دانشگاه صنعتی امیرکبیر
دانشکده مهندسی کامپیوتر و فناوری اطلاعات

راهبردهای جستجوی آگاهانه

«هوش مصنوعی: یک رهیافت نوین»، فصل ۳

مدرس: سیده فاطمه موسوی

نیم‌سال اول ۱۳۹۹–۱۳۹۸

رئوس مطالب

- جستجوی آگاهانه
  - استفاده از دانش خاص مسئله
  - اطلاعات بیشتر در مورد حالت مانند فاصله تا هدف

- در این بخش به موارد زیر خواهیم پرداخت:
  - جستجوی اول بهترین
  - جستجوی اول بهترین حریصانه
  - جستجوی A*
  - بهبود A* با الگوریتم‌هایی مانند IDA*، RBFS و SMA*
  - توابع هیوریستیک

راهبرد جستجوی اول—بهترین

جستجوی اول بهترین – Best First

- ایده: استفاده از یک تابع ارزیاب (Evaluation function) $f(n)$ برای هر گره و بسط مطلوب‌ترین نود بسط‌نیافته
  - کلی‌تر از تابع $g(n)$ یا همان هزینه‌ی رسیدن به گره n است.
  - تابع ارزیاب یک حد بالا بر روی مطلوبیت گره (یا یک حد پایین بر روی هزینه) فراهم می‌آورد.

- پیاده‌سازی
  - صف اولویت: ترتیب گره‌ها در مجموعه‌ی frontier به ترتیب نزولی میزان مطلوبیت است.
  - انتخاب تابع f، راهبرد جستجو را تعیین می‌کند.

ارتباط جستجوهای ناآگاهانه و جستجوی اول بهترین

- با تعریف مناسب توابع ارزیاب می‌توان جستجوهای ناآگاهانه را به شکل جستجوی اول بهترین پیاده‌سازی نمود.

- جستجوی هزینه یکنواخت؟
  - $f(n)=g(n)$
- جستجوی اول سطح؟
  - $f(n)=depth(n)$
- جستجوی اول عمق؟
  - $f(n)=-depth(n)$ یا $f(n)=1/depth(n)$

تابع هیوریستیک

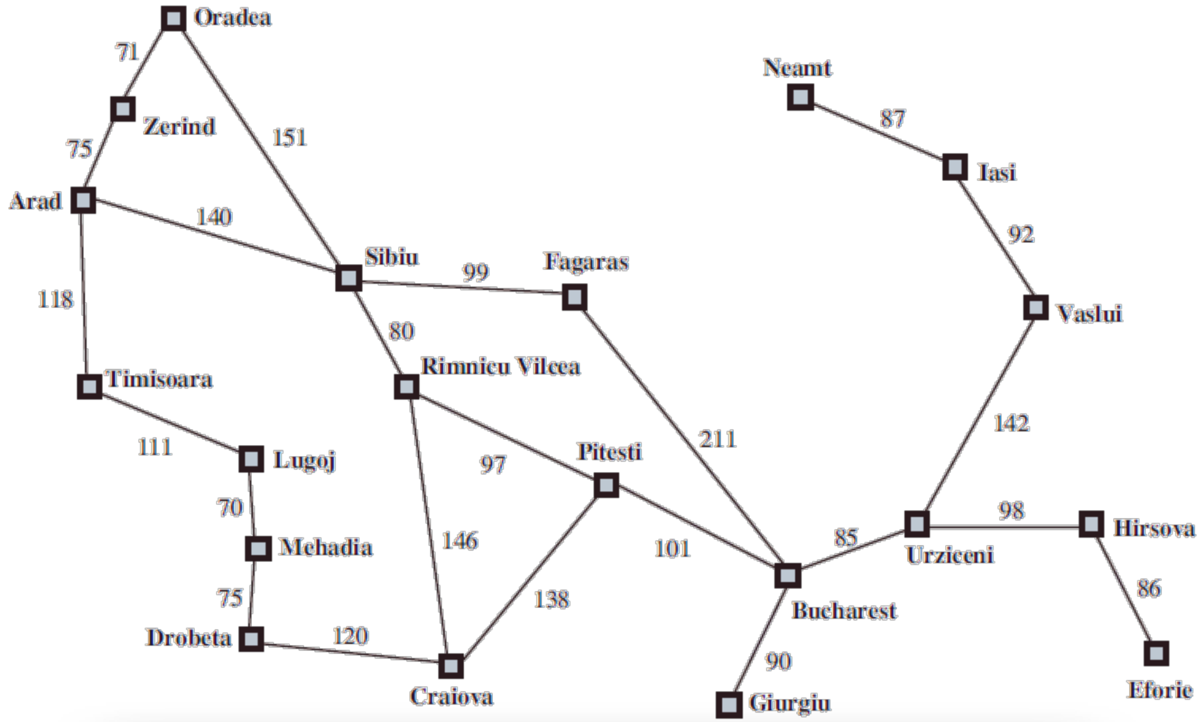
- وارد کردن دانش خاص مسئله در جستجو
  - اطلاعاتی بیشتر از تعریف مسئله به منظور رسیدن هرچه سریع‌تر به یک راه‌حل بهینه
- تابع هیوریستیک می‌تواند به‌عنوان جزئی از تابع ارزیاب $f(n)$ تعریف شود.
- $h(n)$= هزینه‌ی تخمینی ارزان‌ترین مسیر از وضعیت موجود در گره‌ی n به یک وضعیت هدف
  - تنها وابسته به وضعیت n است نه مسیر از ریشه تا n
  - اگر n یک وضعیت هدف باشد آن‌گاه $h(n)=0$
  - $h(n)\geq 0$
- تابع هوریستیک می‌تواند توسط یک قانون سرانگشتی، ساده‌سازی مسئله و حدس‌های تجربی به‌دست آمده باشد که باعث محدود کردن یا کاهش دادن جستجو در دامنه‌هایی می‌شود که پیچیده و مشکل هستند.

جستجوی اول–بهترین حریصانه

- نودی را بسط می‌دهد که به نظر می‌رسد به هدف نزدیک‌تر است.
- تابع ارزیاب: $f(n)=h(n)$
  - برای مثال:

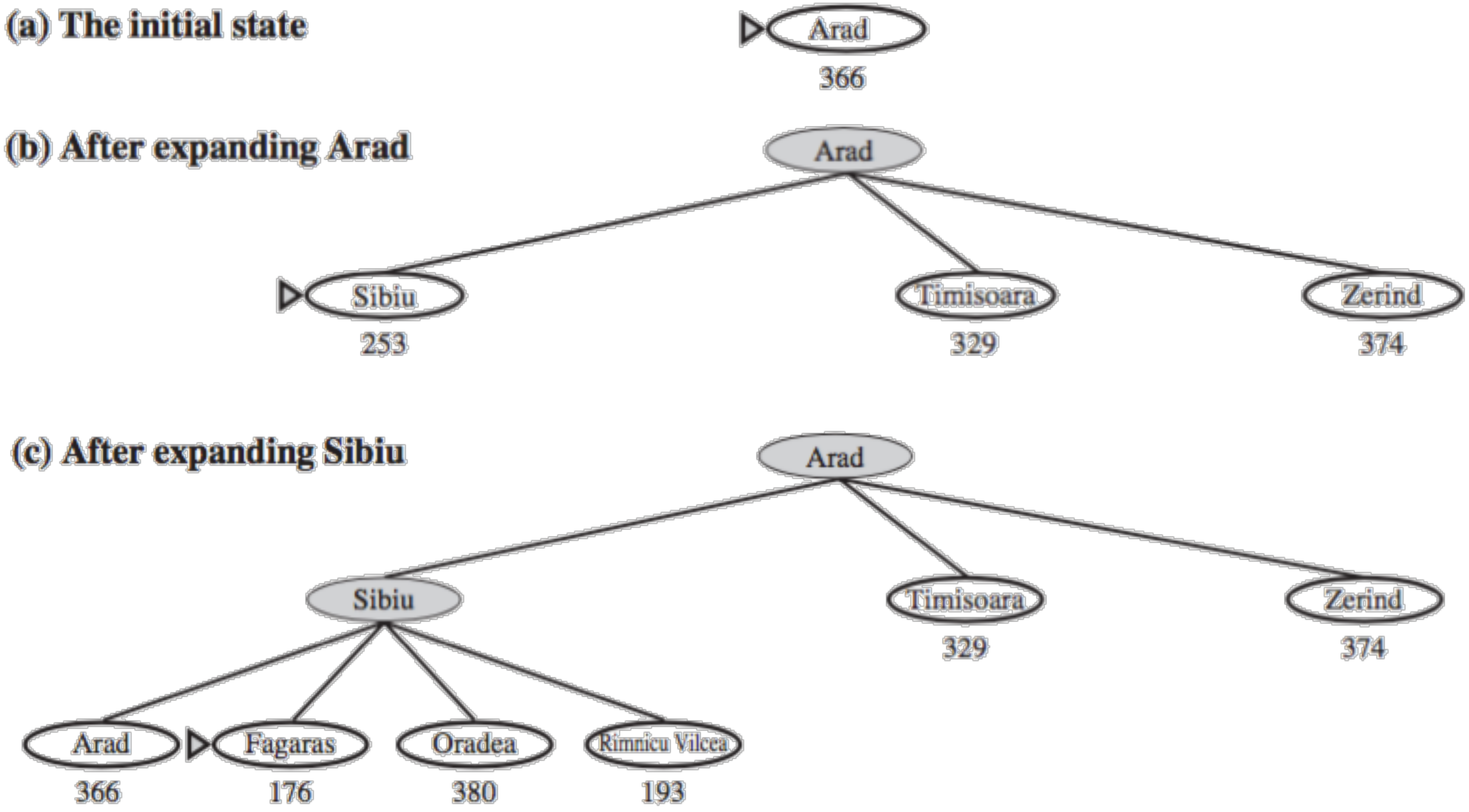
$h_{\text{sld}}(n)$ = فاصله خط مستقیم از شهر n به بخارست

  - نمی‌تواند از تعریف خود مسئله به‌دست آید.
  - وابسته به وضعیت هدف است برای مثال اگر هدف بخارست باشد جدول مقابل را خواهیم داشت.

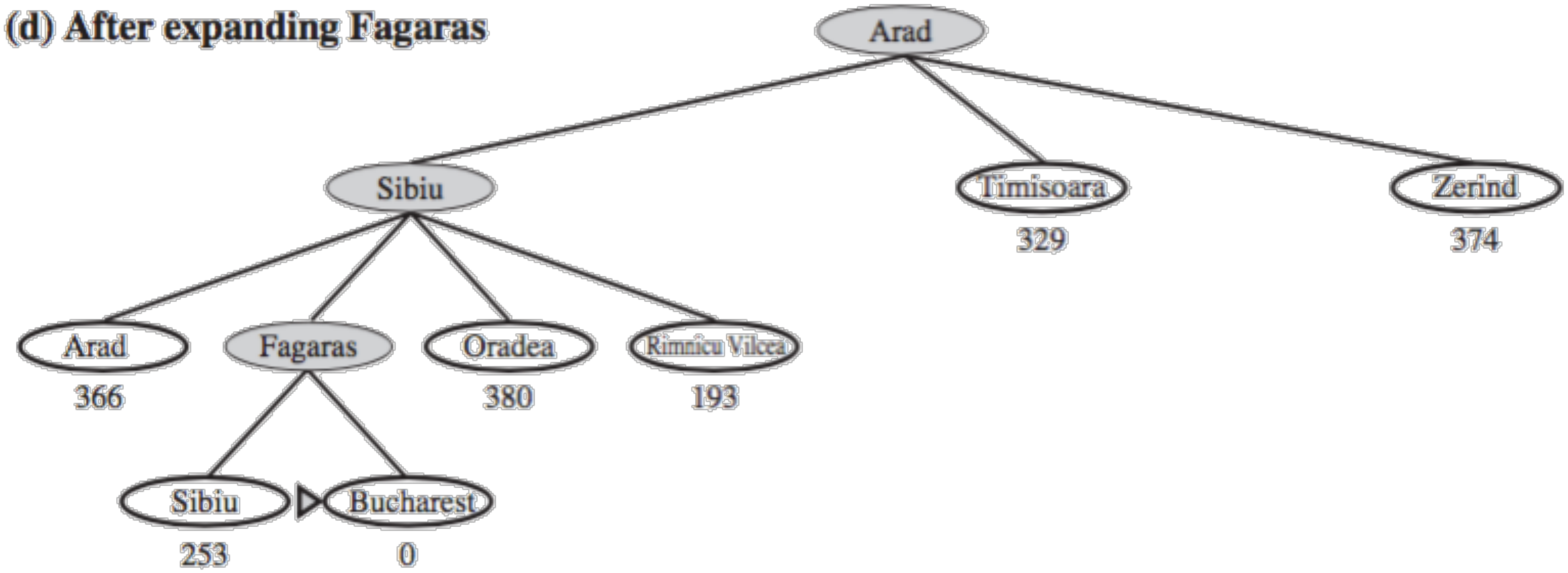


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

جستجوی حریصانه–مثال

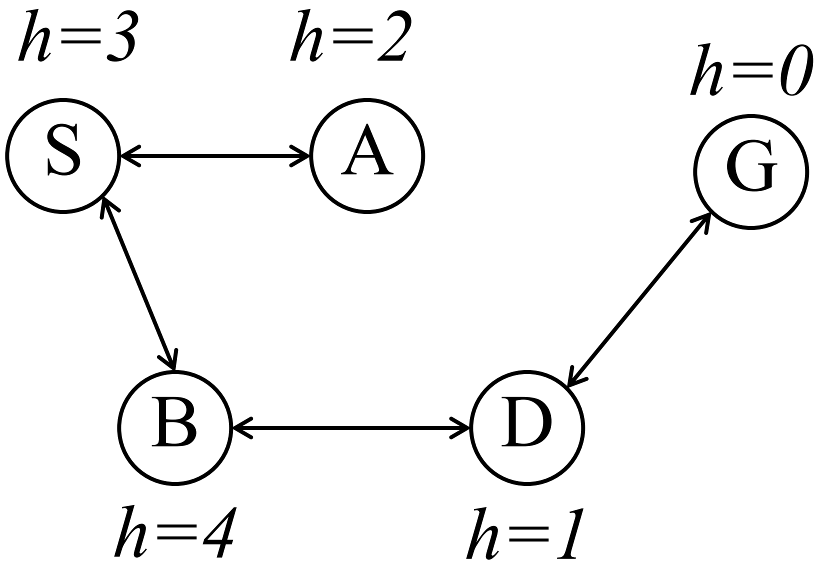


جستجوی حریصانه—مثال



- به هدف رسید!!
- اما بهینه نیست مسیر *Arad, Sibiu, Rimnicu Vilcea, Pitesti* را مشاهده کنید.

ارزیابی جستجوی اول–بهترین حریصانه



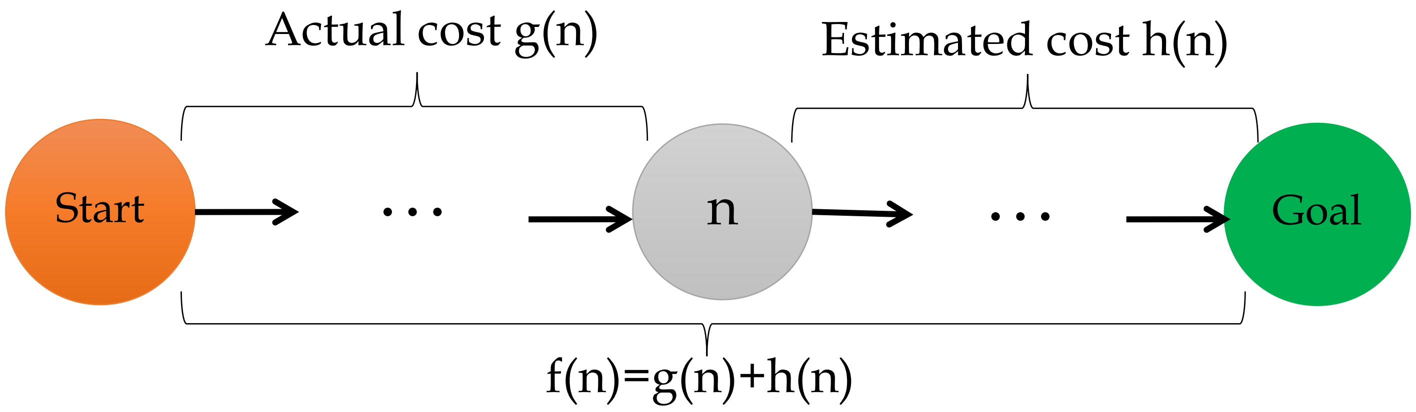
- کامل؟
  - جستجوی درختی: خیر به دلیل حلقه‌های نامتناهی
  - جستجوی گرافی: بله اگر فضای جستجو متناهی باشد.
- بهینگی؟
  - خیر، در ارزیابی هر گره فقط فاصله آن گره تا هدف را ملاک ارزیابی قرار می‌دهد و به فاصله آن گره از ریشه توجهی ندارد. ممکن است گره‌ای که به هدف نزدیک‌تر است فاصله‌اش از ریشه بسیار بیشتر از گره‌های دیگر باشد و در نتیجه هزینه کل مسیر از ریشه تا هدف از طریق آن گره بیشتر شود.

ارزیابی جستجوی اول–بهترین حریصانه

- پیچیدگی زمانی؟
  - در بدترین حالت نمایی است: $O(b^m)$
  - مشابه با جستجوی اول عمق است از این جهت که ترجیح می‌دهد یک مسیر را در تمام طول راه تا هدف دنبال کند، ولی اگر به بن‌بست برسد به عقب برمی‌گردد.
  - با اتخاذ یک تابع هیوریستیک خوب، پیچیدگی به مقدار قابل توجهی می‌تواند کاهش یابد.
  - مقدار کاهش به مسئله و کیفیت هیوریستیک بستگی دارد.
- پیچیدگی فضایی؟
  - همه‌ی گره‌ها را در حافظه نگه می‌دارد: $O(b^m)$

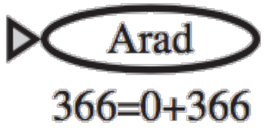
جستجوی A*

- شناخته‌شده‌ترین شکل جستجوی اول–بهترین است.
- تابع ارزیاب: $f(n)=g(n)+h(n)$
  - $g(n)$ هزینه واقعی مسیر از شروع تا گره n
  - $h(n)$ هزینه تخمینی ارزان‌ترین مسیر از نود n تا هدف
  - $f(n)$ هزینه‌ی تخمینی کل مسیر از ریشه درخت تا گره هدف از طریق گره n



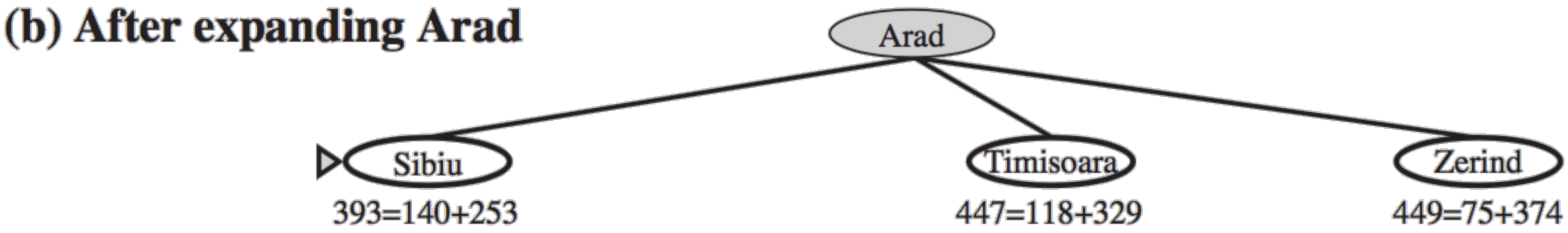
جستجوی A^* – مثال

(a) The initial state



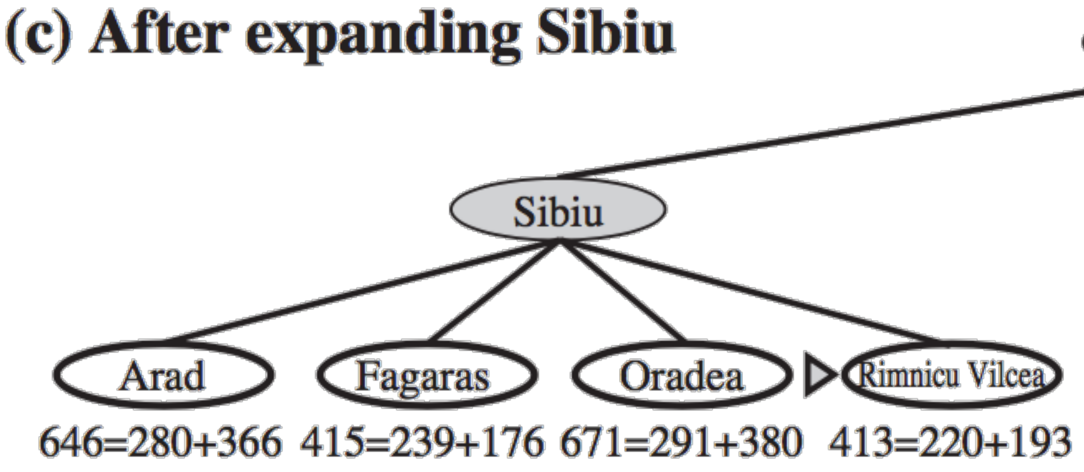
- یافتن Bucharest با شروع از Arad
- $f(Arad) = c(Arad,Arad)+h(Arad)=0+366=366$

جستجوی A^* – مثال



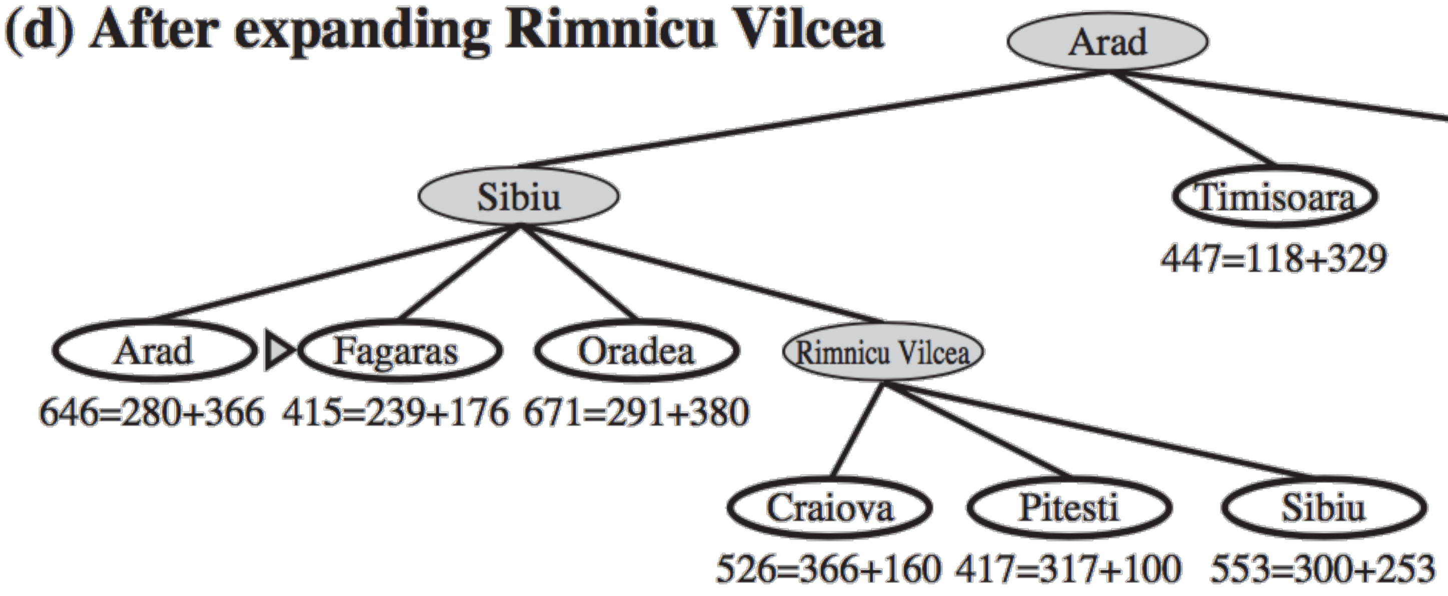
- بسط Arad و تعیین $f(n)$ برای هر گره
  - $f(Sibiu)=c(Arad,Sibiu)+h(Sibiu)=140+253=393$
  - $f(Timisoara)=c(Arad,Timisoara)+h(Timisoara)=118+329=447$
  - $f(Zerind)=c(Arad,Zerind)+h(Zerind)=75+374=449$
- بهترین انتخاب Sibiu است.

جستجوی A* – مثال



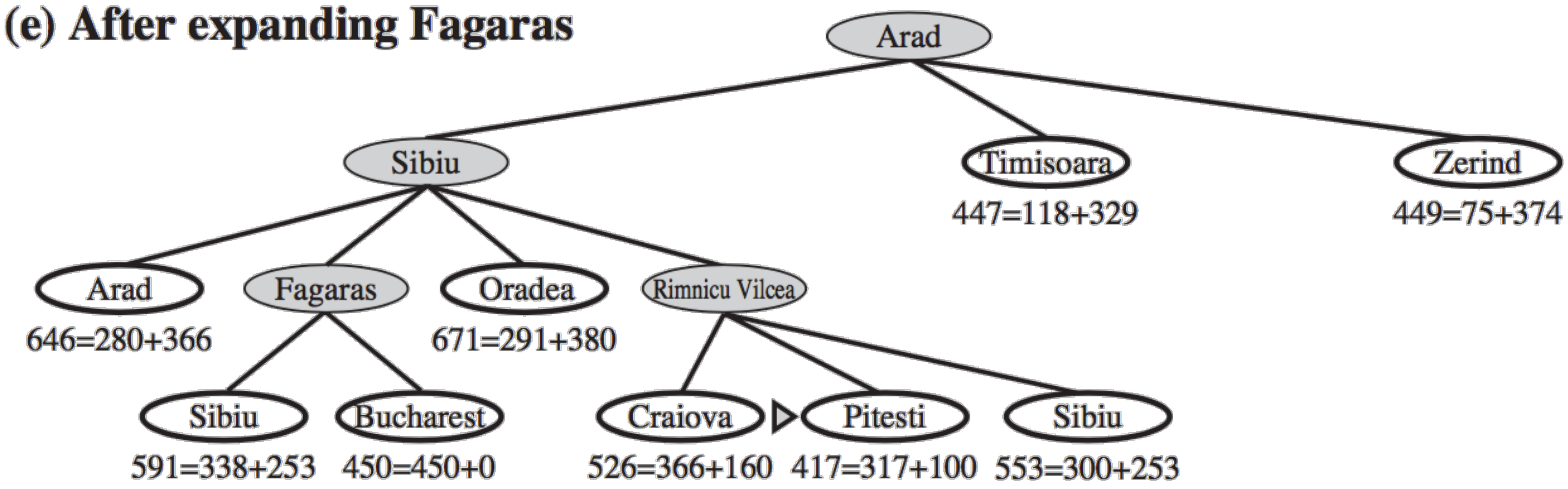
- بسط Sibiu و تعیین $f(n)$ برای هر نود
  - $f(Arad)=c(Sibiu,Arad)+h(Arad)=280+366=646$
  - $f(Fagaras)=c(Sibiu,Fagaras)+h(Fagaras)=239+179=415$
  - $f(Oradea)=c(Sibiu,Oradea)+h(Oradea)=291+380=671$
  - $f(Rimnicu\ Vilcea)=c(Sibiu,Rimnicu\ Vilcea)+ h(Rimnicu\ Vilcea)=220+192=413$
- بهترین انتخاب Rimnicu Vilcea است.

جستجوی A^* – مثال



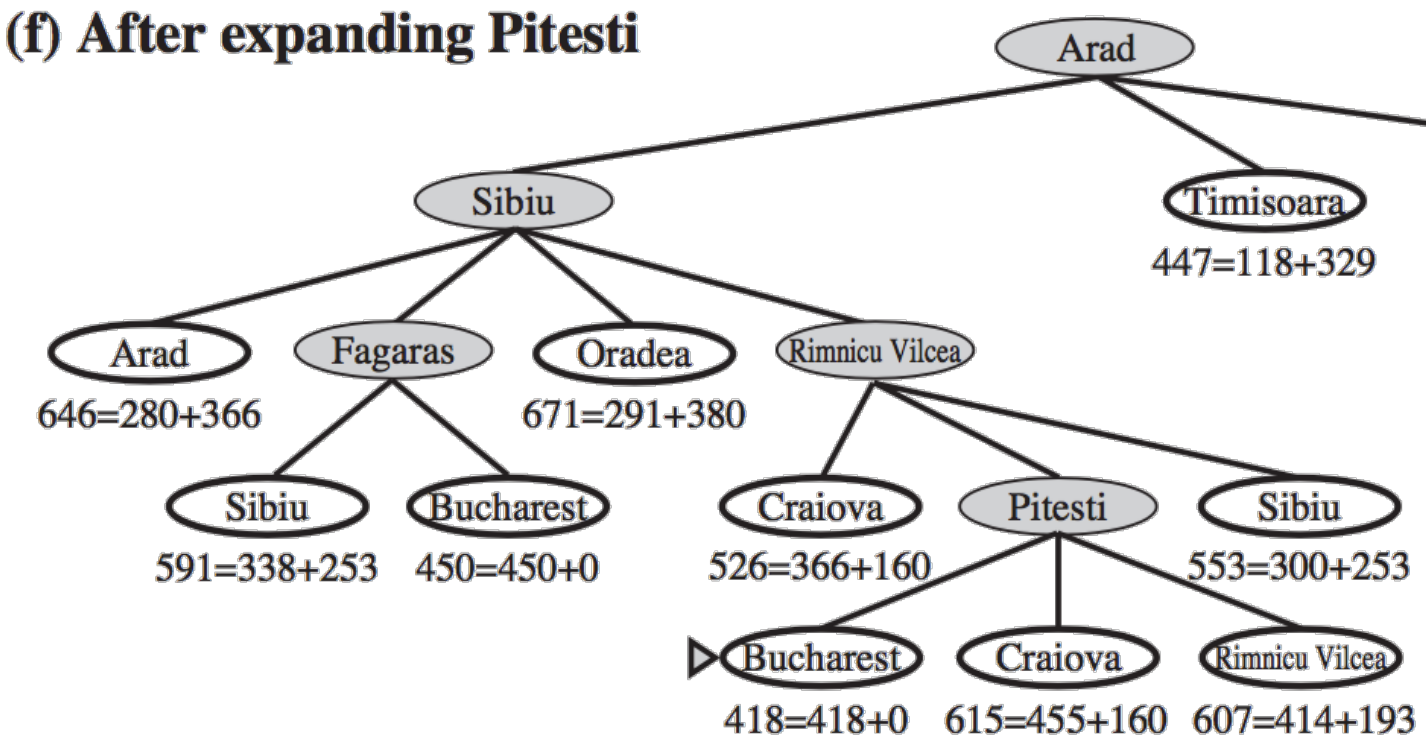
- بسط Rimnicu Vilcea و تعیین $f(n)$ برای هر نود
  - $f(Craiova)=c(Rimnicu\ Vilcea, Craiova)+h(Craiova)=360+160=526$
  - $f(Pitesti)=c(Rimnicu\ Vilcea, Pitesti)+h(Pitesti)=317+100=417$
  - $f(Sibiu)=c(Rimnicu\ Vilcea,Sibiu)+h(Sibiu)=300+253=553$
- بهترین انتخاب Fagaras است.

جستجوی *A – مثال



- بسط Fagaras و تعیین $f(n)$ برای هر نود
  - $f(Sibiu)=c(Fagaras, Sibiu)+h(Sibiu)=338+253=591$
  - $f(Bucharest)=c(Fagaras,Bucharest)+h(Bucharest)=450+0=450$
- بهترین انتخاب Pitesti است!!

جستجوی *A – مثال



- بسط Pitesti و تعیین $f(n)$ برای هر نود
  - $f(Bucharest)=c(Pitesti,Bucharest)+h(Bucharest)=418+0=418$
- بهترین انتخاب Bucharest است!!
- به مقادیر f در طول مسیر بهینه توجه کنید!!

شرط بهینگی جستجوی درختی *A

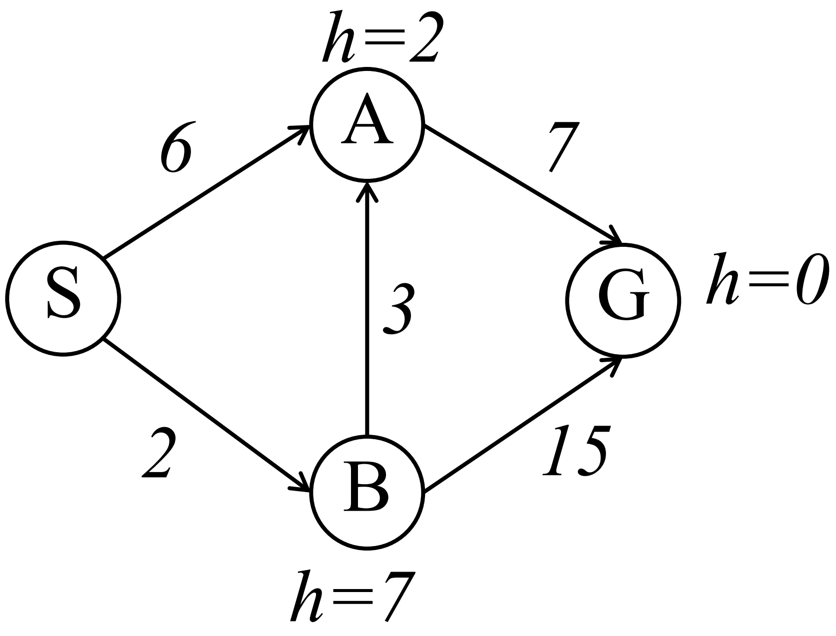
- قابل قبول بودن Admissibility
  - هیوریستیک $h(n)$ قابل قبول است اگر هزینه مسیر هر گره تا هدف را بیشتر از مقدار واقعی تخمین نزند.
  - $h(n)$ یک حد پایین روی هزینه مسیر از n تا هدف است اگر برای هر گره n

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

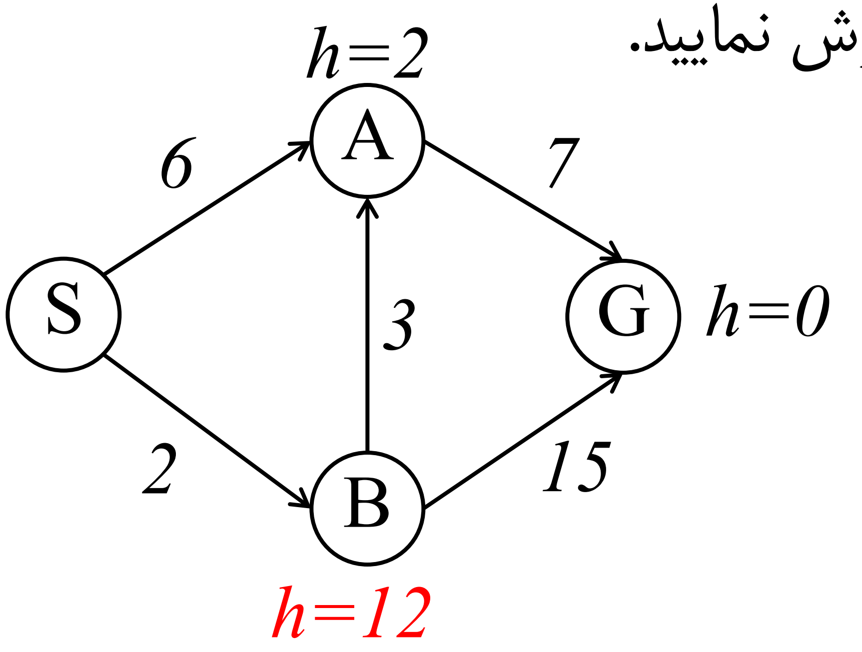
  که $h^*(n)$ هزینه واقعی ارزان‌ترین مسیر به وضعیت هدف از گره n است.
  - برای مثال: $h_{SLD}(n) \leq$ "فاصله واقعی جاده"
  - می‌توان گفت هر تابع هیوریستیک قابل قبول، هزینه هر گره تا هدف را به‌طور خوش‌بینانه یا optimistic تخمین می‌زند.

قابل قبول بودن در مقابل غیرقابل قبول بودن

تمرین: جستجوی درختی A^* را بر روی هر یک از گراف‌های زیر انجام دهید و مسیر حاصل را گزارش نمایید.



Admissible heuristic
Path: S-B-A-G



Inadmissible heuristic
Path: S-A-G

اثبات بهینگی جستجوی درختی A^*

- قضیه: اگر $h(n)$ قابل قبول باشد، A^* با استفاده از جستجوی درختی بهینه خواهد بود.
  - اثبات: فرض کنید هدف نیمه‌بهینه G_2 در frontier قرار دارد و گره n نودی بسط نیافته باشد که در مسیر هدف بهینه G قرار دارد.

1) $h(G_2)=0 \rightarrow f(G_2)=g(G_2)$

2) $h(G)=0 \rightarrow f(G)=g(G)$

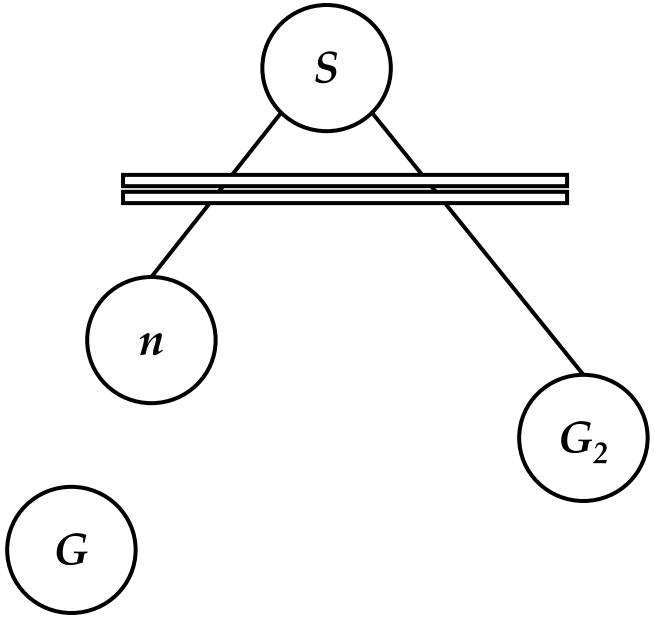
3) G_2 *is suboptimal* $\rightarrow g(G_2)>g(G)$

$\Rightarrow f(G_2)>f(G)$

4) h *is admissible* $\rightarrow h(n) \le h^*(n)$

$\rightarrow g(n)+h(n) \le g(n) + h^*(n)$

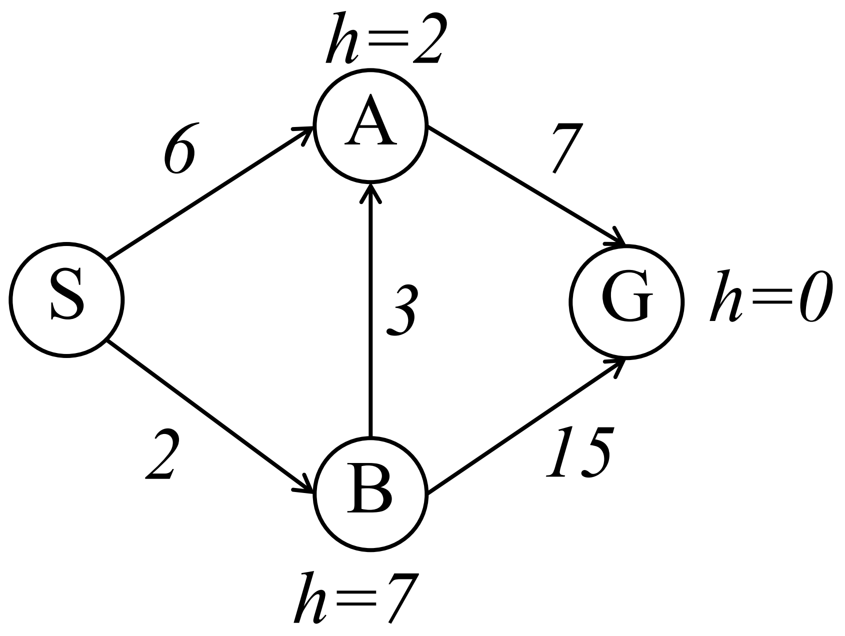
$\rightarrow f(n) \le f(G) \Rightarrow f(n)<f(G_2)$



- در این‌صورت G_2 هرگز برای بسط انتخاب نمی‌شود.

شرط بهینگی جستجوی گرافی *A

- آیا قابل قبول بودن هیوریستیک، بهینگی جستجوی گرافی *A را تضمین می‌کند؟
  - باید توجه کرد که گراف جستجو مسیر بهینه به یک حالت تکراری را کنار می‌گذارد.
- برای مثال در مورد گراف زیر مسیر نیمه‌بهینه S-A-G توسط جستجوی گرافی به‌دست خواهد آمد.

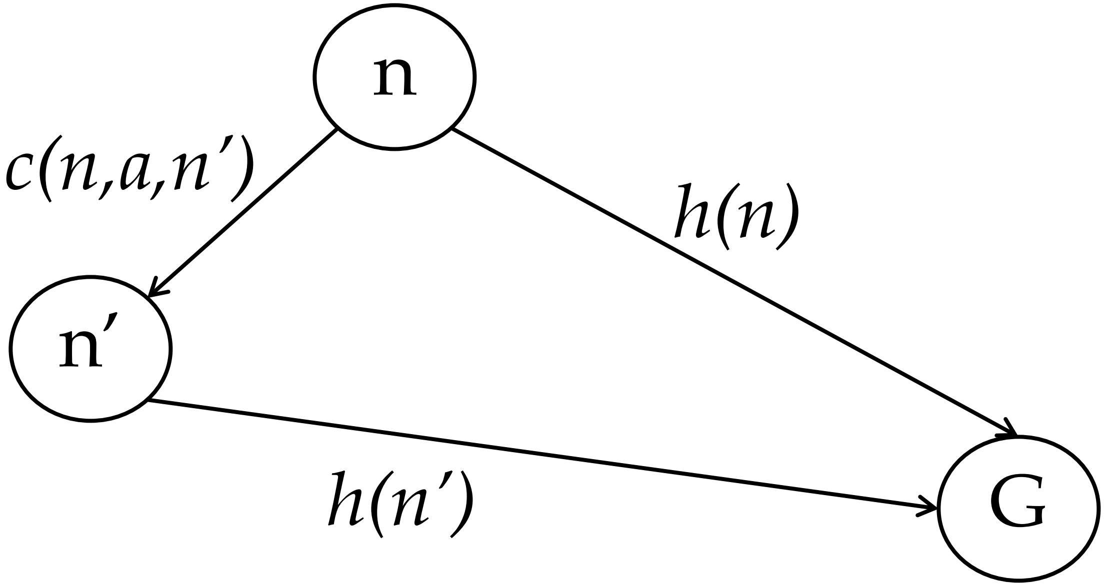


شرط بهینگی جستجوی گرافی A*

- سازگاری Consistent
  - هیوریستیک $h(n)$ سازگار است اگر برای هر گره n و هر پسین آن مانند n' که با انجام عمل a به آن برسیم داشته باشیم:

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

که در آن $c(n,a,n')$ برابر با هزینه مرحله‌ای رفتن از n به n' با انجام عمل a است.



ارتباط قابل قبول بودن و سازگار بودن

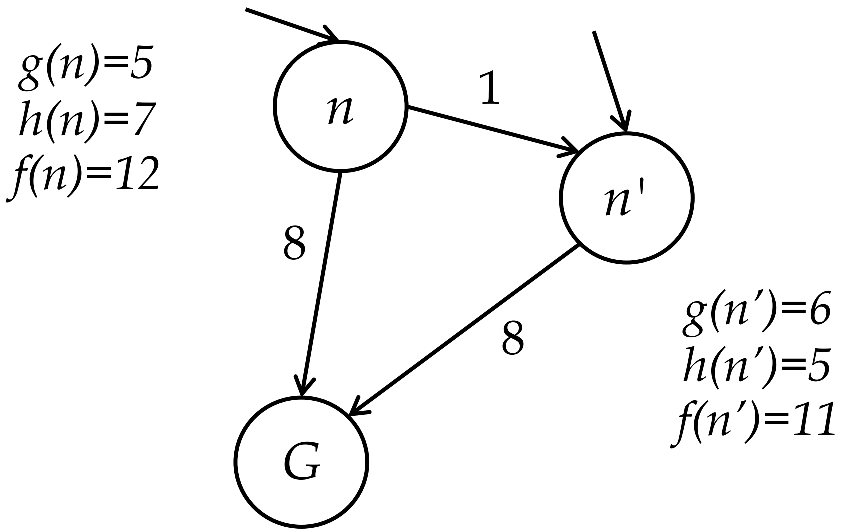
- سازگاری ← قابل قبول بودن
  - همه‌ی توابع هیوریستیک سازگار قابل قبول هستند.

$$n_1 \xrightarrow{c(n_1,a_1,n_2)} n_2 \xrightarrow{c(n_2,a_2,n_3)} n_3 \longrightarrow \cdots \longrightarrow n_k \xrightarrow{c(n_k,a_k,G)} G$$

$$
\begin{aligned}
h(n_1) &\le c(n_1,a_1,n_2) + h(n_2) \\
&\le c(n_1,a_1,n_2) + c(n_2,a_2,n_3) + h(n_3) \\
&\quad \ldots \\
&\le c(n_1,a_1,n_2) + \ldots + c(n_k,a_k,G) + h(G) \\
&\le c(n_1,a_1,n_2) + \ldots + c(n_k,a_k,G) \\
&\le \text{cost of each path from } n_1 \text{ to goal}
\end{aligned}
$$

ارتباط قابل قبول بودن و سازگار بودن

- یک هیوریستیک ممکن است قابل قبول باشد اما سازگار نباشد.



$c(n,a,n')=1$
$h(n)=7$
$h(n')=5$
$h(n) \not\leq h(n') + c(n,a,n')$

- مقدار f برای هیوریستیک‌های قابل قبول ممکن است در طول مسیر کاهش یابد.
- اکثر هیوریستیک‌های قابل قبول در کاربردهای عملی سازگار نیز هستند.
- تمرین: راهی پیشنهاد دهید که هیوریستیک h را سازگار نماید.

اثبات بهینگی جستجوی گرافی A^*

- قضیه: اگر $h(n)$ سازگار باشد، A^* با استفاده از جستجوی گرافی بهینه خواهد بود.
- لم ۱: اگر $h(n)$ سازگار باشد آن‌گاه مقدار $f(n)$ در طول هر مسیری غیرنزولی است.
  - اثبات: فرض کنید n' یک پسین از n باشد

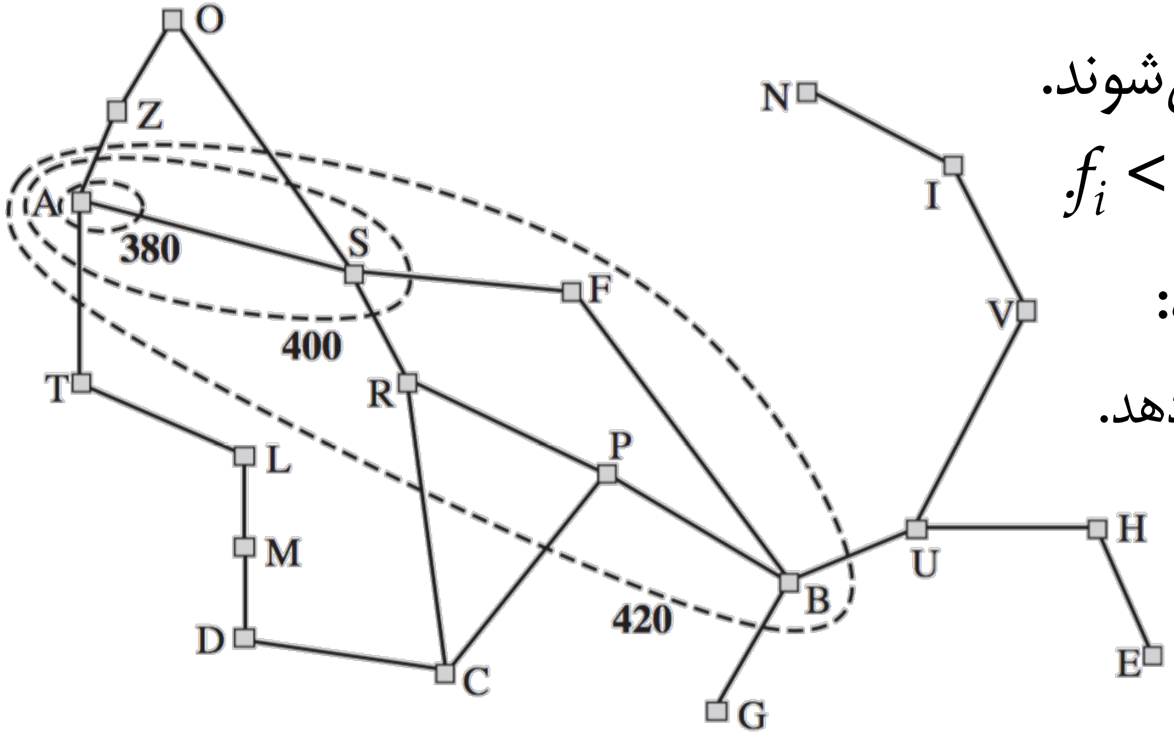
$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n,a,n') + h(n') \\
&\geq g(n) + h(n) \\
&\geq f(n)
\end{aligned}
$$

اثبات بهینگی جستجوی گرافی A^*

- لم ۲: اگر A^* گره n را برای بسط انتخاب کند، راه‌حل بهینه تا آن گره پیدا شده است.
  - اثبات با استفاده از برهان خلف: فرض کنید هنگامی که گره n برای بسط انتخاب می‌شود، مسیر بهینه از ریشه تا n به‌دست نیامده باشد آنگاه باید بتوان از طریق گره دیگری مانند n' که در مجموعه‌ی frontier فعلی قرار دارد با یک مسیر بهینه به حالت موجود در n رسید. علاوه‌براین براساس لم ۱، $f(n') \leq f(n)$ و بنابراین n' باید زودتر انتخاب می‌شد.
- اولین گره‌ی هدفی که برای بسط انتخاب می‌شود باید یک راه‌حل بهینه باشد (برای گره‌های هدف $h=0$ بوده و مقدار f برابر با هزینه‌ی واقعی مسیر می‌باشد).
- دنباله‌ی گره‌های بسط داده شده توسط A^* با استفاده از جستجوی گرافی به‌ترتیب غیرنزولی $f(n)$ می‌باشد.

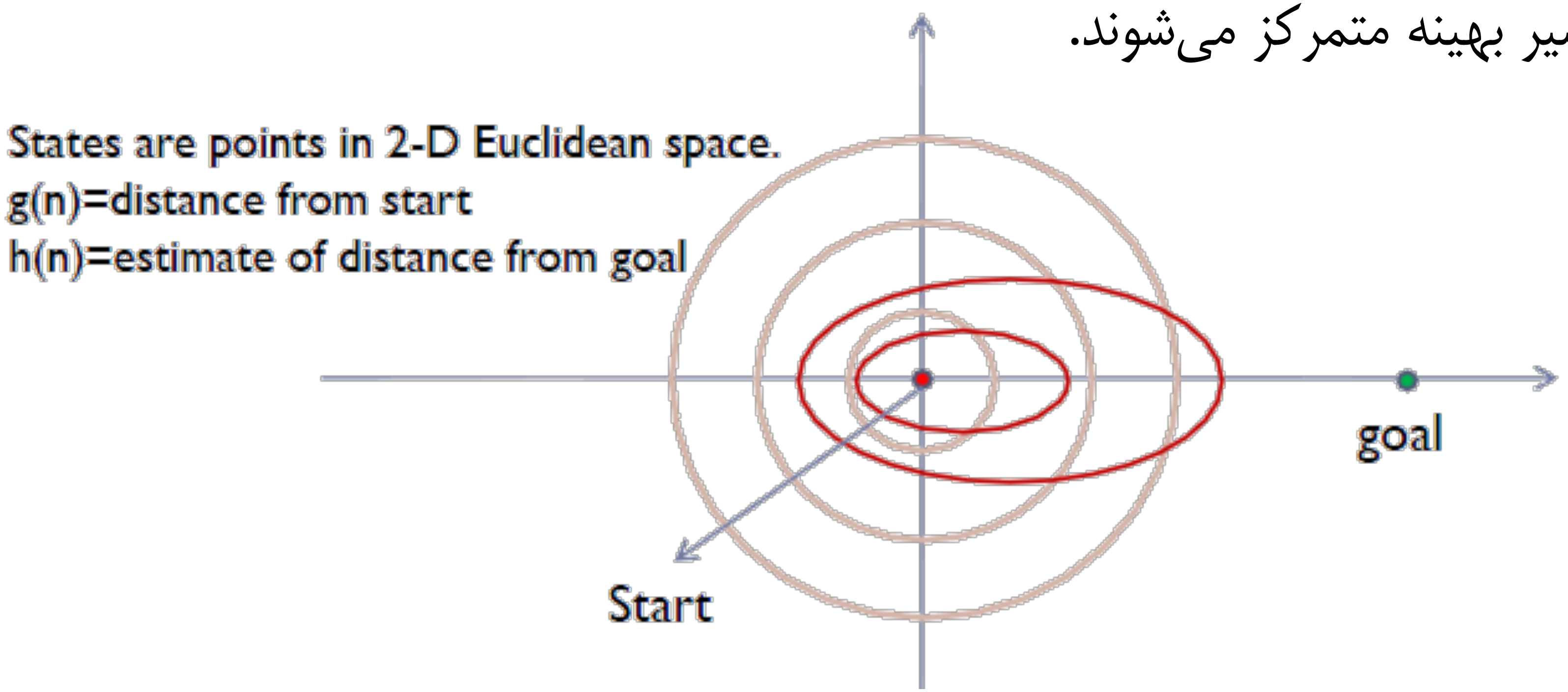
Contours –A* کانتورهای

- A* گره‌ها را به‌ترتیب افزایش مقدار f بسط می‌دهد.
- کانتورهای f به‌تدریج در فضای حالت اضافه می‌شوند.
  - کانتور i، همه‌ی گره‌های با $f=f_i$ را دارد که $f_i < f_{i+1}$.
- اگر C^* هزینه‌ی مسیر راه‌حل بهینه باشد آن‌گاه:
  - A* همه‌ی گره‌های با $f(n) < C^*$ را بسط می‌دهد.
  - A* برخی گره‌هایی که روی «کانتور هدف» ($f(n) = C^*$) هستند را بسط می‌دهد.
  - A* هیچ گره‌ای با $f(n) > C^*$ را بسط نمی‌دهد. (هرس)



کانتورهای *A در مقابل UCS

- در جستجوی هزینه یکنواخت (جستجوی *A با $h(n)=0$) نوارهای پیرامون گره آغازین به شکل دایره خواهند بود.
- *A باعث می‌شود ترازها نامنظم شوند.
  - هرچه $h(n)$ هیوریستیک دقیق‌تری باشد، نوارها به سمت گره هدف کشیده می‌شوند و به‌صورت باریک‌تری پیرامون مسیر بهینه متمرکز می‌شوند.



ارزیابی جستجوی A^*

- کامل؟
  - همان‌طور که نوارهای با f در حال افزایش را اضافه می‌کنیم، باید بالاخره به نواری برسیم که در آن‌جا f مساوی هزینه‌ی مسیر تا یک حالت هدف است.
  - بنابراین اگر تعداد گره‌ها با $f < f(G)=C^*$ متناهی باشد کامل خواهد بود.
    - فاکتور انشعاب متناهی بوده و هزینه‌ی یال‌ها از ε بیشتر باشد.
- پیچیدگی زمانی؟
  - برای اغلب مسائل، تعداد گره‌ها در داخل کانتور هدف (یعنی گره‌هایی با $f(n) \le C^*$) برحسب طول راه حل نمایی است.
  - شرط نمایی نبودن تعداد گره‌ها: خطای تابع هیوریستیک، سریعتر از لگاریتم هزینه واقعی مسیر رشد نکند.

$$\left| h(n) - h^*(n) \right| \le O(\log h^*(n))$$

ارزیابی جستجوی A*

- پیچیدگی فضایی؟
  - نمایی است، A* تمام گره‌ها را در حافظه نگه می‌دارد.
  - فضا مشکل اصلی‌تری نسبت به زمان است.
  - برای جبران این مشکل، نسخه‌های دیگر A* مانند IDA*,RBFS,SMA* مطرح شد.
- بهینگی؟
  - بله، قبلا اثبات کردیم!

- برای هر تابع هیوریستیک سازگار، A* کارآمد بهینه (optimally efficient) است.
  - یعنی هیچ الگوریتم بهینه دیگری که از همان تابع هیوریستیک استفاده کند، تضمین نمی‌کند تعداد گره‌های کمتری نسبت به A* بسط دهد.

تست ۱

فضای زیر را در نظر بگیرید که عامل در هر خانه می‌تواند یکی از چهار حرکت رفتن به بالا، پایین، چپ یا راست را انجام دهد. خانه شماره ۱ وضعیت شروع و خانه شماره ۱۱ وضعیت هدف است. همین‌طور خانه ۸ مسدود است. اگر عامل حرکتی انجام دهد که به خانه ۸ یا دیوارها برخورد کند سر جایش باقی می‌ماند. فرض کنید هر یک از حرکت‌ها یک واحد هزینه دارد. اگر در هر گره از فاصله منهتن آن گره تا هدف به‌عنوان مقدار تابع اکتشافی (هیوریستیک) استفاده شود، سه گره اولی که در الگوریتم A^* گسترش می یابند کدام‌اند؟ اگر شرایطی پیش آمد که دو خانه برای گسترش دقیقا وضعیت یکسانی (از نظر A^*) داشته باشند، خانه با شماره کوچک‌تر انتخاب می‌شود. (کامپیوتر ۹۵)

۱) ۱، ۲، ۳ ✓۲) ۱، ۲، ۵ ۳) ۱، ۴، ۷ ۴) ۱، ۴، ۵

۳	۶	۹	۱۲
۲	۵	۸	۱۱
۱	۴	۷	۱۰

جستجوهای هیوریستیکی با حافظه محدود

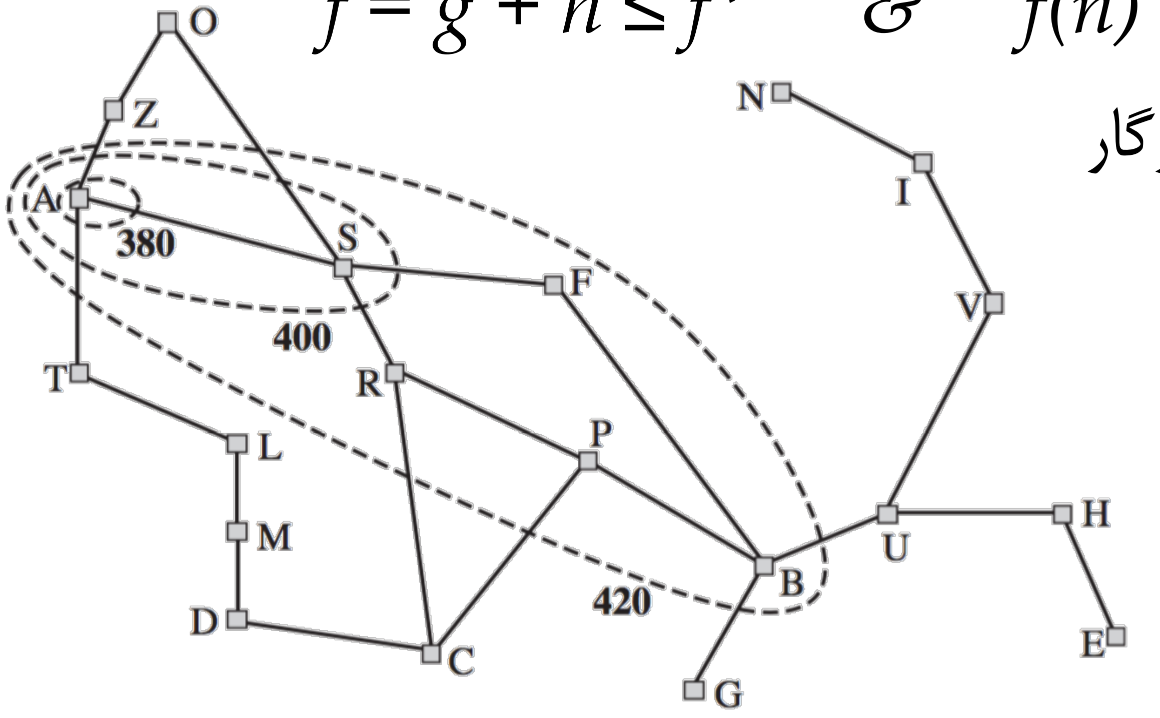
IDA*, RBFS, SMA*

ایده

- الگوریتم *A نیاز به نگه‌داری مجموعه‌ی frontier (و مجموعه‌ی explored) دارد. این موضوع باعث نیاز به حافظه‌ی خیلی زیادی می‌شود.
- ایده استفاده از f یکنوا است یعنی:

$$f = g + h \leq f^* \qquad \& \qquad f(n) \leq f(\text{next node after } n)$$

- آیا هنگامی که h قابل‌قبول باشد ولی سازگار نباشد می‌توان f را یکنوا کرد؟ چطور؟



جستجوی عمیق‌شونده‌ی تکراری A^*

- IDA^*، همانند الگوریتم ILS، از ایده‌ی عمقی تکراری استفاده می‌کند.
  - مکرراً درخت جستجو را به صورت عمقی می‌سازد.
- در هر تکرار، هر شاخه را تا جایی بسط می‌دهد که هزینه‌ی $f(n)$ گره‌های آن از مقدار خاصی (مقدار برش) بیشتر نشود.
  - در هر تکرار درخت جستجو مجدداً از ریشه ساخته می‌شود (همانند IDS).
- مقدار برش در تکرار جدید برابر کم‌ترین مقدار گره‌ای قرار داده می‌شود که از مقدار برش در تکرار قبلی بیشتر باشد.
- IDA^* به همراه جستجوی درختی (و نه گرافی) موجب بهبود پیچیدگی فضایی A^* می‌شود.

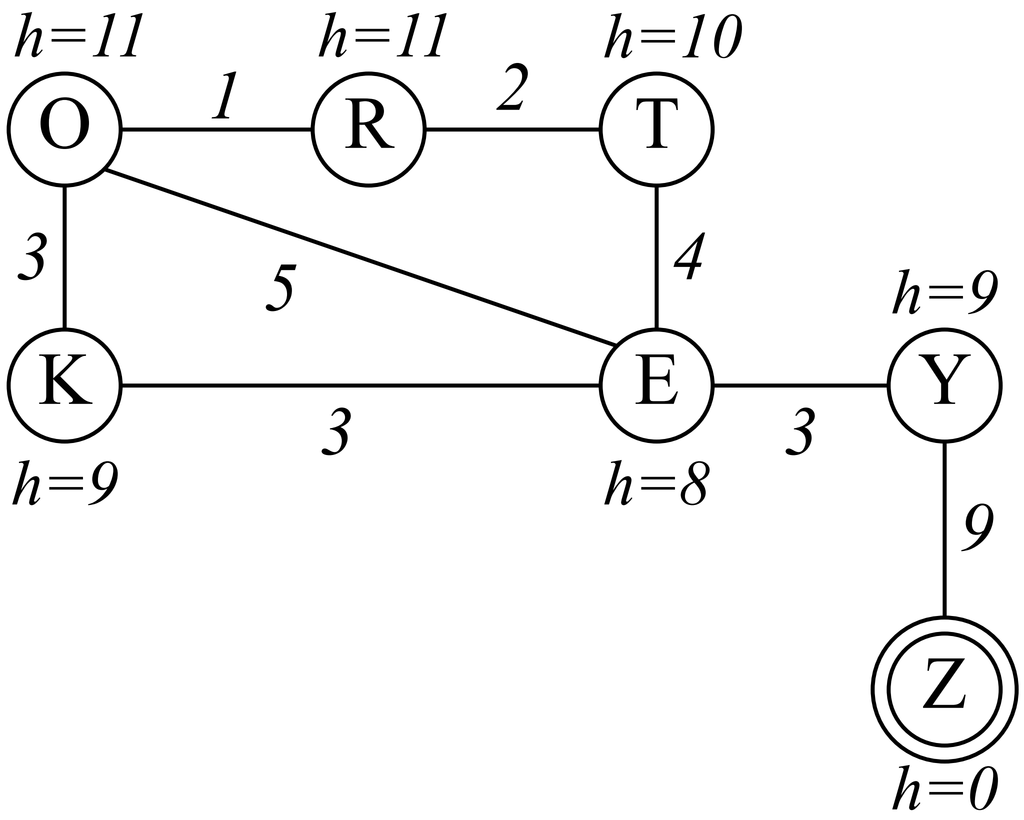
الگوریتم عمیق‌شونده‌ی تکراری A*

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

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

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

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

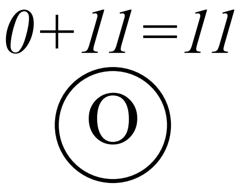
مثال IDA*

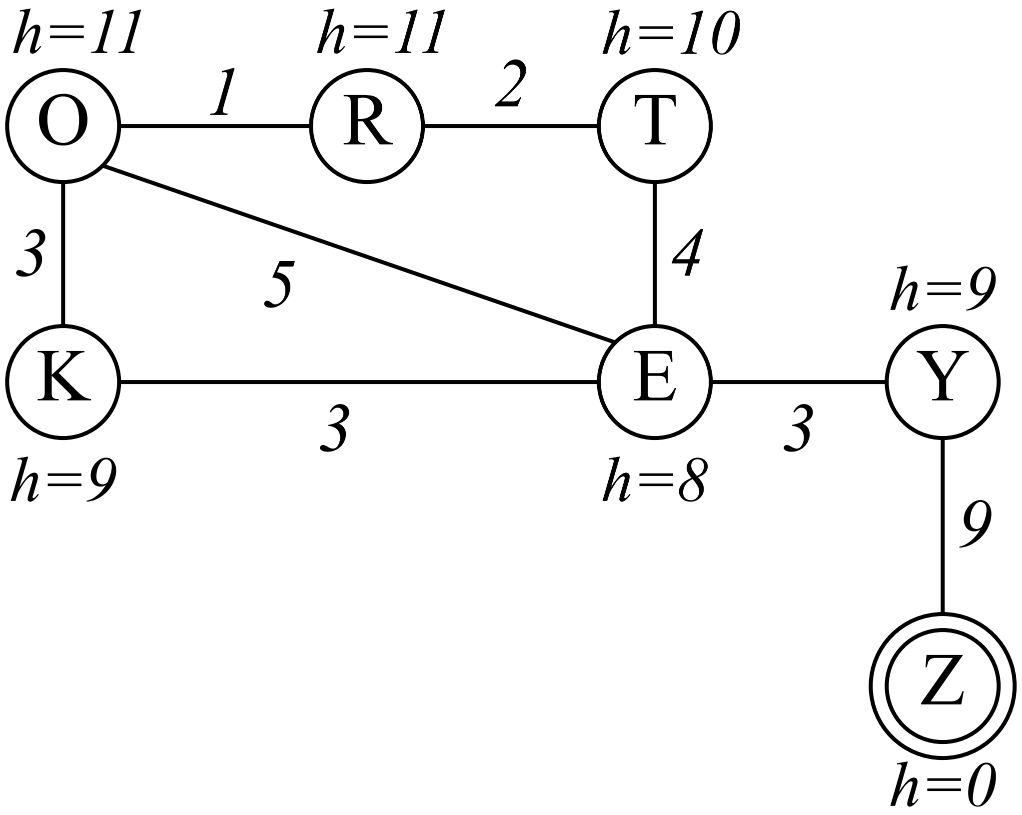
- برای یادآوری نکات جستجوی گرافی، در مثال زیر از IDA* **به همراه جستجوی** گرافی استفاده می‌کنیم.



مثال IDA*

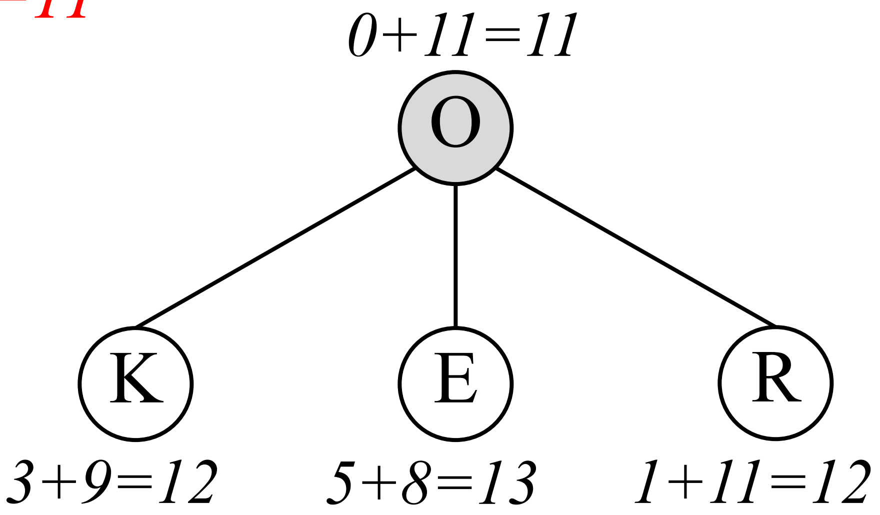
F-limit=11

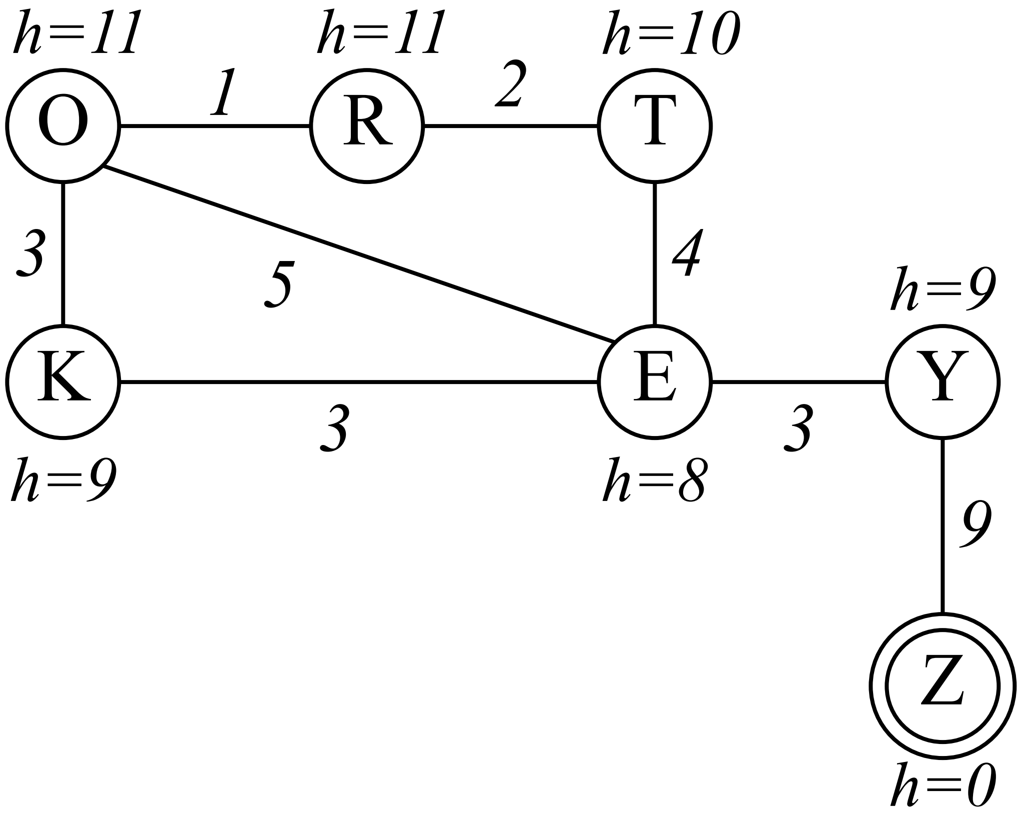




مثال IDA*

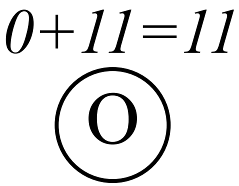
F-limit=11

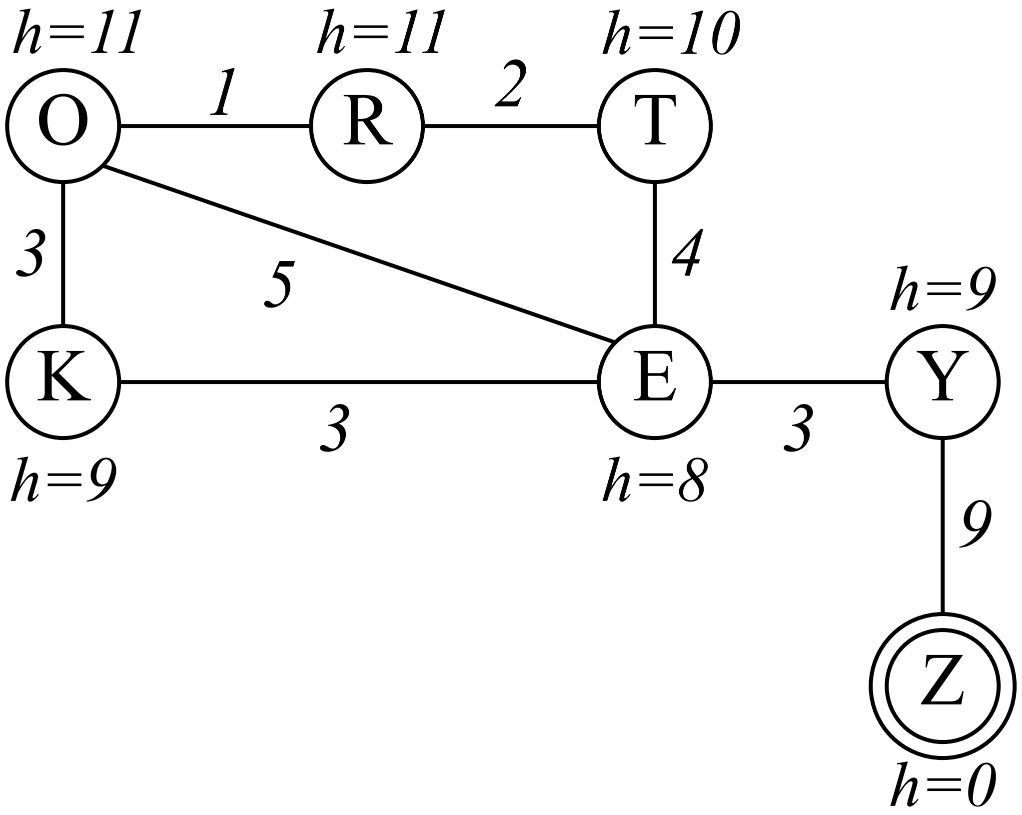




مثال IDA*

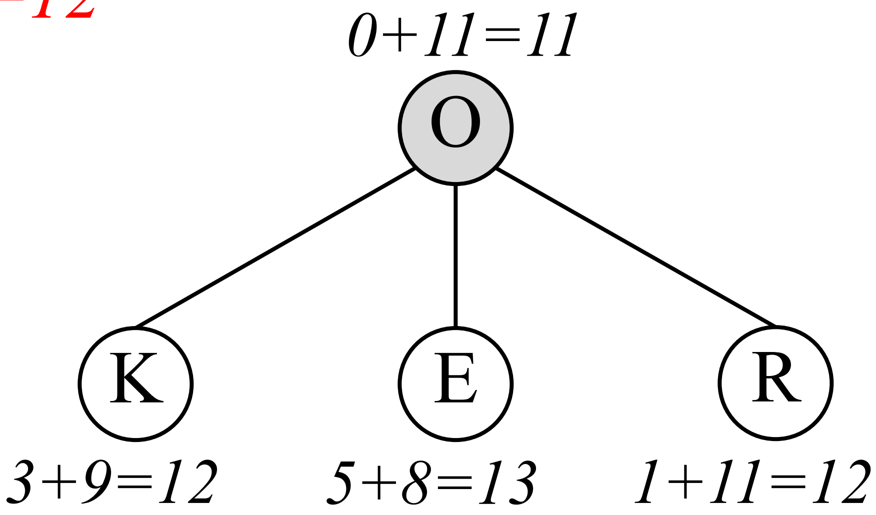
F-limit=12

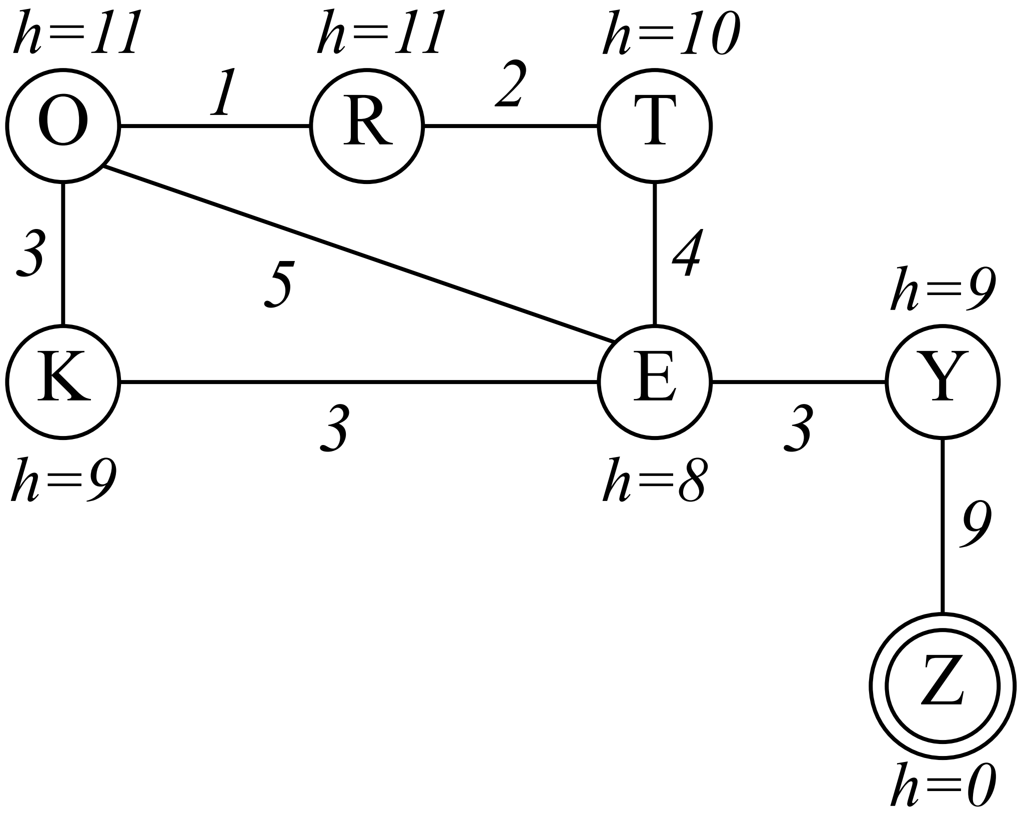




مثال IDA*

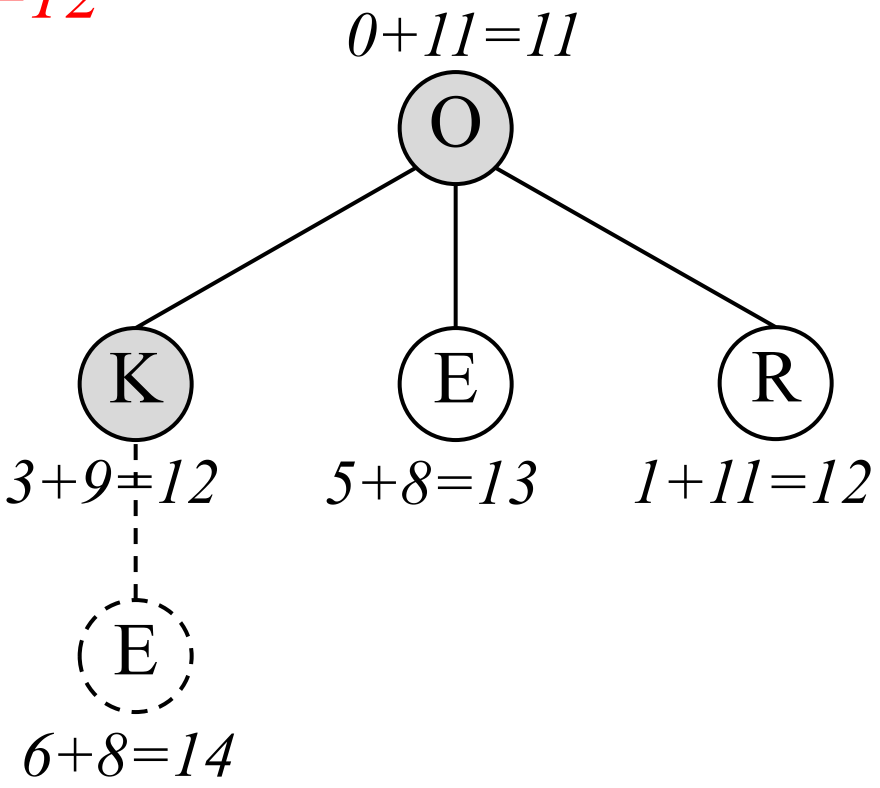
F-limit=12

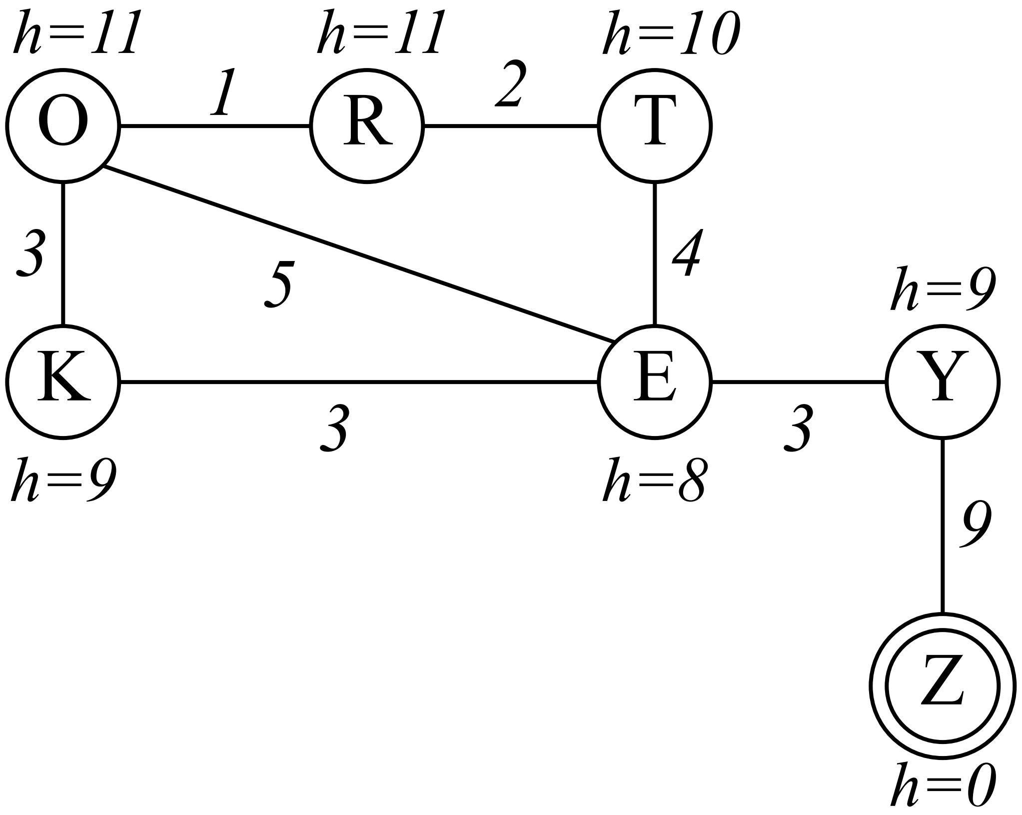




مثال IDA*

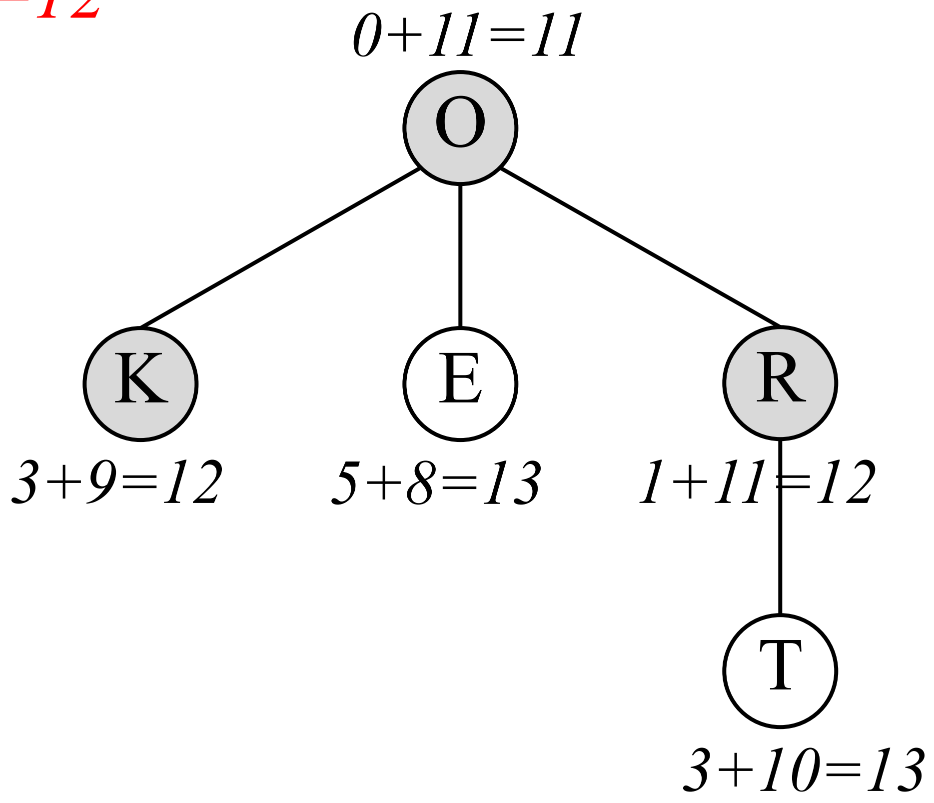
F-limit=12

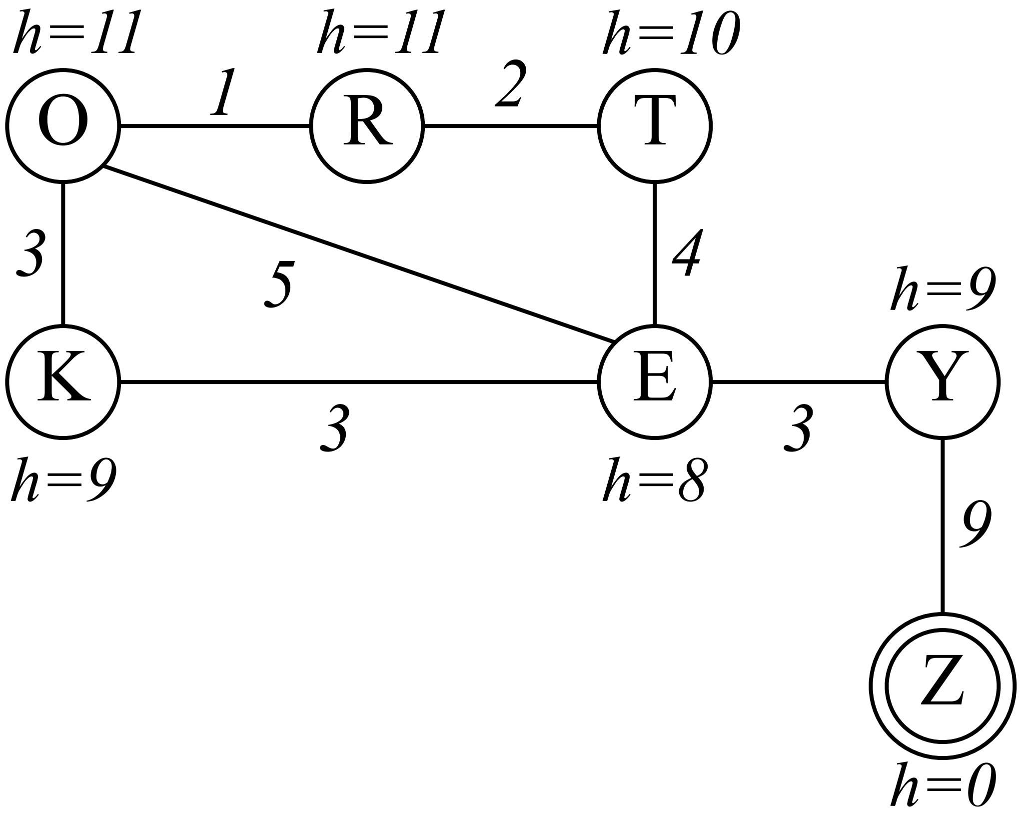




مثال IDA*

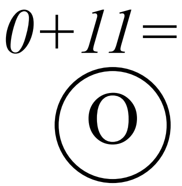
F-limit=12

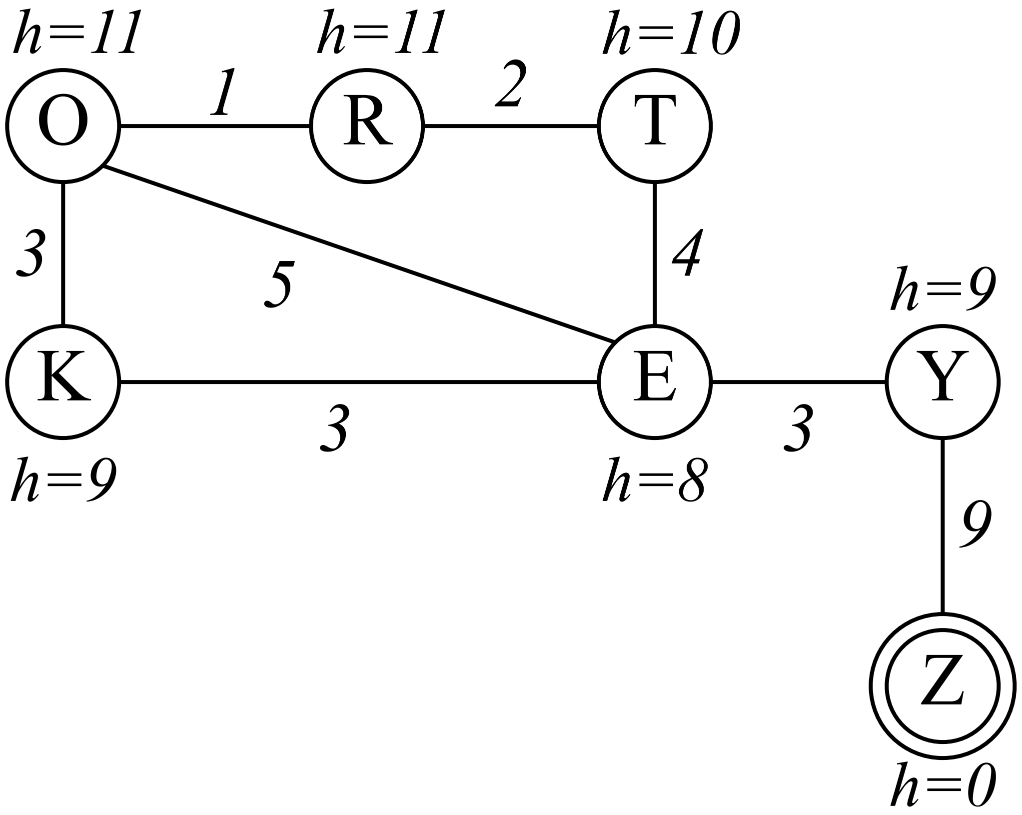




مثال IDA*

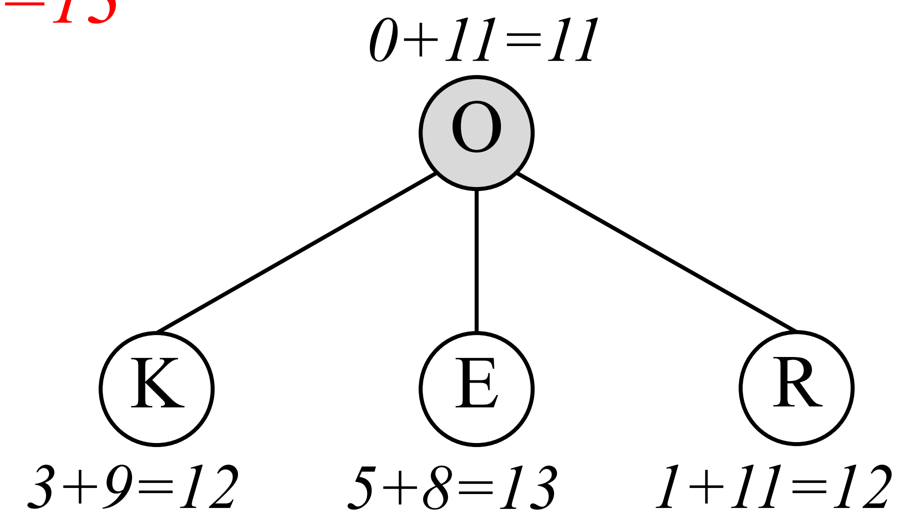
F-limit=13

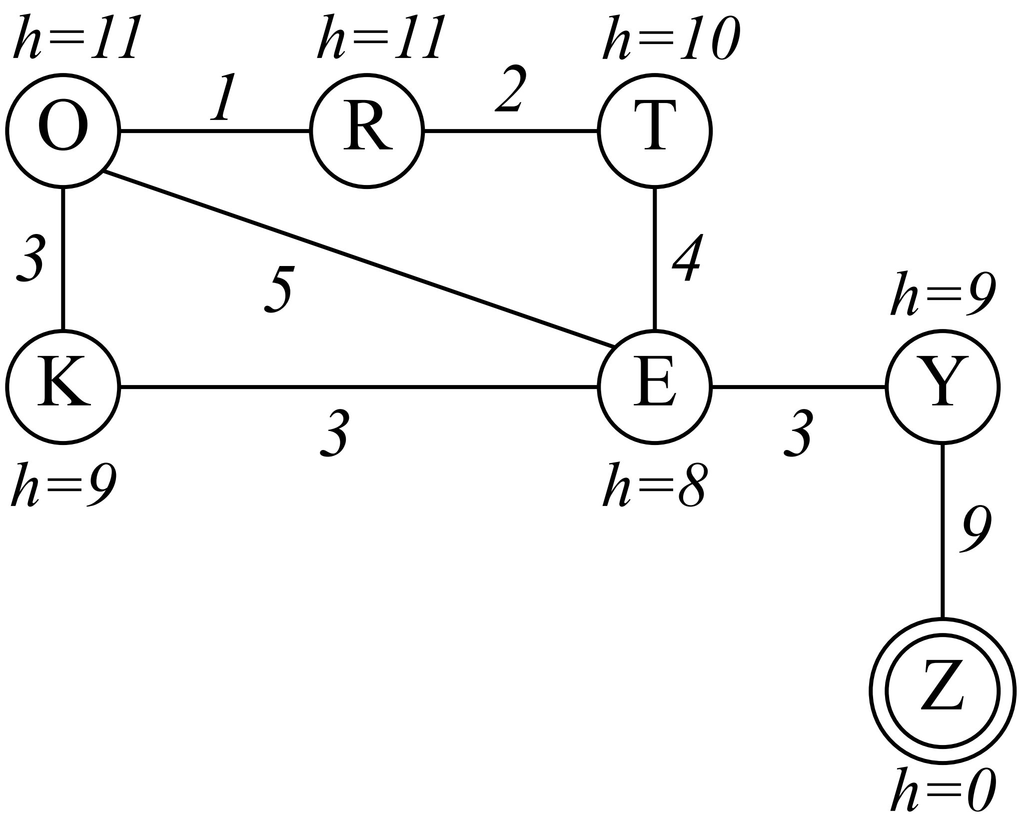




مثال IDA*

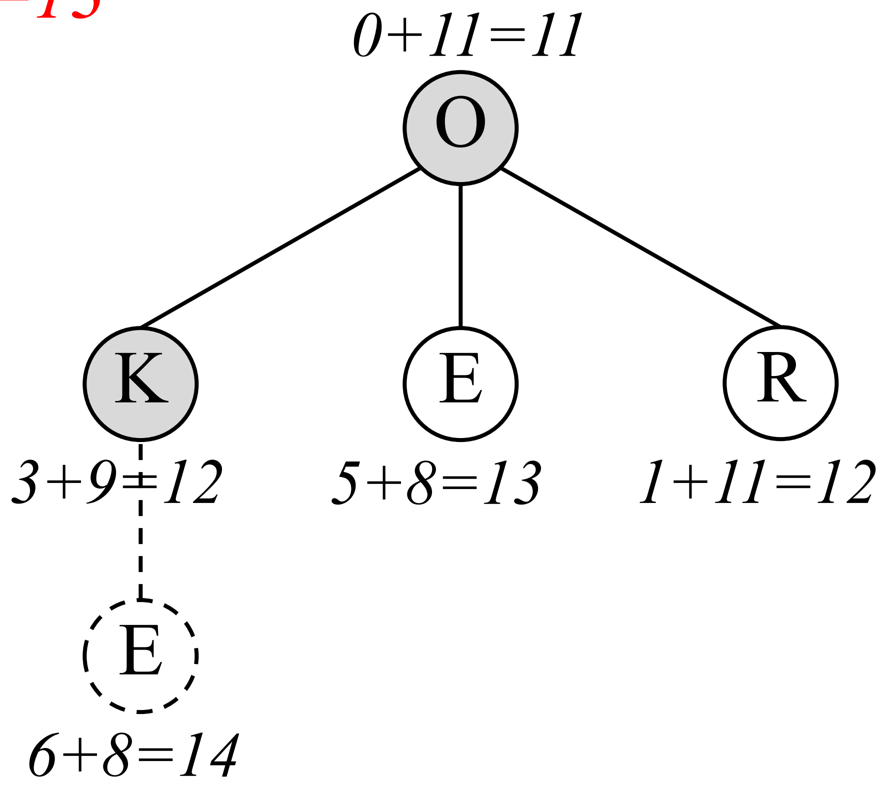
F-limit=13

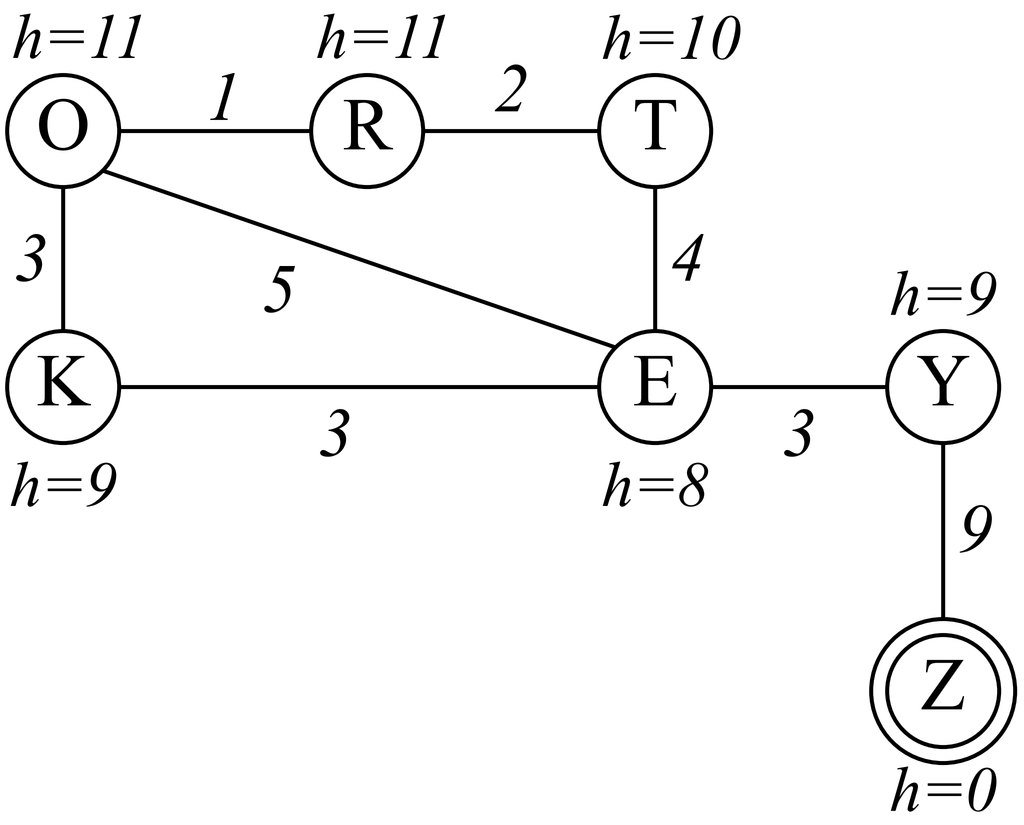




مثال IDA*

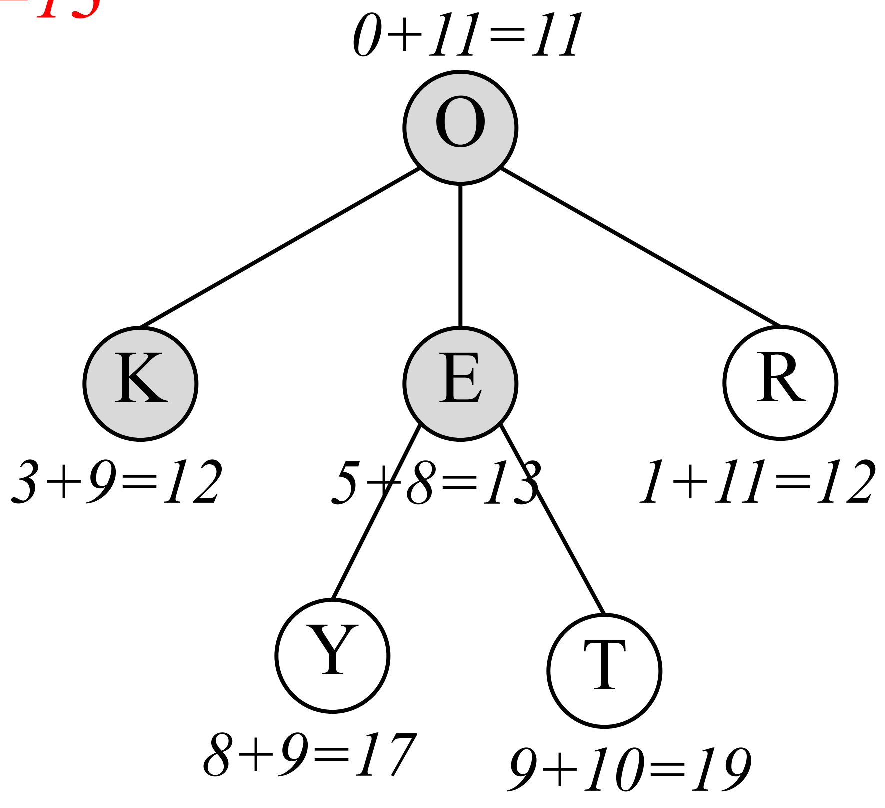
F-limit=13

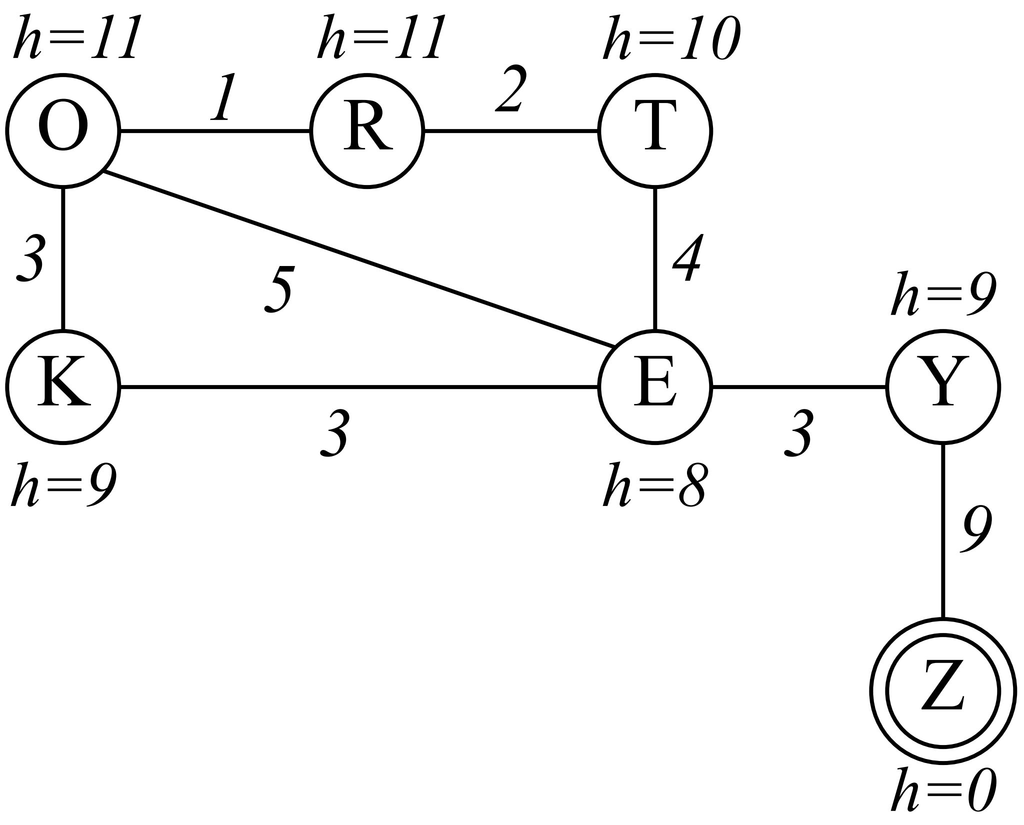




مثال IDA*

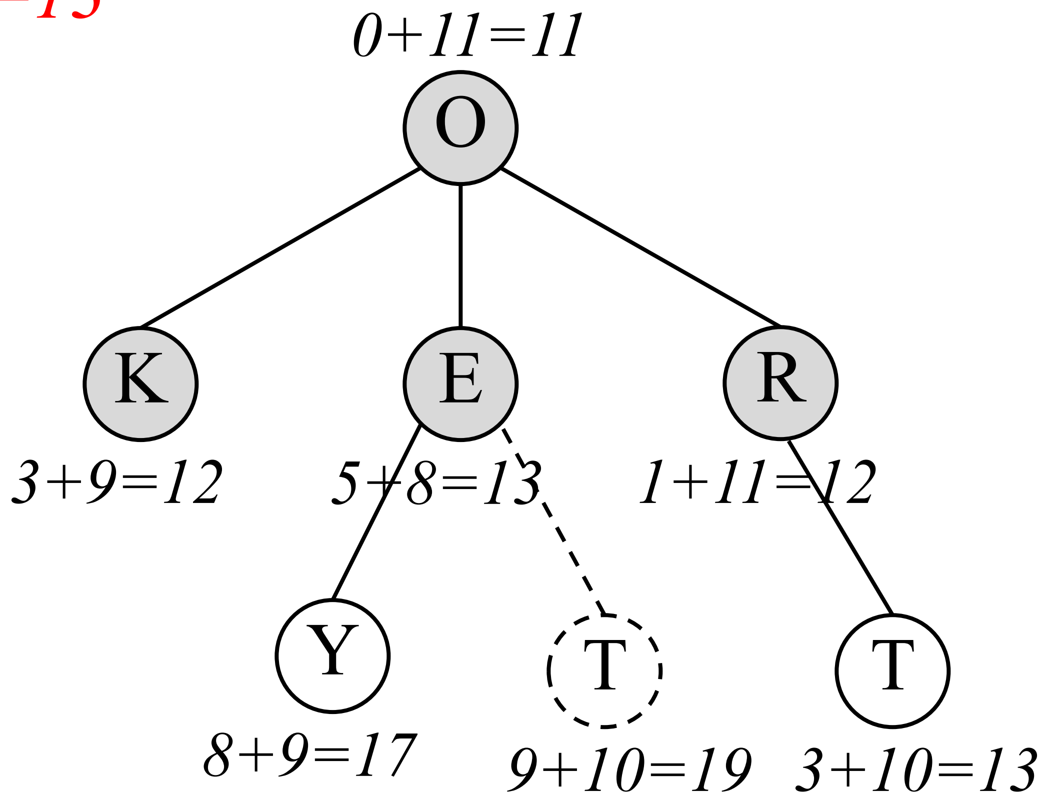
F-limit=13

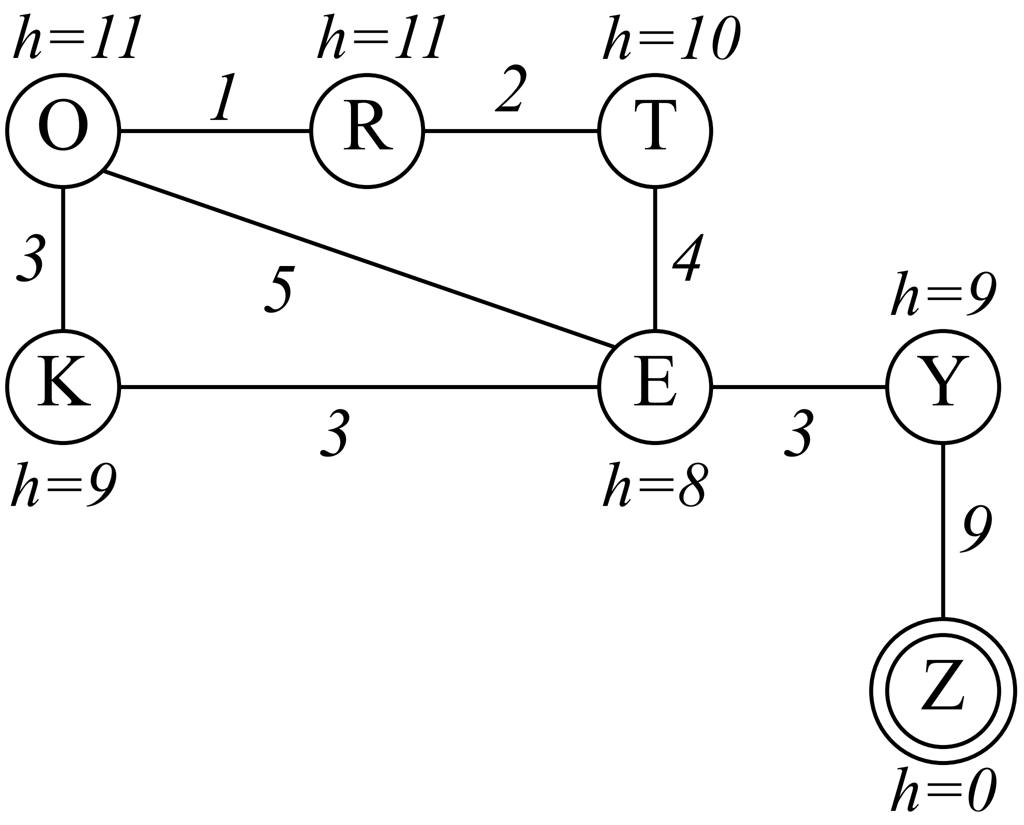




مثال IDA*

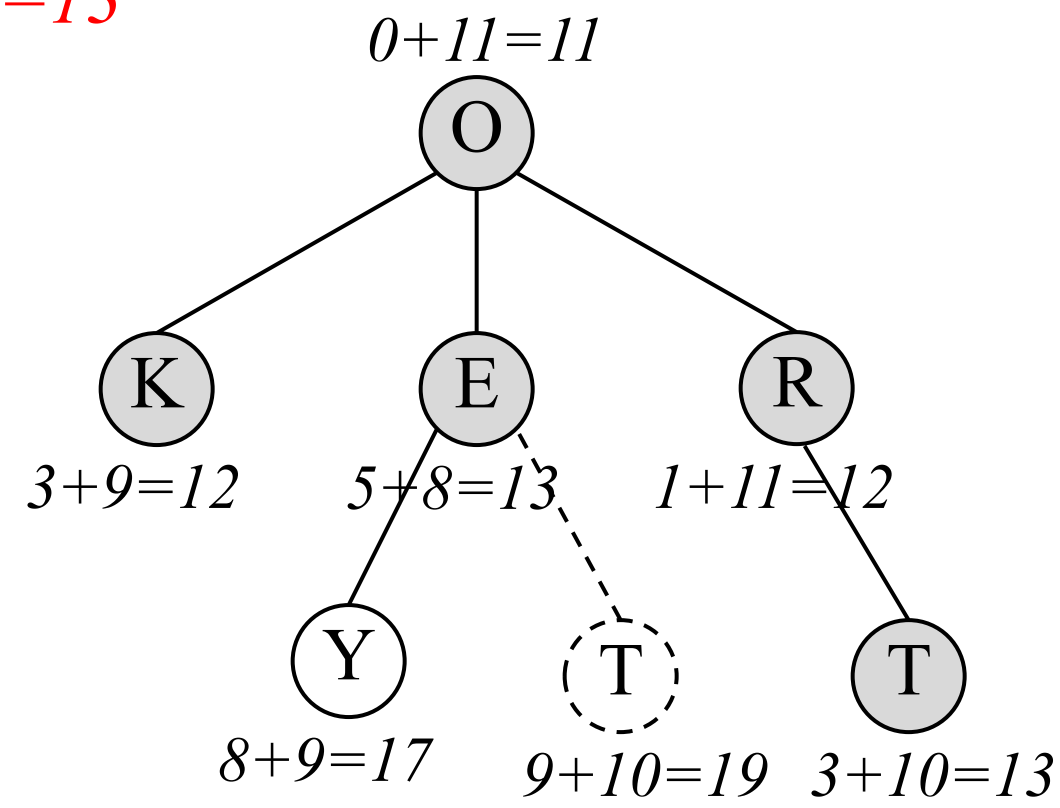
F-limit=13

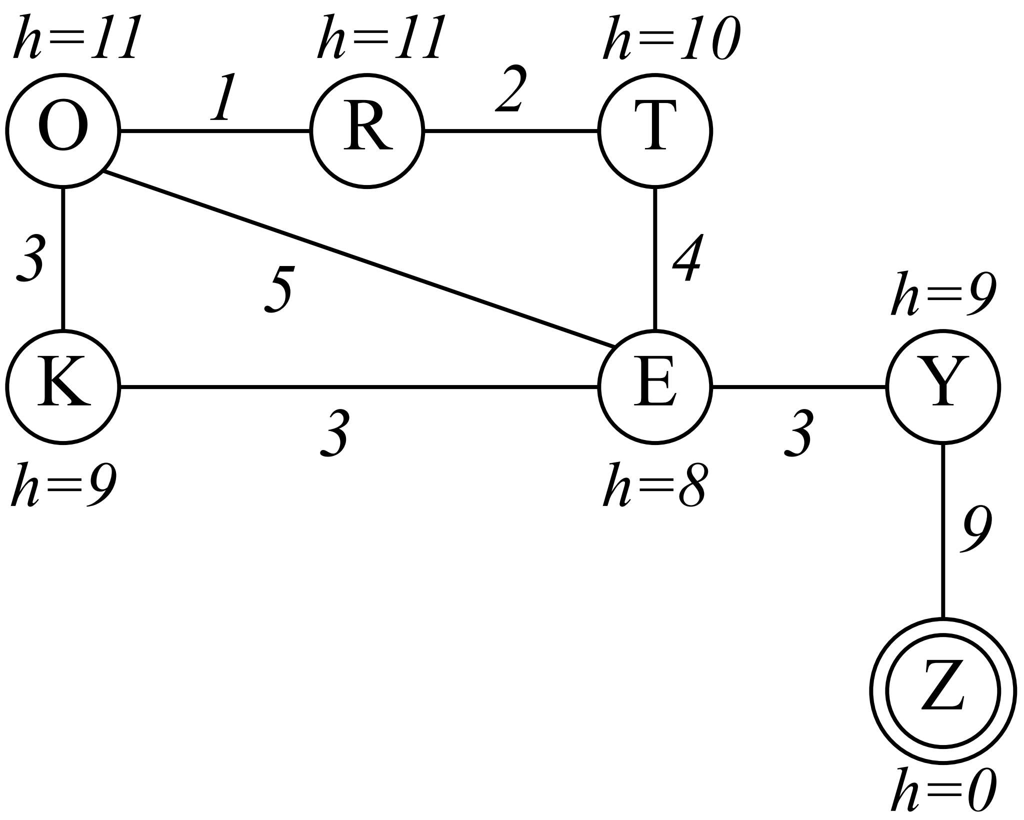




مثال IDA*

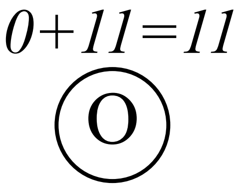
F-limit=13

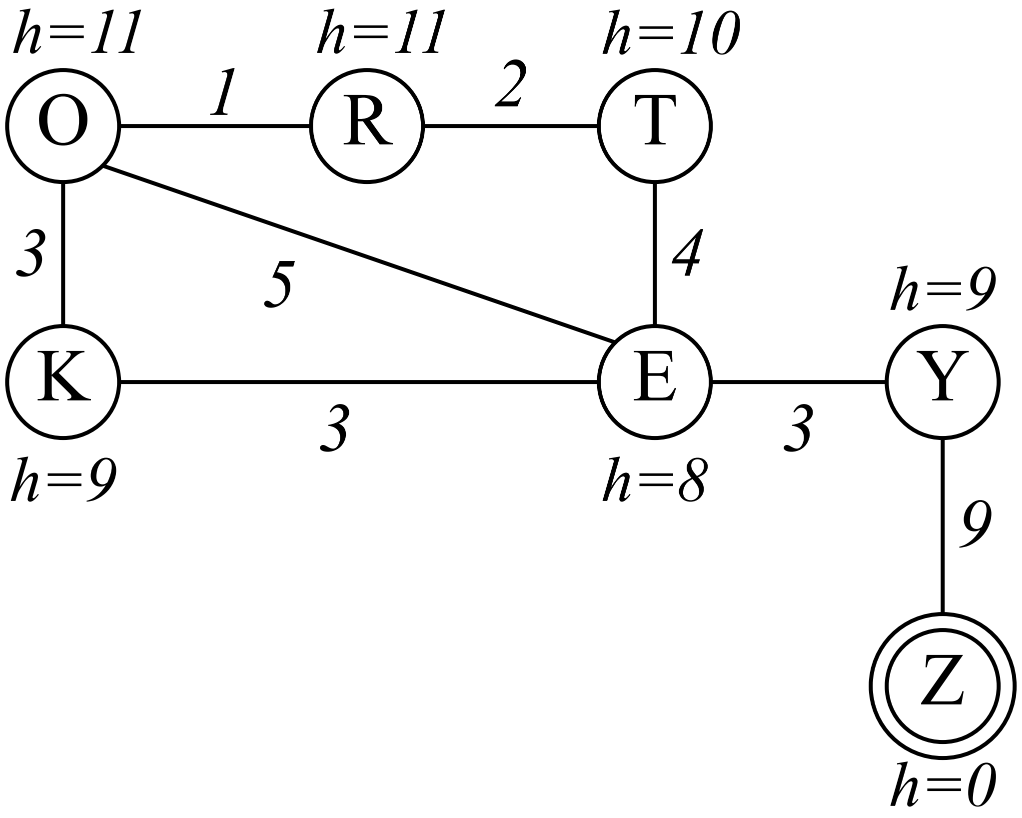




مثال IDA*

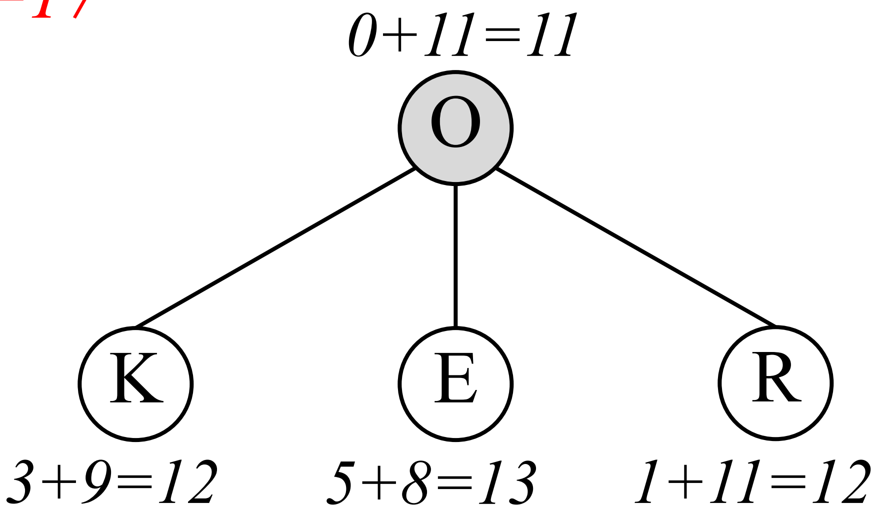
F-limit=17

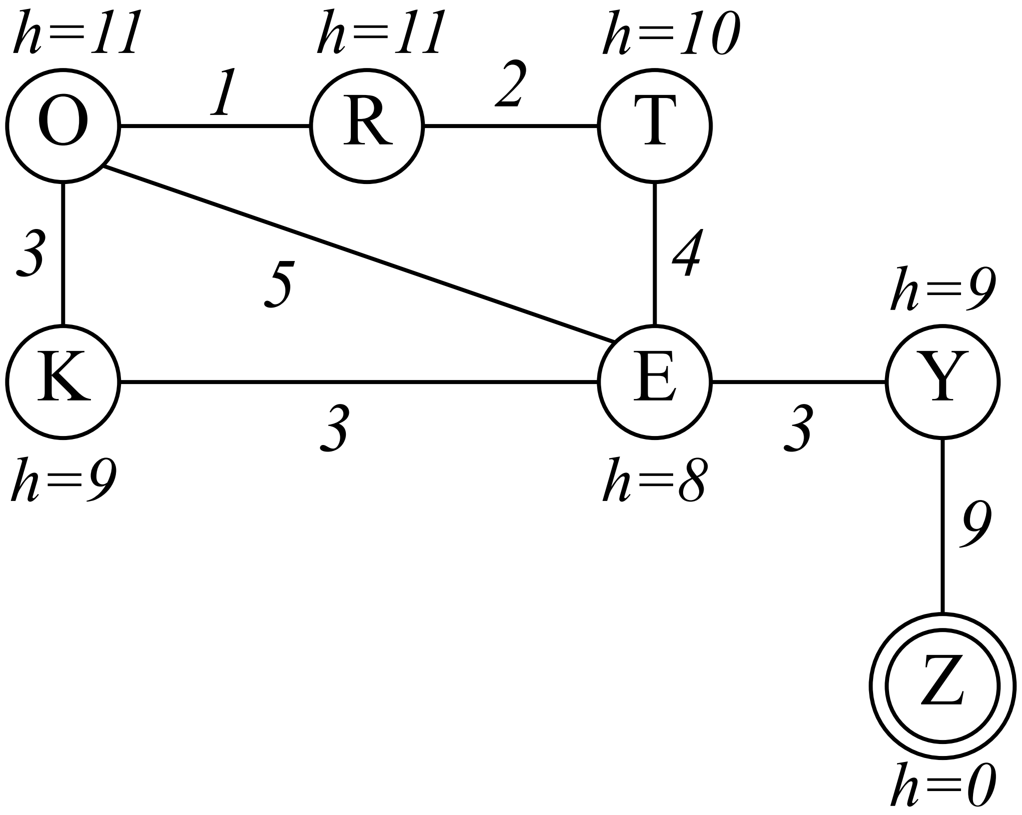




مثال IDA*

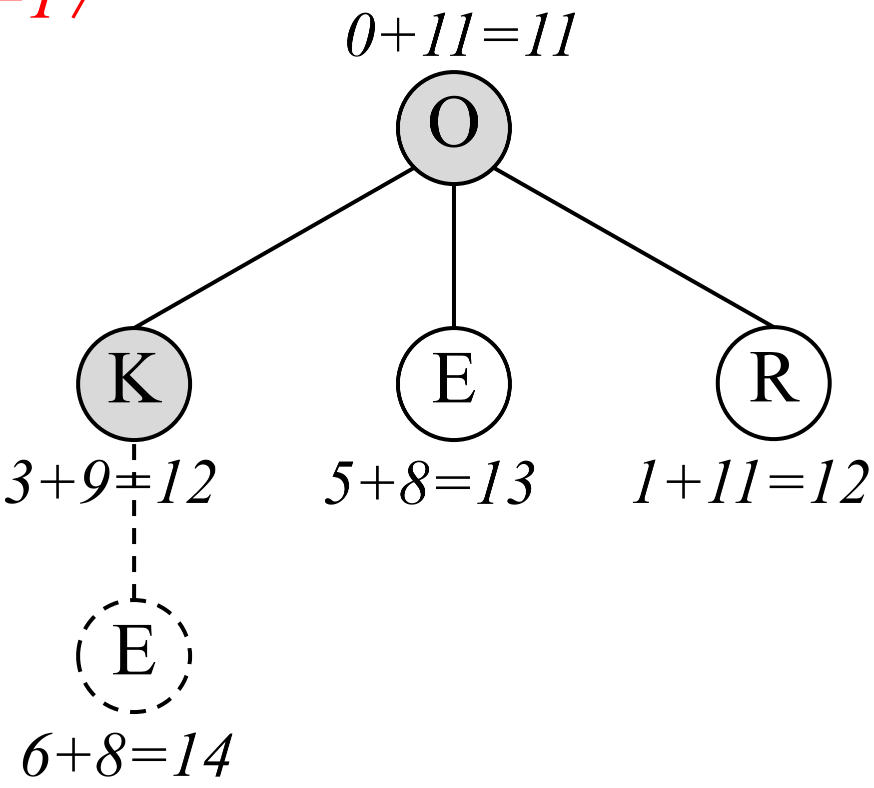
F-limit=17

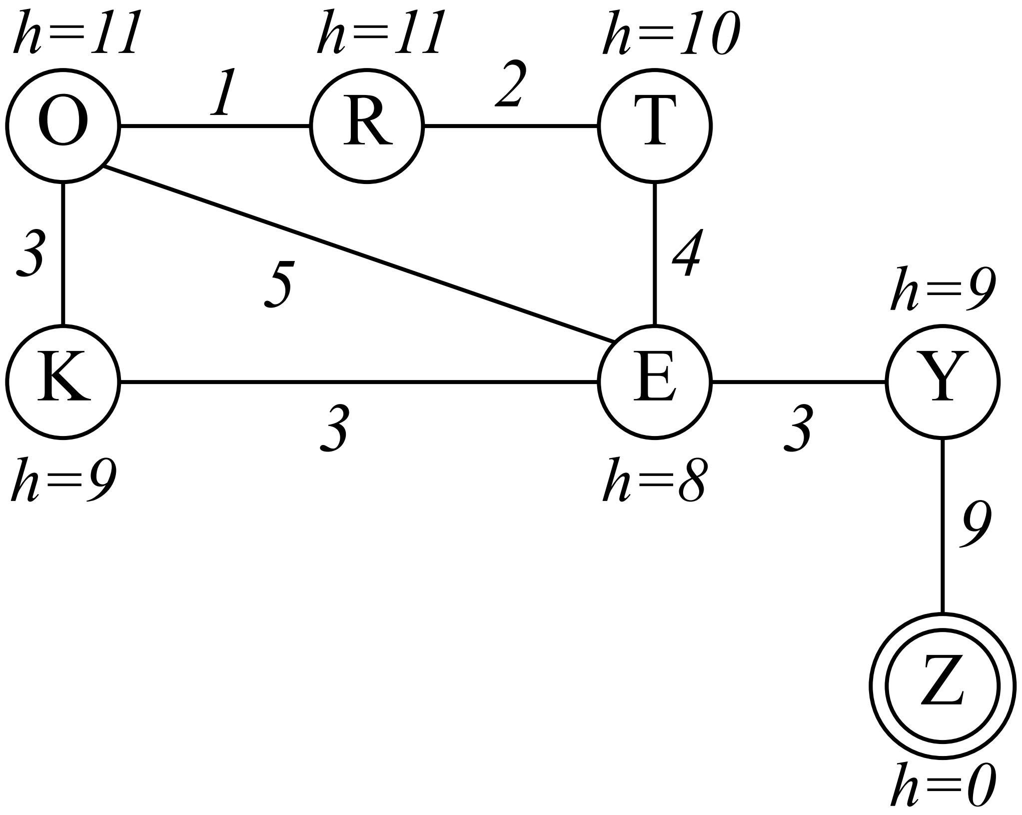




مثال IDA*

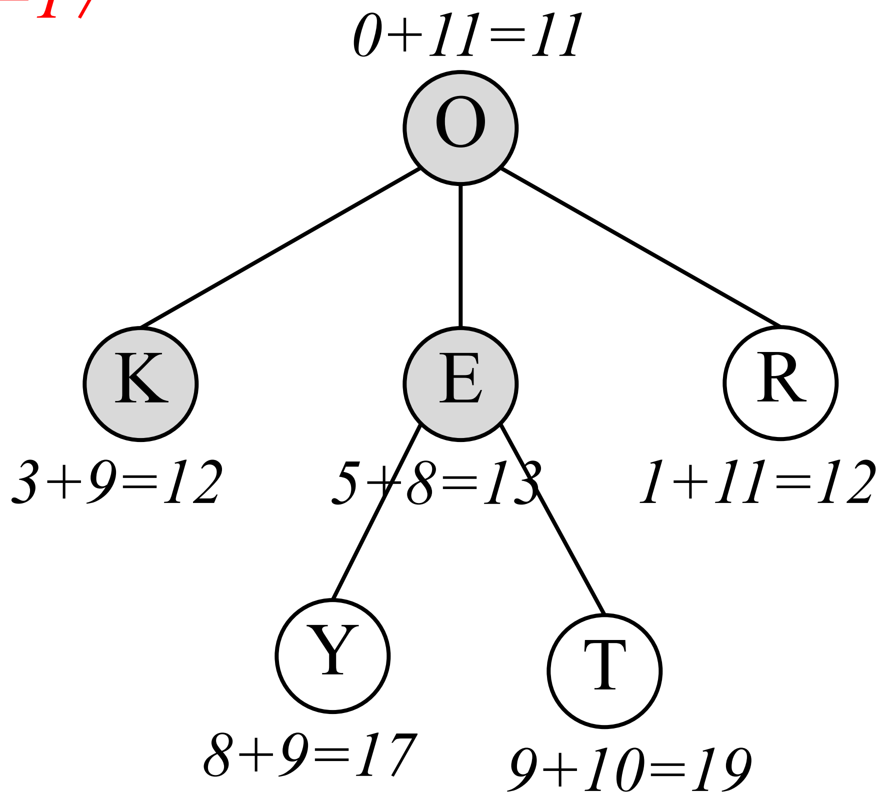
F-limit=17

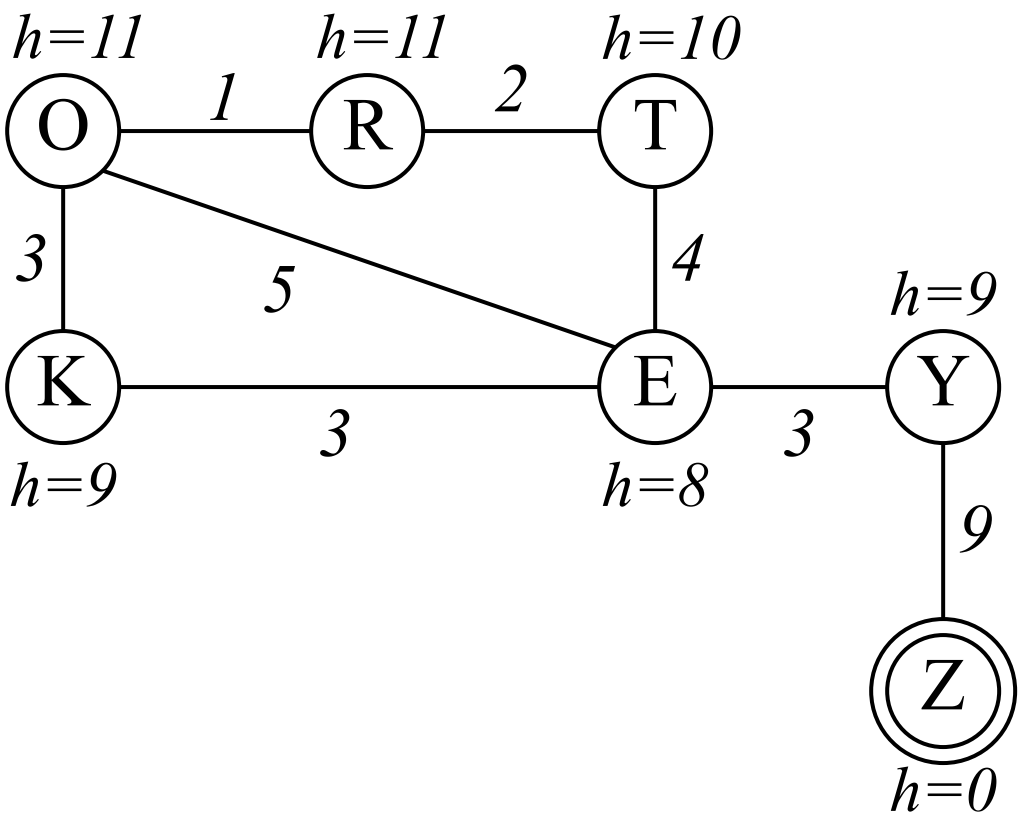




مثال IDA*

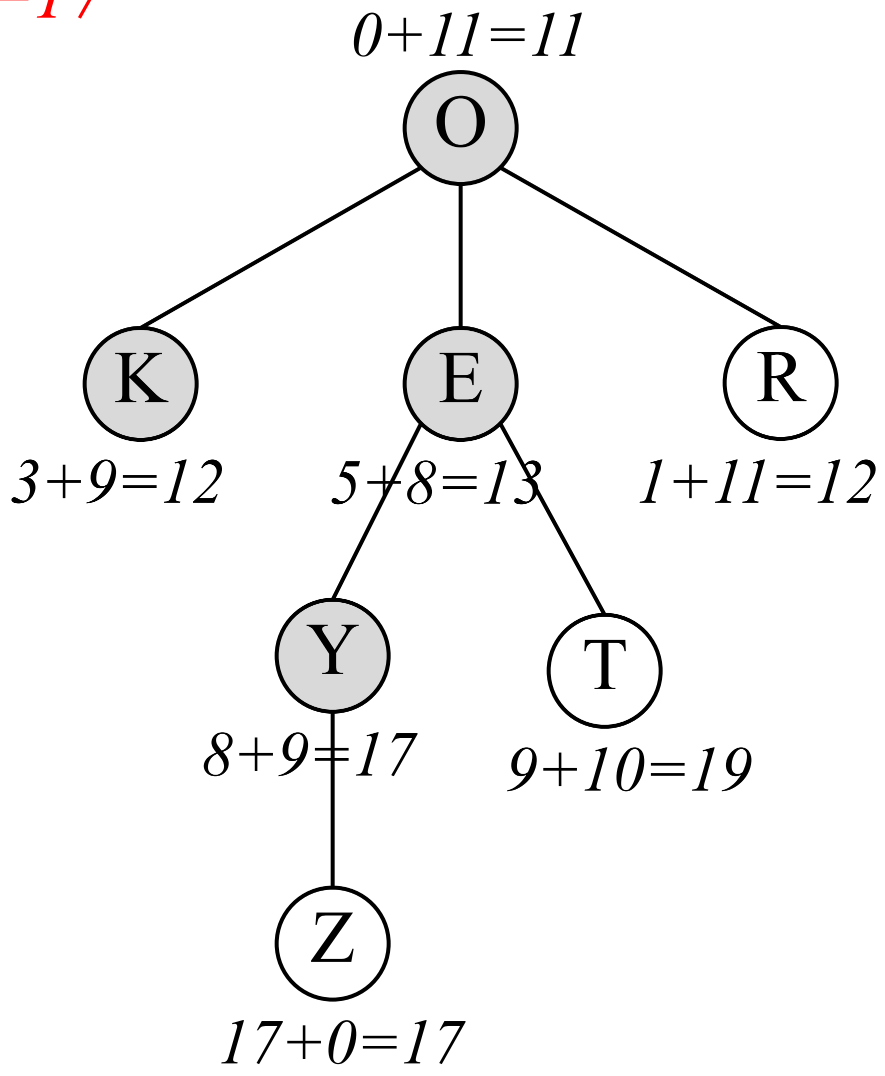
F-limit=17

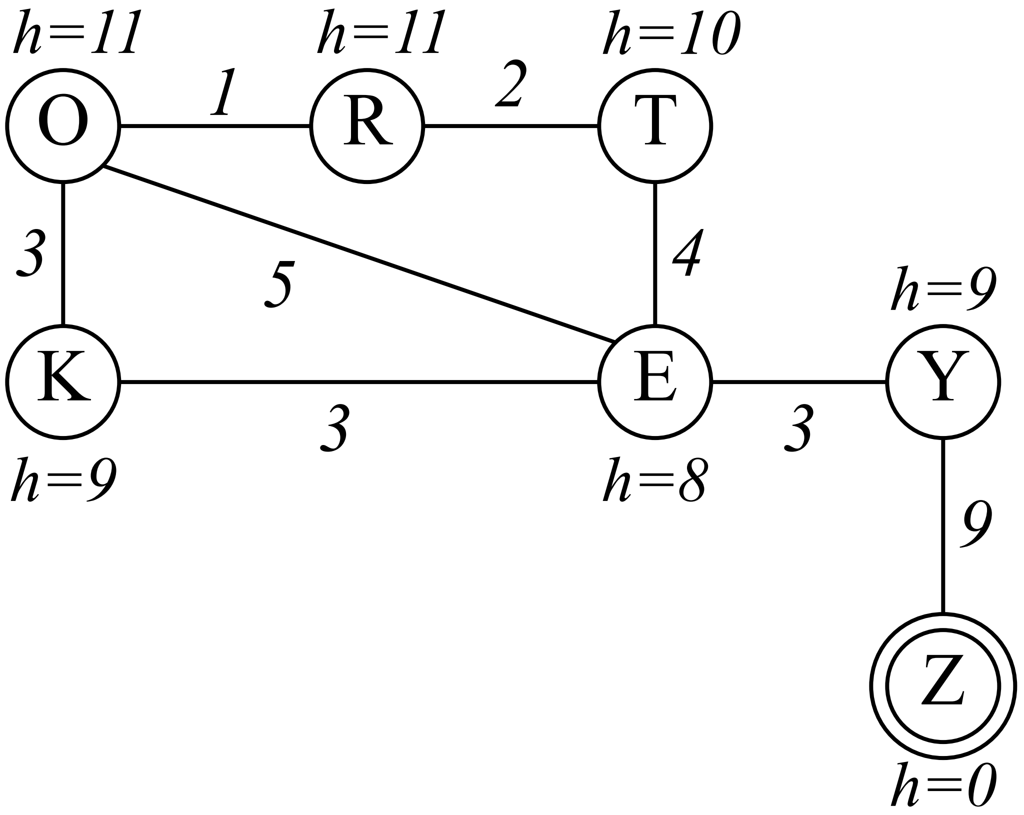




مثال IDA*

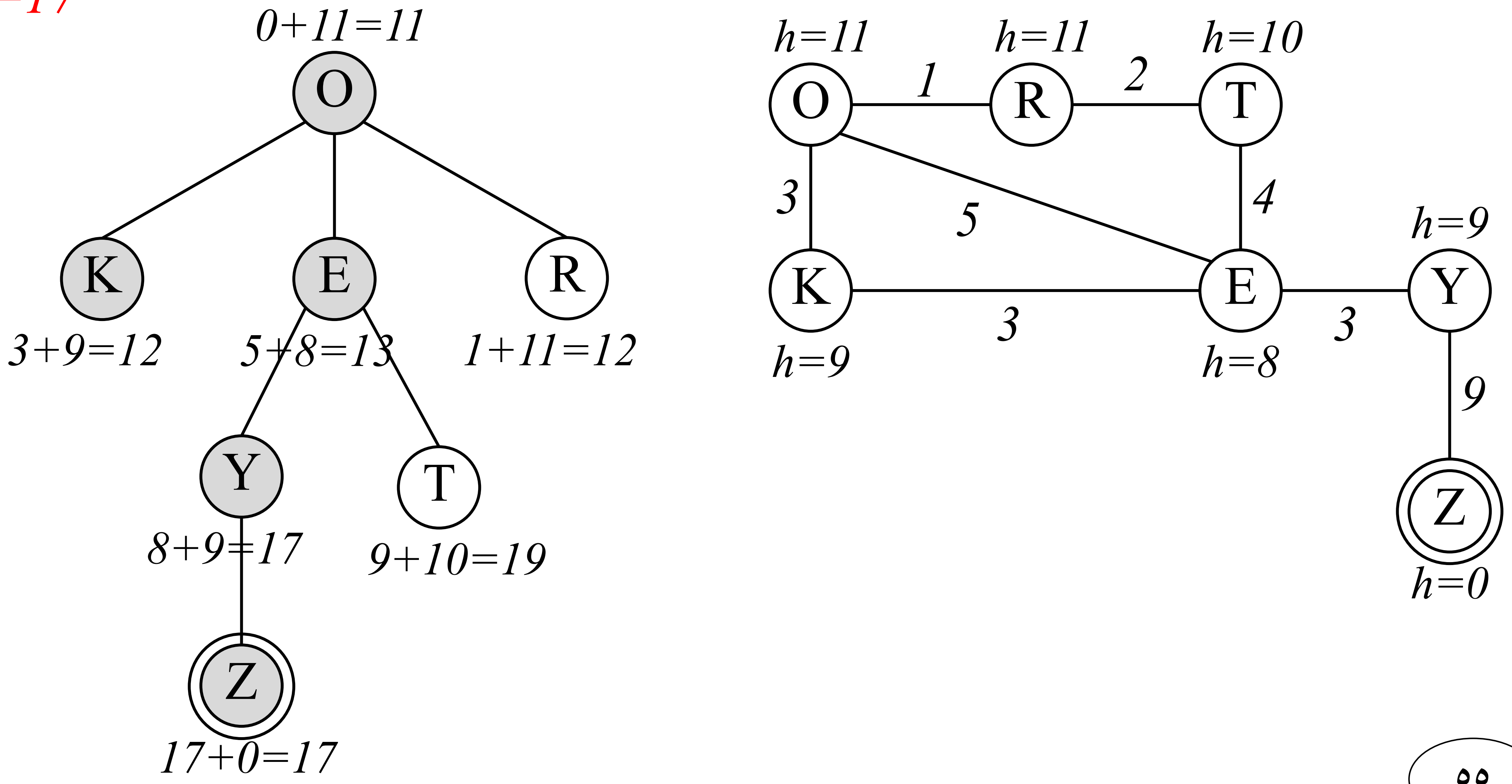
F-limit=17





مثال IDA*

F-limit=17



ارزیابی IDA*

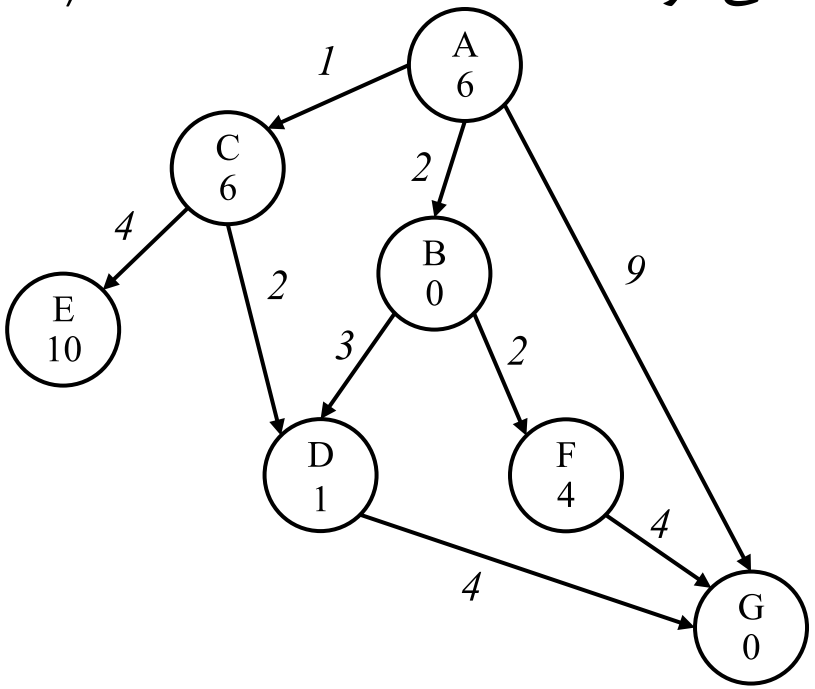
- کامل و بهینه؟
  - بله، اگر تابع هیوریستیک قابل قبول باشد، بهینه است.
  - می‌توان از طریق استقرا روی دفعات تکرار الگوریتم، بهینگی آن را اثبات نمود.
- پیچیدگی زمانی؟
  - در آخرین تکرار الگوریتم، بیشترین تعداد گره تولید می‌شود.
  - همانند A* در بدترین حالت نمایی است.
  - نیازی به یک صف اولویت برای نگه‌داری گره‌های مرزی ندارد. در نتیجه از سربار مرتب‌سازی چنین صفی بی‌نیاز است.
  - اگر مقدار تابع f برای هر حالت متفاوت باشد یعنی در هر تکرار فقط یک گره بیشتر از تکرار قبلی بسط می‌یابد. در این حالت اگر A* تعداد O(N) گره را بسط دهد IDA* باید N بار تکرار شود یعنی $1+2+3+\ldots+O(N)=O(N^2)$ که زمان بسیار زیادی است.

ارزیابی IDA*

- پیچیدگی فضایی؟
  - اگر از جستجوی گرافی استفاده کند عملاً مشکلی را حل نکرده است!! (چرا؟)
  - از جستجوی درختی استفاده می‌کند و در هر مرحله فقط گره‌هایی که مقدار f آن‌ها کم‌تر از *f-limit* است در حافظه نگه‌داری می‌شود. بنابراین پیچیدگی مانند جستجوی عمقی خطی است.
  - در بدترین حالت، از مرتبه $O(b \times (1+\lfloor C^*/\varepsilon \rfloor))$ است.
    - C* هزینه راه‌حل بهینه
    - ε کم‌ترین هزینه اعمال

تست ۲

در گراف زیر، گره A وضعیت شروع و گره G وضعیت هدف است. اعداد کنار هر لبه (link) هزینه عبور آن لبه است. مقدار تابع اکتشافی h هر گره، درون آن نوشته شده است. اگر مقدار آستانه را برابر با عدد ۷ درنظر بگیریم، کدام یک از گزینه‌های زیر، از چپ به راست، ترتیب ملاقات (visit) گره‌های این گراف توسط روش *IDA را نشان می‌دهد. فرض کنید فرزندان هر گره به‌ترتیب حروف الفبا تولید می‌شود و در شرایط مساوی به گره‌ای که زودتر تولید شده، اولویت داده می‌شود. (کامپیوتر ۹۵)



۱) A, B, D, G

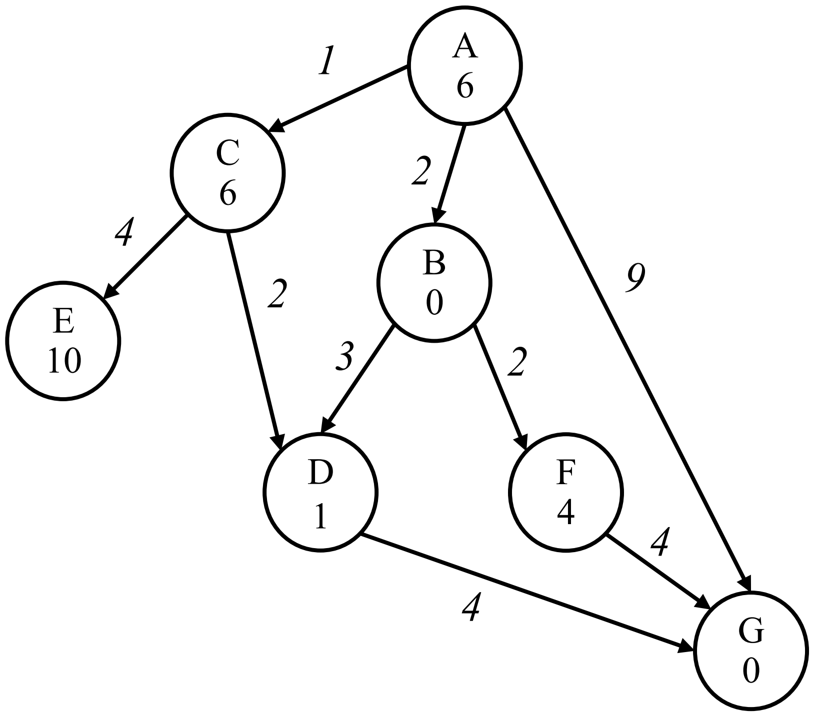
۲) A, C, D, G

۳) A, B, D, C, D, G ✓

۴) A, C, D, B, F, G

تست ۳

کدام یک از گزینه‌های زیر در مورد تابع اکتشافی h سوال قبل از نظر دو ویژگی قابل قبول بودن (Admissibility) و سازگاری (Consistency) صحیح است؟ (کامپیوتر ۹۵)



۱) فقط سازگار است.

۲) فقط قابل قبول است. ✓

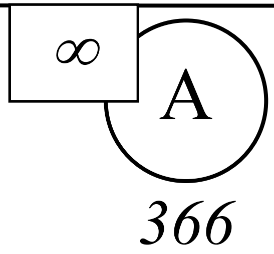
۳) هم قابل قبول است هم سازگار.

۴) نه قابل قبول است نه سازگار.

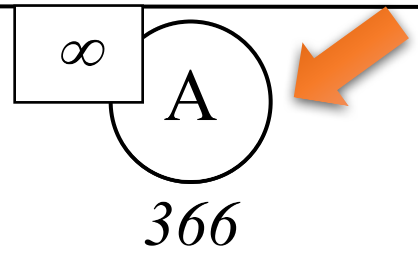
جستجوی اول بهترین بازگشتی – RBFS

ساختاری شبیه به جستجوی عمقی بازگشتی دارد اما به‌جای این‌که دائماً مسیر فعلی را به سمت پایین ادامه دهد، مقدار f بهترین مسیر جانشین از طریق اجداد گره فعلی را نگه می‌دارد. اگر f گره فعلی از این حد تجاوز کند، الگوریتم به عقب برمی‌گردد تا مسیر جانشین را انتخاب نماید. در برگشت به عقب این الگوریتم مقدار f مربوط به بهترین برگ در زیردرخت فراموش‌شده را به یاد می‌آورد و می‌تواند تصمیم بگیرد آیا این زیردرخت باید بعداً دوباره ایجاد شود یا خیر.

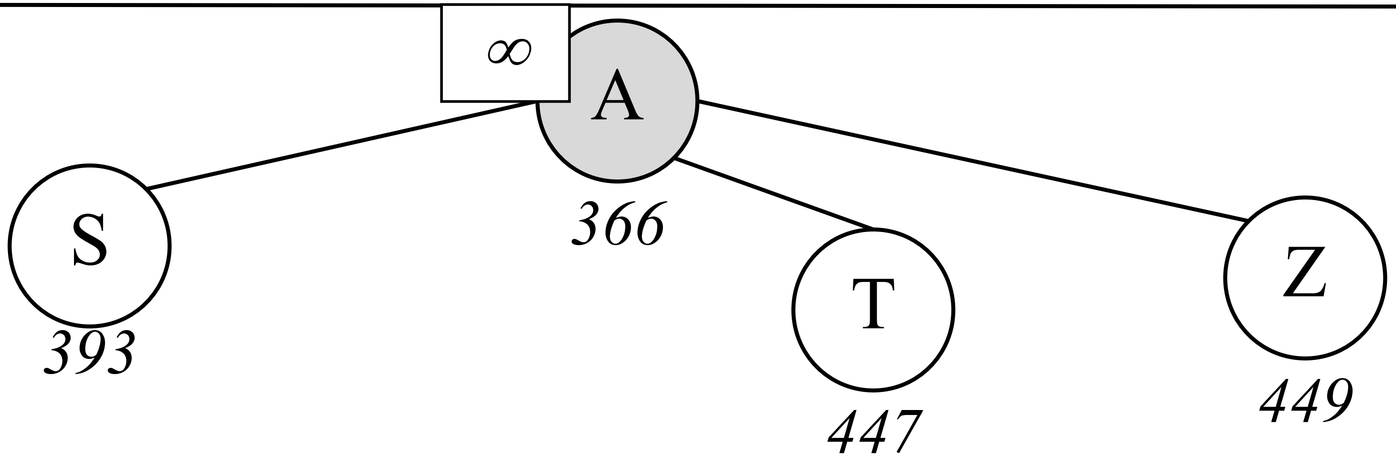
RBFS – مثال



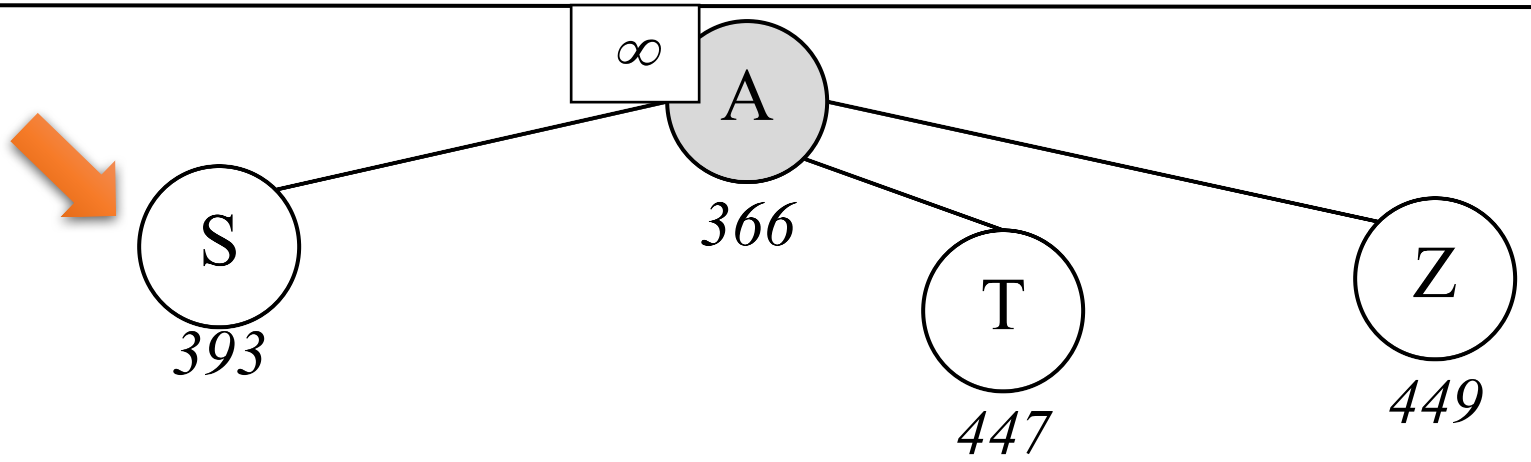
RBFS – مثال



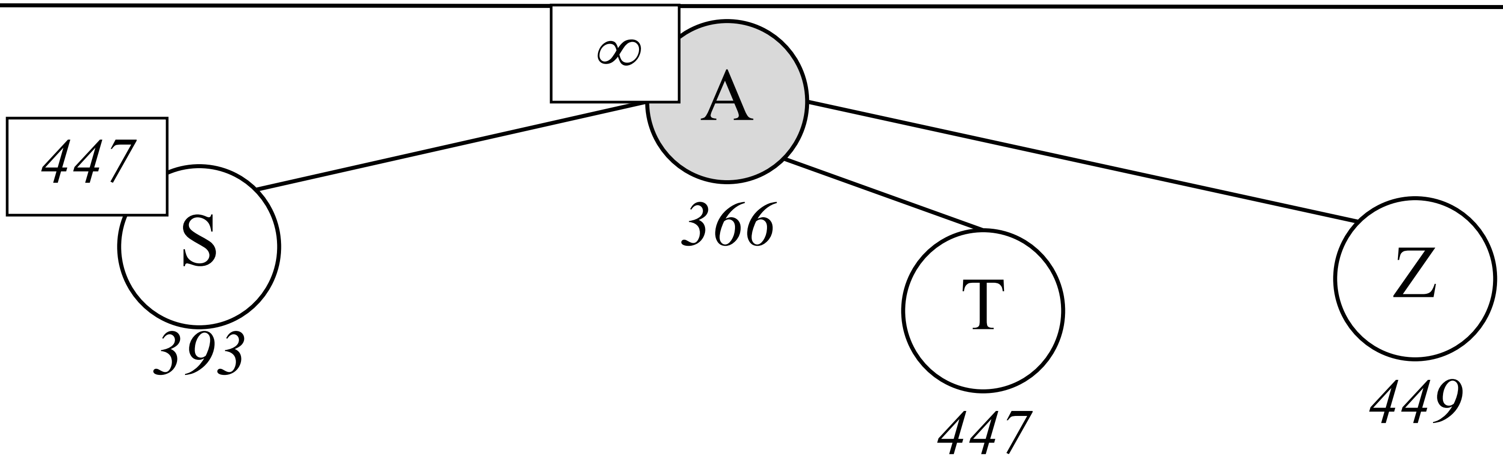
مثال – RBFS



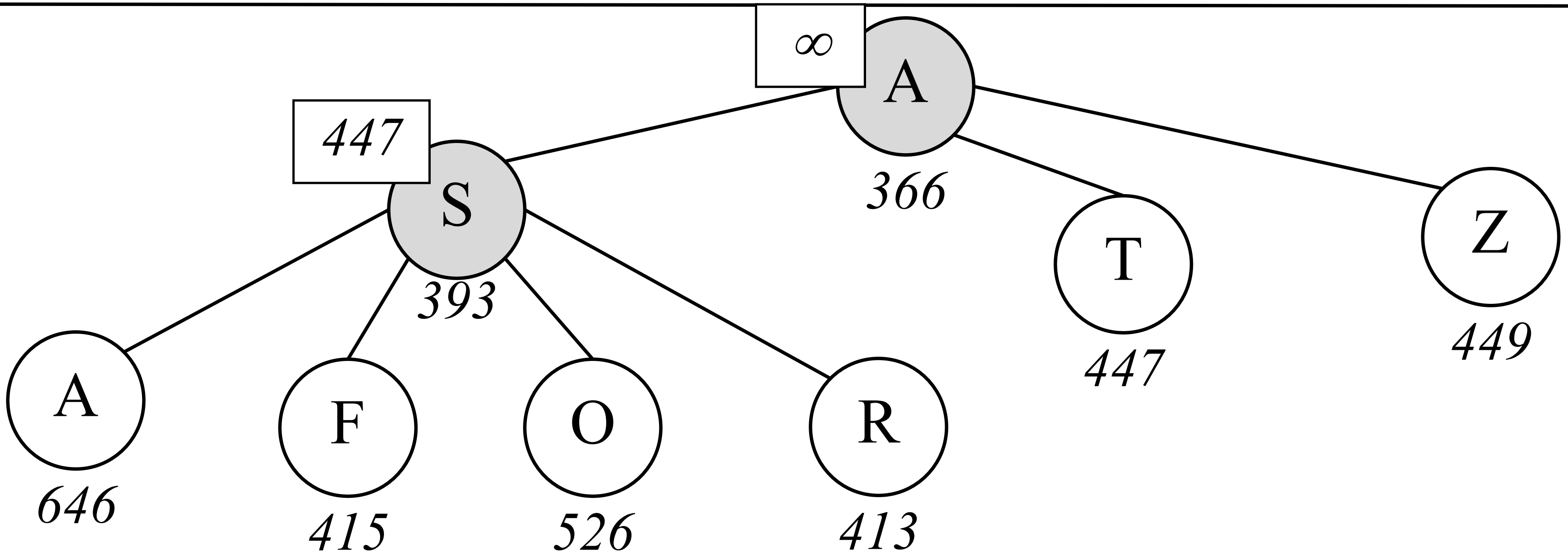
RBFS – مثال



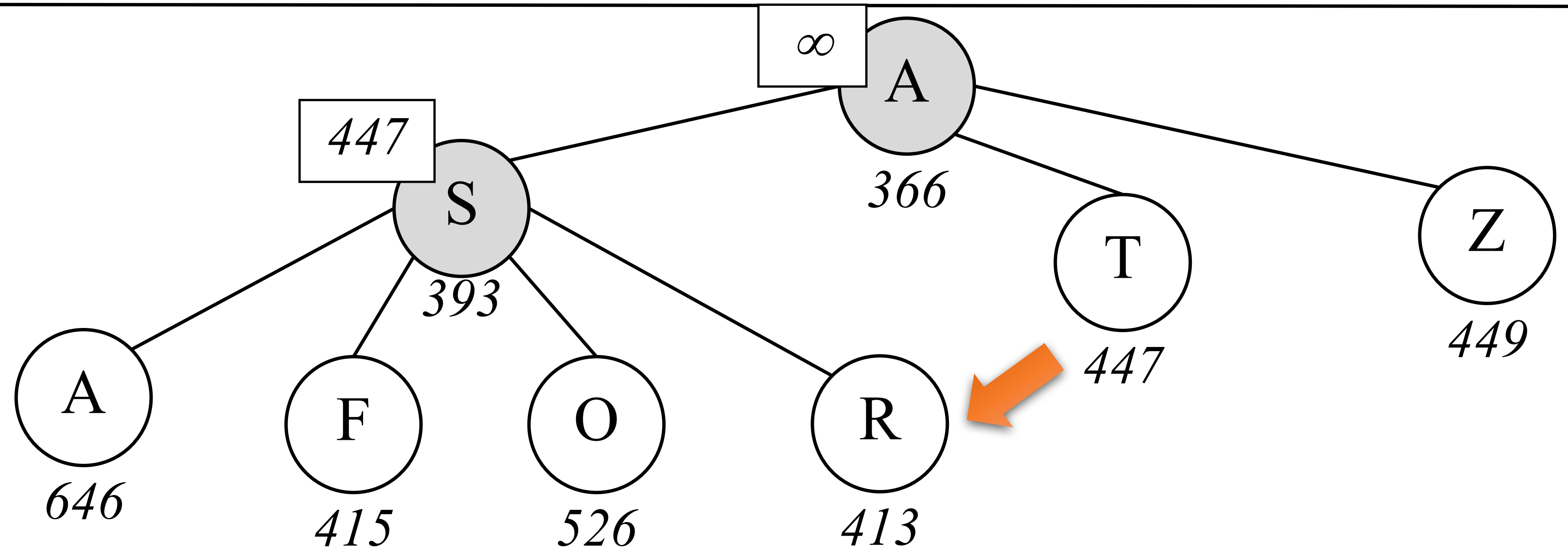
RBFS – مثال



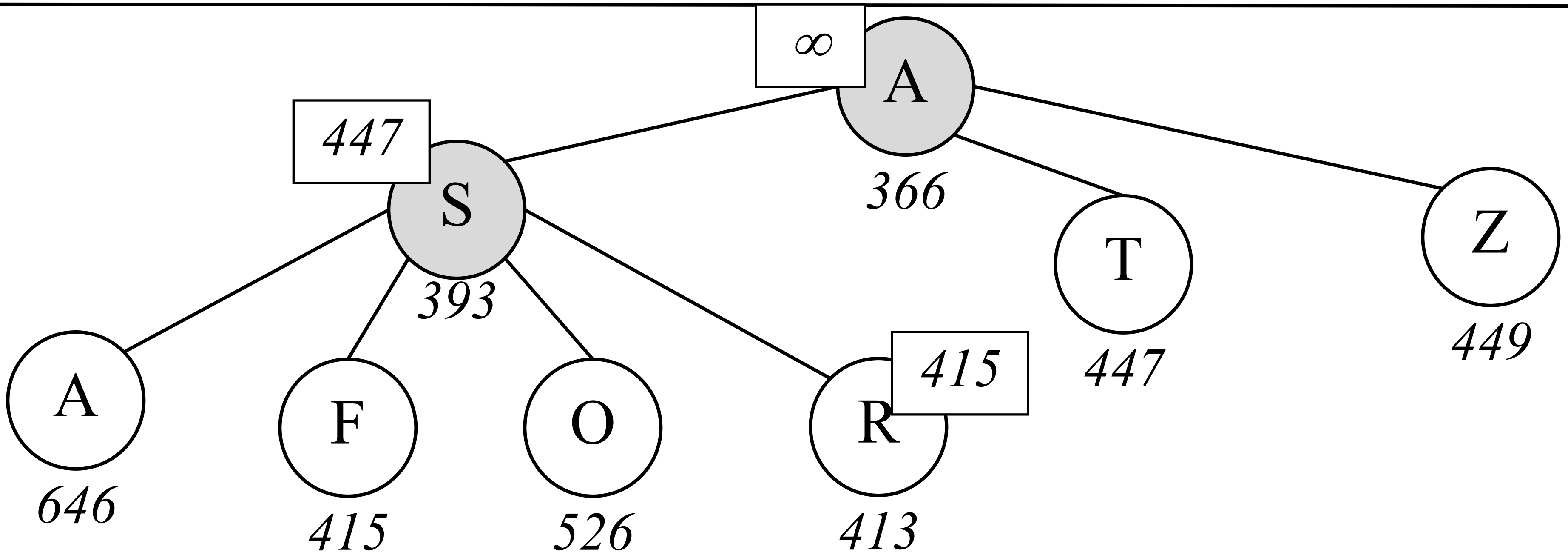
مثال – RBFS



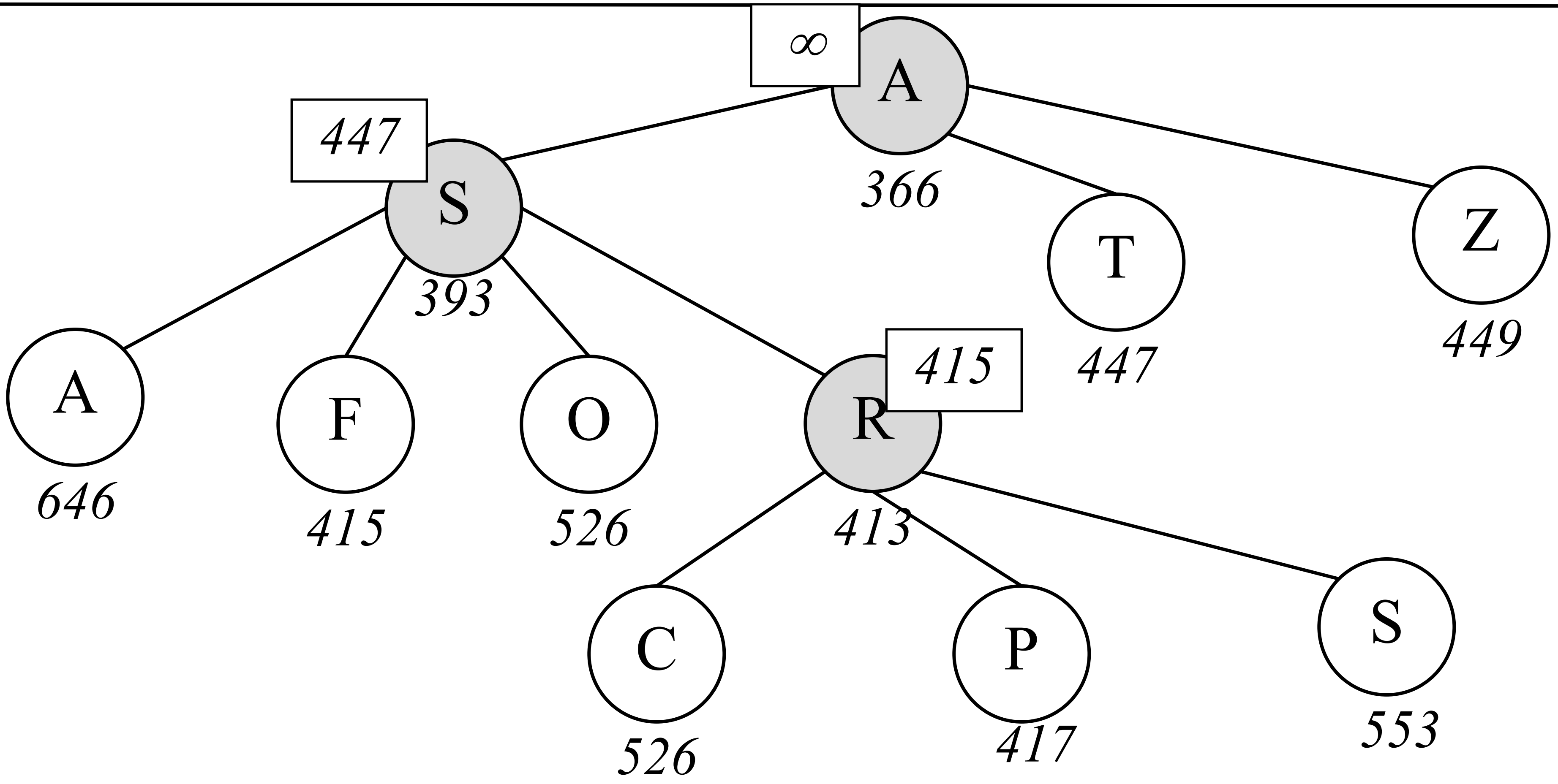
RBFS – مثال



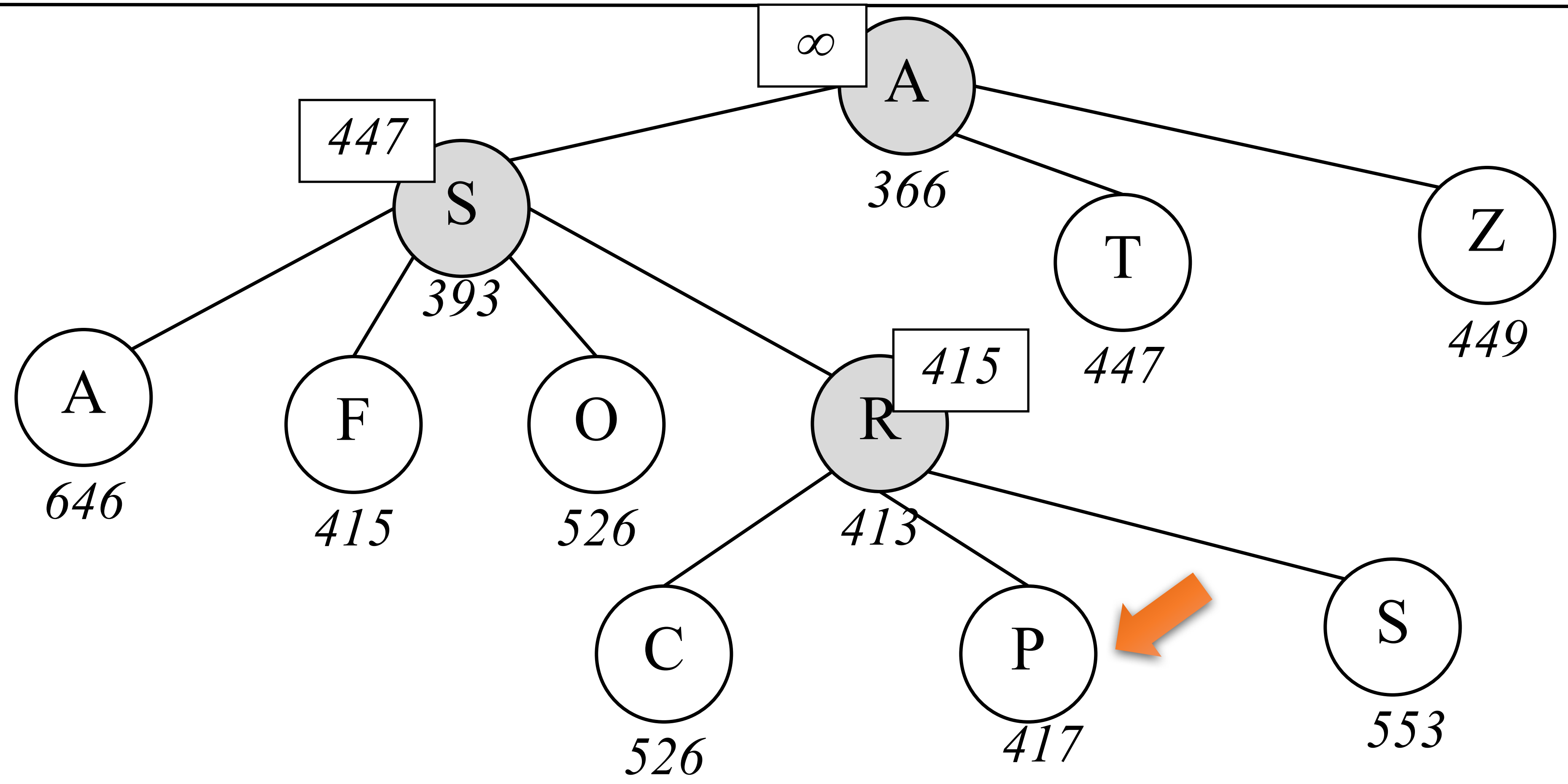
RBFS – مثال



مثال – RBFS

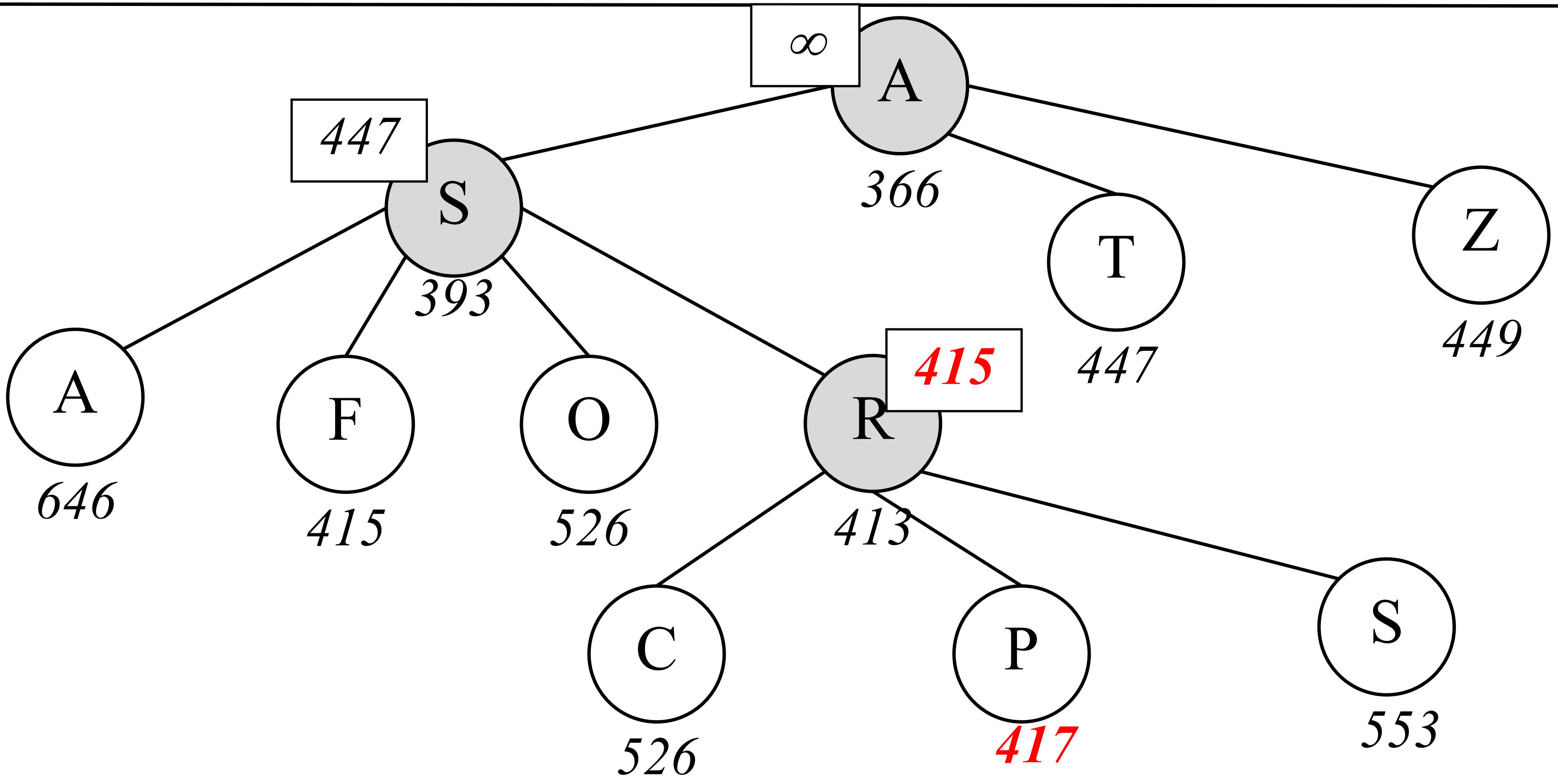

∞
A
366
447
S
393
T
447
Z
449
A
646
F
415
O
526
415
R
413
C
526
P
417
S
553

مثال – RBFS



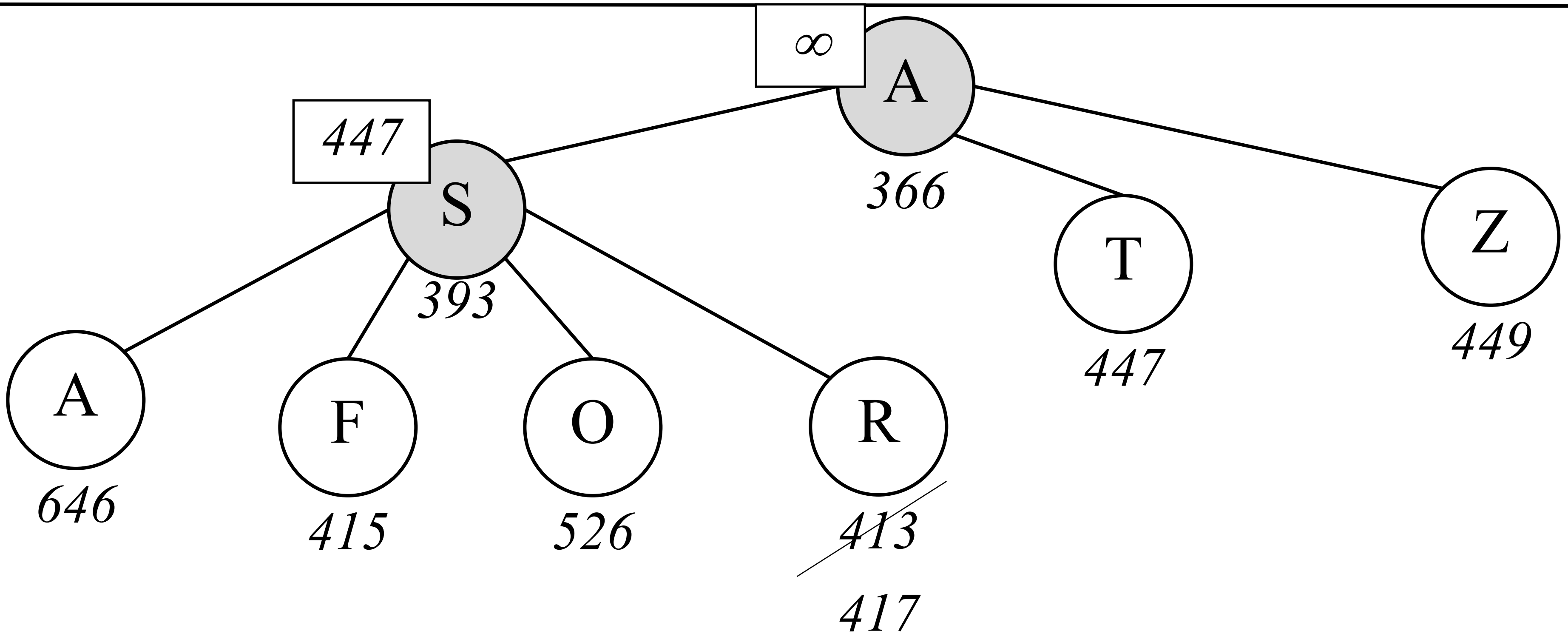
∞
A
366
447
S
393
T
447
Z
449
A
646
F
415
O
526
415
R
413
C
526
P
417
S
553

RBFS – مثال

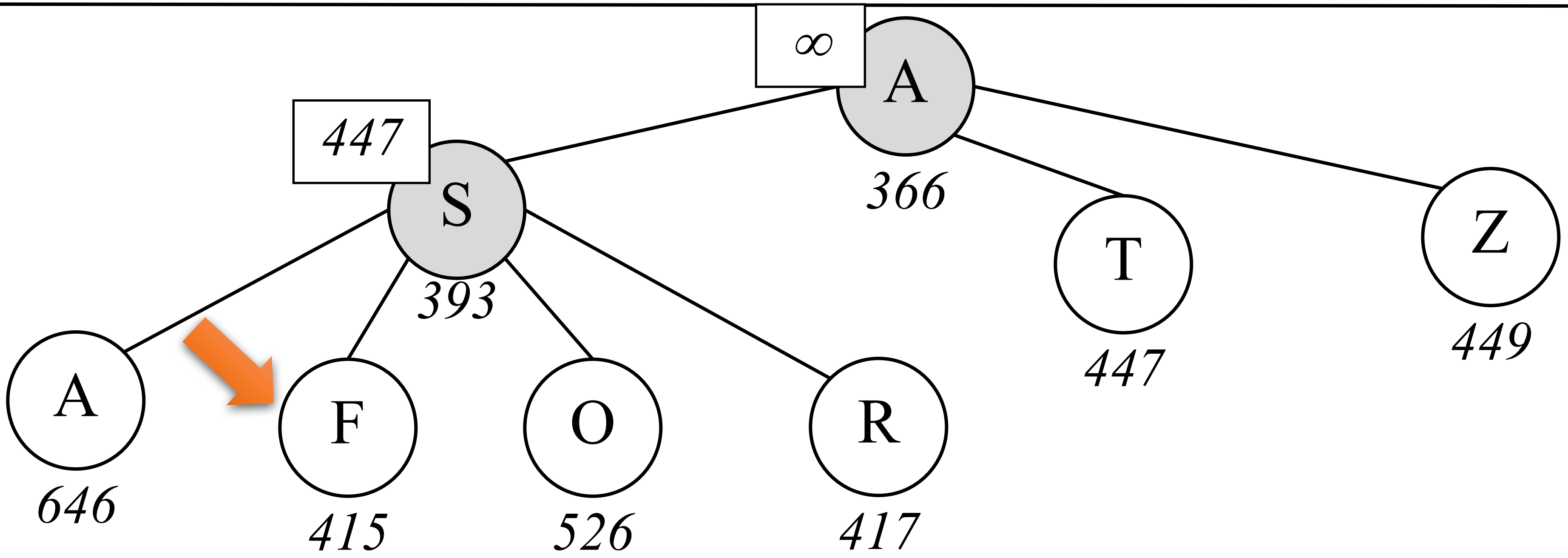


∞
A
366
447
S
393
T
447
Z
449
A
646
F
415
O
526
R
413
415
C
526
P
417
S
553

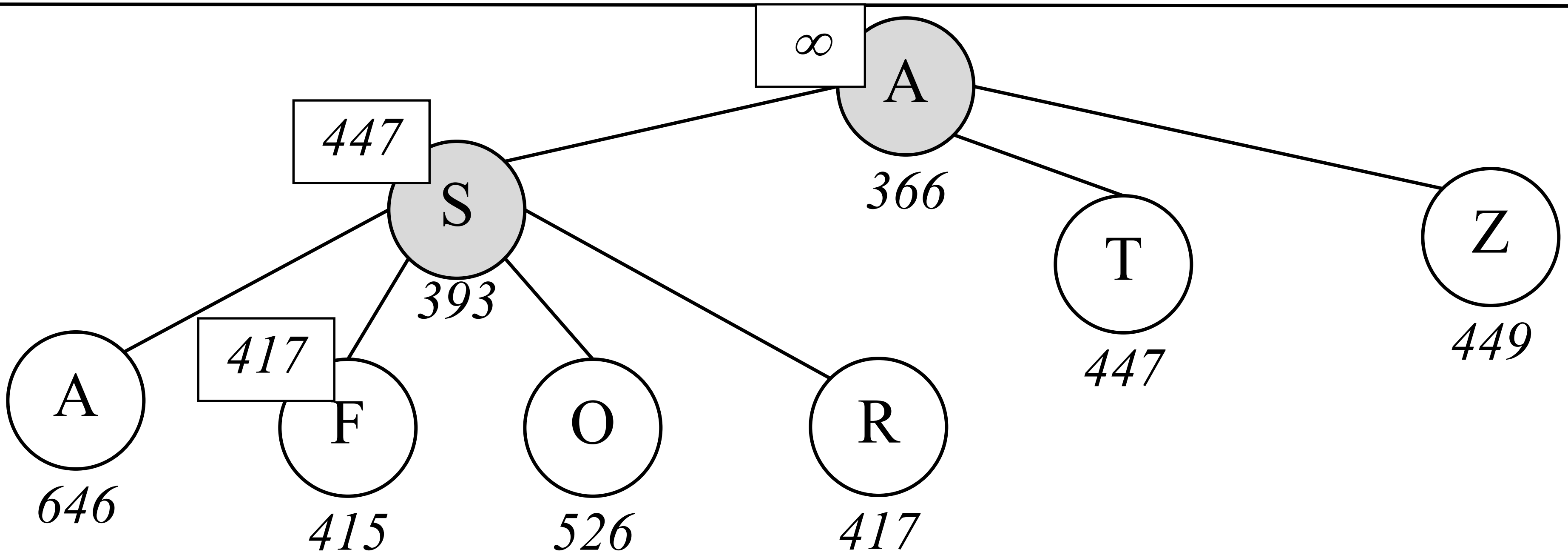
RBFS – مثال



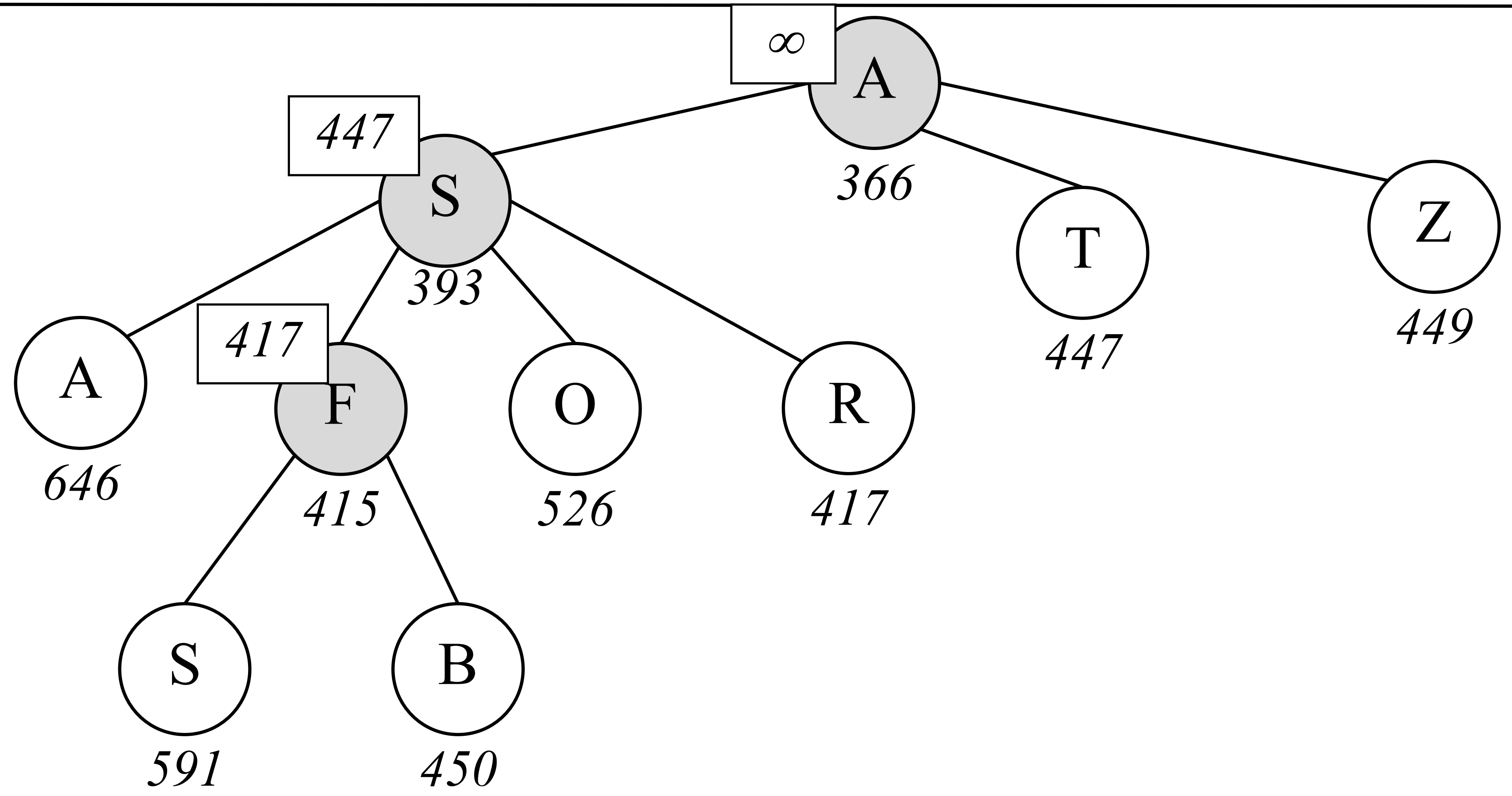
RBFS – مثال



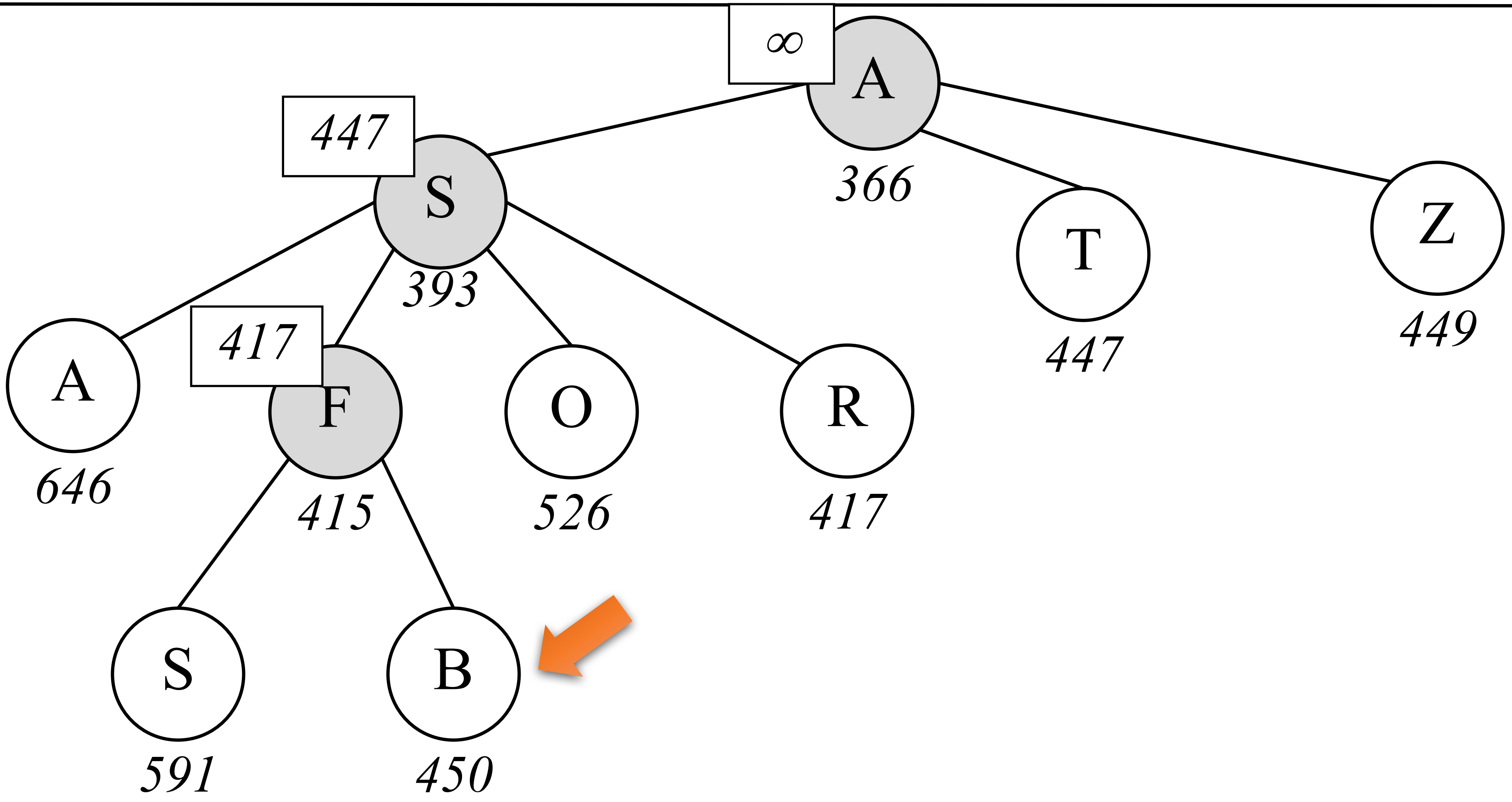
RBFS – مثال



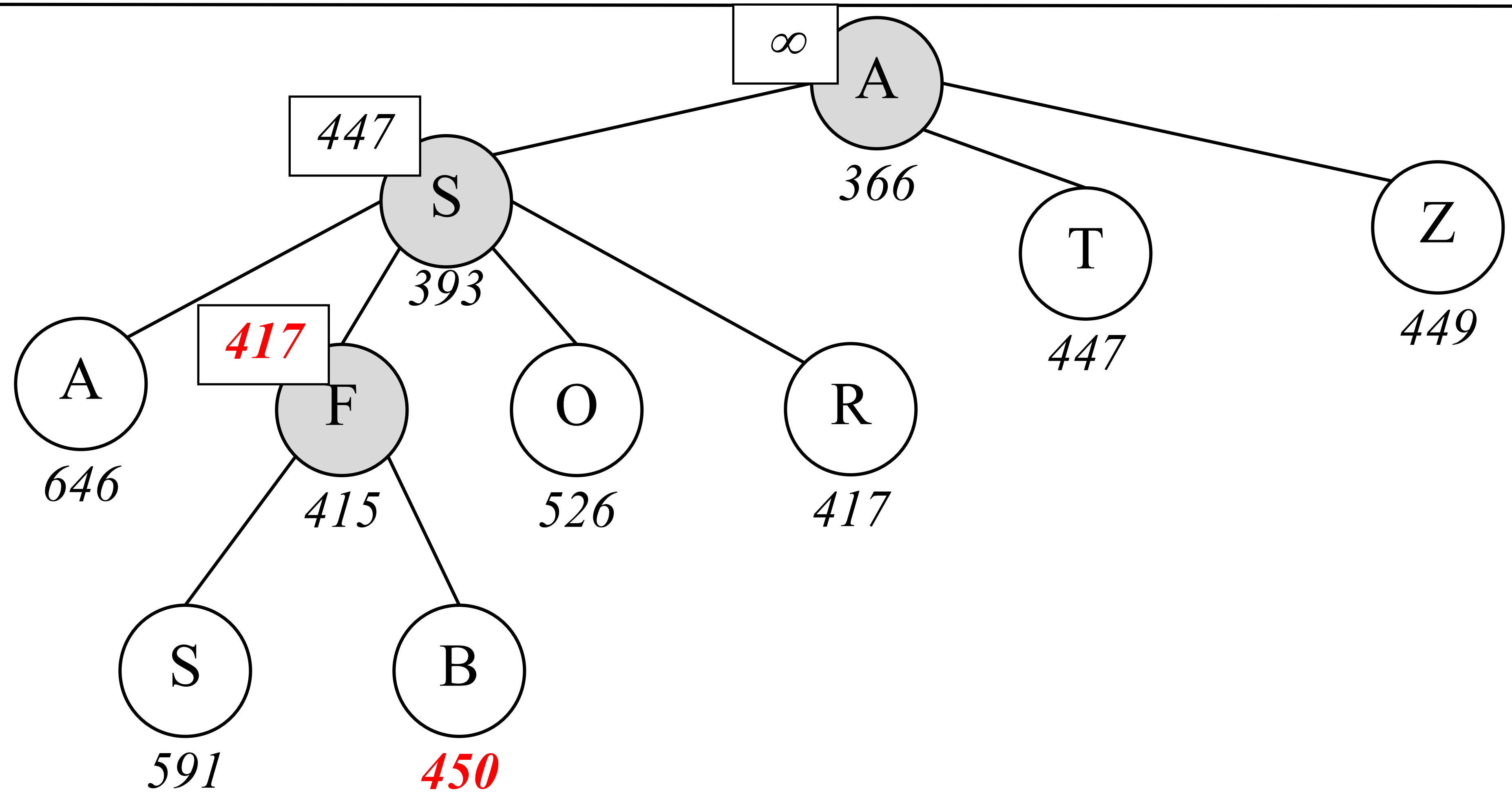
RBFS – مثال



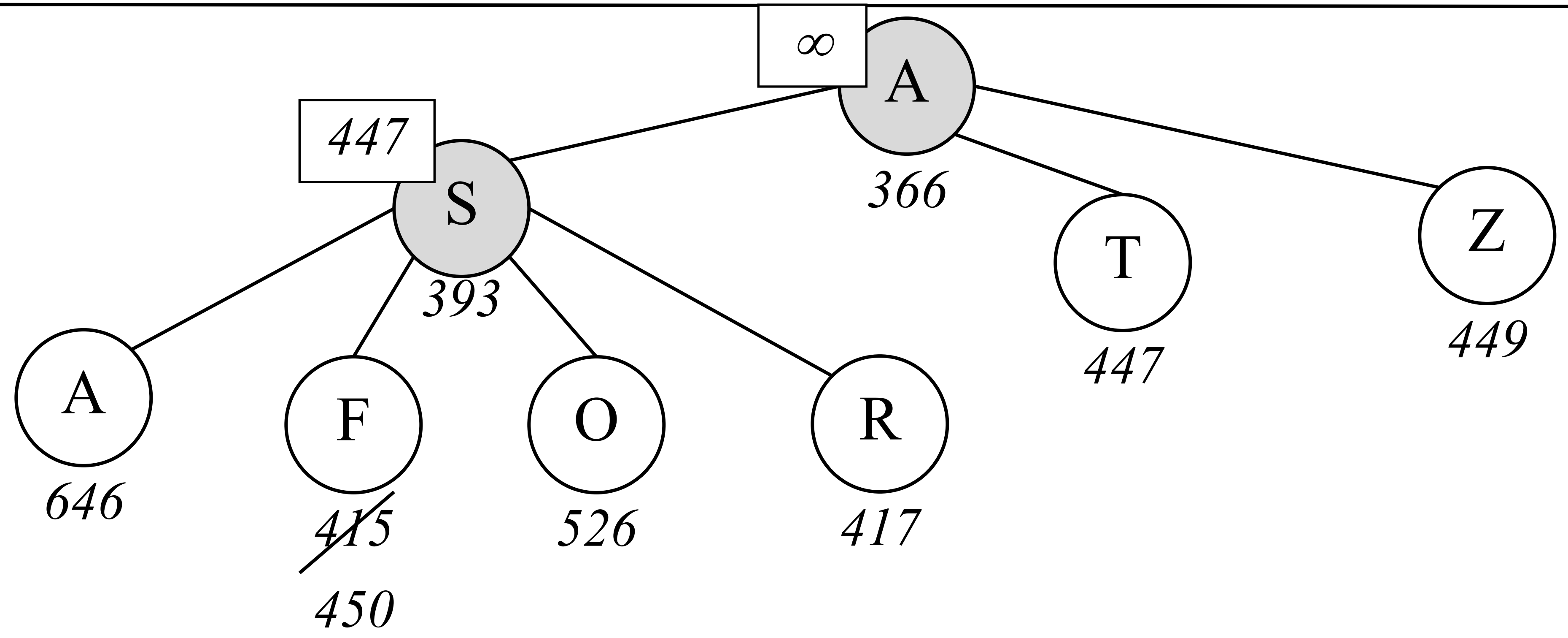
RBFS – مثال



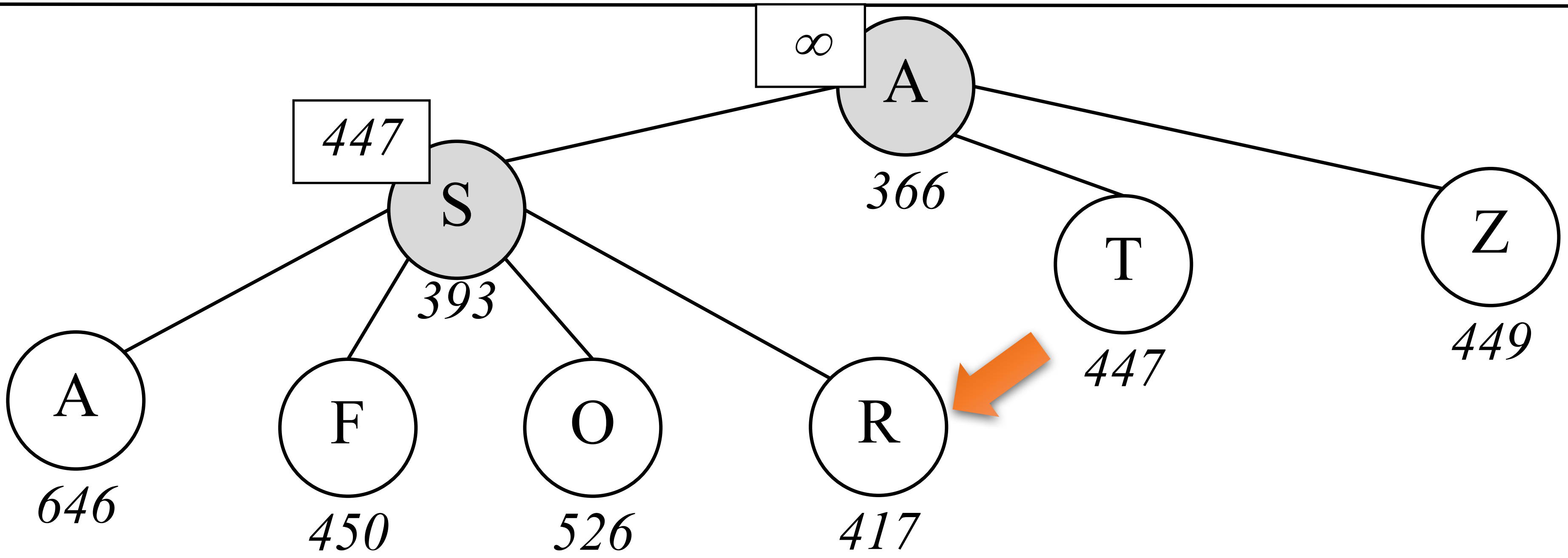
RBFS – مثال



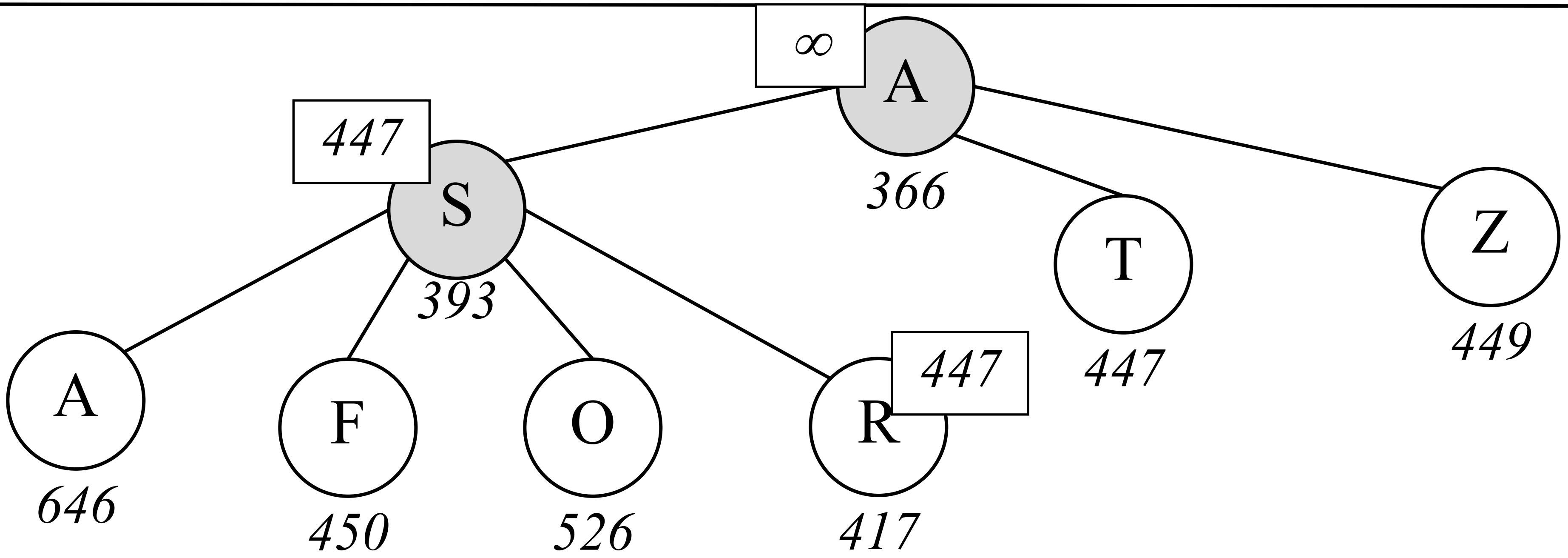
RBFS – مثال



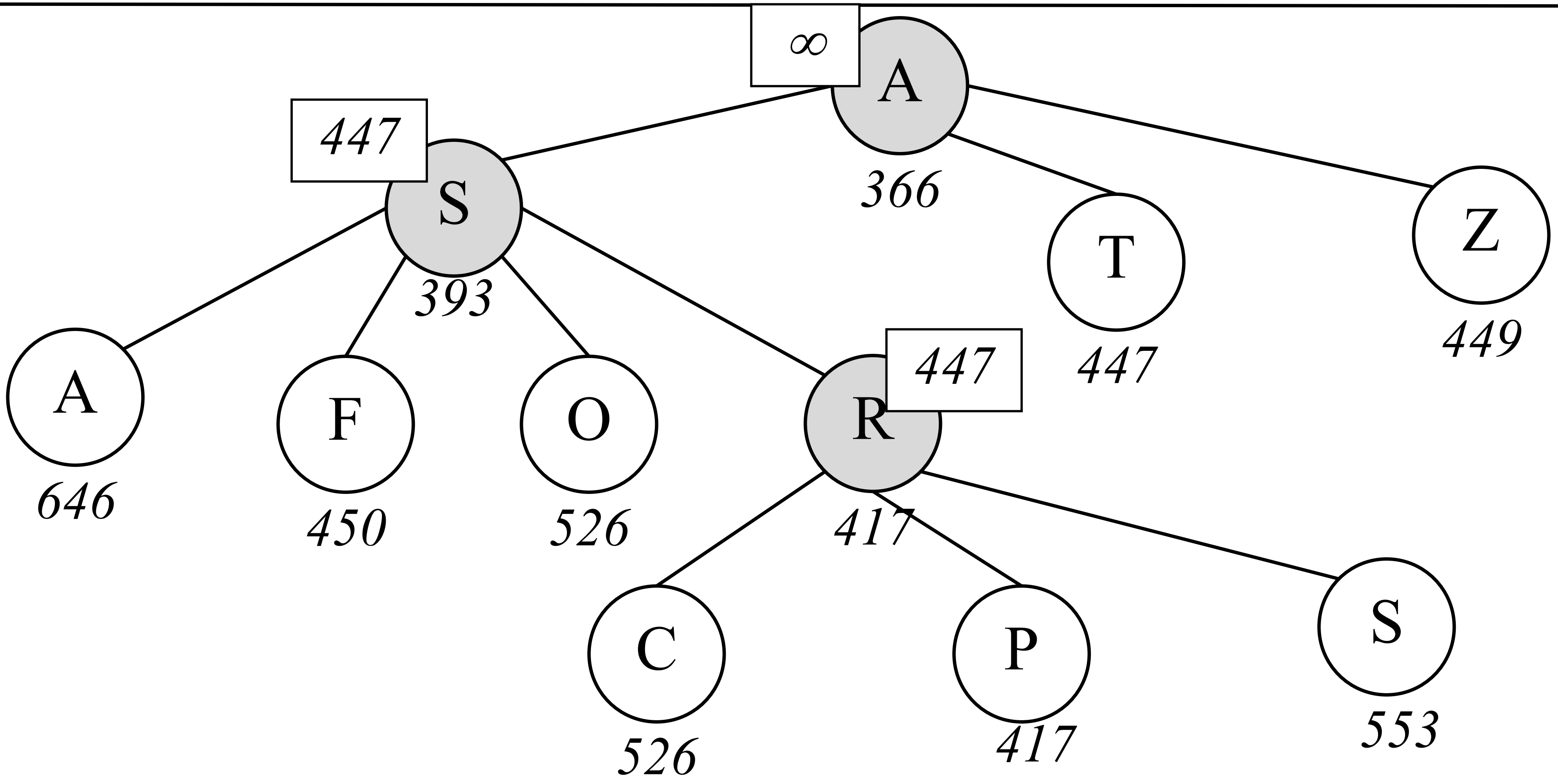
RBFS – مثال



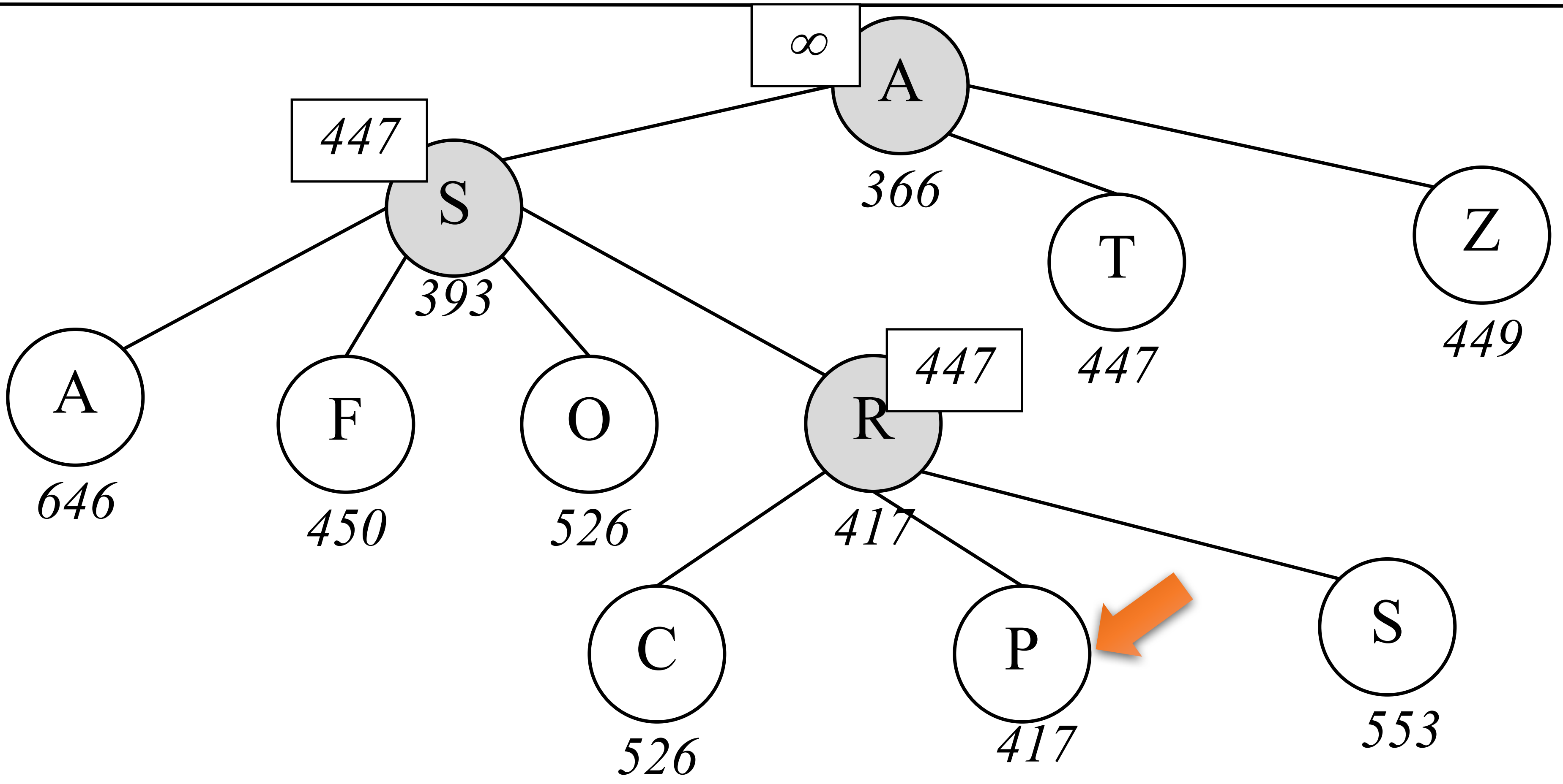
RBFS – مثال



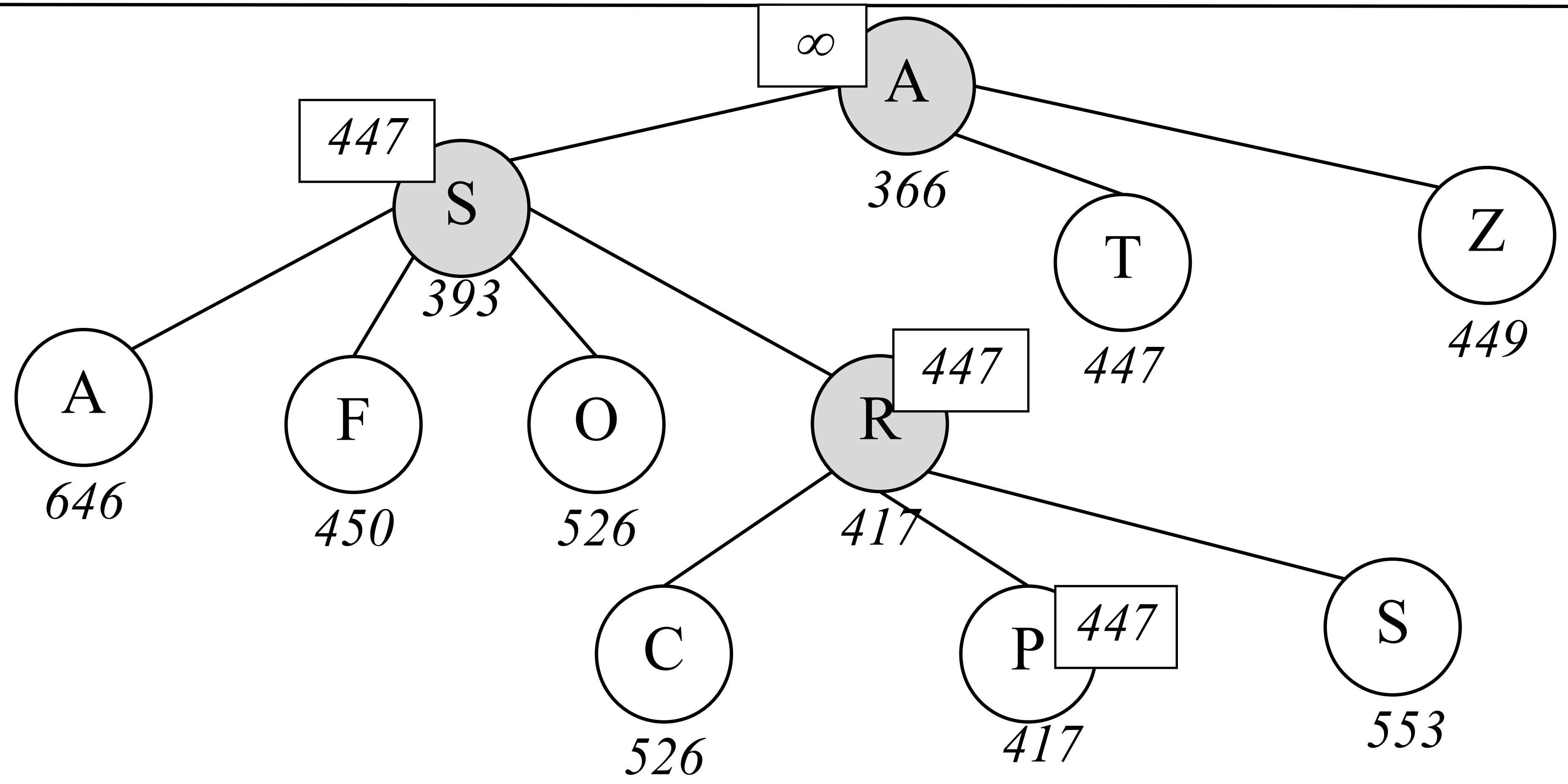
RBFS – مثال


∞
A
366
447
S
393
T
447
Z
449
A
646
F
450
O
526
447
R
417
C
526
P
417
S
553

RBFS – مثال

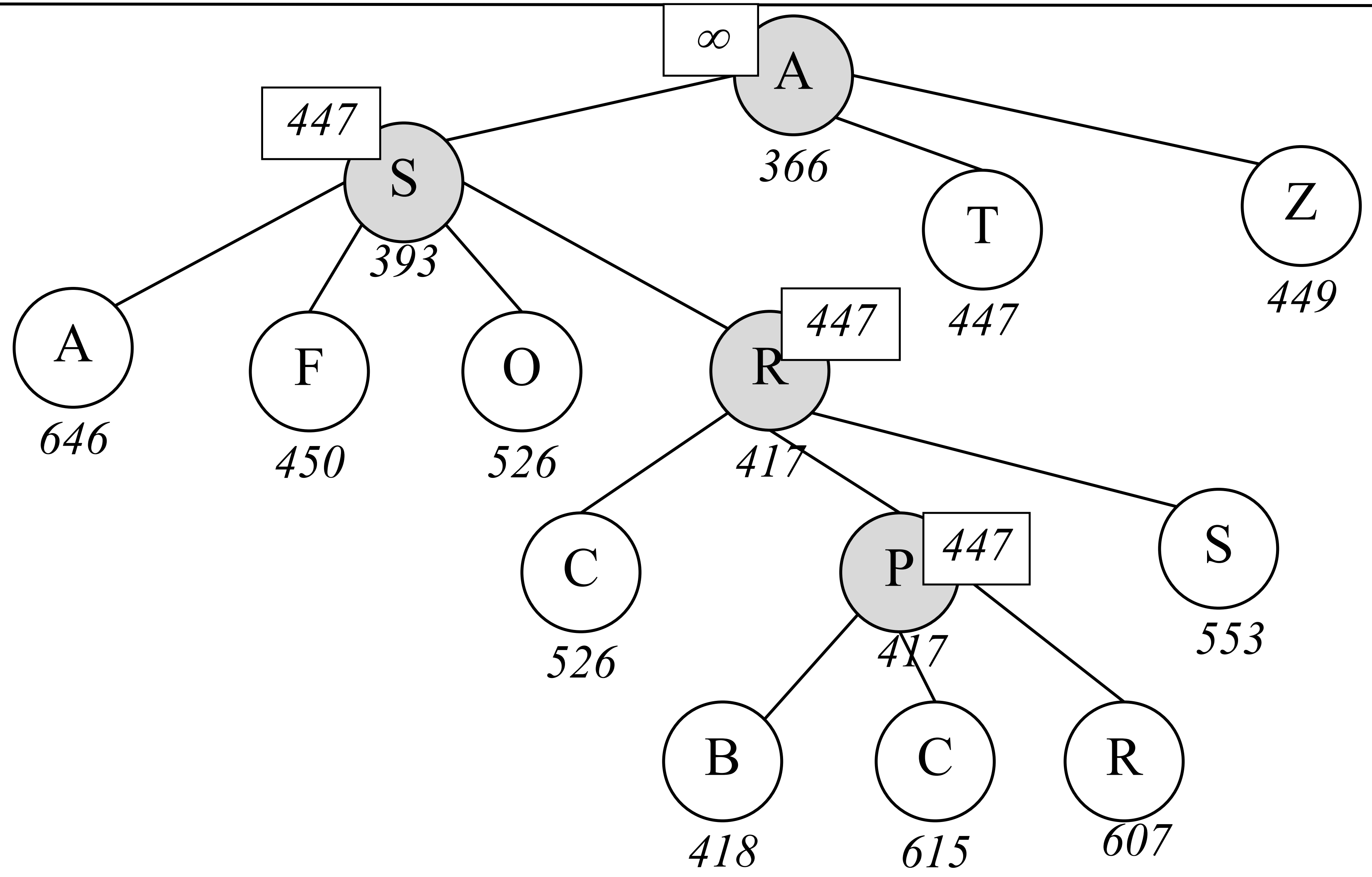

∞
A
366
447
S
393
T
447
Z
449
A
646
F
450
O
526
447
R
417
C
526
P
417
S
553

RBFS – مثال



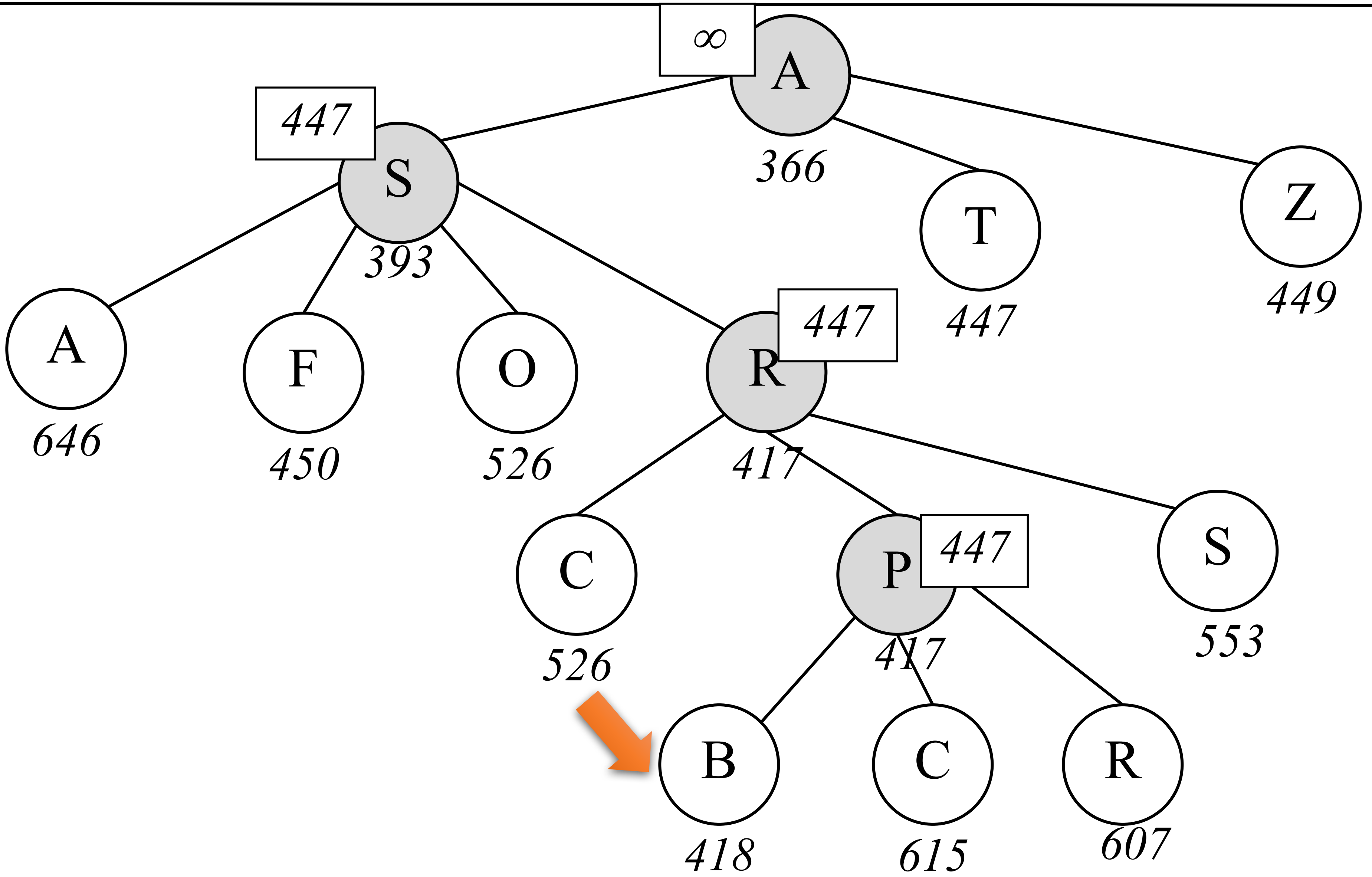
∞
A
366
447
S
393
T
447
Z
449
A
646
F
450
O
526
447
R
417
C
526
447
P
417
S
553

RBFS – مثال

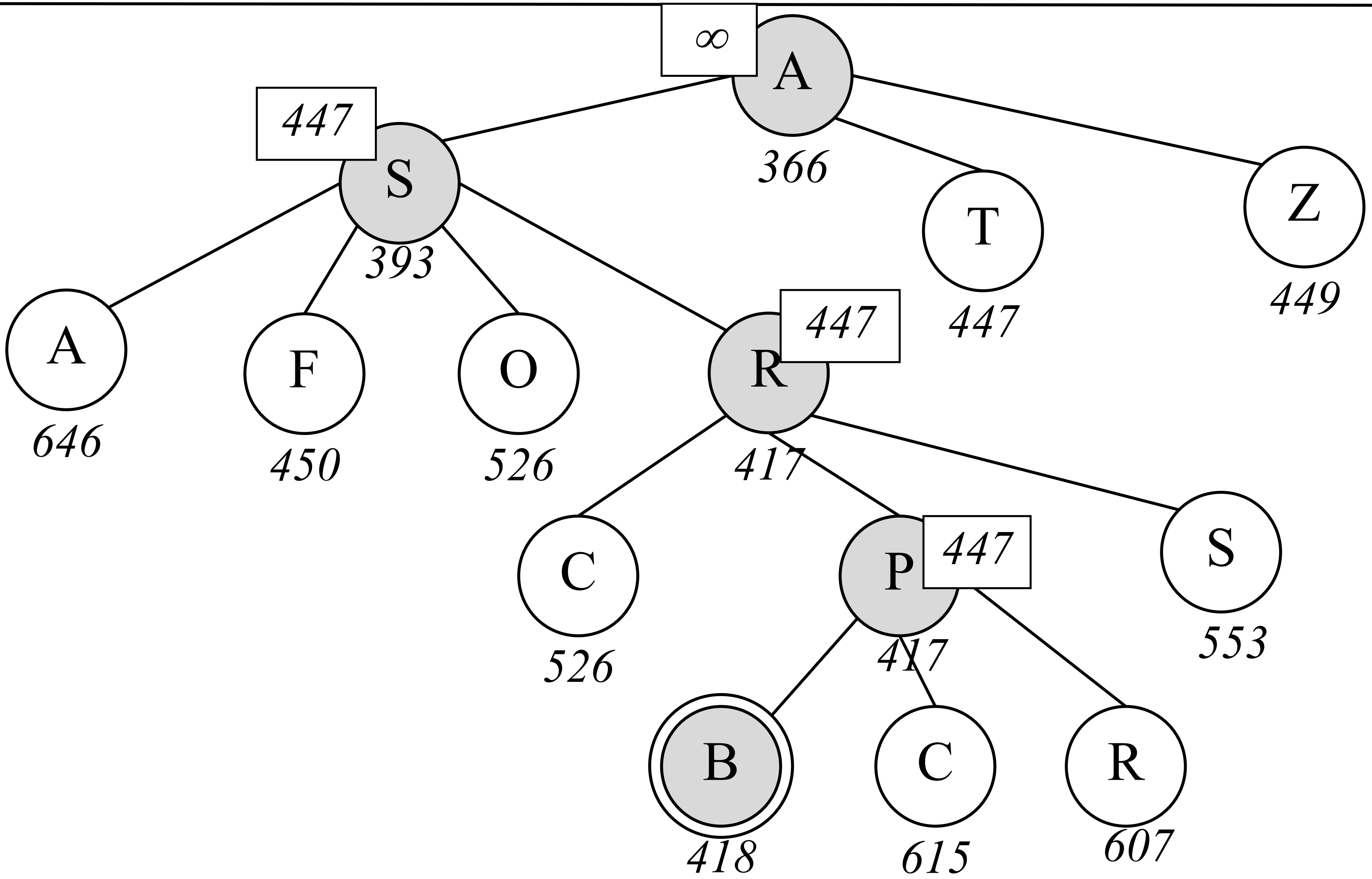


∞
A
366
447
S
393
T
447
Z
449
A
646
F
450
O
526
R
447
417
C
526
P
447
417
S
553
B
418
C
615
R
607

RBFS – مثال


∞
A
366
447
S
393
T
447
Z
449
A
646
F
450
O
526
447
R
417
C
526
P
447
417
S
553
B
418
C
615
R
607

RBFS – مثال



∞
A
366
447
S
393
T
447
Z
449
A
646
F
450
O
526
R
447
417
C
526
P
447
417
S
553
B
418
C
615
R
607

جستجوی اول بهترین بازگشتی – RBFS

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

function RBFS(problem, node, f_limit) returns a solution, or failure and a new f-cost limit
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  successors ← [ ]
  for each action in problem.ACTIONS(node.STATE) do
      add CHILD-NODE(problem, node, action) into successors
  if successors is empty then return failure, ∞
  for each s in successors do  /* update f with value from previous search, if any */
      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 RBFS

- کامل و بهینه؟
  - بله، اگر تابع هیوریستیک قابل قبول باشد بهینه است.
- پیچیدگی زمانی؟
  - RBFS و IDA* ممکن است یک گره را بیشتر از یک بار تولید کنند و بسط دهند زیرا نمی‌توانند حالت‌های تکراری را در غیر از مسیر فعلی بررسی کنند.
  - RBFS کمی از IDA* کارآمدتر است چرا که گره‌های کم‌تری را به‌طور مجدد تولید می‌کند.
- پیچیدگی فضایی؟
  - پیچیدگی فضایی خطی (وابسته به عمق عمیق‌ترین هدف بهینه) دارد چون درخت را به صورت عمقی پیمایش می‌کند.
  - این الگوریتم نیز اگر از جستجوی گرافی استفاده کند عملاً مشکلی را حل نکرده است!!

الگوریتم *SMA

- ایده: استفاده از تمامی حافظه موجود
  - یعنی، گسترش بهترین گره‌ی برگ تا زمانی که حافظه موجود پر شود.
- *SMA دقیقا مثل *A عمل می‌کند، یعنی تا زمانی که حافظه پر شود، بهترین گره (گره‌ای با کم‌ترین مقدار f) را گسترش می‌دهد.
- در صورت پر شدن حافظه همیشه بدترین گره برگ (گرهی با بیشترین مقدار f) را از حافظه حذف می‌کند
- مشابه با RBFS، اطلاعات گره‌ی حذف شده را در گره پدرش ذخیره می‌کند تا در صورتی که شاخه‌های دیگر به خوبی این شاخه نبودند دوباره به این شاخه برگردد.

الگوریتم *SMA

- ممکن است f-cost تمام گره‌های برگ با هم برابر باشند در این‌صورت ممکن است گره‌ای که برای بسط دادن انتخاب شده، (به علت پر بودن حافظه) برای حذف نیز انتخاب شود! برای پرهیز از این مشکل، *SMA همیشه بهترین گره برگی که از همه جدیدتر (عمیق‌تر) است را برای بسط دادن انتخاب می‌کند و همیشه بدترین گره‌ای که از همه قدیمی‌تر (کم عمق‌تر) است را برای حذف انتخاب می‌کند.
- ممکن است به حالتی برسیم که فقط یک برگ (که هدف هم نیست) در درخت باشد و حافظه نیز پر باشد. در این‌صورت نمی‌توان آن گره برگ را بسط داد (چون حافظه خالی نداریم) و اگر آن برگ در مسیر بهینه باشد الگوریتم *SMA نمی‌تواند با مقدار حافظه موجود مسیر بهینه را پیدا کند. در این موارد، *SMA، مقدار f-cost این برگ را بی‌نهایت می‌گذارد تا دیگر انتخ اب نشود.

SMA* الگوریتم

```
function SMA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: Queue, a queue of nodes ordered by f-cost

  Queue ← MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})
  loop do
     if Queue is empty then return failure
     n ← deepest least-f-cost node in Queue
     if GOAL-TEST(n) then return success
     s ← NEXT-SUCCESSOR(n)
     if s is not a goal and is at maximum depth then
        f(s) ← ∞
     else
        f(s) ← MAX(f(n), g(s)+h(s))
     if all of n's successors have been generated then
        update n's f-cost and those of its ancestors if necessary
     if SUCCESSORS(n) all in memory then remove n from Queue
     if memory is full then
        delete shallowest, highest-f-cost node in Queue
        remove it from its parent's successor list
        insert its parent on Queue if necessary
     insert s on Queue
  end
```

مراحل هر فاز SMA*

۱- اگر صف خالی است $failure$ برگردان.

۲- کم‌هزینه‌ترین گره موجود در صف را در n قرار بده. (در شرایط یکسان عمیق‌ترین گره را انتخاب کن و در n قرار بده)

۳- اگر n هدف بود راه‌حل پیدا شده را برگردان.

۴- فرزند بعدی n را که هنوز تولید نشده است را در s قرار بده.

۵- اگر s در ماکزیمم عمق ممکن قرار داشت و هدف نبود آن‌گاه مقدار f آن را بی‌نهایت بگذار در غیر این صورت از رابطه $f(s)=max\{f(n), h(s)+g(s)\}$ برای مقداردهی آن استفاده کن.

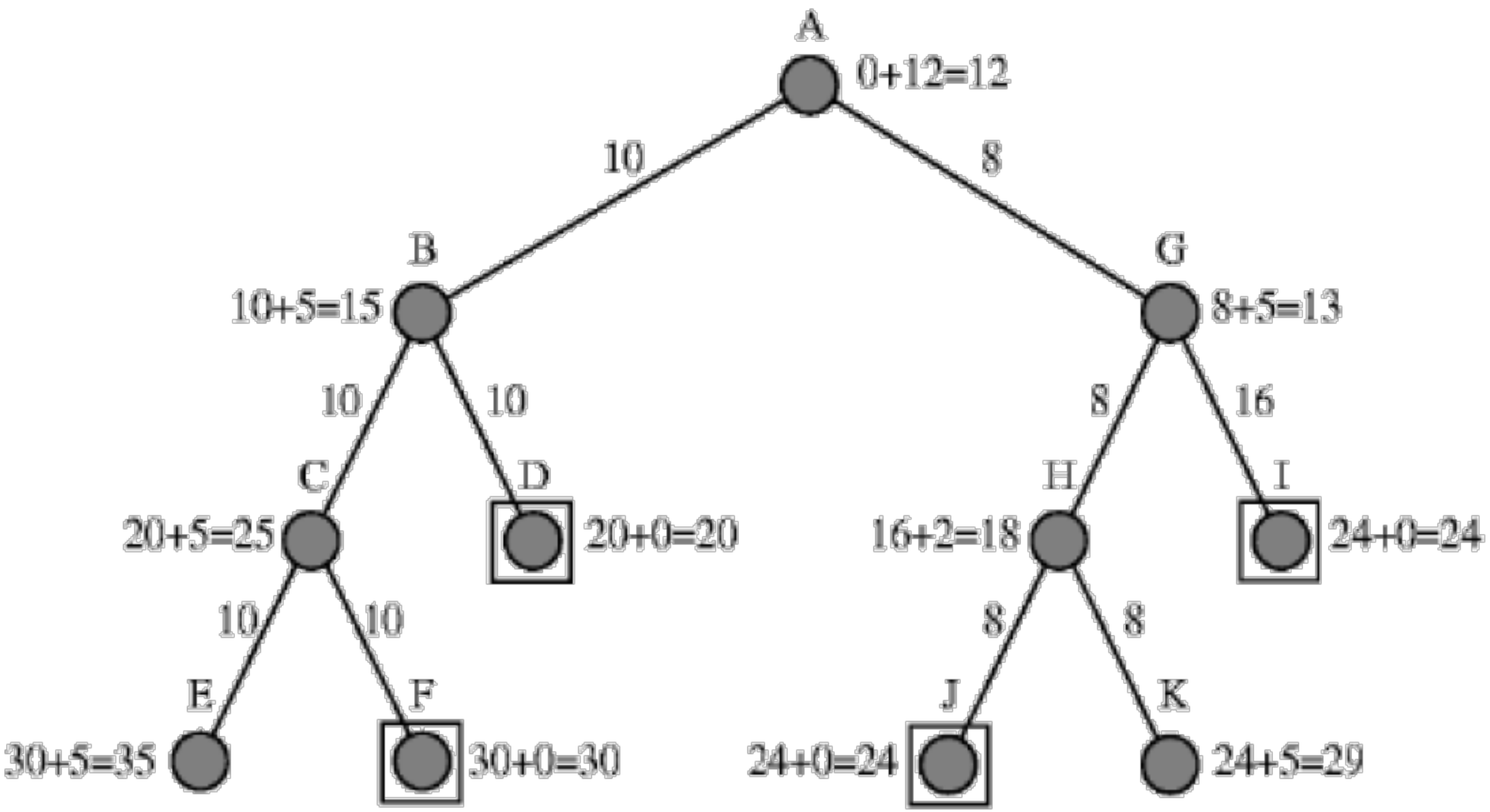
۶- اگر همه فرزندان n تولید شده باشد مقدار f گره n و ماقبل‌های آن را تصحیح کن.

۷- اگر همه فرزندان n در حافظه بود آن‌گاه n را از صف حذف کن.

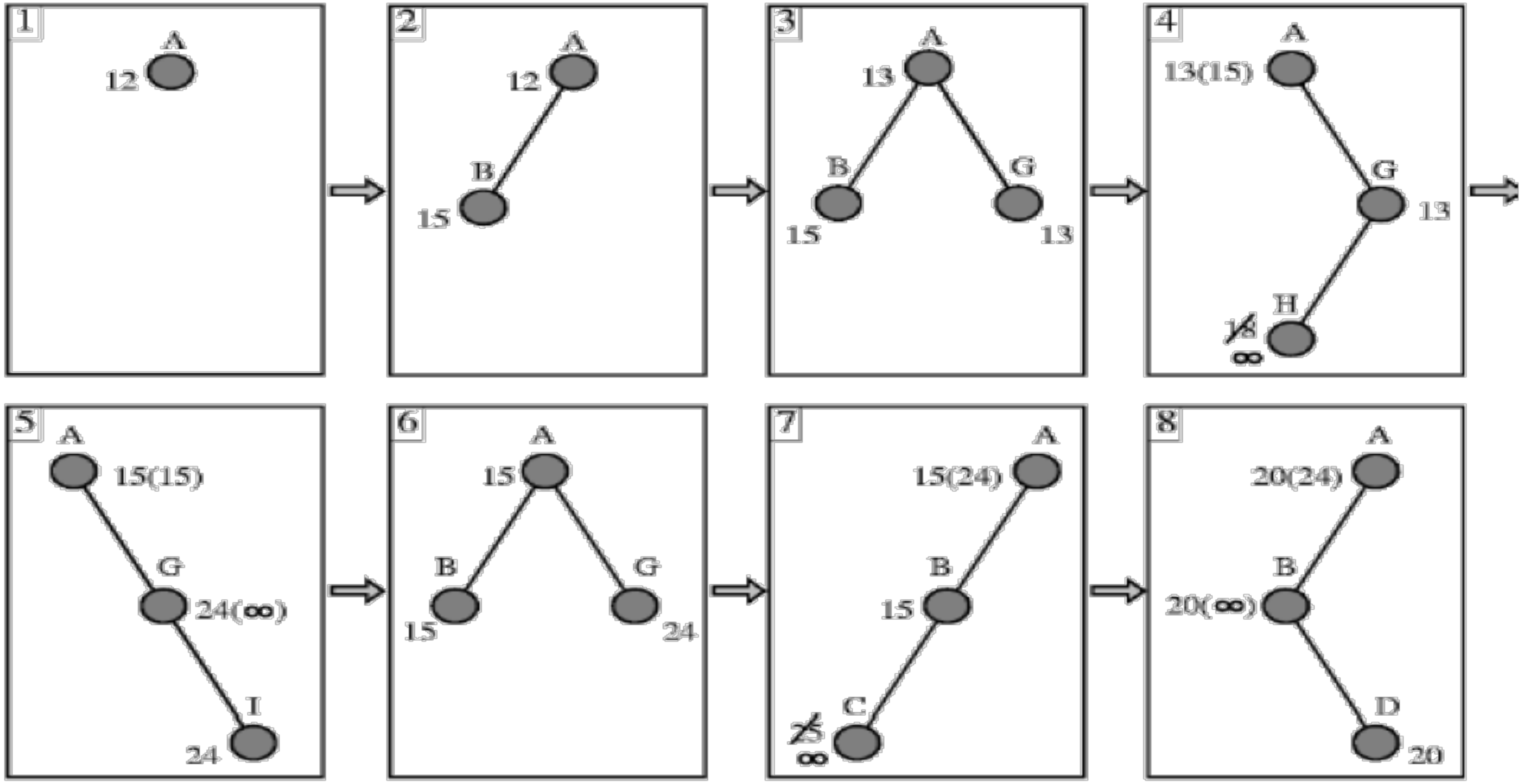
۸- اگر حافظه پر بود آن‌گاه گره‌ای با بیشترین مقدار f را از صف حذف کن (در شرایط یکسان کم‌عمق‌ترین گره را حذف کن). این گره و مقدار f آن را به‌عنوان یادآور والدش (یعنی n) ذخیره کن.

۹- s را به صف اضافه کن.

مثال *SMA با محدودیت ۳ خانه حافظه



مثال *SMA با محدودیت ۳ خانه حافظه



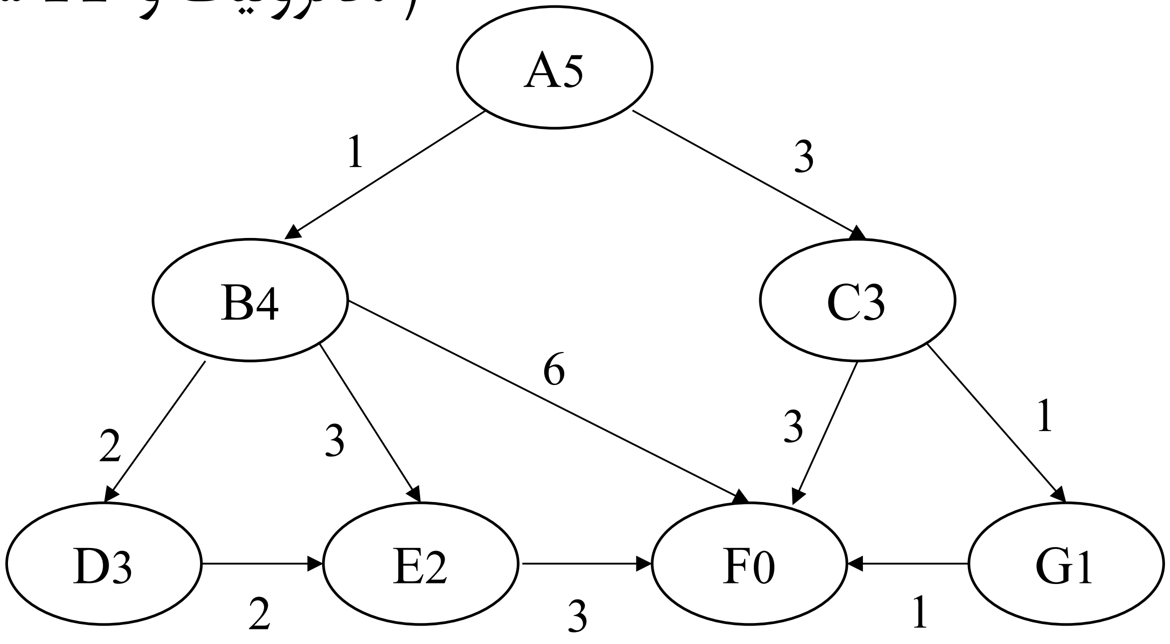
ارزیابی *SMA

- کامل؟
  - بله، اگر عمق سطحی‌ترین گره هدف کمتر یا مساوی اندازه حافظه باشد.
- بهینگی؟
  - بله، اگر عمق راه‌حل بهینه کم‌تر از اندازه حافظه (برحسب تعداد گره) باشد. درغیراین‌صورت بهترین راه‌حل قابل دستیابی با توجه به حافظه محدود را پیدا می‌کند.
- در عمل، ممکن است *SMA بهترین الگوریتم همه‌منظوره برای یافتن راه‌حل‌های بهینه باشد.
  - به‌خصوص اگر فضای حالت به صورت گراف باشد، هزینه‌های مرحله‌ای یکسان نباشند و هزینه تولید یک گره بیشتر از سربار نگه‌داری لیست‌های باز و بسته باشد.
  - در مسائل پیچیده ممکن است *SMA دائماً از یک شاخه به شاخه دیگری برود و دوباره مجبور شود به شاخه اول برگردد (این حالت مشابه مسئله کوبیدگی در سیستم عامل است)

تست ۴

حاصل جستجوی *sma با حداکثر ۳ خانه حافظه بر روی گراف زیر چیست؟ (A گرهی شروع و F گرهی هدف است. اعداد روی یال‌ها هزینه‌ی مسیر و اعداد داخل گره‌ها هزینه‌ی تخمینی گره تا هدف است. ترتیب ملاقات فرزندان به ترتیب حروف الفبا است.)

(مکاترونیک و IT سال ۸۸)



۱- ACF ✓

۲- ABF

۳- ACGF

۴- *SMA پاسخی برای حل این مسئله پیدا نمی‌کند.

تست ۵

کدام یک از جملات زیر صحیح است؟ (فناوری اطلاعات ۸۵)

۱– الگوریتم SMA^* همیشه سریعتر از A^* به جواب می‌رسد.

۲– جستجوهای کور همیشه نیاز به حافظه‌ی کمتری نسبت به جستجوهای مطلع دارند.

۳– الگوریتم اول عمق همیشه با صرف مقدار کمتری از حافظه نسبت به الگوریتم اول پهنا به جواب می‌رسد.

✓ ۴– الگوریتم جستجوی A^* با هر هیوریستیکی همیشه تعداد کمتری گره نسبت به هر الگوریتم مطلع دیگر با همان هیوریستیک بسط می‌دهد.

تست ۶

در مقایسه بین روش‌های مختلف جستجو از نظر حافظه‌بری، اگر بخواهیم روش‌ها را از پیچیده‌ترین تا ساده‌ترین (از نظر پیچیدگی حافظه) مرتب نماییم، کدام گزینه در اغلب موارد صحیح است ؟ (مهندسی کامپیوتر ۸۷)

۱- $RBFS \rightarrow BFS \rightarrow SMA^* \rightarrow A^*$

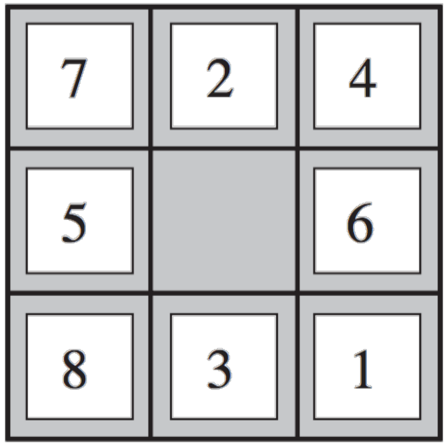
۲- $RBFS \rightarrow BFS \rightarrow A^* \rightarrow SMA^*$

۳- $BFS \rightarrow A^* \rightarrow RBFS \rightarrow SMA^*$

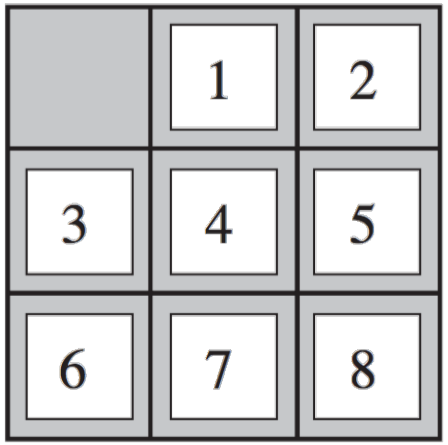
۴- $BFS \rightarrow A^* \rightarrow SMA^* \rightarrow RBFS$ ✓

توابع هیوریستیک

فضای حالت مسئله پازل ۸تایی



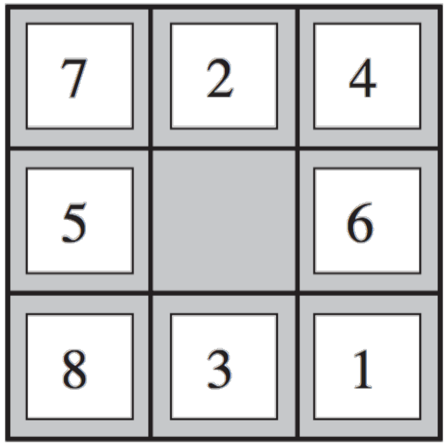
Start State



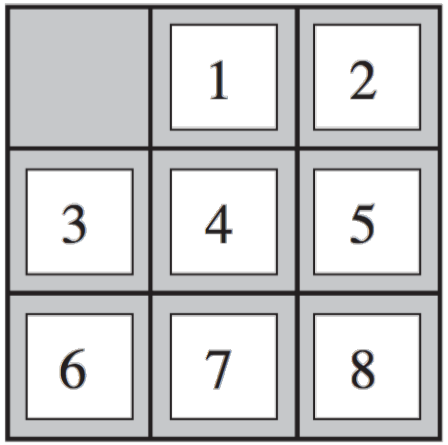
Goal State

- ضریب انشعاب: حدود ۳ (چرا؟)
- به‌طور میانگین محاسبه‌شده است که پس از طی حدود ۲۲ گام به حالت هدف پازل ۸تایی خواهیم رسید. (یعنی عمق ۲۲)
- فضای حالت بزرگ
  - تعداد حالات جستجوی درختی: $3^{22} \approx 3.1\times10^{10}$
  - تعداد حالات جستجوی گرافی: $9!/2 \approx 181440$
- درنتیجه یک تابع هیوریستیک می‌تواند فرایند جستجو را کاهش دهد.

توابع هیوریستیک قابل قبول پازل ۸تایی



Start State



Goal State

- h_1 = تعداد کاشی‌هایی که در جای درست خود قرارنگرفته‌اند.
  - هر مربعی که سر جای خود نباشد حداقل یک خانه باید جابه‌جا شود تا به وضعیت به هدف برسد.
  - $h_1(s)=8$
- h_2 = مجموع فاصله‌های منهتن هر یک از مربع‌ها با جای مورد انتظارش (مجموع فواصل افقی و عمودی کاشی‌ها تا موقعیت هدف)
  - هر مربعی که سر جای خود نیست باید حداقل به اندازه‌ی فاصله منهتن آن مربع تا مکان مورد انتظارش جابه‌جا شود تا به وضعیت هدف برسد.
  - $h_2(s)=3+1+2+2+2+3+3+2=18$

تأثیر دقت هیوریستیک بر روی کارایی

- ضریب انشعاب مؤثر Effective branching factor یا b*: ضریب انشعابی است که یک درخت پر به عمق d باید داشته باشد تا حاوی N+1 گره باشد.
  - d: عمق راه‌حل در مسئله مورد نظر
  - N: تعداد گره‌های تولیدشده توسط الگوریتم A* تا رسیدن به این راه‌حل

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

- ضریب مؤثر انشعاب می‌تواند در نمونه‌های مختلف یک مسئله متفاوت باشد، ولی معمولاً برای مسائلی که به اندازه کافی سخت هستند، نسبتا ثابت است.
  - b* راهنمایی خوب در مورد سودمندی کلی هیوریستیک است.
  - مقدار b* برای یک هیوریستیک خوب به یک نزدیک است.

تأثیر دقت هیوریستیک بر روی کارایی

- مقایسه هزینه‌های جستجو و ضرایب مؤثر انشعاب
  - مقادیر نشان داده شده، متوسطِ ۱۰۰ نمونه از پازل ۸تایی برای هر عمق راه‌حل است.

	Search Cost (nodes generated)			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

کیفیت هیوریستیک

- اگر رابطه‌ی $h_2(n) \geq h_1(n)$ برای هر n از هیوریستیک‌های قابل قبول h_1 و h_2 برقرار باشد آن‌گاه می‌گوییم h_2 بر h_1 غلبه (dominate) می‌کند و برای جستجو مناسب‌تر است.
  - اگر h_2 بر h_1 غلبه داشته باشد، هر گره‌ای که توسط A^* به همراه h_2 بسط می‌یابد توسط A^* به همراه h_1 نیز بسط می‌یابد و ممکن است h_1 منجر به بسط گره‌های بیشتری نیز شود. (چرا؟)
  - مشکل هیوریستیکی که برای هر n رابطه‌ی $h(n)=h^*(n)$ برقرار است، چیست؟
- اگر برای مسئله‌ای دو هیوریستیک داشته باشیم که هیچ‌کدام بر دیگری غلبه نکند، چه باید کرد؟
  - ماکزیمم هیوریستیک‌های قابل قبول یک هیوریستیک قابل قبول و دقیق‌تر است.

$$h(n) = max(h_1(n), h_2(n))$$

تست ۷

فرض کنید سه تابع مکاشفه‌ای $h_1(n)$، $h_2(n)$ و $h_3(n)$ قابل قبول باشند. یک الگوریتم A^* با کدام یک از توابع زیر جواب بهینه را تولید می‌کند؟ (فناوری اطلاعات ۹۳)

۱) $h(n) = h_1(n) + h_2(n) + h_3(n)$

۲) $h(n) = h_1(n) \times h_2(n) \times h_3(n)$

۳) $h(n) = \max\left(\min\left(h_1(n), h_2(n), h_{3(n)}\right), h_1(n) \times h_2(n) \times h_3(n), h_1(n) + h_2(n) + h_3(n)\right)$

۴) $h(n) = \min\left(\max\left(h_1(n), h_2(n), h_{3(n)}\right), h_1(n) \times h_2(n) \times h_3(n), h_1(n) + h_2(n) + h_3(n)\right)$ ✓

تولید هیوریستیک‌ها

مسائل تعدیل‌شده (Relaxed Problem)

- اگر قوانین و محدودیت‌های اعمال یک مسئله را کم کنیم آن مسئله به یک مسئله‌ی راحت تبدیل می‌شود.
- راه‌حل بهینه برای مسئله‌ی تعدیل‌شده یک هیوریستیک قابل قبول برای مسئله‌ی اصلی است.
- از آن‌جا که هیوریستیک به‌دست آمده، یک هزینه‌ی دقیق برای مسئله تعدیل‌شده می‌باشد باید از نامساوی مثلث تبعیت کند و در نتیجه سازگار است.
- مسائل تولید شده باید بتوانند اساسا بدون جستجو حل شوند.

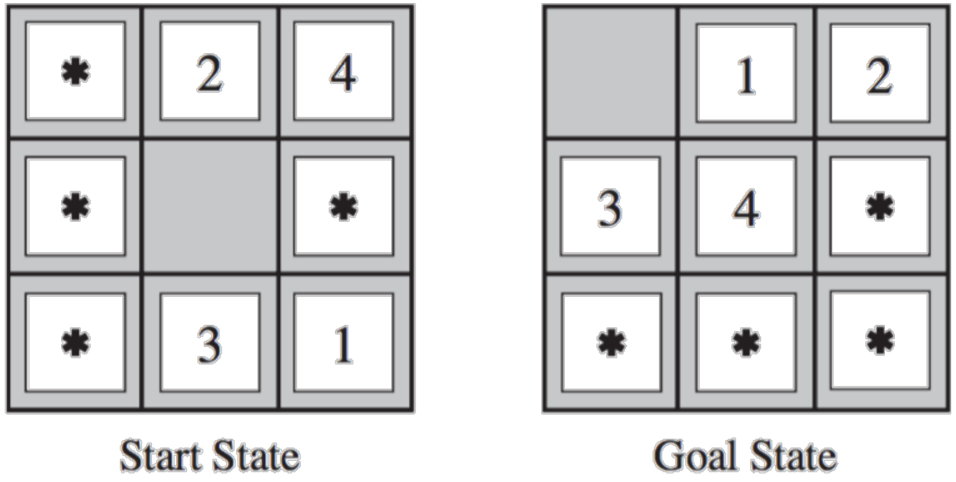
تولید هیوریستیک‌ها

- برای مثال در مسئله پازل ۸تایی قانون این است که مربع A به مربع B قابل انتقال است اگر B مجاور با A باشد و B خالی باشد.
- مسائل تعدیل‌شده:
  - مربع A می‌تواند به مربع B حرکت کند اگر مجاور آن باشد. (صرف نظر از خالی یا پر بودن آن)
  - مربع A می‌تواند به مربع B منتقل شود اگر B خالی باشد. (صرف‌نظر از مجاورت)
  - مربع A می‌تواند به مربع B منتقل شود. (صرف‌نظر از هر دو شرط)
- هیوریستیک‌های قابل قبول برای مسئله‌ی اصلی (h_1 و h_2)، هزینه‌های مسیر بهینه برای مسائل تعدیل‌شده هستند.
  - مورد اول: یک مربع می‌تواند به هر مربع همسایه منتقل شود. ← هیوریستیک فاصله منهتن $h_2(n)$
  - مورد سوم: یک مربع می‌تواند به هر جایی انتقال یابد.← $h_1(n)$

تولید هیوریستیک‌ها

بانک‌های اطلاعاتی الگو (Pattern database)

- هزینه راه‌حل برای یک زیرمسئله از یک مسئله خاص
  - ذخیره هزینه راه‌حل‌های دقیق برای هر زیرمسئله ممکن
- قابل قبول؟
  - هزینه راه‌حل بهینه برای این زیرمسئله یک حد پایین برای هزینه مسئله کامل است.



تولید هیوریستیک‌ها

- با در نظر گرفتن زیرمسائل مختلف و در حقیقت داشتن بانک‌های اطلاعاتی الگوی مختلف، می توان هیوریستیک‌های حاصل را با برداشتن مقدار بیشینه با هم ترکیب کرد.
  - پازل ۱۵تایی: ۱۰۰۰ برابر تعداد نود کم‌تری نسبت به استفاده از h_2 تولید می‌کند.
- آیا جمع زدن هیوریستیک‌های زیرمسائل مختلف به‌جای ماکزیمم گرفتن منجر به یک هیوریستیک قابل قبول می‌شود؟
  - برای مثال، هیوریسیتک بانک اطلاعاتی ۱–۲–۳–۴ و هیوریستیک بانک اطلاعاتی ۵–۶–۷–۸
  - خیر، زیرا برای یک حالت خاص تقریبا به‌طور قطع چند حرکت مشترک دارند.

تولید هیوریستیک‌ها

- اگر به‌جای کل هزینه‌ی حل زیرمسئله ۱-۲-۳-۴ تنها تعداد حرکاتی را بشماریم که شامل ۱-۲-۳-۴ می‌شود و برای زیرمسئله ۵-۶-۷-۸ نیز همین کار را انجام دهیم، دو بانک اطلاعاتی مجزا (disjoint pattern database) به‌دست آورده‌ایم.
- در این حالت، مجموع هیوریستیک‌ها یک هیوریستیک قابل قبول خواهد بود.
  - پازل ۱۵تایی: ۱۰۰۰۰ برابر تعداد نود کم‌تری نسبت به استفاده از h_2 تولید می‌کند.
  - پازل ۲۴تایی: ۱۰۰۰۰۰۰ برابر تعداد نود کم‌تری نسبت به استفاده از h_2 تولید می‌کند.

تولید هیوریستیک‌ها

یادگیری هیوریستیک‌ها از تجربه

- یادگیری $h(n)$ از نمونه‌هایی که قبلا به‌طور بهینه حل شده‌اند. (پیش‌بینی هزینه‌ی راه‌حل برای حالات دیگر)
- روش‌های مورد استفاده: شبکه‌های عصبی، درخت‌های تصمیم‌گیری و دیگر روش‌های یادگیری استقرایی.
  - استفاده از ویژگی‌های حالت به جای توصیف خام حالت برای تولید هیوریستیک بهتر
- مثال: گردآوری نمونه‌های حل‌شده‌ی زیادی از مسئله پازل ۸تایی
  - $X_1(n)$: تعداد کاشی‌هایی که در جای درست خود قرار نگرفته‌اند.
  - $X_2(n)$: تعداد زوج کاشی‌هایی که در حالت هدف نیز مجاور همدیگر قرار دارند.
  - استفاده از ترکیبی از ویژگی‌ها برای پیش‌بینی بهتر $h(n)$ (برای مثال ترکیب خطی وزن‌دار)