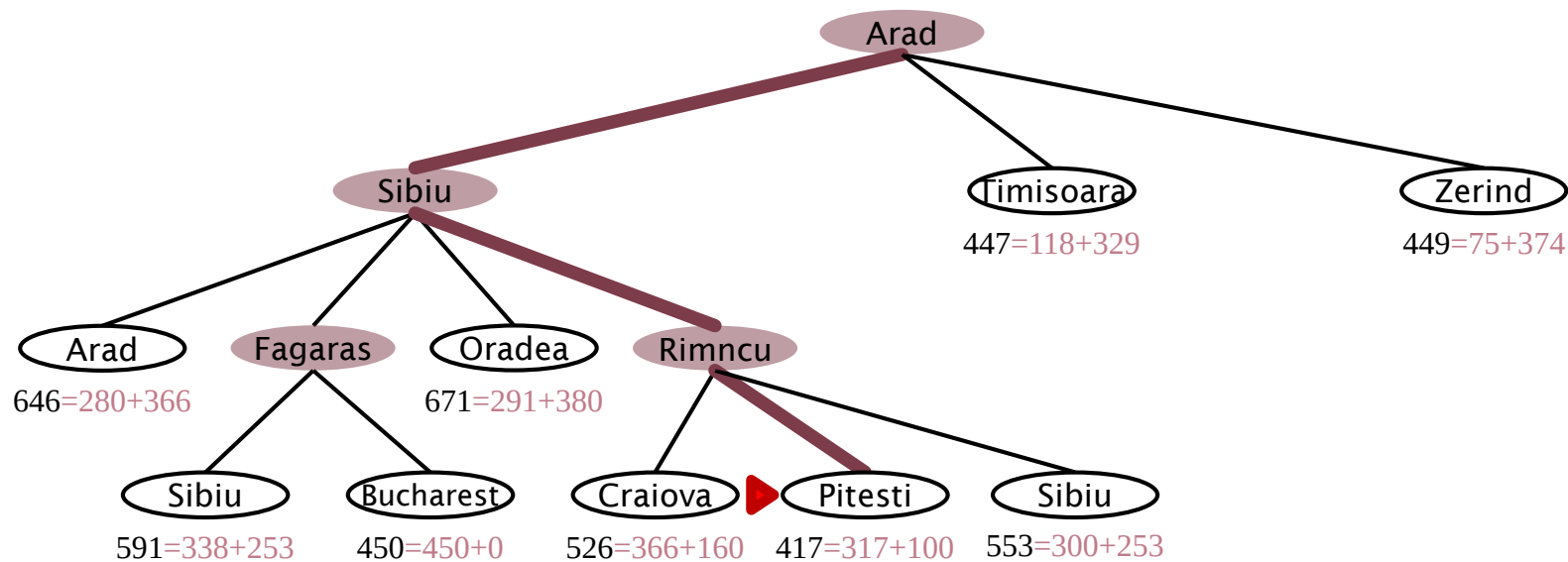
$$f(n) = g(n) + h(n)$$

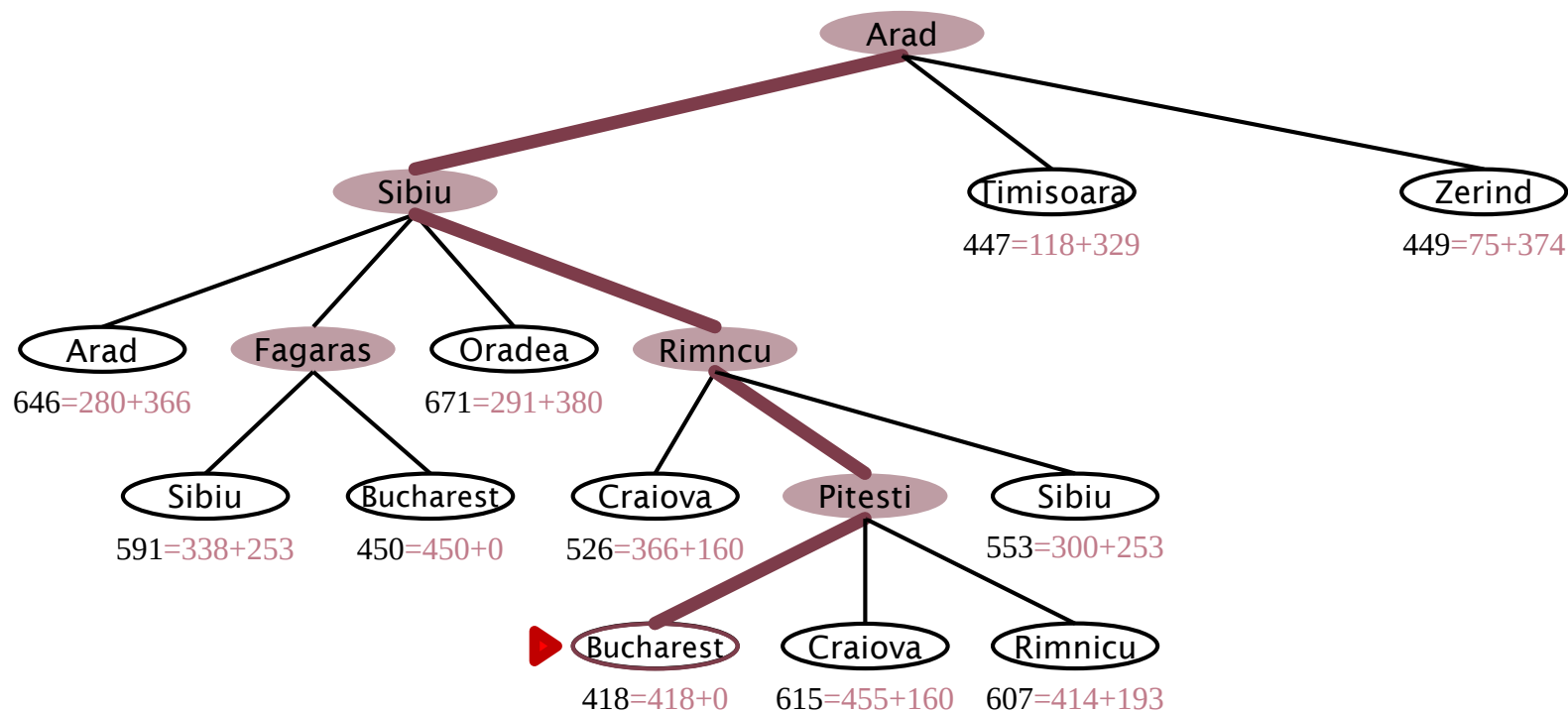






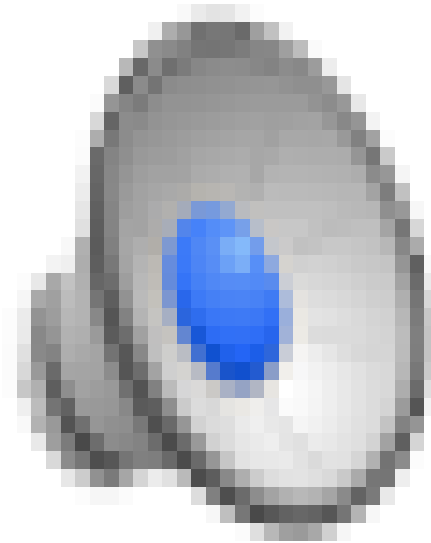






جستجوی A^* : اجرای نمایشی

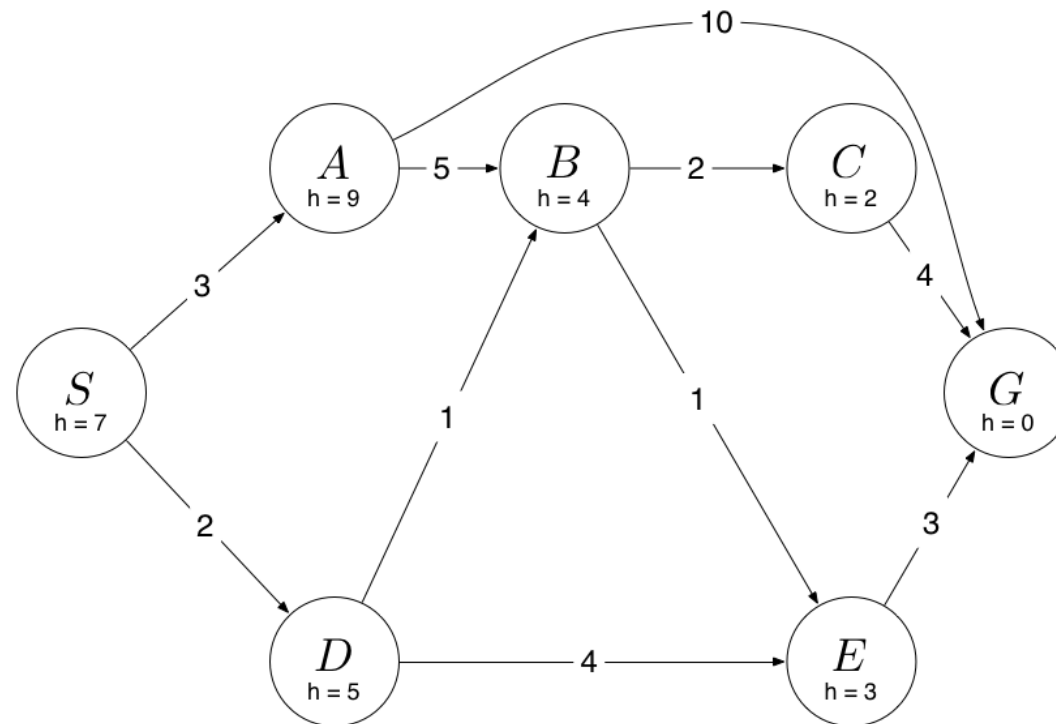
۳۳



پرسش کلاسی: اجرای جستجوی A^*

۳۴

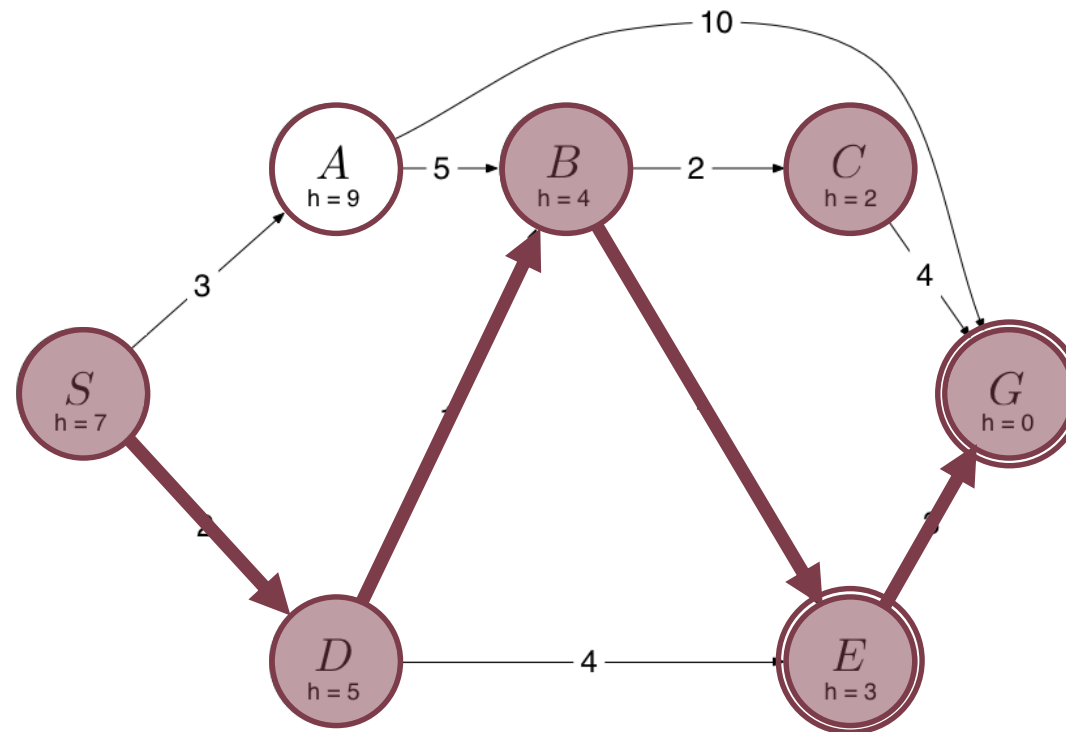
□ س. اگر بر روی گراف زیر از جستجوی درختی A^* استفاده شود، مسیر یافته شده کدام است؟



پرسش کلاسی: اجرای جستجوی A^*

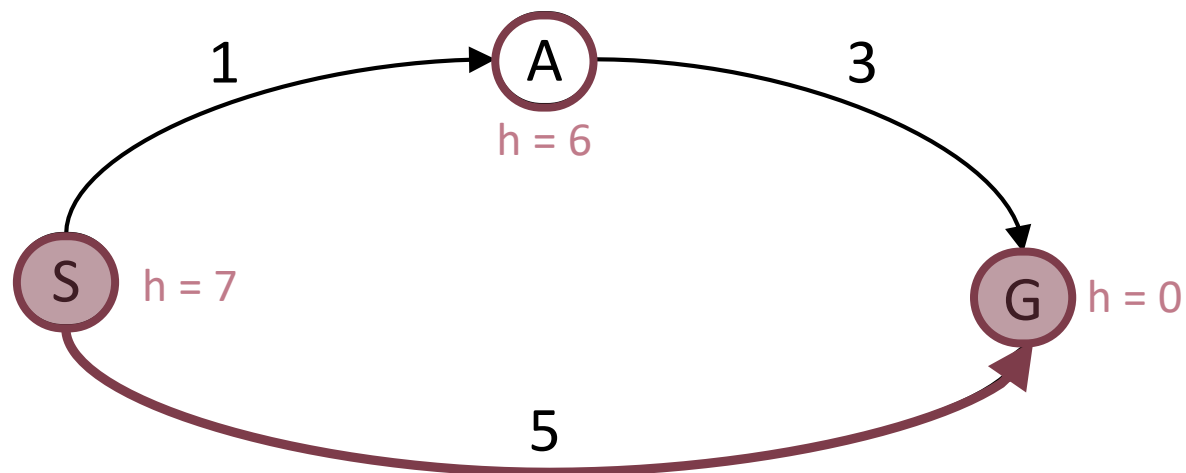
۳۵

□ س. اگر بر روی گراف زیر از جستجوی درختی A^* استفاده شود، مسیر یافته شده کدام است؟



بررسی بهینگی A^*

۳۶



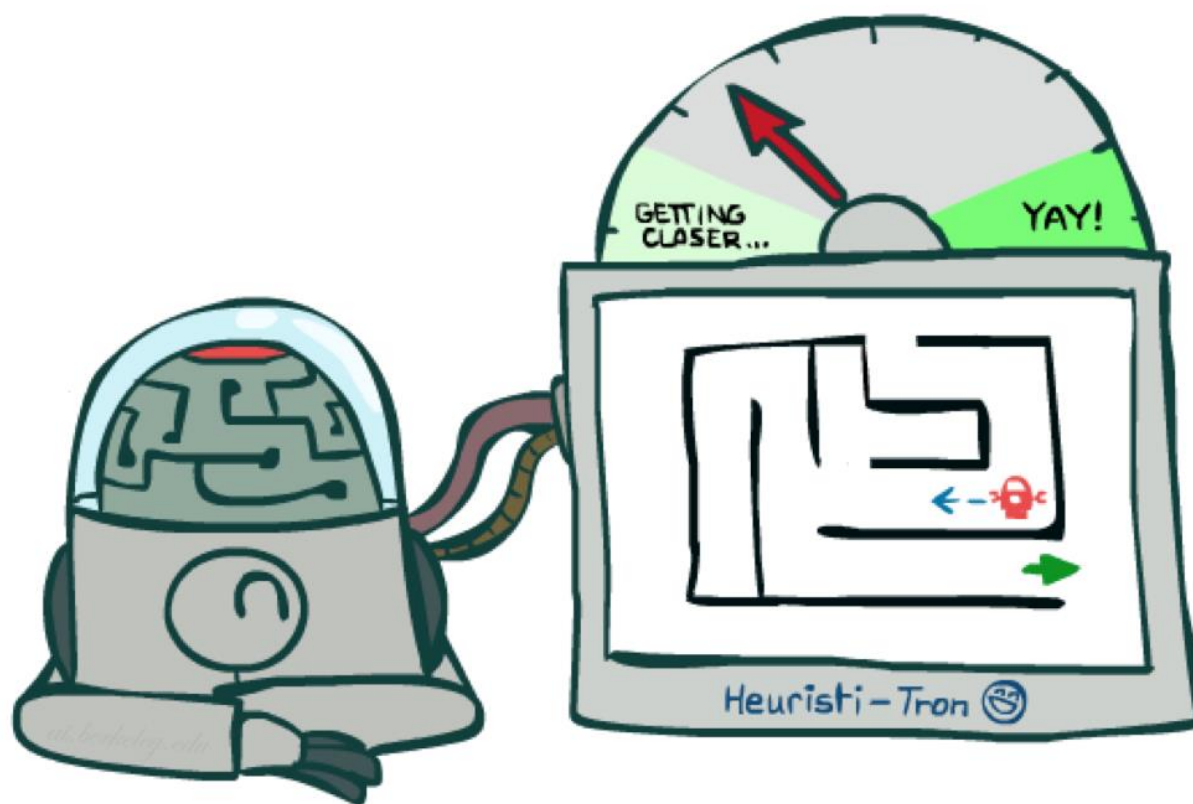
□ آیا A^* در گراف بالا مسیر بهینه را برمی گرداند؟ ایراد کار در کجاست؟

□ هزینه واقعی مسیر بد $>$ هزینه تخمینی مسیر خوب

□ بنابراین تخمین ها باید از مقدار واقعی کمتر باشند.

هیوریستیک‌های قابل قبول

۳۷



بررسی بهینگی A^*

۳۸

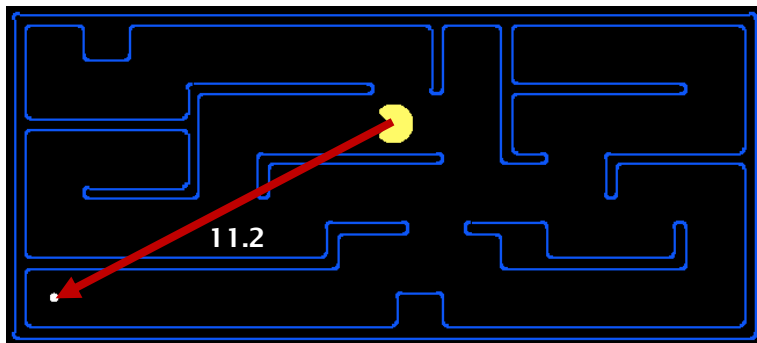
□ آیا جستجوی A^* بهینه است؟

□ بله، به شرطی که تابع هیوریستیک **قابل قبول** باشد.

□ هیوریستیک **قابل قبول**. تابع هیوریستیک $h(n)$ قابل قبول است اگر به ازای هر گره مانند n همواره داشته باشیم:

$$0 \leq h(n) \leq h^*(n)$$

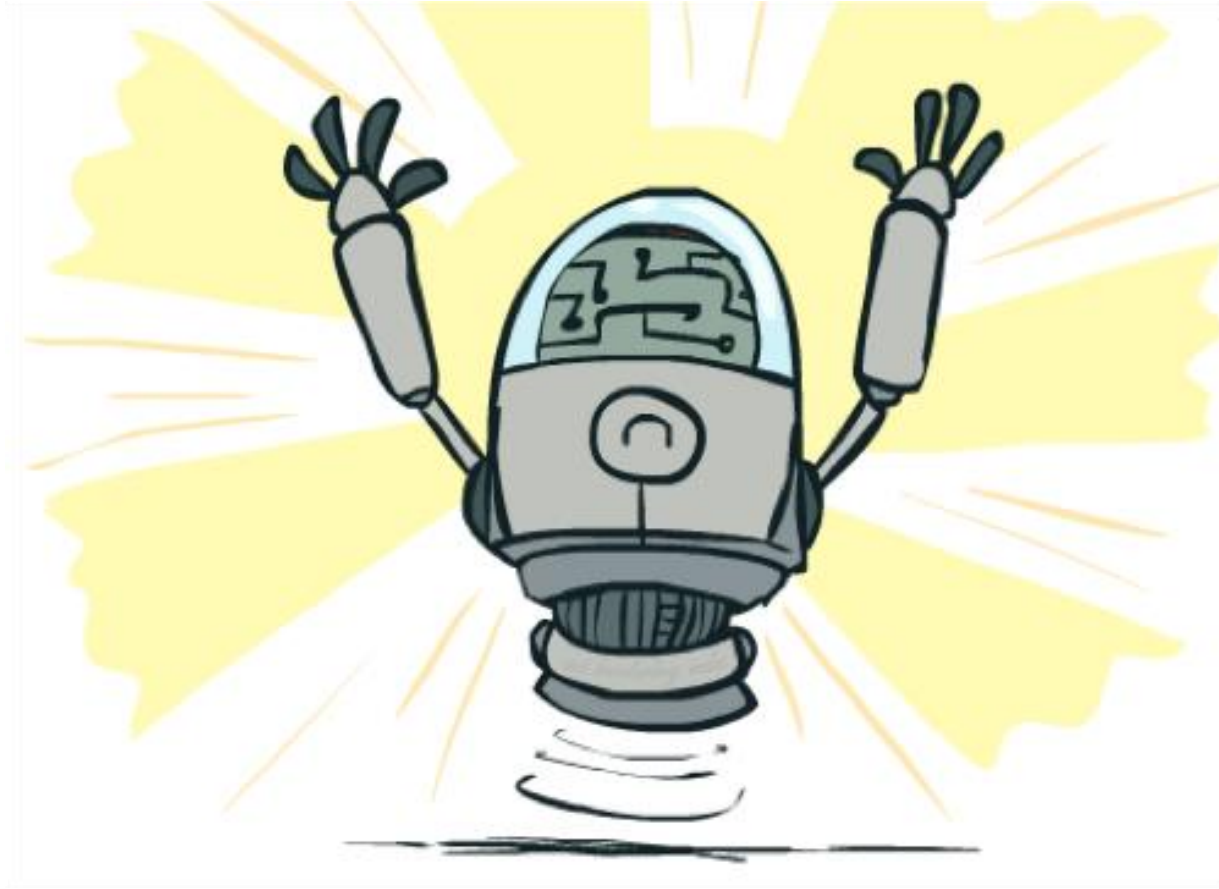
به طوری که $h^*(n)$ بیانگر هزینه واقعی کوتاه‌ترین مسیر از n تا هدف است.



□ مثال. هیوریستیک فاصله مستقیم در مسائل مسیریابی.

اثبات بهینگی A^* در جستجوی درختی

۳۹



اثبات بهینگی A^* در جستجوی درختی

۴۰

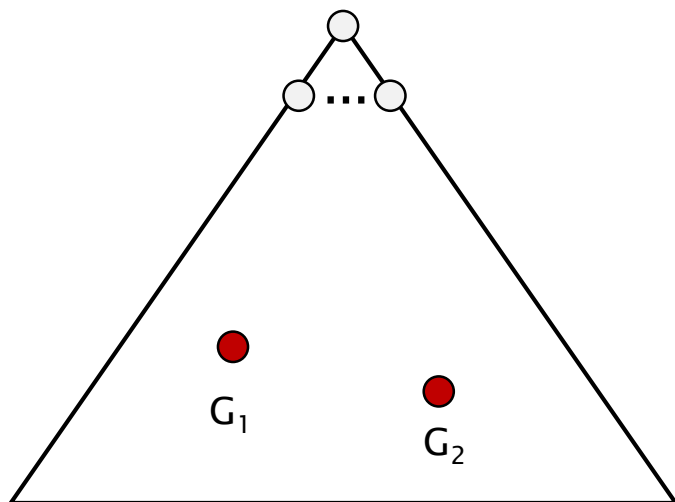
□ قضیه. اگر تابع هیوریستیک قابل قبول باشد، آنگاه A^* در جستجوی درختی بهینه است.

□ فرضیات.

□ گره G_1 یک گره هدف بهینه است.

□ گره G_2 یک گره هدف زیربهینه است.

□ هیوریستیک h قابل قبول است.



□ ادعا. گره G_1 قبل از گره G_2 از صف اولویت برداشته می شود.

اثبات بهینگی A^* در جستجوی درختی

۴۱

□ **قضیه.** اگر تابع هیوریستیک قابل قبول باشد، آنگاه A^* در جستجوی درختی بهینه است.

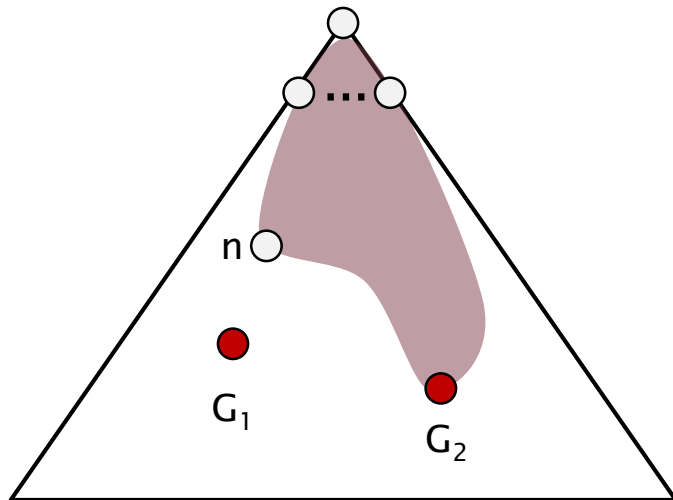
□ **اثبات.**

□ فرض کنید G_2 در صف قرار دارد.

□ یکی از اجداد G_1 نیز در صف قرار دارد. (گره n)

□ ادعا. گره n قبل از G_2 گسترش می‌یابد:

■ $f(n)$ کوچک‌تر یا مساوی $f(G_1)$ است.



$$\begin{aligned}
 f(n) &= g(n) + h(n) \\
 f(G_1) &= g(G_1) + h(G_1) = g(G_1) \\
 &= g(n) + h^*(n)
 \end{aligned}$$

صفر



$$\underbrace{f(n)}_{g(n) + h(n)} \leq \underbrace{f(G_1)}_{g(n) + h^*(n)}$$

اثبات بهینگی A^* در جستجوی درختی

۴۲

□ **قضیه.** اگر تابع هیوریستیک قابل قبول باشد، آنگاه A^* در جستجوی درختی بهینه است.

□ **اثبات.**

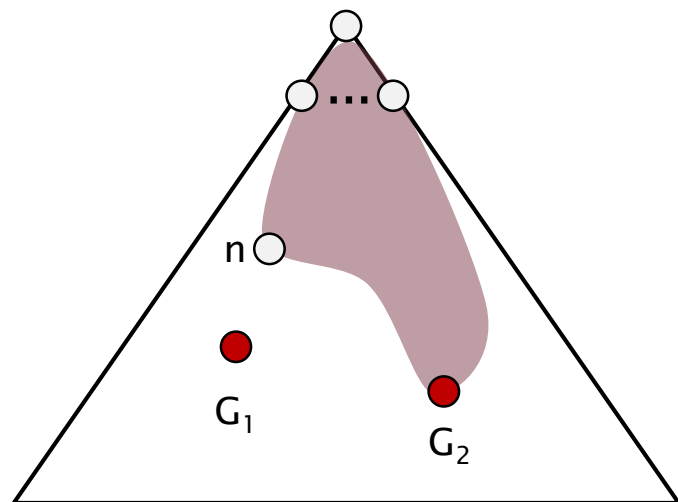
□ فرض کنید G_2 در صف قرار دارد.

□ یکی از اجداد G_1 نیز در صف قرار دارد. (گره n)

□ ادعا. گره n قبل از G_2 گسترش می‌یابد:

■ $f(n)$ کوچک‌تر یا مساوی $f(G_1)$ است.

■ $f(G_1)$ کوچک‌تر از $f(G_2)$ است.



$$g(G_1) < g(G_2)$$

$$g(G_1) + h(G_1) < g(G_2) + h(G_2)$$

$$f(G_1) < f(G_2)$$

اثبات بهینگی A^* در جستجوی درختی

۴۳

□ قضیه. اگر تابع هیوریستیک قابل قبول باشد، آنگاه A^* در جستجوی درختی بهینه است.

□ اثبات.

□ فرض کنید G_2 در صف قرار دارد.

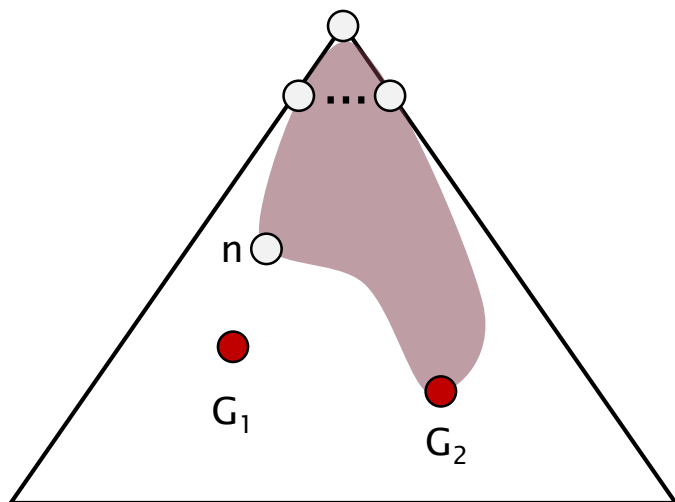
□ یکی از اجداد G_1 نیز در صف قرار دارد. (گره n)

□ ادعا. گره n قبل از G_2 گسترش می‌یابد:

■ $f(n)$ کوچک‌تر یا مساوی $f(G_1)$ است.

■ $f(G_1)$ کوچک‌تر از $f(G_2)$ است.

■ گره n قبل از G_2 گسترش می‌یابد.



$$f(n) \leq f(G_1) < f(G_2)$$

اثبات بهینگی A^* در جستجوی درختی

۴۴

□ قضیه. اگر تابع هیوریستیک قابل قبول باشد، آنگاه A^* در جستجوی درختی بهینه است.

□ اثبات.

□ فرض کنید G_2 در صف قرار دارد.

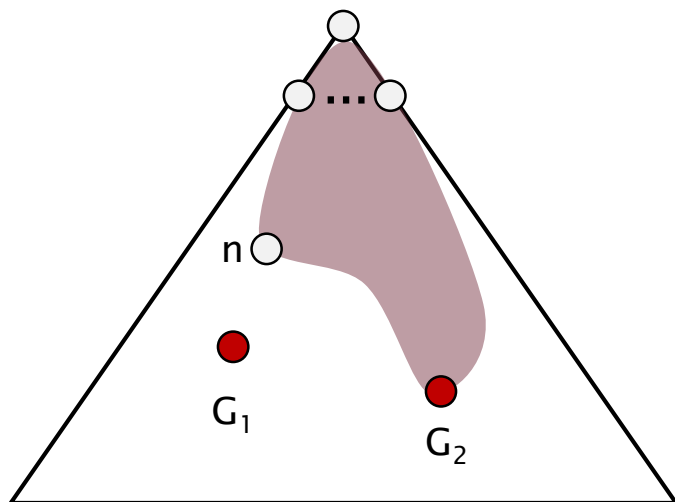
□ یکی از اجداد G_1 نیز در صف قرار دارد. (گره n)

□ ادعا. گره n قبل از G_2 گسترش می‌یابد:

■ $f(n)$ کوچک‌تر یا مساوی $f(G_1)$ است.

■ $f(G_1)$ کوچک‌تر از $f(G_2)$ است.

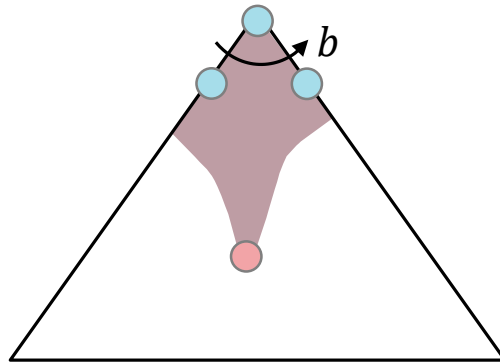
■ گره n قبل از G_2 گسترش می‌یابد.



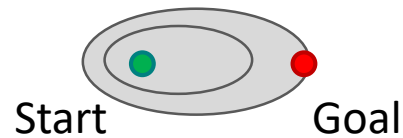
□ بنابراین تمام اجداد G_1 از جمله خود آن، قبل از G_2 گسترش می‌یابند.

ویژگی‌های A^*

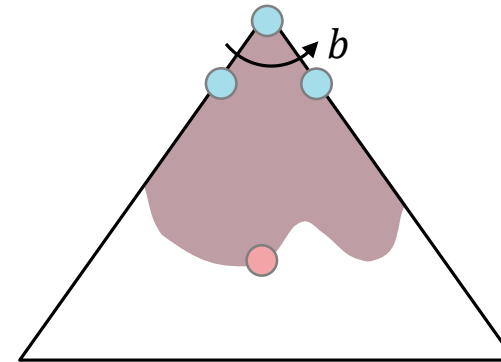
۴۵



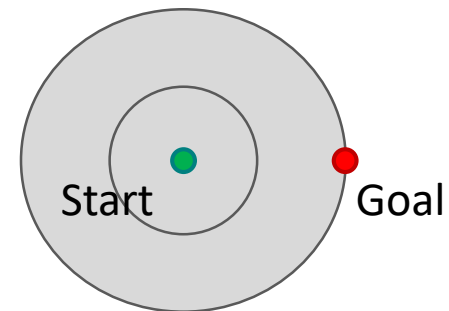
جستجوی A^*



جستجوی A^* عمدتاً گره‌ها را در جهت هدف گسترش می‌دهد، اما برای تفهیمین بهینگی گاهی اوقات نیز به سمت اطراف حرکت می‌کند.



جستجوی هزینه یکنواخت



جستجوی هزینه یکنواخت بدون توجه به هدف، در همه جهات‌ها به طور یکسان جستجو می‌کند.

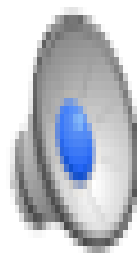
جستجوی A^*



جستجوی هزینه یکنواخت



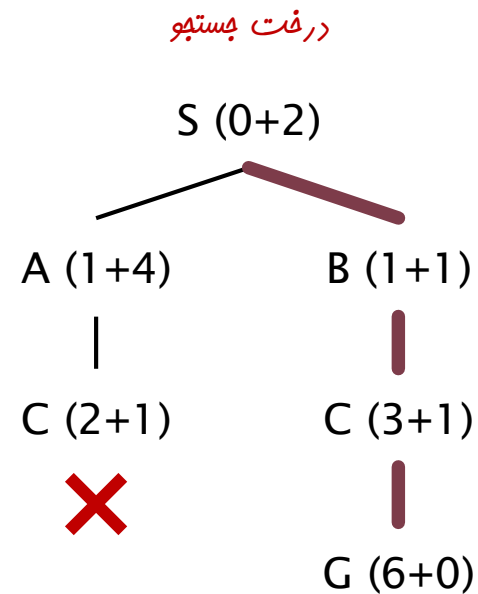
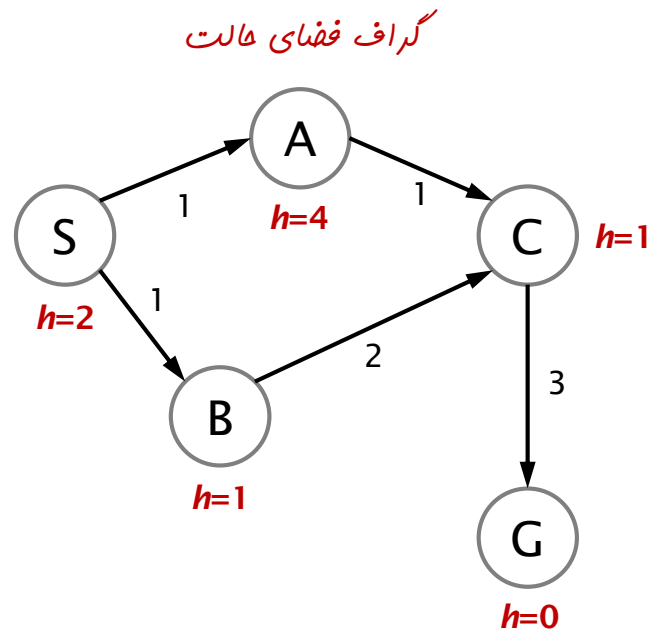
جستجوی حریصانه



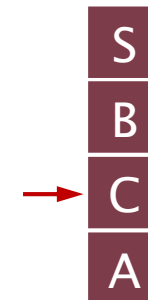
بهینگی A^* در جستجوی گرافی

۴۷

□ مثال. یافتن مسیر از S به G.



حالت‌های
گسترش یافته



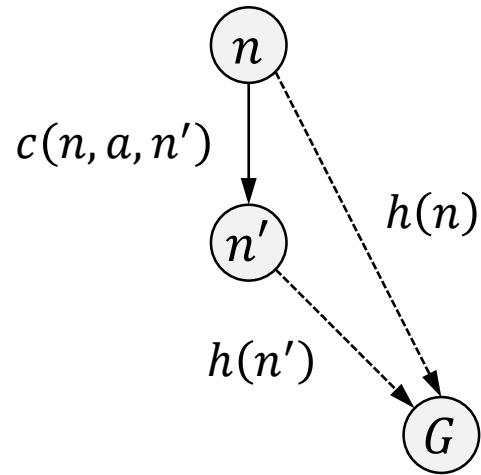
توجه. در این مثال اگرچه تابع هیوریستیک قابل قبول است، اما A^* قادر به یافتن راه حل بهینه نیست.

توابع هیوریستیک سازگار

۴۸

□ هیوریستیک سازگار. یک هیوریستیک سازگار است اگر:

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



□ یکنوایی. اگر h سازگار باشد، آنگاه f در طول هر مسیری غیرکاهشی است.

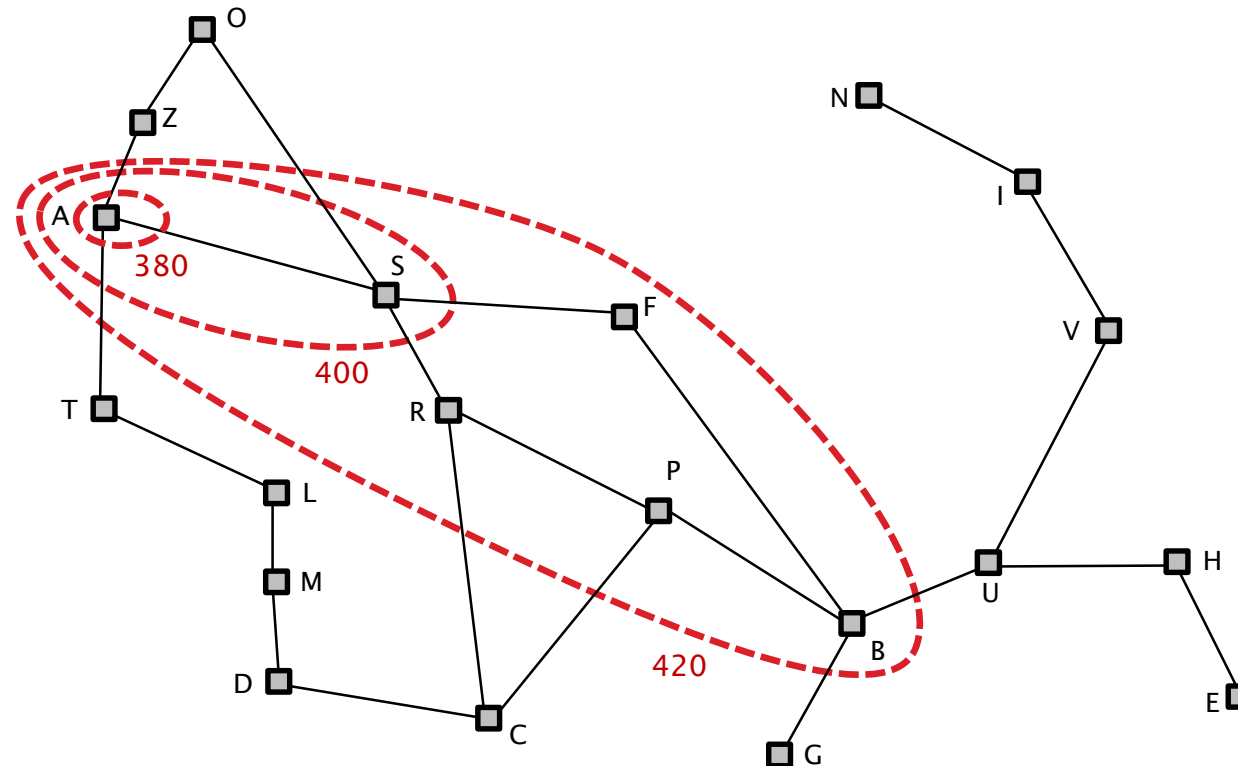
$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

□ قضیه. اگر h سازگار باشد، آنگاه A^* در جستجوی گرافی بهینه است.

بهینگی A^* در جستجوی گرافی

۴۹

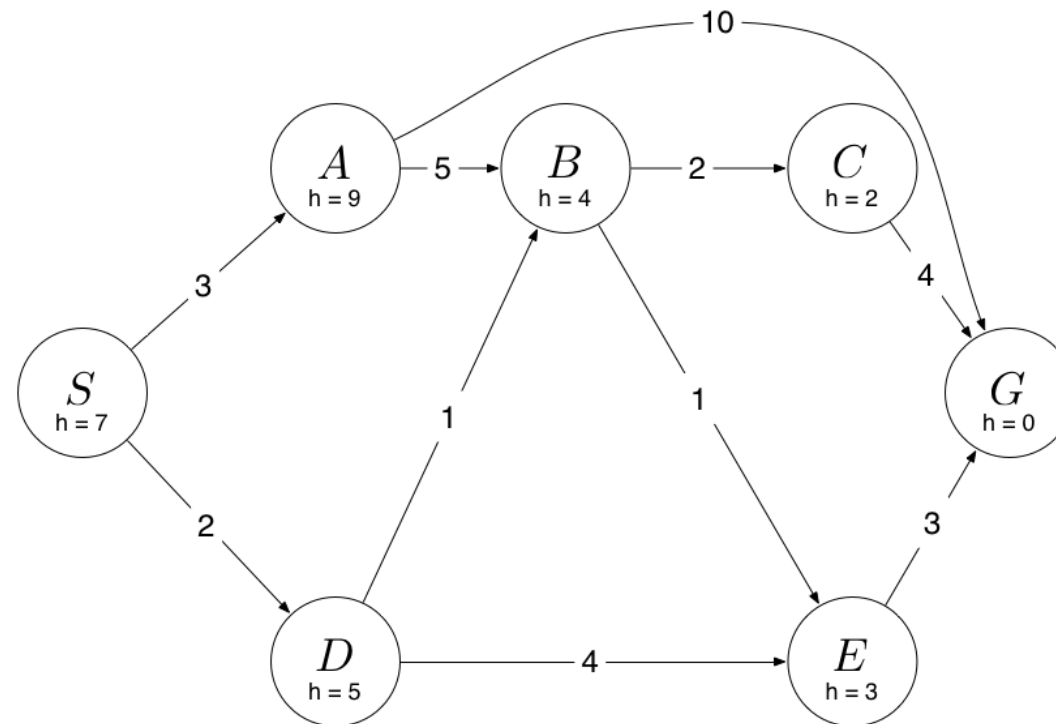
□ لم. A^* گره‌ها را به ترتیب صعودی مقادیر f گسترش می‌دهد:
□ A^* به ترتیب کانتورهای f را از کوچک به بزرگ اضافه می‌کند.



پرسش کلاسی: توابع هیوریستیک سازگار

۵۰

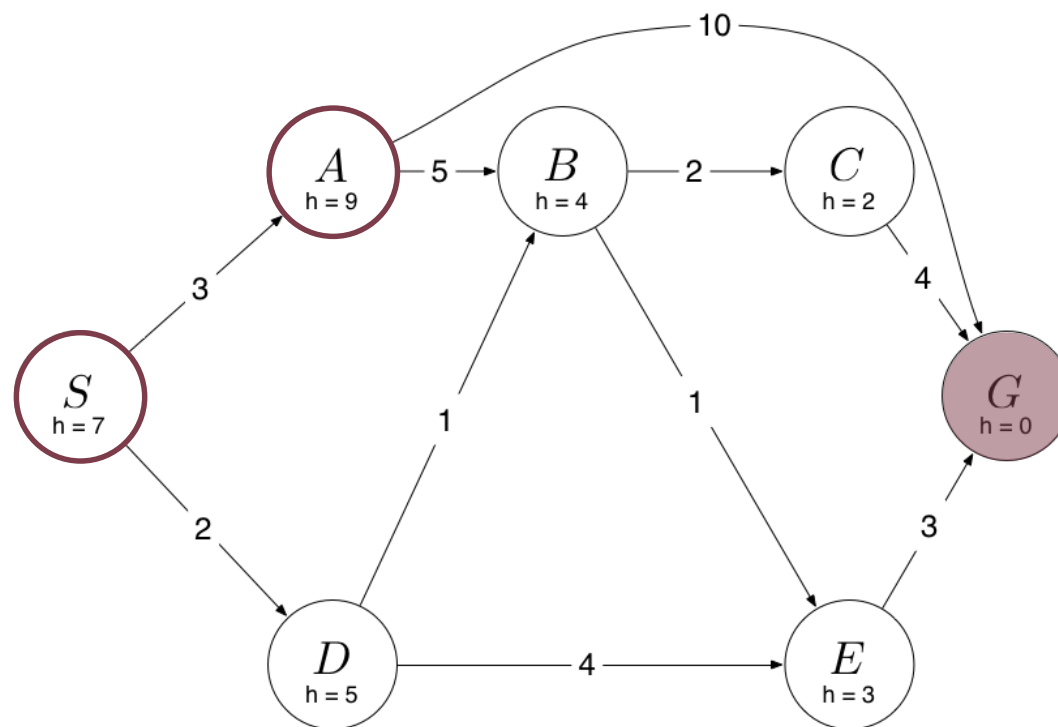
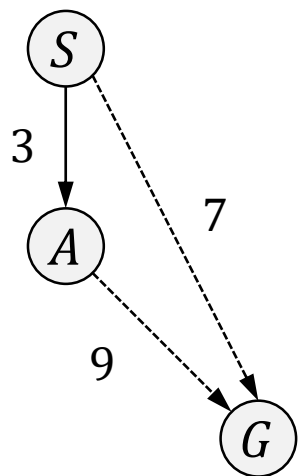
□ س. آیا تابع هیوریستیک داده شده در گراف زیر سازگار است؟



پرسش کلاسی: توابع هیوریستیک سازگار

۵۱

□ س. آیا تابع هیوریستیک داده شده در گراف زیر سازگار است؟



□ کامل. بله، مگر آن که تعداد نامحدودی گره با $f(n) \leq f(G)$ وجود داشته باشد.

□ گره‌ای با فاکتور انشعاب نامحدود وجود داشته باشد.

□ مسیری با هزینه محدود اما تعداد گره‌های نامحدود وجود داشته باشد. (عملگر با هزینه منفی یا صفر)

□ پیچیدگی زمانی. نمایی، بر حسب خطای نسبی هیوریستیک ضرب در هزینه راه‌حل.

□ مگر آن که نرخ رشد خطای تابع هیوریستیک بیشتر از لگاریتم هزینه مسیر واقعی نباشد.

$$|h(n) - h^*(n)| \leq O(\log h^*(n))$$

- پیچیدگی حافظه. نمایی، زیرا تمام گره‌های تولید شده را در حافظه نگه می‌دارد.
- بهینه. بله، نمی‌تواند کانتور f_{i+1} را گسترش دهد مگر آن که f_i تمام شده باشد.
 - تمامی گره‌ها با $f(n) < f^*$ گسترش داده می‌شوند.
 - برخی گره‌ها با $f(n) = f^*$ گسترش داده می‌شوند.
 - هیچ گره‌ای با $f(n) > f^*$ گسترش داده نمی‌شود.
- A^* دارای کارایی بهینه است.
- یعنی در بین تمام استراتژی‌های **بهینه**، کمترین تعداد گره‌های ممکن را تولید می‌کند.



□ جستجوی درختی.

□ جستجوی A^* بهینه است اگر هیوریستیک **قابل قبول** باشد.

□ جستجوی هزینه یکنواخت یک حالت خاص است ($h = 0$)

□ جستجوی گرافی.

□ جستجوی A^* بهینه است اگر هیوریستیک **سازگار** باشد.

□ جستجوی هزینه یکنواخت بهینه است ($h = 0$ سازگار است)

□ هر تابع هیوریستیک سازگار، قابل قبول نیز هست.

□ به طور کلی، اغلب توابع هیوریستیک قابل قبول طبیعی سازگار نیز هستند، به ویژه اگر از مسائل راحت شده به دست آمده باشند.

جستجوی A^* : خلاصه

۵۵

□ جستجوی A^* هم از هزینه پیموده شده استفاده می کند و هم از هزینه باقیمانده.

□ جستجوی A^* با توابع هیوریستیک قابل قبول یا سازگار بهینه است.

□ طراحی هیوریستیک یک نکته کلیدی است: اغلب به روش راحت سازی!



جستجوی هزینه یکنواخت



جستجوی مریضانه



جستجوی A^*

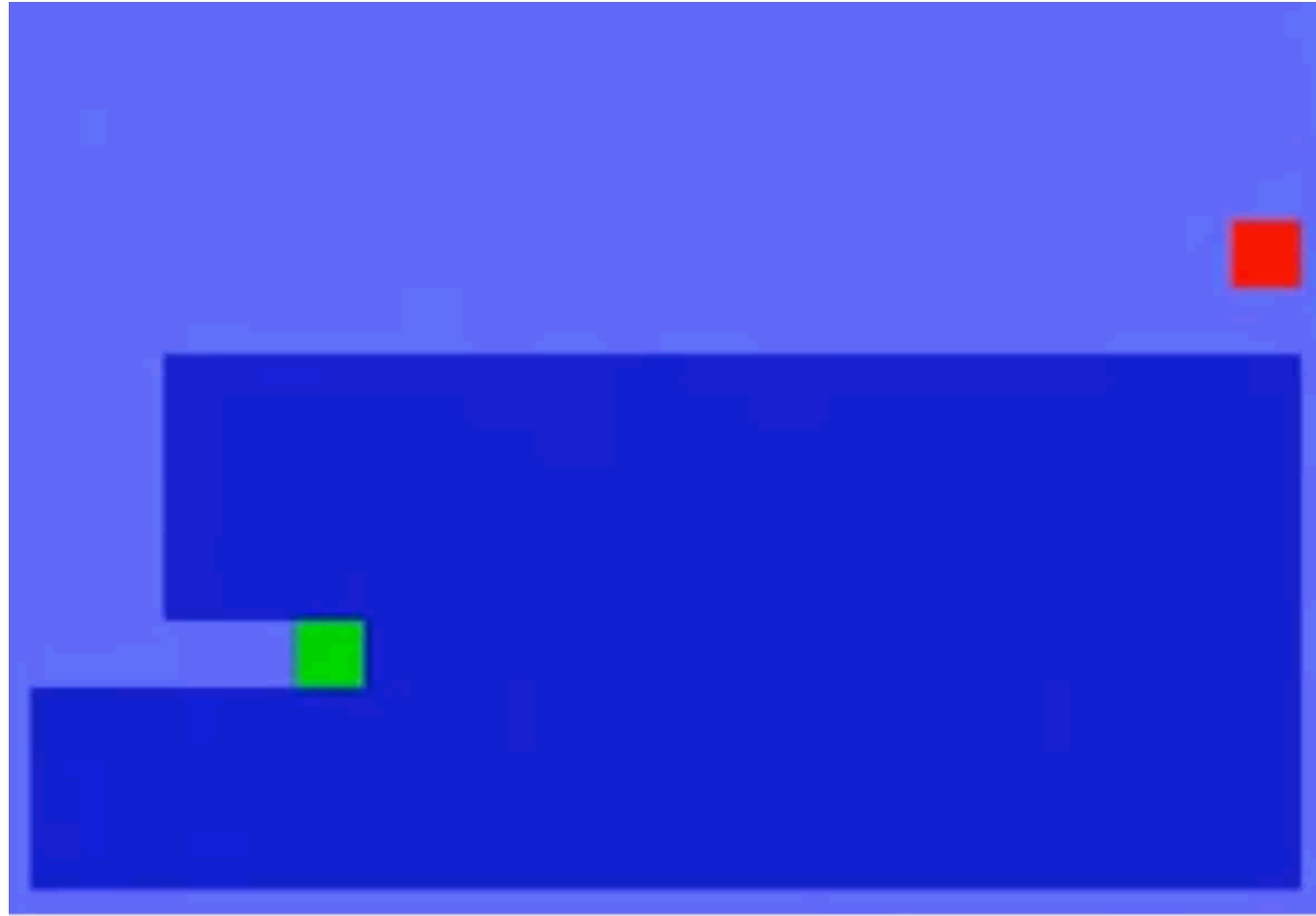
پرسش کلاسی: کدام الگوریتم جستجو؟

۵۶



پرسش کلاسی: کدام الگوریتم جستجو؟

۵۷



پرسش کلاسی: کدام الگوریتم جستجو؟

۵۸



پرسش کلاسی: کدام الگوریتم جستجو؟

۵۹



پرسش کلاسی: کدام الگوریتم جستجو؟

۶۰



روش‌های بهبود حافظه در A^*

۶۱

□ یادآوری. پیچیدگی زمانی و پیچیدگی حافظه A^* در اغلب مسائل **نمایی** است، مگر آن که برای آن مسئله یک تابع هیوریستیک بسیار دقیق وجود داشته باشد.

□ ایراد اصلی A^* . مصرف حافظه نمایی

□ روش‌های کاهش مصرف حافظه در A^* .

□ جستجوی IDA^* (مصرف حافظه خطی)

□ جستجوی RBFS (مصرف حافظه خطی)

□ جستجوی SMA^* (مصرف حافظه به اندازه حافظه موجود)

جستجوی اول-بهترین بازگشتی

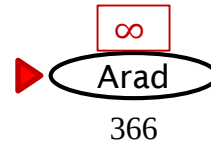
۶۲

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) return a solution or failure
    return RBFS(problem, MAKE-NODE(problem.INITIAL-STATE,  $\infty$ ))

function RBFS(problem, node, f_limit) return a solution or failure and a new f_cost limit
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    successors  $\leftarrow$  EXPAND(node, problem)
    if successors is empty then return failure,  $\infty$ 
    for each s in successors do
        s.f  $\leftarrow$  max(s.g + s.h, node.f)
    loop do
        best  $\leftarrow$  the lowest f-value node in successors
        if best.f > f_limit then return failure, best.f
        alternative  $\leftarrow$  the second lowest f-value among successors
        result, best.f  $\leftarrow$  RBFS(problem, best, min(f_limit, alternative))
    if result  $\neq$  failure then return result
```

جستجوی اول-بهترین بازگشتی: اجرا

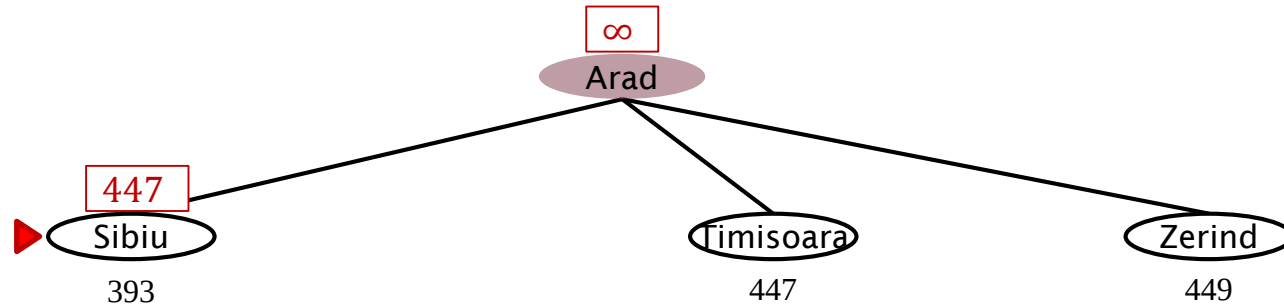
۶۳



```
function RBFS(problem, node, f_limit) return a solution or failure and a new
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  successors ← EXPAND(node, problem)
  if successors is empty then return failure, ∞
  for each s in successors do
    s.f ← max(s.g + s.h, node.f)
  loop do
    best ← the lowest f-value node in successors
    if best.f > f_limit then return failure, best.f
    alternative ← the second lowest f-value among successors
    result, best.f ← RBFS(problem, best, min(f_limit, alternative))
    if result ≠ failure then return result
```

جستجوی اول-بهترین بازگشتی: اجرا

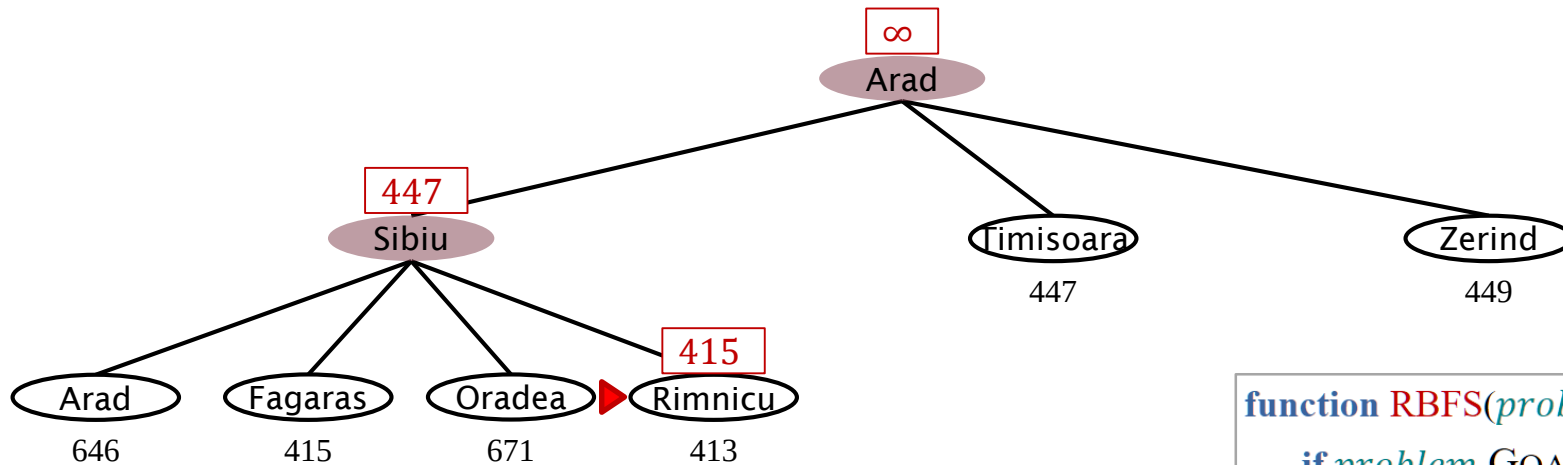
۶۴



```
function RBFS(problem, node, f_limit) return a solution or failure and a new
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  successors ← EXPAND(node, problem)
  if successors is empty then return failure, ∞
  for each s in successors do
    s.f ← max(s.g + s.h, node.f)
  loop do
    best ← the lowest f-value node in successors
    if best.f > f_limit then return failure, best.f
    alternative ← the second lowest f-value among successors
    result, best.f ← RBFS(problem, best, min(f_limit, alternative))
    if result ≠ failure then return result
```

جستجوی اول-بهترین بازگشتی: اجرا

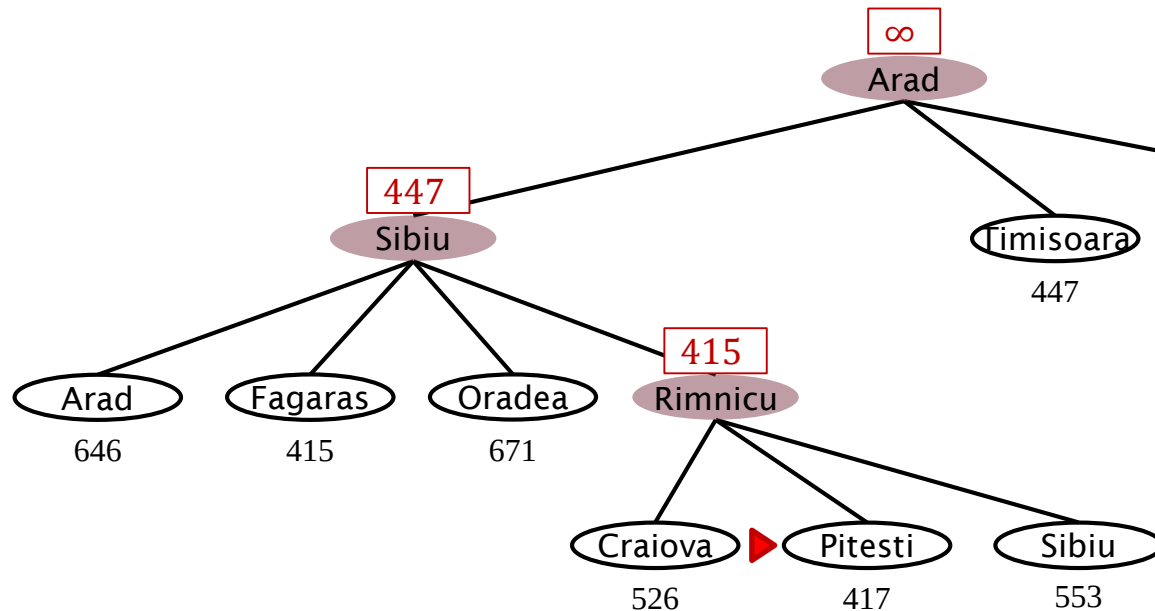
۶۵



```
function RBFS(problem, node, f_limit) return a solution or failure and a new
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  successors ← EXPAND(node, problem)
  if successors is empty then return failure,  $\infty$ 
  for each s in successors do
    s.f ← max(s.g + s.h, node.f)
  loop do
    best ← the lowest f-value node in successors
    if best.f > f_limit then return failure, best.f
    alternative ← the second lowest f-value among successors
    result, best.f ← RBFS(problem, best, min(f_limit, alternative))
  if result ≠ failure then return result
```

جستجوی اول-بهترین بازگشتی: اجرا

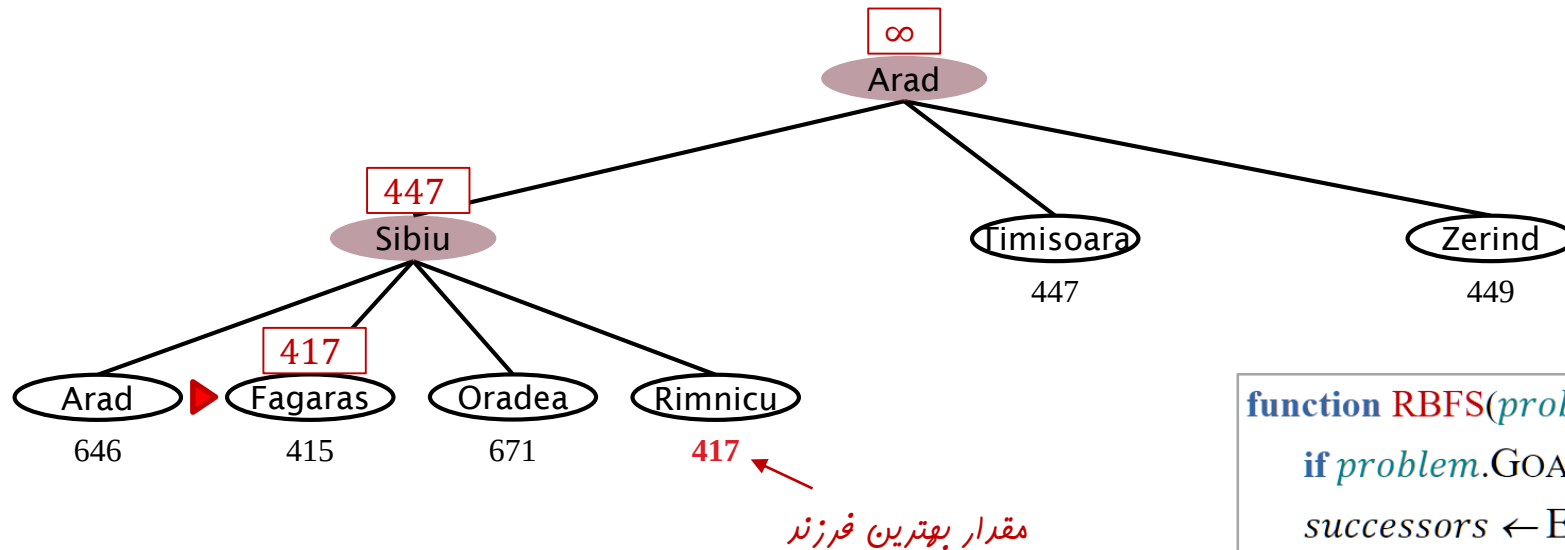
۶۶



```
function RBFS(problem, node, f_limit) return a solution or failure and a new
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
successors ← EXPAND(node, problem)
if successors is empty then return failure, ∞
for each s in successors do
    s.f ← max(s.g + s.h, node.f)
loop do
    best ← the lowest f-value node in successors
    if best.f > f_limit then return failure, best.f
    alternative ← the second lowest f-value among successors
    result, best.f ← RBFS(problem, best, min(f_limit, alternative))
    if result ≠ failure then return result
```

جستجوی اول-بهترین بازگشتی: اجرا

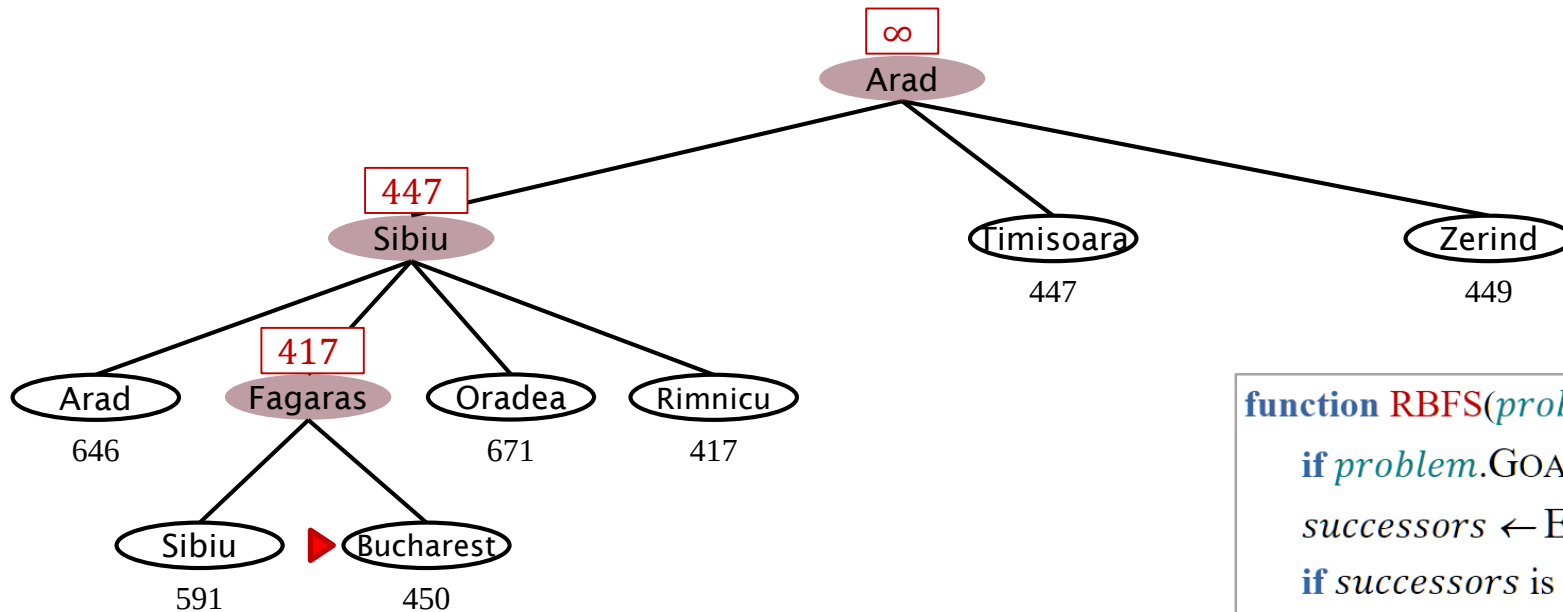
۶۷



```
function RBFS(problem, node, f_limit) return a solution or failure and a new  
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)  
successors ← EXPAND(node, problem)  
if successors is empty then return failure, ∞  
for each s in successors do  
    s.f ← max(s.g + s.h, node.f)  
loop do  
    best ← the lowest f-value node in successors  
    if best.f > f_limit then return failure, best.f  
    alternative ← the second lowest f-value among successors  
    result, best.f ← RBFS(problem, best, min(f_limit, alternative))  
    if result ≠ failure then return result
```

جستجوی اول-بهترین بازگشتی: اجرا

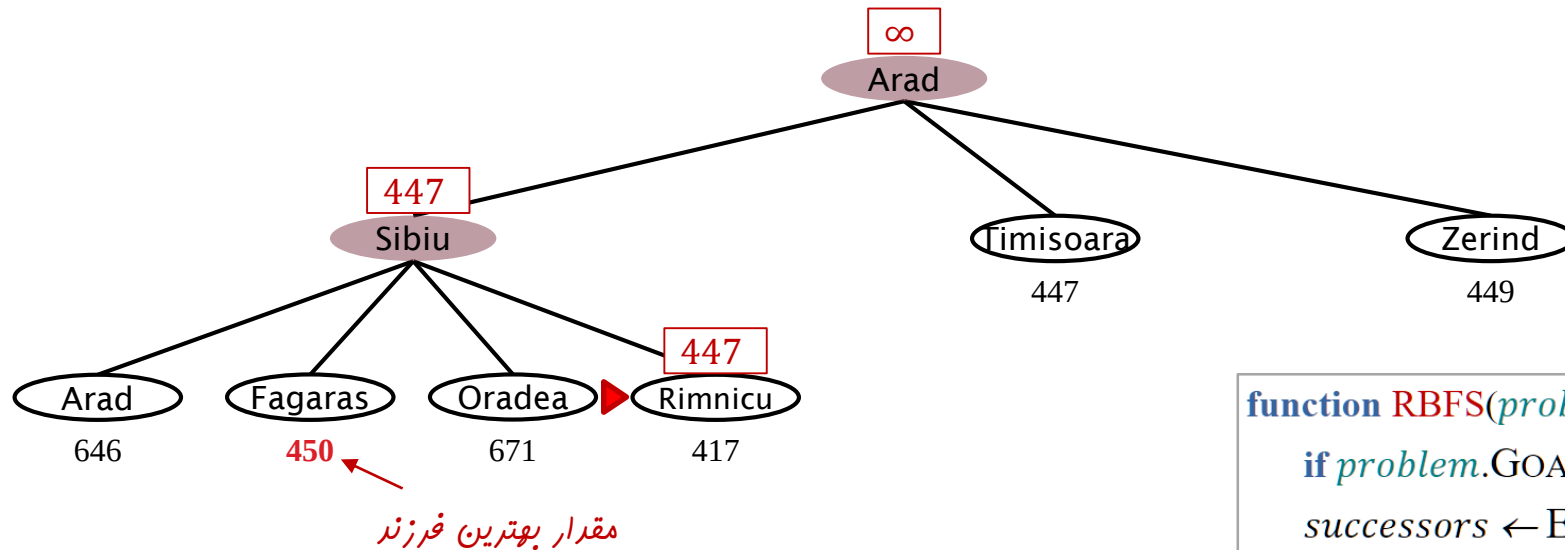
۶۸



```
function RBFS(problem, node, f_limit) return a solution or failure and a new
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
successors ← EXPAND(node, problem)
if successors is empty then return failure, ∞
for each s in successors do
    s.f ← max(s.g + s.h, node.f)
loop do
    best ← the lowest f-value node in successors
    if best.f > f_limit then return failure, best.f
    alternative ← the second lowest f-value among successors
    result, best.f ← RBFS(problem, best, min(f_limit, alternative))
    if result ≠ failure then return result
```

جستجوی اول-بهترین بازگشتی: اجرا

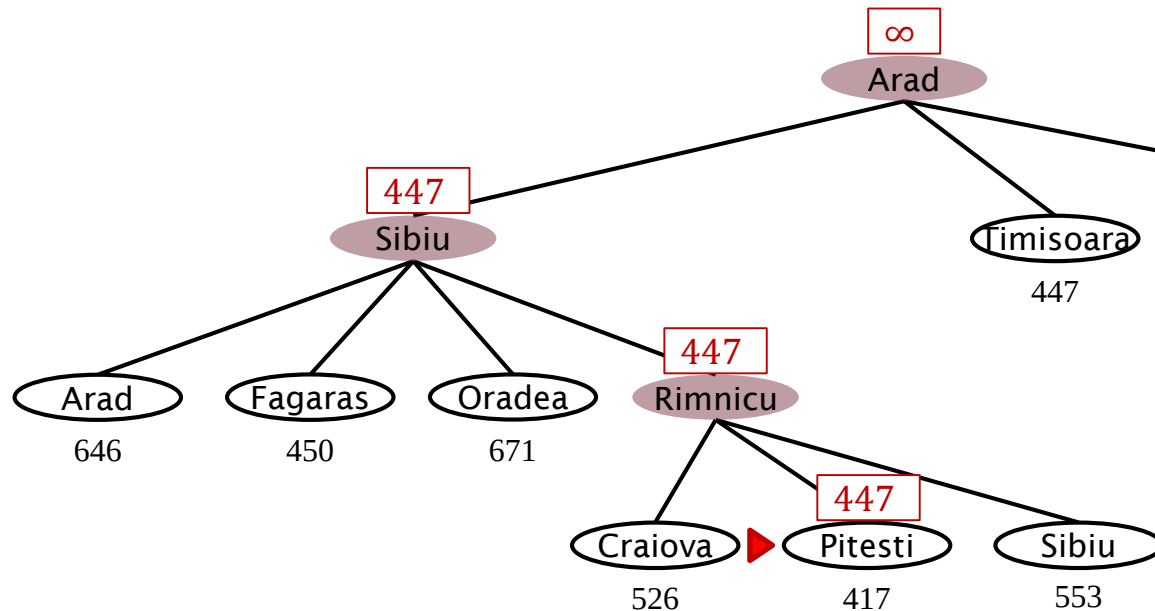
۶۹



```
function RBFS(problem, node, f_limit) return a solution or failure and a new  
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)  
successors ← EXPAND(node, problem)  
if successors is empty then return failure, ∞  
for each s in successors do  
    s.f ← max(s.g + s.h, node.f)  
loop do  
    best ← the lowest f-value node in successors  
    if best.f > f_limit then return failure, best.f  
    alternative ← the second lowest f-value among successors  
    result, best.f ← RBFS(problem, best, min(f_limit, alternative))  
    if result ≠ failure then return result
```

جستجوی اول-بهترین بازگشتی: اجرا

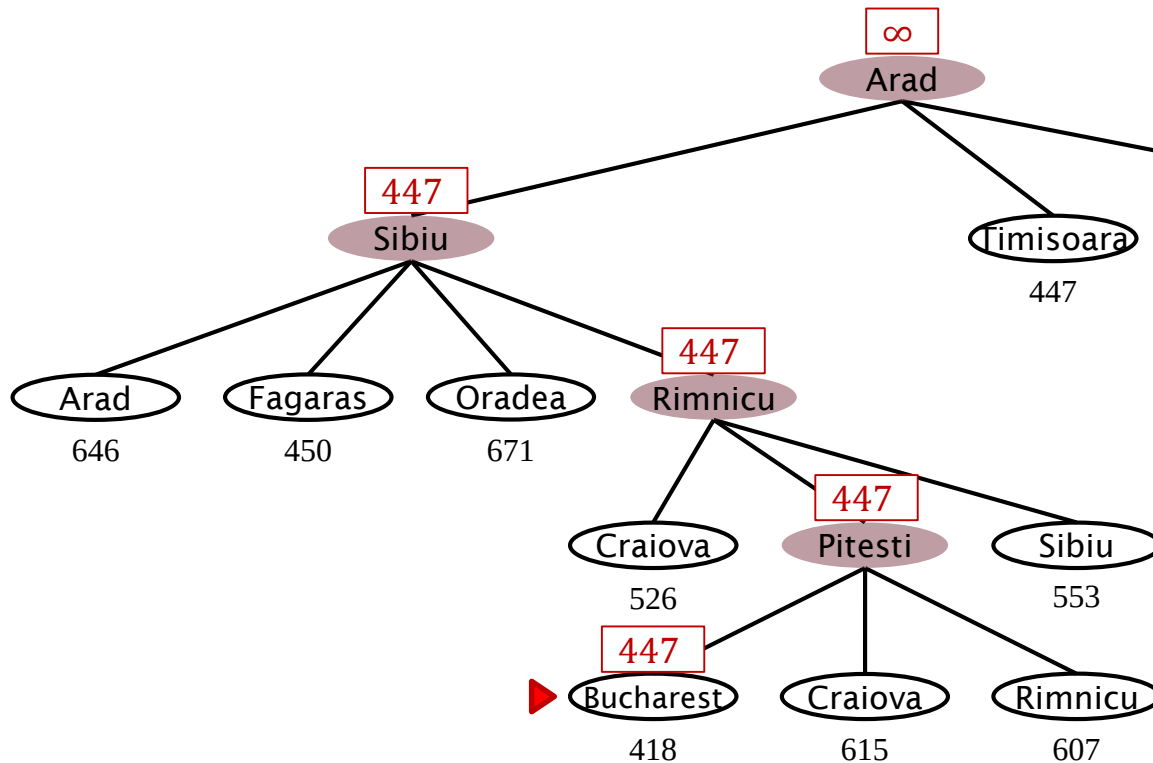
۷۰



```
function RBFS(problem, node, f_limit) return a solution or failure and a new
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
successors ← EXPAND(node, problem)
if successors is empty then return failure, ∞
for each s in successors do
    s.f ← max(s.g + s.h, node.f)
loop do
    best ← the lowest f-value node in successors
    if best.f > f_limit then return failure, best.f
    alternative ← the second lowest f-value among successors
    result, best.f ← RBFS(problem, best, min(f_limit, alternative))
    if result ≠ failure then return result
```

جستجوی اول-بهترین بازگشتی: اجرا

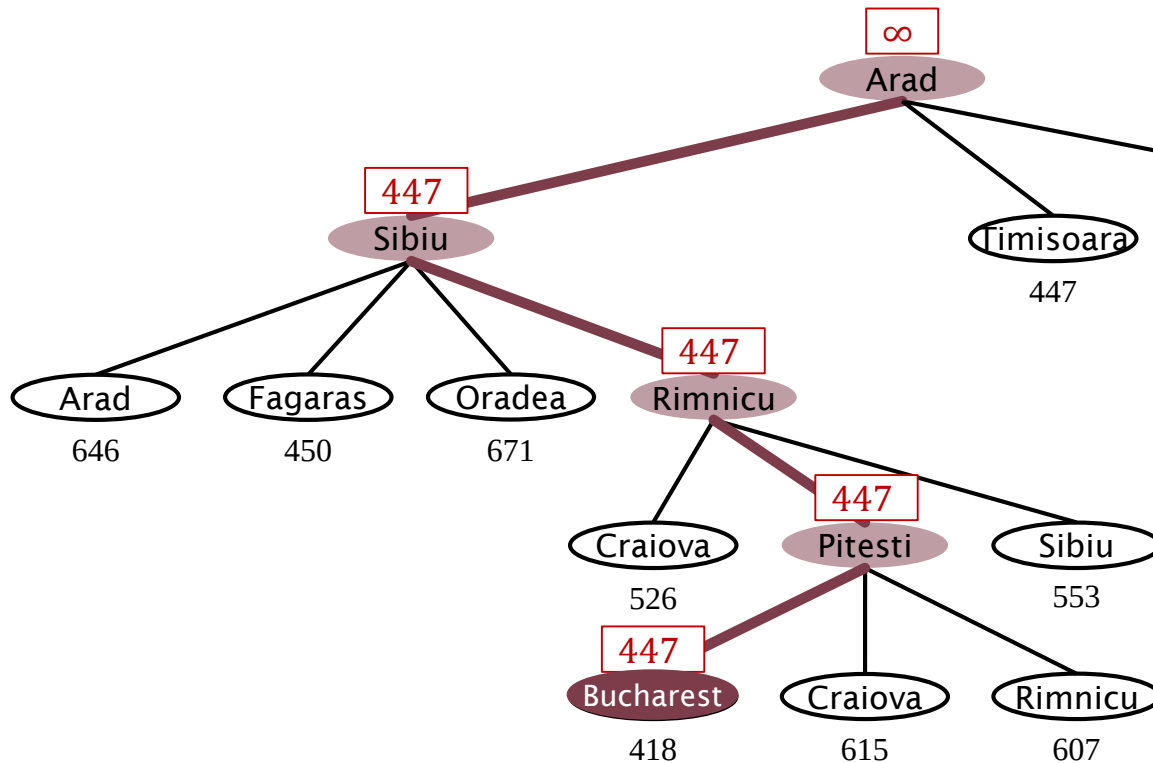
۷۱



```
function RBFS(problem, node, f_limit) return a solution or failure and a new
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
successors ← EXPAND(node, problem)
if successors is empty then return failure, ∞
for each s in successors do
    s.f ← max(s.g + s.h, node.f)
loop do
    best ← the lowest f-value node in successors
    if best.f > f_limit then return failure, best.f
    alternative ← the second lowest f-value among successors
    result, best.f ← RBFS(problem, best, min(f_limit, alternative))
    if result ≠ failure then return result
```

جستجوی اول-بهترین بازگشتی: اجرا

۷۲



function RBFS(*problem*, *node*, *f_limit*) **return** a solution or failure and a new

if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)

successors ← EXPAND(*node*, *problem*)

if *successors* is empty **then return** failure, ∞

for each *s* **in** *successors* **do**

s.*f* ← max(*s*.*g* + *s*.*h*, *node*.*f*)

loop do

best ← the lowest *f*-value node in *successors*

if *best*.*f* > *f_limit* **then return** failure, *best*.*f*

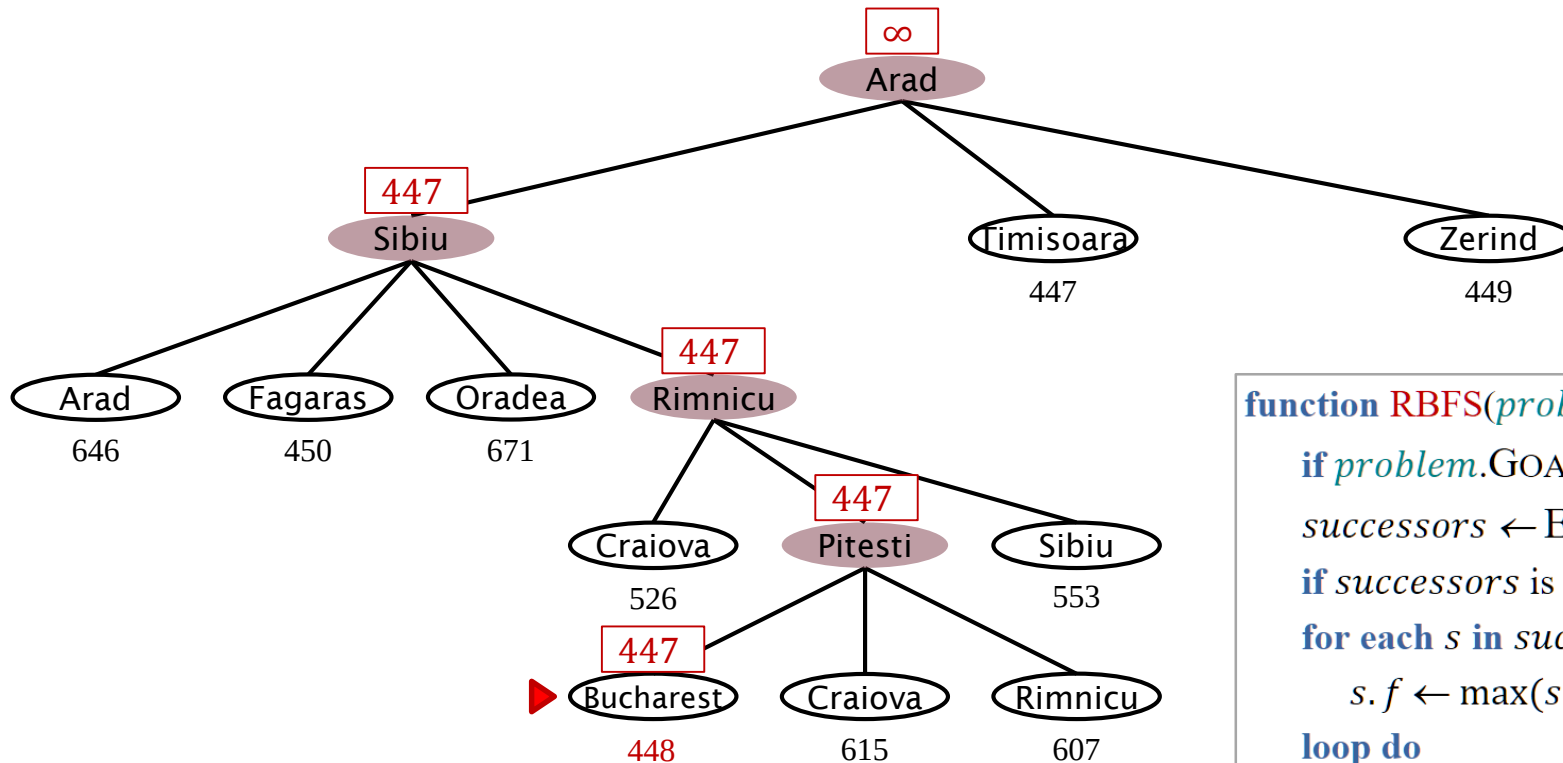
alternative ← the second lowest *f*-value among *successors*

result, *best*.*f* ← RBFS(*problem*, *best*, min(*f_limit*, *alternative*))

if *result* ≠ failure **then return** *result*

جستجوی اول-بهترین بازگشتی: اجرا

۷۳



اگر در این مرحله، مقدار هزینه بهترین فرزند (بخارست) به جای ۴۱۸ برابر با ۴۴۸ بود، چه اتفاقی می افتاد؟

```

function RBFS(problem, node, f_limit) return a solution or failure and a new
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
successors ← EXPAND(node, problem)
if successors is empty then return failure, ∞
for each s in successors do
    s.f ← max(s.g + s.h, node.f)
loop do
    best ← the lowest f-value node in successors
    if best.f > f_limit then return failure, best.f
    alternative ← the second lowest f-value among successors
    result, best.f ← RBFS(problem, best, min(f_limit, alternative))
    if result ≠ failure then return result
  
```

ارزیابی جستجوی اول-بهترین بازگشتی

۷۴

- پیچیدگی زمانی. RBFS کمی کاراتر از IDA* است.
 - هنوز گسترش تکراری گره‌ها وجود دارد [تغییر عقیده]
- پیچیدگی حافظه. خطی
 - IDA* فقط یک عدد را نگهداری می‌کند [حد فعلی f-cost]
- تعیین پیچیدگی زمانی مشکل است.
 - به دقت تابع هیوریستیک و میزان تغییر بهترین مسیر در اثر بسط گره‌ها بستگی دارد.
 - مانند IDA* در معرض افزایش نمایی پیچیدگی زمانی قرار دارد.
- RBFS و IDA* هر دو از میزان بسیار کم حافظه رنج می‌برند.
 - با اینکه حافظه زیادی وجود دارد، نمی‌توانند از آن استفاده کنند.

- ایده. استفاده از کل حافظه موجود!
- گسترش بهترین گره برگی تا زمانی که حافظه پر نشده است.
- حذف بدترین گره برگی در صورت پر بودن حافظه و سپس گسترش بهترین گره برگی
- اگر هزینه تمام برگ‌ها برابر باشند چه می‌شود؟
- ممکن است یک گره هم برای گسترش و هم برای حذف انتخاب شود.
- گسترش: بهترین گره برگی که از همه جدیدتر است.
- حذف: بدترین گره برگی که از همه قدیمی‌تر است.
- کامل بودن و بهینگی.
- SMA* کامل است اگر حداقل یک راه‌حل قابل دستیابی وجود داشته باشد.
- SMA* بهینه است اگر راه‌حل بهینه قابل دستیابی باشد.

کاربردهای A*

۷۶



□ کاربردها.

- مسیریابی
- بازی‌های کامپیوتری
- برنامه‌ریزی منابع
- برنامه‌ریزی حرکت روبات
- تحلیل زبانی
- ترجمه ماشینی
- شناسایی گفتار
- ...

ابداع توابع هیوریستیک

۷۷



ابداع توابع هیوریستیک

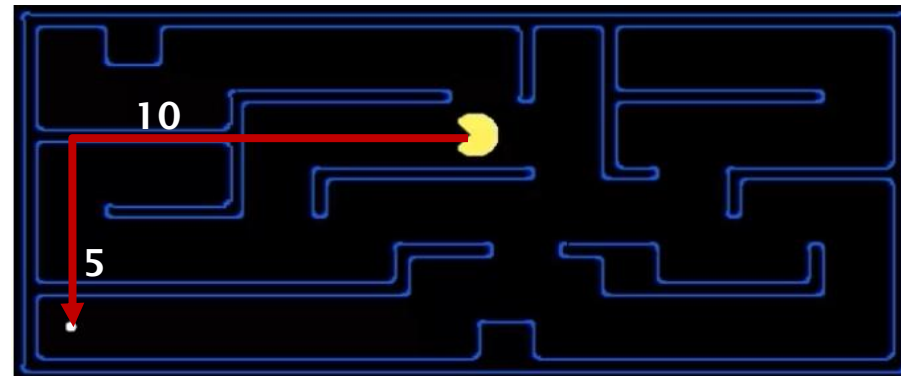
۷۸

□ حل مسائل سخت به صورت بهینه، بستگی به وجود توابع هیوریستیک قابل قبول دارد.

□ روش راحت سازی.

■ در اغلب موارد، توابع هیوریستیک با حل کردن مسائل راحت شده به دست می آیند.

□ مسئله راحت شده. نسخه ای از مسئله اصلی که در آن یک یا چند محدودیت حذف شده اند.



فاصله مانه تانی

مثال: محمای ۸

۷۹

7	2	4
5		6
8	3	1

حالت شروع



3	7	1
2	4	5
	8	6

اعمال

	1	2
3	4	5
6	7	8

حالت هدف

□ حالت‌ها؟

□ تعداد حالت‌ها؟

□ اعمال؟

□ هزینه؟

مثال: معمای ۸

۸۰

7	2	4
5		6
8	3	1

حالت شروع



3	7	1
2	4	5
8		6

اعمال

	1	2
3	4	5
6	7	8

حالت هدف

- حالت‌ها؟ هر چیدمان از ۸ کاشی بر روی صفحه
- تعداد حالت‌ها؟ ۹!
- اعمال؟ حرکت دادن یک کاشی به یک خانه مجاور و خالی
- هزینه؟ ۱ به ازای هر عمل

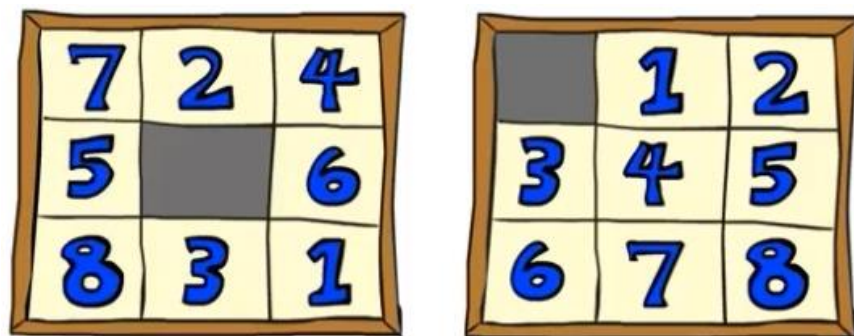
راحت سازی

۸۱

□ محدودیت‌ها. [برای حرکت دادن یک کاشی از خانه A به خانه B]

I. خانه B باید خالی باشد.

II. خانه B باید همسایه خانه A باشد.



حالت شروع

حالت هدف

□ مسائل راحت شده.

□ نسخه ۱. هر دو محدودیت حذف شوند.

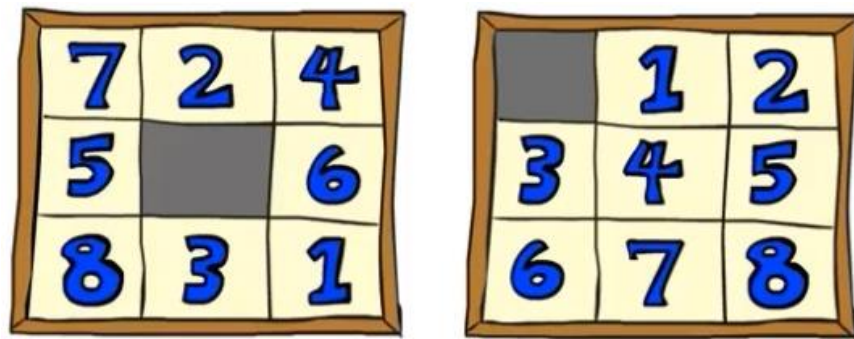
□ نسخه ۲. محدودیت I حذف شود.

□ نسخه ۳. محدودیت II حذف شود.

نکته اصلی. هزینه دقیق یک مسئله راحت شده را می‌توان به عنوان تخمینی از هزینه واقعی مسئله اصلی در نظر گرفت.

راحت سازی ۱

۸۲



$$h_1(start) = 8$$

□ مسائل راحت شده.

□ نسخه ۱. هر دو محدودیت حذف شوند.

□ نسخه ۲. محدودیت I حذف شود.

□ نسخه ۳. محدودیت II حذف شود.

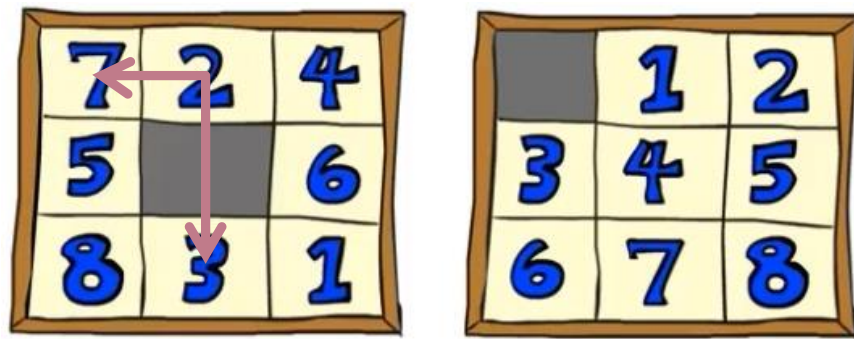
□ هزینه حل مسئله راحت شده = تعداد کاشی‌هایی که در مکان نادرست قرار دارند.

عمق ۲۴	عمق ۱۴	عمق ۴	
$5/4 \times 10^{10}$	۳,۴۷۳,۹۴۱	۱۱۲	IDS
۳۹,۱۳۵	۵۳۹	۱۳	h_1

مقایسه تعداد گره‌های تولید شده به وسیله جستجوی عمیق کننده تکراری و جستجوی A^*

راحت سازی ۲

۸۳



$$h_2(start) = 18$$

□ مسائل راحت شده.

□ نسخه ۱. هر دو محدودیت حذف شوند.

□ نسخه ۲. محدودیت I حذف شود.

□ نسخه ۳. محدودیت II حذف شود.

□ هزینه حل مسئله راحت شده = مجموع فواصل مانهاتانی کاشی‌ها تا مکان هدف.

عمق ۲۴	عمق ۱۴	عمق ۴	
$5/4 \times 10^{10}$	۳,۴۷۳,۹۴۱	۱۱۲	IDS
۳۹,۱۳۵	۵۳۹	۱۳	h_1
۱,۶۴۱	۱۱۳	۱۲	h_2

مقایسه تعداد گره‌های تولید شده به وسیله جستجوی عمیق کننده تکراری و جستجوی A^*

مثال: معمای ۸

□ هیوریستیک h_1

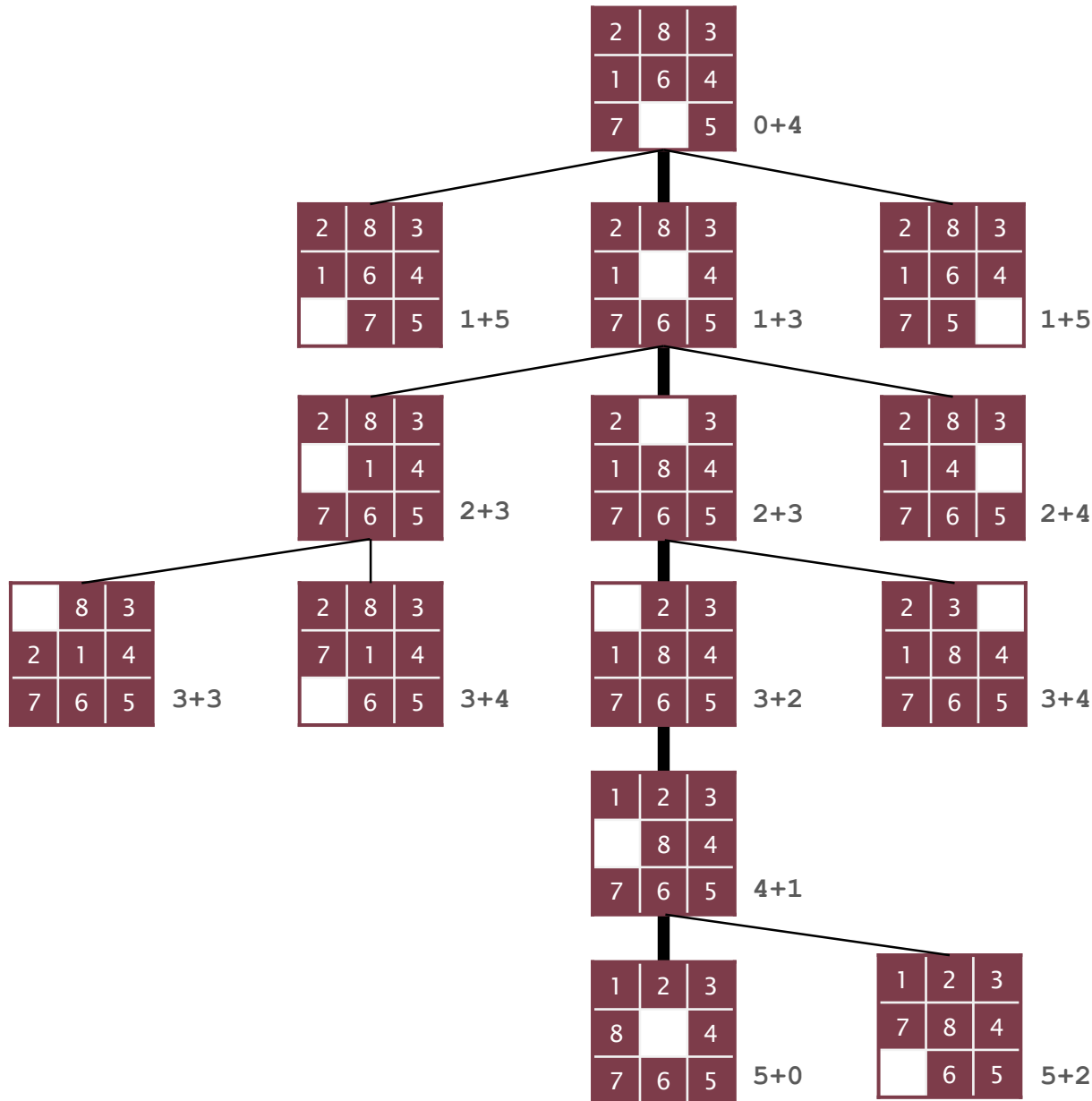
□ تعداد کاشی‌هایی که در مکان نادرست هستند

□ حالت هدف.

1	2	3
8		4
7	6	5

□ راه‌حل.

[بالا، بالا، چپ، پایین، راست]



مثال: معمای ۸

ہیوریستیک h_2 □

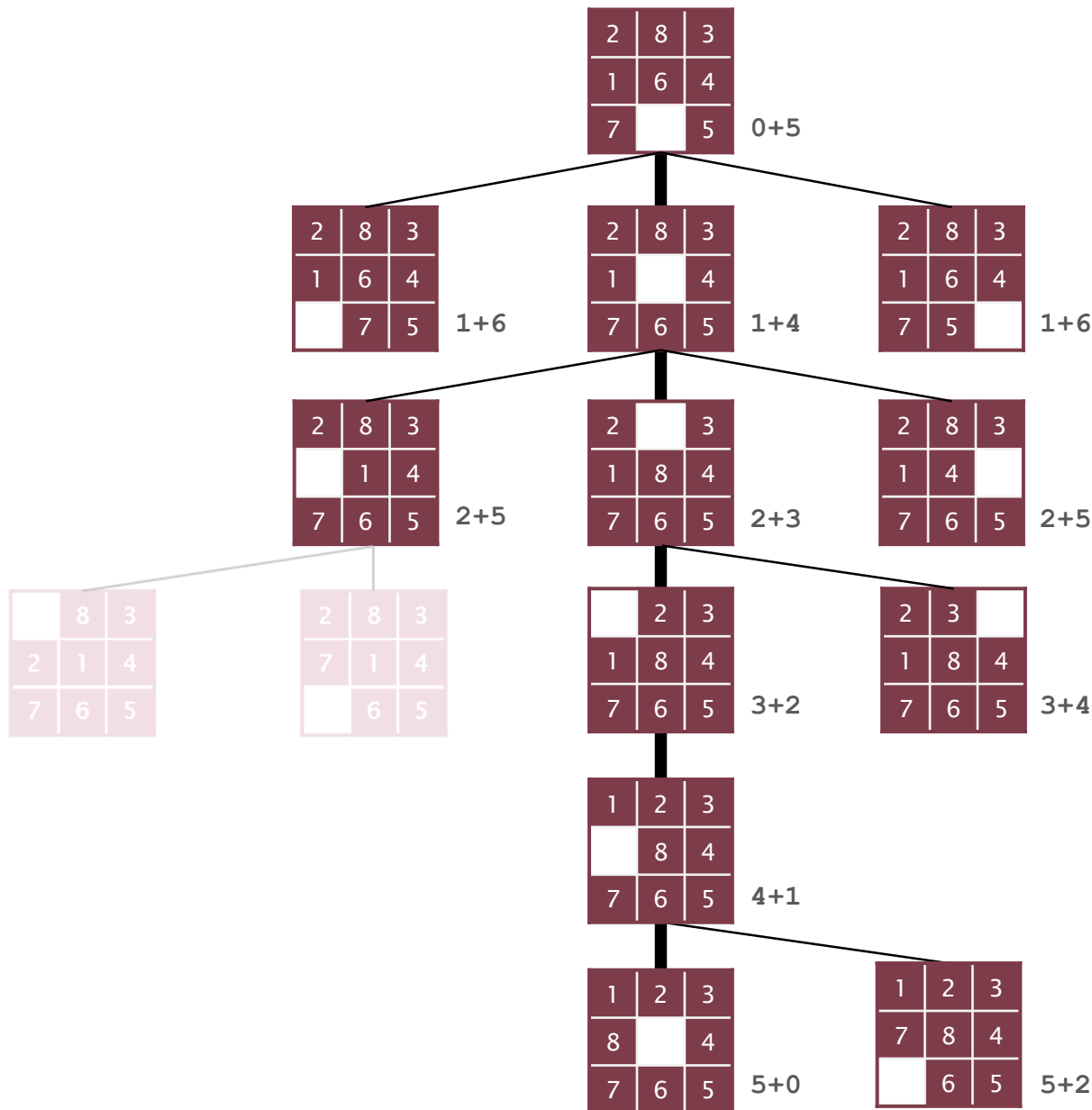
■ مجموع فواصل مانهاتانی کاشی‌ها تا هدف

□ حالت هدف.

1	2	3
8		4
7	6	5

□ راه حل.

[بالا، بالا، چپ، پایین، راست]



□ س. آیا می توان از **هزینه واقعی** به عنوان یک هیوریستیک استفاده نمود؟

□ آیا این هیوریستیک قابل قبول است؟

□ آیا این هیوریستیک باعث صرفه جویی در تعداد گره های تولید شده می شود؟

□ مشکل استفاده از این هیوریستیک چیست؟



□ **توجه.** در ابداع توابع هیوریستیک، مسئله اصلی برقراری یک توازن مناسب میان **کیفیت** تابع هیوریستیک و **هزینه** محاسبه آن به ازای هر گره است.

□ هیوریستیک های دقیق تر تعداد گره های کمتری تولید می کنند، اما محاسبه آنها معمولاً به زمان بیشتری نیاز دارد.

مقایسه کیفیت توابع هیوریستیک

۸۷

- پرسش. آیا هیوریستیک h_2 همواره تعداد گره های کمتری نسبت به h_1 تولید می کند؟
- پاسخ. بله، زیرا h_2 بر h_1 تسلط دارد.

□ تسلط. اگر به ازای هر n داشته باشیم:

$$h_2(n) \geq h_1(n) \quad (\text{الف})$$

(ب) و هر دو تابع هیوریستیک قابل قبول باشند،

آنگاه h_2 بر h_1 تسلط دارد.

عمق ۲۴	عمق ۱۴	عمق ۴	
۳۹,۱۳۵	۵۳۹	۱۳	h_1
۱,۶۴۱	۱۱۳	۱۲	h_2

ابداع توابع هیوریستیک: خلاصه

□ راحت‌سازی. می‌توان هیوریستیک‌های قابل قبول را برای یک مسئله، از **هزینه دقیق** راه‌حل یک نسخه راحت شده از مسئله به دست آورد.

□ مثال. [معمای ۸] یک کاشی می‌تواند از خانه A به خانه B حرکت کند، اگر A همسایه B باشد و B خالی باشد.

□ اگر یک کاشی بتواند به **هر خانه‌ای** حرکت کند، آنگاه هیوریستیک h_1 کوتاه‌ترین راه‌حل را می‌دهد.

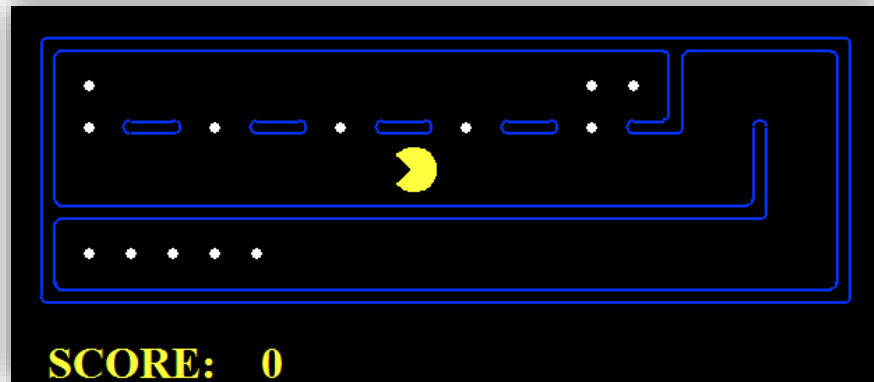
□ اگر یک کاشی بتواند به **هر خانه همسایه** حرکت کند، آنگاه هیوریستیک h_2 کوتاه‌ترین راه‌حل را می‌دهد.

□ نکته کلیدی. هزینه راه‌حل بهینه در یک مسئله راحت، بیشتر از هزینه راه‌حل بهینه در مسئله اصلی نیست!

پرسش کلاسی: توابع هیوریستیک قابل قبول

۸۹

□ مسئله‌ای را در نظر بگیرید که در آن هدف پکمن خوردن همه غذاهای موجود است. کدام یک از توابع هیوریستیک زیر برای این مسئله قابل قبول هستند؟



□ تعداد غذاهای باقیمانده

□ فاصله تا نزدیک‌ترین غذا

□ فاصله تا دورترین غذا

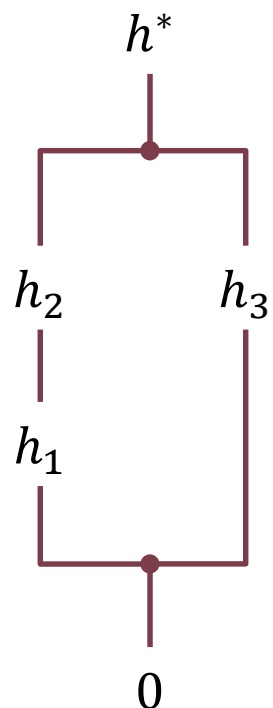
□ فاصله تا نزدیک‌ترین غذا به علاوه فاصله تا دورترین غذا

□ هیچ کدام

پرسش کلاسی: ترکیب توابع هیوریستیک

۹۰

□ کدام یک از گزاره‌های زیر صحیح هستند؟



- میانگین چند تابع هیوریستیک قابل قبول یک تابع هیوریستیک قابل قبول است.
- بیشینه و کمینه چند تابع هیوریستیک قابل قبول یک تابع هیوریستیک قابل قبول است.
- بیشینه چند تابع هیوریستیک بر کمینه آنها تسلط دارد.
- کمینه چند تابع هیوریستیک بر بیشینه آنها تسلط دارد.