



دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر و فناوری اطلاعات

راهبردهای جستجوی آگاهانه

«هوش مصنوعی: یک رهیافت نوین»، فصل ۳

مدرس: سیده فاطمه موسوی

نیمسال دوم ۱۳۹۸-۱۳۹۹

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^* و $RBFS$
 - توابع هیوریستیک

راهبرد جستجوی اول-بهترین

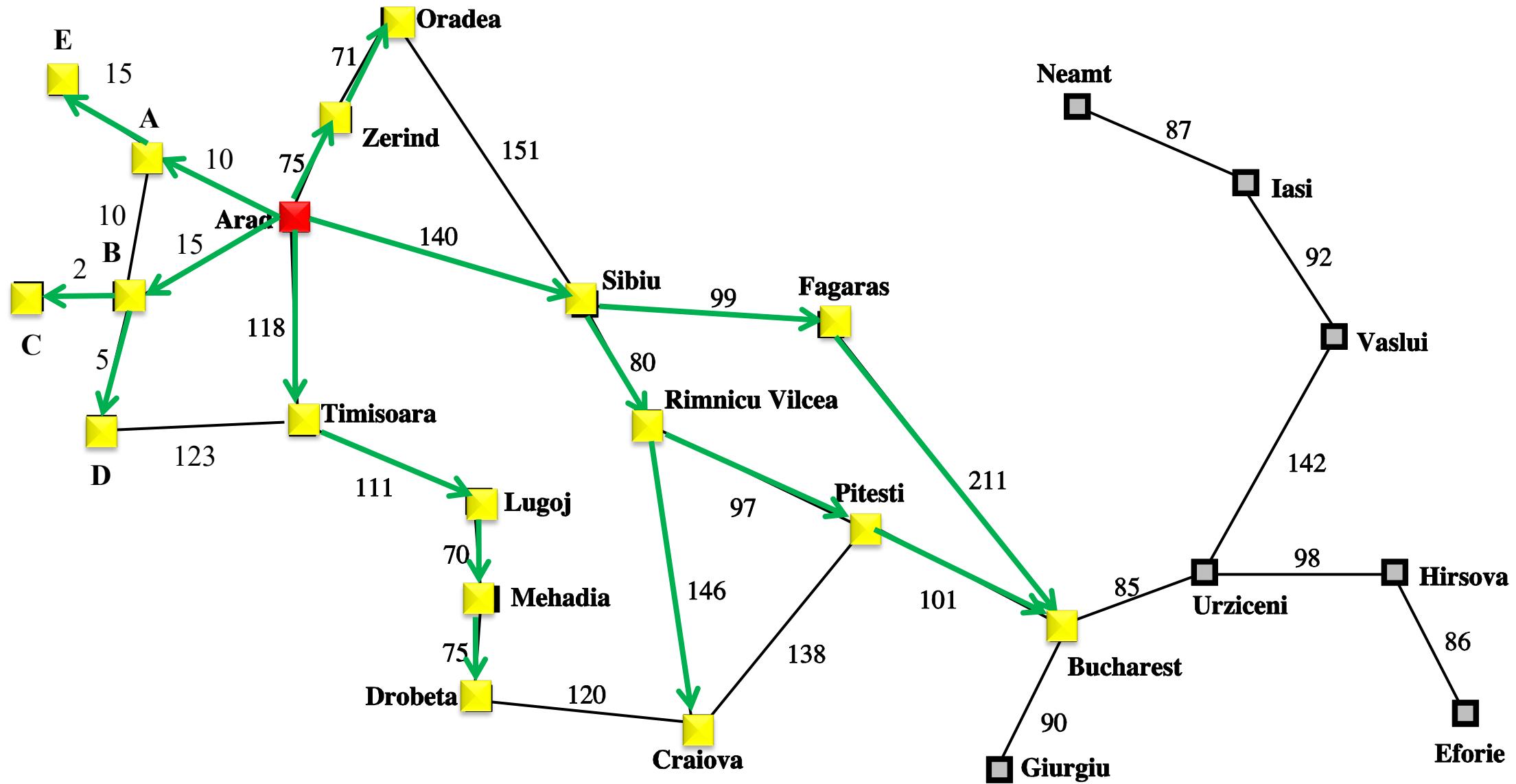
جستجوی اول بهترین – Best First

- ایده: استفاده از یک تابع ارزیاب ($f(n)$) (Evaluation function) 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.

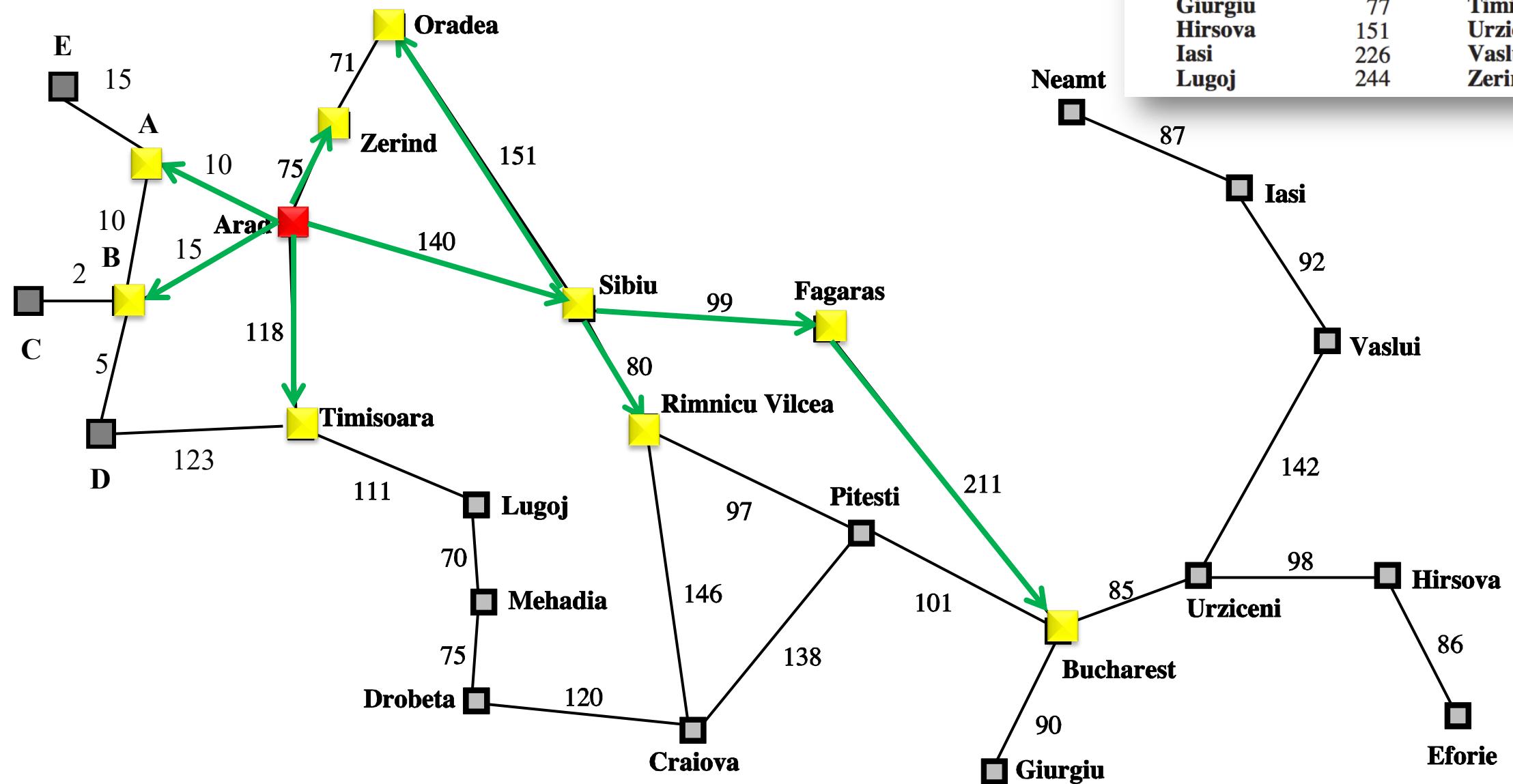
ارتباط جستجوهای ناآگاهانه و جستجوی اول بهترین

- 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؟
 - $f(n)=\text{depth}(n)$
- جستجوی اول عمق؟
 - $f(n)=-\text{depth}(n)$ یا $f(n)=1/\text{depth}(n)$

ایده کلی جستجوهای آگاهانه



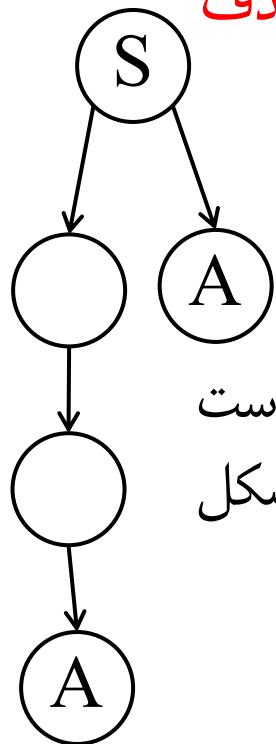
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



γ

تابع هیوریستیک

- وارد کردن دانش خاص مسئله در جستجو
- اطلاعاتی بیشتر از تعریف مسئله به منظور رسیدن هرچه سریع‌تر به یک راه حل بهینه
- تابع هیوریستیک می‌تواند به عنوان جزئی از تابع ارزیاب $f(n)$ تعریف شود.
- $h(n) = h(n)$ 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 به دست آمده باشد که باعث محدود کردن یا کاهش دادن جستجو در دامنه‌هایی می‌شود که پیچیده و مشکل هستند.

جستجوی اول-بهترین حریصانه

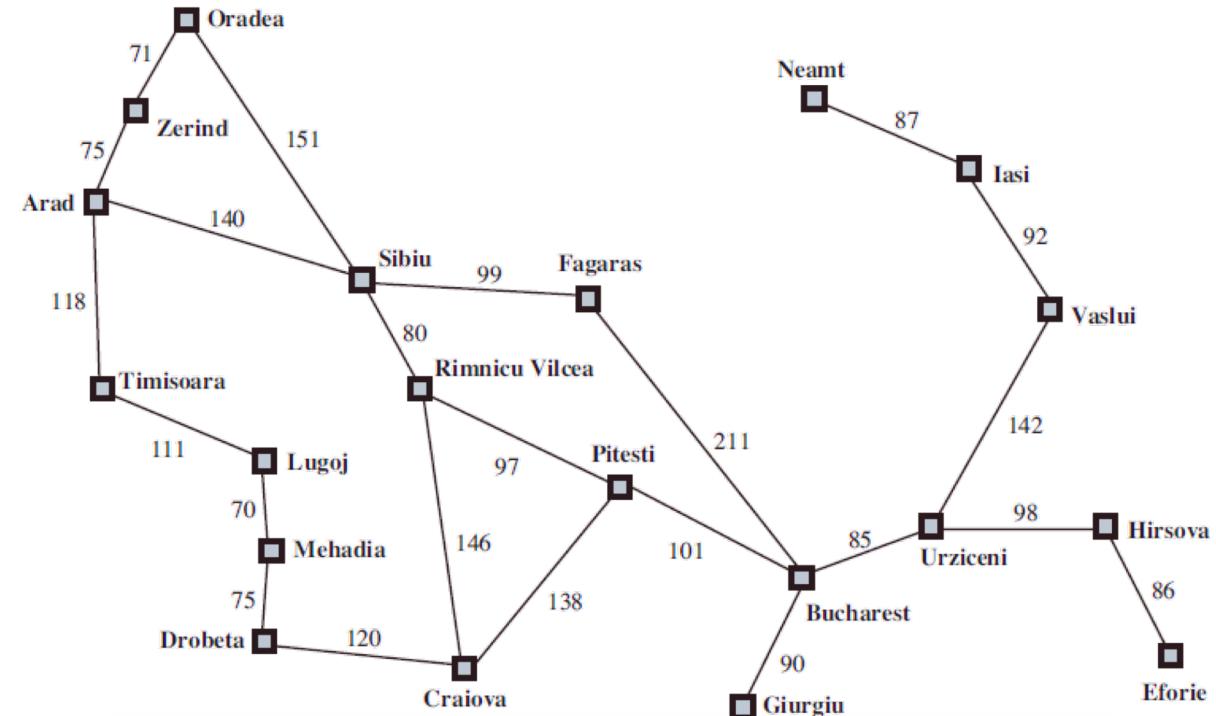
- نودی را بسط می‌دهد که به نظر می‌رسد به هدف نزدیک‌تر است.

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

- برای مثال:

$f(n) = h_{\text{std}}(n)$ = 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.



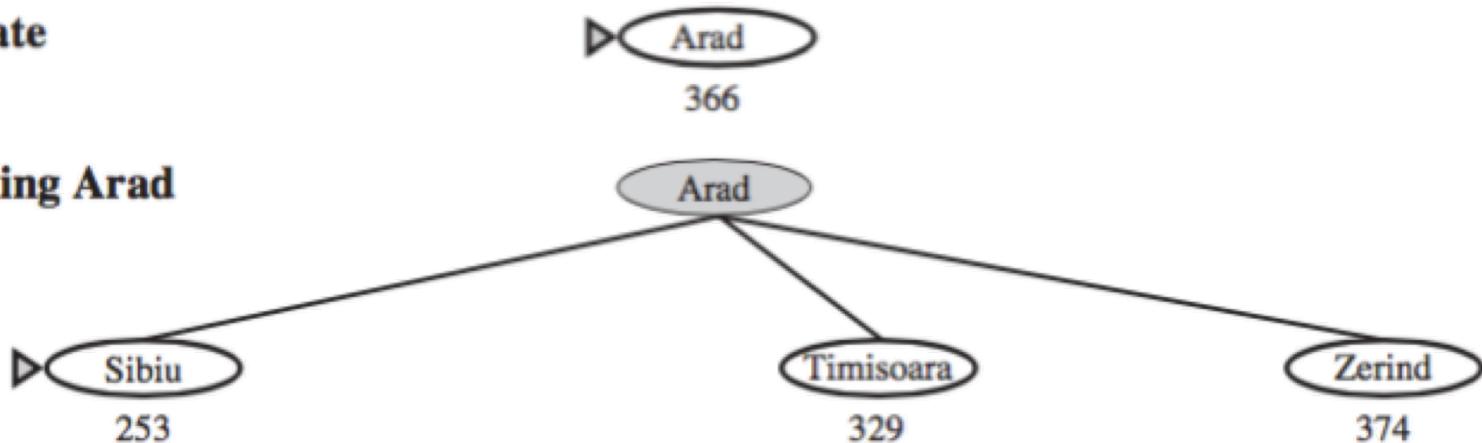
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

جستجوی حریصانه - مثال

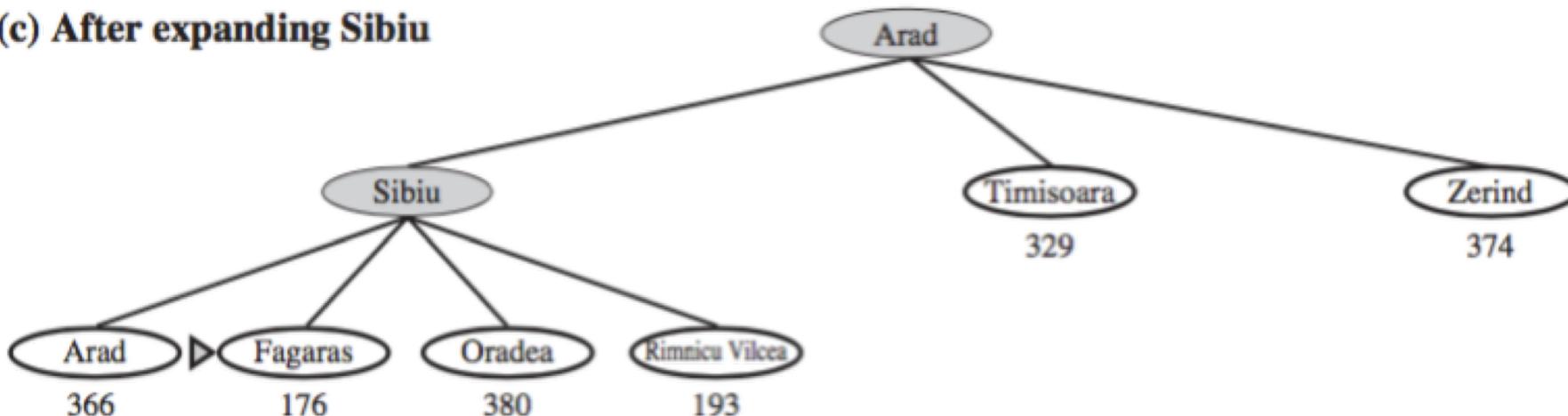
(a) The initial state



(b) After expanding Arad

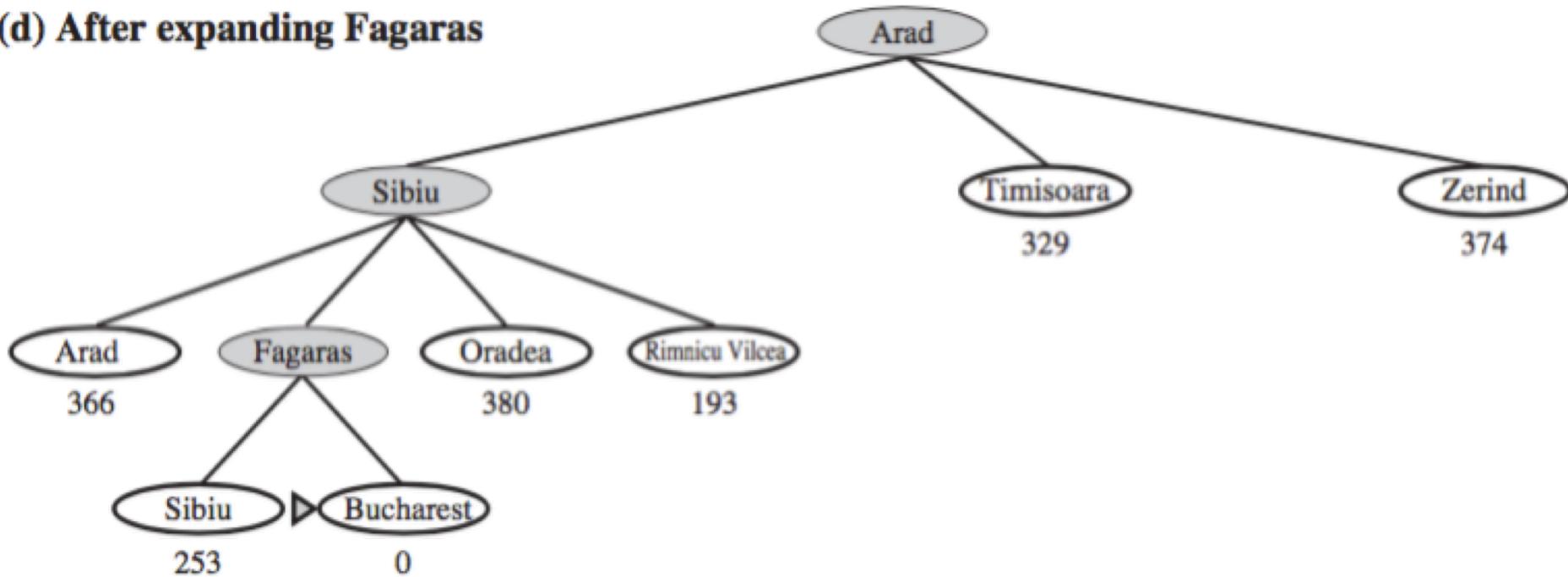


(c) After expanding Sibiu



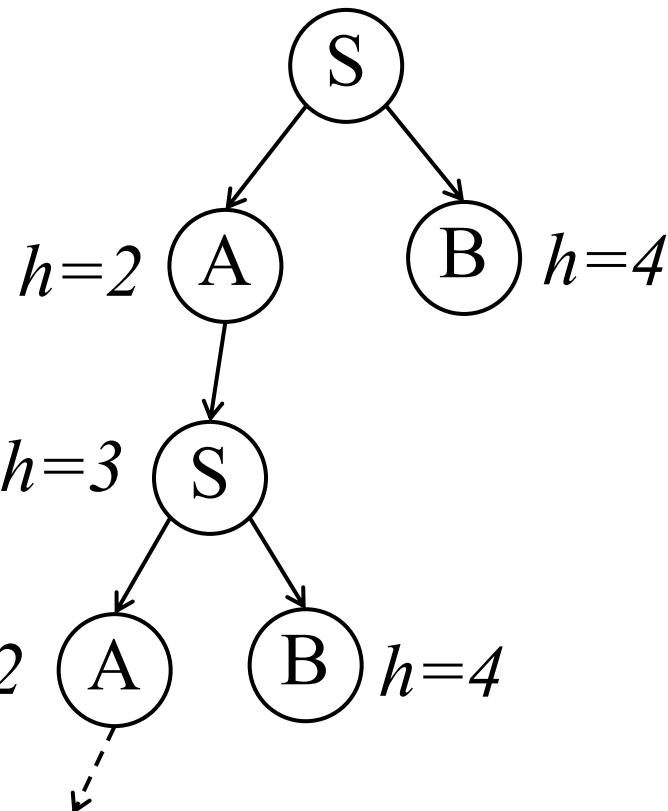
جستجوی حریصانه - مثال

(d) After expanding Fagaras

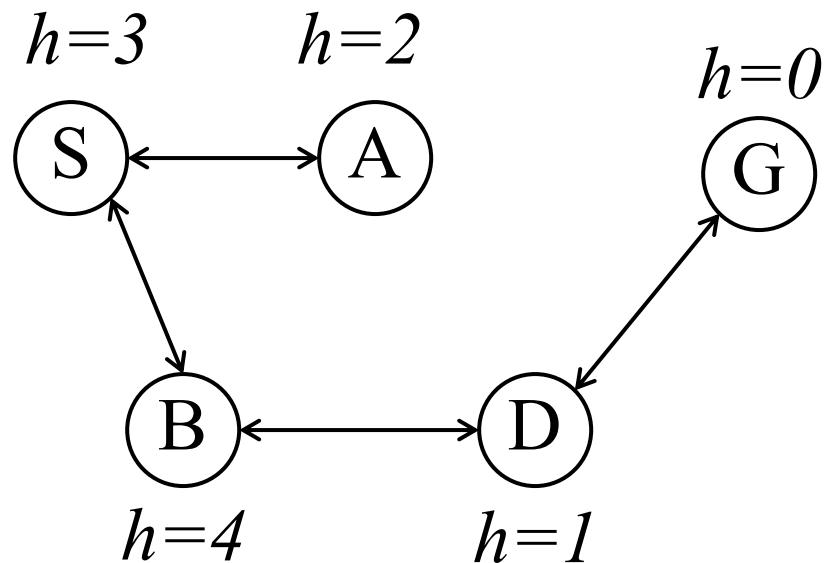


- به هدف رسید!!
- اما بهینه نیست مسیر *Arad, Sibiu, Rimnicu Vilcea, Pitesti* را مشاهده کنید.

ارزیابی جستجوی اول-بهترین حریصانه



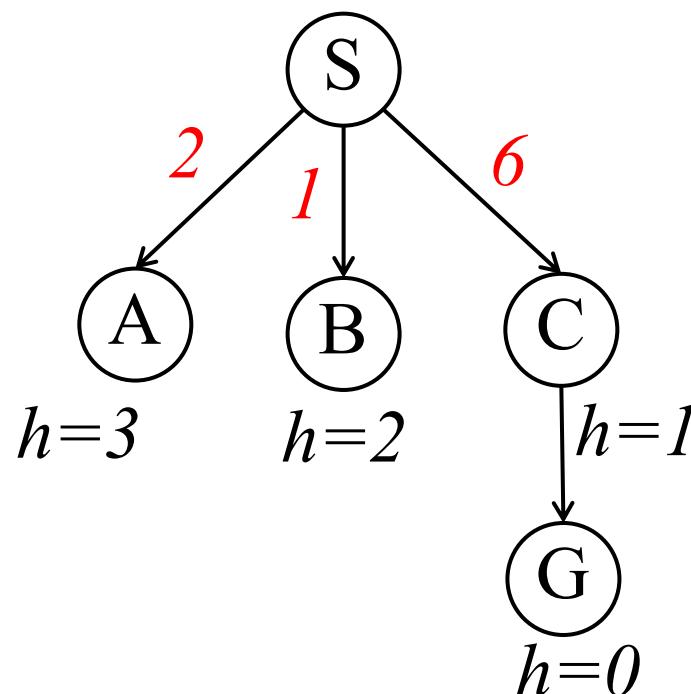
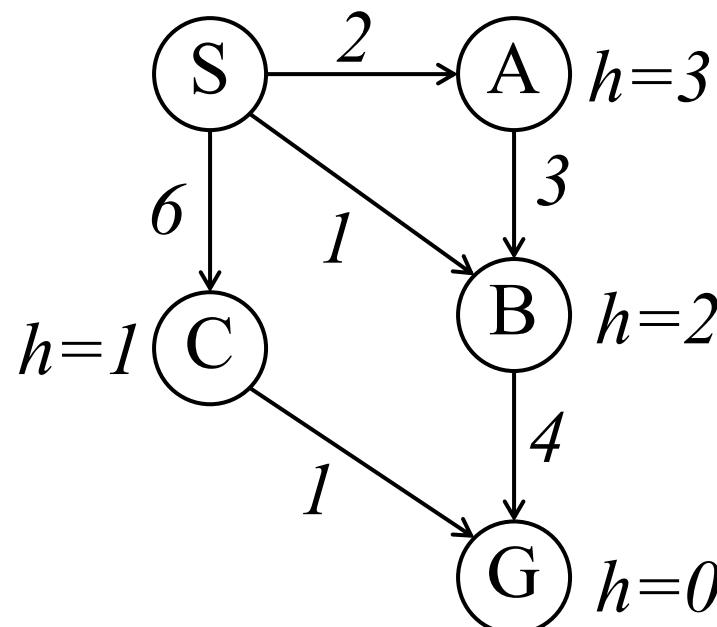
- کامل؟
- جستجوی درختی: خیر به دلیل حلقه‌های نامتناهی
- جستجوی گرافی: بله اگر فضای جستجو متناهی باشد.



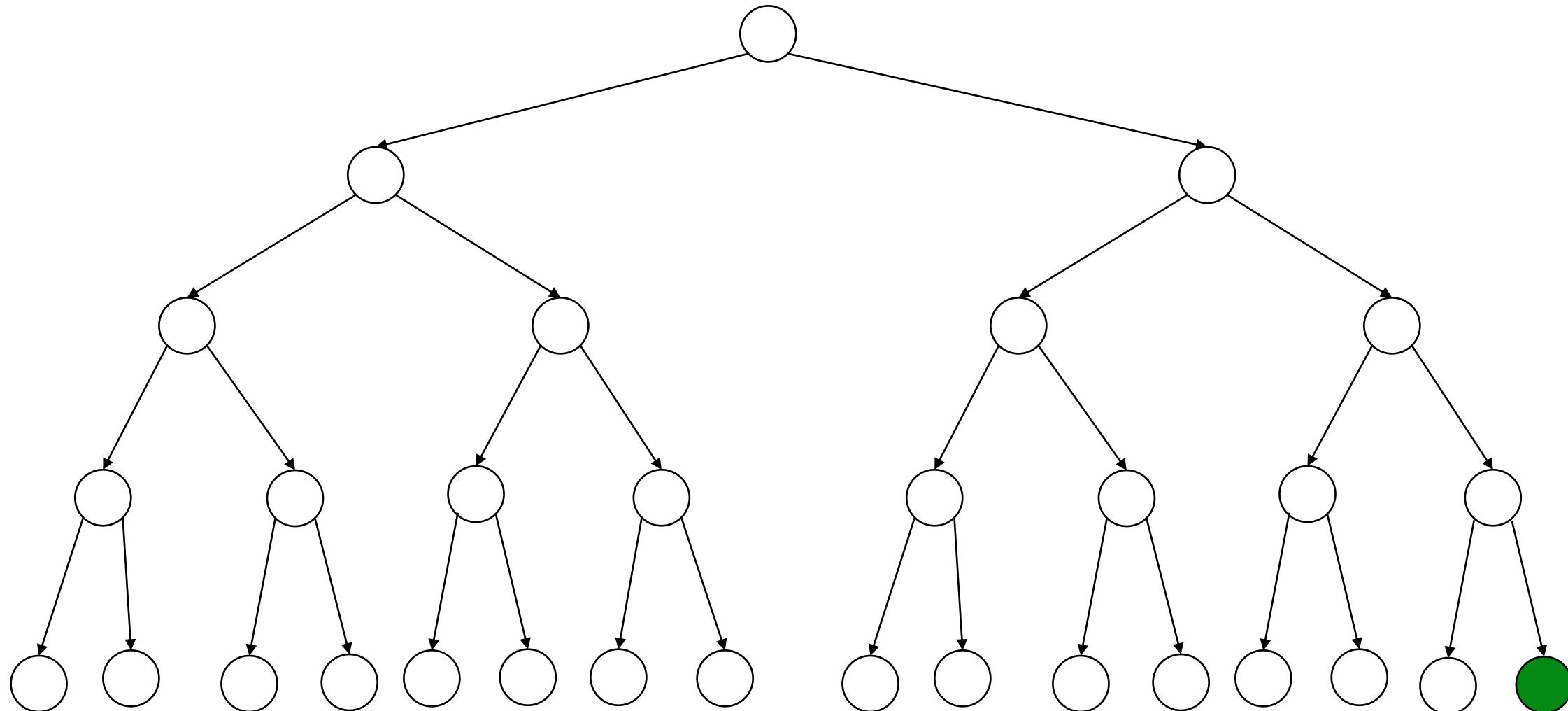
ارزیابی جستجوی اول-بهترین حریصانه

- 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 گرهایی که به هدف نزدیک‌تر است فاصله‌اش از ریشه بسیار بیشتر از گرهای دیگر باشد و در نتیجه هزینه کل مسیر از ریشه تا هدف از طریق آن گره بیشتر شود.

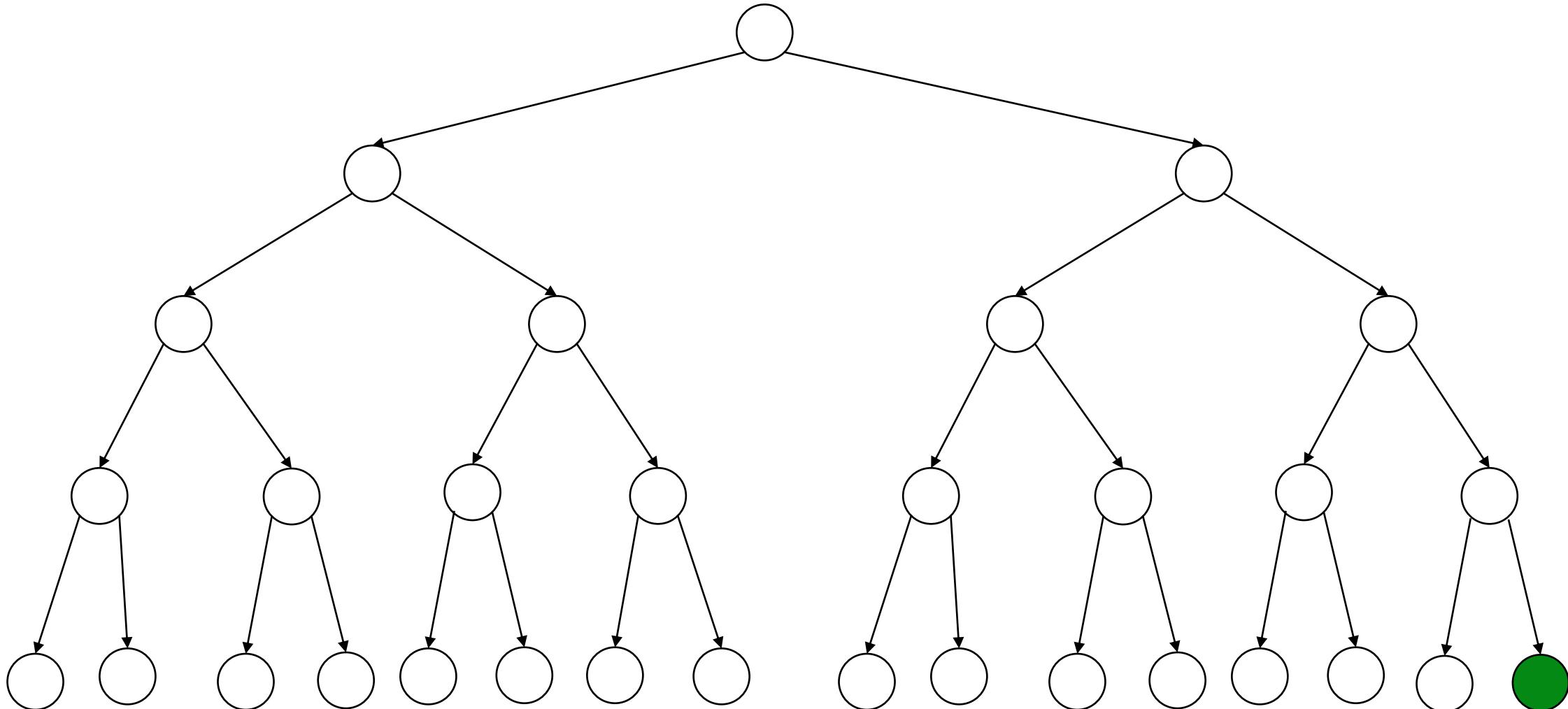


پیچیدگی زمانی جستجوی اول بهترین حریصانه



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

پیچیدگی حافظه‌ای جستجوی اول بهترین حریصانه

 $O(b^m)$

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 کنده و لی اگر به بن‌بست برسد به عقب برمی‌گردد.

- با اتخاذ یکتابع هیوریستیک خوب، پیچیدگی به مقدار قابل توجهی می‌تواند کاهش یابد.

- مقدار کاهش به مسئله و کیفیت هیوریستیک بستگی دارد.

- پیچیدگی فضایی؟

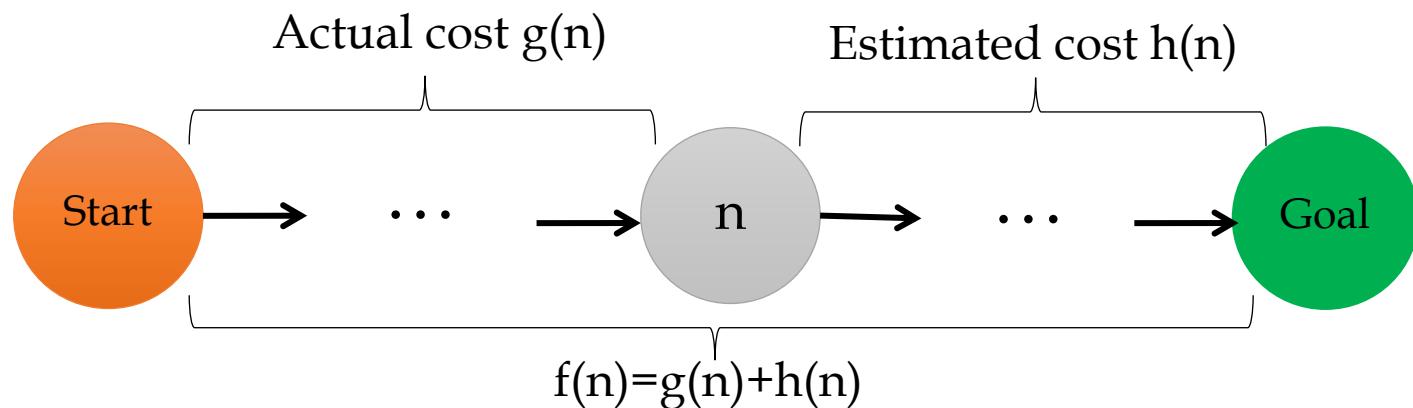
- همه‌ی گره‌ها را در حافظه نگه می‌دارد: $O(b^m)$

جستجوی A*

- شناخته شده ترین شکل جستجوی اول-بهترین است.

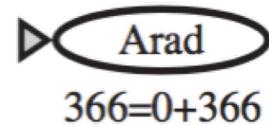
$$f(n) = g(n) + h(n)$$

- تابع ارزیاب: $f(n) = g(n) + h(n)$
- $g(n)$ هزینه واقعی مسیر از شروع تا گره n
- $h(n)$ هزینه تخمینی ارزان ترین مسیر از نود n تا هدف
- $f(n)$ هزینه تخمینی کل مسیر از ریشه درخت تا گره هدف از طریق گره n



جستجوی A^* - مثال

(a) The initial state

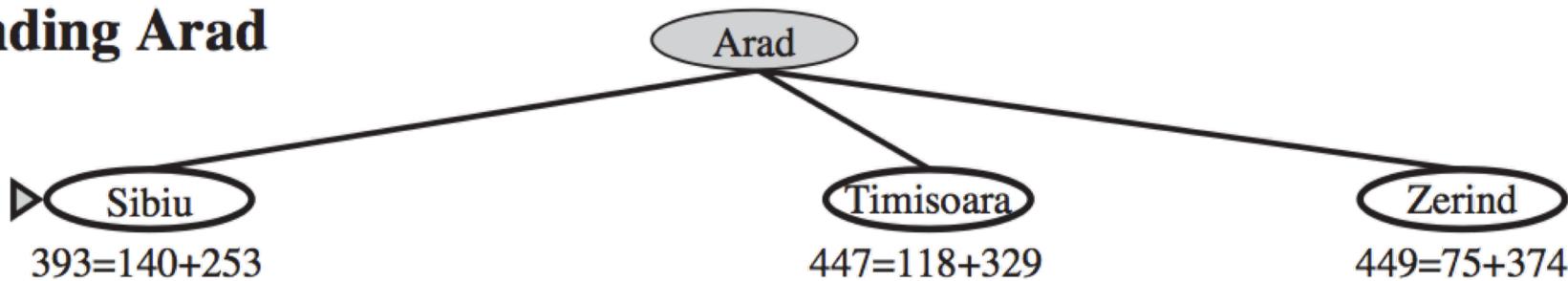


• یافتن Arad با شروع از Bucharest

$$f(Arad) = c(Arad, Arad) + h(Arad) = 0 + 366 = 366 \quad •$$

جستجوی A* - مثال

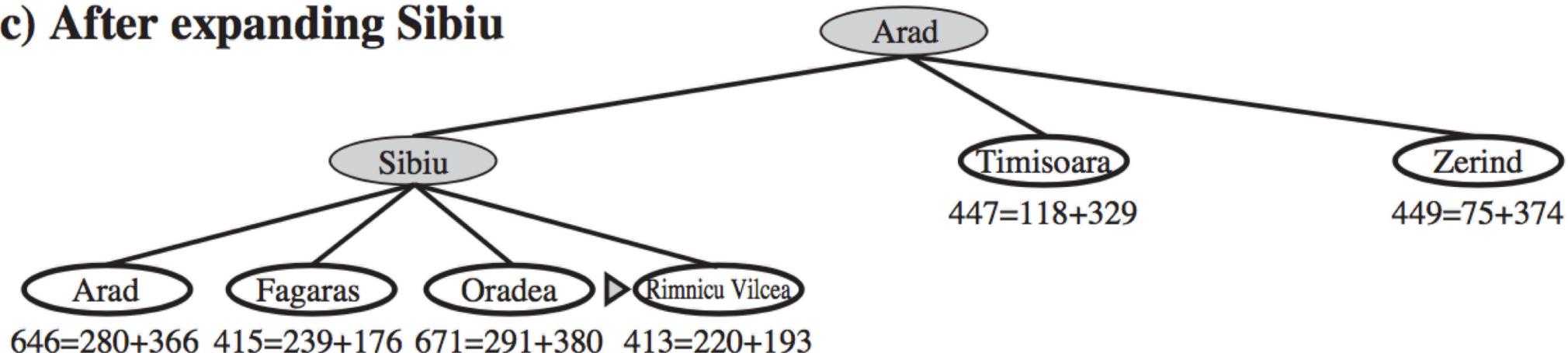
(b) After expanding Arad



- 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

(c) After expanding Sibiu



• بسط Sibiu و تعیین $f(n)$ برای هر نود

$$f(\text{Arad}) = c(\text{Arad}, \text{Sibiu}) + c(\text{Sibiu}, \text{Arad}) + h(\text{Arad}) = 280 + 366 = 646 \quad \bullet$$

$$f(\text{Fagaras}) = c(\text{Arad}, \text{Sibiu}) + c(\text{Sibiu}, \text{Fagaras}) + h(\text{Fagaras}) = 239 + 179 = 415 \quad \bullet$$

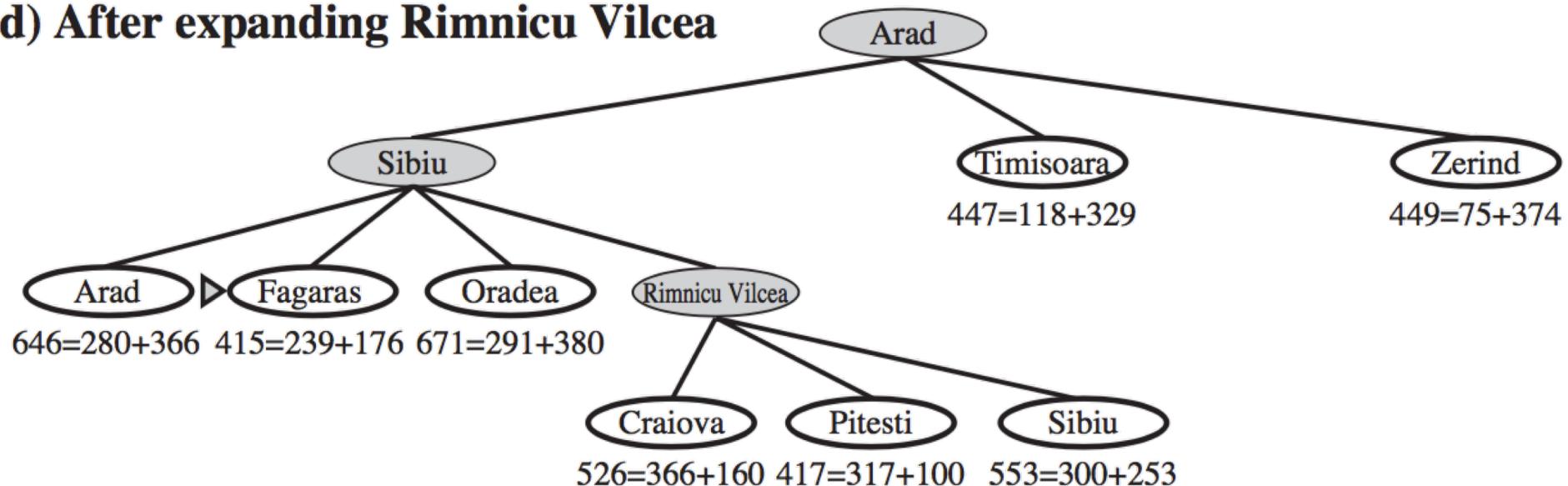
$$f(\text{Oradea}) = c(\text{Arad}, \text{Sibiu}) + c(\text{Sibiu}, \text{Oradea}) + h(\text{Oradea}) = 291 + 380 = 671 \quad \bullet$$

$$f(\text{Rimnicu Vilcea}) = c(\text{Arad}, \text{Sibiu}) + c(\text{Sibiu}, \text{Rimnicu Vilcea}) + h(\text{Rimnicu Vilcea}) = 220 + 192 = 413 \quad \bullet$$

• بهترین انتخاب Rimnicu Vilcea است.

جستجوی A^* - مثال

(d) After expanding Rimnicu Vilcea



بسط $f(n)$ و تعیین برای هر نود Rimnicu Vilcea

$$f(\text{Craiova}) = c(\text{Arad}, \text{Sibiu}) + c(\text{Sibiu}, \text{Rimnicu Vilcea}) + c(\text{Rimnicu Vilcea}, \text{Craiova}) + h(\text{Craiova}) = 360 + 160 = 526$$

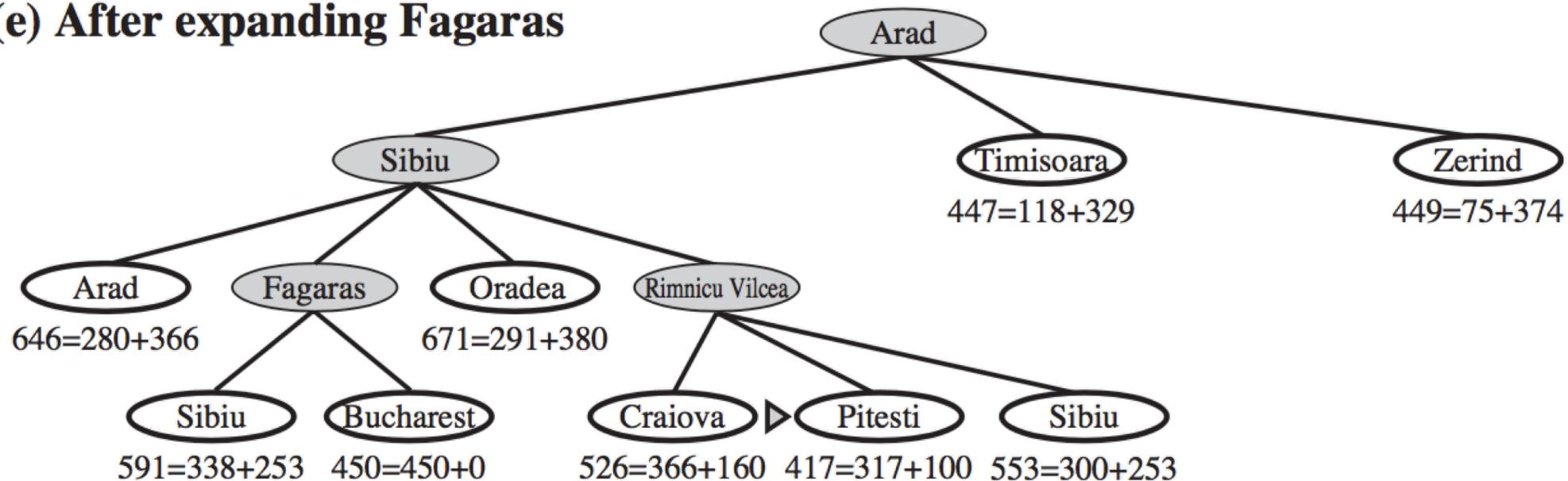
$$f(\text{Pitesti}) = c(\text{Arad}, \text{Sibiu}) + c(\text{Sibiu}, \text{Rimnicu Vilcea}) + c(\text{Rimnicu Vilcea}, \text{Pitesti}) + h(\text{Pitesti}) = 317 + 100 = 417$$

$$f(\text{Sibiu}) = c(\text{Arad}, \text{Sibiu}) + c(\text{Sibiu}, \text{Rimnicu Vilcea}) + c(\text{Rimnicu Vilcea}, \text{Sibiu}) + h(\text{Sibiu}) = 300 + 253 = 553$$

بهترین انتخاب Fagaras است.

جستجوی A* - مثال

(e) After expanding Fagaras



- بسط Fagaras و تعیین $f(n)$ برای هر نود

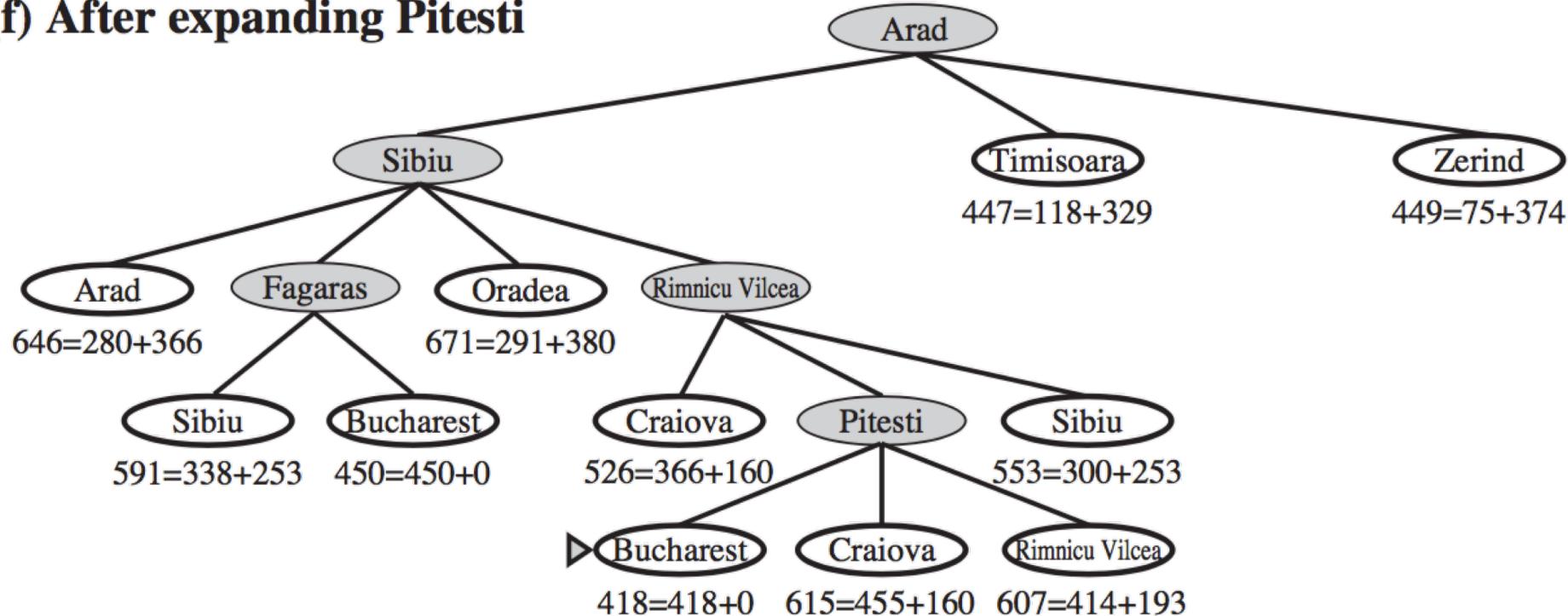
$$f(\text{Sibiu}) = c(\text{Arad}, \text{Sibiu}) + c(\text{Sibiu}, \text{Fagaras}) + c(\text{Fagaras}, \text{Sibiu}) + h(\text{Sibiu}) = 338 + 253 = 591$$

$$f(\text{Bucharest}) = c(\text{Arad}, \text{Sibiu}) + c(\text{Sibiu}, \text{Fagaras}) + h(\text{Bucharest}) = 450 + 0 = 450$$

- بهترین انتخاب Pitesti است!!

جستجوی A* - مثال

(f) After expanding Pitesti



• بسط Pitesti و تعیین $f(n)$ برای هر نود

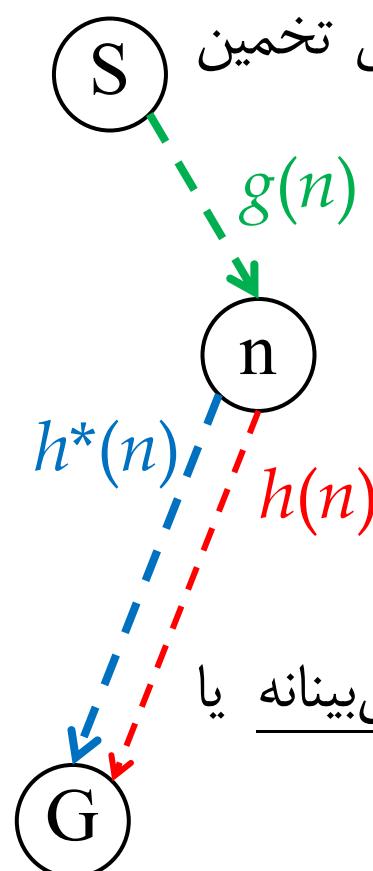
$$f(\text{Bucharest}) = c(\text{Arad}, \text{Sibiu}) + c(\text{Sibiu}, \text{Rimnicu Vilcea}) + c(\text{Rimnicu Vilcea}, \text{Pitesti}) + c(\text{Pitesti}, \text{Bucharest}) + h(\text{Bucharest}) = 418 + 0 = 418$$

• بهترین انتخاب Bucharest است!!

• به مقادیر f در طول مسیر بهینه توجه کنید!!

شرط بهینگی جستجوی درختی A^*

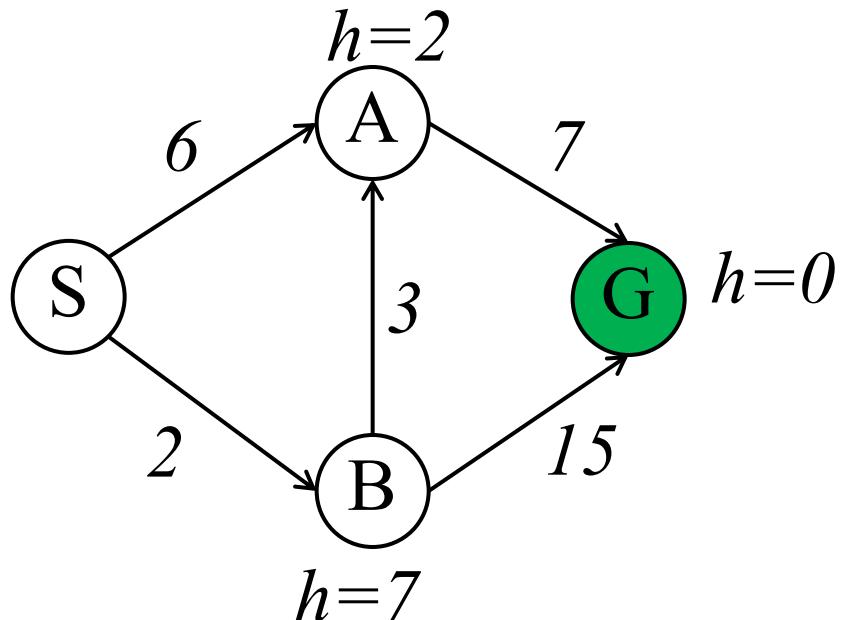
قابل قبول بودن Admissibility



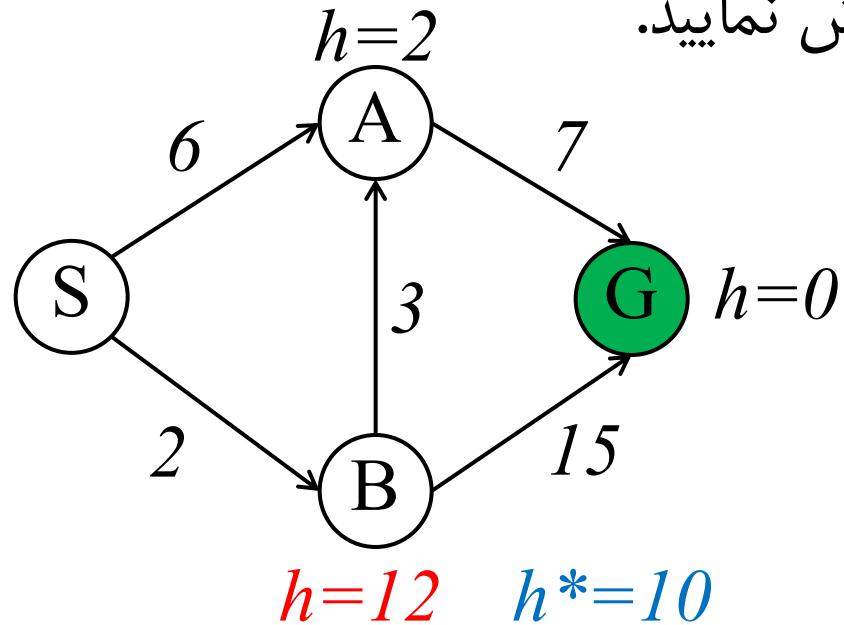
- قابل قبول بودن Admissibility
- هیوریستیک $h(n)$ قابل قبول است اگر هزینه مسیر هر گره تا هدف را بیشتر از مقدار واقعی تخمین نزند.
- یک حد پایین روی هزینه مسیر از n تا هدف است اگر برای هر گره n
$$h(n) \leq h^*(n)$$
 که $h^*(n)$ هزینه واقعی ارزان‌ترین مسیر به وضعیت هدف از گره n است.
- برای مثال: $\geq h_{SLD}(n)$ "فاصله واقعی جاده"
- می‌توان گفت هر تابع هیوریستیک قابل قبول، هزینه هر گره تا هدف را به‌طور خوب‌بینانه 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.

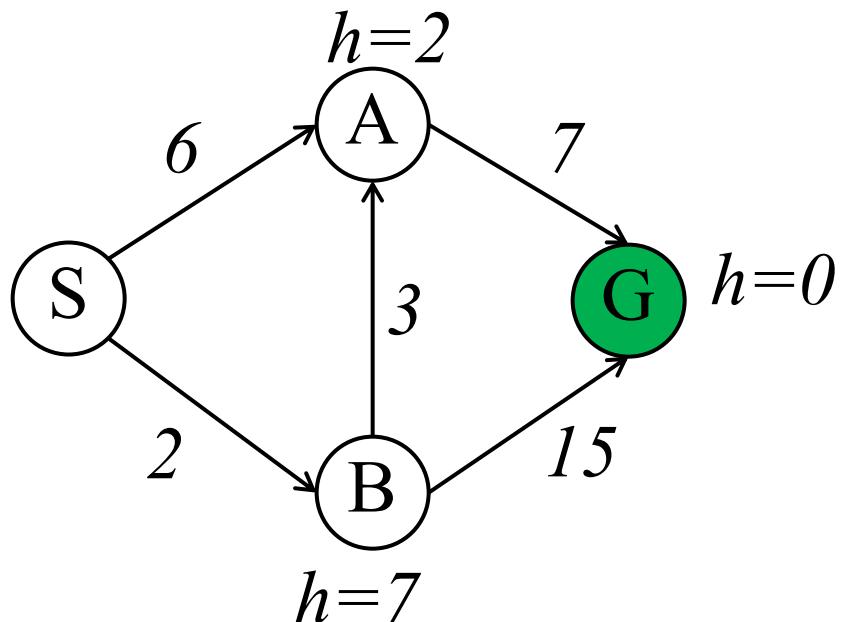


Admissible heuristic
Path: $S-B-A-G$

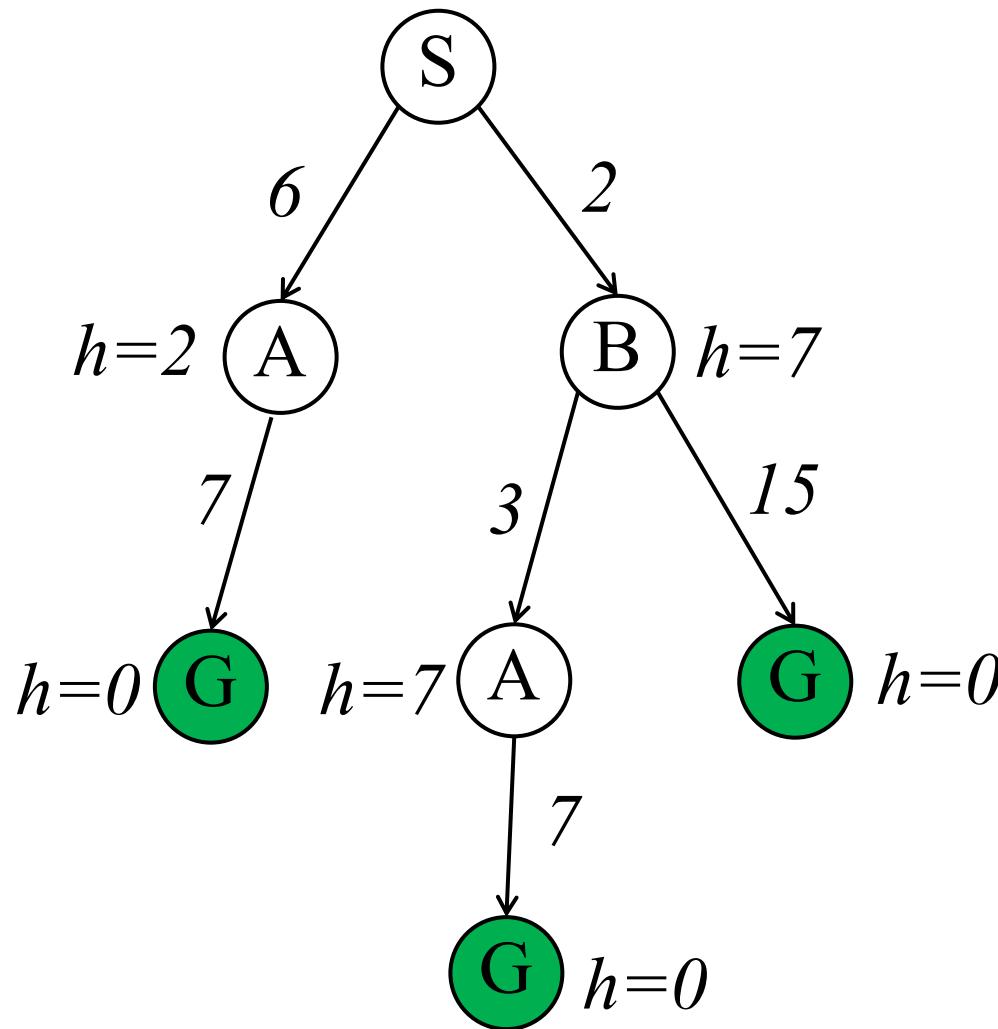


Inadmissible heuristic
Path: $S-A-G$

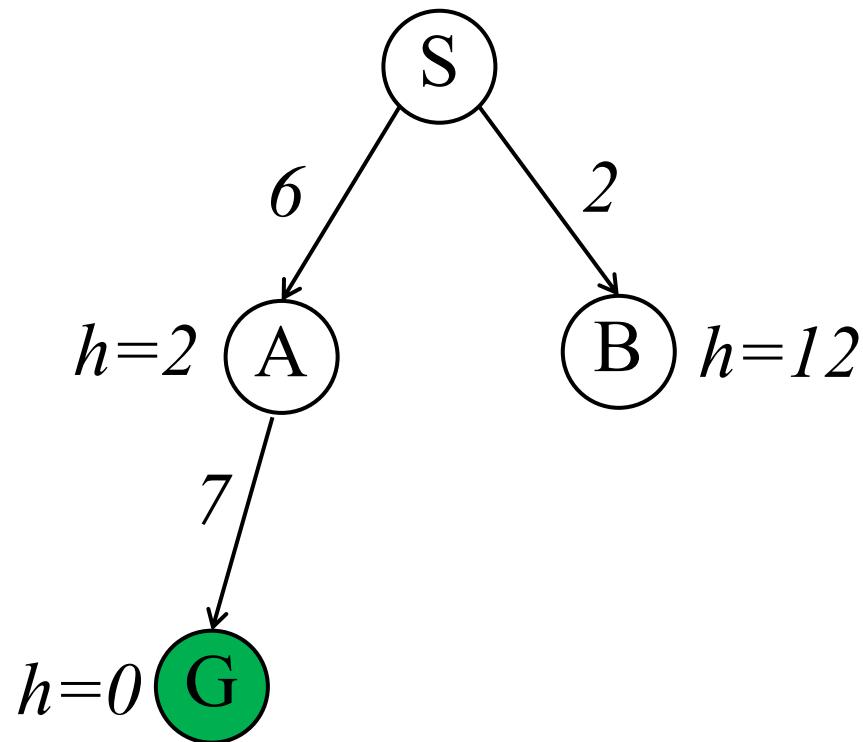
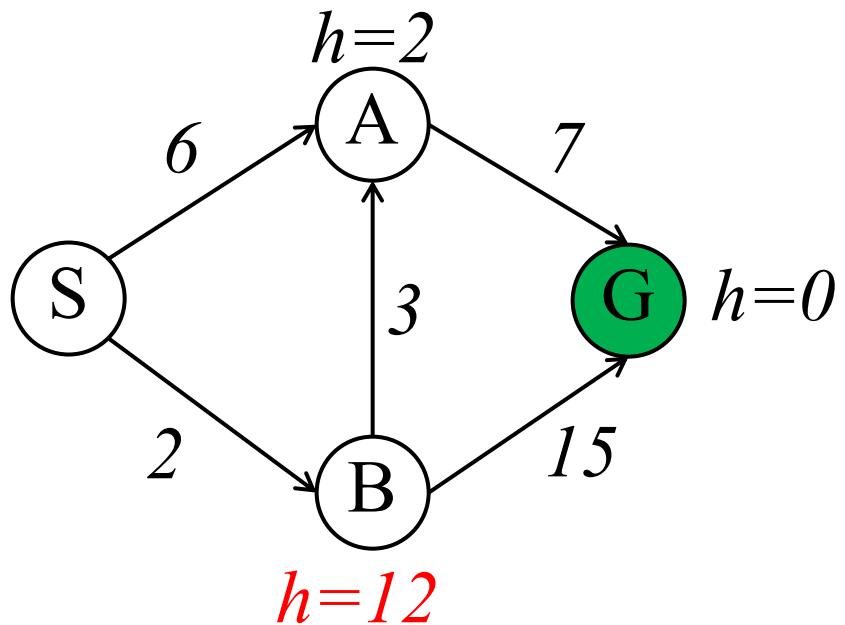
قابل قبول بودن در مقابل غیرقابل قبول بودن



*Admissible heuristic
Path: S-B-A-G*



قابل قبول بودن در مقابل غیرقابل قبول بودن



*Inadmissible heuristic
Path: S-A-G*

اثبات بھینگی جستجوی درختی A^*

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

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

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

$$3) G_2 \text{ is suboptimal} \rightarrow g(G_2) > g(G)$$

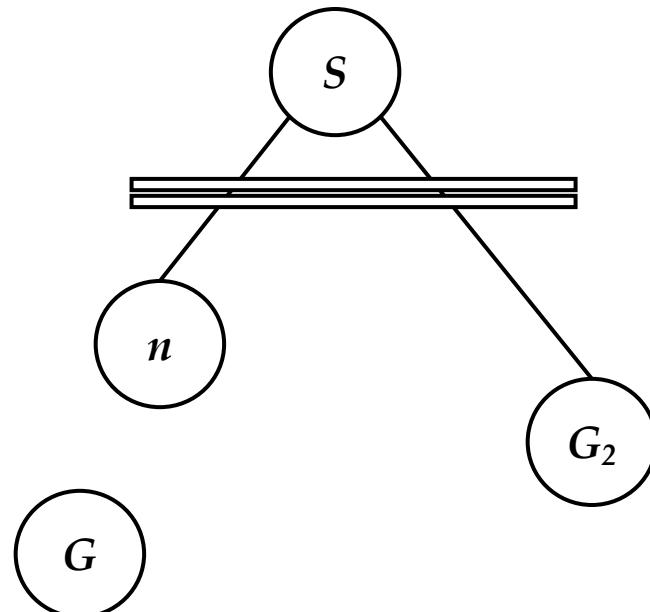
$$\Rightarrow f(G_2) > f(G)$$

$$4) h \text{ is admissible} \rightarrow h(n) \leq h^*(n)$$

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

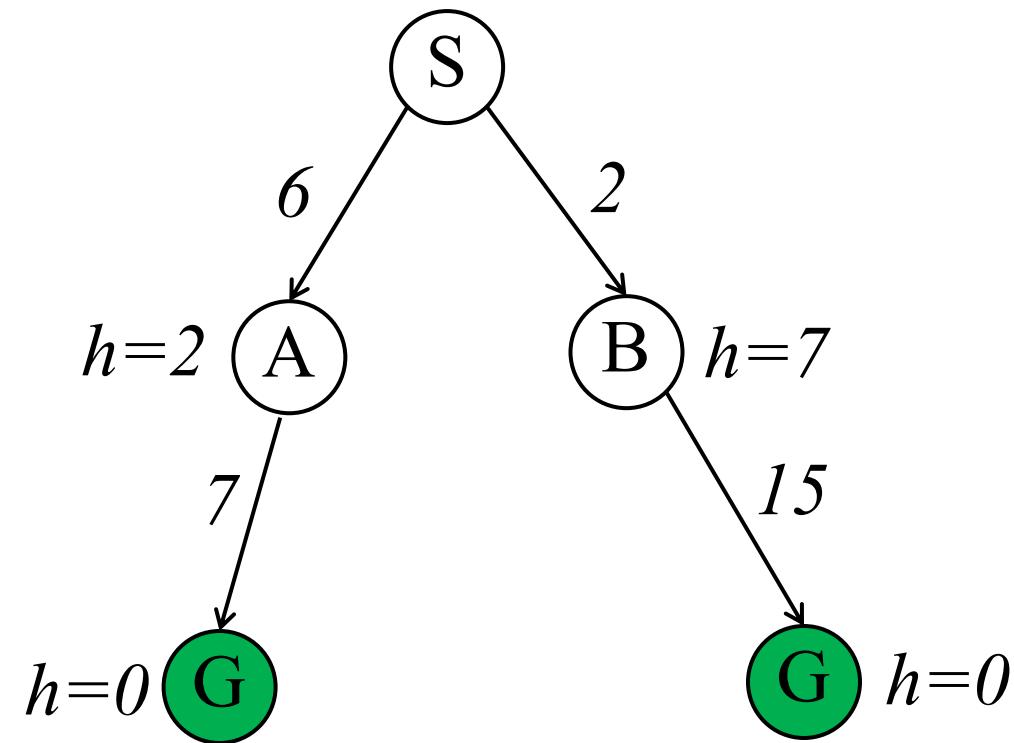
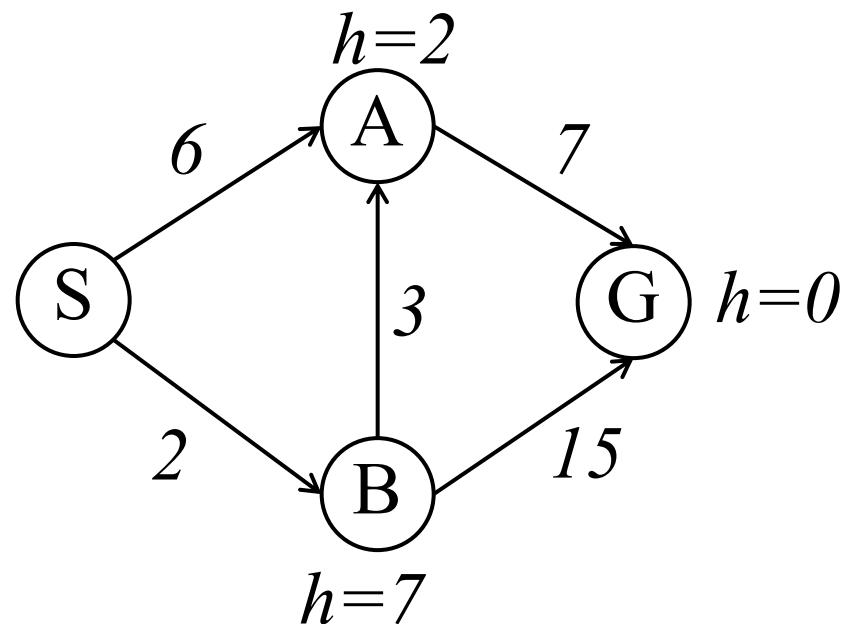
$$\rightarrow f(n) \leq g(G)=f(G) \Rightarrow f(n) < f(G_2)$$

- در این صورت G_2 هرگز برای بسط انتخاب نمی شود.



شرط بھینگی جستجوی گرافی A*

- آیا قابل قبول بودن هیوریستیک، بھینگی جستجوی گرافی A^* را تضمین می کند؟
- باید توجه کرد که گراف جستجو مسیر بھینه به یک حالت تکراری را کنار می گذارد.
- برای مثال در مورد گراف زیر مسیر نیمه بھینه S-A-G توسط جستجوی گرافی به دست خواهد آمد.



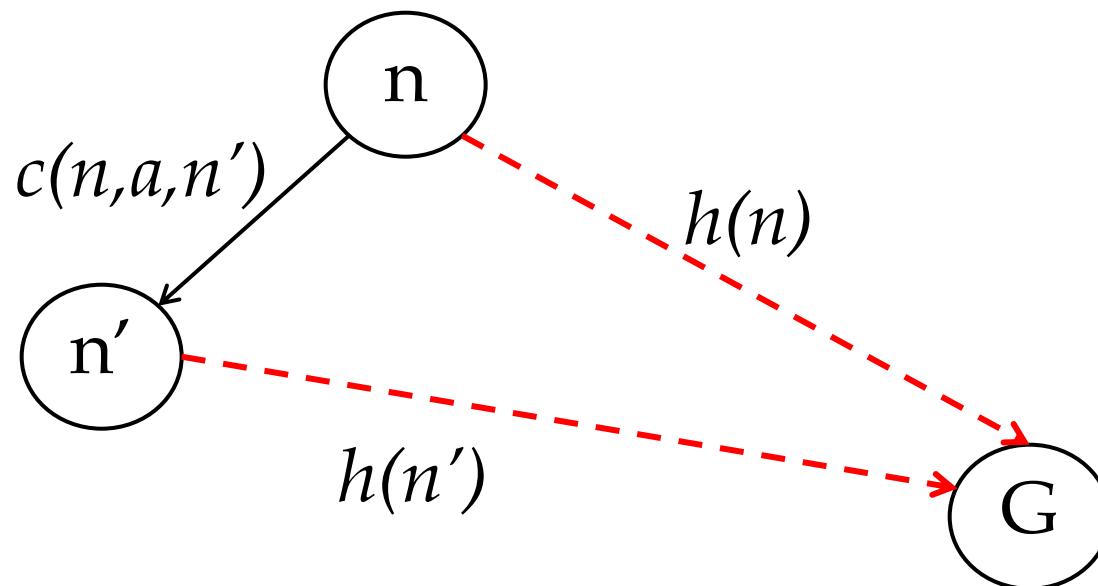
شرط بهینگی جستجوی گرافی A*

- 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:

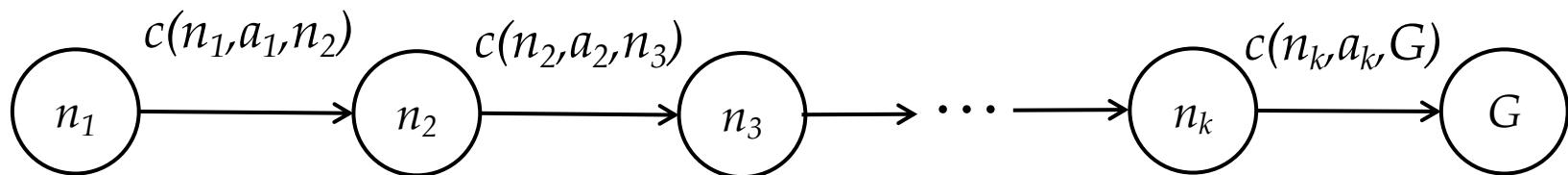
$$h(n) \leq c(n, a, n') + h(n')$$

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 \leftarrow قابل قبول بودن
- همه‌ی توابع هیوریستیک سازگار قابل قبول هستند.



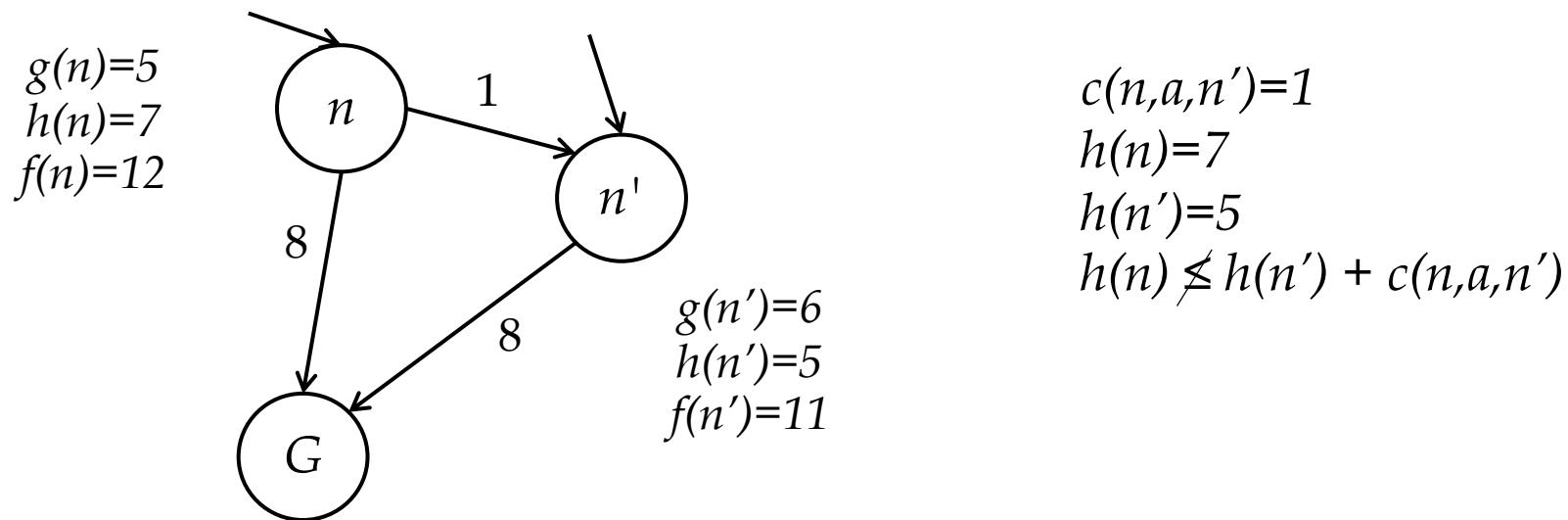
$$\begin{aligned} h(n_1) &\leq c(n_1, a_1, n_2) + h(n_2) \\ &\leq c(n_1, a_1, n_2) + c(n_2, a_2, n_3) + h(n_3) \end{aligned}$$

...

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

ارتباط قابل قبول و سازگار بودن

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



- 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^*

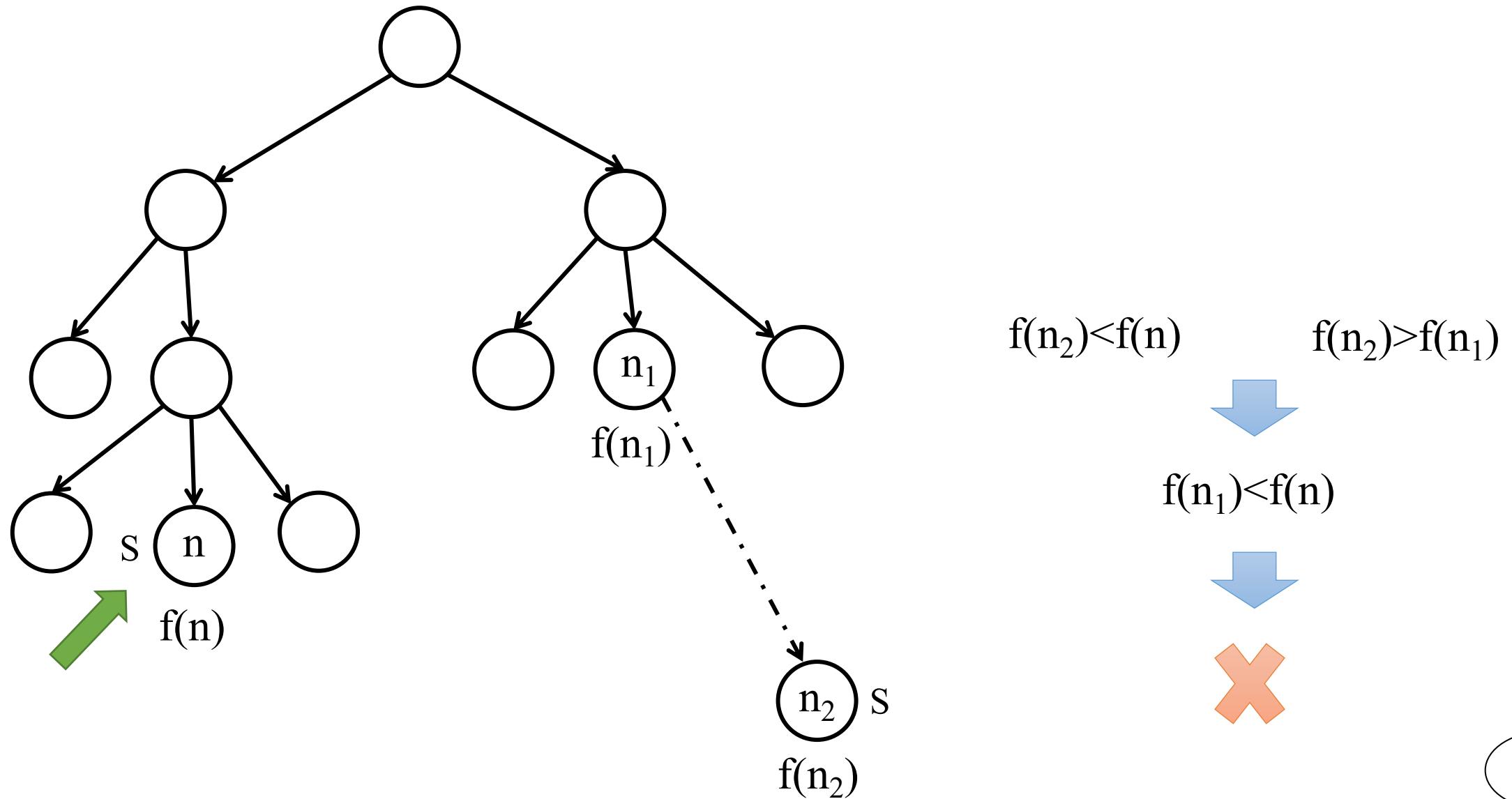
- قضیه: اگر $h(n)$ سازگار باشد، A^* با استفاده از جستجوی گرافی بهینه خواهد بود.
- لم ۱: اگر $h(n)$ سازگار باشد آن‌گاه مقدار $f(n)$ در طول هر مسیری غیرنژولی است.
- اثبات: فرض کنید n' یک پسین از n باشد

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

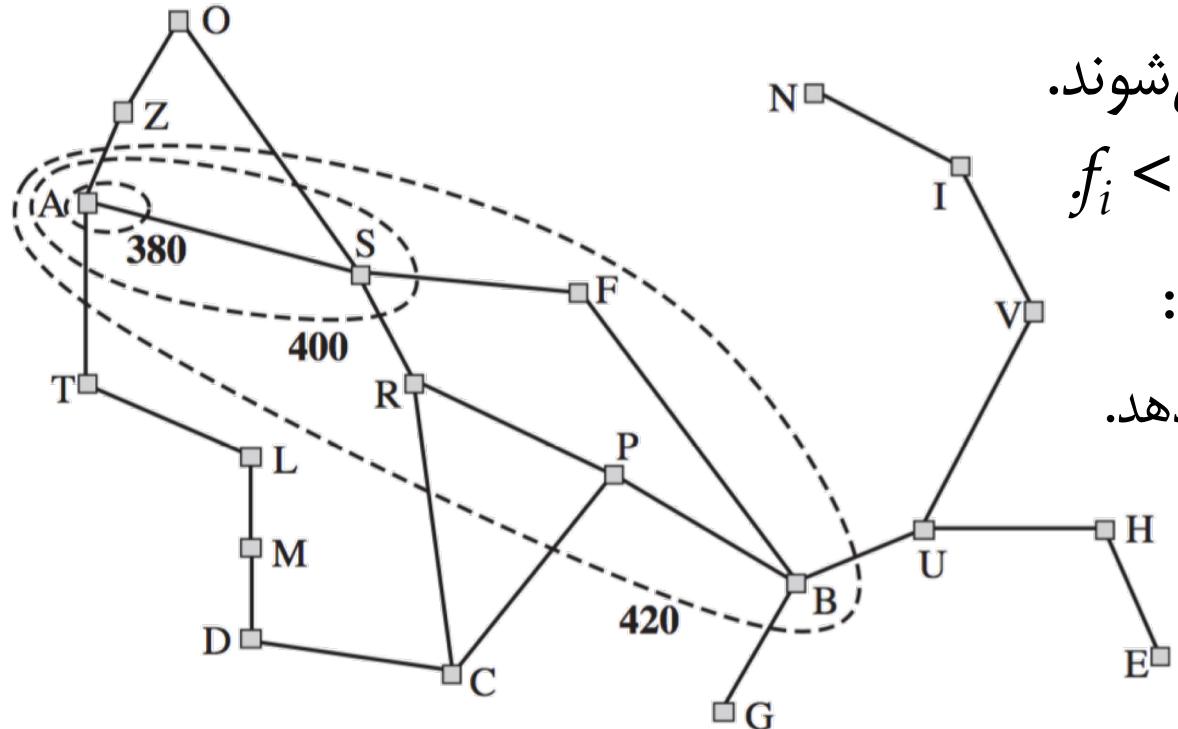
اثبات بهینگی جستجوی گرافی A^*

- لم ۲: اگر A^* گره n را برای بسط انتخاب کند، راه حل 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 به دست نیامده باشد آنگاه باید بتوان از طریق گره دیگری مانند n_1 که در مجموعه frontier فعلی قرار دارد با یک مسیر بهینه به حالت موجود در n رسید. علاوه بر این براساس لم ۱، $f(n_1) \leq f(n)$ و بنابراین n_1 باید زودتر انتخاب می‌شد.
- اولین گرهی هدفی که برای بسط انتخاب می‌شود باید یک راه حل 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 به ترتیب غیرنزولی $f(n)$ می‌باشد.

A* جستجوی گرافی



کانتورهای Contours – A*



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

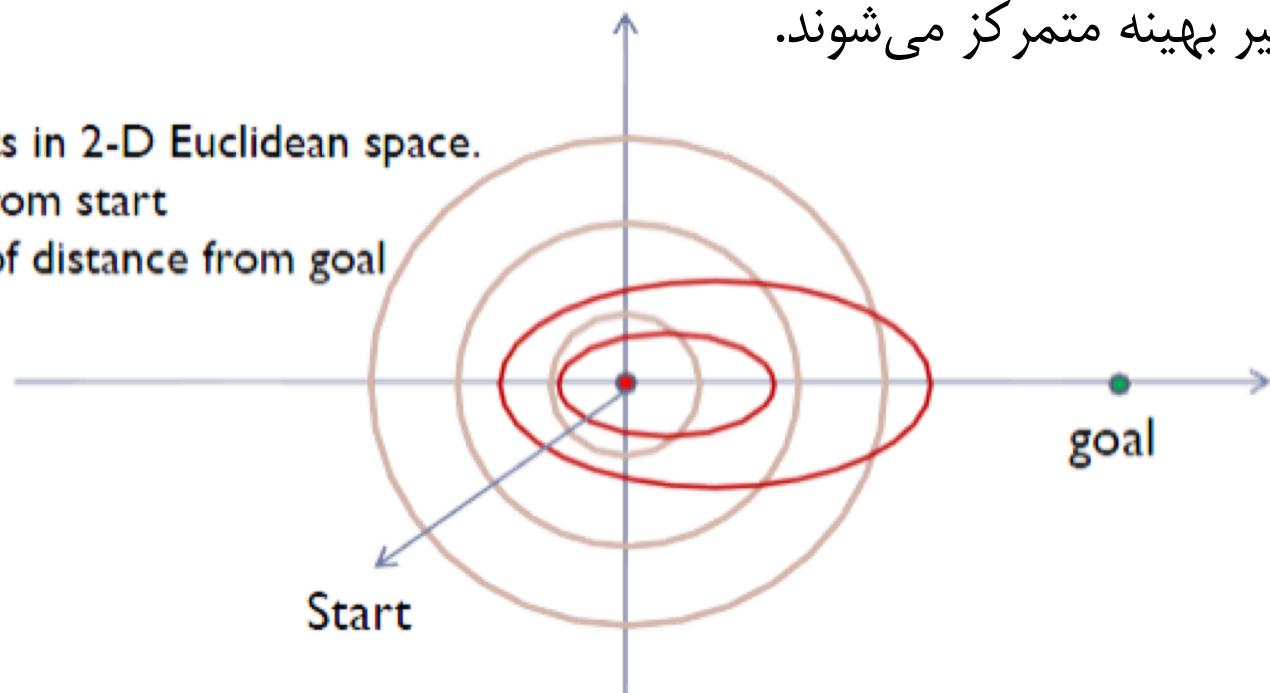
کانتورهای A^* در مقابل UCS

- 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 به صورت باریک‌تری پیرامون مسیر بهینه متمرکز می‌شوند.

States are points in 2-D Euclidean space.

$g(n)$ =distance from start

$h(n)$ =estimate of distance from goal



ارزیابی جستجوی A^*

- کامل؟
- همان طور که نوارهای با f در حال افزایش را اضافه می‌کنیم، باید بالاخره به نواری بررسیم که در آن جا f مساوی هزینه‌ی مسیر تا یک حالت هدف است.
- بنابراین اگر تعداد گره‌ها با $f(G) = C^*$ متناهی باشد کامل خواهد بود.
 - فاکتور انشعاب متناهی بوده و هزینه‌ی یال‌ها از ۴ بیشتر باشد.
- پیچیدگی زمانی؟
- برای اغلب مسائل، تعداد گره‌ها در داخل کانتور هدف (یعنی گره‌هایی با $f(n) \leq C^*(n)$) برحسب طول راه حل نمایی است.
- شرط نمایی نبودن تعداد گره‌ها: خطای تابع هیوریستیک، سریعتر از لگاریتم هزینه واقعی مسیر رشد نکند.

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

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، قبلاً 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 گرهای 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 به عنوان مقدار تابع اکتشافی (هیوریستیک) استفاده شود، سه گره اولی که در الگوریتم A^* گسترش می‌یابند کدام‌اند؟ اگر شرایطی پیش آمد که دو خانه برای گسترش دقیقاً وضعیت یکسانی (از نظر A^*) داشته باشند، خانه با شماره کوچک‌تر انتخاب می‌شود.

۳	۶	۹	۱۲
۲	۵	۸	۱۱
۱	۴	۷	۱۰

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

مقایسه A* و UCS

مقایسه UCS و A* در مسئله مسیریابی ربات



$g(n) < C^*$

$g(n)+h(n) < C^*$

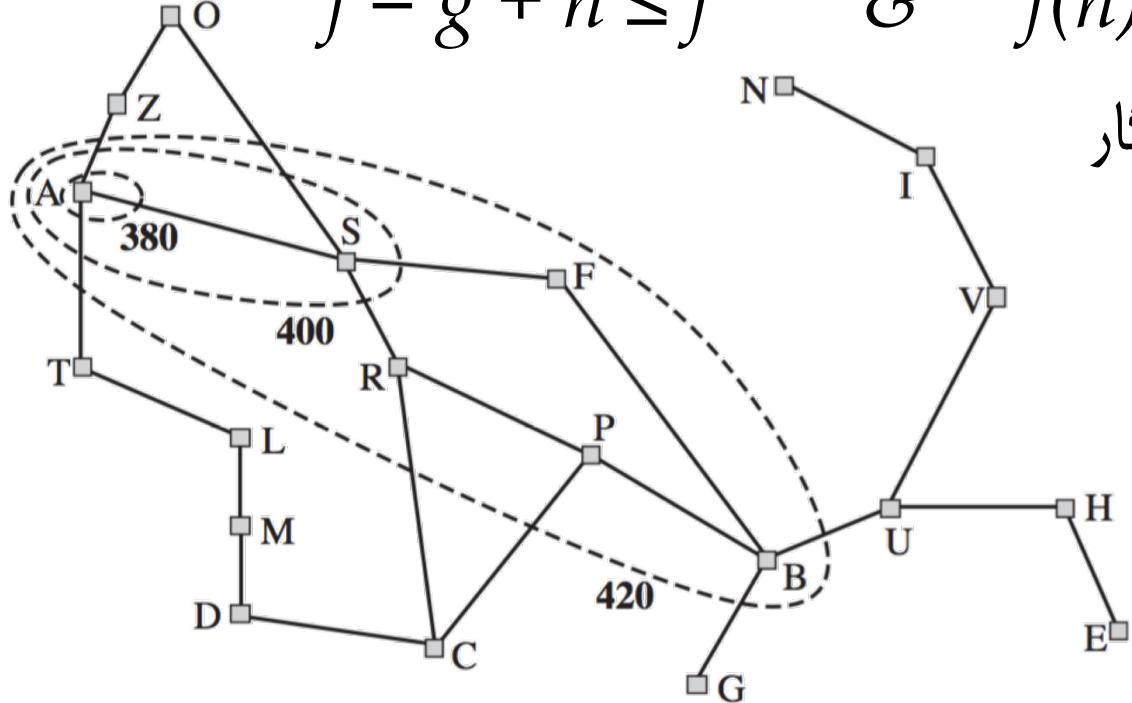
$g(n)+5h(n) < C^*$

جستجوهای هیوریستیکی با حافظه محدود

IDA*, RBFS, SMA*

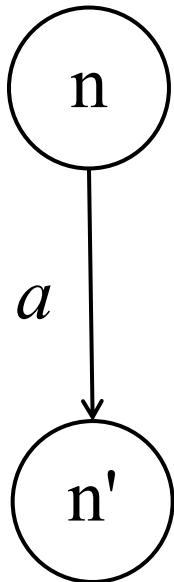
- الگوریتم A^* نیاز به نگهداری مجموعه‌ی frontier (و مجموعه‌ی explored) دارد. این موضوع باعث نیاز به حافظه‌ی خیلی زیادی می‌شود.
- ایده استفاده از f یکنوا است یعنی:

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



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

یکنوا کردن تابع ارزیاب



$$f(n') = \max(g(n') + h(n'), f(n))$$

$$\text{if } g(n') + h(n') \geq f(n) \rightarrow f(n') = g(n') + h(n')$$

$$\begin{aligned} \text{if } g(n') + h(n') < f(n) \rightarrow f(n') &= f(n) = g(n) + h(n) \\ &= g(n') + h(n) - c(n, a, n') \end{aligned}$$

$$h(n') = h(n) - c(n, a, n')$$

جستجوی عمیق‌شوندهٔ تکراری A^*

- IDA^* ، همانند الگوریتم ILS، از ایدهٔ عمقی تکراری استفاده می‌کند.
- مکرراً درخت جستجو را به صورت عمقی می‌سازد.
- در هر تکرار، هر شاخه را تا جایی بسط می‌دهد که هزینهٔ $f(n)$ 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.

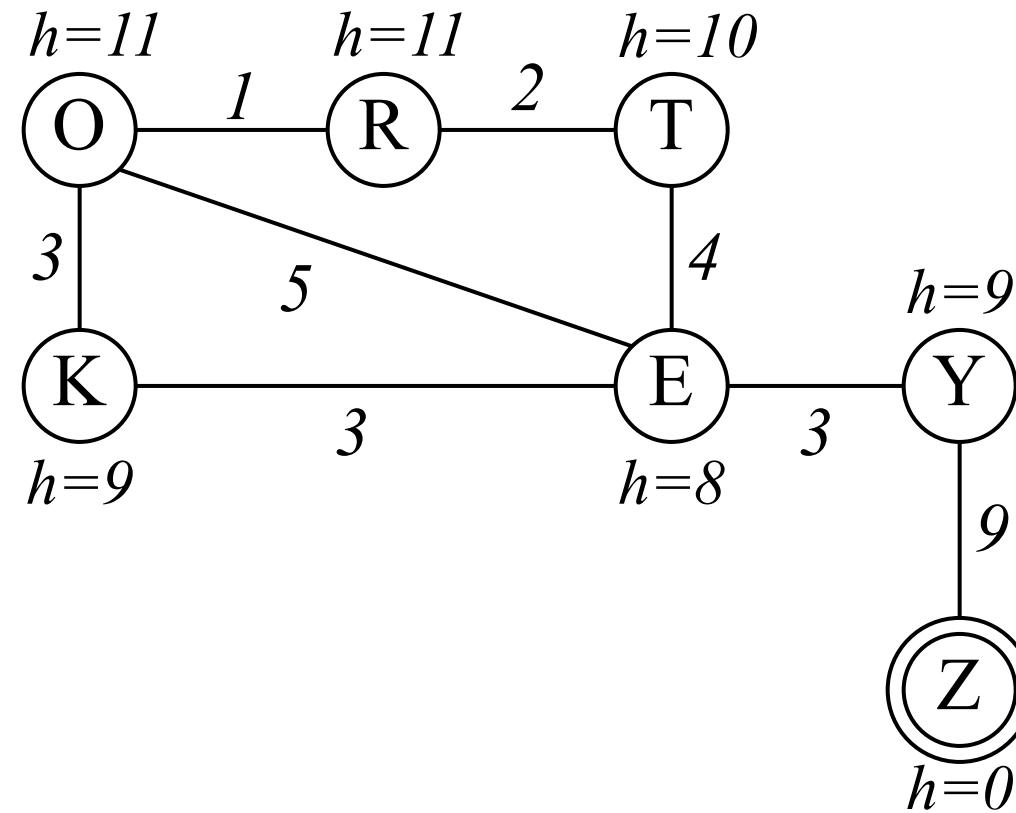
الگوریتم عمیق‌شوندهٔ تکراری A*

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

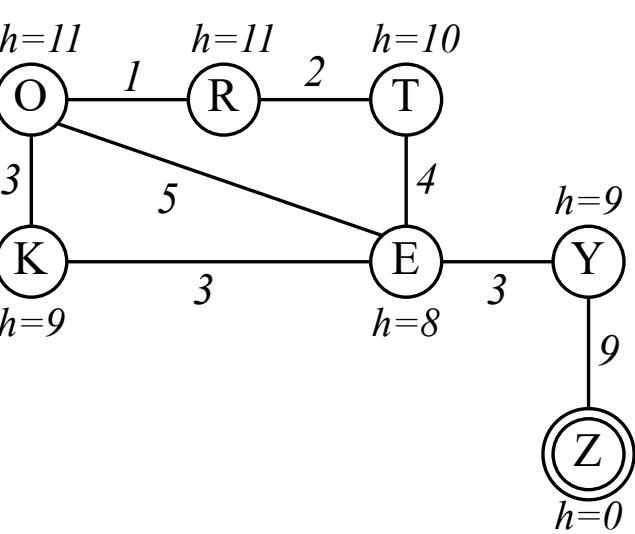
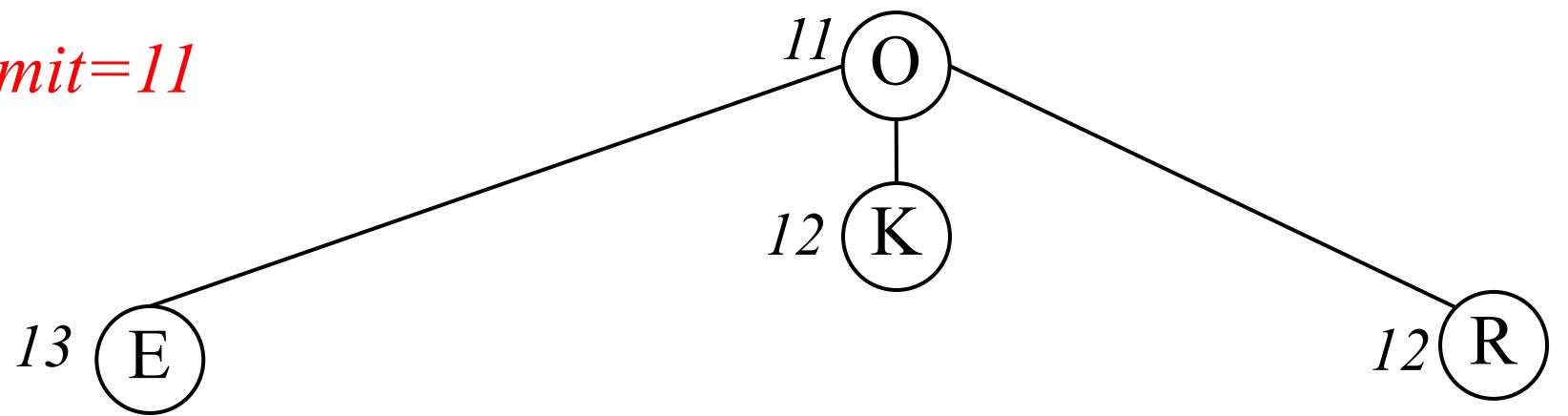
    root ← MAKE-NODE(INITIAL-STATE[problem])
    f-limit ← f- COST(root)
    loop do
        solution, f-limit ← DFS-CONTOUR(root, f-limit)
        if solution is non-null then return solution
        if f-limit = ∞ then return failure; end

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 ∞

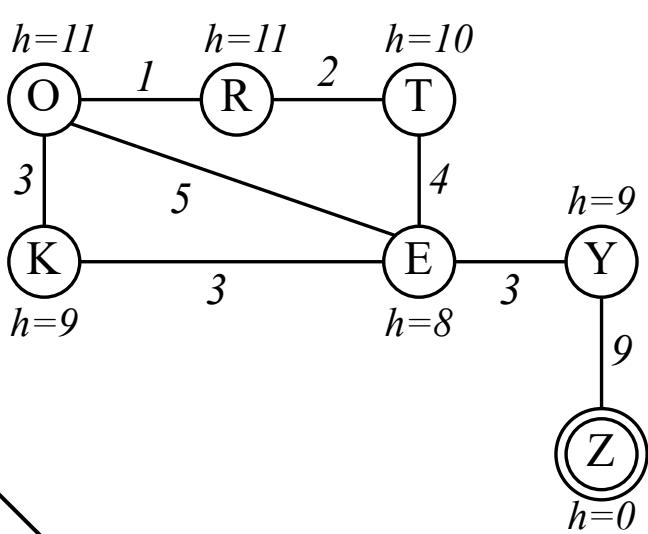
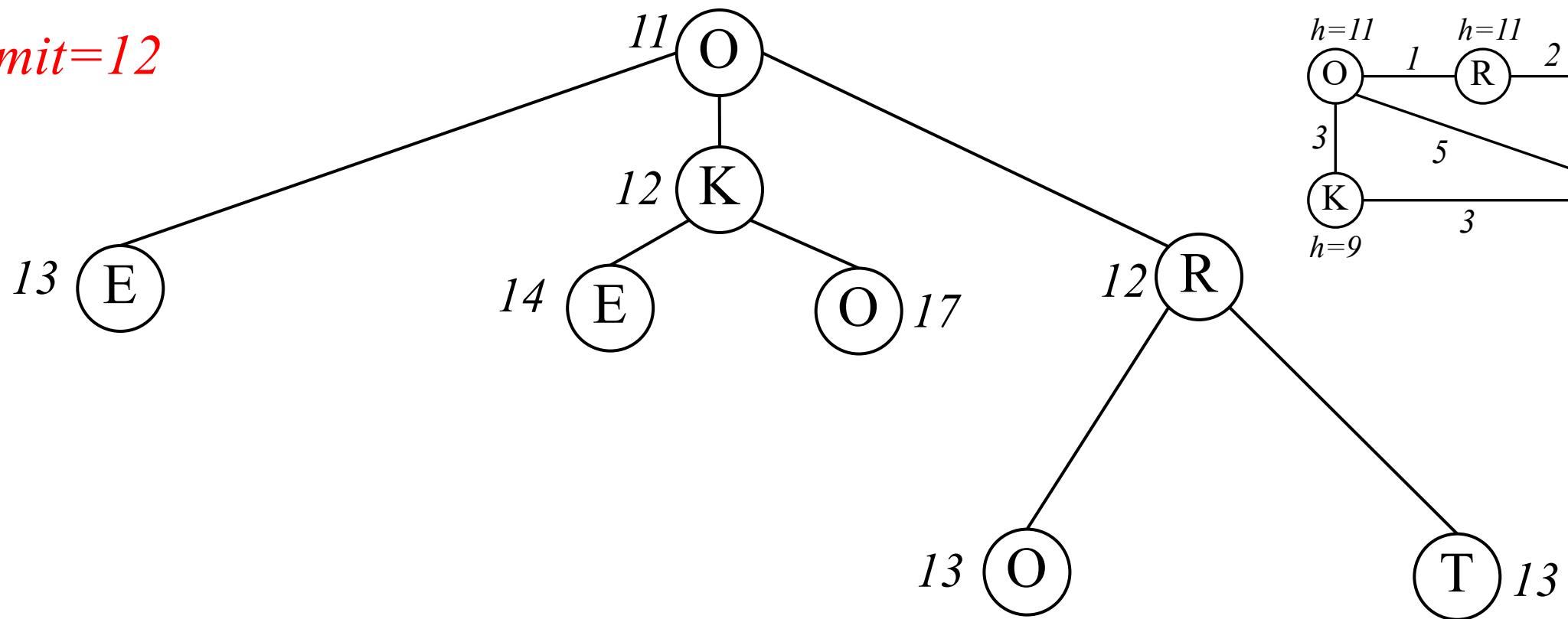
    if f- COST[node] > f-limit then return null, f- COST[node]
    if GOAL-TEST[problem](STATE[node]) then return node, f-limit
    for each node s in SUCCESSORS(node) do
        solution, new-f ← DFS-CONTOUR(s, f-limit)
        if solution is non-null then return solution, f-limit
        next-f ← MIN(next-f,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



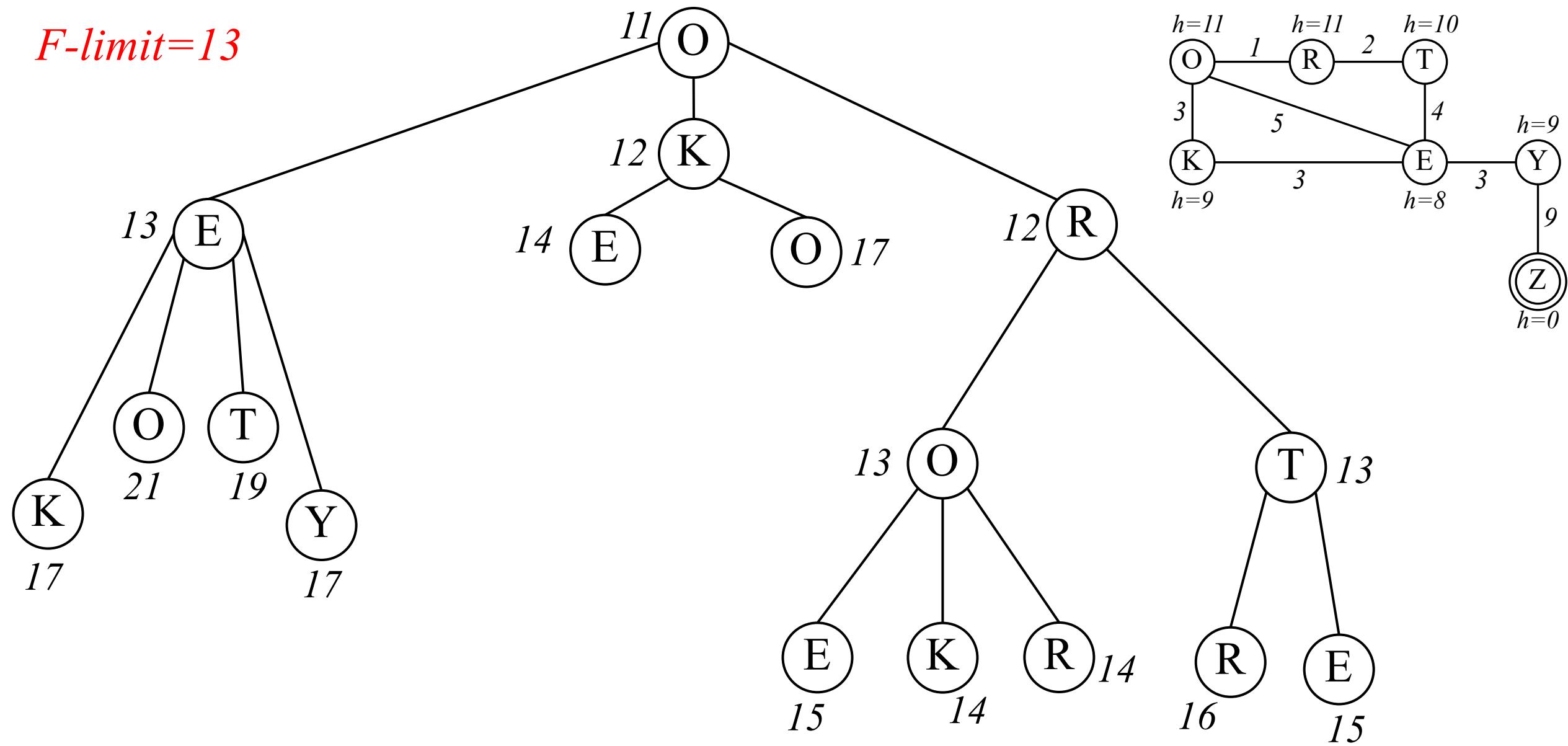
F-limit=11



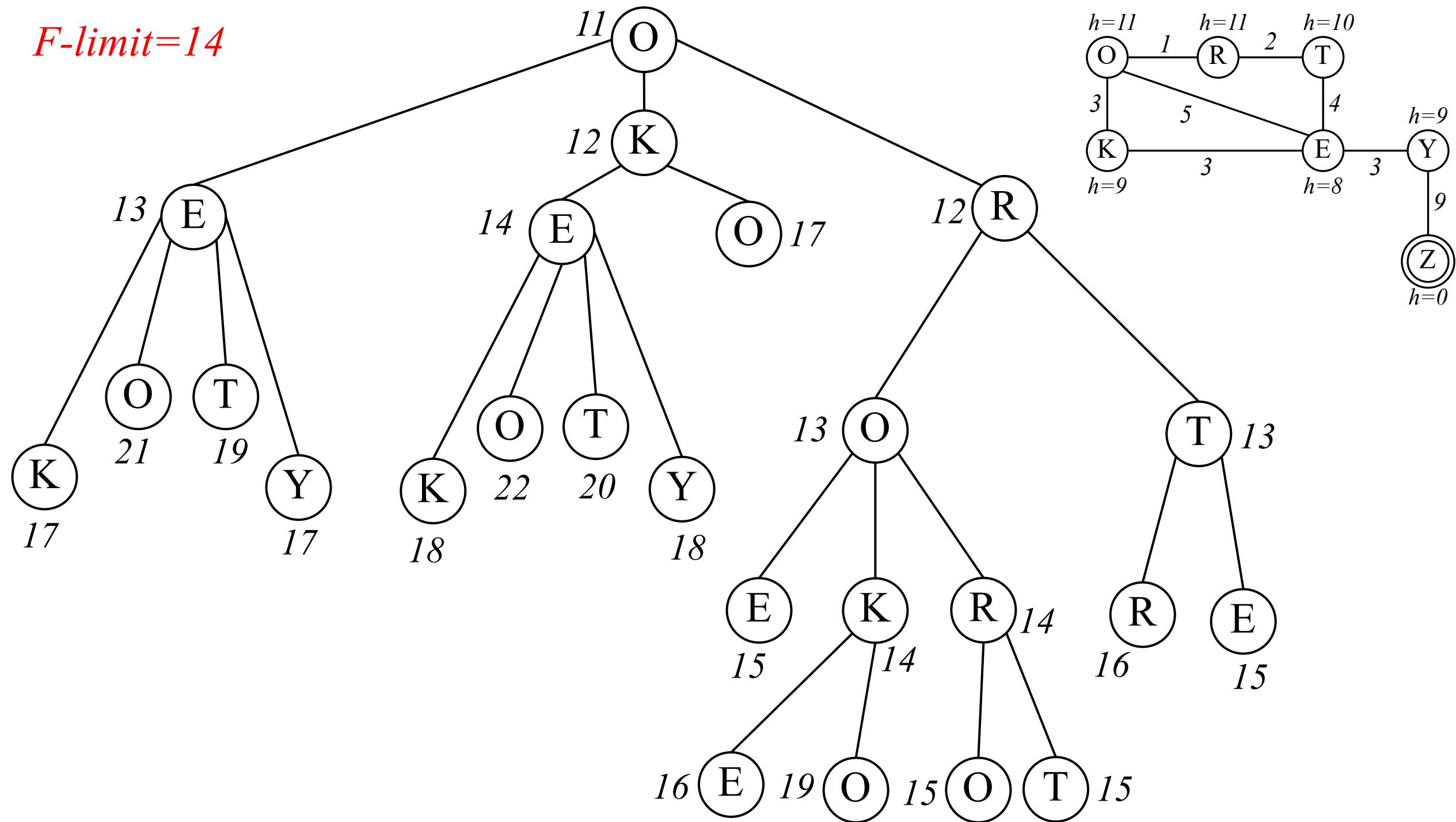
F-limit=12



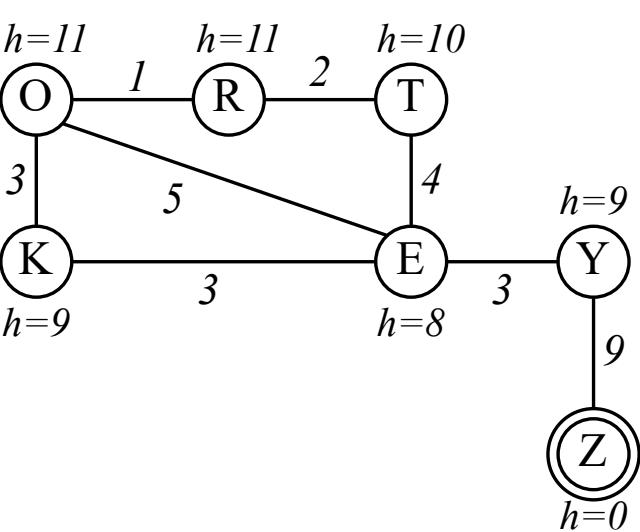
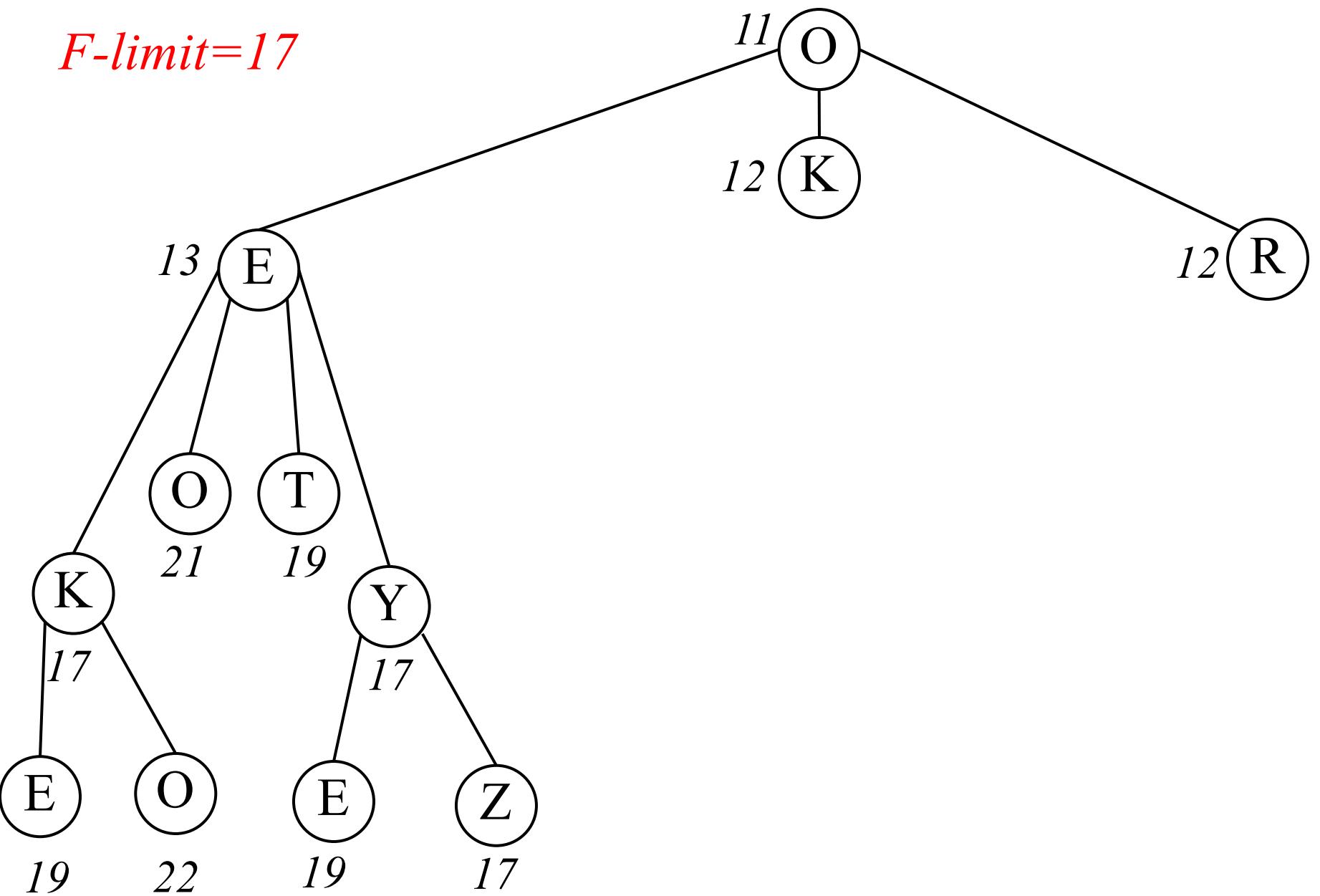
F-limit=13



F-limit=14



F-limit=17



ارزیابی IDA*

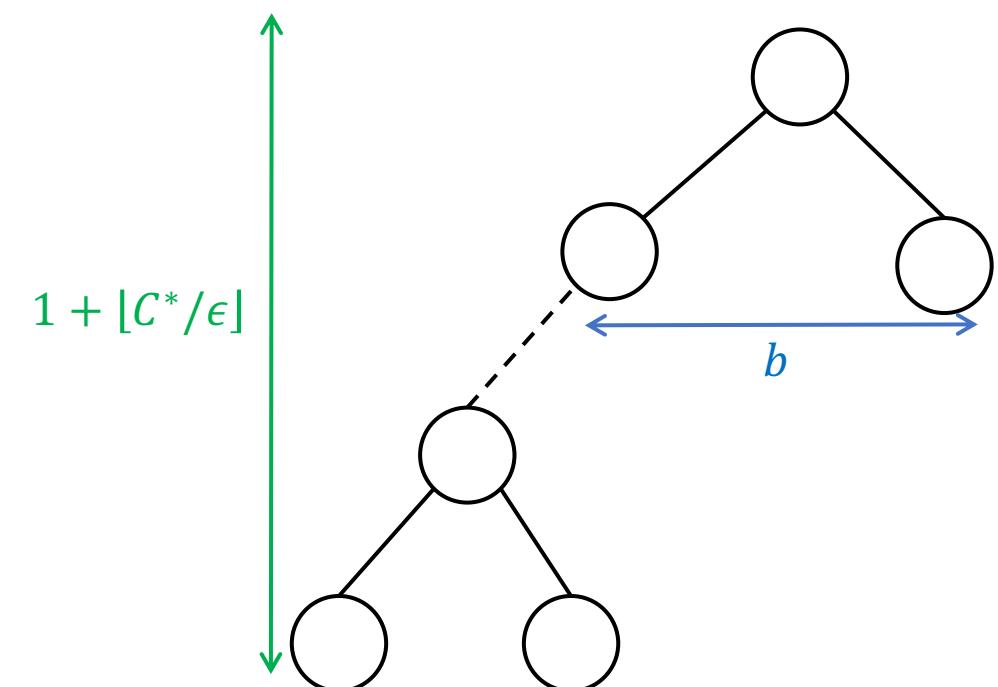
- کامل و بهینه؟
- بله، اگر تابع هیوریستیک قابل قبول باشد، بهینه است.
- می‌توان از طریق استقرا روی دفعات تکرار الگوریتم، بهینگی آن را اثبات نمود.
- پیچیدگی زمانی؟
- همانند A^* در بدترین حالت نمایی است.
- نیازی به یک صفت اولویت برای نگهداری گره‌های مرزی ندارد. در نتیجه از سربار مرتب‌سازی چنین صفتی بی‌نیاز 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* N بار تکرار شود یعنی $O(N) = O(N^2)$ که زمان بسیار زیادی است.

- 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.

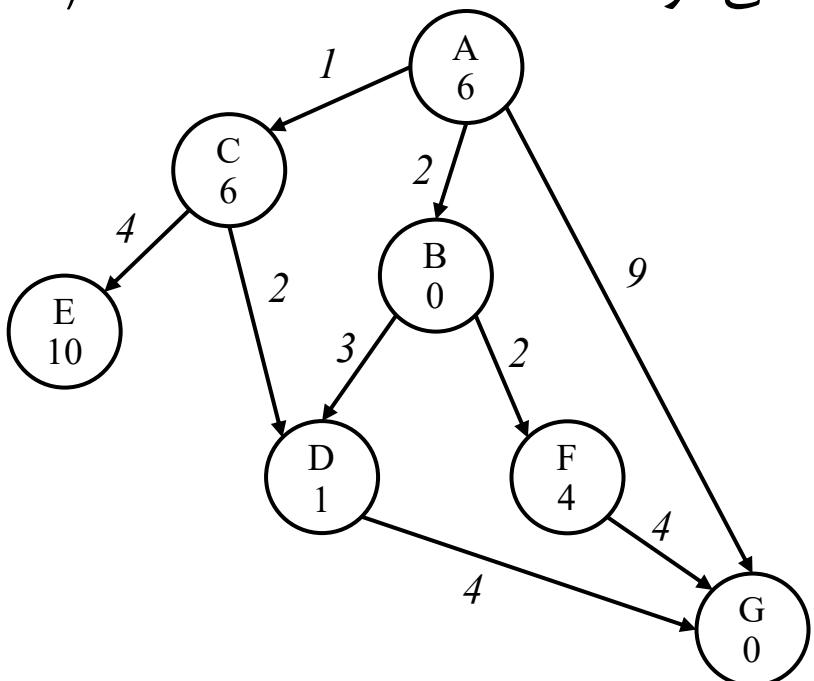
- در بدترین حالت، از مرتبه $O(b \times (1 + \lfloor C^*/\epsilon \rfloor))$ است.

- C^* هزینه راه حل 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 در نظر 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 به ترتیب حروف الفبا تولید می‌شود و در شرایط مساوی به گره‌ای که زودتر تولید شده، اولویت داده می‌شود.



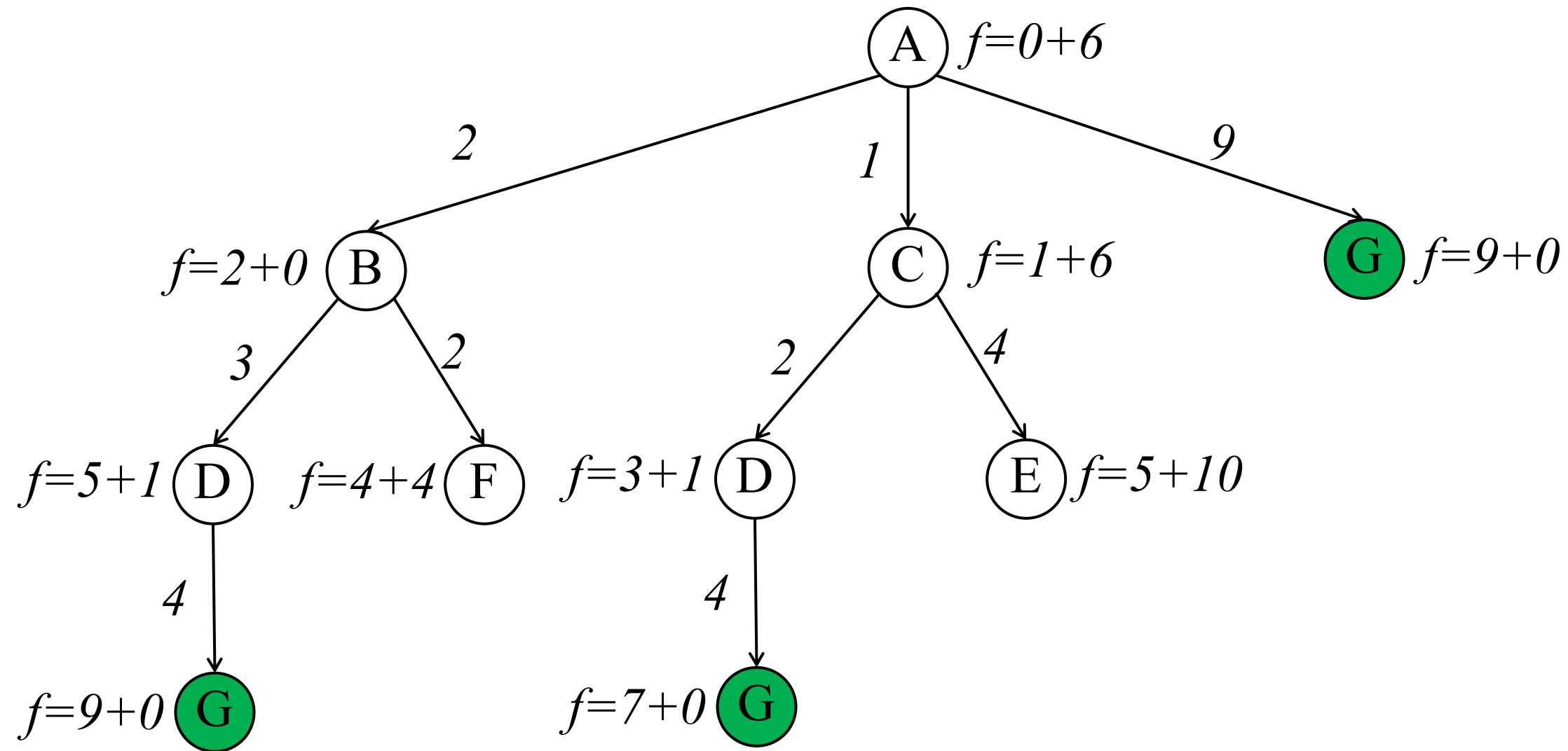
A, B, D, G (۱)

A, C, D, G (۲)

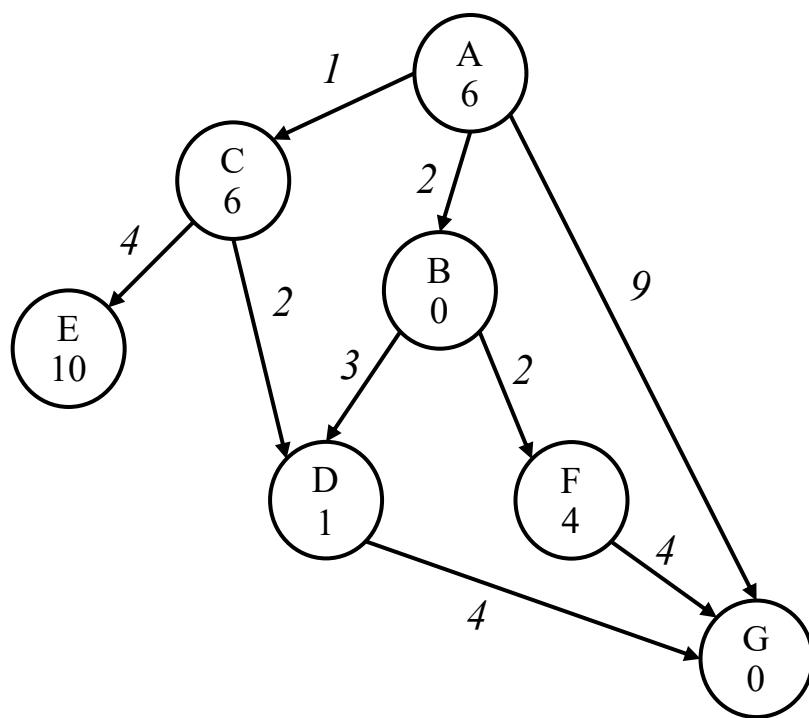
A, B, D, C, D, G (۳) ✓

A, C, D, B, F, G (۴)

حل تست ۲



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

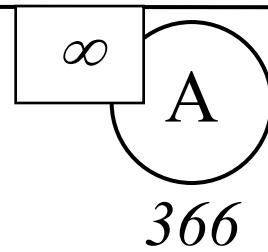


- ۱) فقط سازگار است.
 - ۲) فقط قابل قبول است. ✓
 - ۳) هم قابل قبول است هم سازگار.
 - ۴) نه قابل قبول است نه سازگار.
- $$h(C) \leq c(C, a, D) + h(D)$$
- $$6 \leq 2 + 1$$

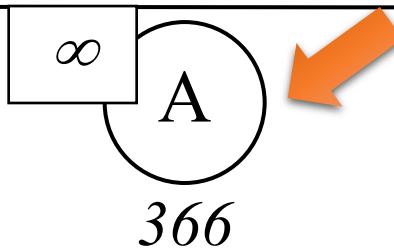
جستجوی اول بهترین بازگشتی – RBFS

ساختاری شبیه به جستجوی عمقی بازگشتی دارد اما به جای این که 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 بر می‌گردد تا مسیر جانشین را انتخاب نماید. در برگشت به عقب این الگوریتم مقدار f مربوط به بهترین برگ در زیردرخت فراموش شده را به یاد می‌آورد و می‌تواند تصمیم بگیرد آیا این زیردرخت باید بعداً دوباره ایجاد شود یا خیر.

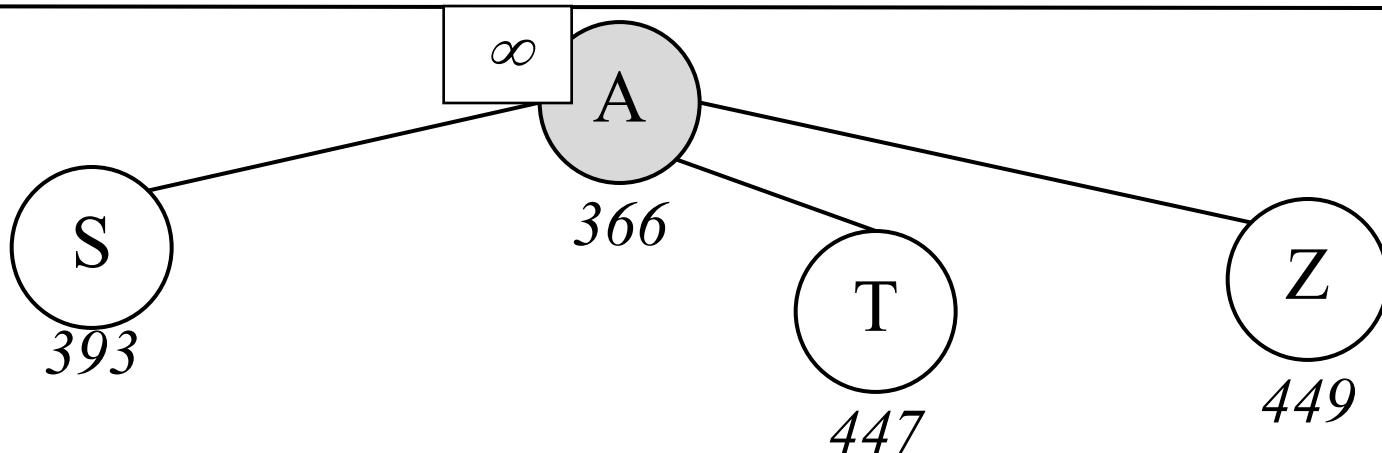
مثال – RBFS



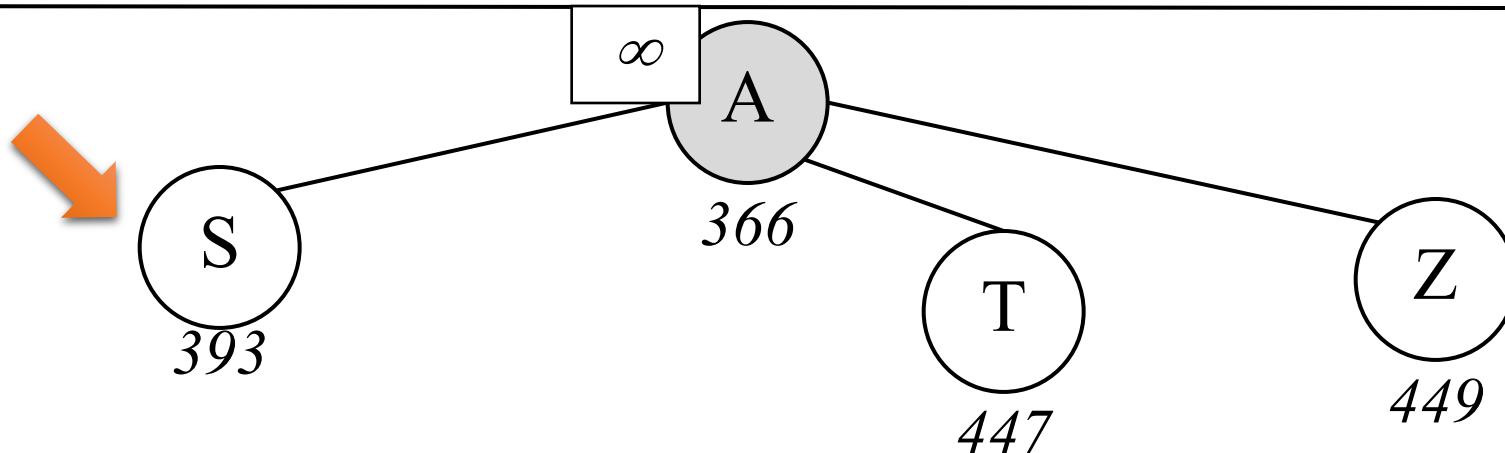
مثال – RBFS



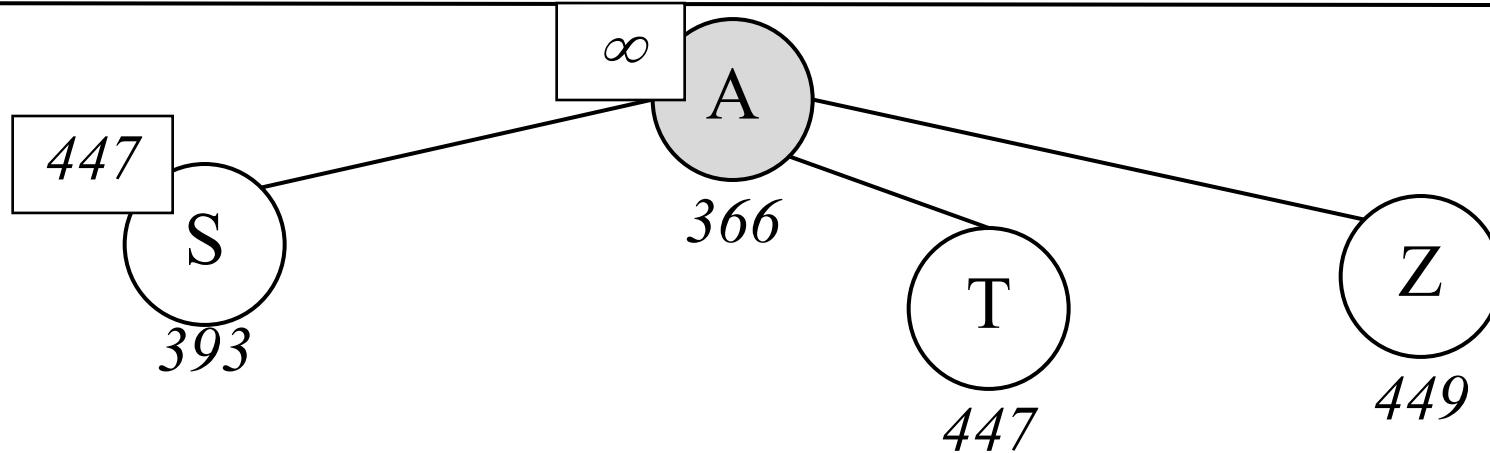
مثال – RBFS



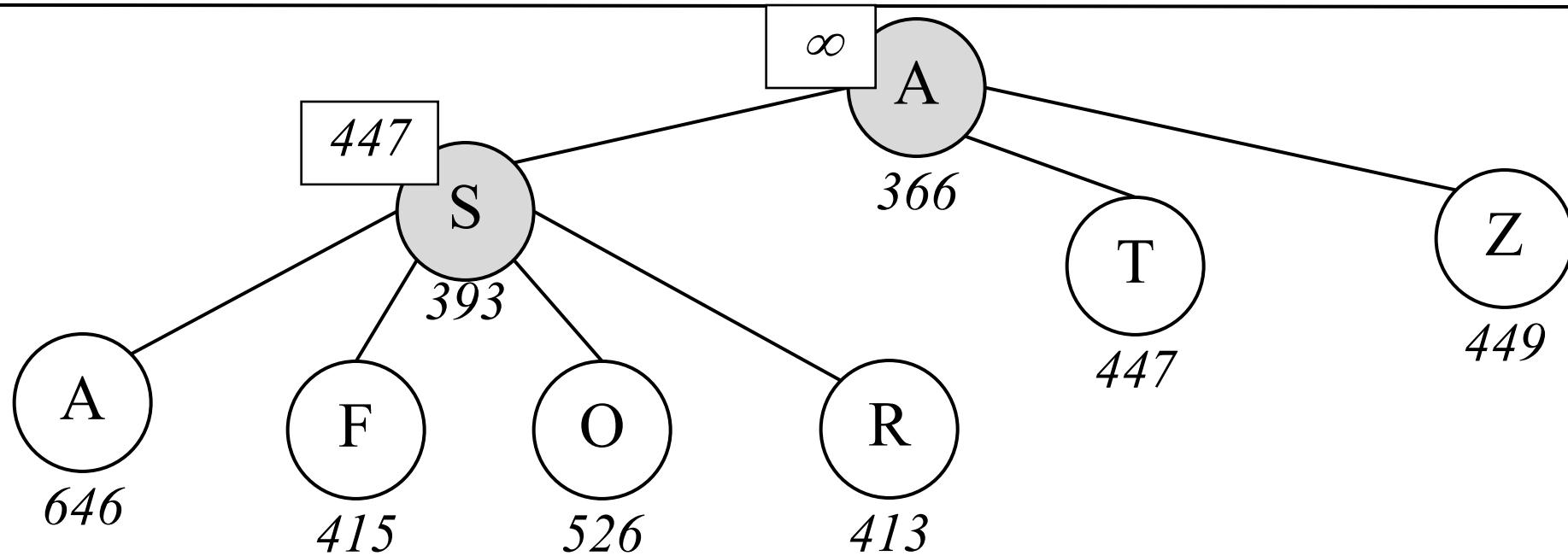
مثال – RBFS



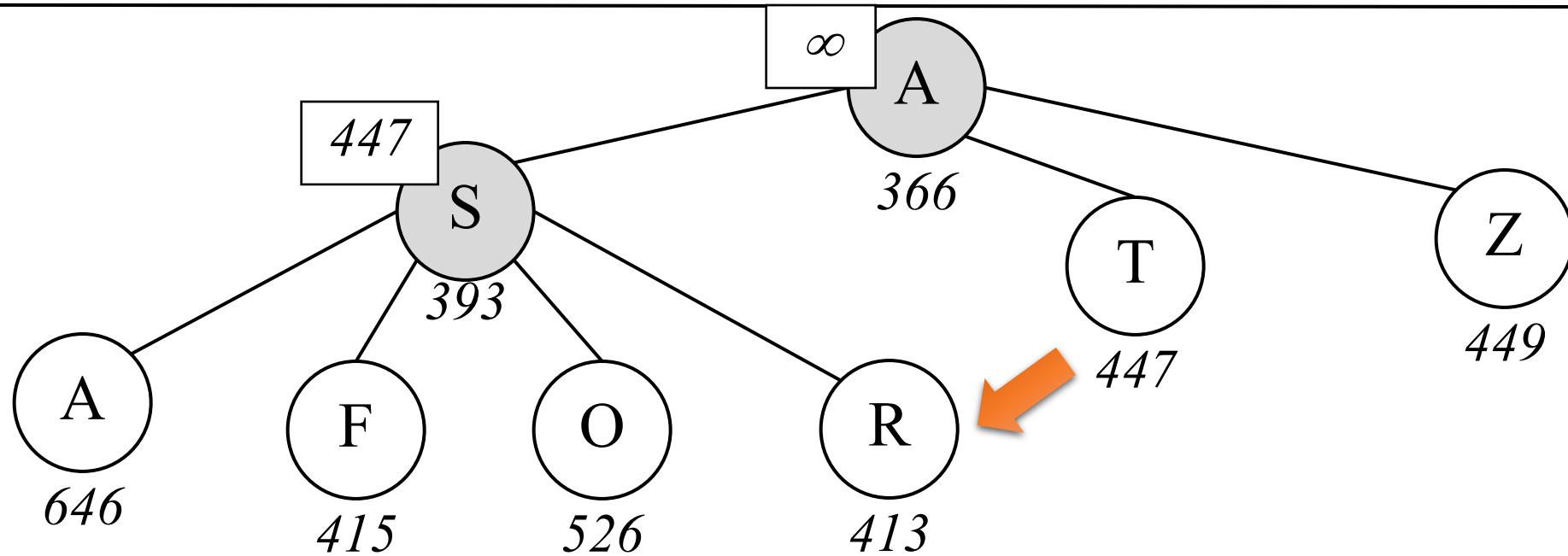
مثال – RBFS



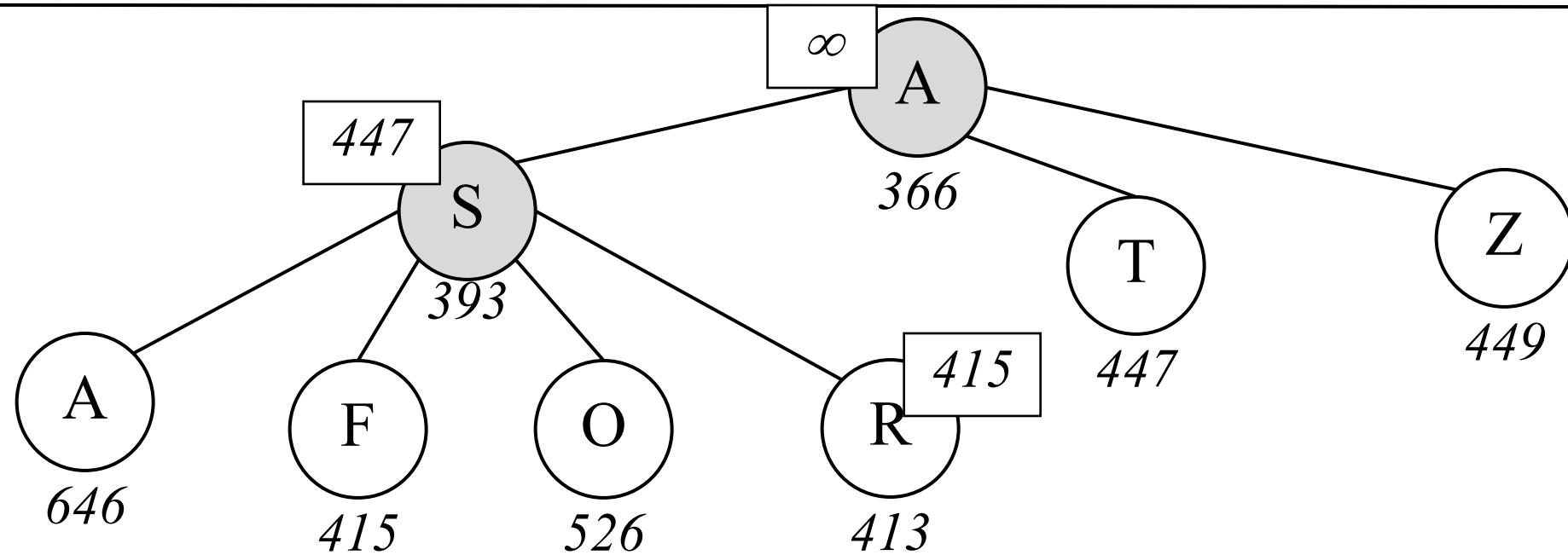
مثال – RBFS



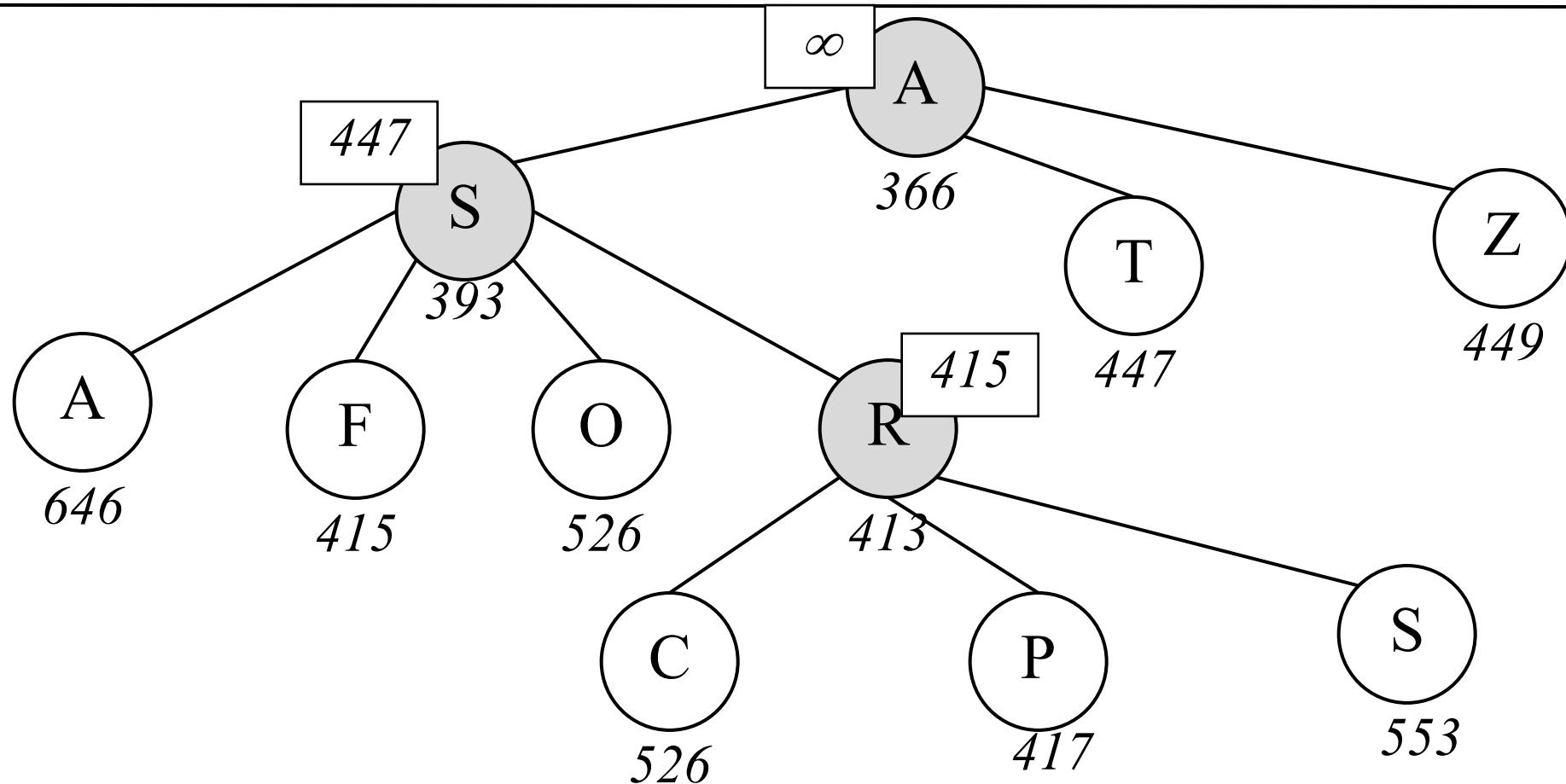
مثال – RBFS



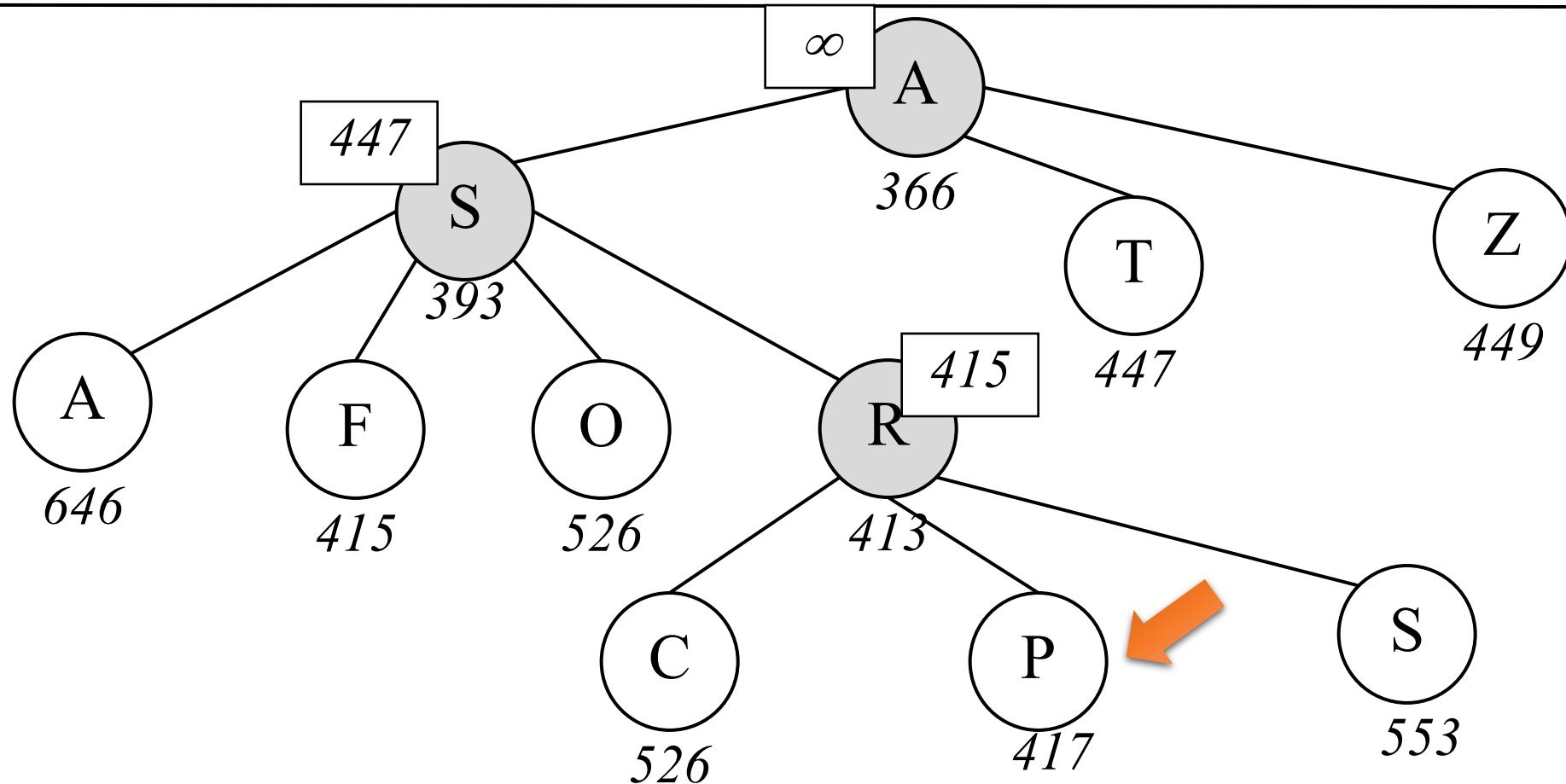
مثال – RBFS



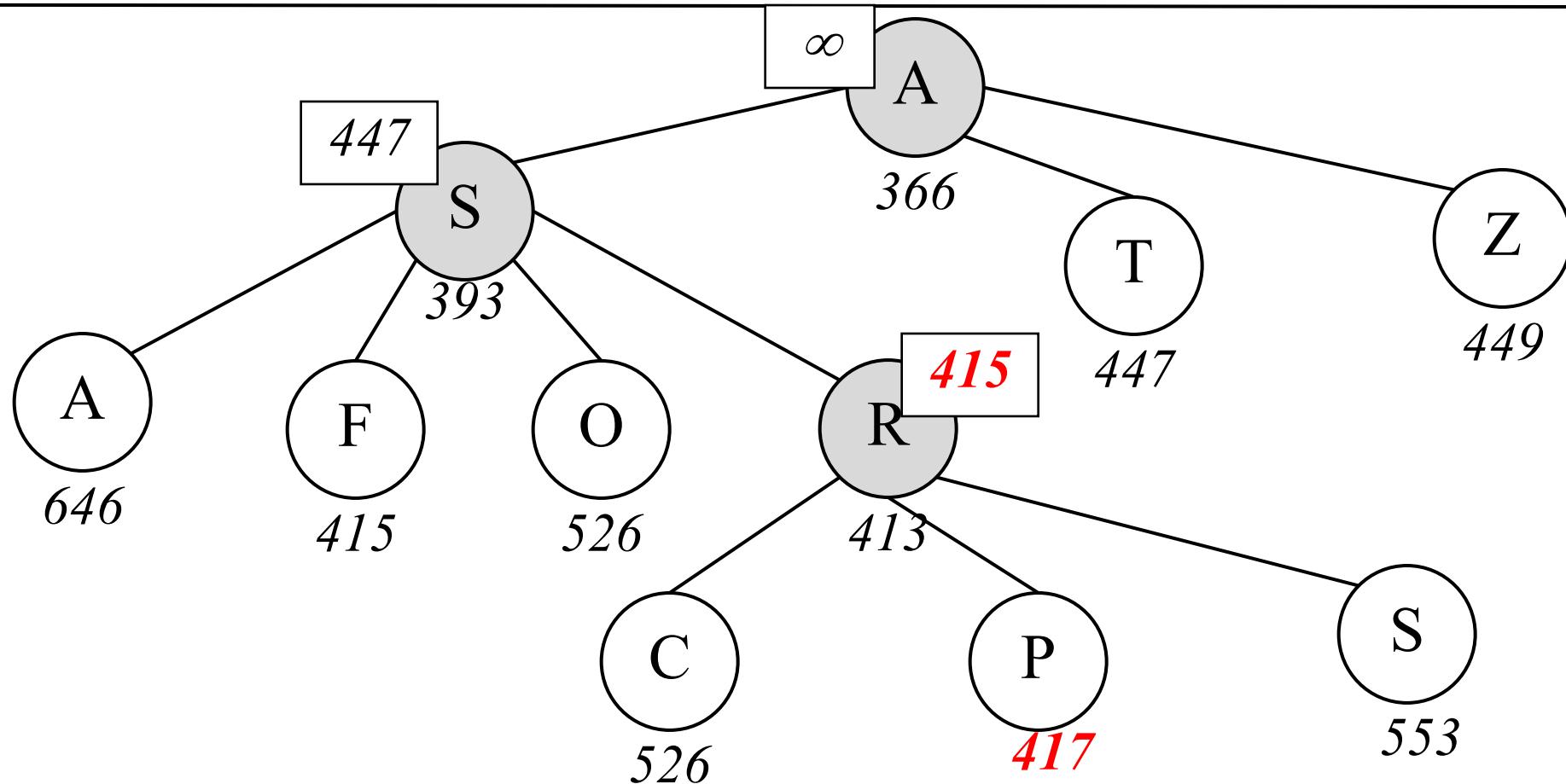
مثال – RBFS



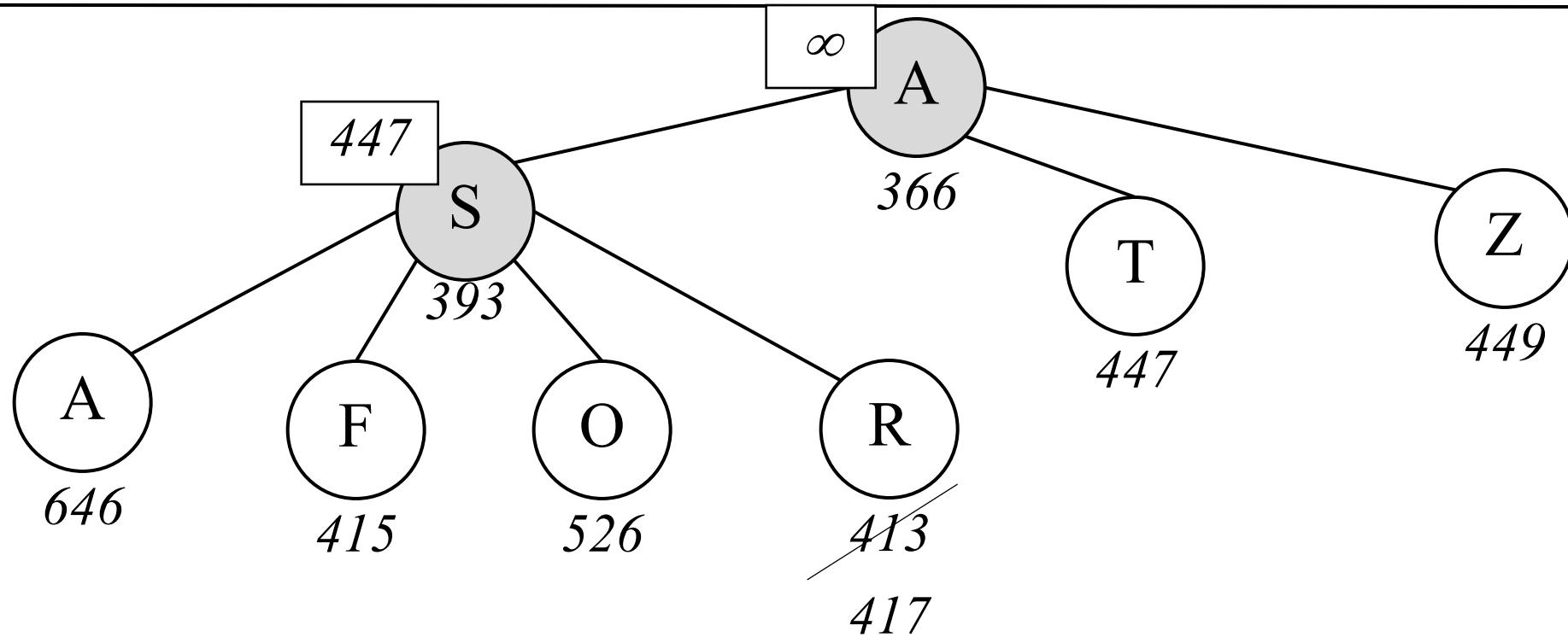
مثال – RBFS



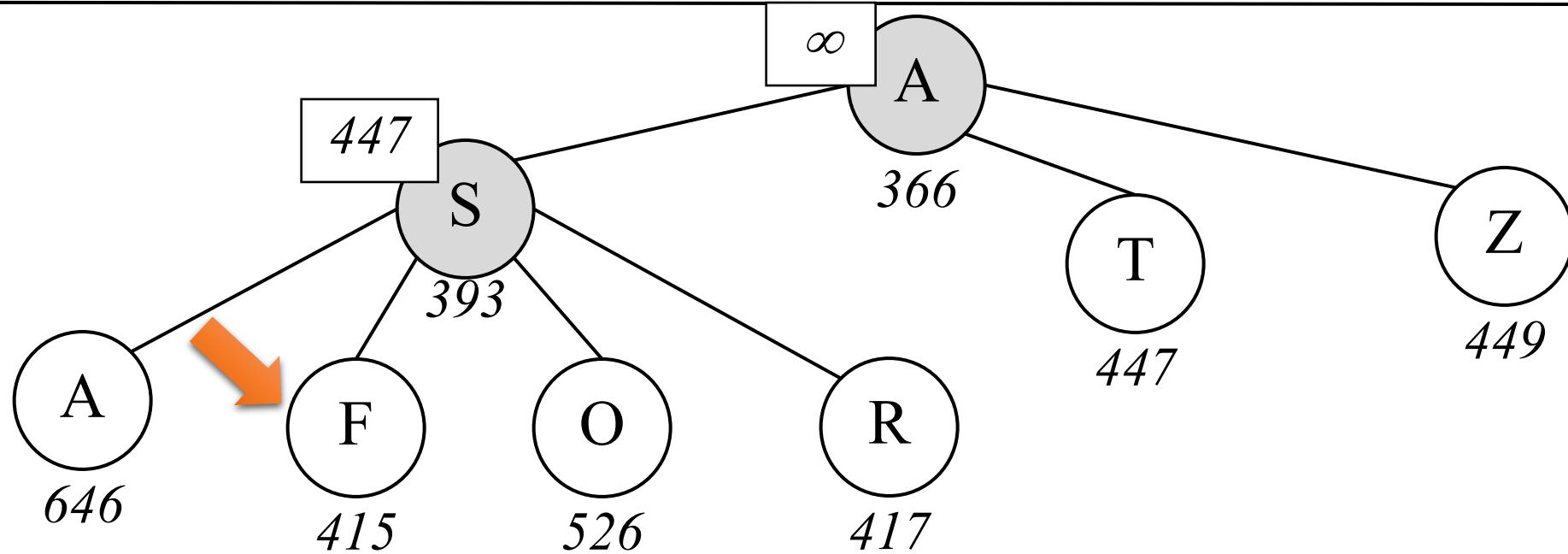
مثال – RBFS



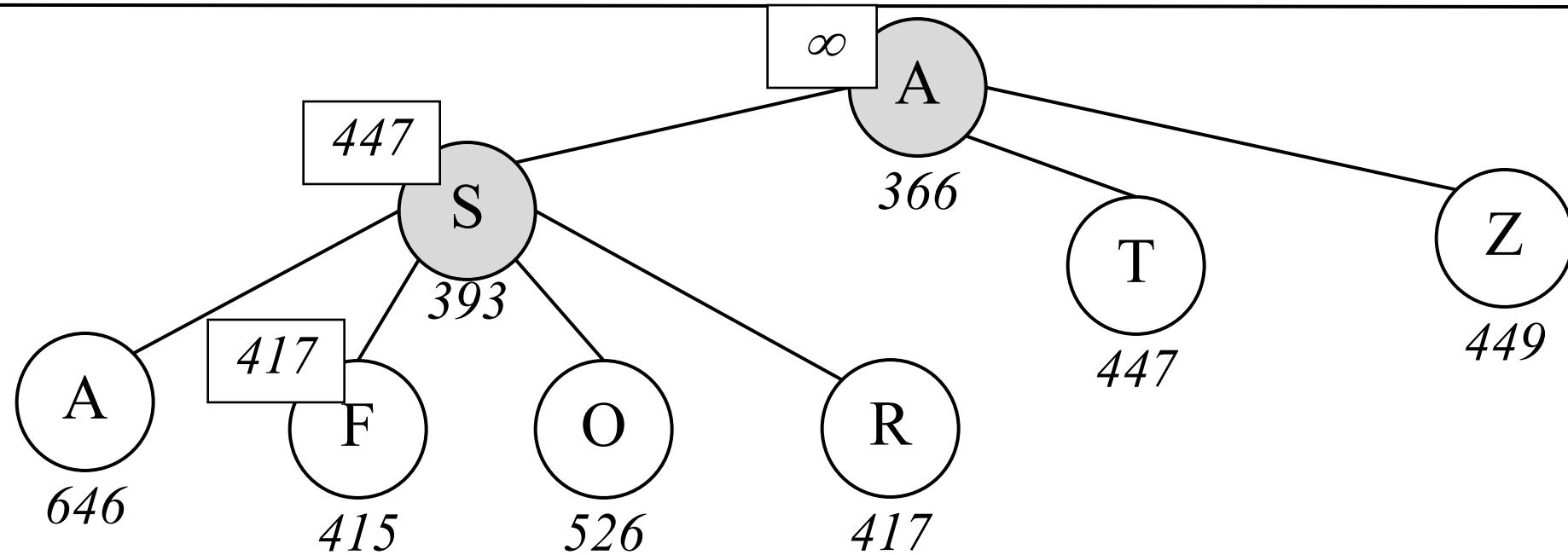
مثال – RBFS



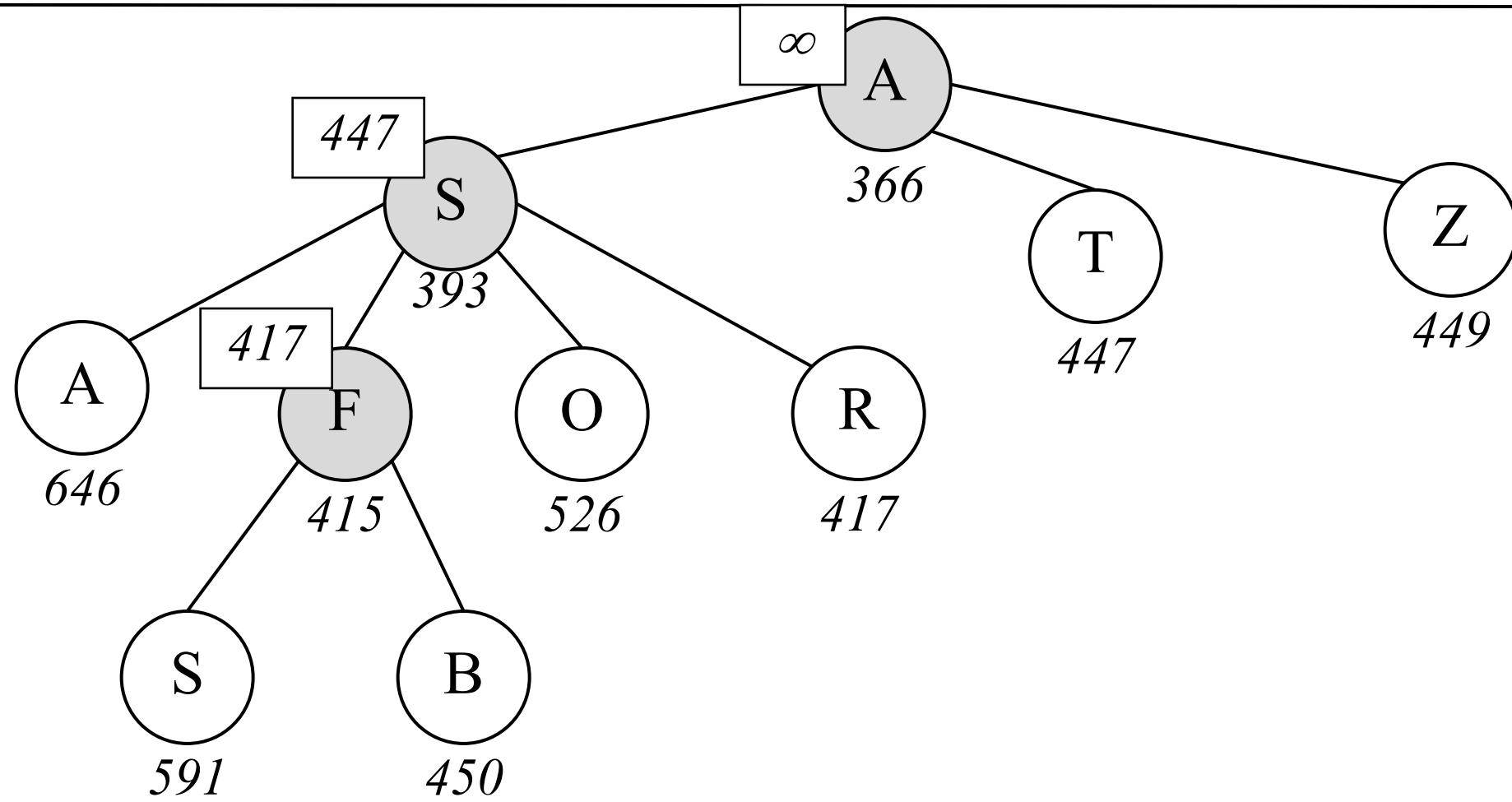
مثال – RBFS



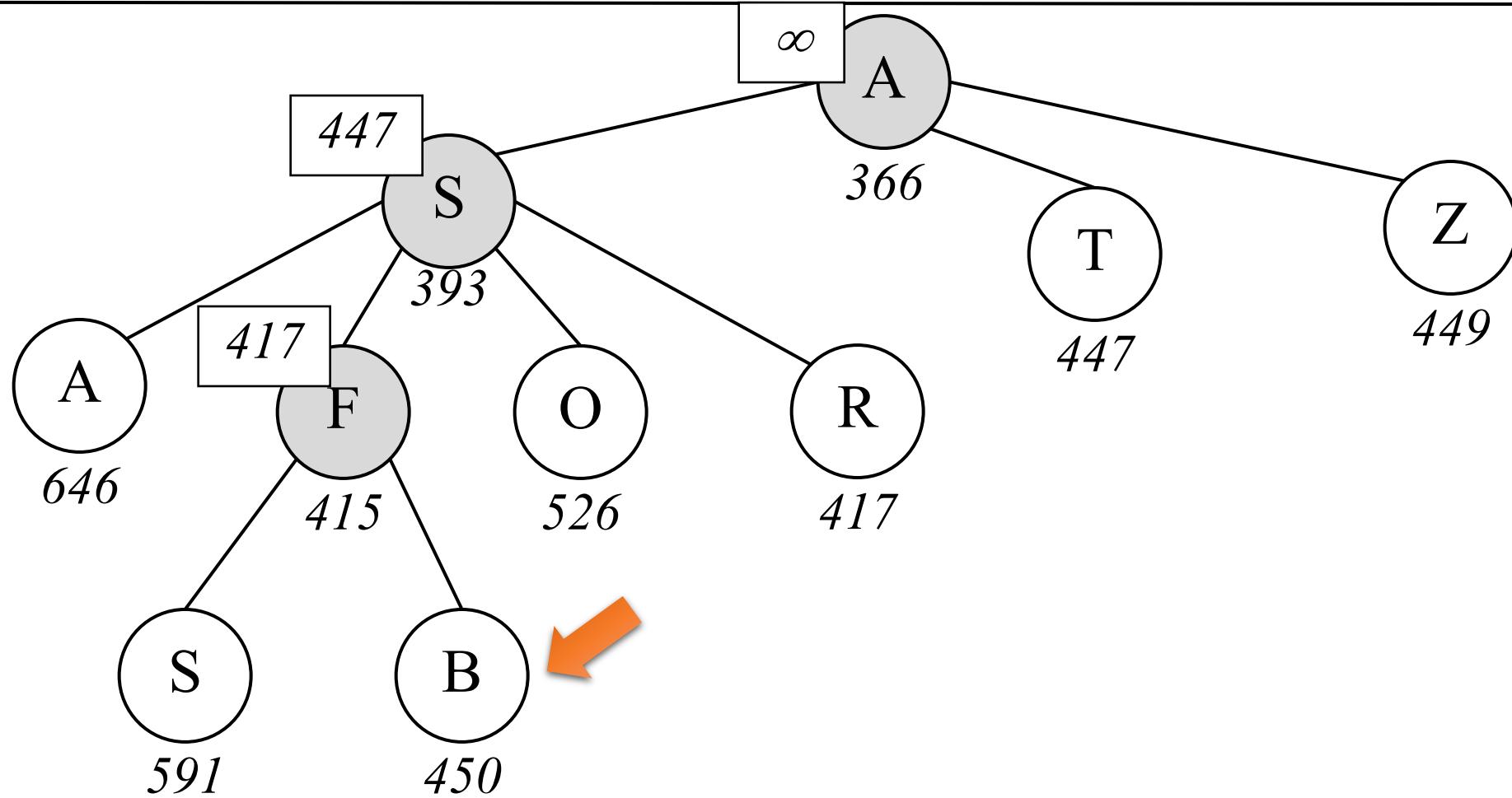
مثال – RBFS



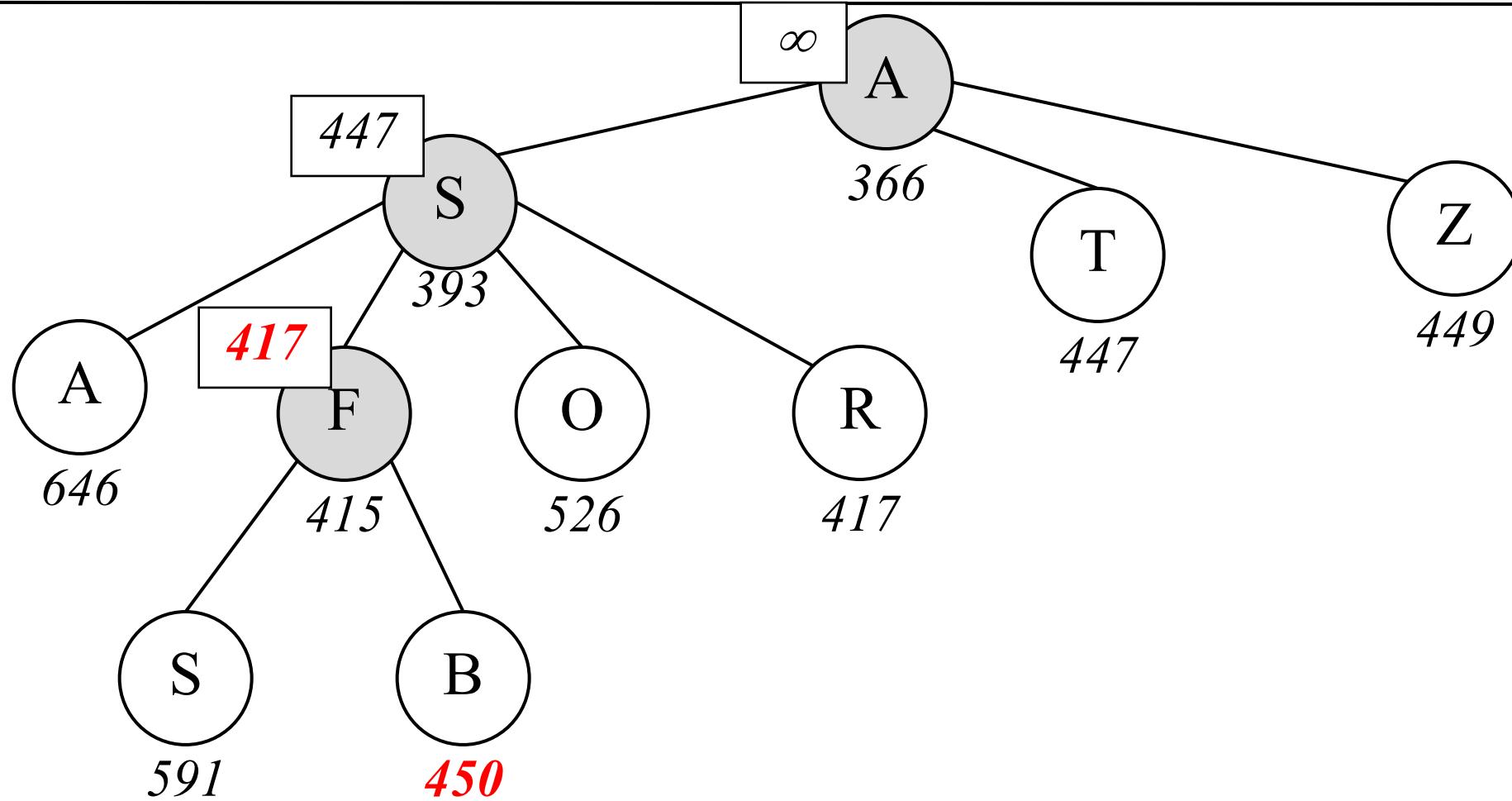
مثال – RBFS



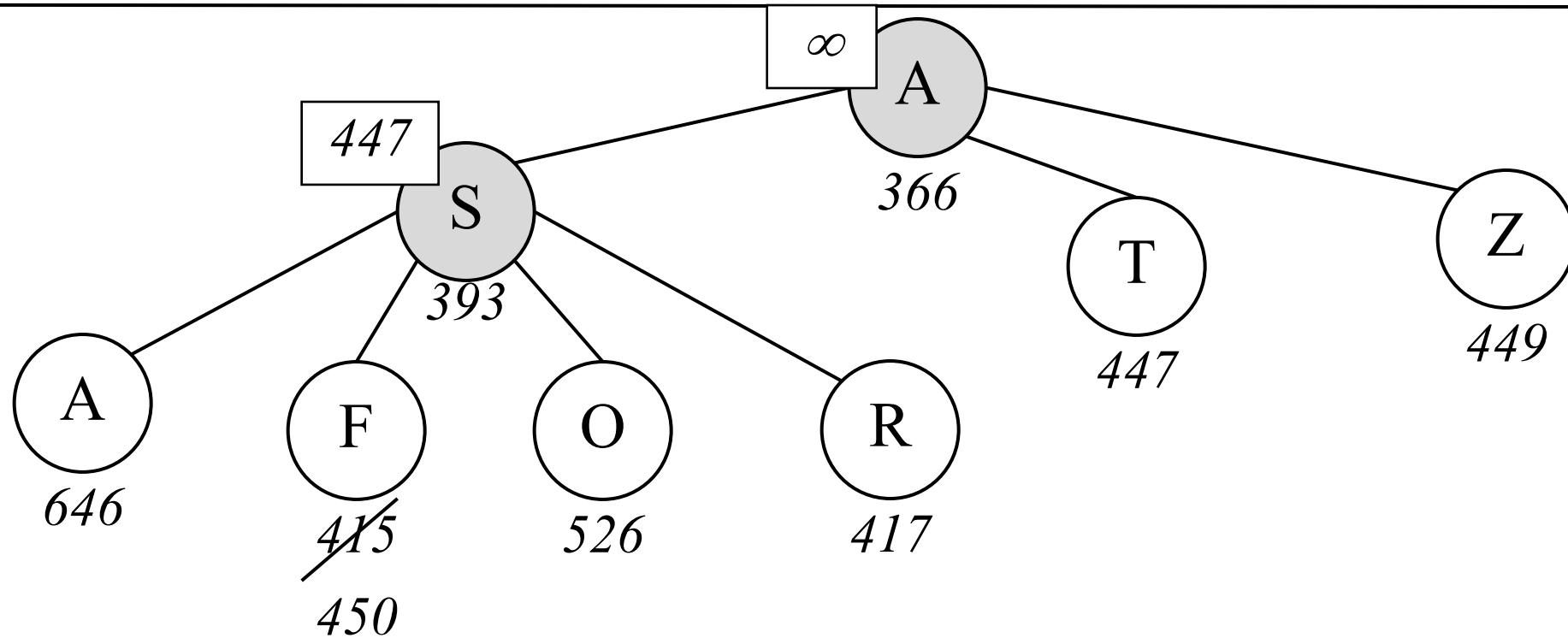
مثال – RBFS



مثال – RBFS

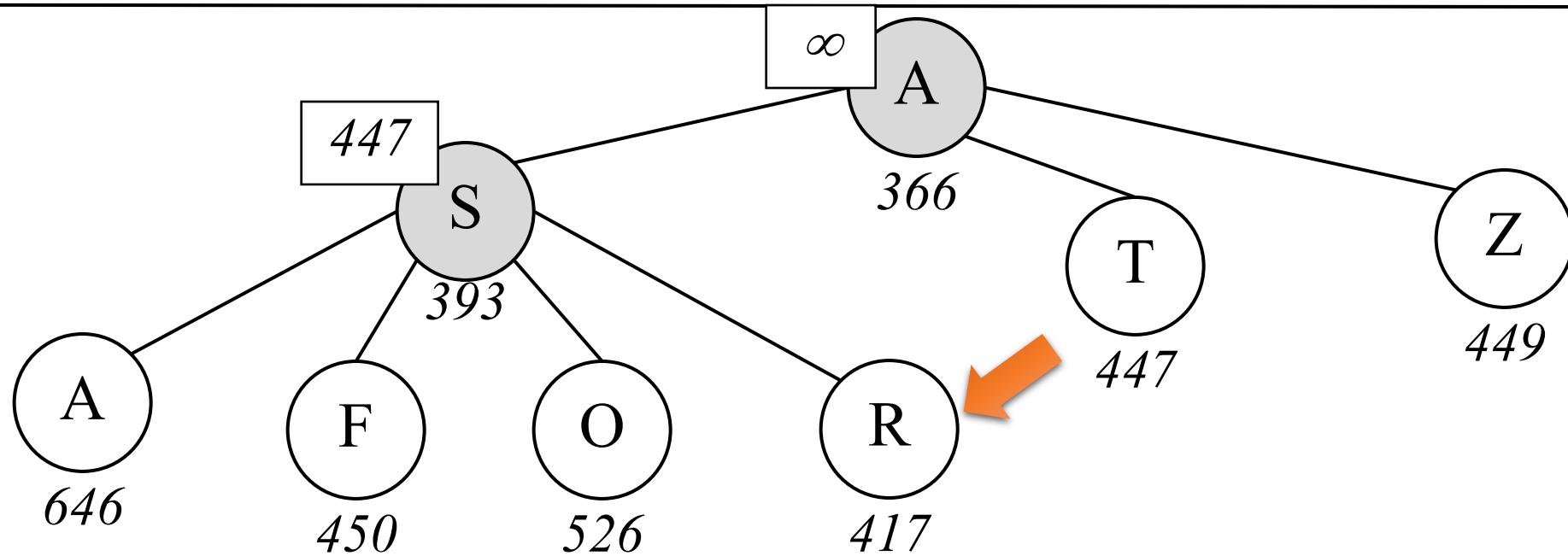


مثال – RBFS

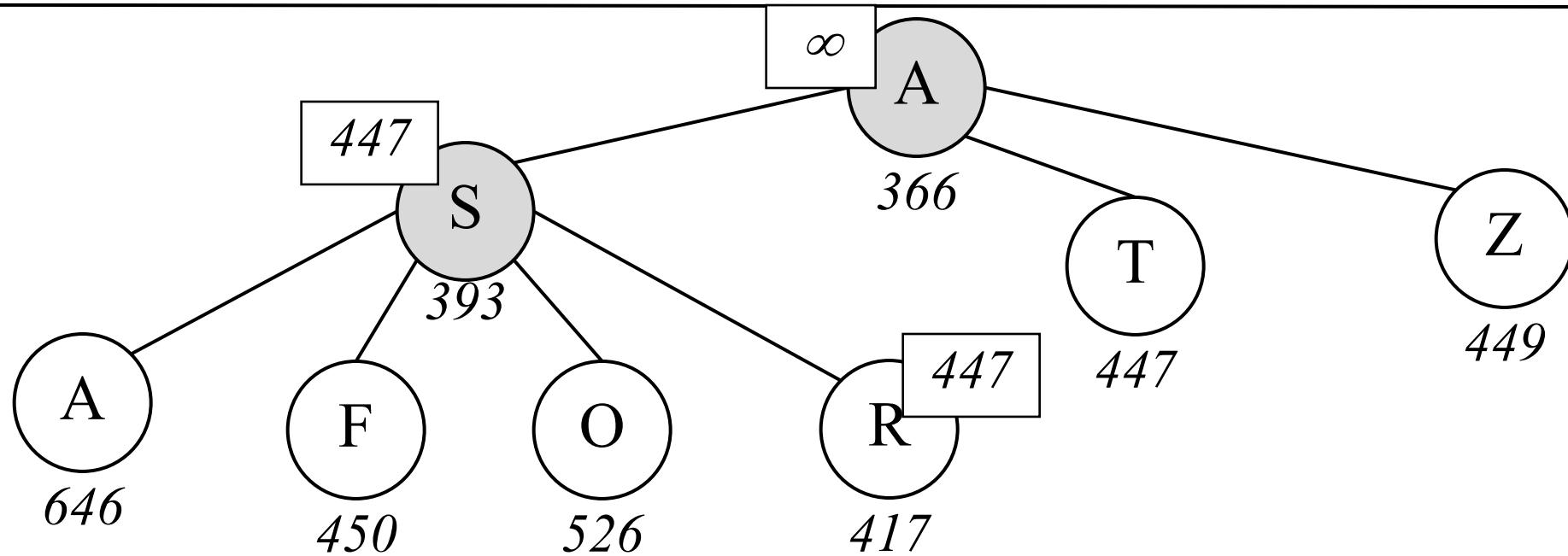


w

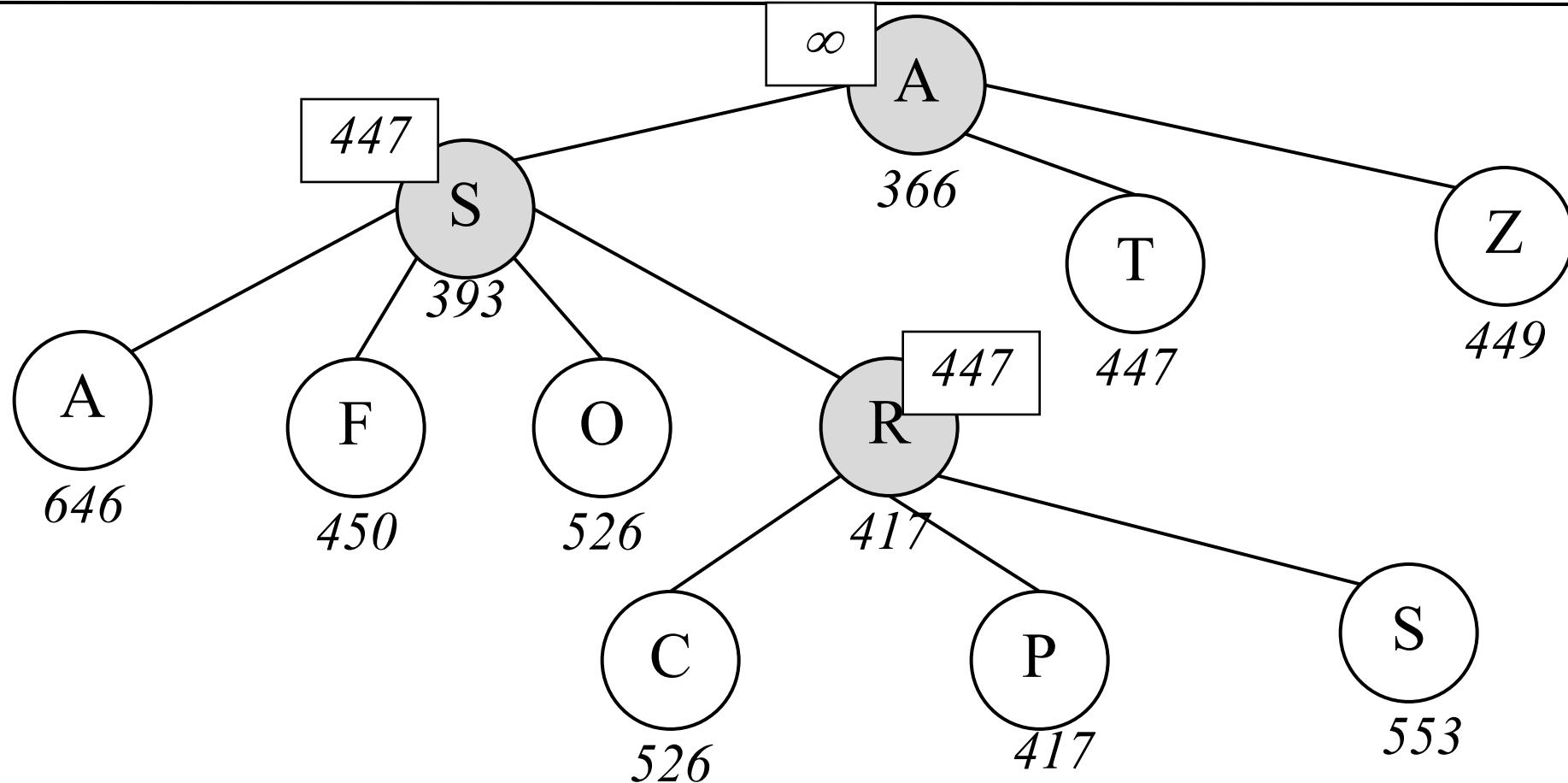
مثال – RBFS



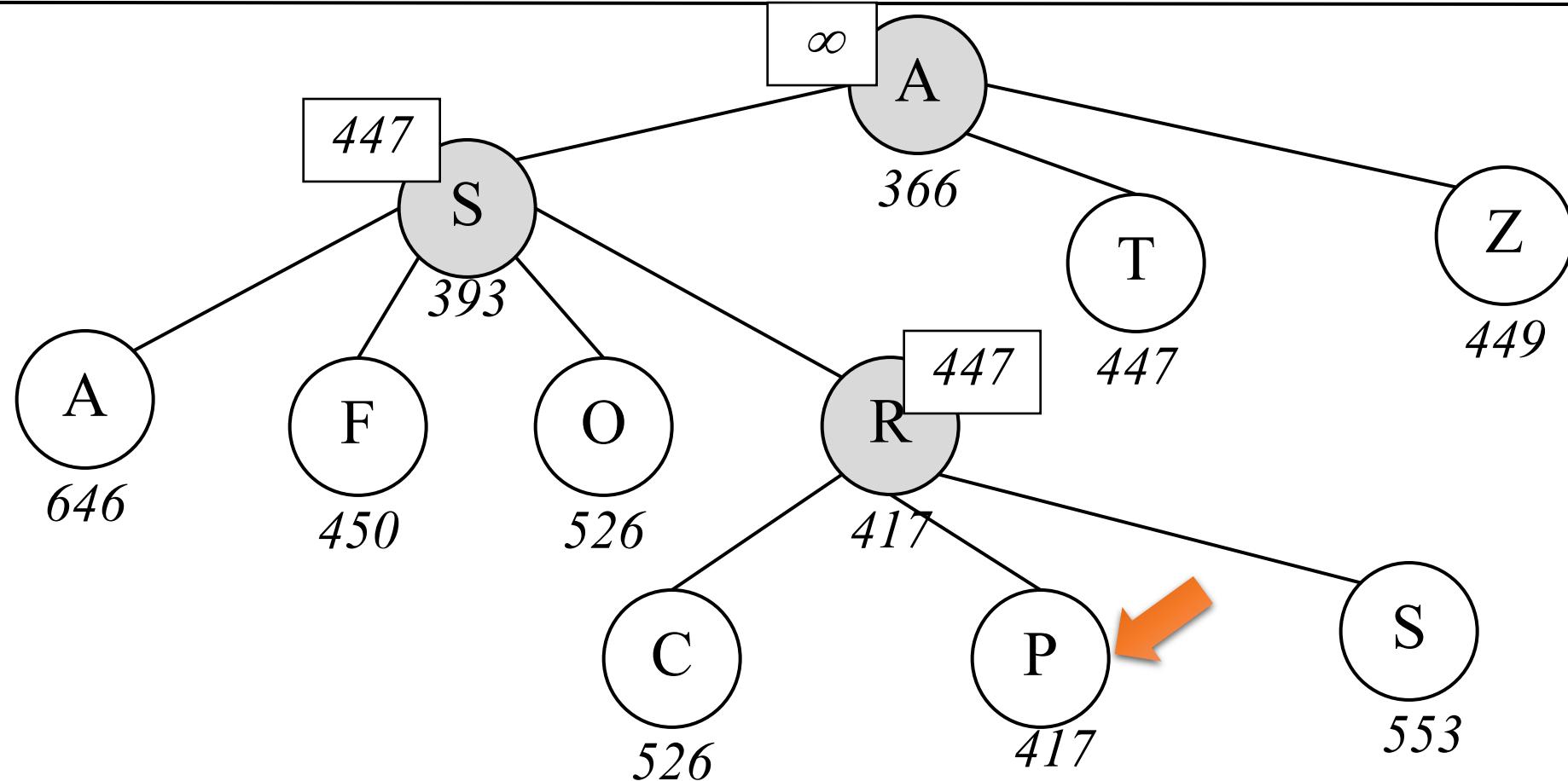
مثال – RBFS



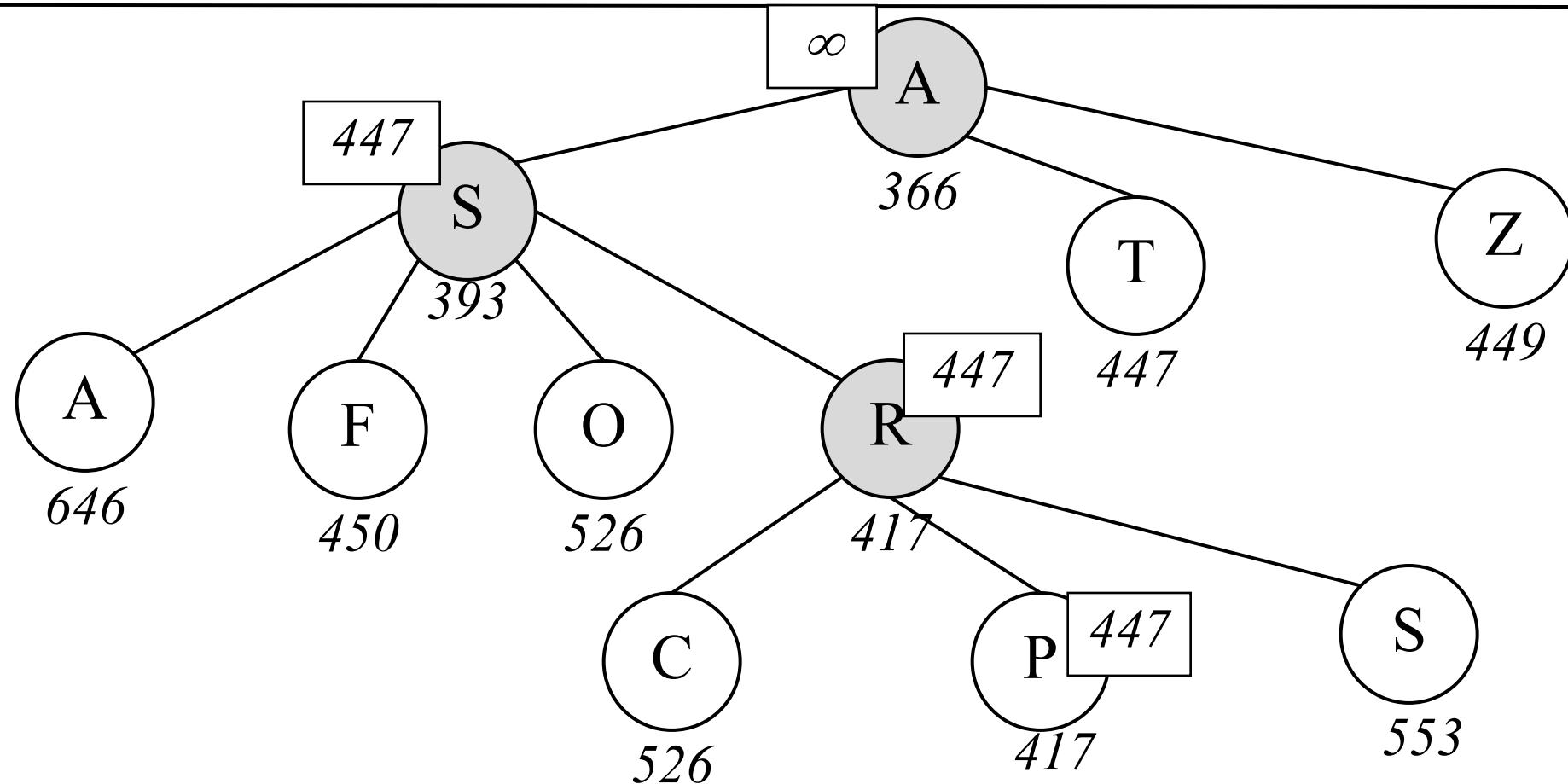
مثال – RBFS



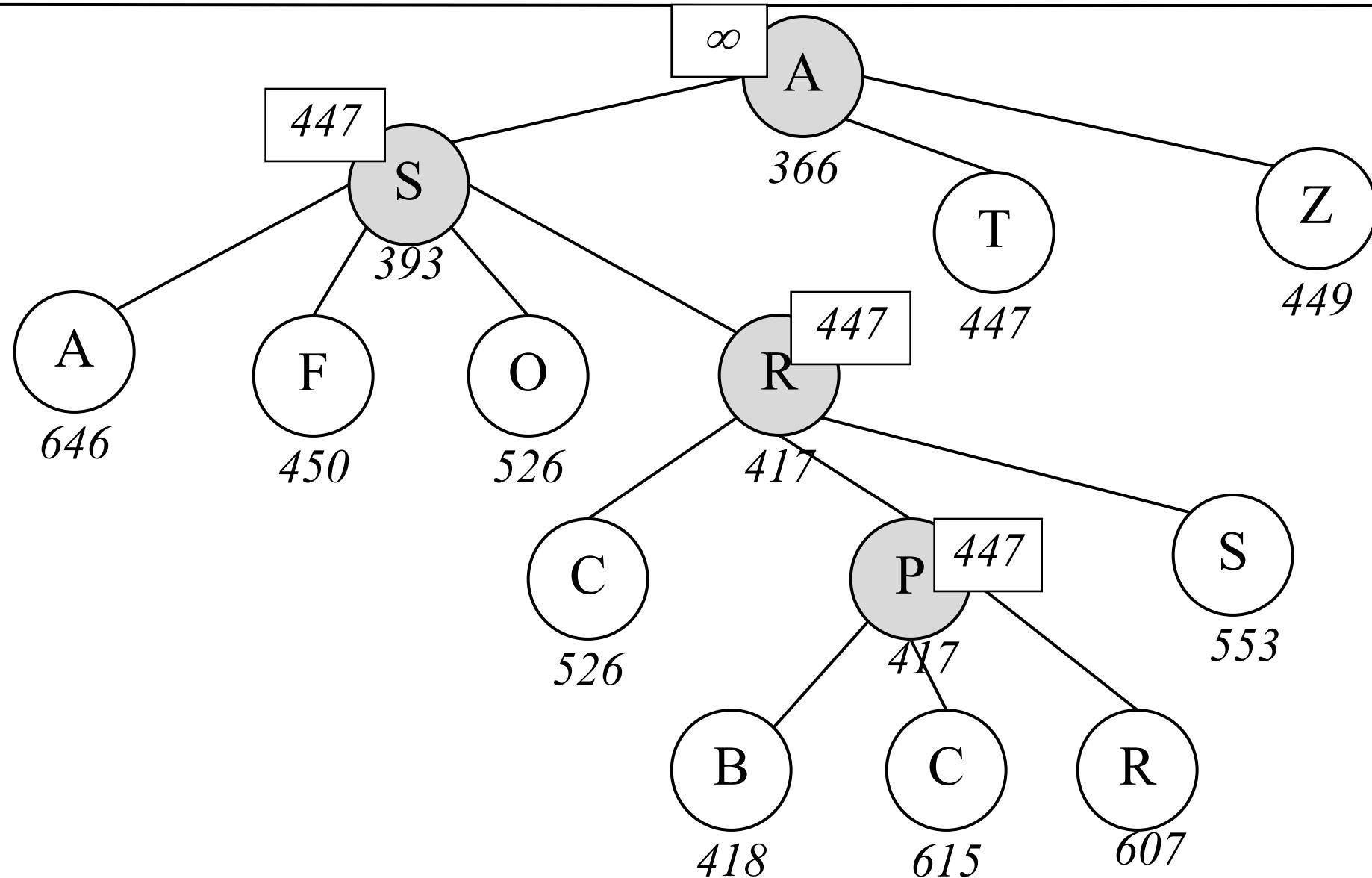
مثال – RBFS



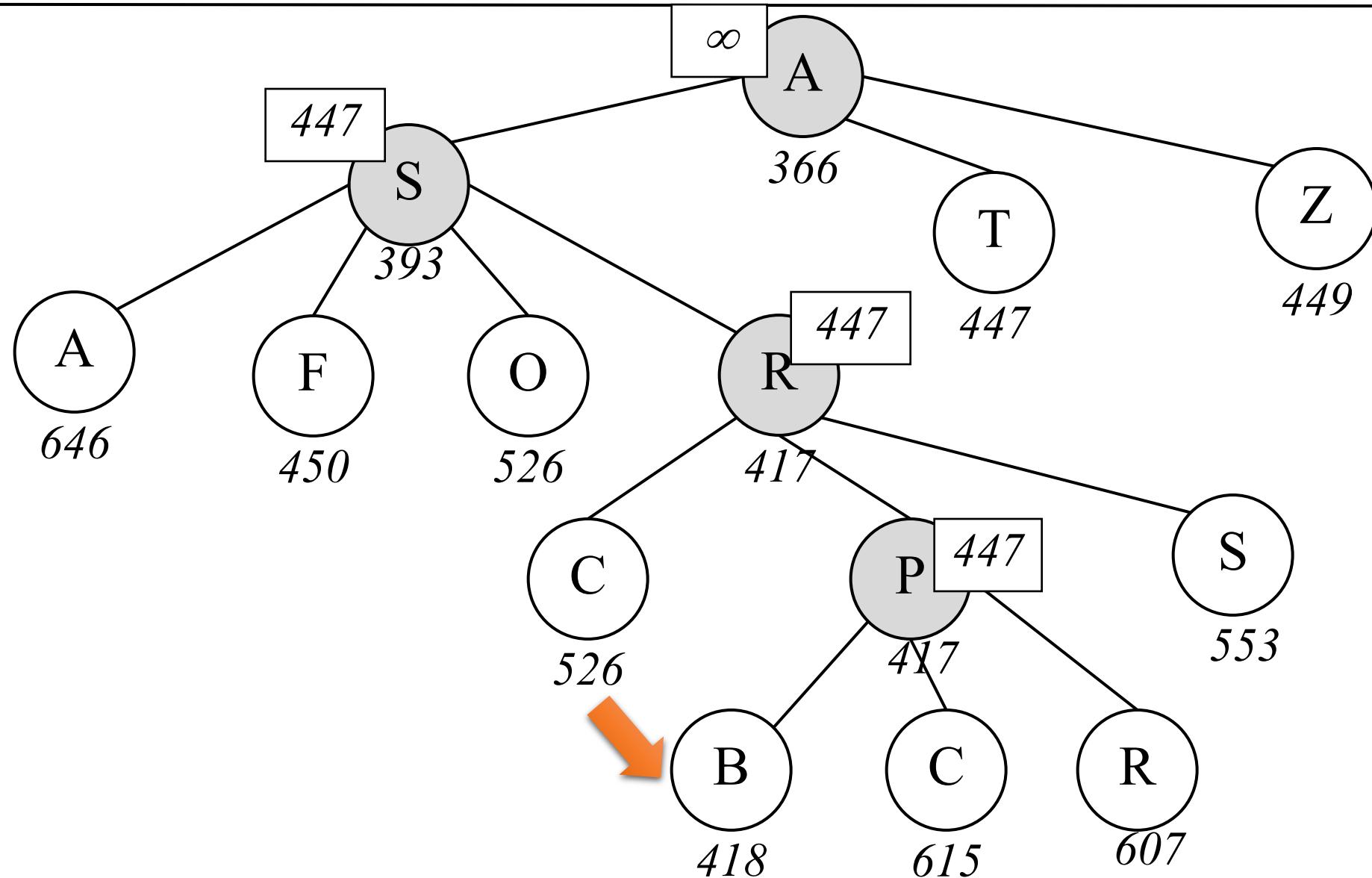
مثال – RBFS



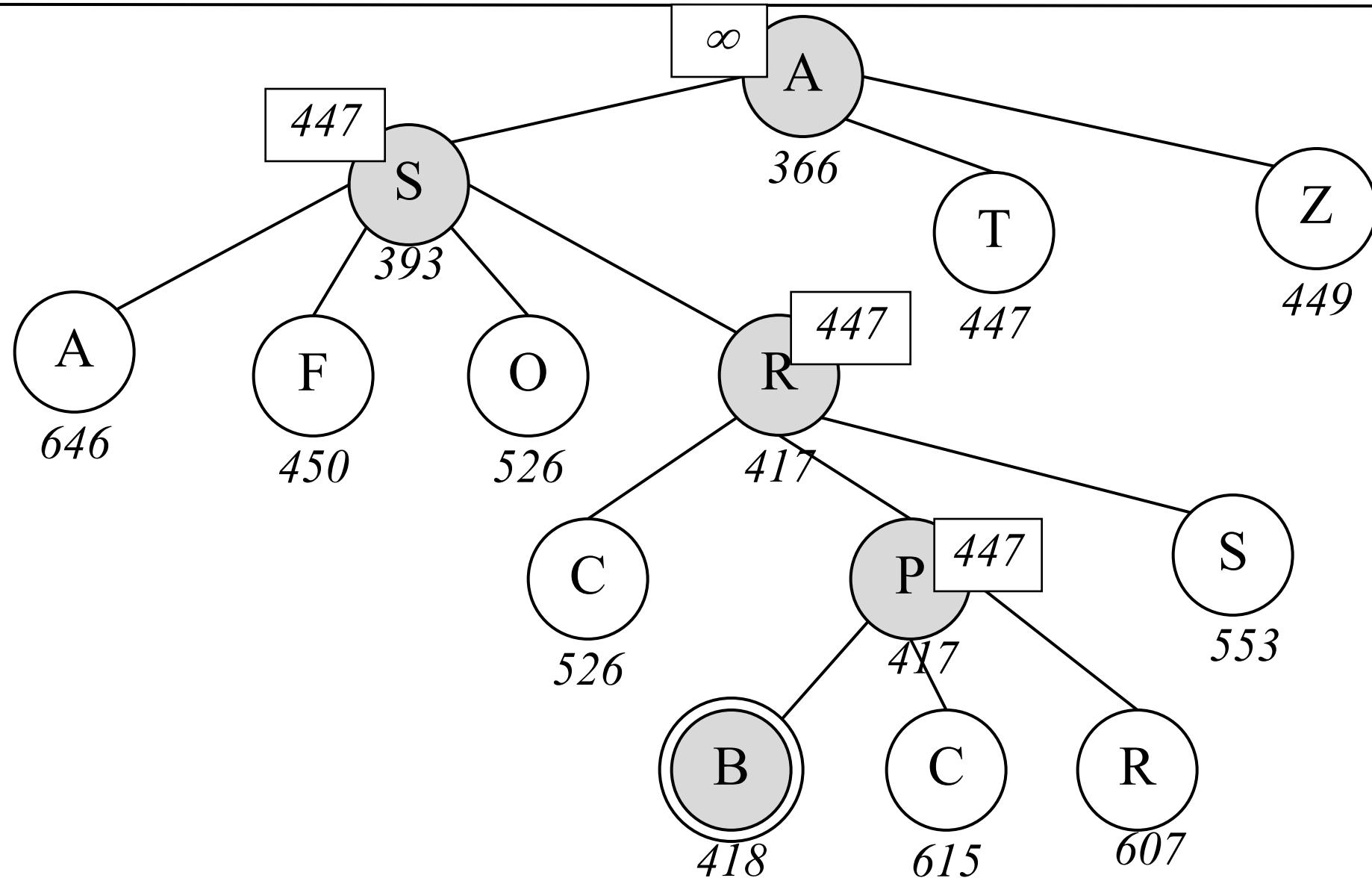
مثال – RBFS



مثال – RBFS



مثال – RBFS



جستجوی اول بهترین بازگشتی – RBFS

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

```
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  $\leftarrow$  []
```

```
    for each action in problem.ACTIONS(node.STATE) do
```

```
        add CHILD-NODE(problem, node, action) into successors
```

```
    if successors is empty then return failure,  $\infty$ 
```

```
    for each s in successors do /* update f with value from previous search, if any */
```

```
        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
```

- 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 کند 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 عمیقی پیماش 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* دقیقاً 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 برگرد.

الگوریتم SMA*

- ممکن است تمام گره‌های برگ با هم برابر باشند در این صورت 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 این صورت 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 موجود مسیر بهینه را پیدا کند. در این موارد، SMA* مقدار f-cost این برگ را بی‌نهایت می‌گذارد تا دیگر انتخاب نشود.

الگوریتم SMA *

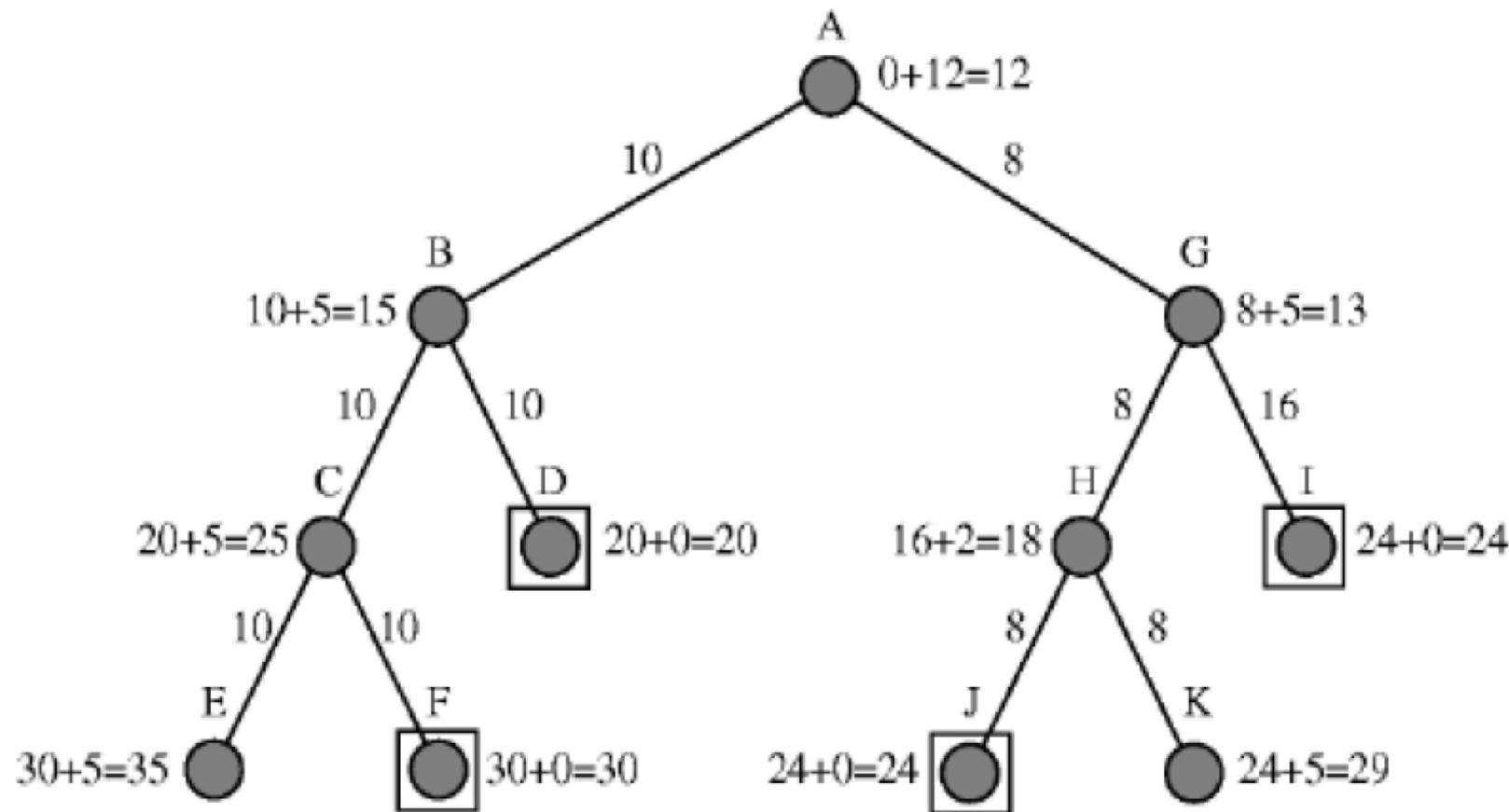
```
function SMA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: Queue, a queue of nodes ordered by f-cost

  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 $failure$ بروگردان.
- 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 راه حل پیدا شده را بروگردان.
- 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 آن گاه 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 آن گاه n را از صف حذف کن.
- ۸- اگر حافظه پر بود آن گاه 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 به عنوان 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

$$Q = \{A\}$$



فاز صفر

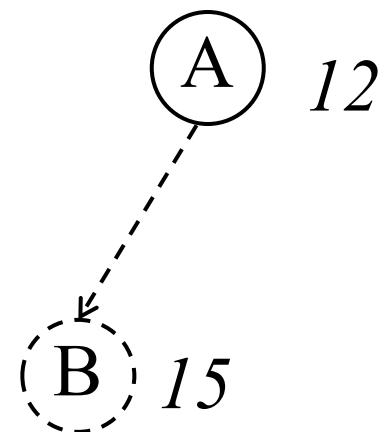
$$Q = \{A\}$$



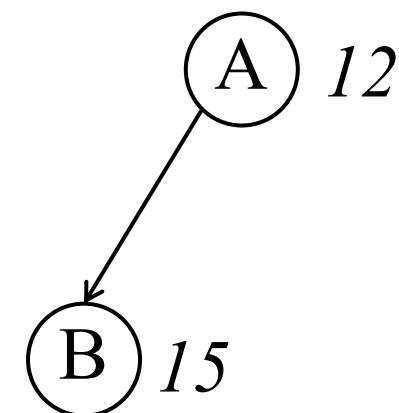
$$n = A$$

$$s = B$$

$$Q = \{A\}$$



$$Q = \{A, B\}$$

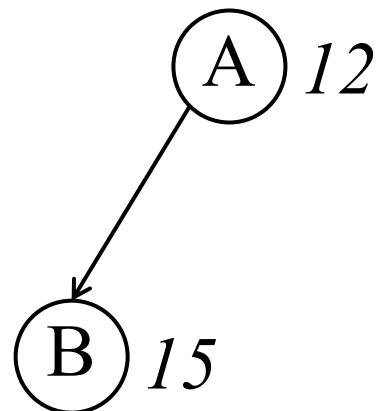


فاز اول

مثال * SMA با محدودیت ۳ خانه حافظه

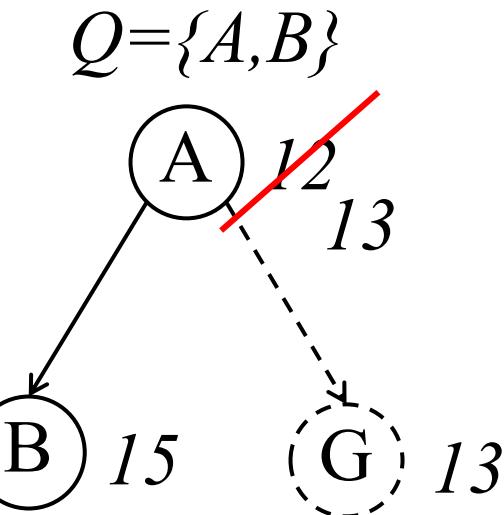
فاز دوم

$$Q = \{A, B\}$$

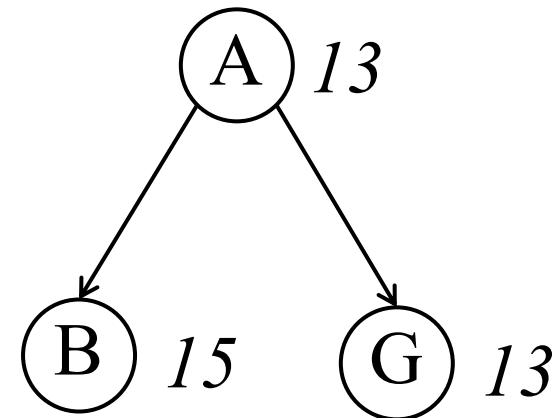


$$n = A$$

$$s = G$$



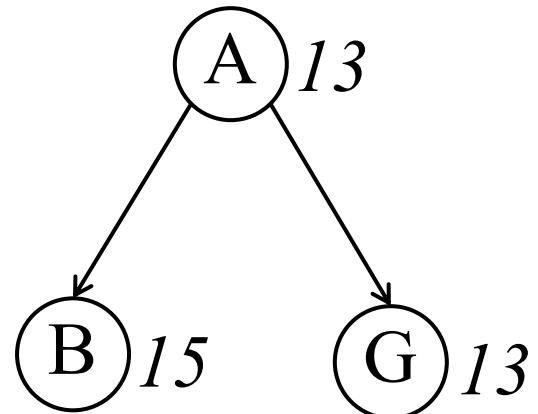
$$Q = \{A, B, G\}$$



مثال * SMA با محدودیت ۳ خانه حافظه

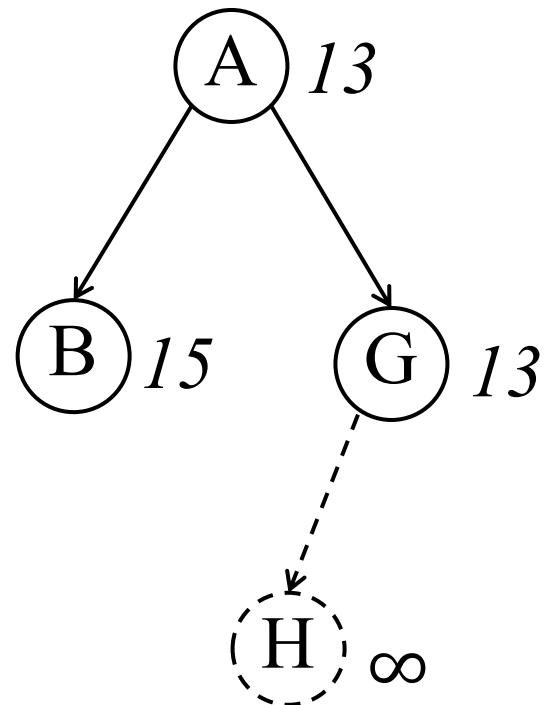
فاز سوم

$$Q = \{A, B, G\}$$

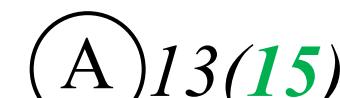


$$Q = \{A, B, G\}$$

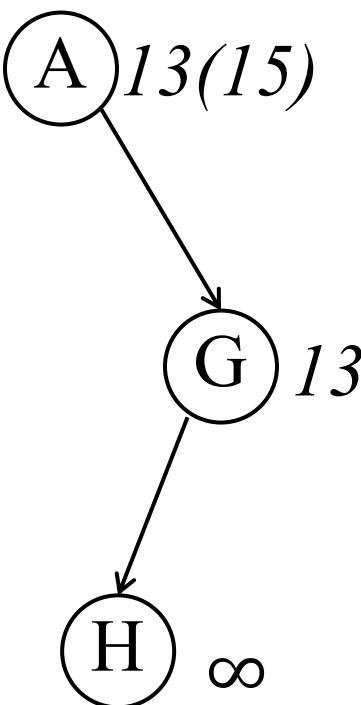
$$n = G$$
$$s = H$$



$$Q = \{A, G\}$$



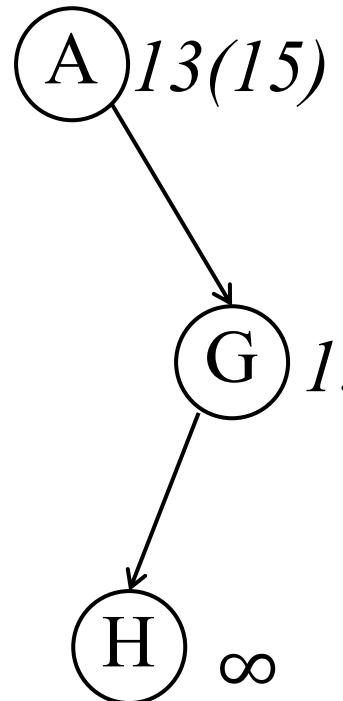
$$Q = \{A, G, H\}$$



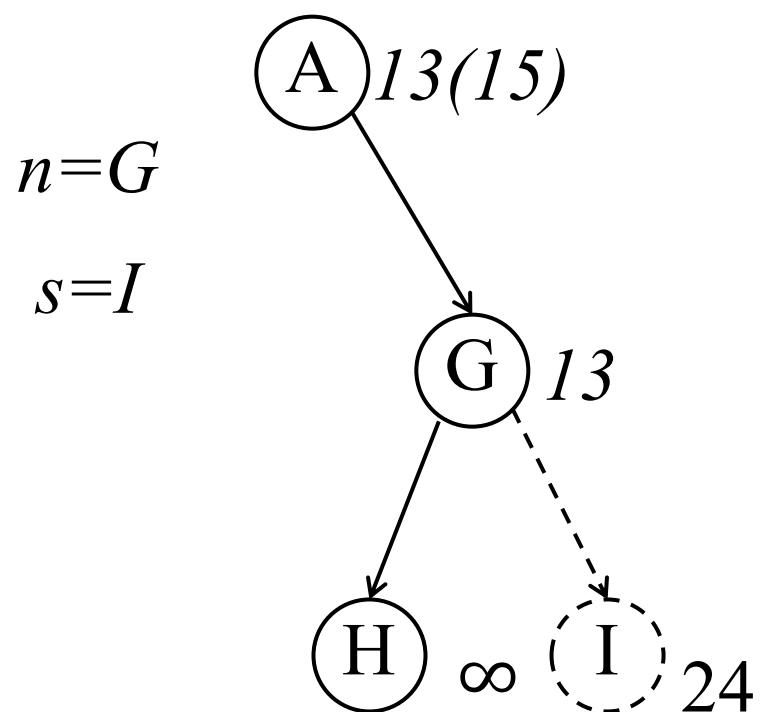
مثال * SMA با محدودیت ۳ خانه حافظه

فاز چهارم

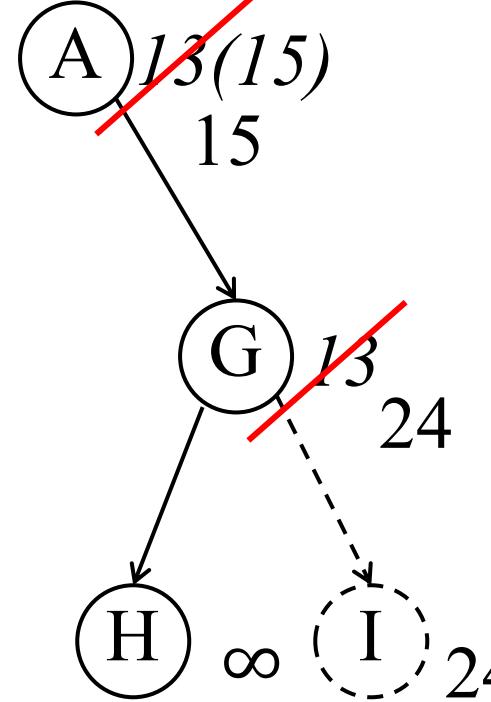
$$Q = \{A, G, H\}$$



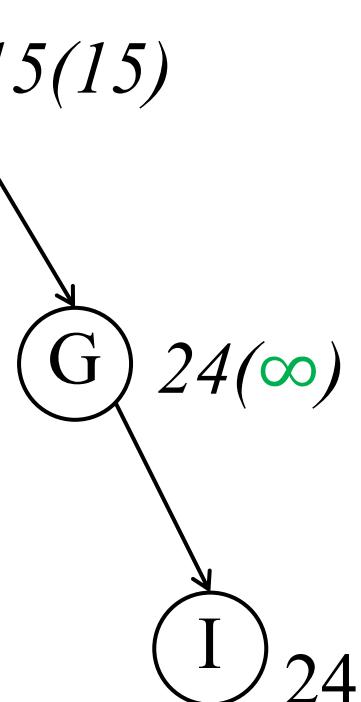
$$Q = \{A, G, H\}$$



$$Q = \{A, G, H\}$$



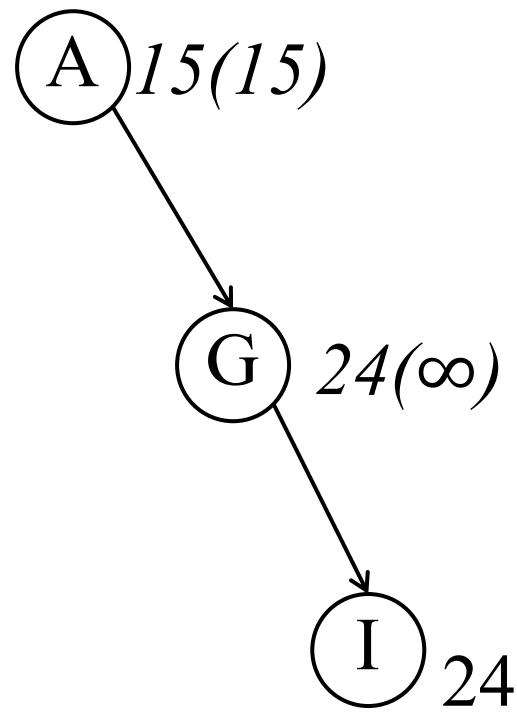
$$Q = \{A, G, I\}$$



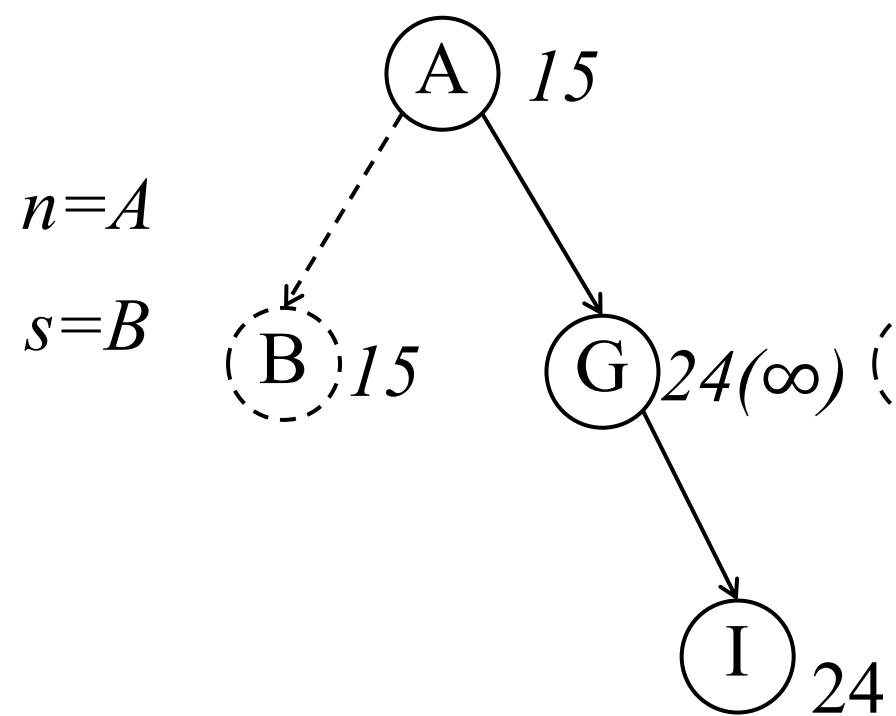
مثال * SMA با محدودیت ۳ خانه حافظه

فاز پنجم

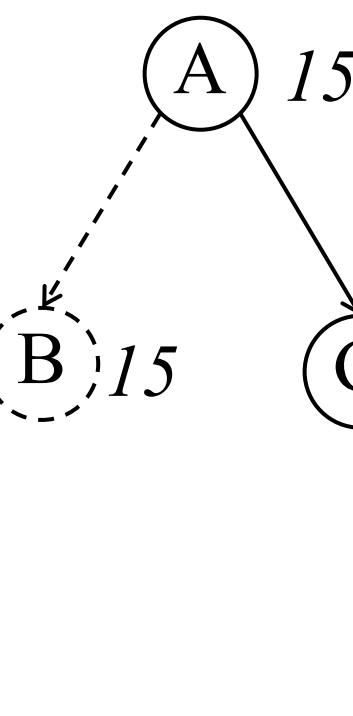
$$Q = \{A, G, I\}$$



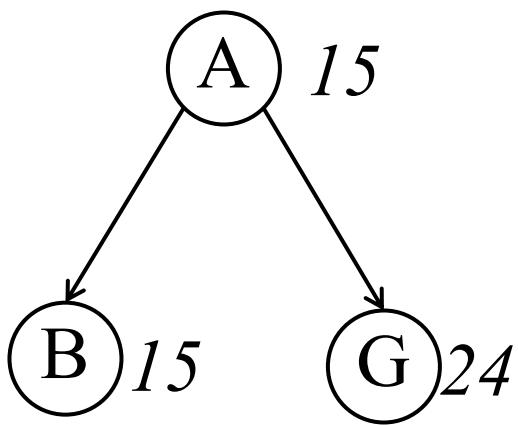
$$Q = \{A, G, I\}$$



$$Q = \{A, G\}$$



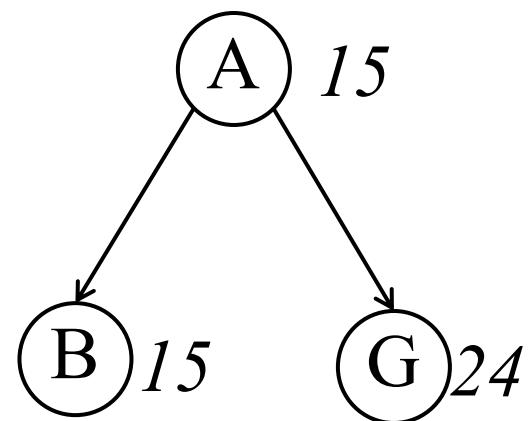
$$Q = \{A, G, B\}$$



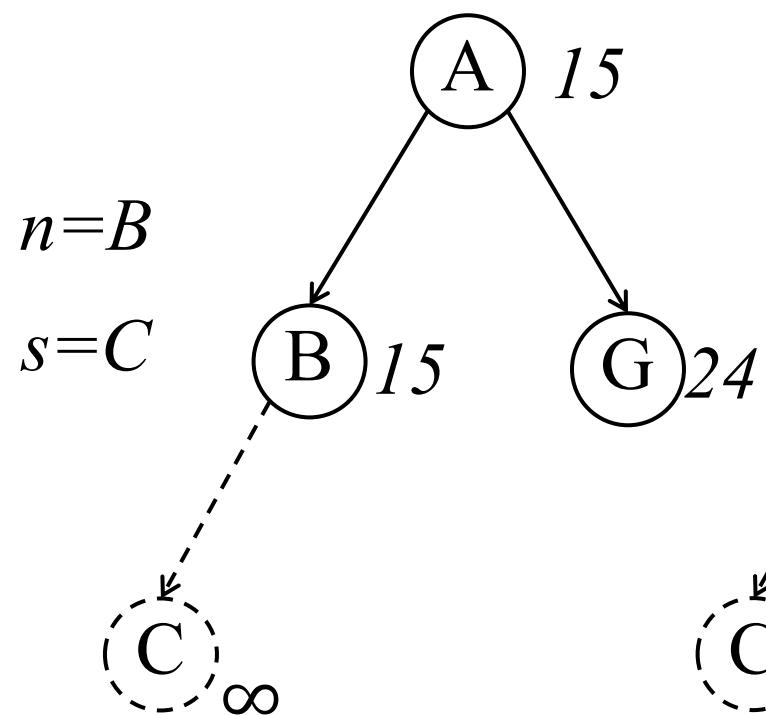
مثال * SMA با محدودیت ۳ خانه حافظه

فاز ششم

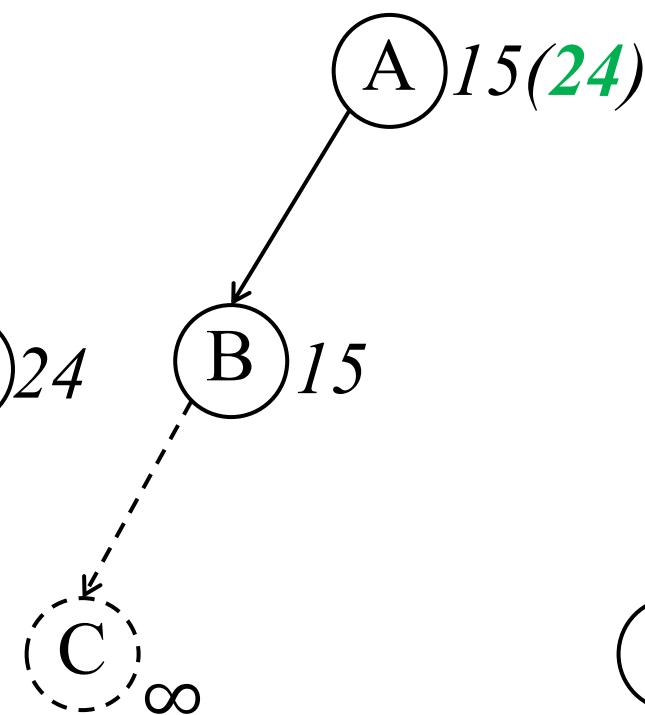
$$Q = \{A, G, B\}$$



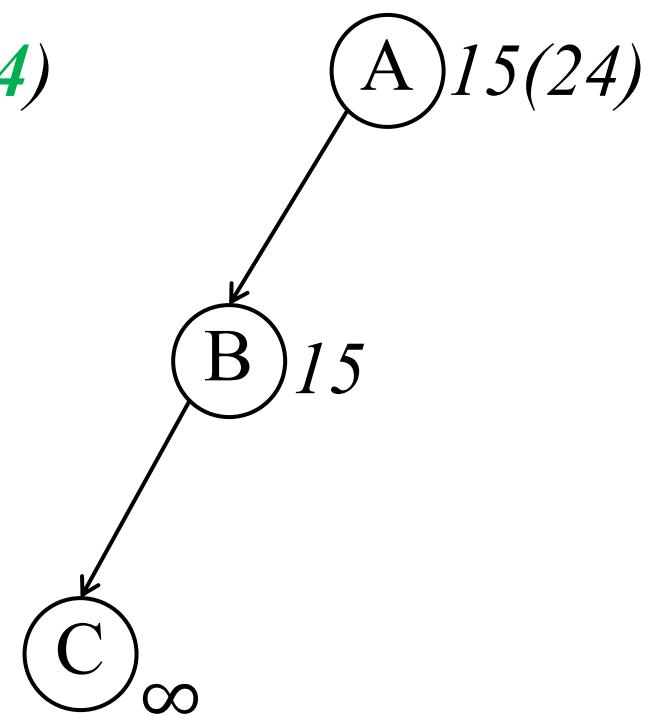
$$Q = \{A, G, B\}$$



$$Q = \{A, B\}$$



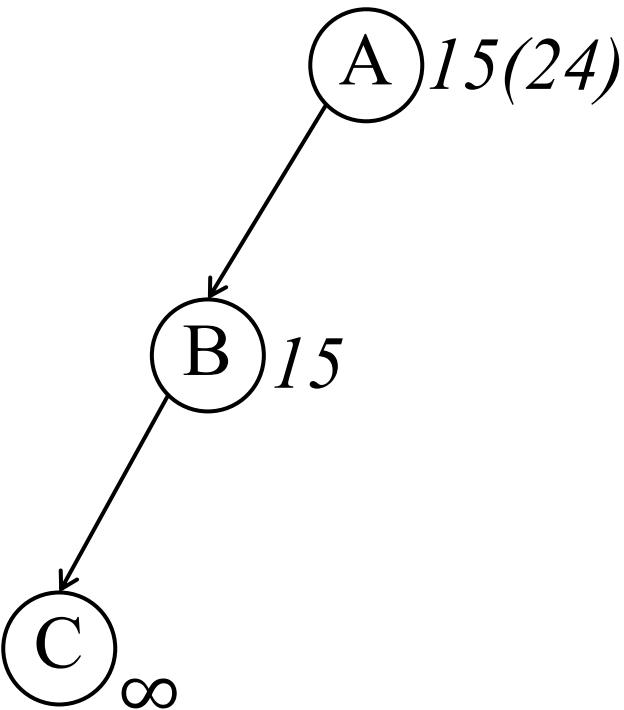
$$Q = \{A, B, C\}$$



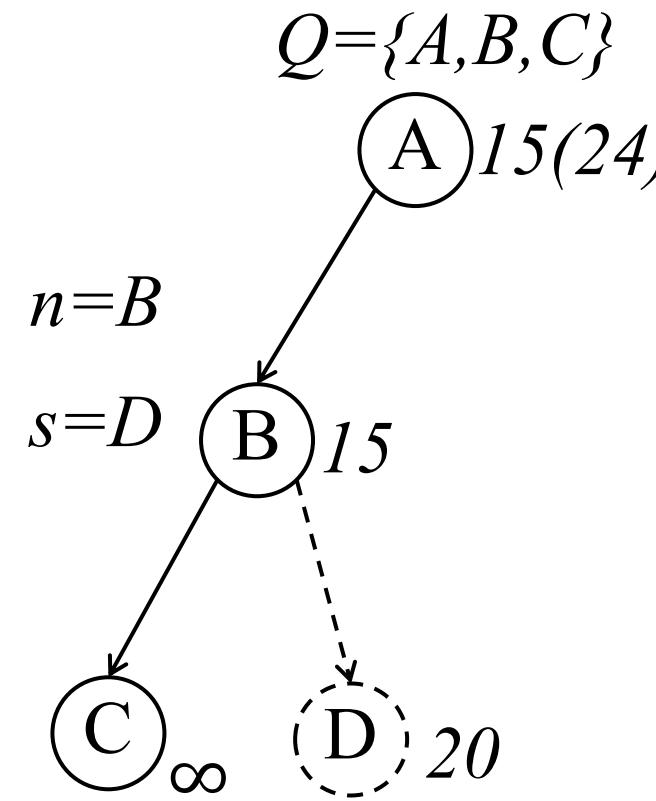
مثال * SMA با محدودیت ۳ خانه حافظه

فاز هفتم

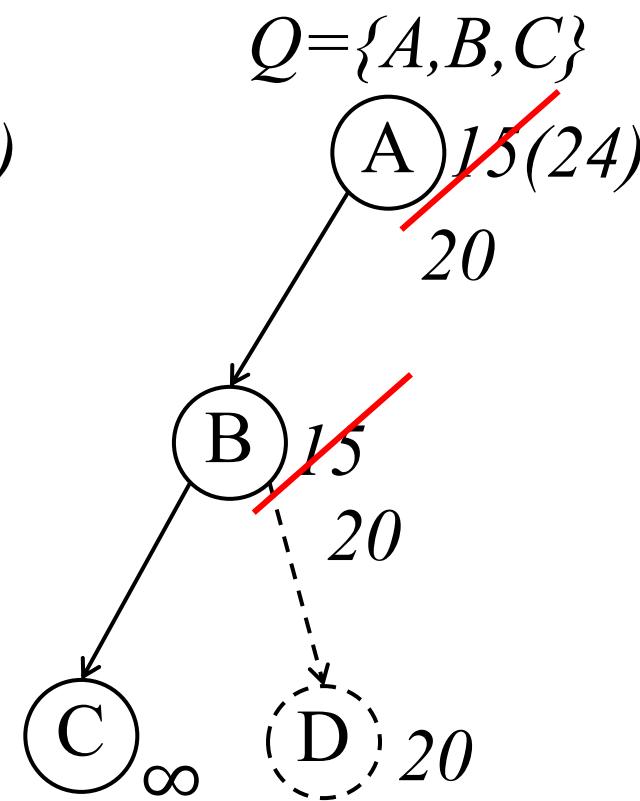
$$Q = \{A, B, C\}$$



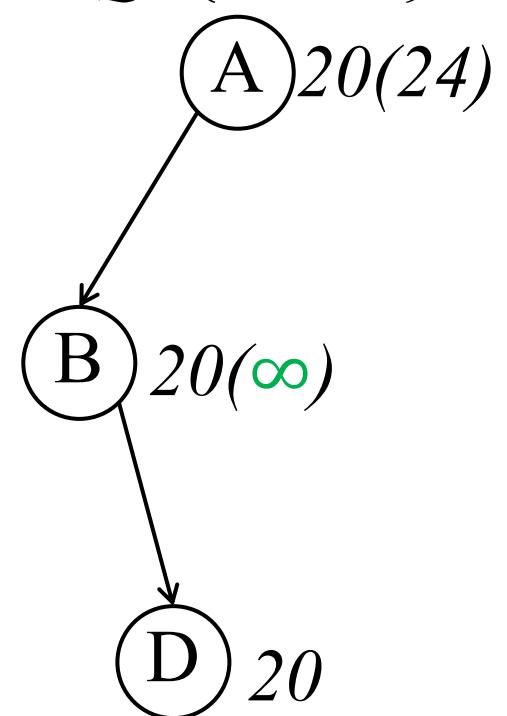
$$Q = \{A, B, C\}$$



$$Q = \{A, B, C\}$$



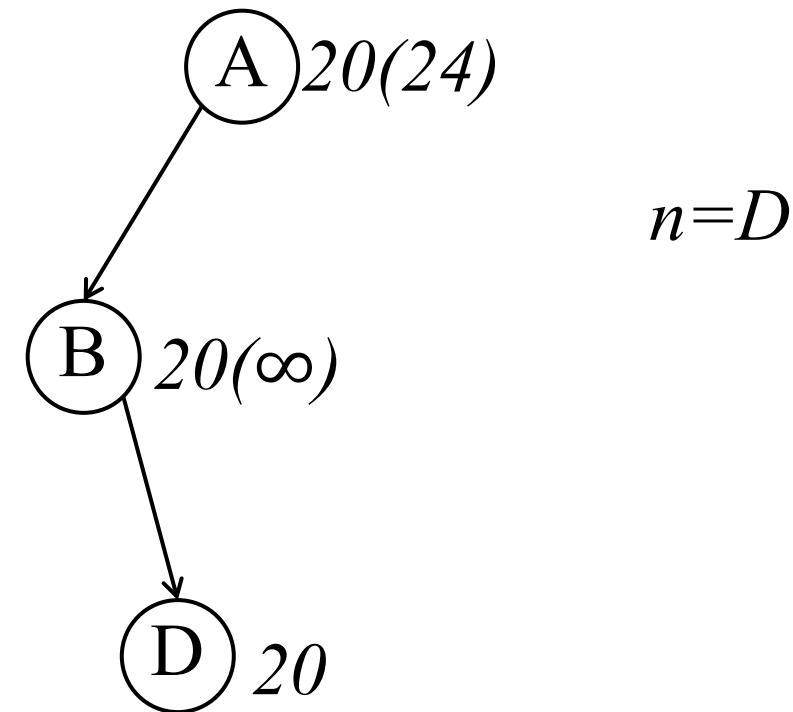
$$Q = \{A, B, D\}$$



مثال * SMA با محدودیت ۳ خانه حافظه

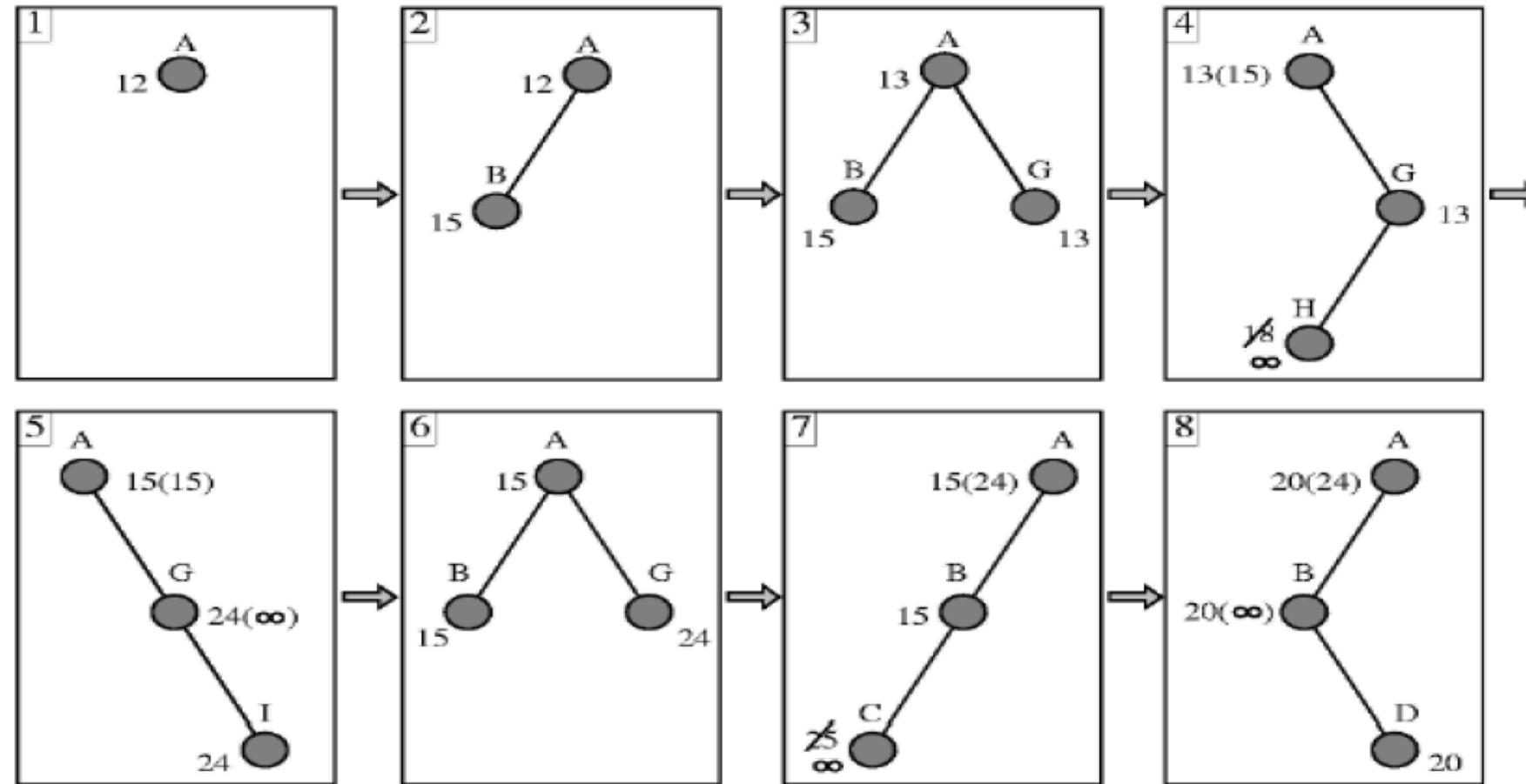
فاز هشتم

$$Q = \{A, B, D\}$$



$$n=D$$

مثال * SMA با محدودیت ۳ خانه حافظه



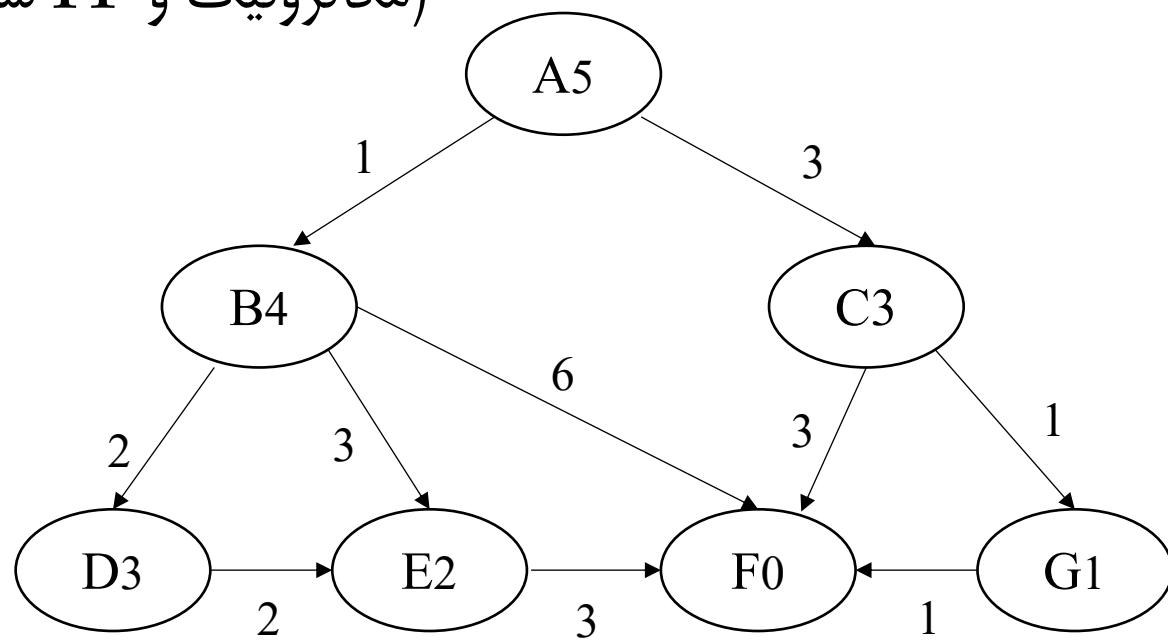
ارزیابی SMA*

- کامل؟
- بله، اگر عمق سطحی‌ترین گره هدف کمتر یا مساوی اندازه حافظه باشد.
- بهینگی؟
- بله، اگر عمق راه حل بهینه کمتر از اندازه حافظه (برحسب تعداد گره) باشد. در غیراین صورت بهترین راه حل قابل دستیابی با توجه به حافظه محدود را پیدا می‌کند.
- در عمل، ممکن است SMA* بهترین الگوریتم همه‌منظوره برای یافتن راه حل‌های بهینه باشد.
- در مسائل پیچیده ممکن است SMA* دائمًا از یک شاخه به شاخه دیگری برود و دوباره مجبور شود به شاخه اول برگرد (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)



ACF - ۱ ✓

ABF - ۲

ACGF - ۳

SMA* - ۴
این مسئله پیدا نمی‌کند.

تست ۵

(فناوری اطلاعات ۸۵)

کدام یک از جملات زیر صحیح است؟

- 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)

RBFS → BFS → SMA* → A* -۱

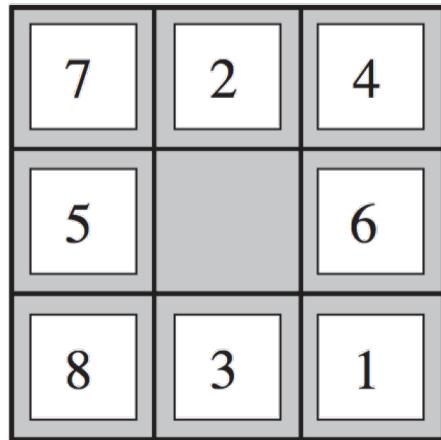
RBFS → BFS → A* → SMA* -۲

BFS → A* → RBFS → SMA* -۳

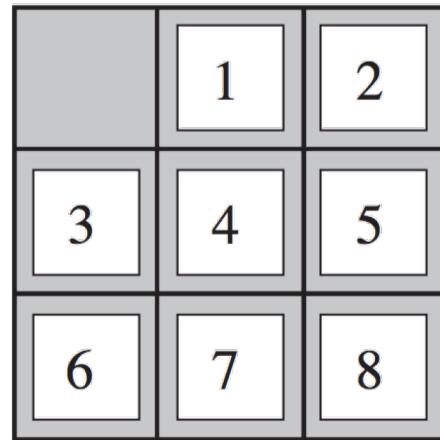
BFS → A* → SMA* → RBFS -۴ ✓

توابع هپوریستیک

فضای حالت مسئله پازل ۸ تایی



Start State



Goal State

- ضریب انشعاب: حدود ۳ (چرا؟)

به طور میانگین محاسبه شده است که پس از طی حدود ۲۲ گام به حالت هدف پازل ۸ تایی خواهیم رسید. (یعنی عمق ۲۲)

- فضای حالت بزرگ

تعداد حالات جستجوی درختی: $3^{22} \approx 3.1 \times 10^{10}$

تعداد حالات جستجوی گرافی: $9!/2 \approx 181440$

- در نتیجه یک تابع هیوریستیک می‌تواند فرایند جستجو را کاهش دهد.

توابع هیوریستیک قابل قبول پازل ۸ تایی

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- $h_1 =$ تعداد کاشی‌هایی که در جای درست خود قرار نگرفته‌اند.

- هر مربعی که سر جای خود نباشد حداقل یک خانه باید جابه‌جا شود تا به وضعیت به هدف برسد.

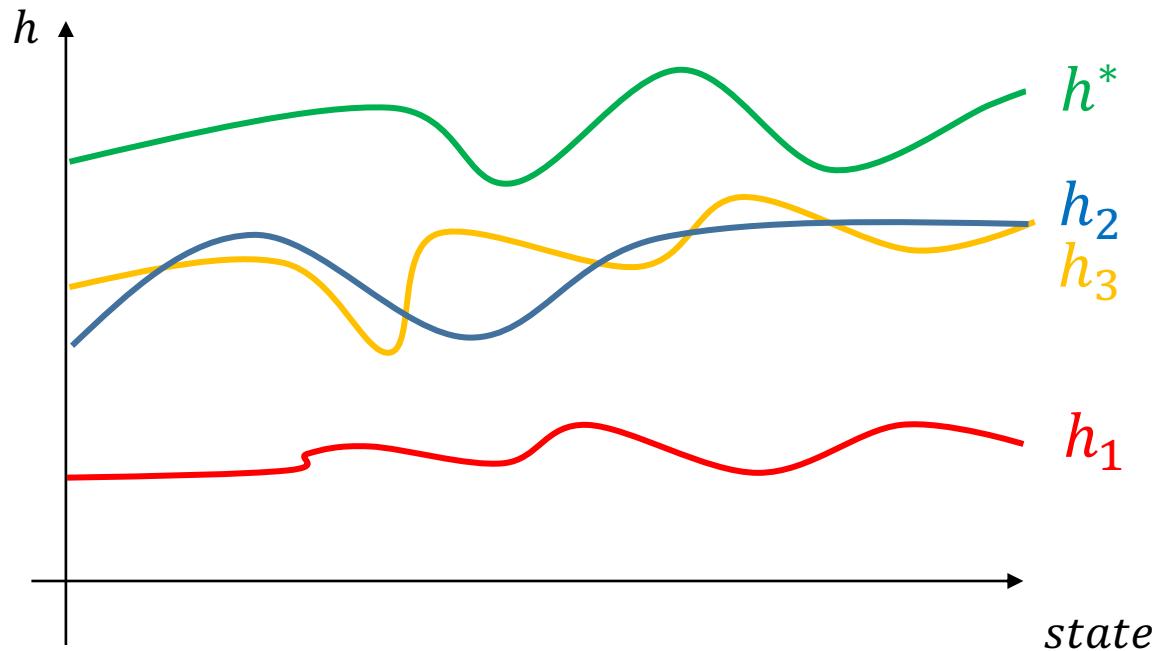
$$h_1(s) = 8$$

- $h_2 =$ 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.

$$h_2(s) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$$

کیفیت هیوریستیک



$$g(n) + h_1(n) \leq g(n) + h_2(n) \leq C^* \quad \text{نودهای بسط یافته:}$$

$$h_4(n) = \max(h_2(n), h_3(n))$$

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.

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

تأثیر دقت هیوریستیک بر روی کارایی

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

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

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

تأثیر دقیق هیوریستیک بر روی کارایی

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

d	Search Cost (nodes generated)			Effective Branching Factor		
	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

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

$$h(n) = h_1(n) + h_2(n) + h_3(n) \quad (۱)$$

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

$$h(n) = \max(\min(h_1(n), h_2(n), h_3(n)), h_1(n) \times h_2(n) \times h_3(n), h_1(n) + h_2(n) + h_3(n)) \quad (۳)$$

$$h(n) = \min(\max(h_1(n), h_2(n), h_3(n)), h_1(n) \times h_2(n) \times h_3(n), h_1(n) + h_2(n) + h_3(n)) \quad (۴) \quad \checkmark$$

مسائل تعدیل شده (Relaxed Problem)

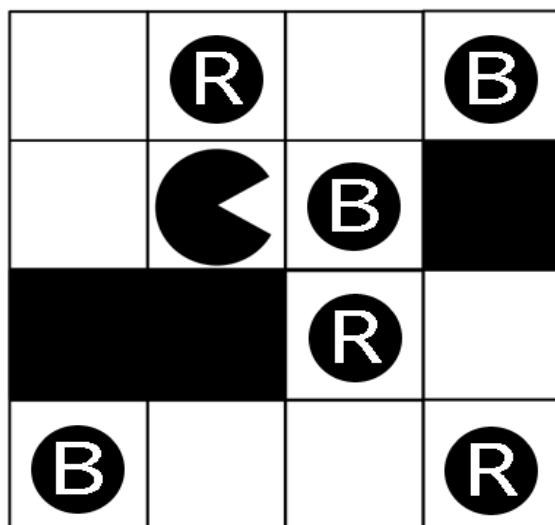
- اگر قوانین و محدودیت‌های اعمال یک مسئله را کم کنیم آن مسئله به یک مسئله‌ی راحت تبدیل می‌شود.
- راه حل بهینه برای مسئله‌ی تعدیل شده یک هیوریستیک قابل قبول برای مسئله‌ی اصلی است.
- از آنجا که هیوریستیک به دست آمده، یک هزینه‌ی دقیق برای مسئله تعدیل شده می‌باشد باید از نامساوی مثلث تبعیت کند و در نتیجه سازگار است.
- مسائل تولید شده باید بتوانند اساساً 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 تعدیل شده:
 - مربع A می‌تواند به مربع B منتقل شود. (صرف‌نظر از هر دو شرط)
 - مربع A می‌تواند به مربع B حرکت کند اگر مجاور آن باشد. (صرف‌نظر از خالی یا پر بودن آن)
 - مربع A می‌تواند به مربع B منتقل شود اگر B خالی باشد. (صرف‌نظر از مجاورت)
- هیوریستیک‌های قابل قبول برای مسئله‌ی اصلی (h_1 و h_2)، هزینه‌های مسیر بهینه برای مسائل تعدیل شده هستند.
- مورد اول: یک مربع می‌تواند به هر جایی انتقال یابد. $\leftarrow h_1(n)$
- مورد دوم: یک مربع می‌تواند به هر مربع همسایه منتقل شود. $\leftarrow h_2(n)$ هیوریستیک فاصله منهتن

تمرین

فرض کنید k قرص قرمز و k قرص آبی در محیطی $n \times m$ شامل چندین مانع وجود دارد. عامل می‌خواهد هر دو نوع قرص را امتحان کند. بنابراین، بازی هنگامی تمام می‌شود که حداقل یک قرص قرمز و یک قرص آبی توسط عامل خورده شود (ممکن است عامل از یک نوع قرص بیشتر از یکی بخورد). عامل می‌تواند به بالا، پایین، چپ و یا راست حرکت کند، اما امکان حرکت قطری ندارد. همچنین نمی‌تواند از سلول‌های شامل مانع رد شود. شکل رو به رو یک نمونه از چنین محیطی را نشان می‌دهد.



$$K = 3, N = 4, M = 4$$

- ه) قابل قبول بودن یا نبودن هر یک از هیوریستیک‌های زیر را با ذکر دلیل مشخص کنید.
- تعداد قرص‌های باقیمانده
 - کوچکترین فاصله منهتن تا هر یک از قرص‌های باقیمانده

تولید هیوریستیک‌ها

بانک‌های اطلاعاتی الگو (Pattern database)

- هزینه راه حل برای یک زیرمسئله از یک مسئله خاص
- دخیره هزینه راه حل‌های دقیق برای هر زیرمسئله ممکن
- قابل قبول؟
- هزینه راه حل بهینه برای این زیرمسئله یک حد پایین برای هزینه مسئله کامل است.

3	1	2
4	7	5
6		8

3	1	2
4	5	
7	6	8

3	1	2
*	*	*
*	3	1

Start state

	1	2
3	4	*
*	*	*

Goal state

	1	2
3	4	5
6	7	8

1		2
3	4	5
6	7	8

3	1	2
	4	5
6	7	8

1	4	2
3		5
6	7	8

1	2	
3	4	5
6	7	8

3	1	2
6	4	5
	7	8

3	1	2
4		5
6	7	8

1	4	2
3	5	
6	7	8

1	4	2
3	7	5
6		8

1	4	2
	3	5
6	7	8

1	2	5
3	4	
6	7	8

3	1	2
6	4	5
7		8

3		2
4	1	5
6	7	8

3	1	2
4	7	5
6		8

3	1	2
4	5	
6	7	8

تولید هیوریستیک‌ها

- با در نظر گرفتن زیرمسائل مختلف و در حقیقت داشتن بانک‌های اطلاعاتی الگوی مختلف، می‌توان هیوریستیک‌های حاصل را با برداشتن مقدار بیشینه با هم ترکیب کرد.
- پازل ۱۵ تایی: ۱۰۰۰ برابر تعداد نود کمتری نسبت به استفاده از h^2 تولید می‌کند.
- آیا جمع زدن هیوریستیک‌های زیرمسائل مختلف بهجای ماکریم گرفتن منجر به یک هیوریستیک قابل قبول می‌شود؟
- برای مثال، هیوریستیک بانک اطلاعاتی ۱-۲-۳-۴ و هیوریستیک بانک اطلاعاتی ۵-۶-۷-۸
- خیر، زیرا برای یک حالت خاص تقریباً به طور قطع چند حرکت مشترک دارند.

تولید هیوریستیک‌ها

- اگر به جای 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) به دست آورده‌ایم.
- در این حالت، مجموع هیوریستیک‌ها یک هیوریستیک قابل قبول خواهد بود.
- پازل ۱۵ تایی: ۱۰۰۰ برابر تعداد نود کمتری نسبت به استفاده از h_2 تولید می‌کند.
- پازل ۲۴ تایی: ۱۰۰۰۰۰ برابر تعداد نود کمتری نسبت به استفاده از h_2 تولید می‌کند.

تولید هیوریستیک‌ها

یادگیری هیوریستیک‌ها از تجربه

- یادگیری $h(n)$ از نمونه‌هایی که قبلاً به‌طور بهینه حل شده‌اند. (پیش‌بینی هزینه‌ی راه حل 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 حل شده‌ی زیادی از مسئله پازل ۸تاوی
 - $X_1(n)$: 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 وزن دار)

$$h(n) = w_1 X_1(n) + w_2 X_2(n)$$

یادگیری هیوریستیک‌ها از تجربه

3	1	2
6	4	5
	7	8

1		2
3	4	5
6	7	8

