



دانشگاه علم و صنعت
دانشکده مهندسی کامپیوتر

جستجوی آگاهانه

«هوش مصنوعی: رهیافتی نوین»، فصل ۳

مدرس: آرش عبدی هجراندوست

نیمسال دوم ۱۴۰۱-۱۴۰۲

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^*



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؟

$$f(n)=g(n) \quad \blacklozenge$$

❖ جستجوی اول سطح؟

$$f(n)=\text{depth}(n) \quad \blacklozenge$$

❖ جستجوی اول عمق؟

$$f(n)=-\text{depth}(n) \quad \text{یا} \quad f(n)=1/\text{depth}(n) \quad \blacklozenge$$

تابع هیوریستیک

- ❖ وارد کردن دانش خاص مسئله در جستجو
- ❖ اطلاعاتی بیشتر از تعریف مسئله به منظور رسیدن هرچه سریع تر به یک راه حل بهینه
- ❖ تابع هیوریستیک می تواند به عنوان جزئی از تابع ارزیاب $f(n)$ تعریف شود.
- ❖ $h(n)$ = هزینه ی تخمینی ارزان ترین مسیر از وضعیت موجود در گره ی n به یک وضعیت هدف
- ❖ تنها وابسته به وضعیت n است نه مسیر از ریشه تا n
- ❖ اگر n یک وضعیت هدف باشد آن گاه $h(n)=0$
- ❖ $h(n) \geq 0$
- ❖ تابع هیوریستیک می تواند توسط یک قانون سرانگشتی، ساده سازی مسئله و حدس های تجربی به دست آمده باشد که باعث محدود کردن یا کاهش دادن جستجو در دامنه هایی می شود که پیچیده و مشکل هستند.

جستجوی اول-بهترین حریصانه

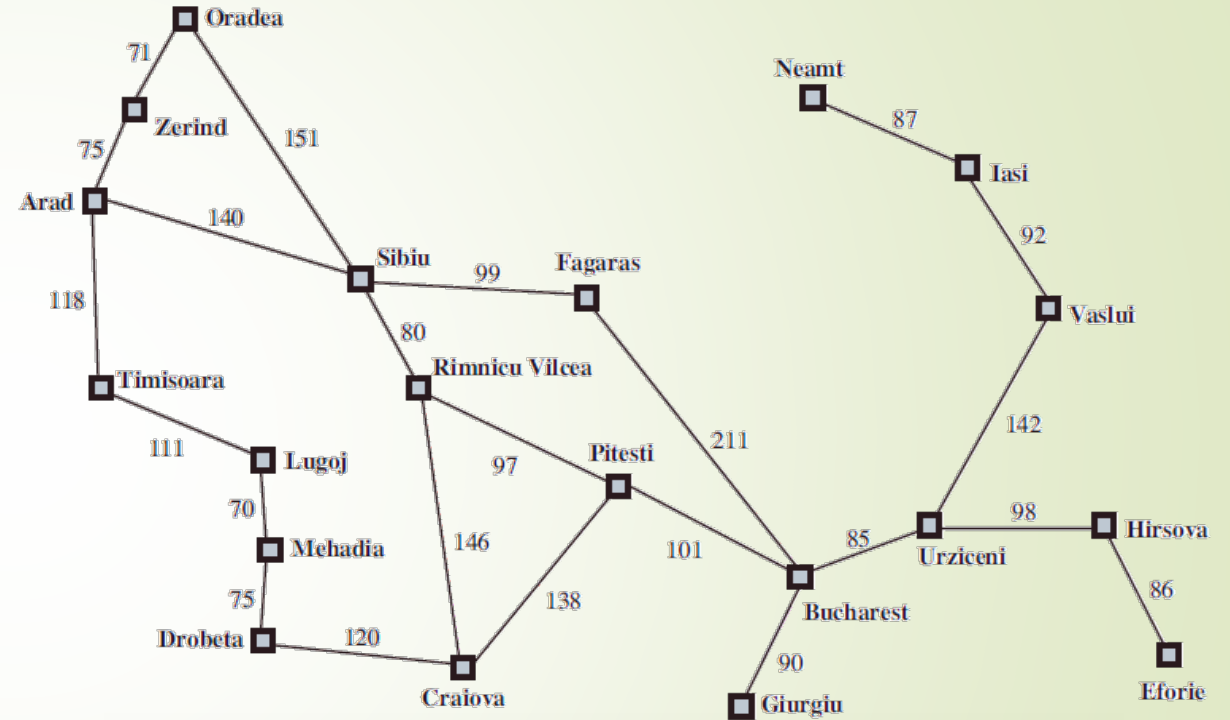
❖ نودی را بسط می‌دهد که به نظر می‌رسد به هدف نزدیک‌تر است.

❖ تابع ارزیاب: $f(n)=h(n)$

❖ برای مثال:

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

❖ وابسته به وضعیت هدف است برای مثال اگر هدف بخارست باشد جدول مقابل را خواهیم داشت.



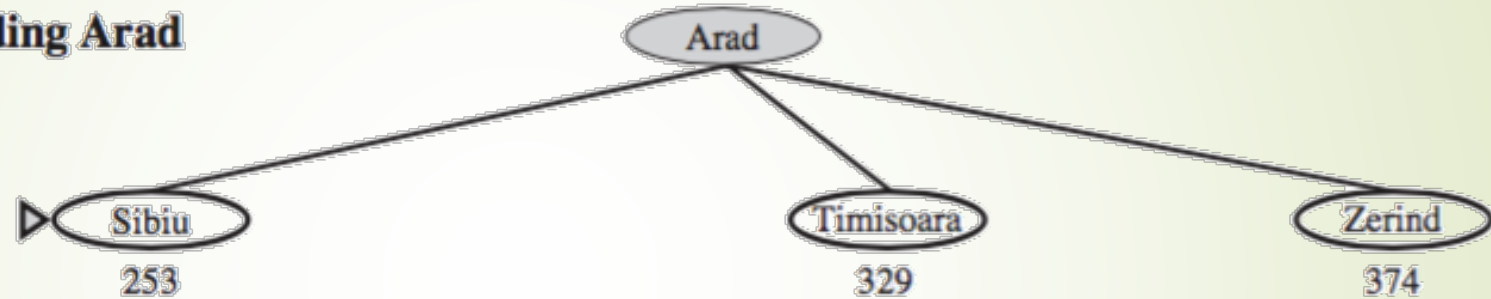
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

جستجوی حریصانه – مثال

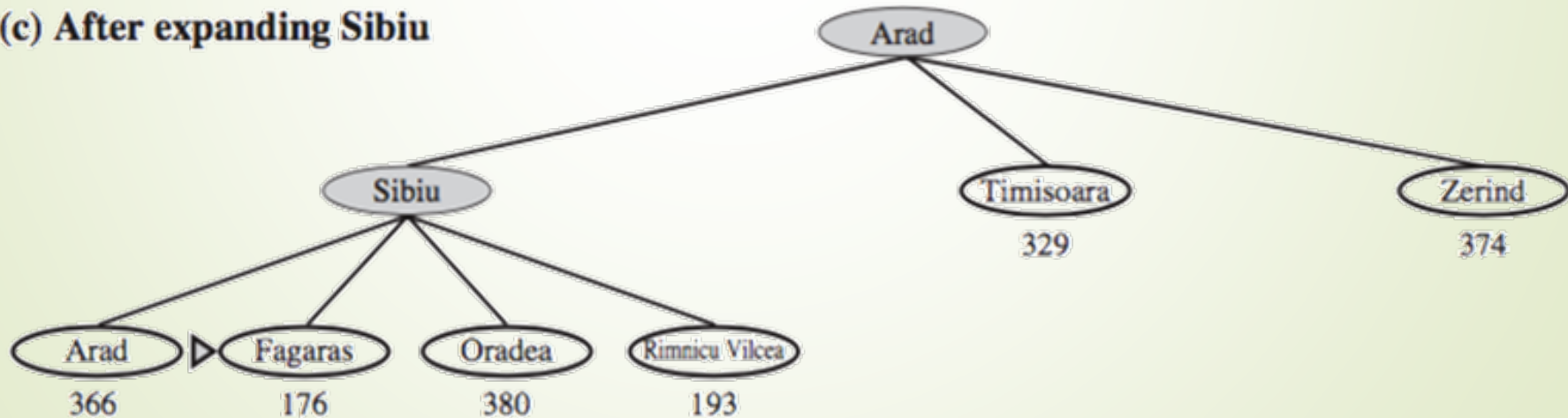
(a) The initial state



(b) After expanding Arad

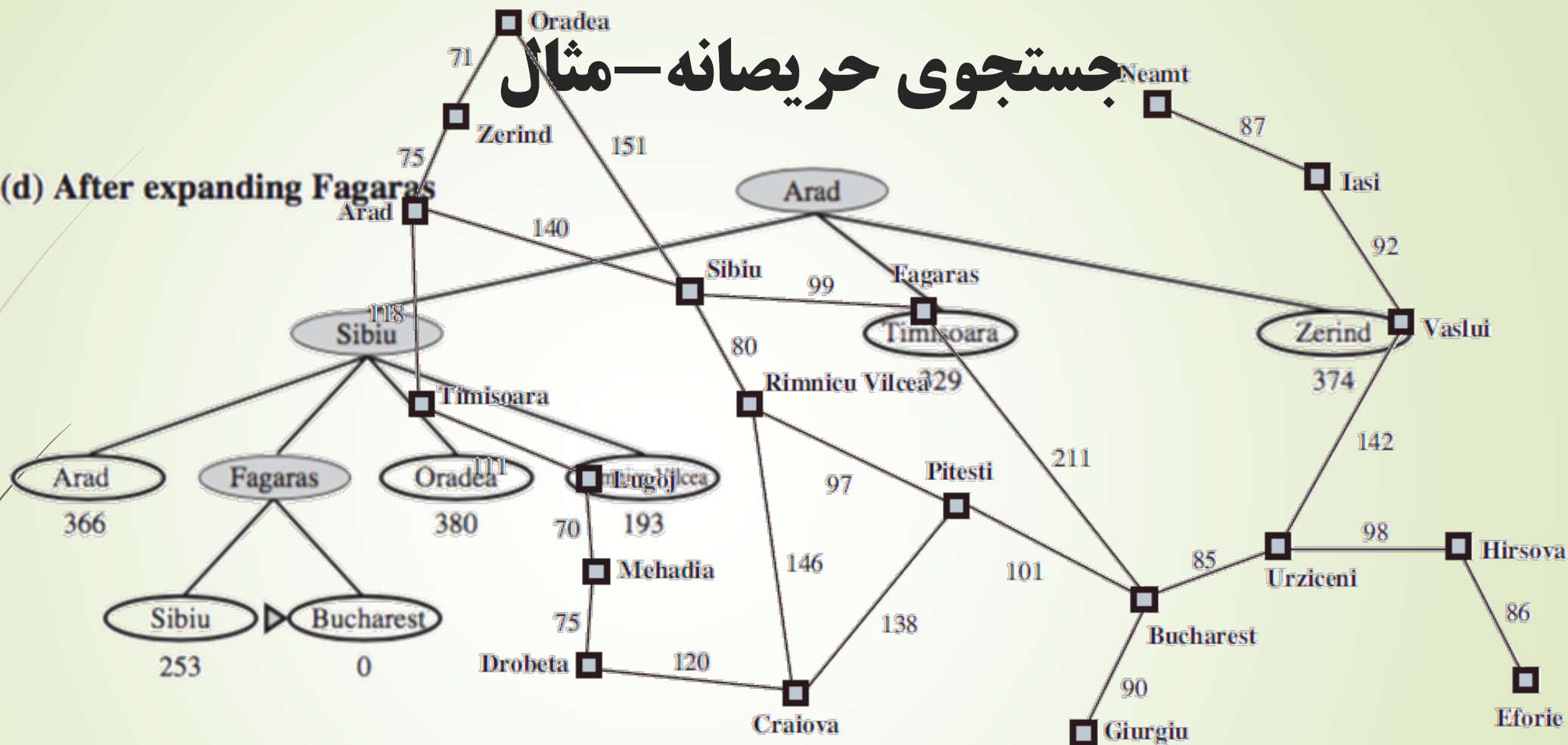


(c) After expanding Sibiu



جستجوی حریصانه – مثال

(d) After expanding Fagaras

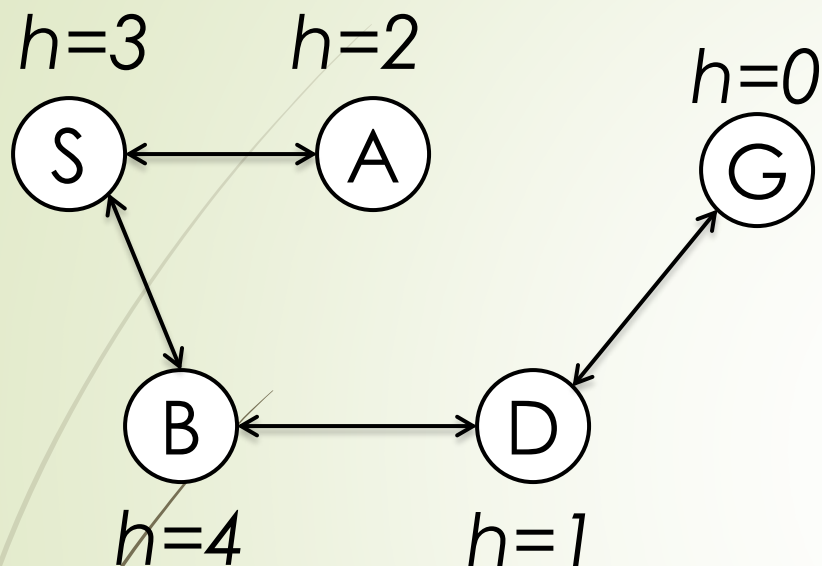


❖ به هدف رسید!!

❖ 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.

❖ 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: $O(b^m)$

❖ و با در نظر گرفتن هزینه مرتب نگه داشتن frontier بر حسب تابع h ؟

❖ میتواند مشابه با جستجوی اول عمق عمل کند زیرا ترجیح می‌دهد یک مسیر را در تمام طول راه تا هدف دنبال کند (اگر h اقتضا کند)، ولی اگر به بن‌بست برسد به عقب برمی‌گردد.

❖ اگر در مسیرهای عمیق غلط، هزینه تخمینی تا هدف نزدیک به صفر باشد، همواره این مسیرهای غلط تا برگ (بن بست) دنبال می‌شوند.

❖ 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؟

$O(b^m)$

❖ همه‌ی گره‌ها را در حافظه نگه می‌دارد.

جستجوی A^*

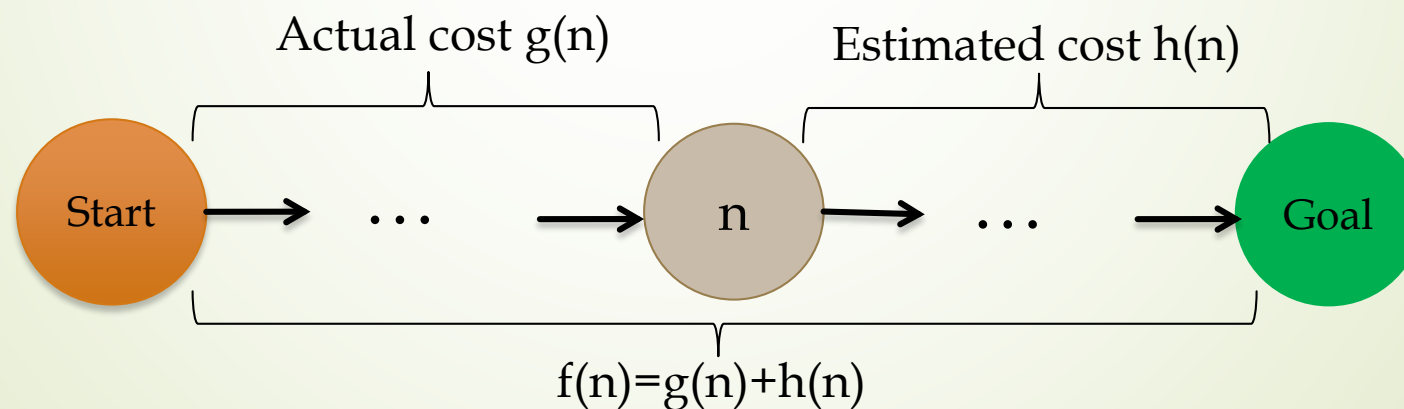
❖ شناخته شده ترین شکل جستجوی اول-بهترین است.

❖ تابع ارزیاب: $f(n) = g(n) + h(n)$

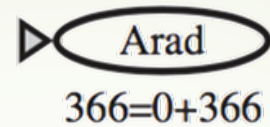
❖ $g(n)$ هزینه واقعی مسیر از شروع تا گره n

❖ $h(n)$ هزینه تخمینی 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



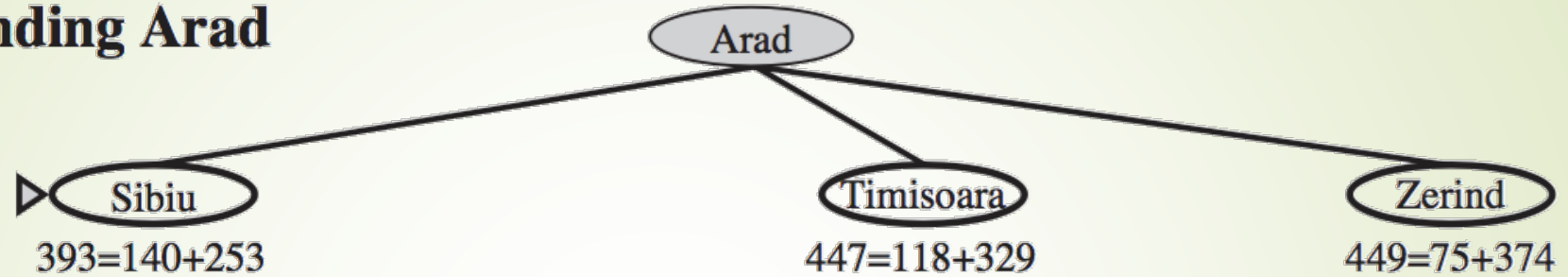
(a) The initial state

• یافتن Bucharest با شروع از Arad

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

جستجوی A^* – مثال

(b) After expanding Arad



- بسط Arad و تعیین $f(n)$ برای هر گره

- $f(\text{Sibiu}) = c(\text{Arad}, \text{Sibiu}) + h(\text{Sibiu}) = 140 + 253 = 393$

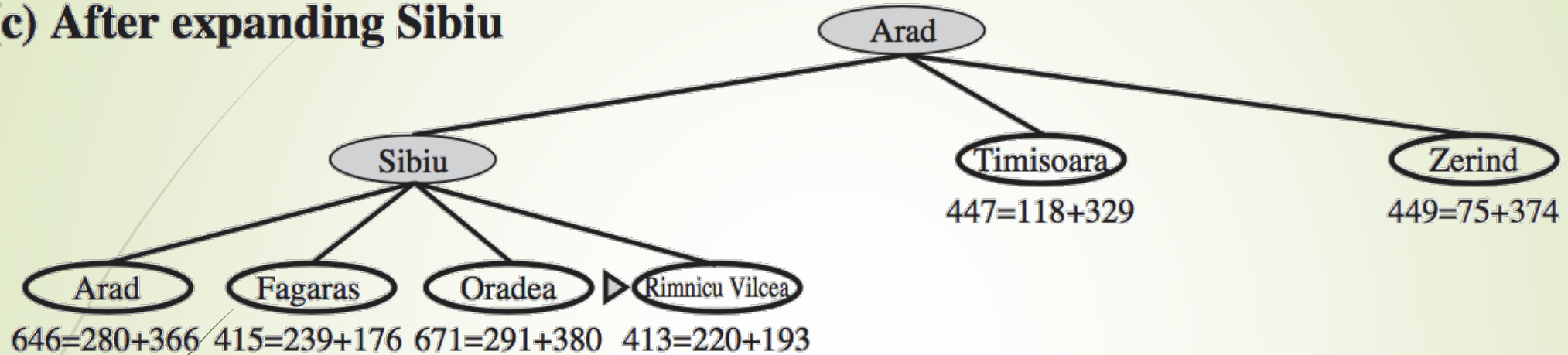
- $f(\text{Timisoara}) = c(\text{Arad}, \text{Timisoara}) + h(\text{Timisoara}) = 118 + 329 = 447$

- $f(\text{Zerind}) = c(\text{Arad}, \text{Zerind}) + h(\text{Zerind}) = 75 + 374 = 449$

- بهترین انتخاب Sibiu است.

جستجوی A^* – مثال

(c) After expanding Sibiu



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

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

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

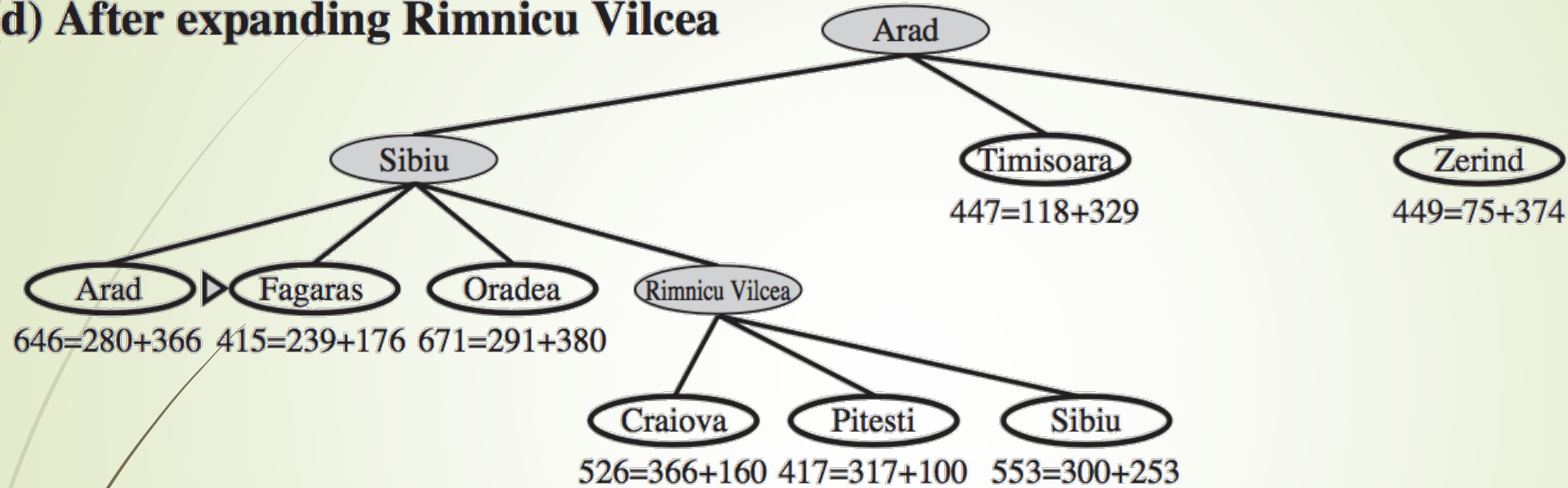
• $f(\text{Oradea}) = c(\text{Sibiu}, \text{Oradea}) + h(\text{Oradea}) = 291 + 380 = 671$

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

• بهترین انتخاب Rimnicu Vilcea است.

جستجوی A^* – مثال

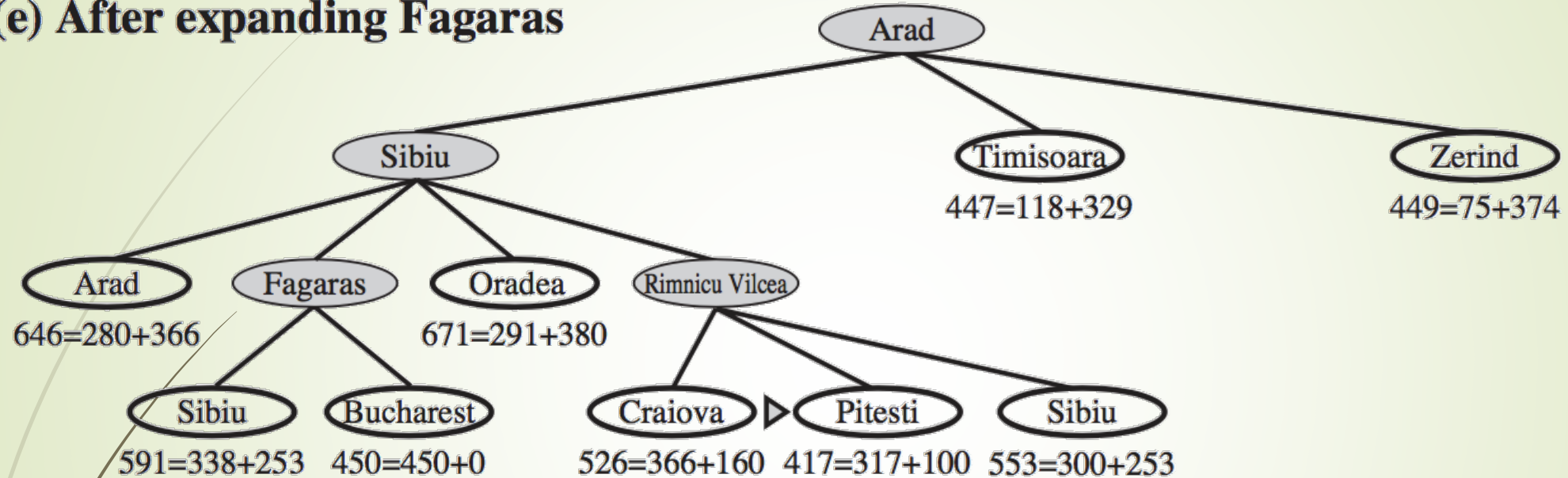
(d) After expanding Rimnicu Vilcea



- 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

(e) After expanding Fagaras



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

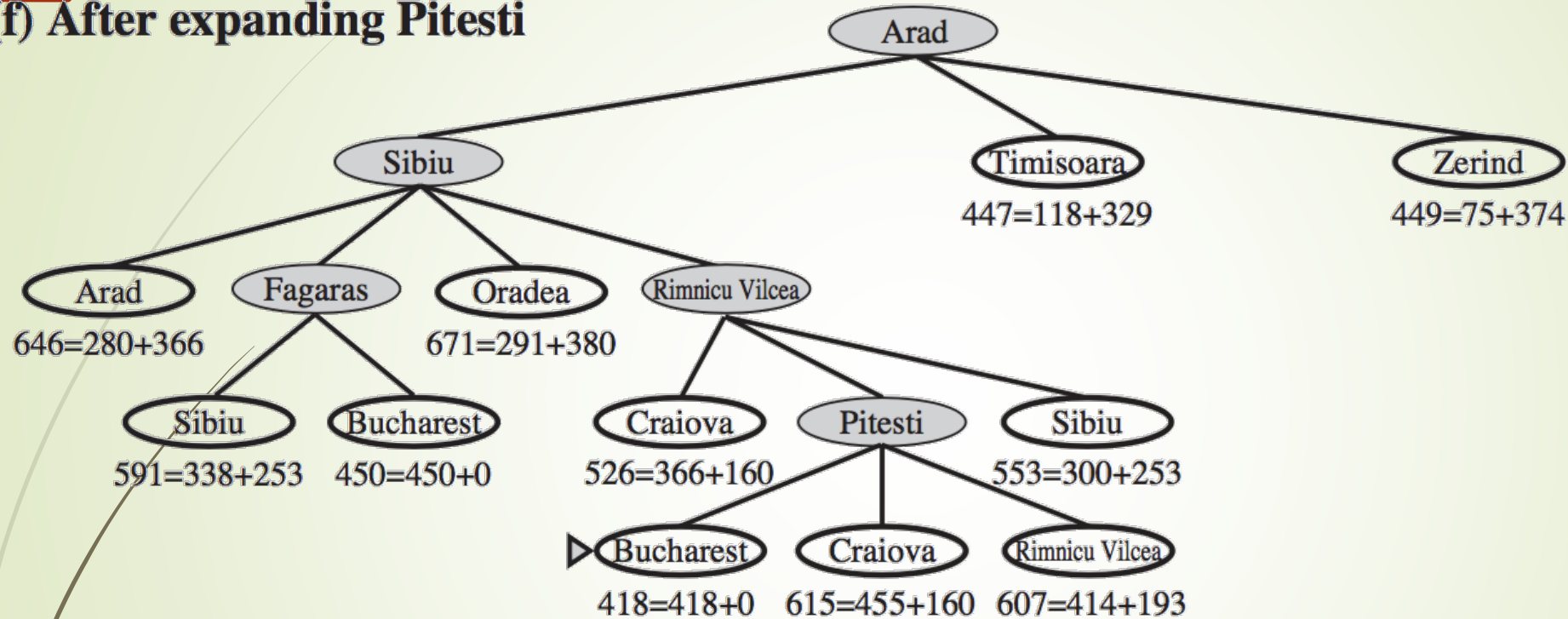
• $f(\text{Sibiu}) = c(\text{Fagaras}, \text{Sibiu}) + h(\text{Sibiu}) = 338 + 253 = 591$

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

• بهترین انتخاب Pitesti است!!

جستجوی A^* - مثال

(f) After expanding Pitesti



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

- $f(\text{Bucharest}) = c(\text{Pitesti}, \text{Bucharest}) + h(\text{Bucharest}) = 418 + 0 = 418$

- بهترین انتخاب Bucharest است!!

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

شرط بهینگی جستجوی درختی A^*

❖ قابل قبول بودن Admissibility

❖ 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

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

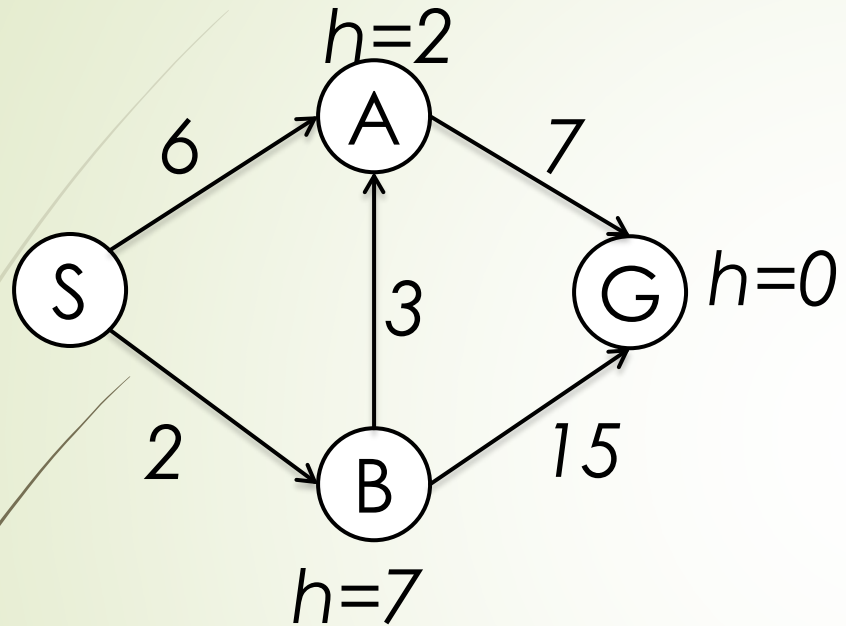
که $h^*(n)$ هزینه واقعی ارزان‌ترین مسیر به وضعیت هدف از گره n است.

❖ برای مثال: $h_{SLD}(n) \geq$ "فاصله واقعی جاده"

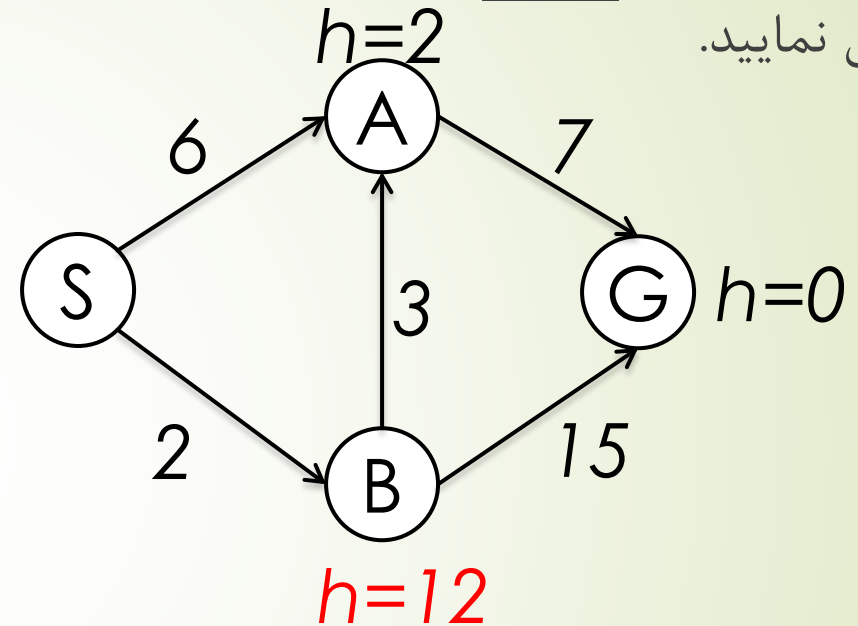
❖ می‌توان گفت هر تابع هیوریستیک قابل قبول، هزینه هر گره تا هدف را به‌طور خوش‌بینانه یا optimistic تخمین می‌زند.

قابل قبول بودن در مقابل غیر قابل 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

سوال: آیا مانند جستجوی هزینه یکنواخت، وقتی نودی برای بسط انتخاب می‌شود، مسیر بهینه تا آن نود پیدا شده است؟
خیر - پس باید اجازه داد مسیر اکتشاف شده، دوباره در مجموعه مرزی وارد شود (مقدارش بروز شود) و دوباره اکتشاف شود.

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 نیمه بهینه G_2 در frontier قرار دارد و گره n نودی بسط نیافته باشد که در مسیر هدف بهینه G قرار دارد.

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

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

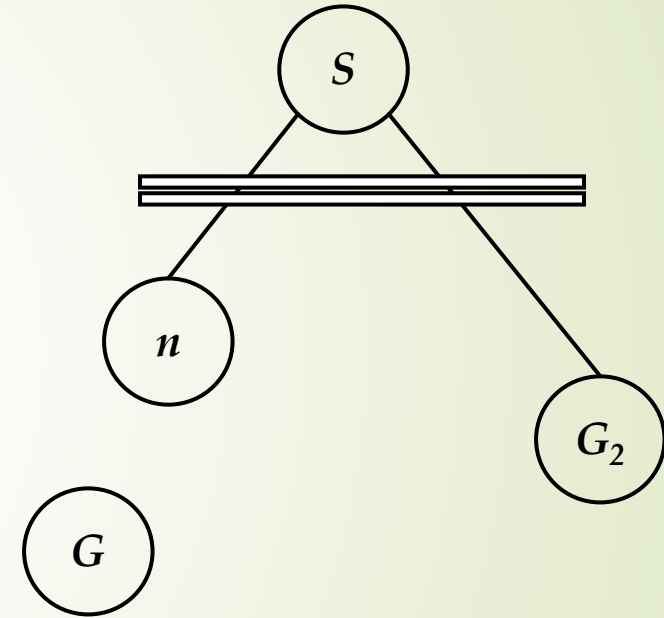
$$3) G_2 \text{ is suboptimal} \rightarrow g(G_2) > g(G)$$

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

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

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

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



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

❖ هزینه واقعی G_2 (مسیر غیر بهینه) بیشتر از هزینه واقعی G (مسیر بهینه) است.

❖ هزینه تخمینی در هر نقطه از مسیر بهینه، کمتر مساوی هزینه واقعی G است (لذا کمتر از هزینه واقعی G_2 است).

❖ یکی از نودهای مجموعه مرزی در هر لحظه، حتما بخشی از مسیر بهینه است.

❖ چرا باید به جای آن نود، G_2 انتخاب شود؟

آیا $g(n)$ هزینه مسیر بهینه تا n

است یا مسیر غیر بهینه؟

در درخت جستجو، مسیرهای متفاوت به یک

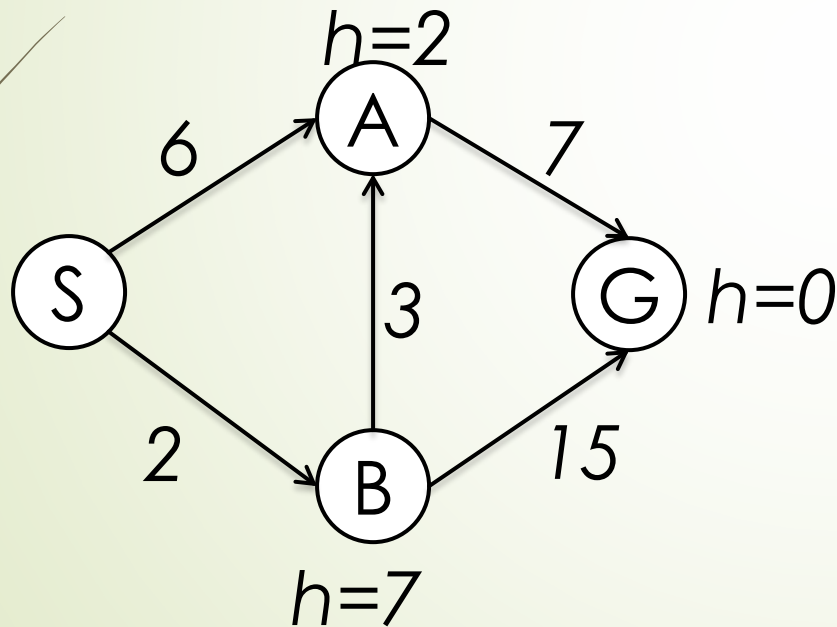
حالت یکسان، در نودهای متفاوت قرار دارند

شرط بهینگی جستجوی گرافی A^*

❖ آیا قابل قبول بودن هیوریستیک، بهینگی جستجوی گرافی A^* را تضمین می کند؟

❖ باید توجه کرد که **گراف** جستجو، مسیر [بهینه] به یک حالت تکراری را کنار می گذارد.

❖ برای مثال در مورد گراف زیر مسیر نیمه بهینه S-A-G توسط جستجوی گرافی به دست خواهد آمد.



شرط بهینگی جستجوی گرافی A^*

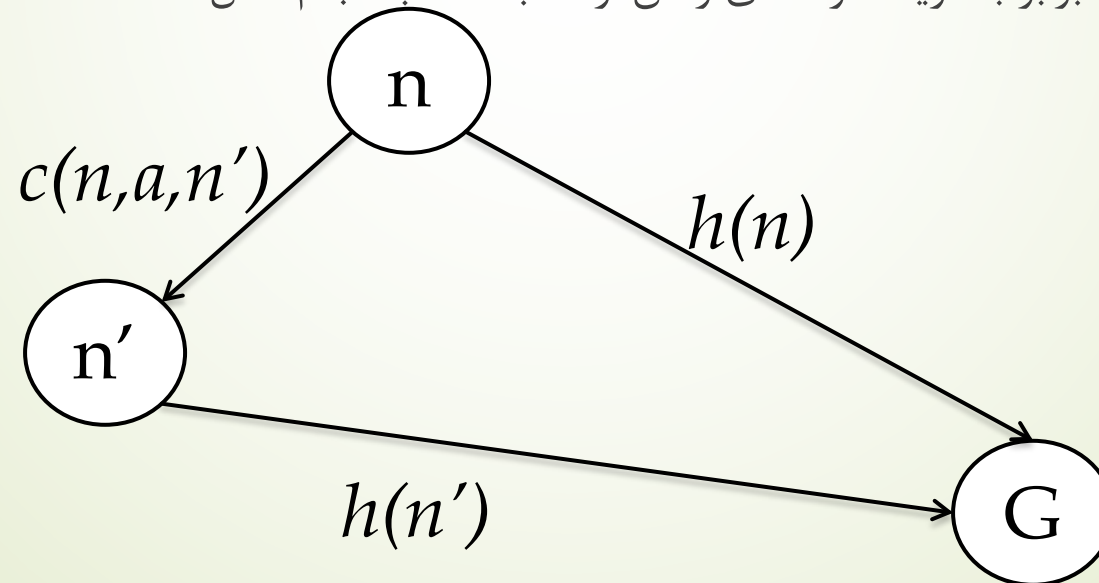
❖ سازگاری Consistent

❖ 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:

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

❖

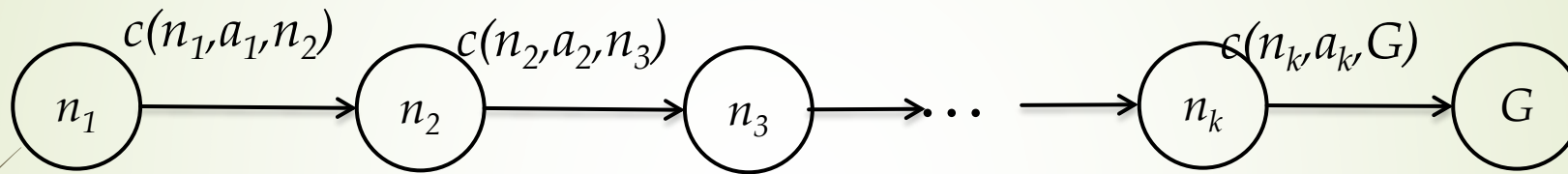
که در آن $c(n, a, n')$ برابر با هزینه مرحله‌ای رفتن از n به n' با انجام عمل a است.



ارتباط قابل قبول بودن و سازگار بودن

❖ سازگاری ← قابل قبول بودن

❖ همه‌ی توابع هیوریستیک سازگار قابل قبول هستند.

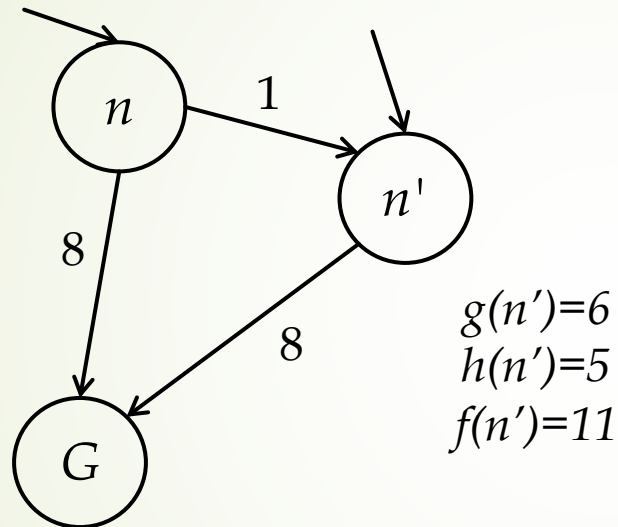


$$\begin{aligned} h(n_1) &\leq c(n_1, a_1, n_2) + h(n_2) \\ &\leq c(n_1, a_1, n_2) + c(n_2, a_2, n_3) + h(n_3) \\ &\dots \\ &\leq c(n_1, a_1, n_2) + \dots + c(n_k, a_k, G) + h(G) \\ &\leq c(n_1, a_1, n_2) + \dots + c(n_k, a_k, G) \\ &\leq \text{cost of each path from } n_1 \text{ to goal} \end{aligned}$$

ارتباط قابل قبول بودن و سازگار بودن

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

$$\begin{aligned}g(n) &= 5 \\ h(n) &= 7 \\ f(n) &= 12\end{aligned}$$



$$\begin{aligned}g(n') &= 6 \\ h(n') &= 5 \\ f(n') &= 11\end{aligned}$$

$$\begin{aligned}c(n, a, n') &= 1 \\ h(n) &= 7 \\ h(n') &= 5 \\ h(n) &\not\leq h(n') + c(n, a, n')\end{aligned}$$

❖ مقدار f برای هیوریستیک‌های قابل قبول ممکن است در طول مسیر کاهش یابد.

❖ به دلیل خوش بینی بیش از حد هیوریستیک‌های طول مسیر

❖ اکثر هیوریستیک‌های قابل قبول در کاربردهای عملی سازگار نیز هستند.

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 آن گاه 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، راه حل بهینه تا آن گره پیدا شده است.
- ❖ اثبات با استفاده از برهان خلف: فرض کنید هنگامی که گره n برای بسط انتخاب می شود، مسیر بهینه از ریشه تا n به دست نیامده باشد آنگاه باید بتوان از طریق گره دیگری مانند n' که در مجموعه frontier فعلی قرار دارد با یک مسیر بهینه به حالت موجود در n رسید. علاوه بر این براساس لم ۱، $f(n') \leq f(n)$ و بنابراین n' باید زودتر انتخاب می شد.
- ❖ اولین گرهی هدفی که برای بسط انتخاب می شود باید یک راه حل بهینه باشد (برای گره های هدف $h=0$ بوده و مقدار f برابر با هزینه ی واقعی مسیر می باشد).
- ❖ دنباله ی گره های بسط داده شده توسط A^* با استفاده از جستجوی گرافی به ترتیب غیرنزولی $f(n)$ می باشد.

اثبات بهینگی جستجوی گرافی A^*

❖ تشریح بیشتر لم ۲:

❖ لم ۲: اگر A^* گره n را برای بسط انتخاب کند، راه حل بهینه تا آن گره پیدا شده است.

❖ گره n برای بسط انتخاب شده است. **فرض** کنیم از طریق گره n' با مسیر بهینه تر بتوان به n رسید.

❖ وقتی از مسیر متفاوت به حالت یکسان میرسیم، $g(n)$ و در نتیجه $f(n)$ های متفاوت داریم.

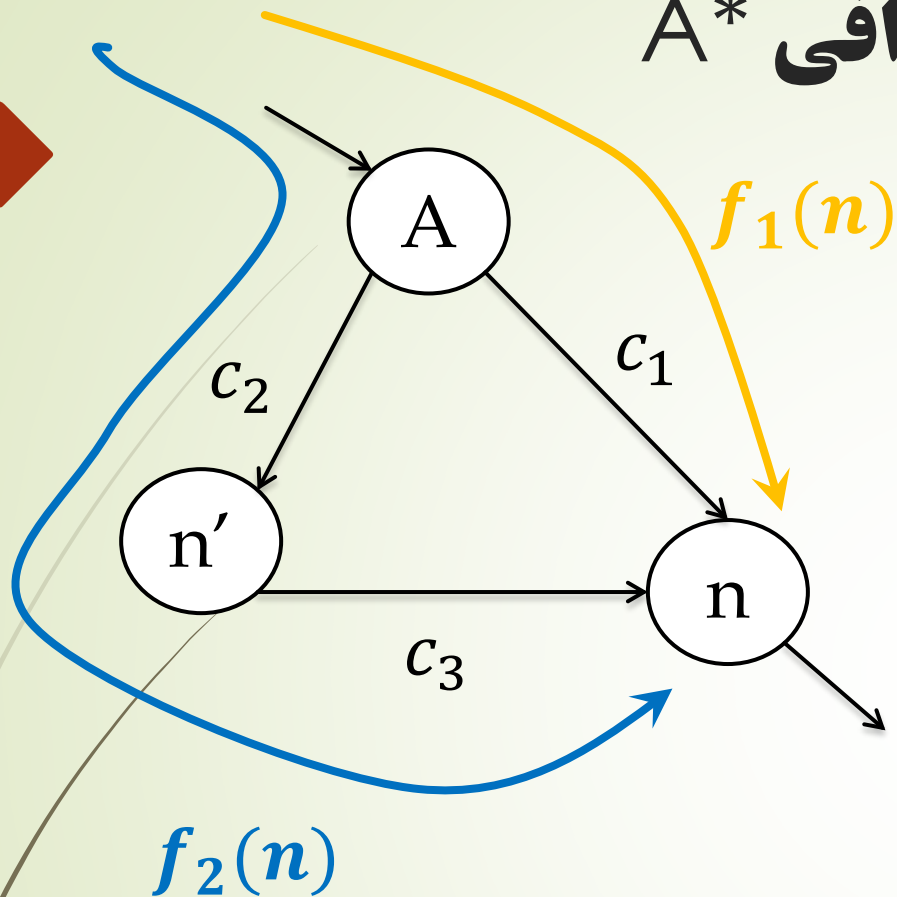
$$f_1(n) = g(A) + c_1 + h(n)$$

$$f_2(n) = g(A) + c_2 + c_3 + h(n)$$

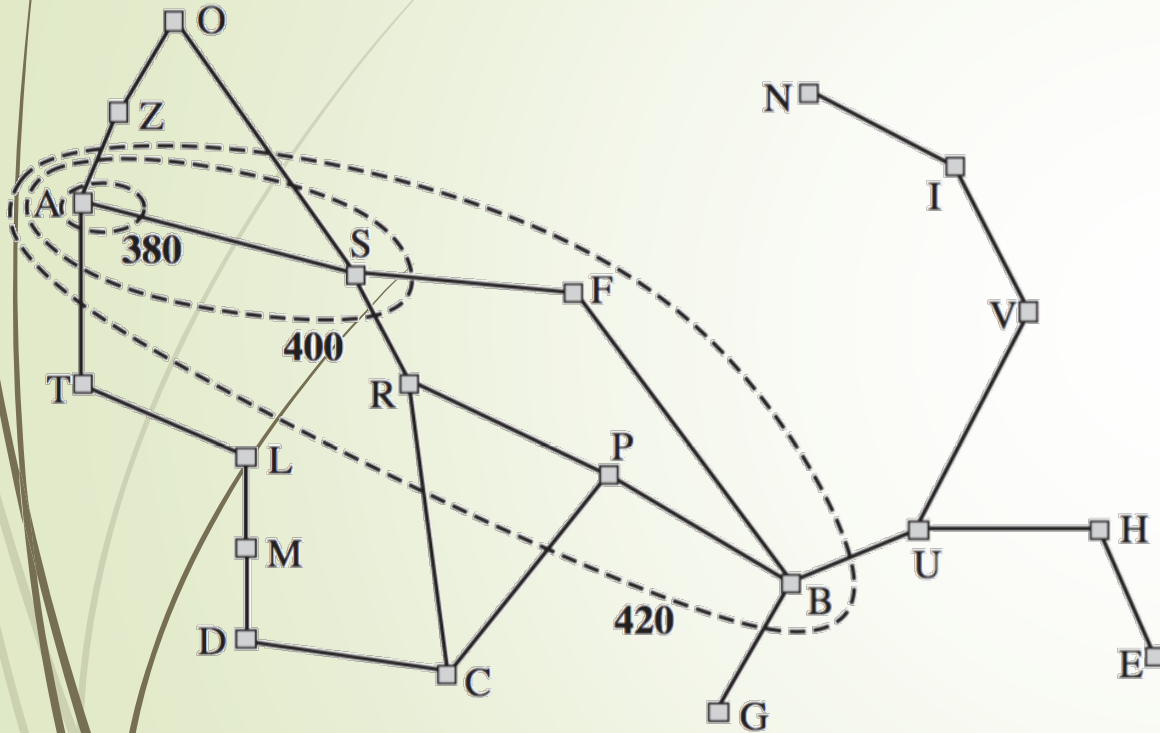
فرض : $c_2 + c_3 < c_1 \Rightarrow f_2(n) < f_1(n)$

طبق سازگاری : $f(n') \leq f_2(n) \Rightarrow f(n') < f_1(n)$

❖ بنابراین n' باید زودتر انتخاب می شد



کانتورهای A^* – Contours



❖ A^* گره‌ها را به ترتیب افزایش مقدار f بسط می‌دهد.

❖ کانتورهای f به تدریج در فضای حالت اضافه می‌شوند.

❖ کانتور i ، همه‌ی گره‌های با $f=f_i$ را دارد که $f_i < f_{i+1}$

❖ اگر C^* هزینه‌ی مسیر راه‌حل بهینه باشد آن گاه:

❖ 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.

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 می شود ترازها نامنظم شوند.

❖ هرچه $h(n)$ هیوریستیک دقیق تر باشد، نوارها به سمت گره هدف کشیده می شوند و به صورت باریک تری پیرامون مسیر بهینه متمرکز می شوند.

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 می کنیم، باید بالاخره به نواری برسیم که در آن جا f مساوی هزینه ی مسیر تا یک حالت هدف است.

❖ بنابراین اگر تعداد گره ها با $f(G) = C^* < f$ متناهی باشد کامل خواهد بود.

❖ فاکتور انشعاب متناهی بوده و هزینه ی یال ها از ϵ بیشتر باشد.

❖ پیچیدگی زمانی؟

❖ برای اغلب مسائل، تعداد گره ها در داخل کانتور هدف (یعنی گره هایی با $f(n) \leq C^*$) بر حسب طول راه حل نمایی است.

ارزیابی جستجوی A^*

❖ پیچیدگی فضایی؟

❖ نمایی است، A^* تمام گره‌ها را در حافظه نگه می‌دارد.

❖ مانند هر روش جستجوی گرافی دیگر که Explored Set را نگهداری میکند.

❖ فضا مشکل اصلی‌تری نسبت به زمان است.

❖ برای جبران این مشکل، نسخه‌های دیگر A^* مانند IDA*, RBFS 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 گره‌های کمتری نسبت به A^* بسط دهد.


تست

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 منتهن آن گره تا هدف به‌عنوان 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.

3	6	9	12
2	5	8	11
1	4	7	10

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

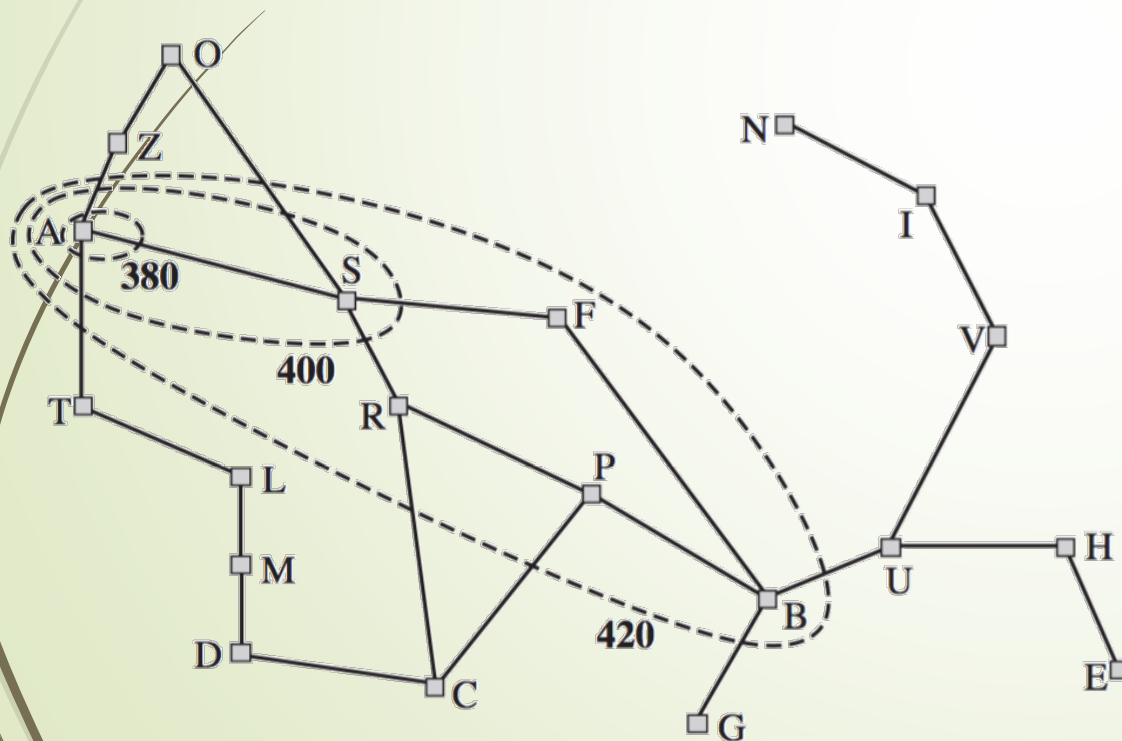




جستجوهای هیوریستیک با حافظه محدود

ایده

❖ الگوریتم A^* نیاز به نگهداری مجموعه‌ی frontier (و مجموعه‌ی explored) دارد. این موضوع باعث نیاز به حافظه‌ی خیلی زیادی می‌شود.



جستجوی عمیق شونده‌ی تکراری A^*

- ❖ IDA^* ، همانند الگوریتم IDS، 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).
- ❖ انتخاب نود در هر تکرار بر اساس جستجوی اول عمق است.
- ❖ تفاوت اصلی با IDS: مقدار برش بر اساس تابع $f=g+h$ تعیین می‌شود نه عمق.
- ❖ مقدار برش در تکرار جدید برابر کم‌ترین مقدار f گره‌ای قرار داده می‌شود که از مقدار برش در تکرار قبلی بیشتر باشد.
- ❖ بر حسب کانتورهای A^* ، عمق بیشتر می‌شود. بنابراین روح A^* (اگر اعتقاد داشته باشیم!) حاکم است، گرچه جستجوی اول عمق، مبنای جستجو است.
- ❖ ممکن است نودهایی در عمق کمتر به دلیل f بالاتر بسط داده نشوند (خارج از کانتور باشند) در حالی که نودهای عمیق‌تر به دلیل f پایین‌تر داخل کانتور قرار گرفته باشند.
- ❖ IDA^* به همراه جستجوی درختی (و نه گرافی) موجب بهبود پیچیدگی فضایی A^* می‌شود.

الگوریتم عمیق شونده‌ی 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 \leftarrow MAKE-NODE(INITIAL-STATE[*problem*])

f-limit \leftarrow *f*- COST(*root*)

loop do

solution, *f-limit* \leftarrow 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* \leftarrow DFS-CONTOUR(*s*, *f-limit*)

if *solution* is non-null **then return** *solution*, *f-limit*

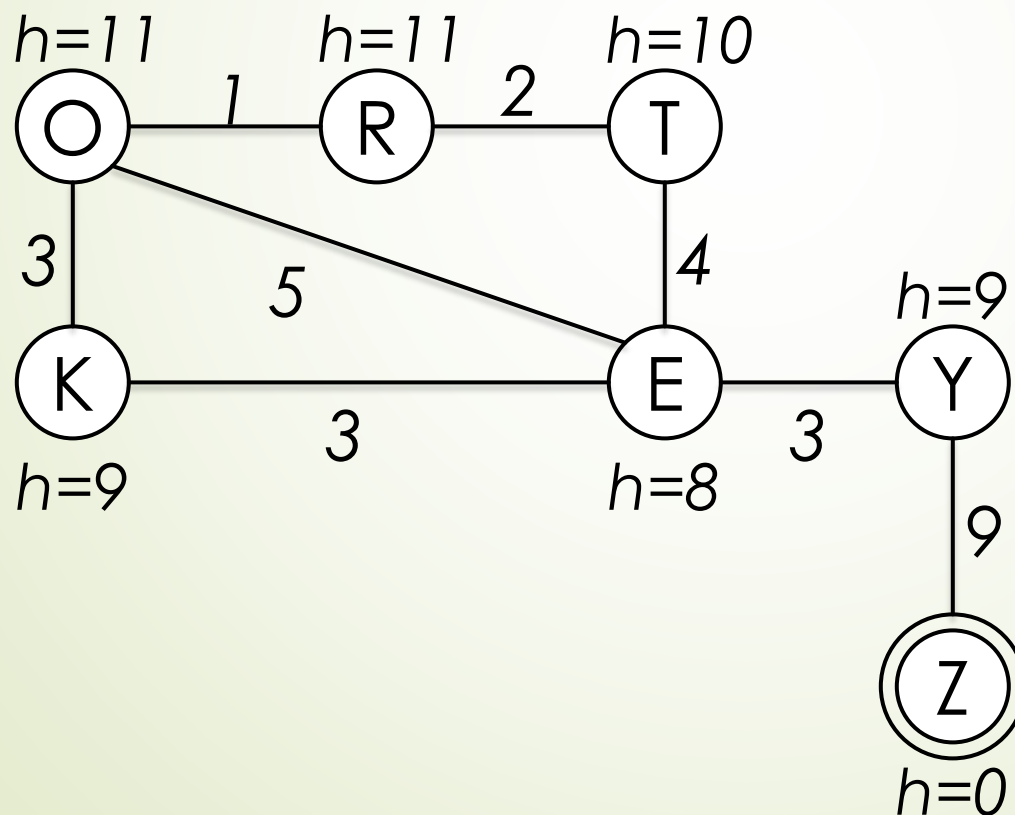
next-f \leftarrow MIN(*next-f*, *new-f*); **end**

return null, *next-f*

مثال IDA*

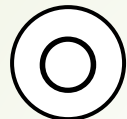
❖ برای یادآوری نکات جستجوی گرافی، در مثال زیر از IDA* به همراه جستجوی گرافی استفاده می‌کنیم.

❖ نود بسط داده شده در هر جستجوی عمقی (از جمله نود پدر) دوباره بسط داده نمی‌شود.

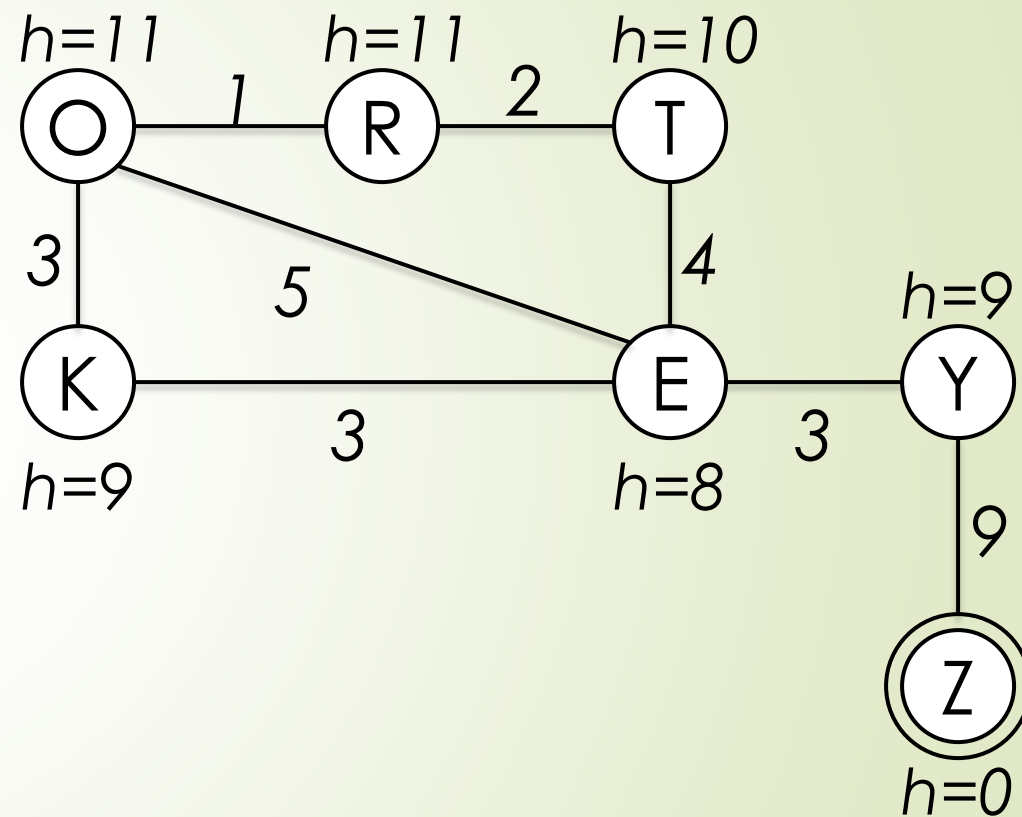


$F\text{-limit} = 11$

$$0 + 11 = 11$$

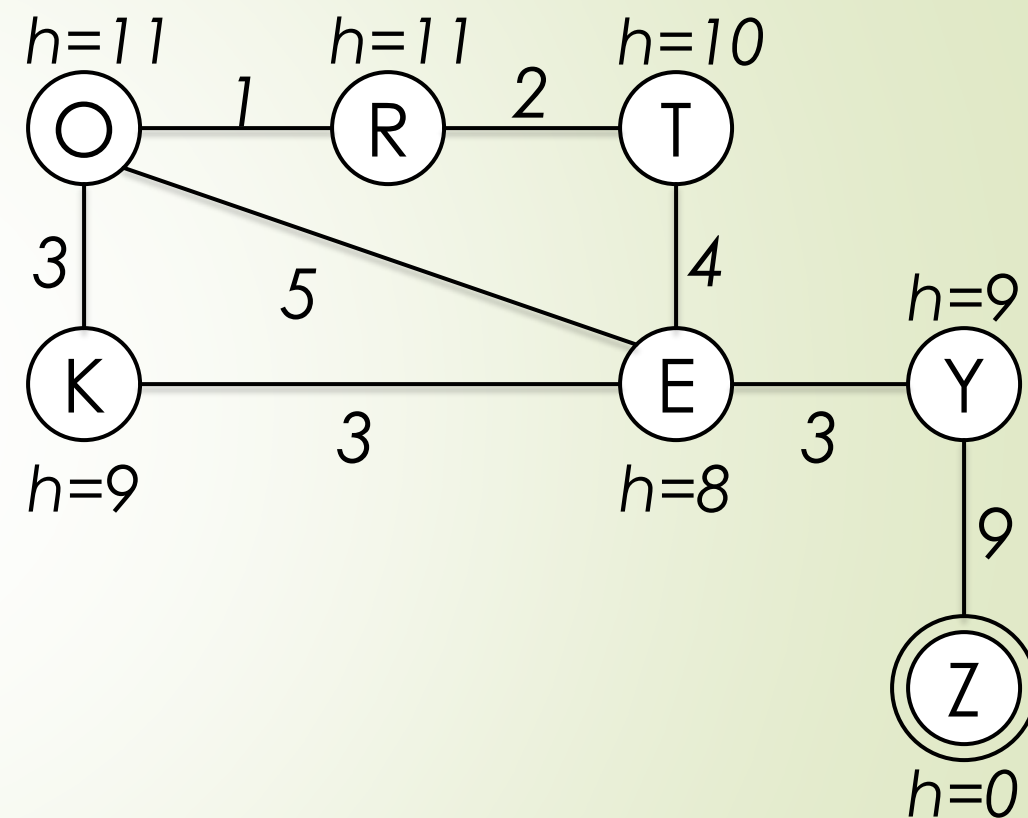
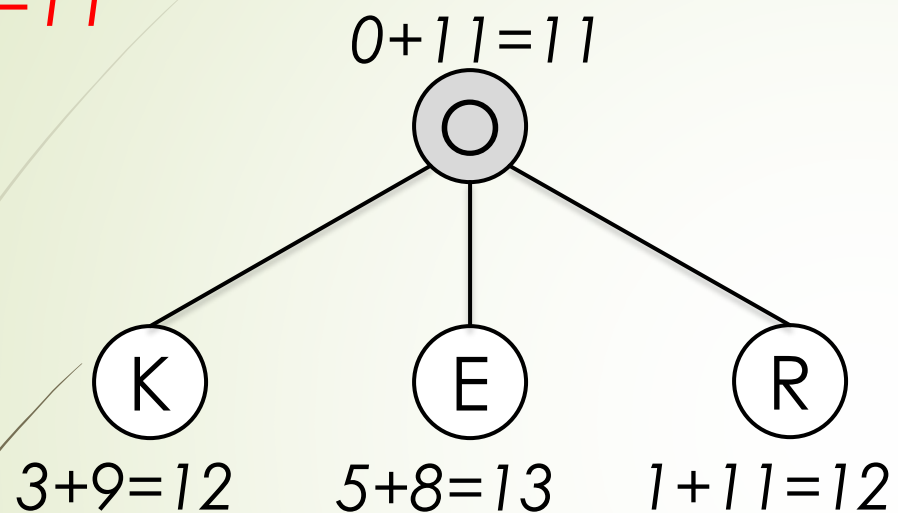


مثال IDA*



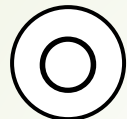
مثال IDA*

$F\text{-limit}=11$

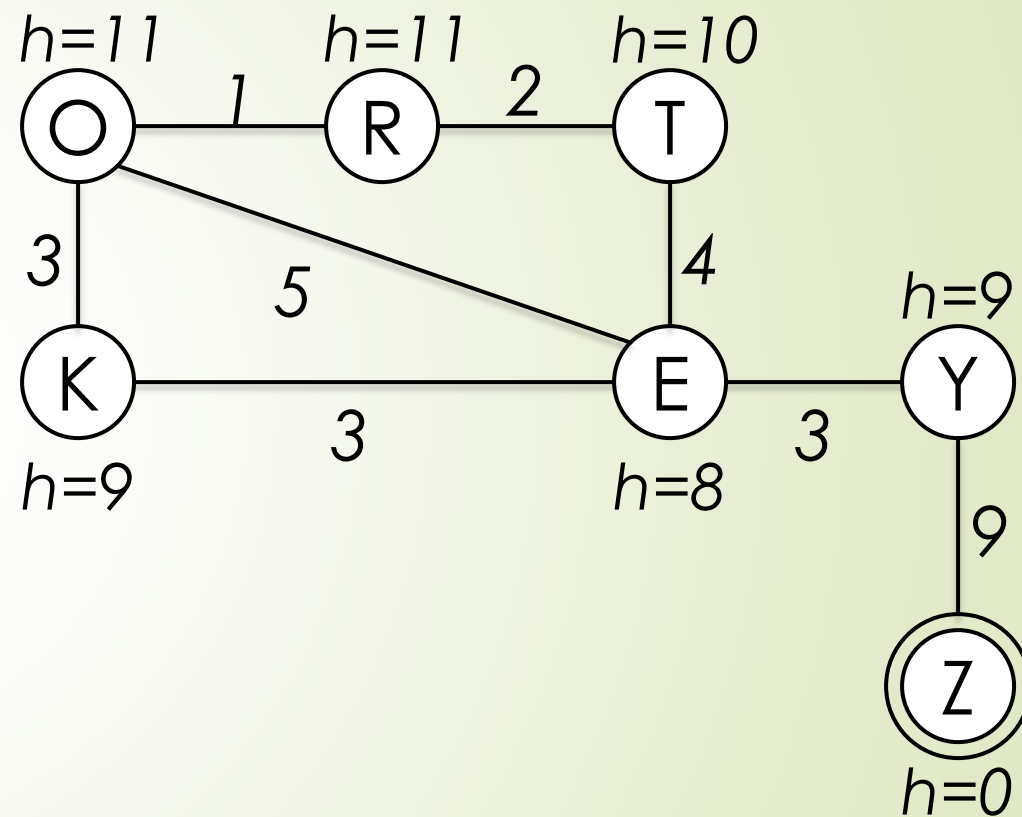


$F\text{-limit}=12$

$$0+11=11$$

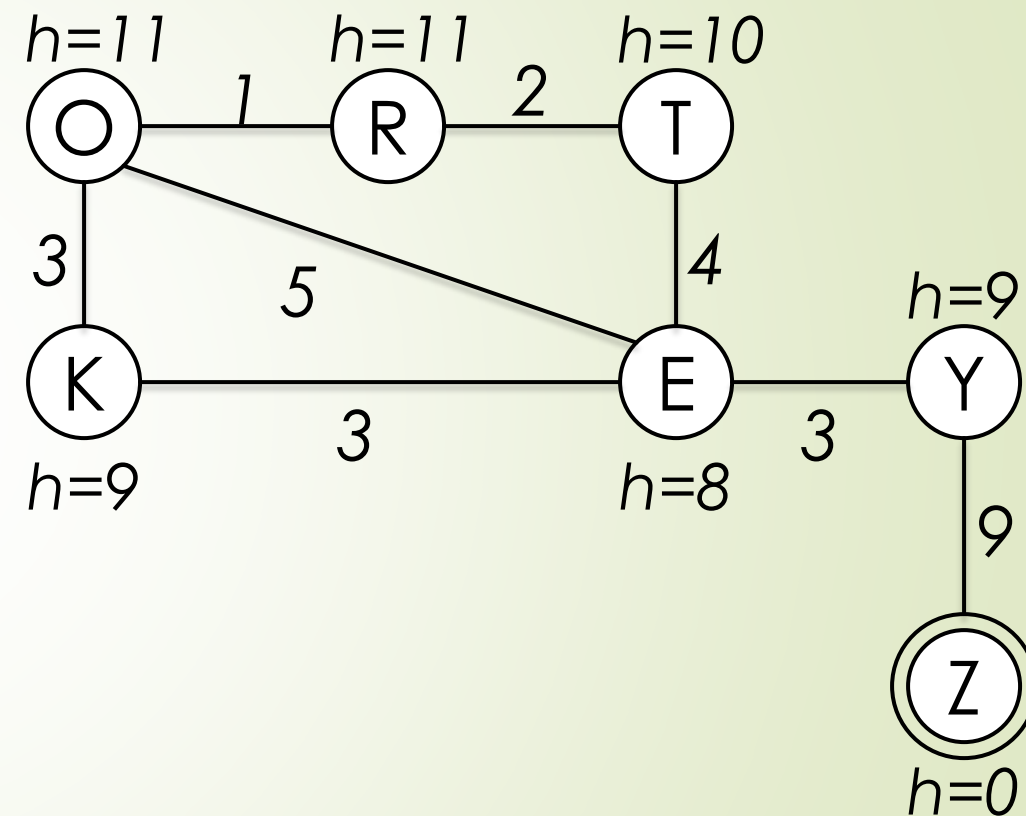
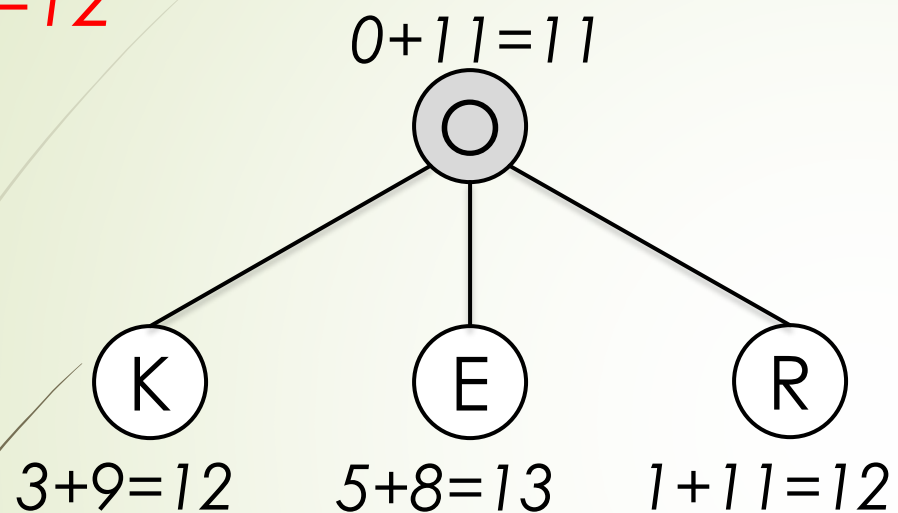


مثال IDA*



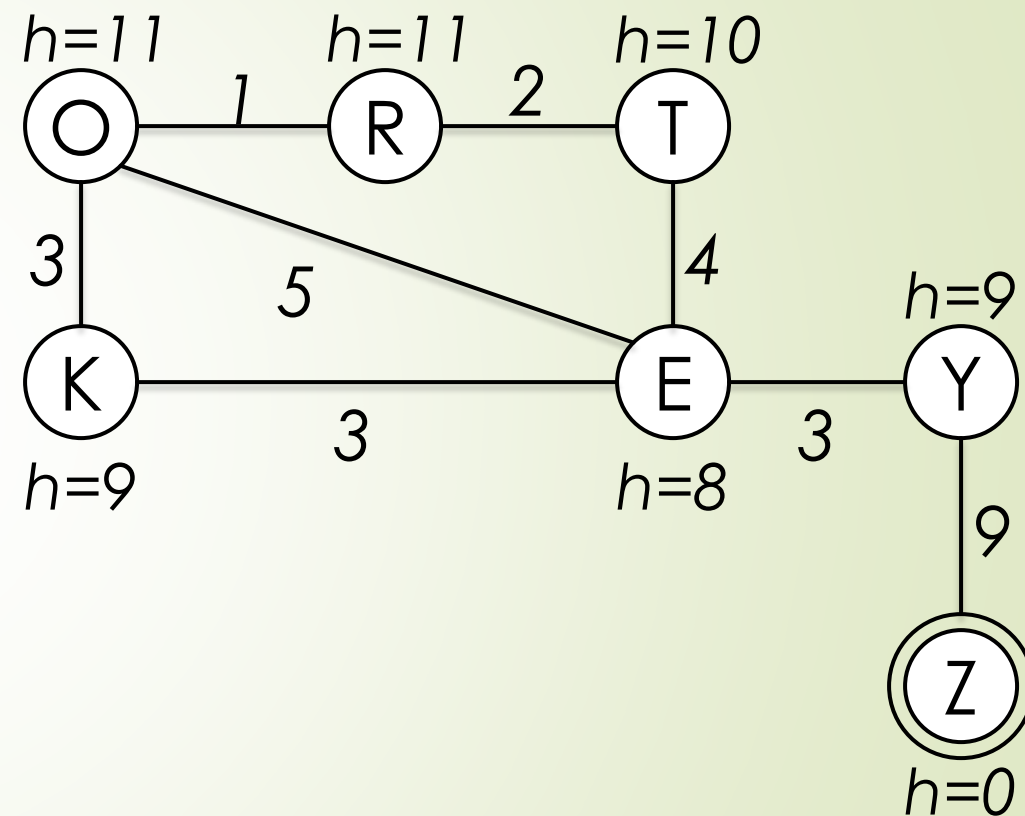
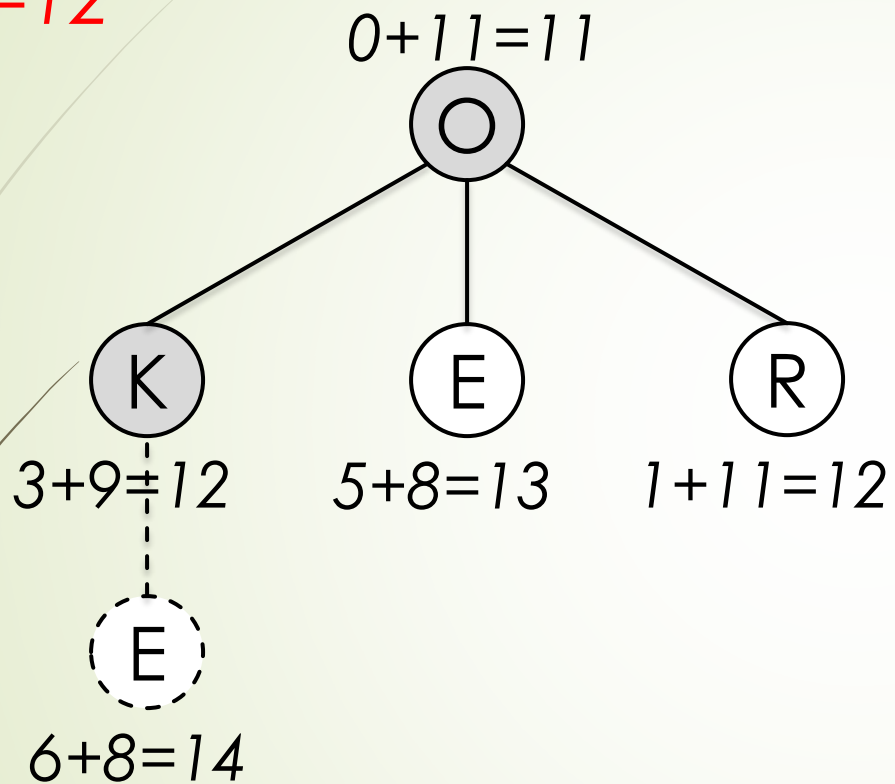
مثال IDA*

$F\text{-limit}=12$



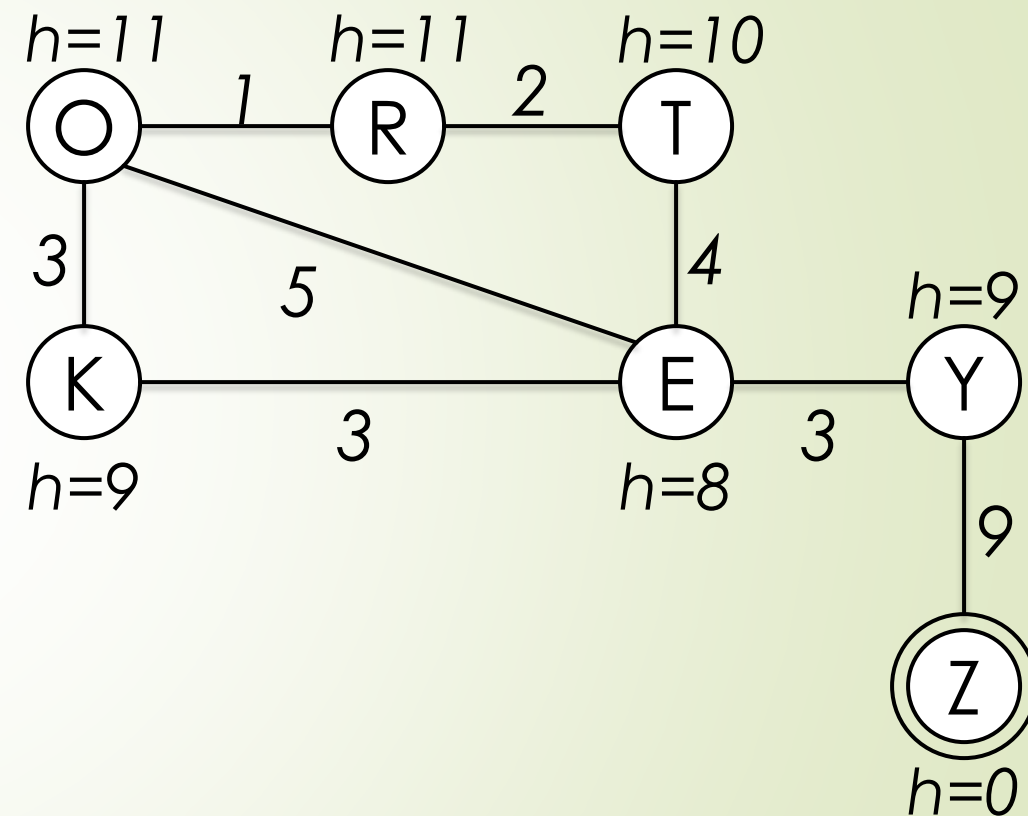
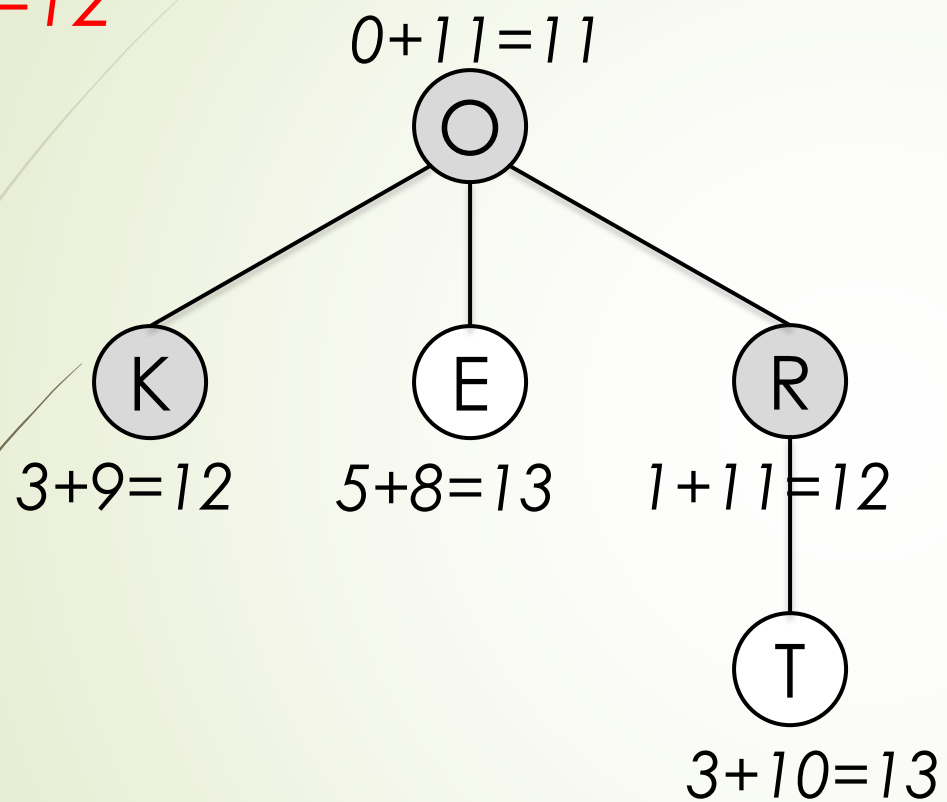
مثال IDA*

$F\text{-limit}=12$



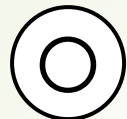
مثال IDA*

$F\text{-limit}=12$

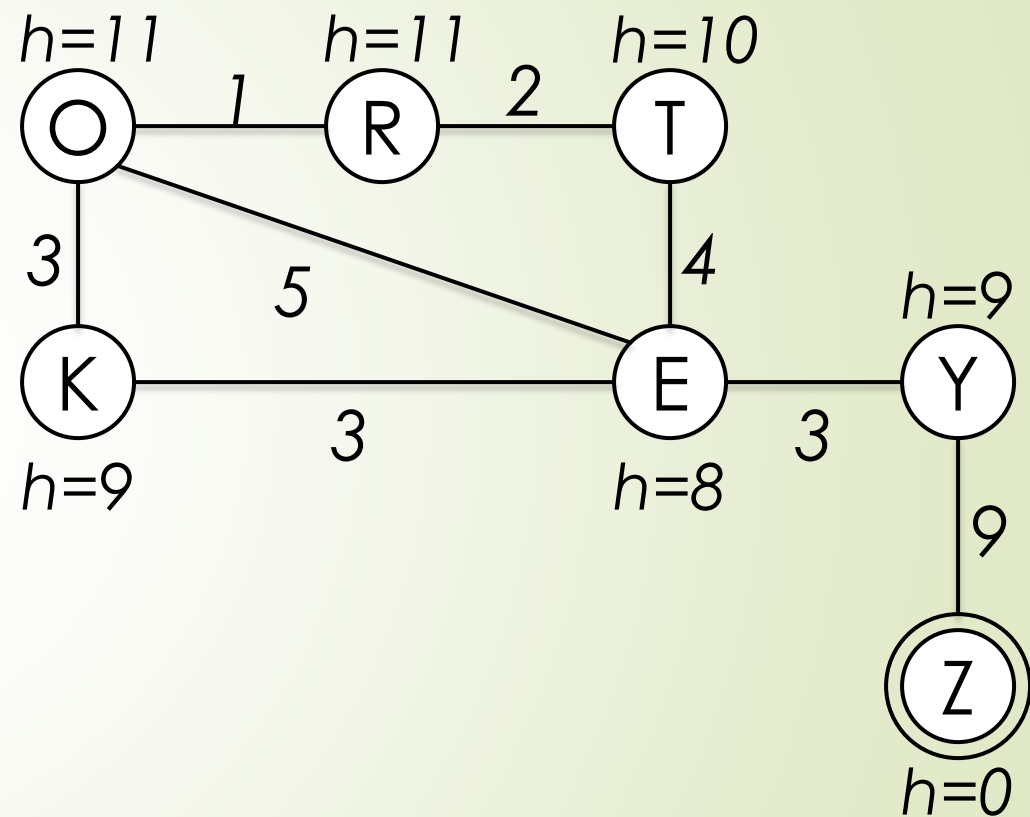


$F\text{-limit}=13$

$$0+11=11$$

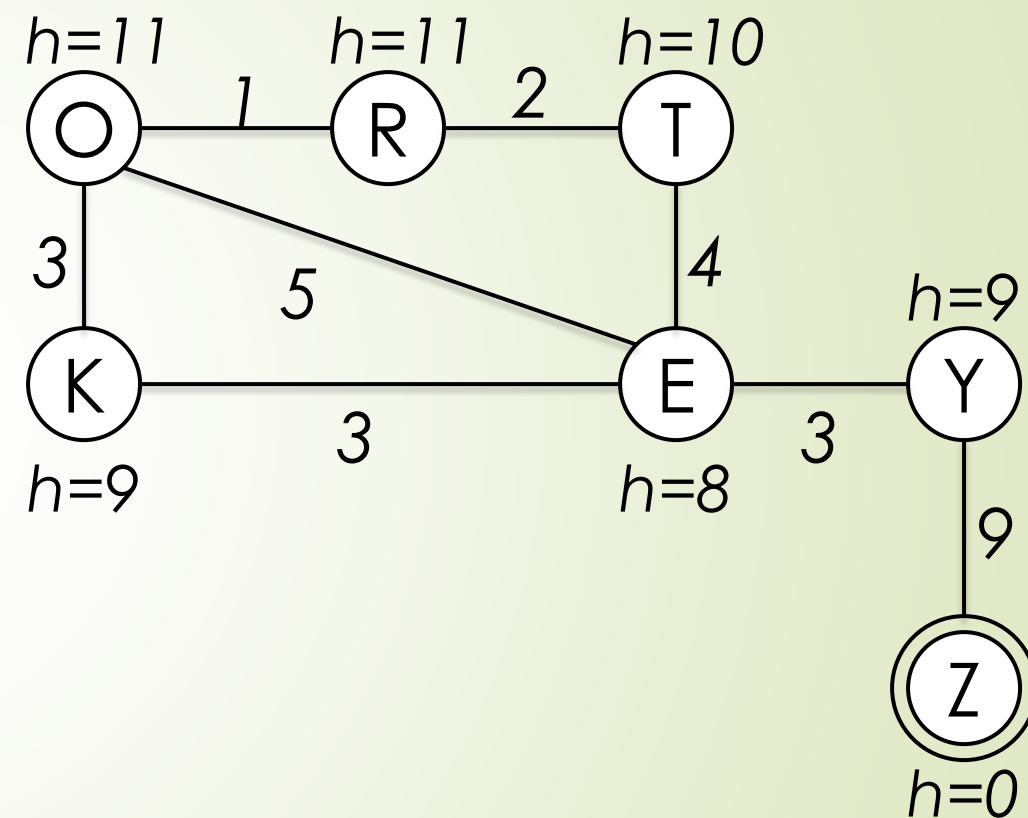
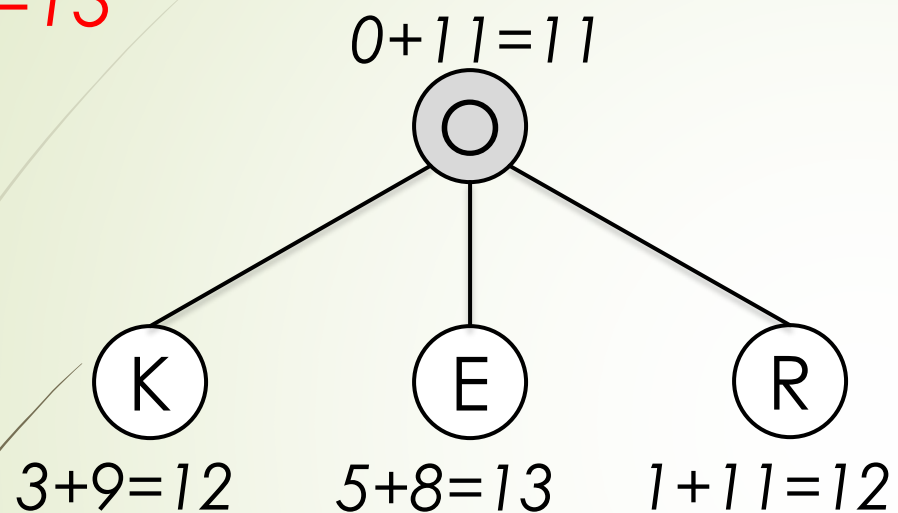


مثال IDA*



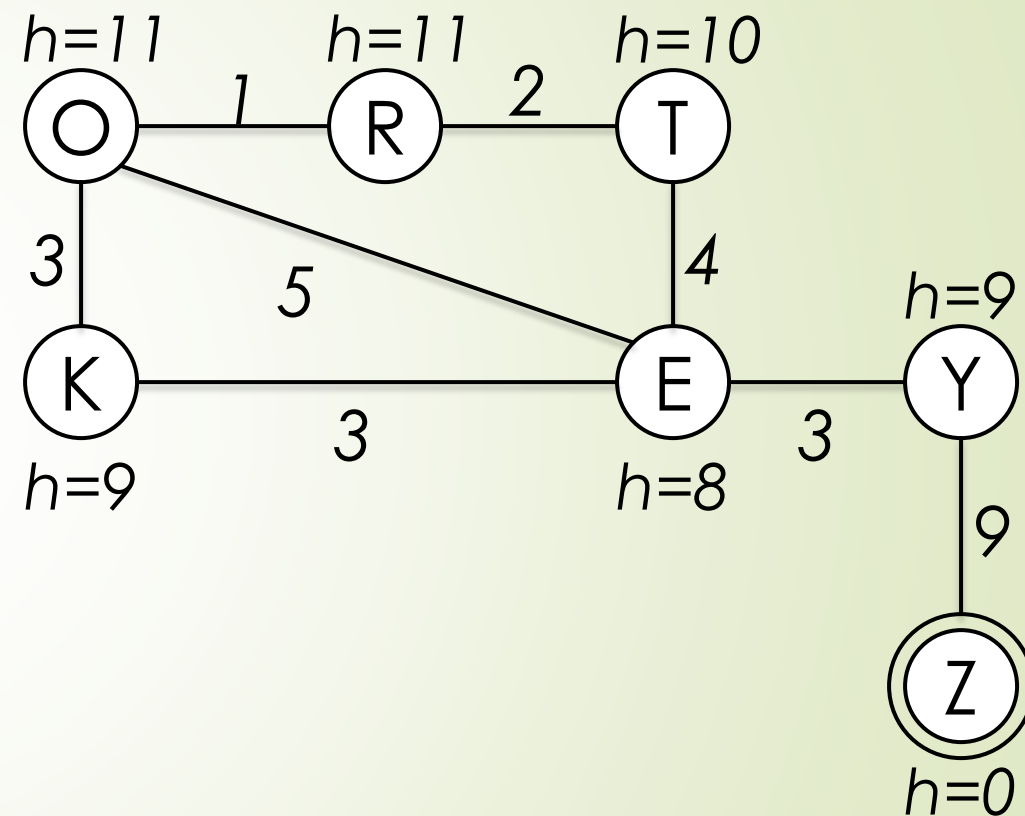
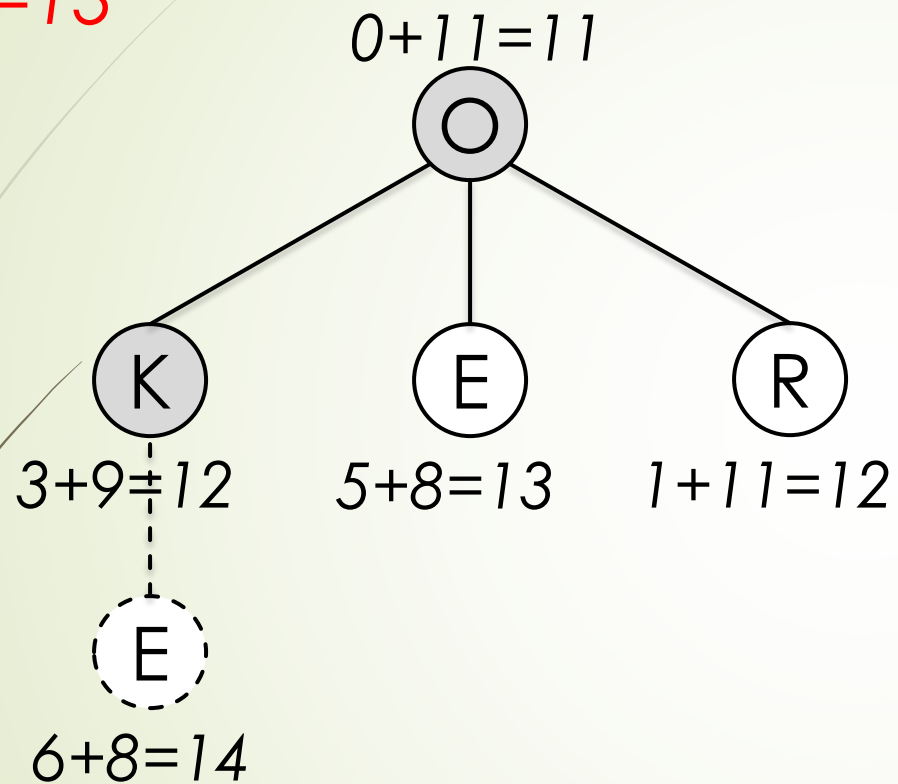
مثال IDA*

$F\text{-limit}=13$



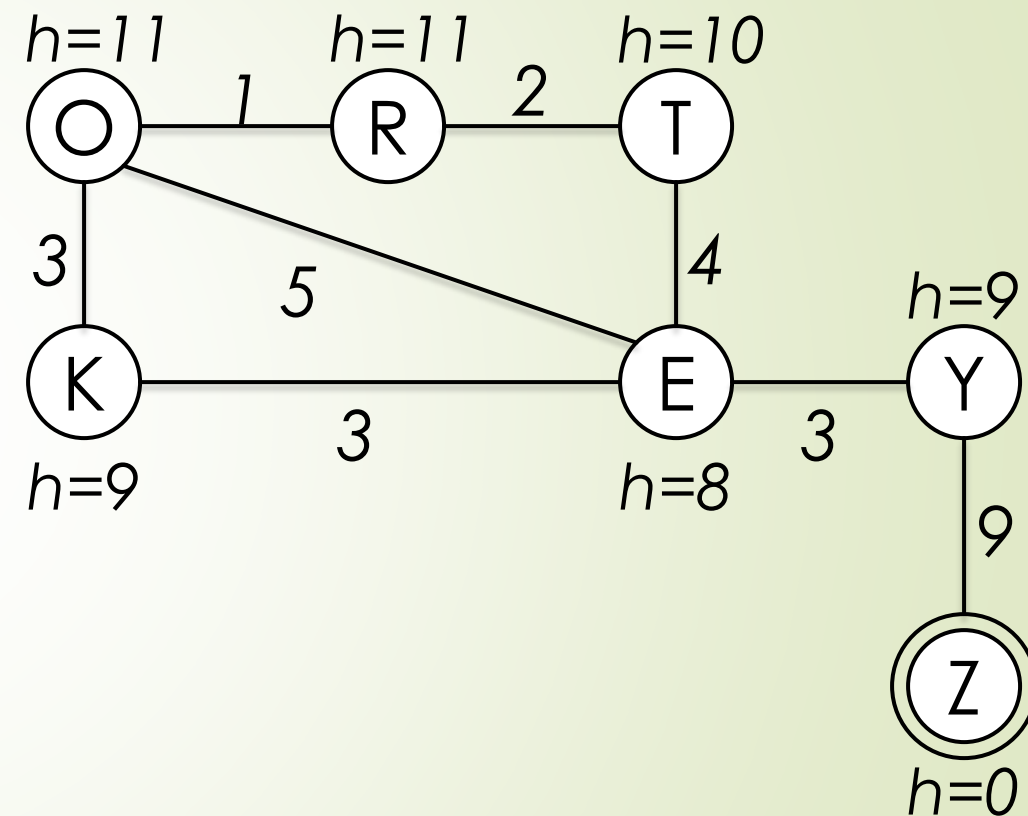
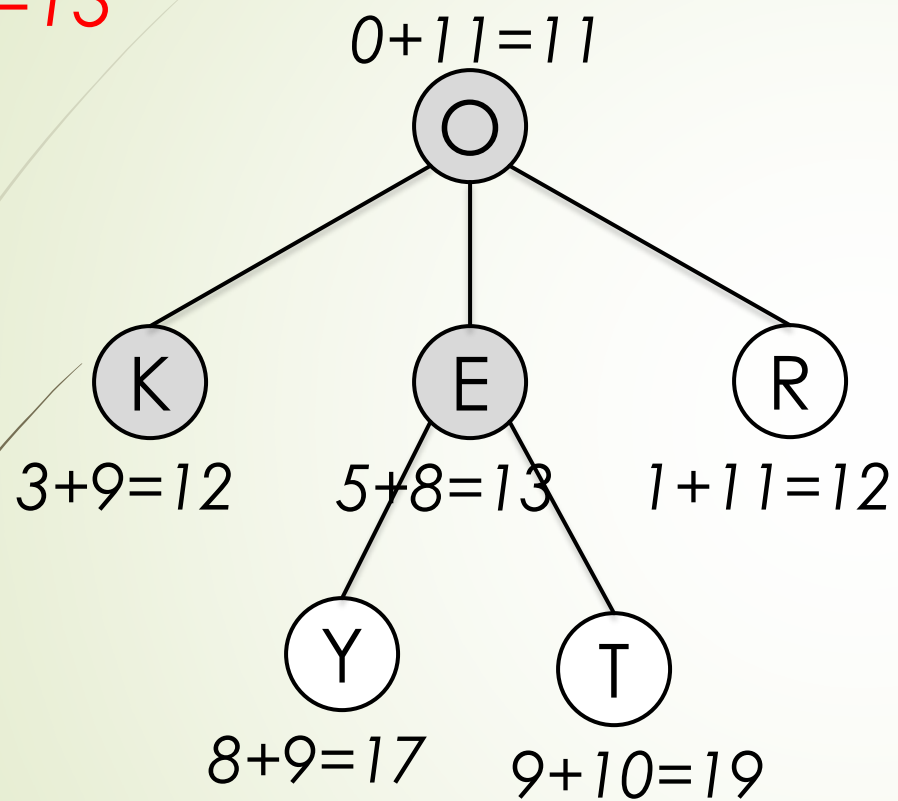
مثال IDA*

$F\text{-limit}=13$



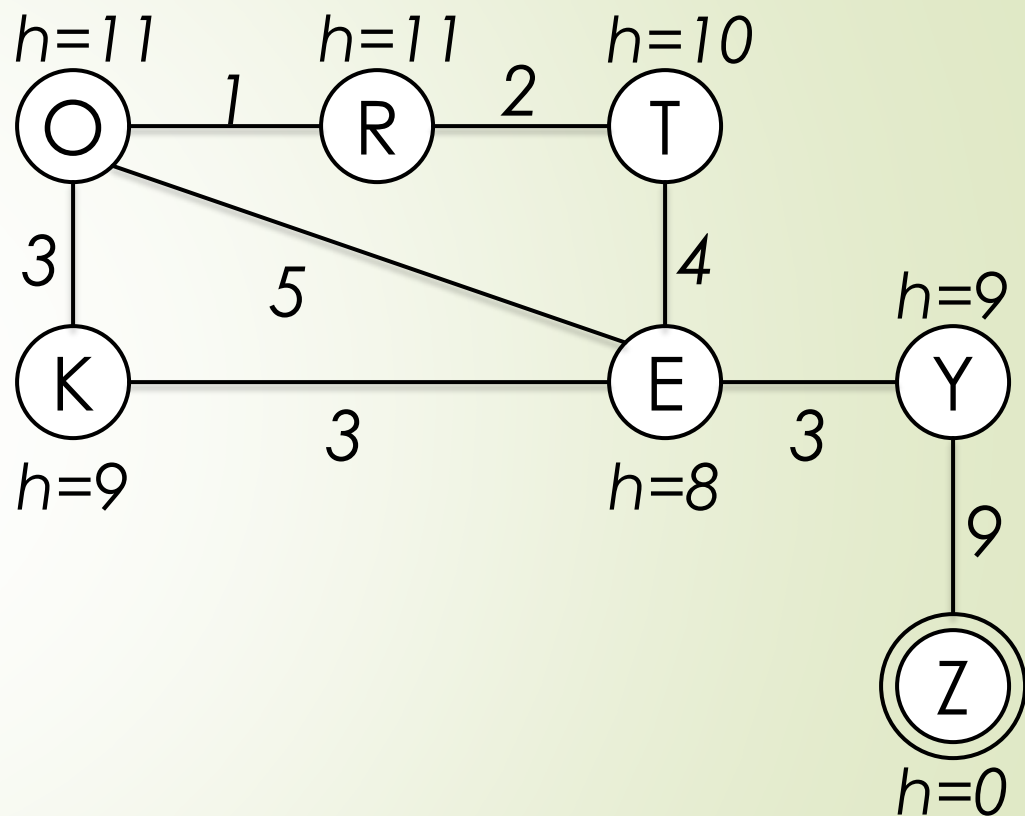
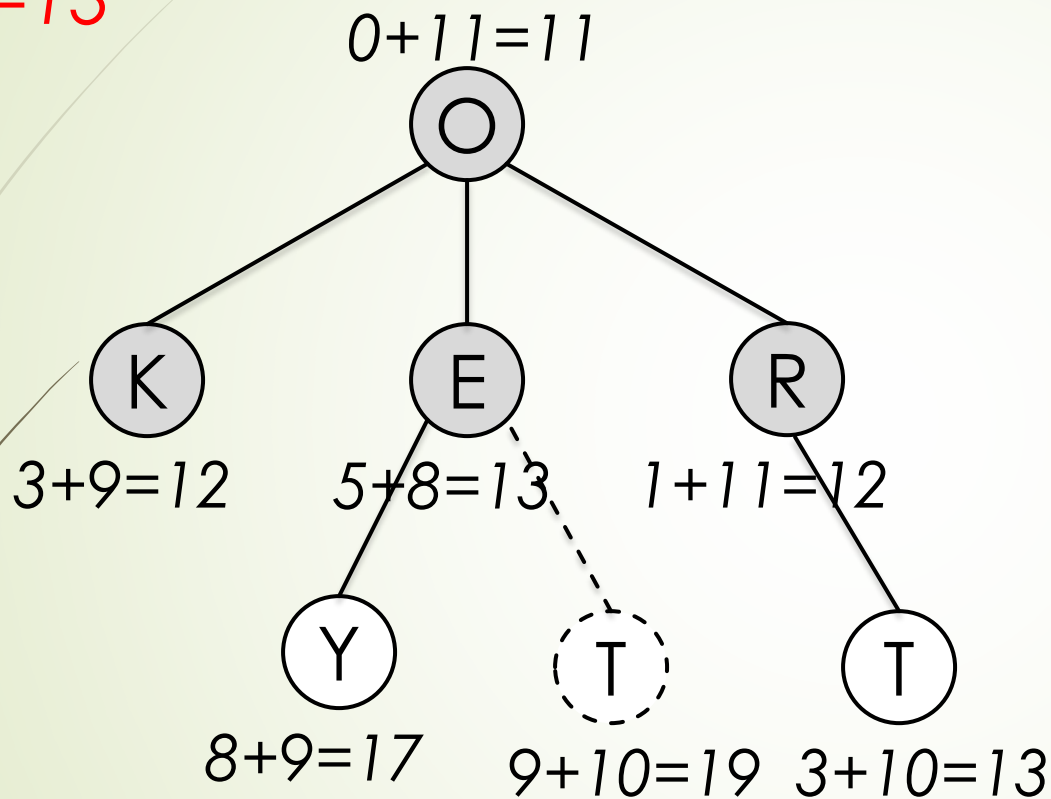
مثال IDA*

$F\text{-limit}=13$



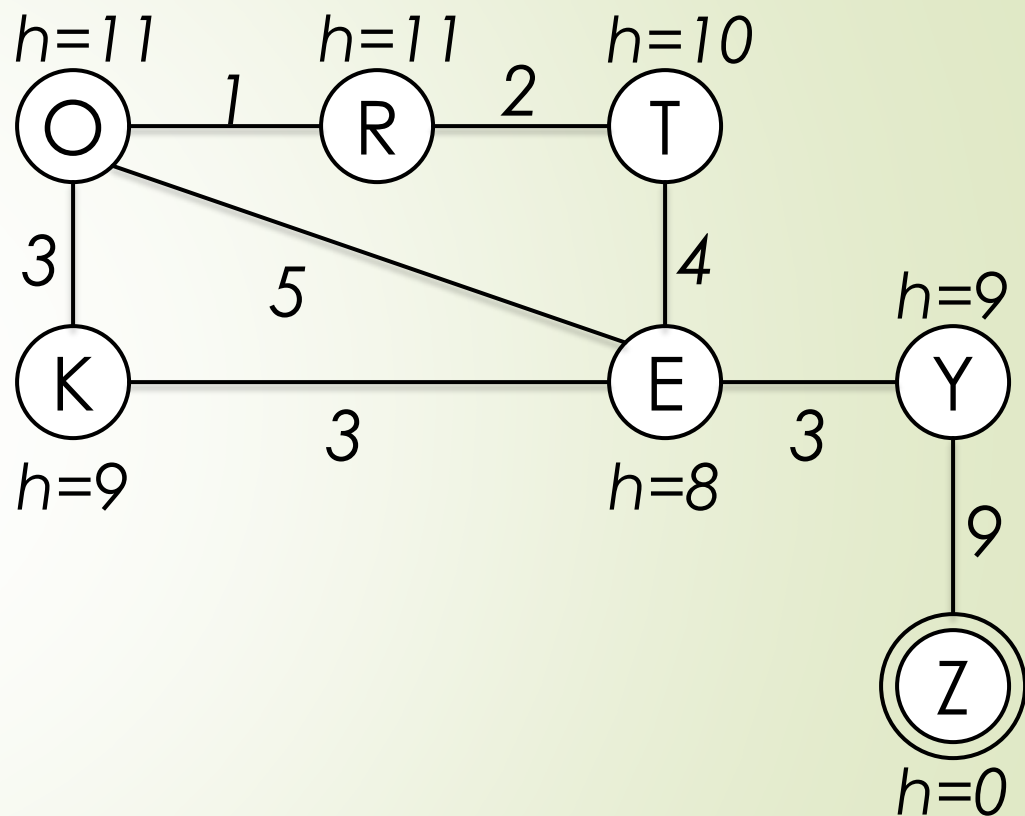
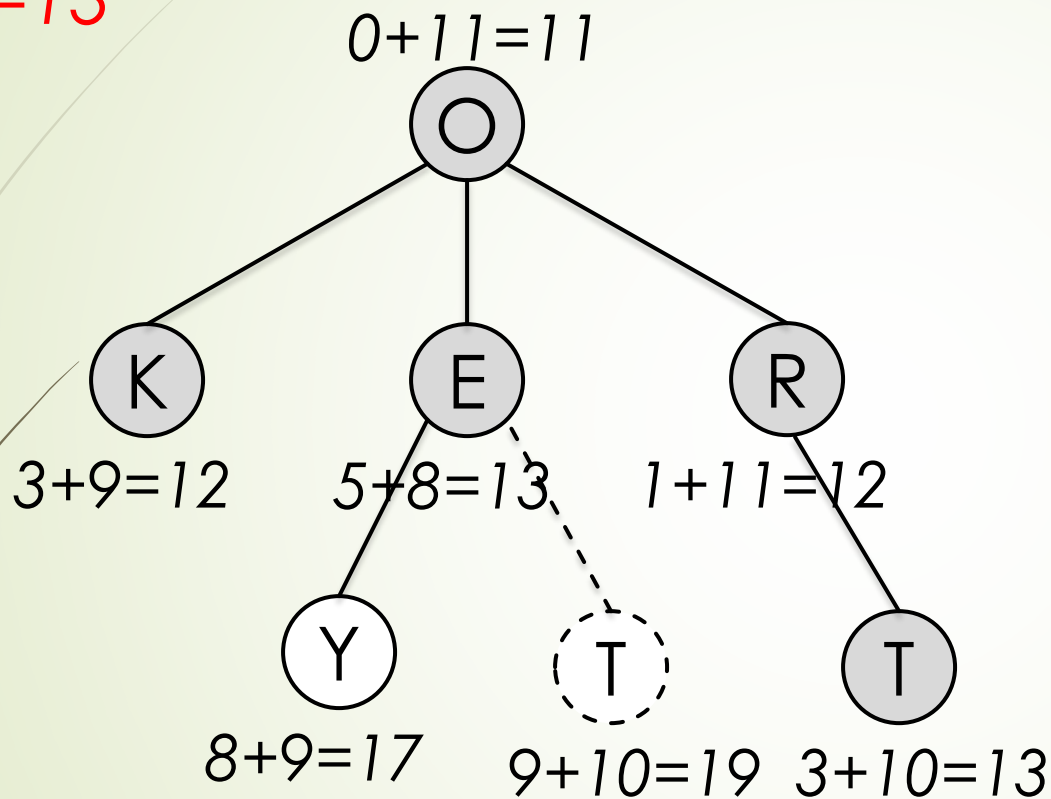
مثال IDA*

$F\text{-limit}=13$



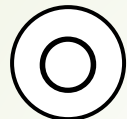
مثال IDA*

$F\text{-limit}=13$

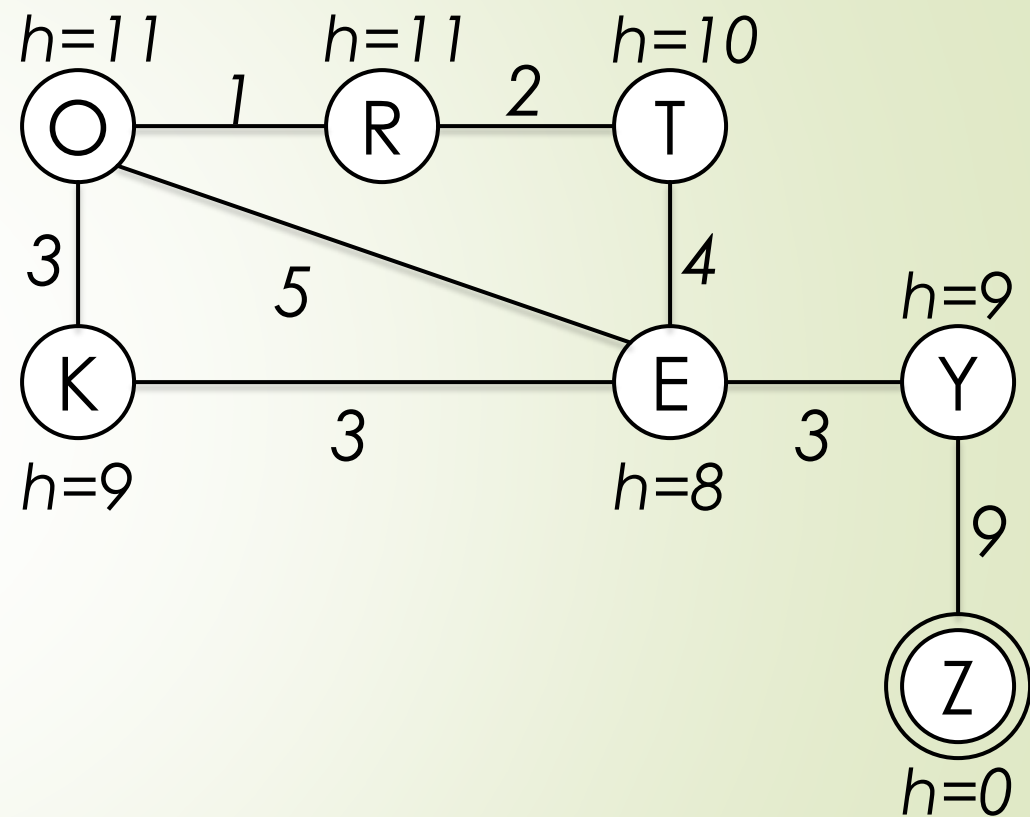


$F\text{-limit}=17$

$$0+11=11$$

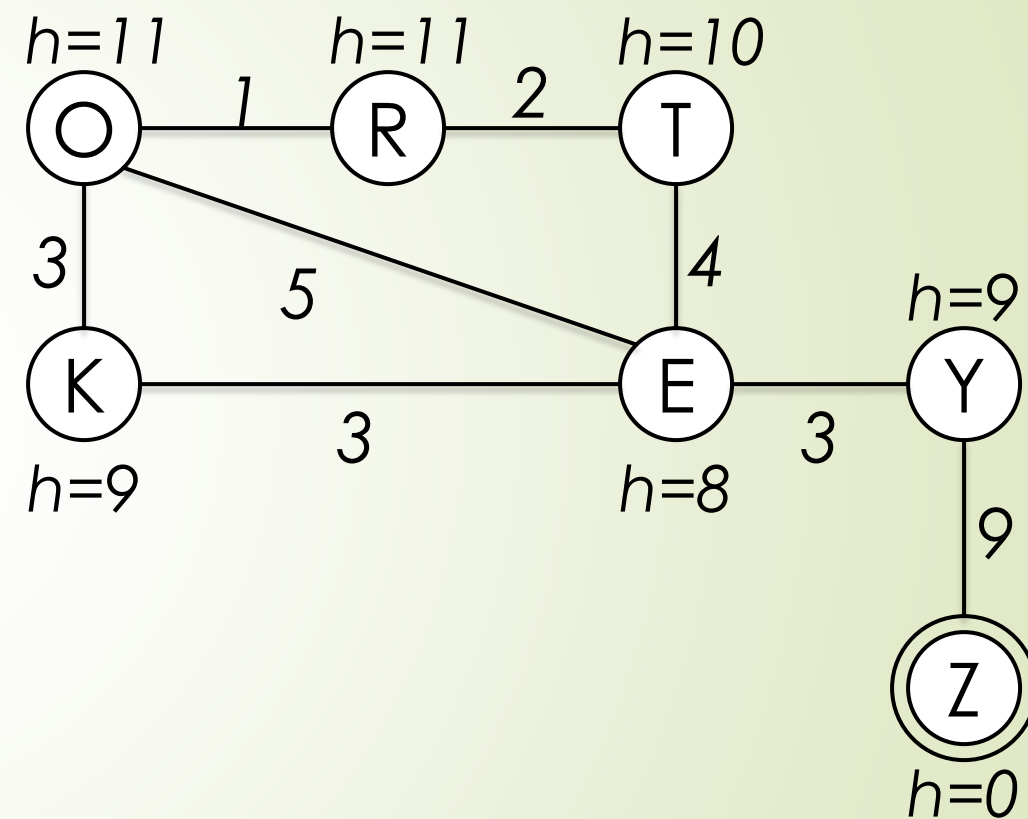
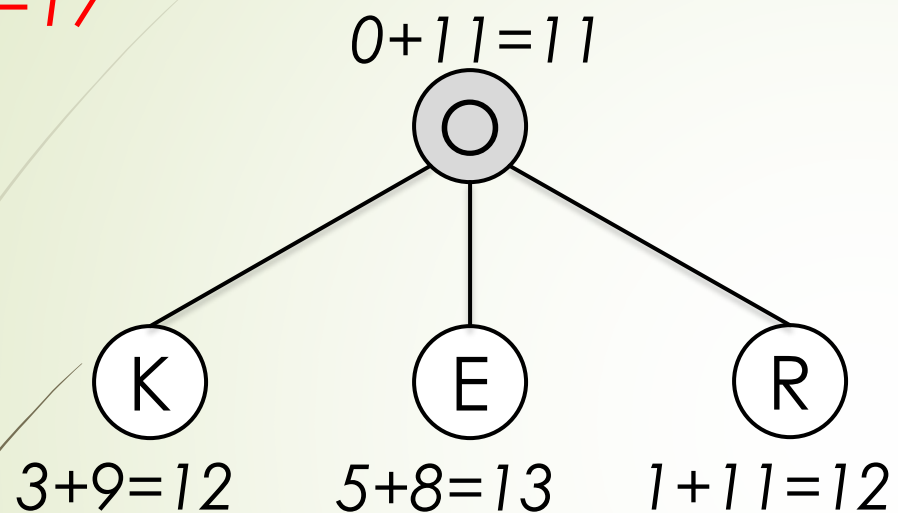


مثال IDA*



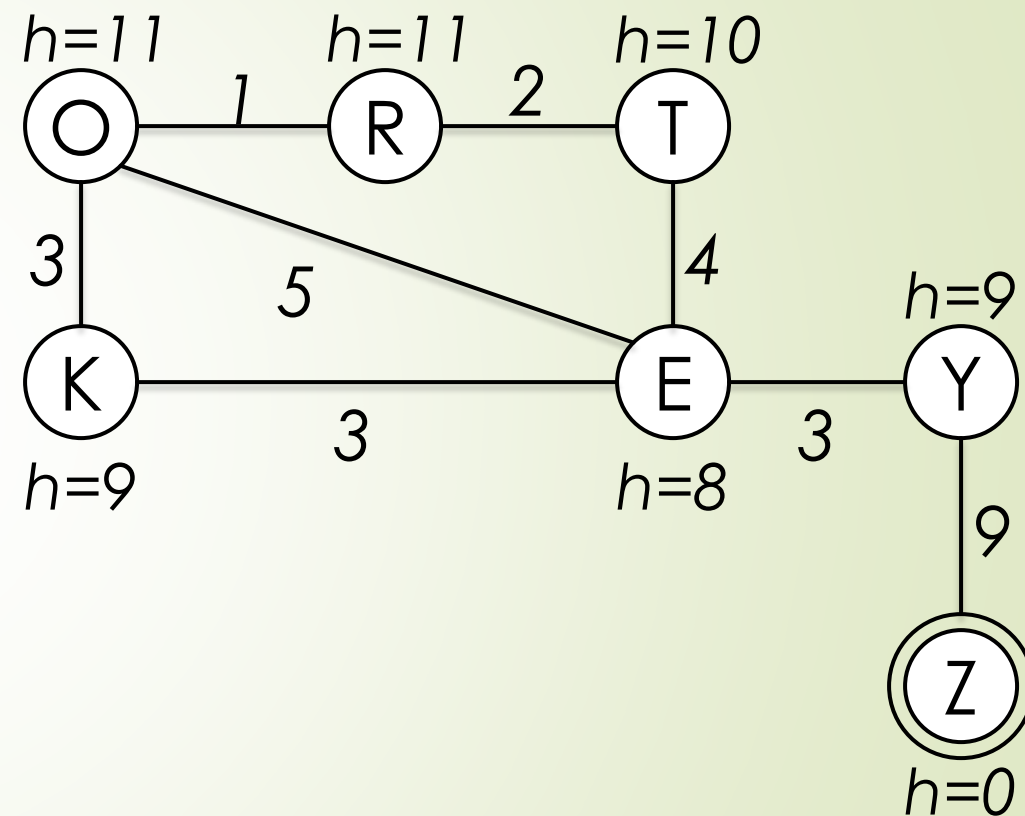
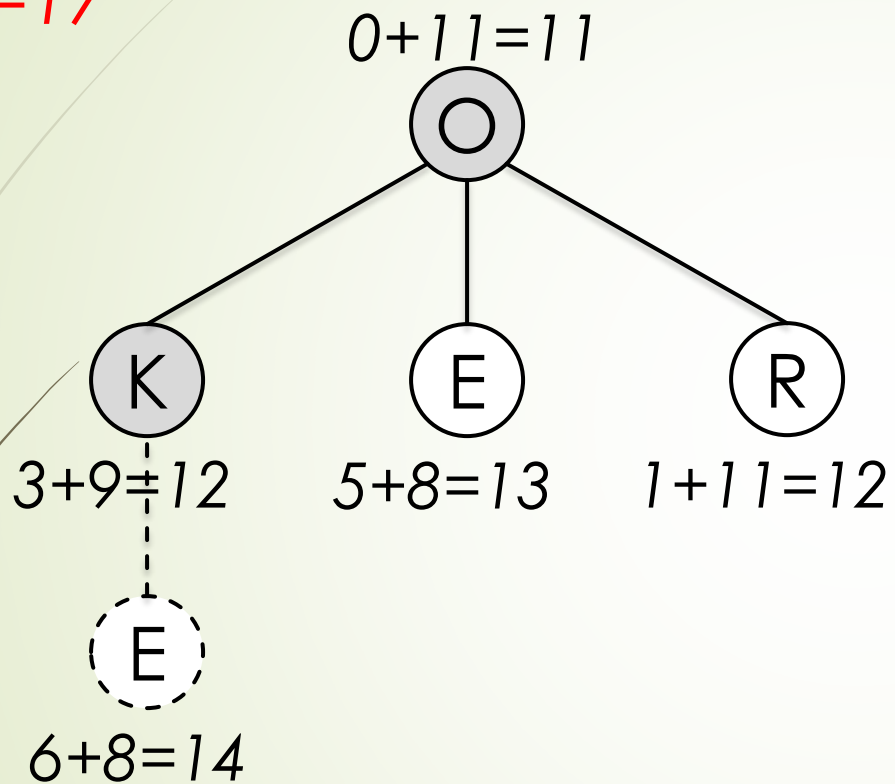
مثال IDA*

$F\text{-limit}=17$



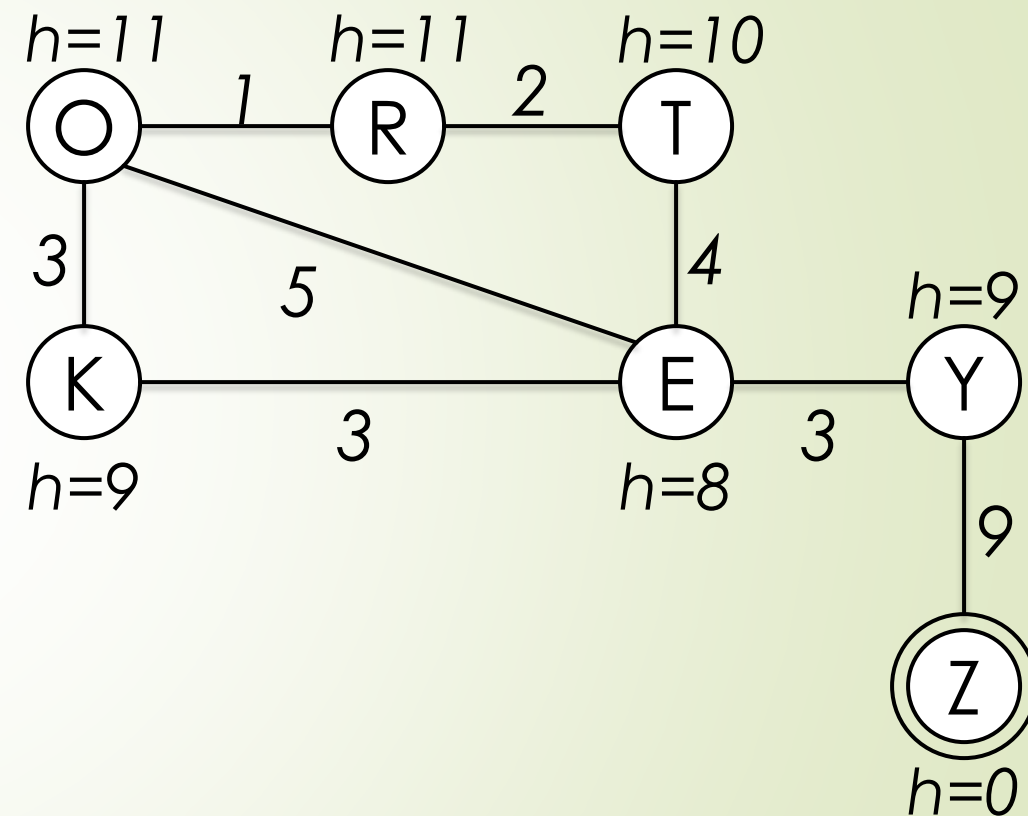
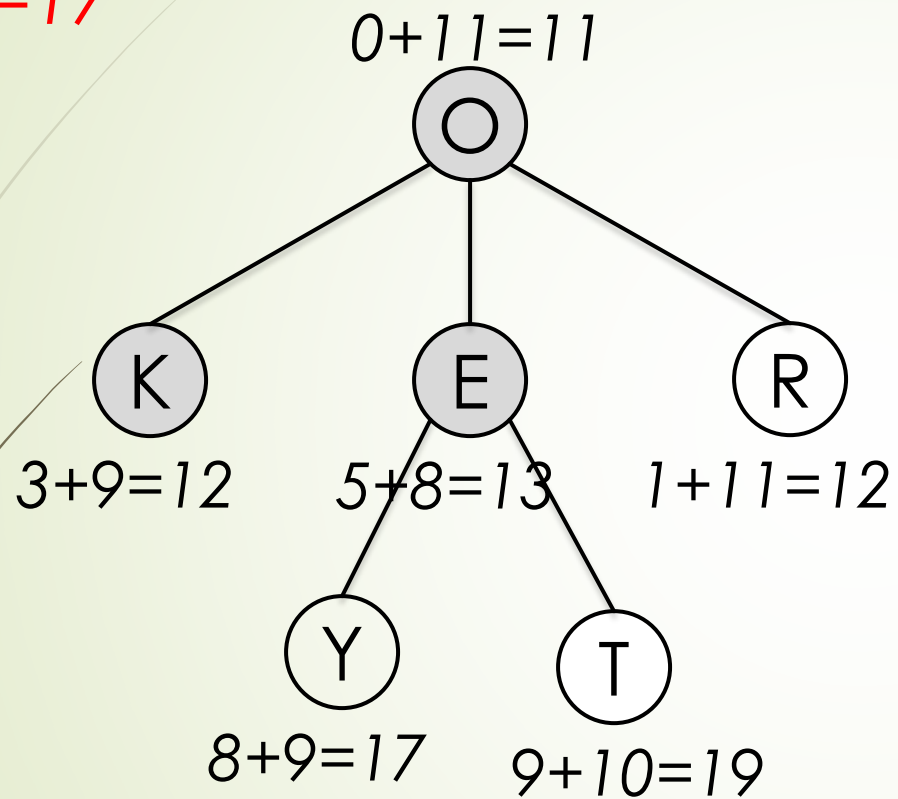
مثال IDA*

$F\text{-limit}=17$



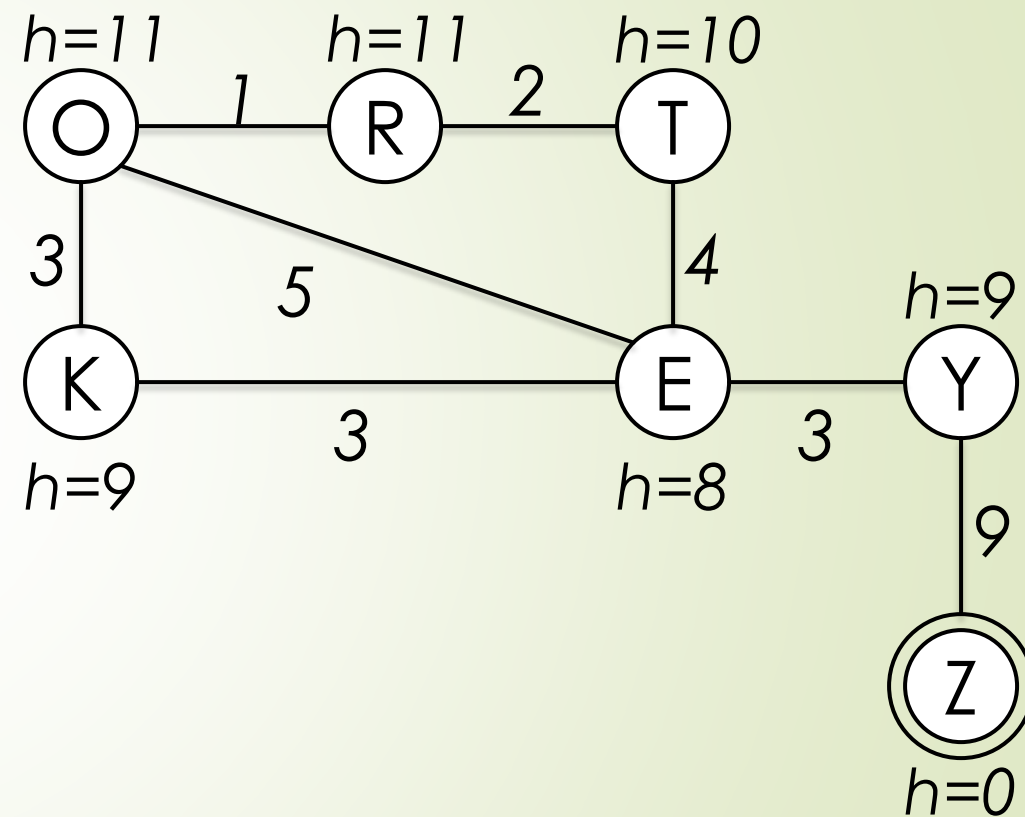
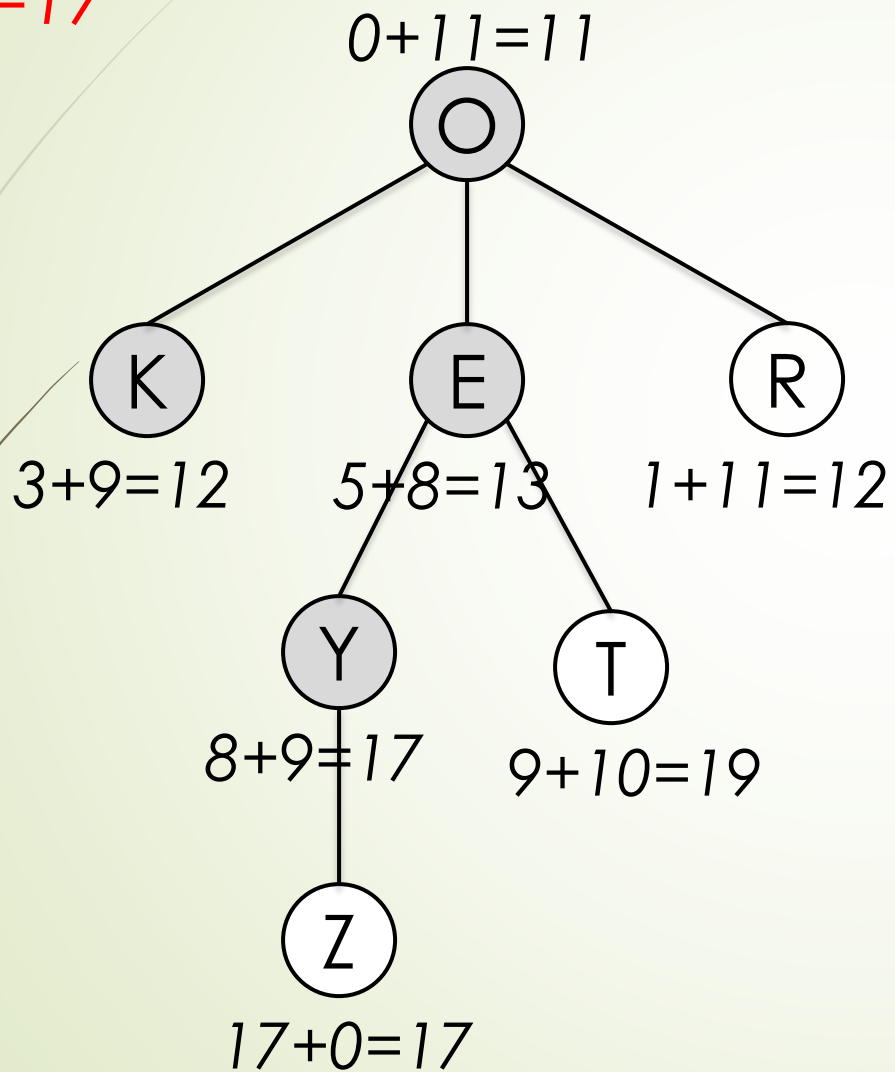
مثال IDA*

$F\text{-limit}=17$



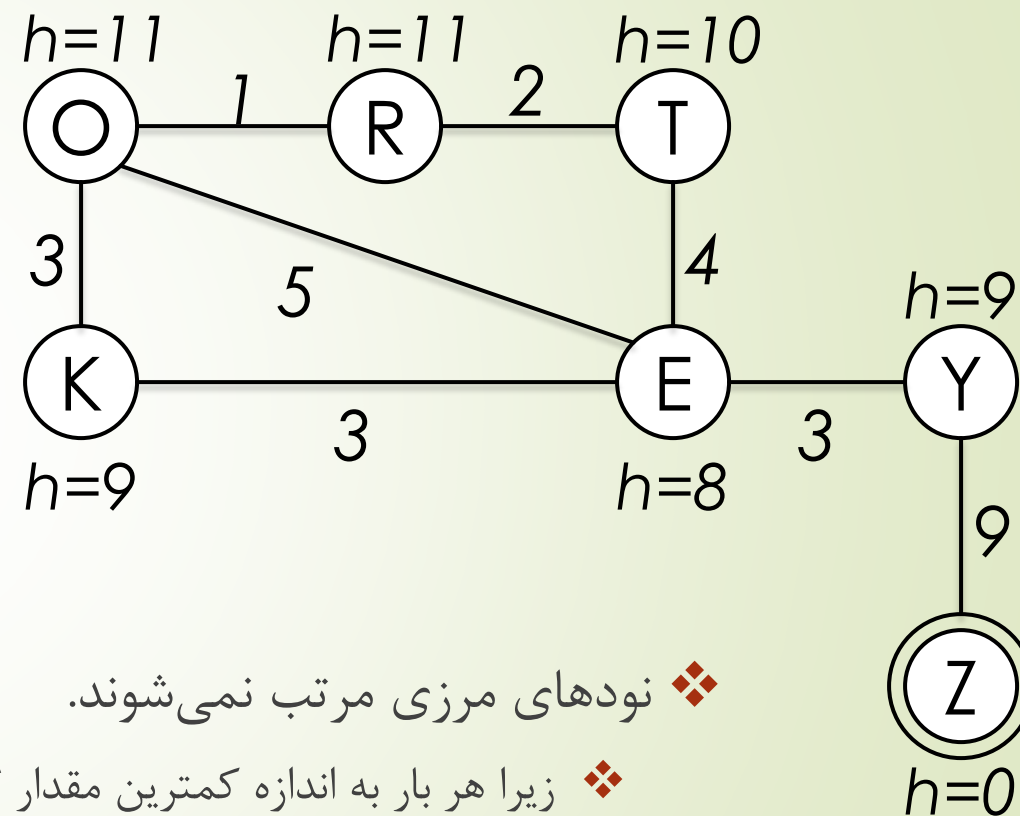
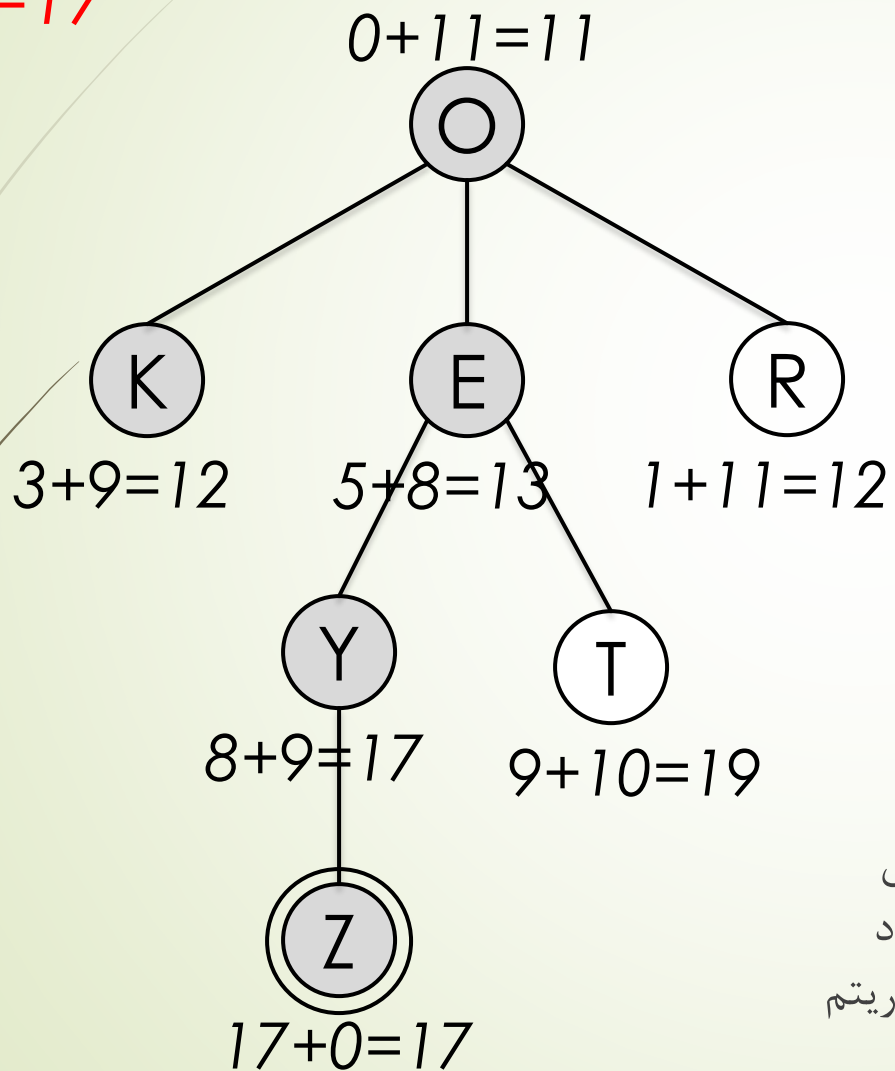
مثال IDA*

$F\text{-limit}=17$



مثال IDA*

$F\text{-limit}=17$



❖ نودهای مرزی مرتب نمی‌شوند.

❖ زیرا هر بار به اندازه کمترین مقدار لازم محدودیت بزرگتر میشود. لذا با بسط هر نودی اگر به جواب برسیم، جواب بهتری (با بسط نود دیگر) وجود ندارد، وگرنه در مرحله قبلی الگوریتم (با محدودیت کوچکتر) کشف می‌شد.

ارزیابی IDA*

❖ کامل و بهینه؟

❖ بله، اگر تابع هیوریستیک قابل قبول باشد، بهینه است.

❖ پیچیدگی زمانی؟

❖ در آخرین تکرار الگوریتم، بیشترین تعداد گره تولید می شود.

❖ همانند A^* در بدترین حالت نمایی است.

❖ **نیازی به یک صف اولویت برای نگه داری گره های مرزی ندارد. در نتیجه از سربار مرتب سازی چنین صفی بی نیاز است.**

❖ اگر مقدار تابع f برای هر حالت متفاوت باشد (هیچ دو گره ای در درخت، f برابر نداشته باشند)، در هر تکرار فقط یک گره بیشتر از تکرار قبلی بسط می یابد.

❖ در این حالت اگر A^* تعداد $O(N)$ گره را بسط دهد IDA^* باید N بار تکرار شود یعنی $1+2+3+...+O(N)=O(N^2)$ که زمان بسیار زیادی است.

ارزیابی IDA*

❖ پیچیدگی فضایی؟

❖ اگر از جستجوی گرافی استفاده کند عملاً مشکلی را حل نکرده است!! (چرا؟)

❖ نگهداری مجموعه Explored و نیاز به حافظه (مشکل اصلی A^*)

❖ از جستجوی درختی استفاده می کند و در هر مرحله فقط گره هایی که مقدار f آن ها کم تر از $f\text{-limit}$ است در حافظه نگهداری می شود. بنابراین پیچیدگی مانند جستجوی عمقی، خطی است.

❖ تفاوت با A^* با جستجوی درختی؟

❖ عدم نیاز به مرتب سازی نودهای مرزی

❖ عدم ذخیره سازی نودهای با f بیشتر از $f\text{-limit}$

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

❖ C^* هزینه راه حل بهینه

❖ ϵ کم ترین هزینه اعمال

تست

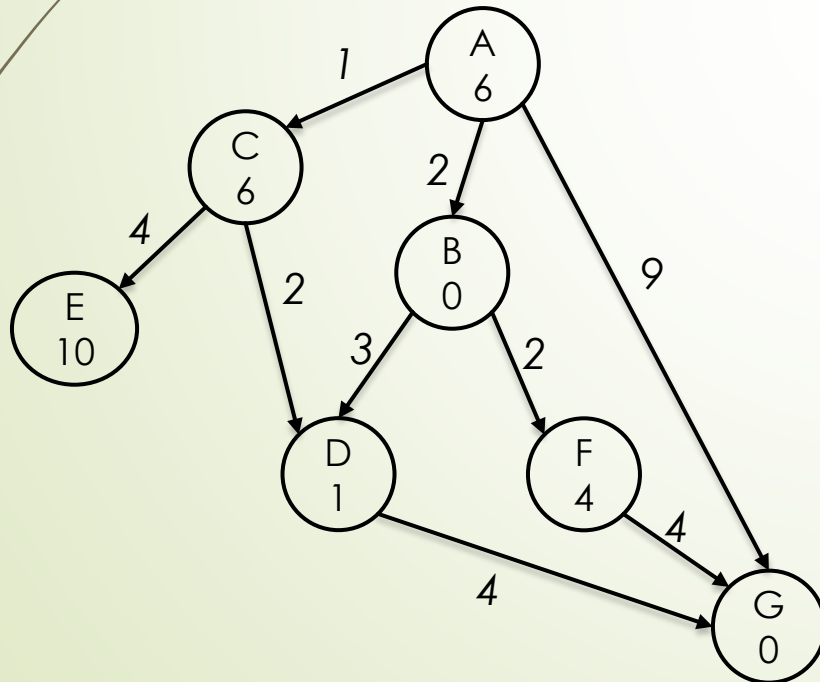
در گراف زیر، گره A وضعیت شروع و گره G وضعیت هدف است. اعداد کنار هر لبه (link) هزینه عبور آن لبه است. مقدار تابع اکتشافی h هر گره، درون آن نوشته شده است. اگر مقدار آستانه را برابر با عدد ۷ در نظر بگیریم، کدام یک از گزینه‌های زیر، از چپ به راست، ترتیب ملاقات (visit) گره‌های این گراف توسط روش IDA* را نشان 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.

(۱) A, B, D, G

(۲) A, C, D, G

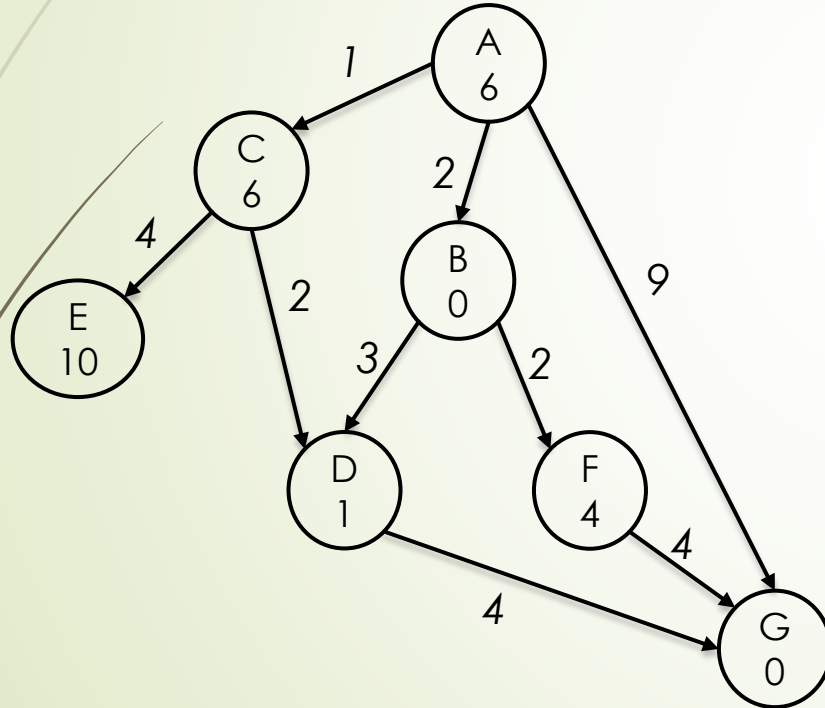
(۳) A, B, D, C, D, G ✓

(۴) A, C, D, B, F, G



تست

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؟



(۱) فقط سازگار است.

(۲) فقط قابل قبول است. ✓

(۳) هم قابل قبول است هم سازگار.

(۴) نه قابل قبول است نه سازگار.

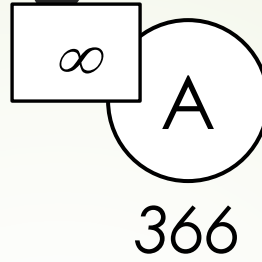
جستجوی اول بهترین بازگشتی – RBFS

Recursive Best-First Search

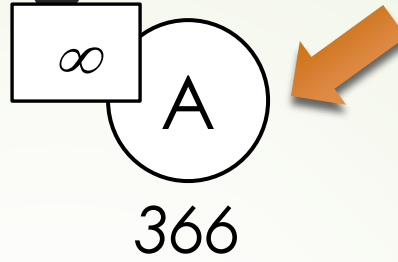
ساختاری شبیه به جستجوی عمقی بازگشتی دارد اما به جای این که دائماً مسیر فعلی را به سمت پایین ادامه دهد، مقدار f بهترین مسیر جانشین از طریق اجداد گره فعلی را نگه می‌دارد. اگر f گره فعلی از این حد تجاوز کند، الگوریتم به عقب برمی‌گردد تا مسیر جانشین را انتخاب نماید. در برگشت به عقب این الگوریتم مقدار f مربوط 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.

هنگام بسط یک گره، غیر از جانشین، نیازی به ذخیره سازی همزادها نیست، پدر میتواند مجدداً همزادها را تولید کند اگر لازم شود.

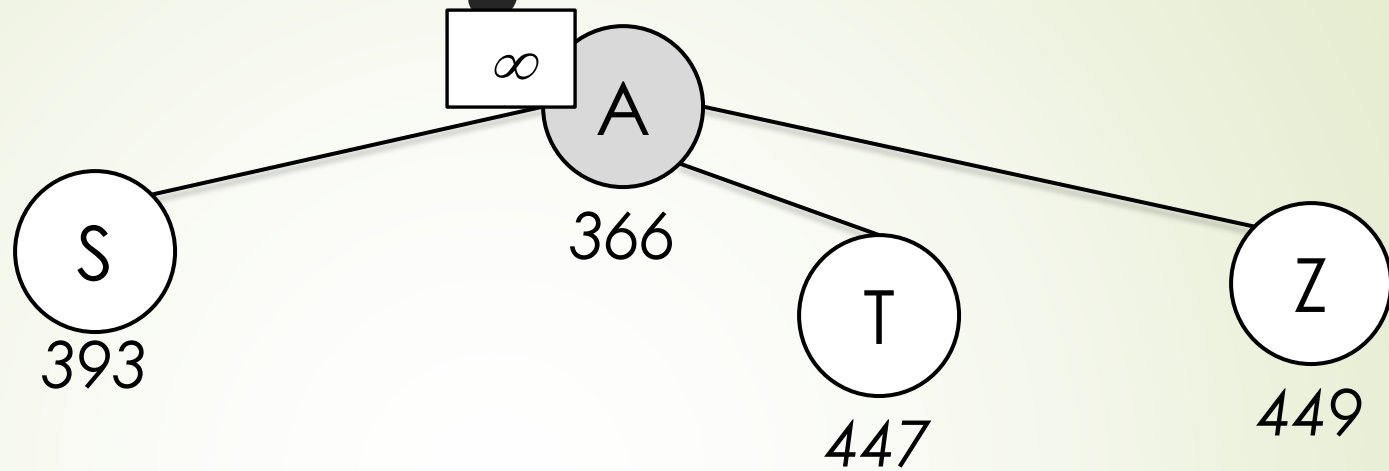
RBFS – مثال



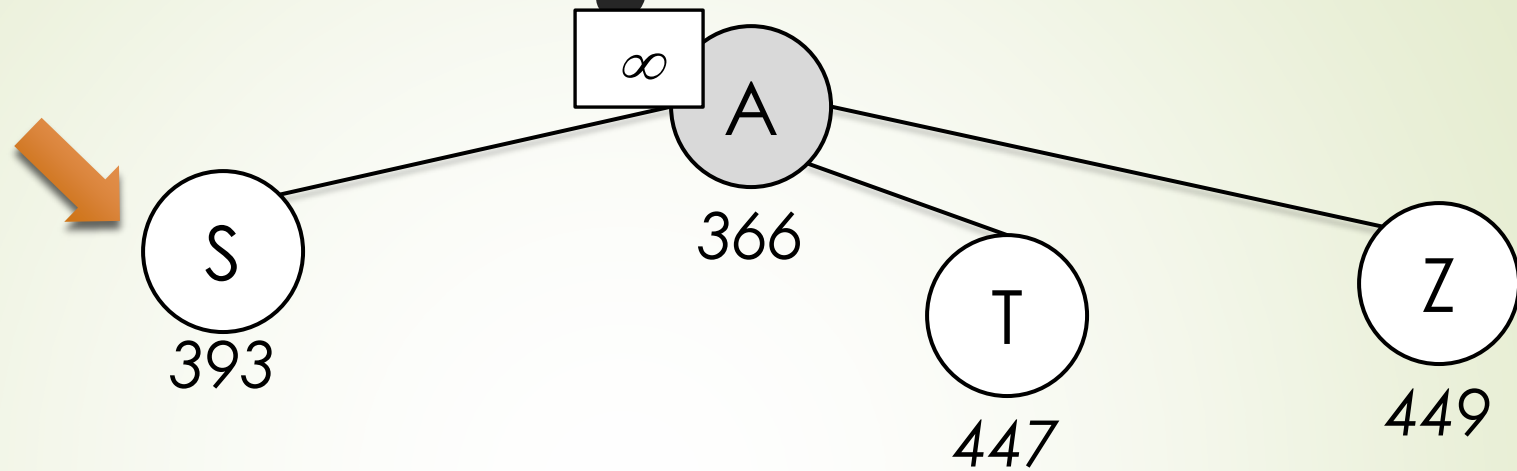
RBFS – مثال



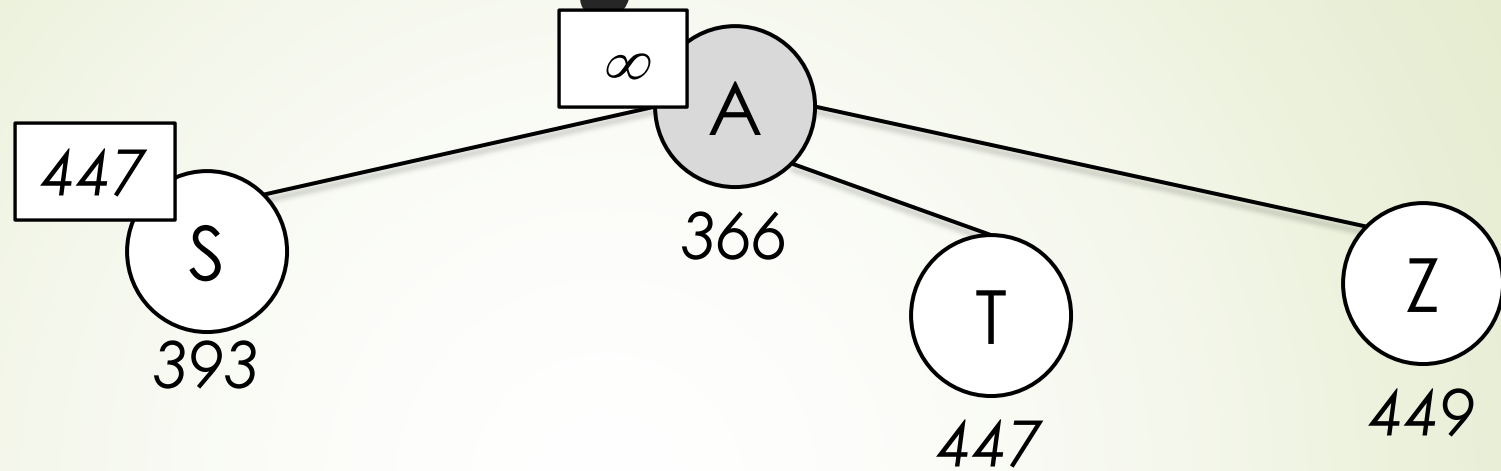
RBFS – مثال



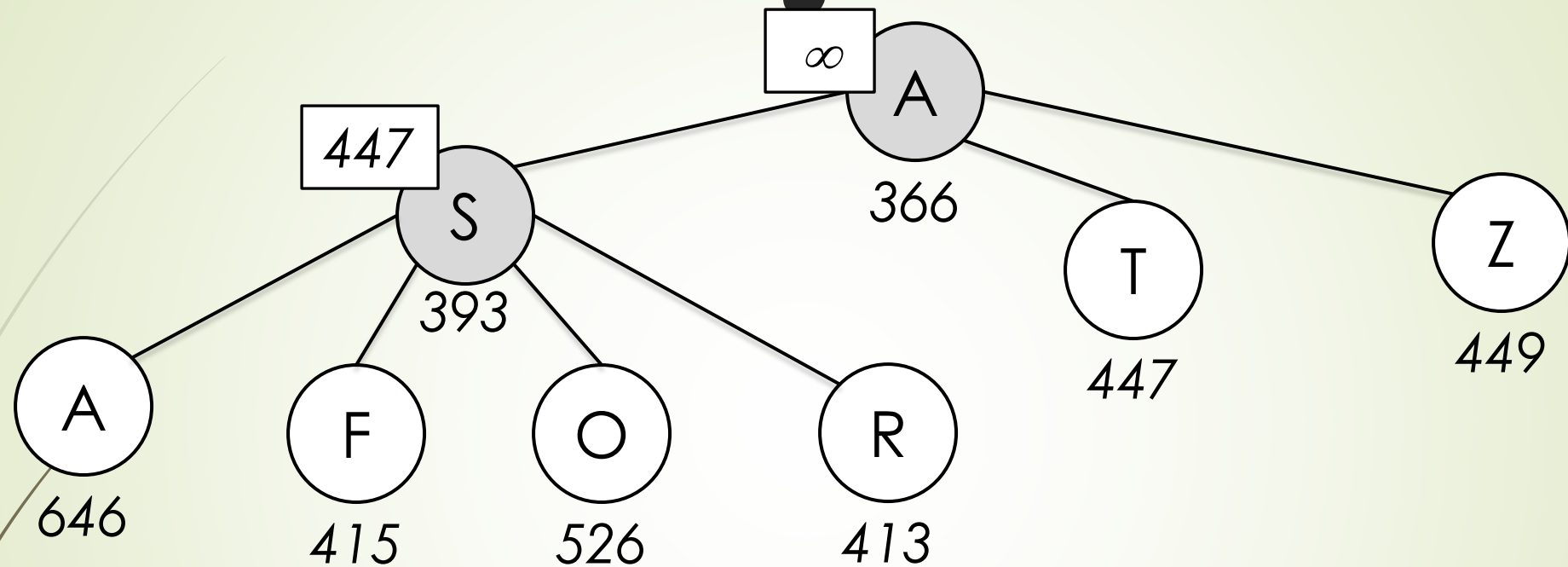
RBFS – مثال



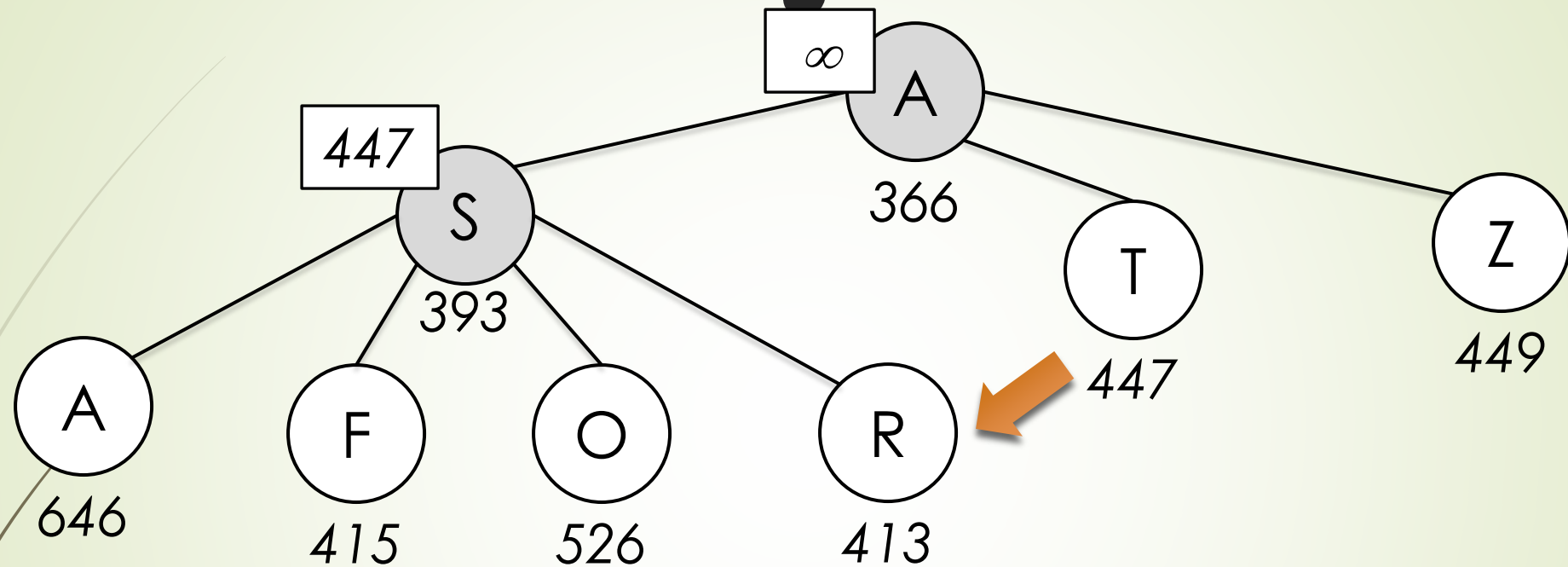
RBFS – مثال



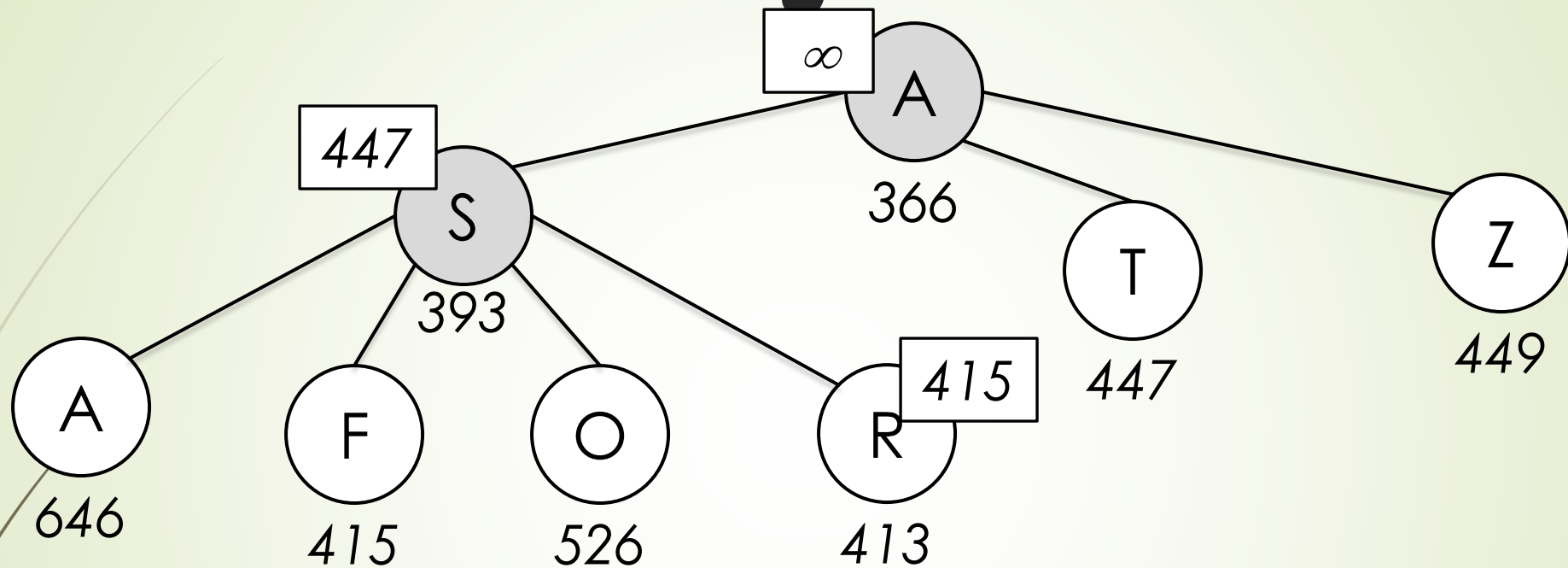
RBFS – مثال



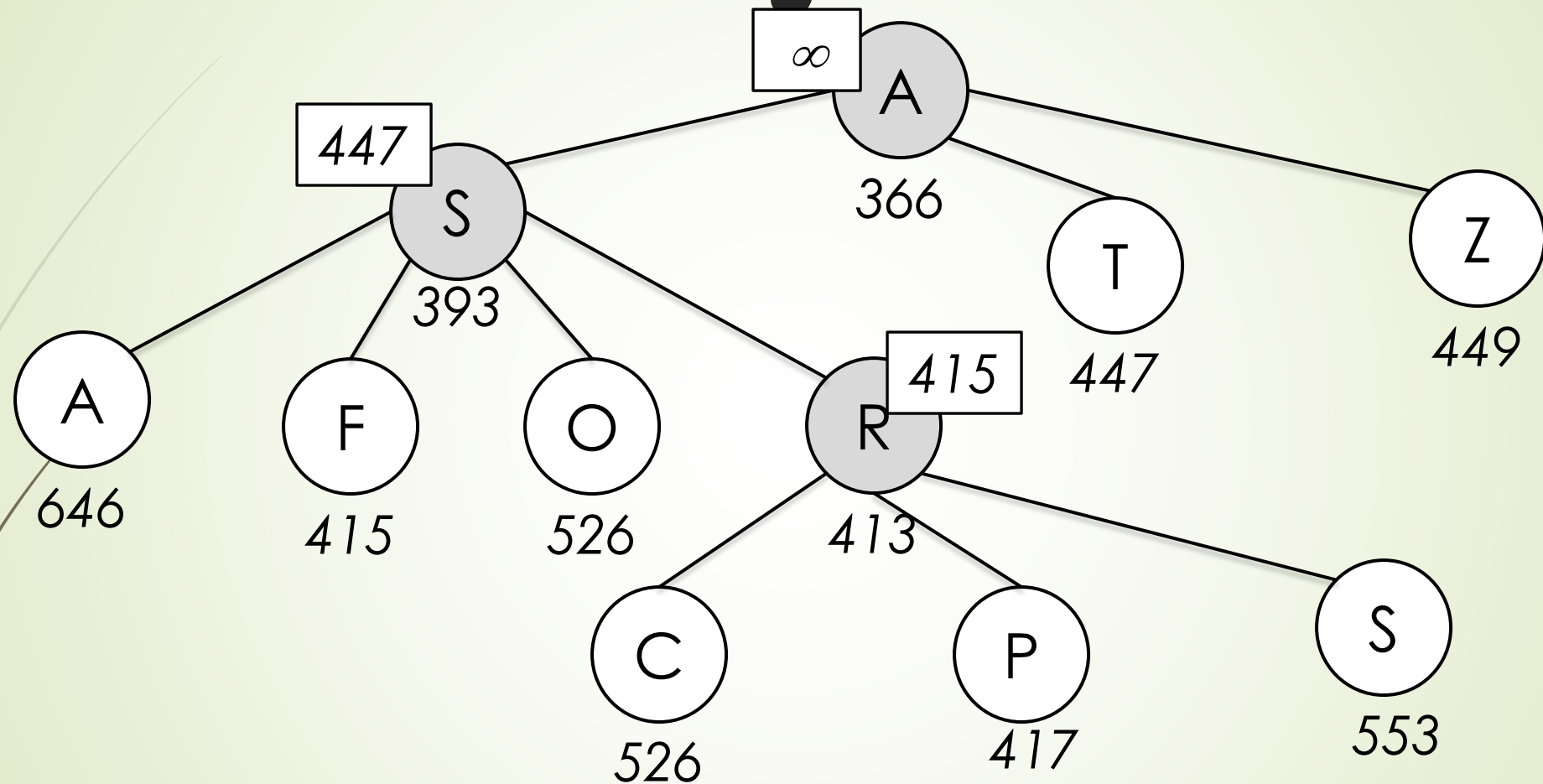
RBFS – مثال



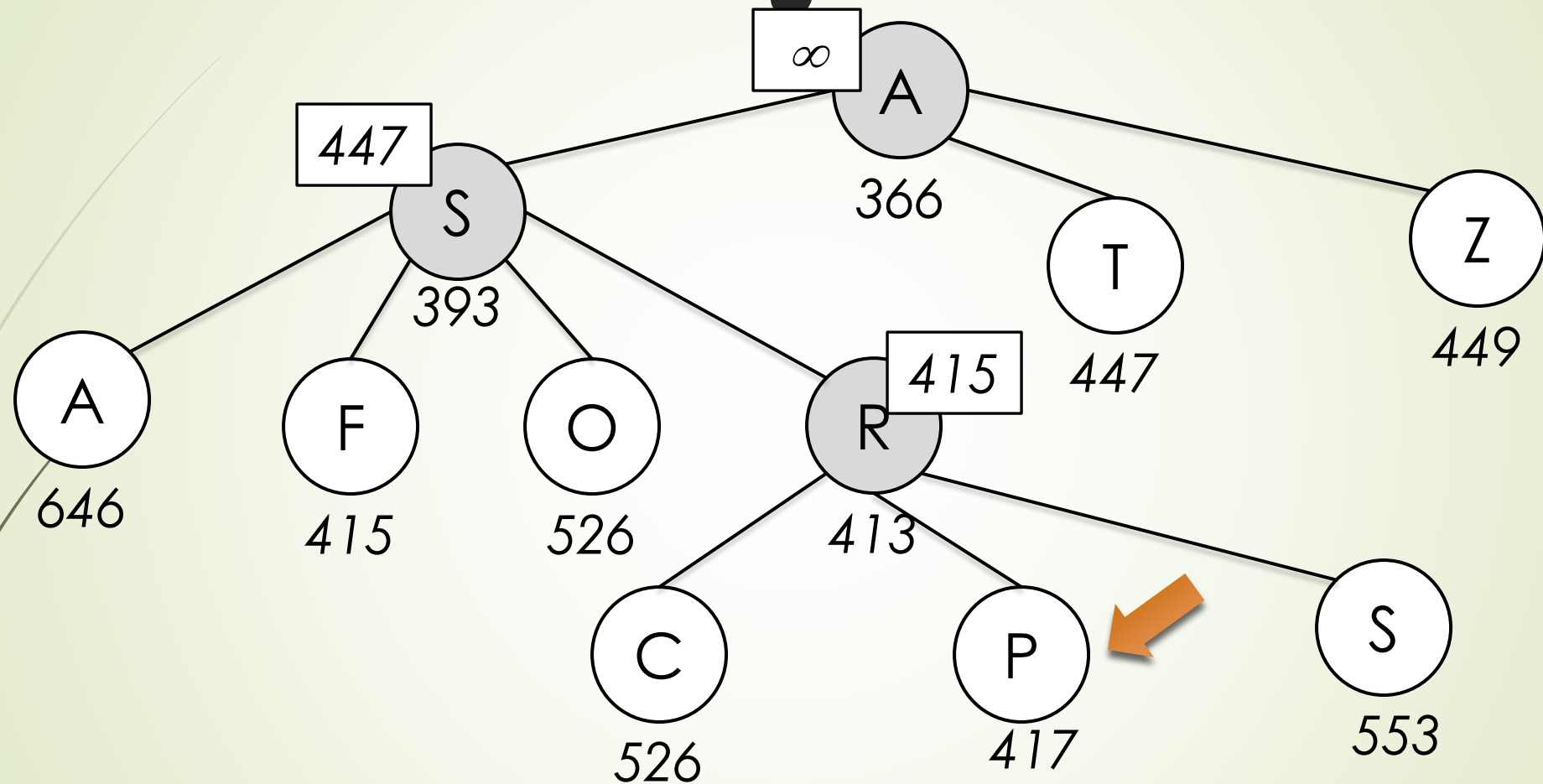
RBFS – مثال



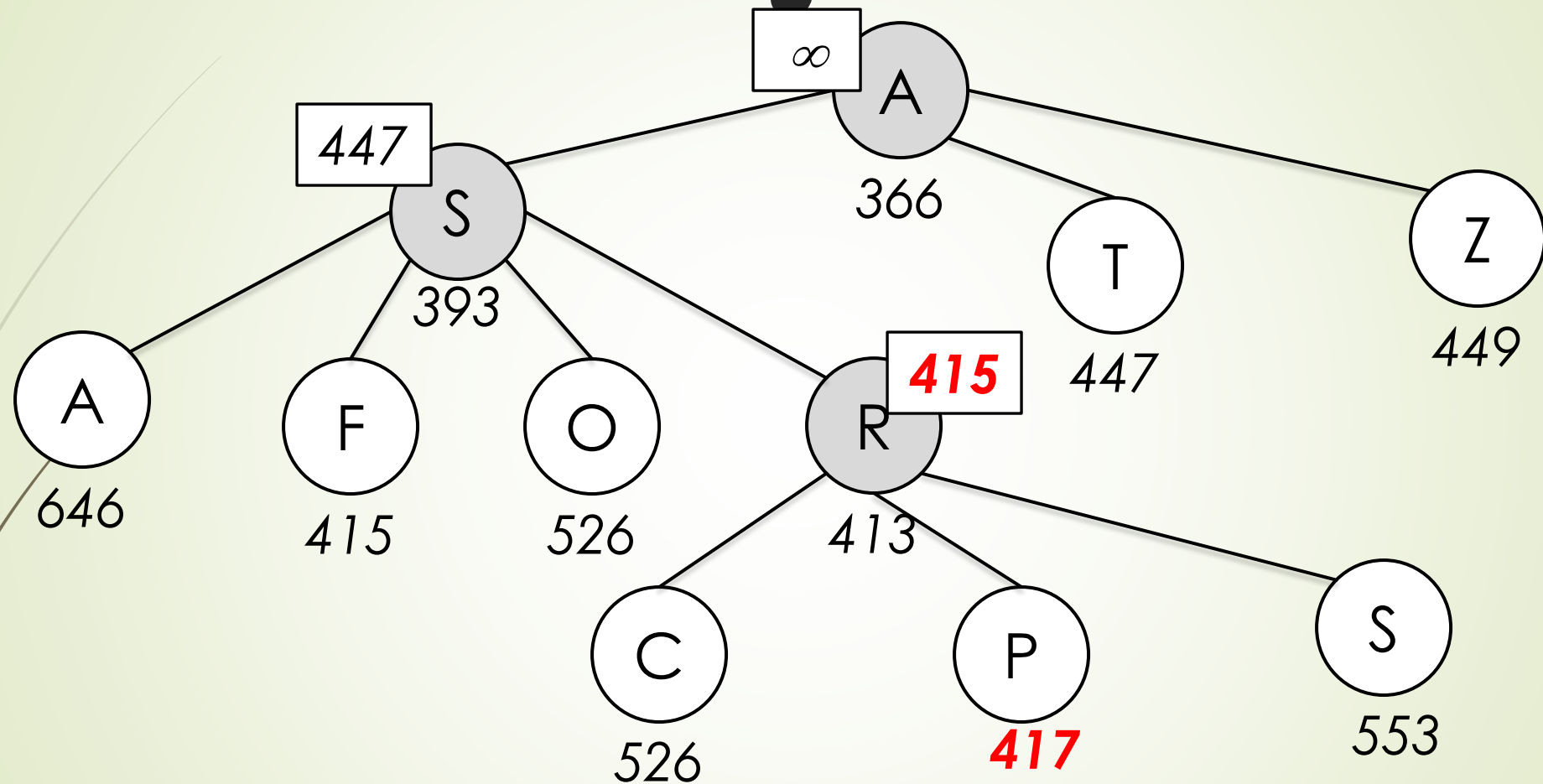
RBFS – مثال



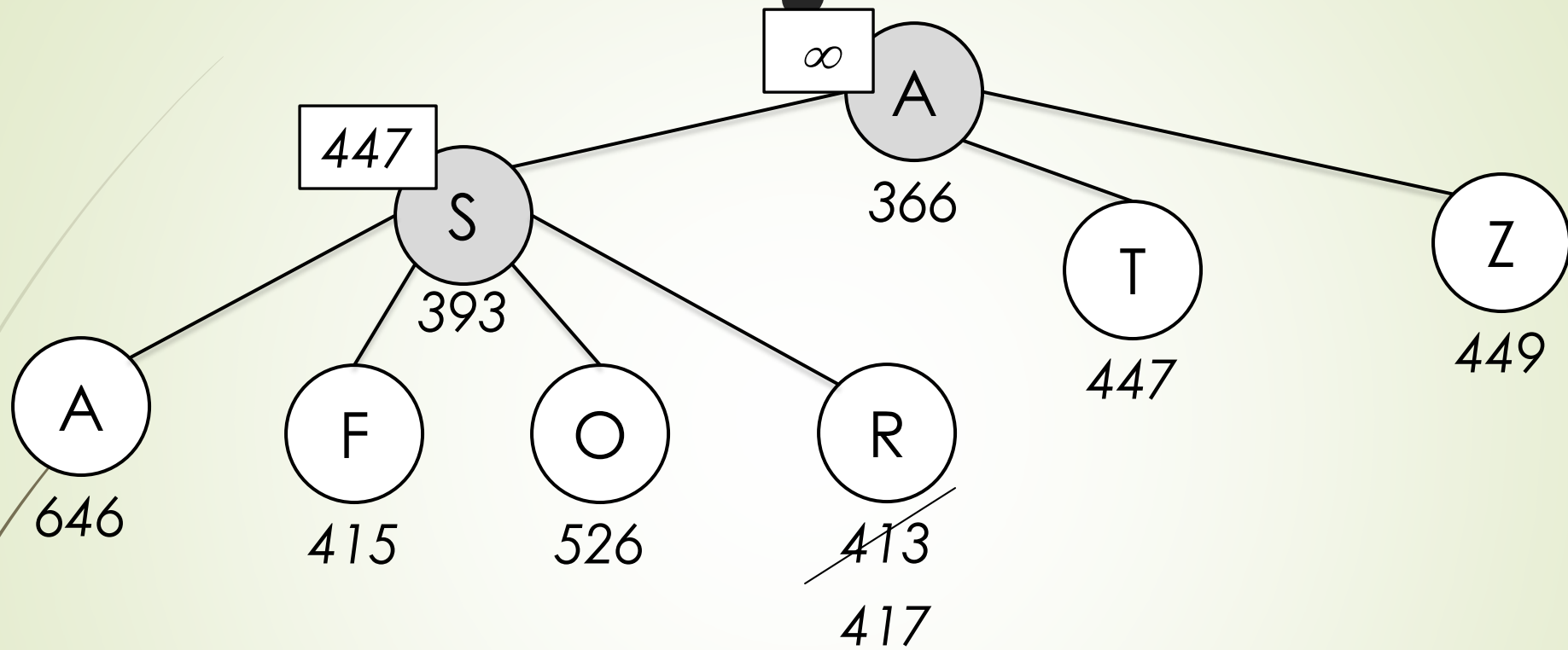
RBFS – مثال



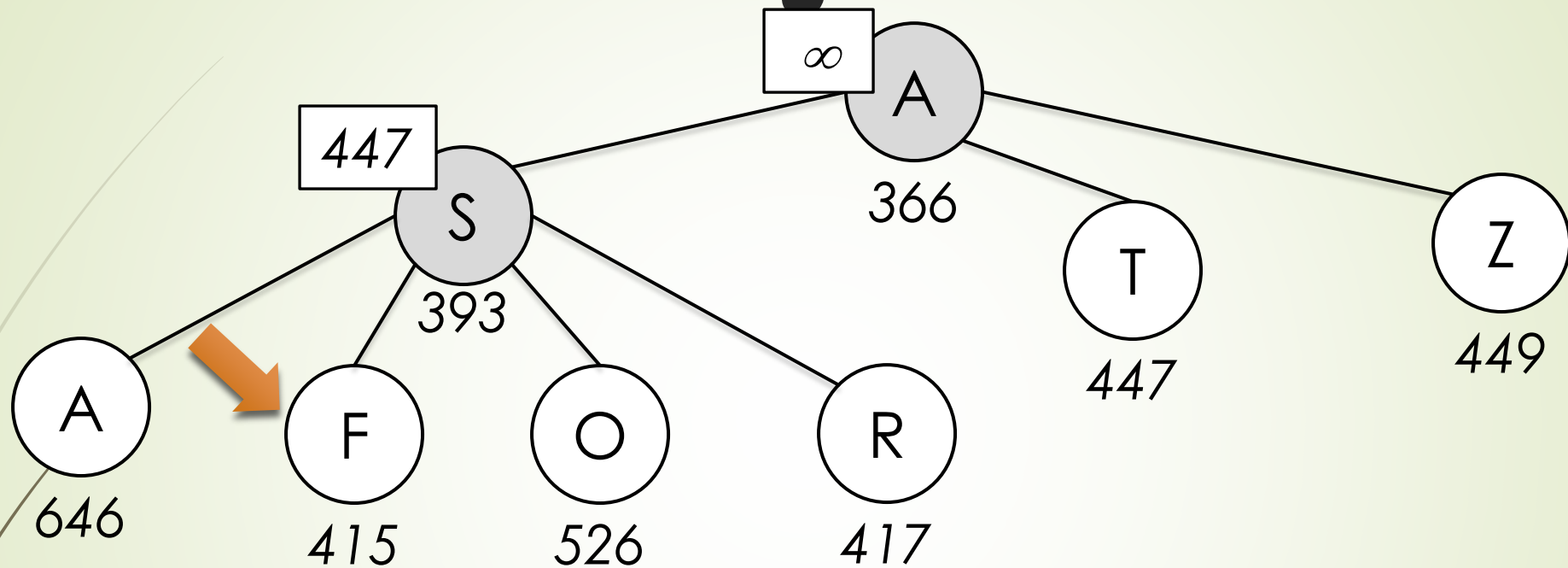
RBFS – مثال



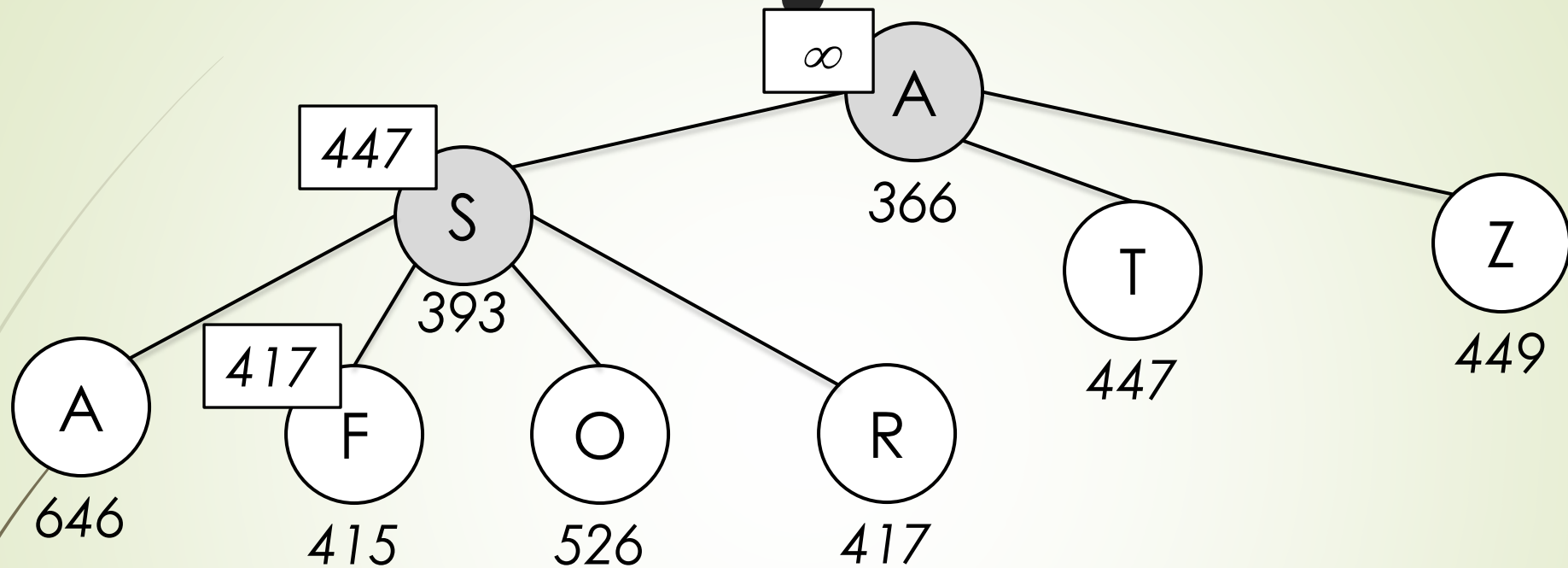
RBFS – مثال



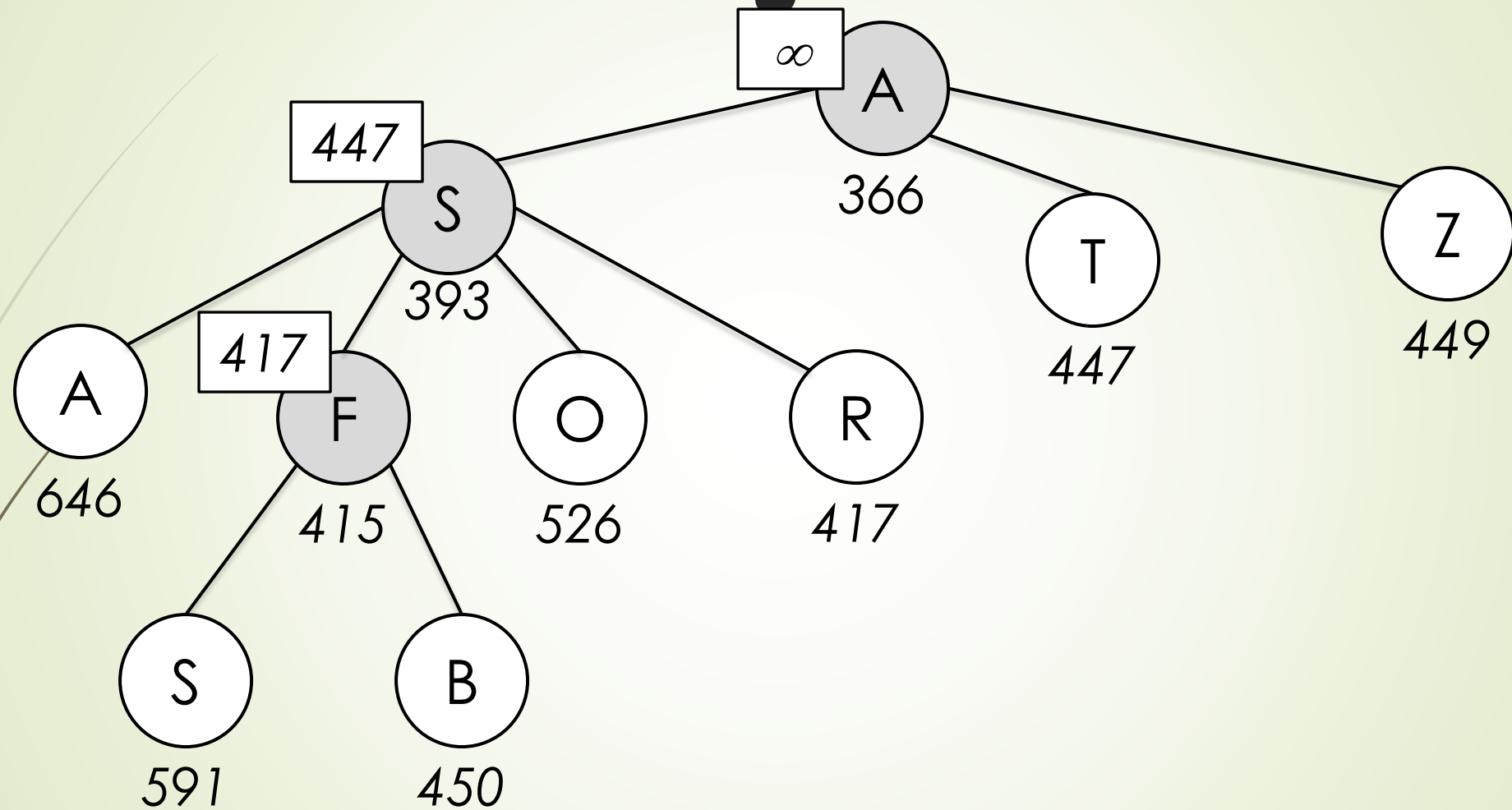
RBFS – مثال



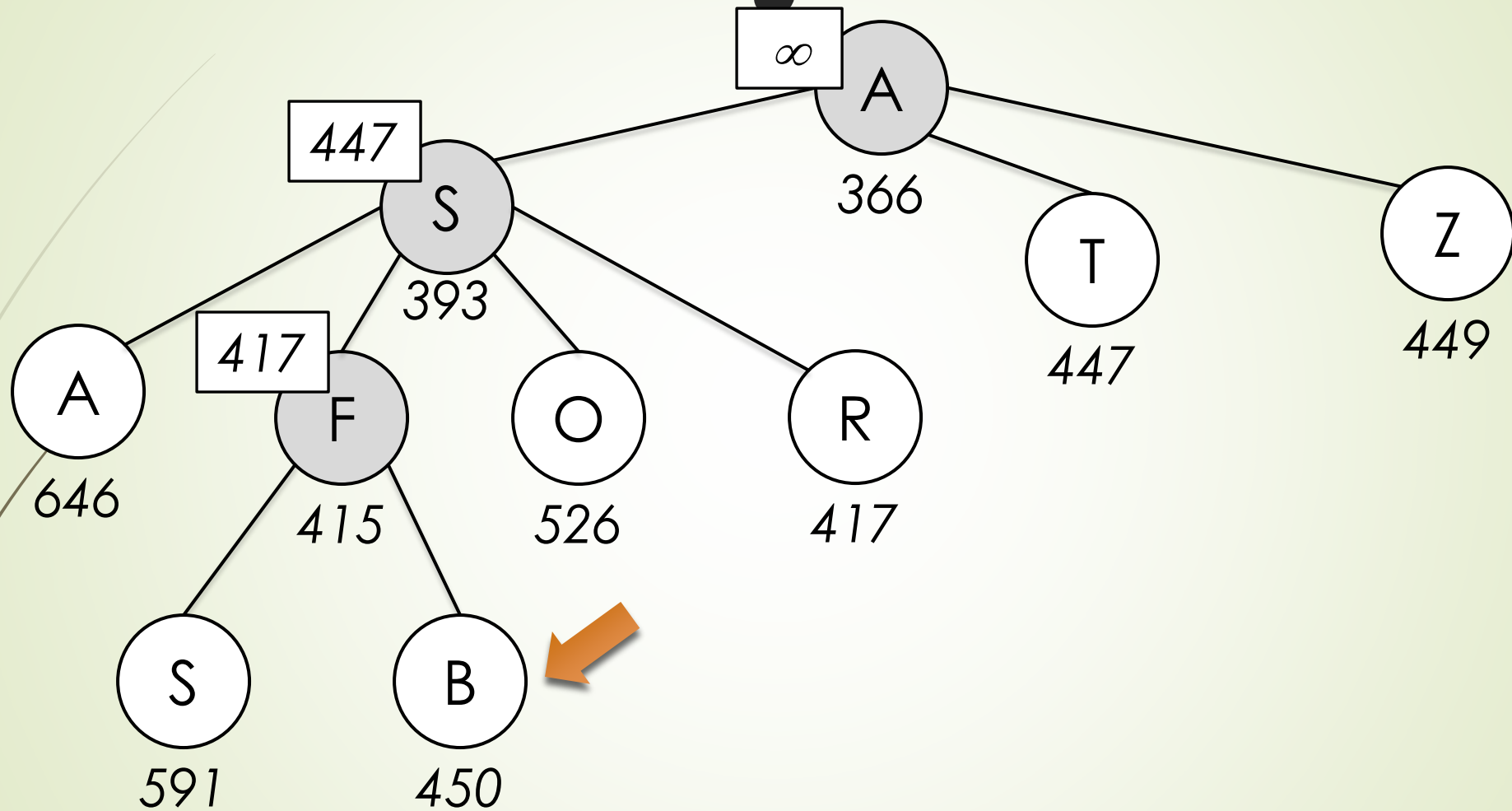
RBFS – مثال



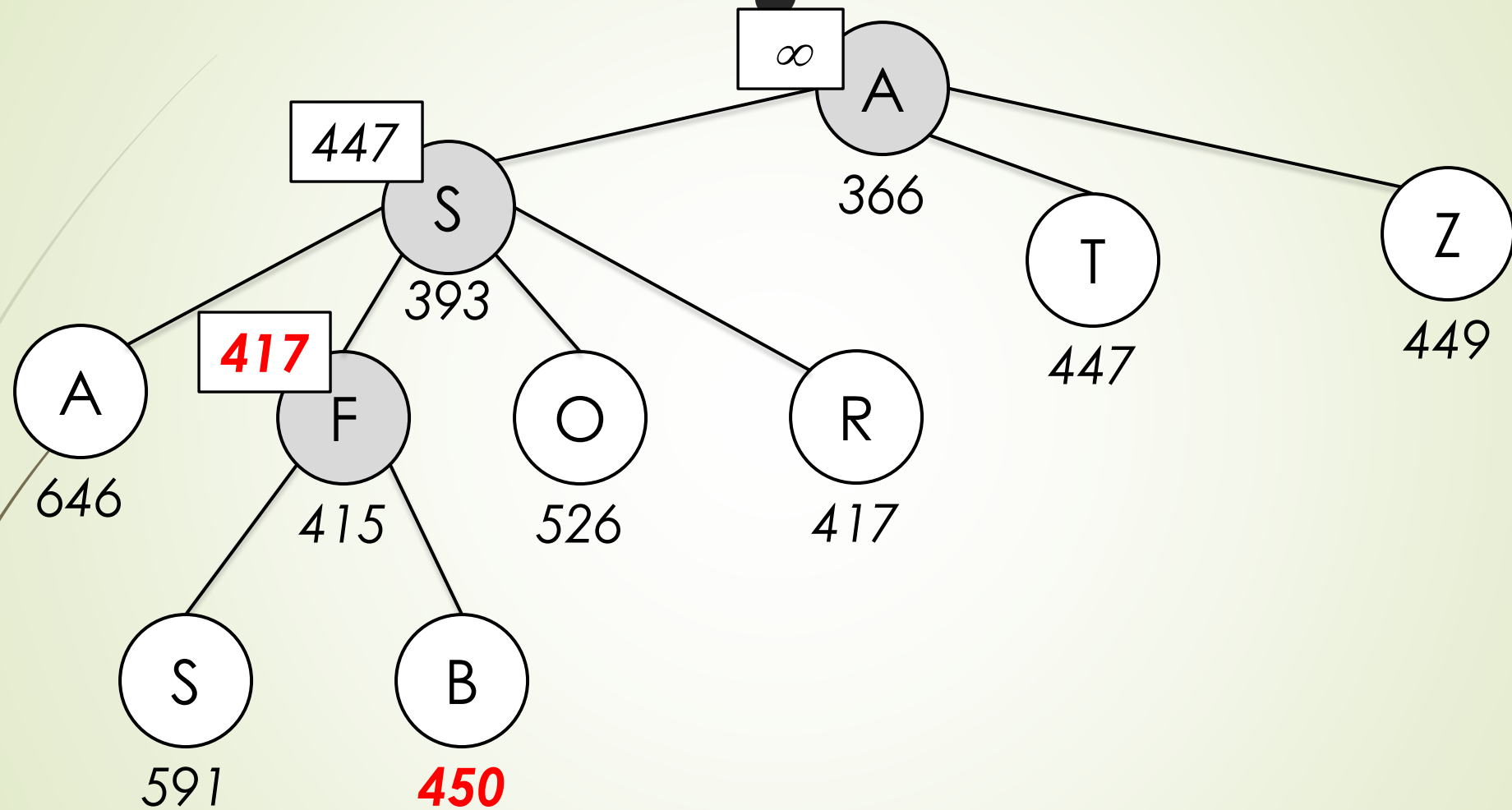
RBFS – مثال



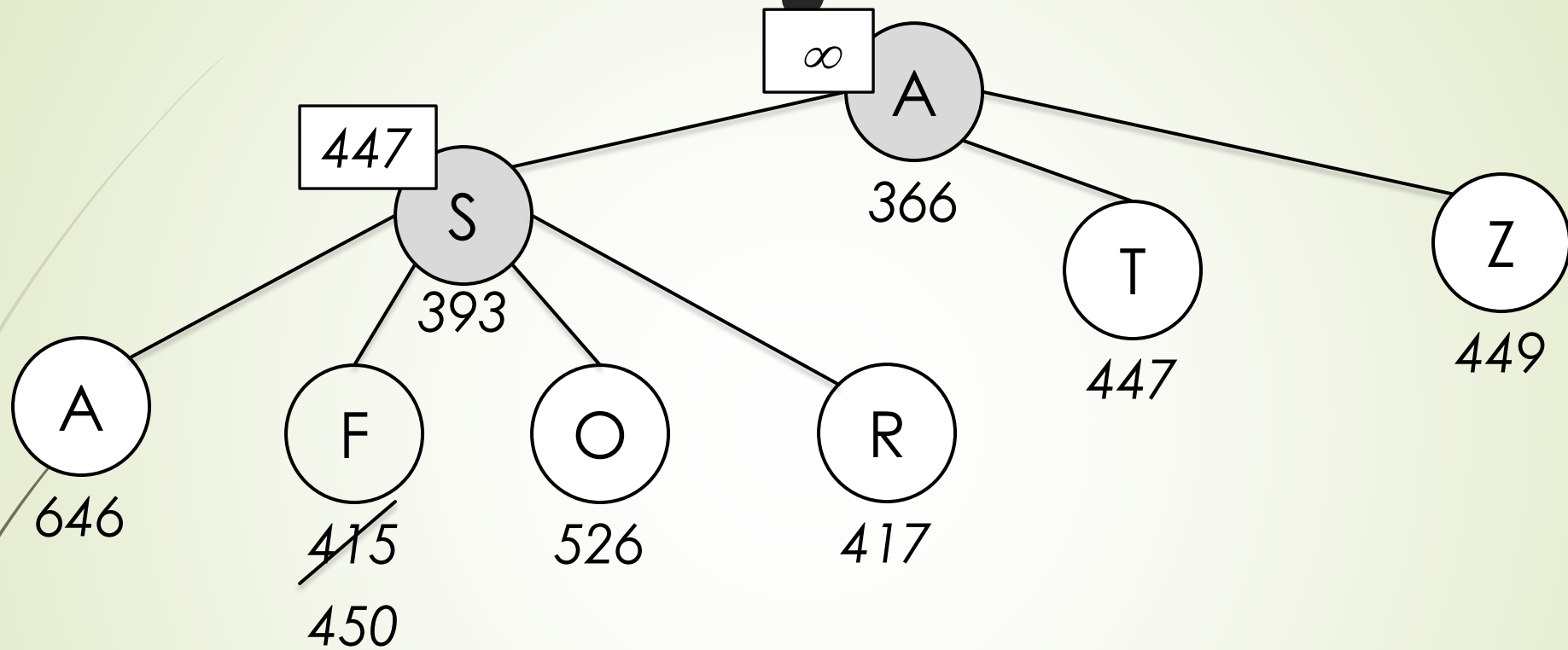
RBFS – مثال



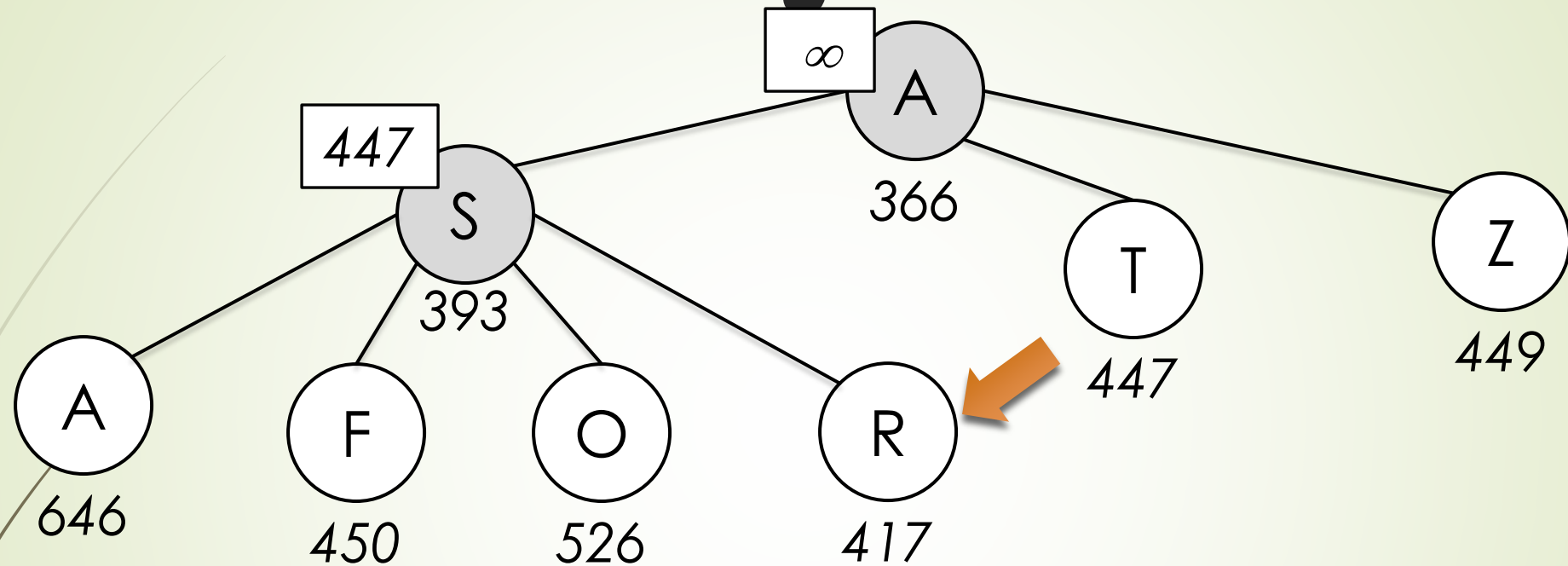
RBFS – مثال



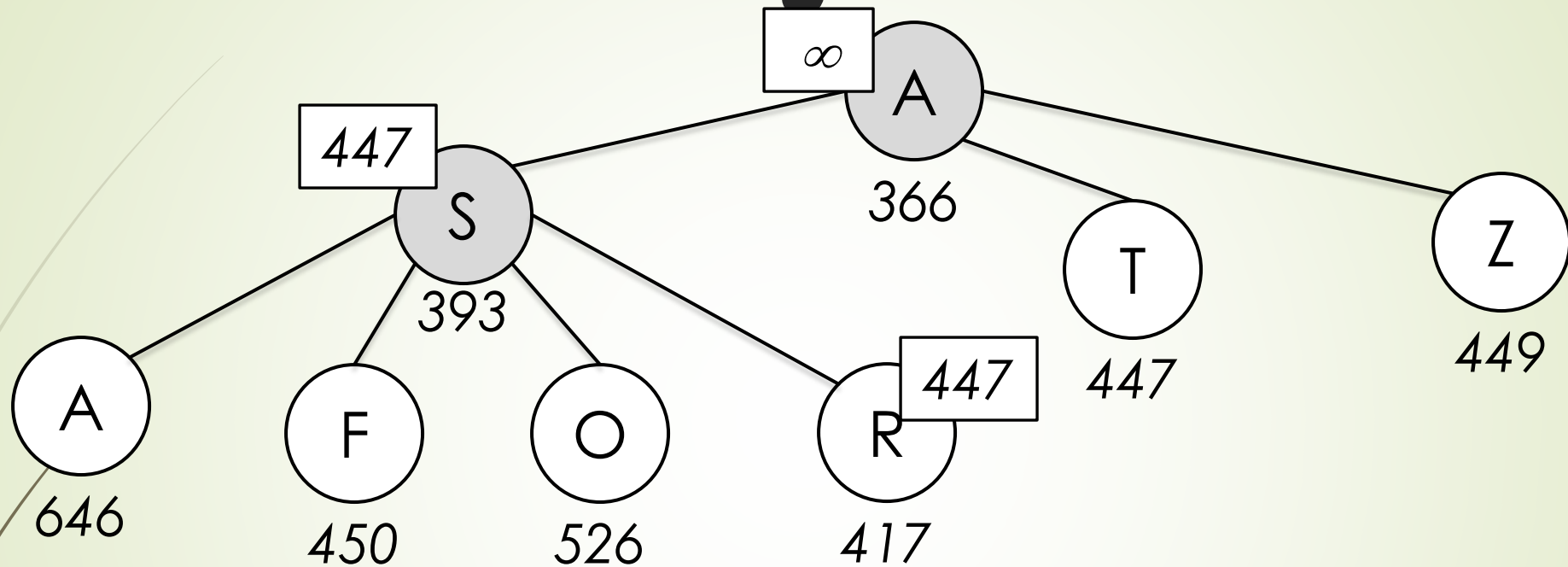
RBFS – مثال



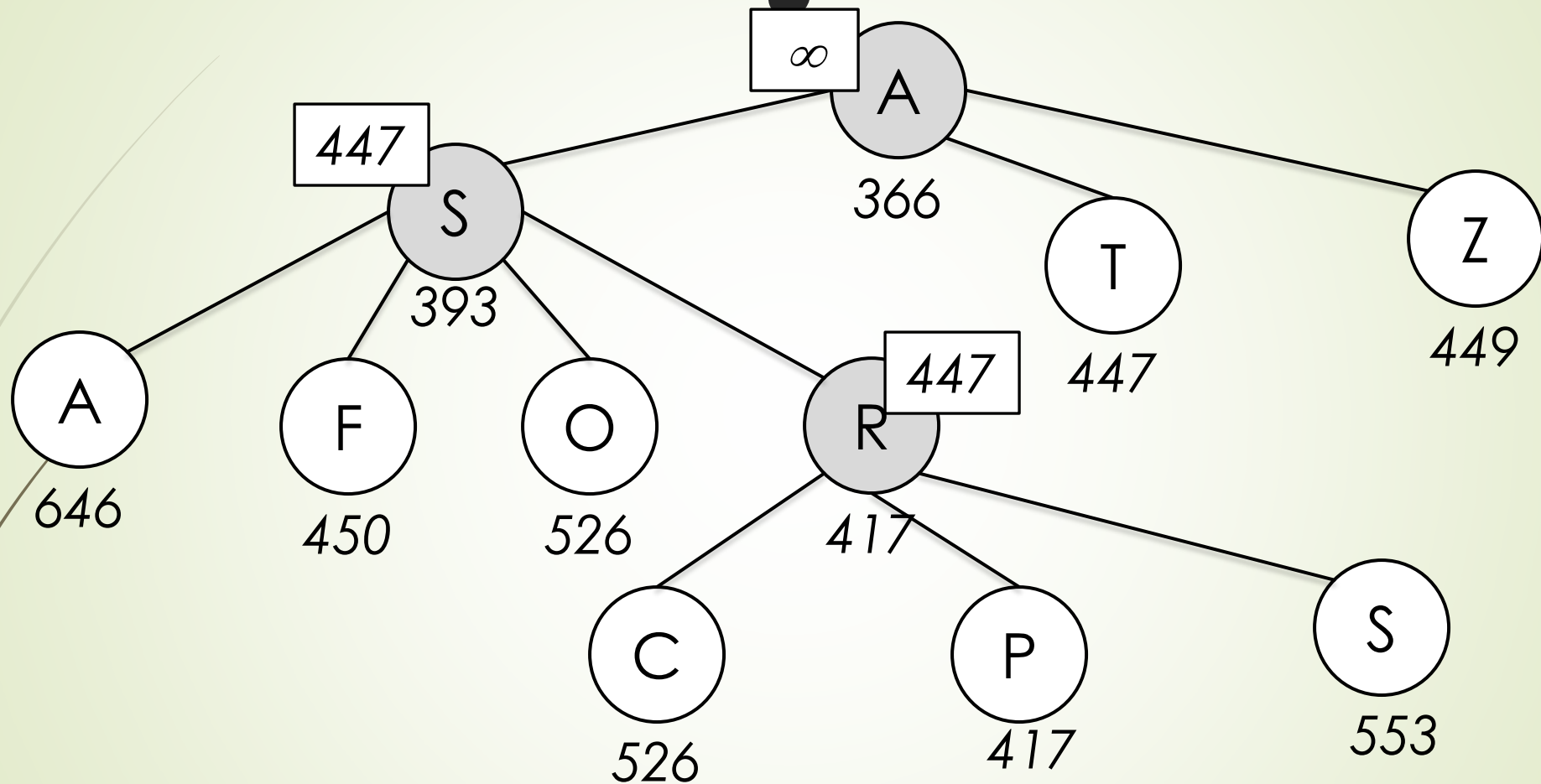
RBFS – مثال



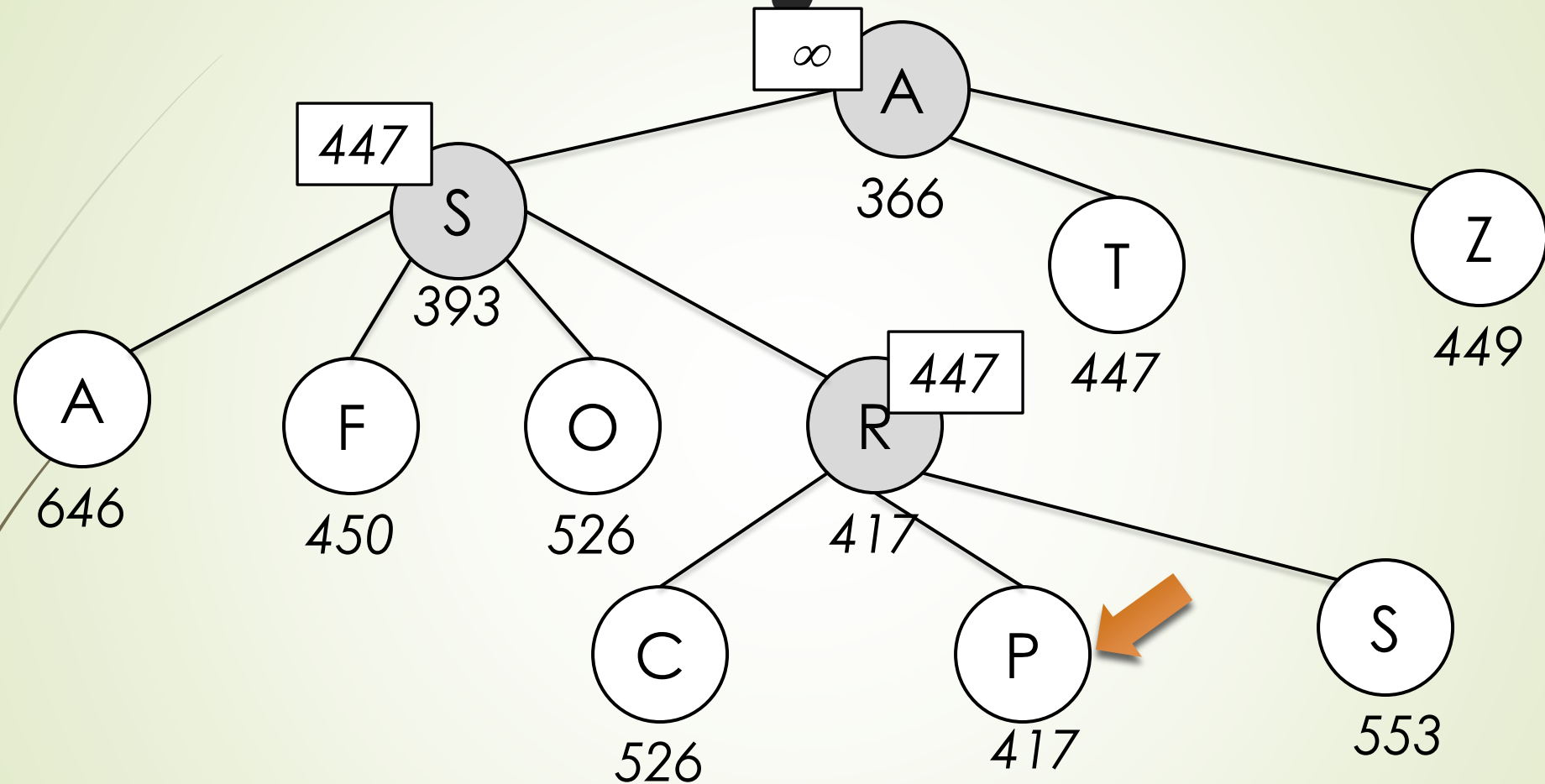
RBFS – مثال



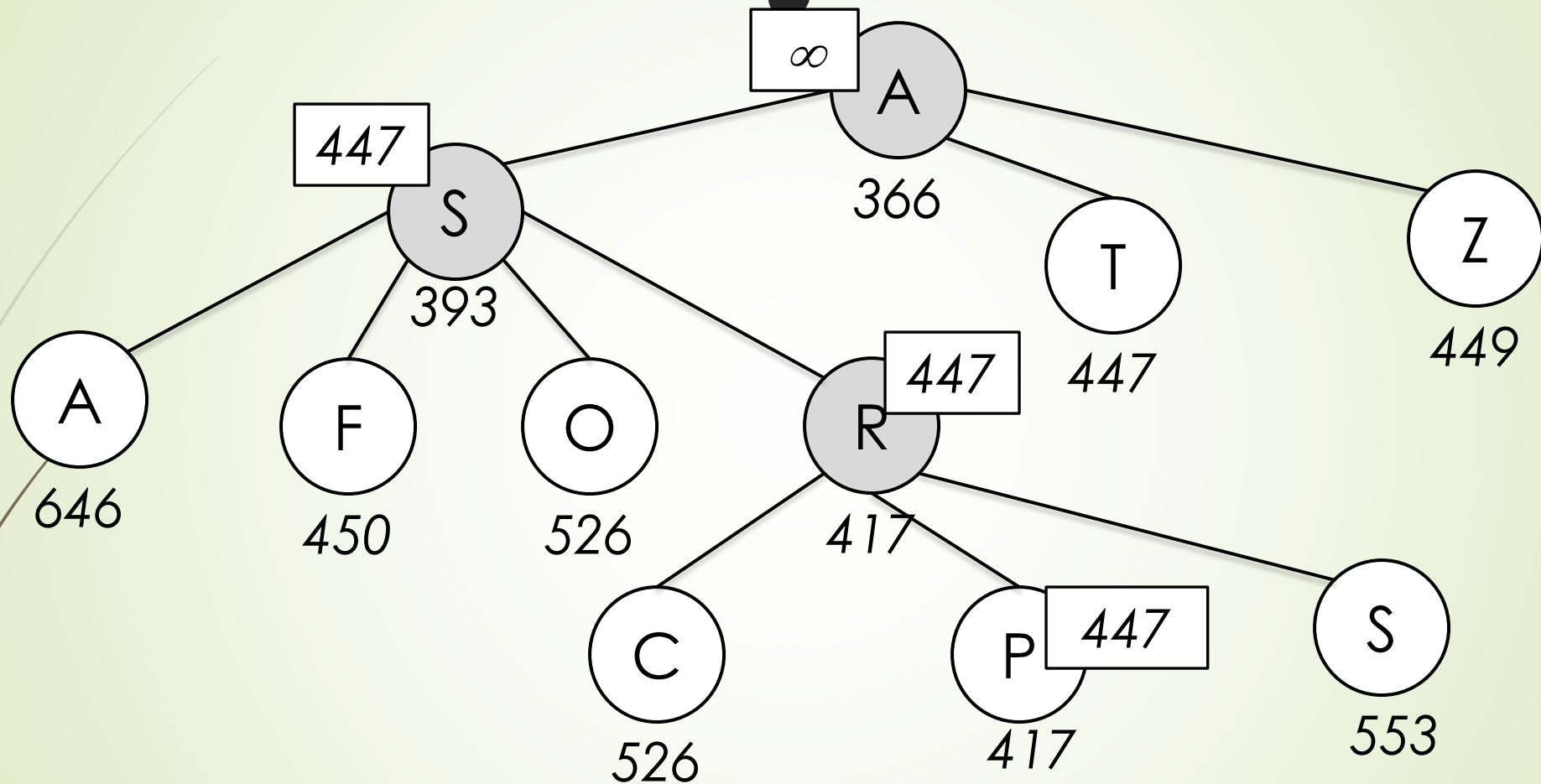
RBFS – مثال



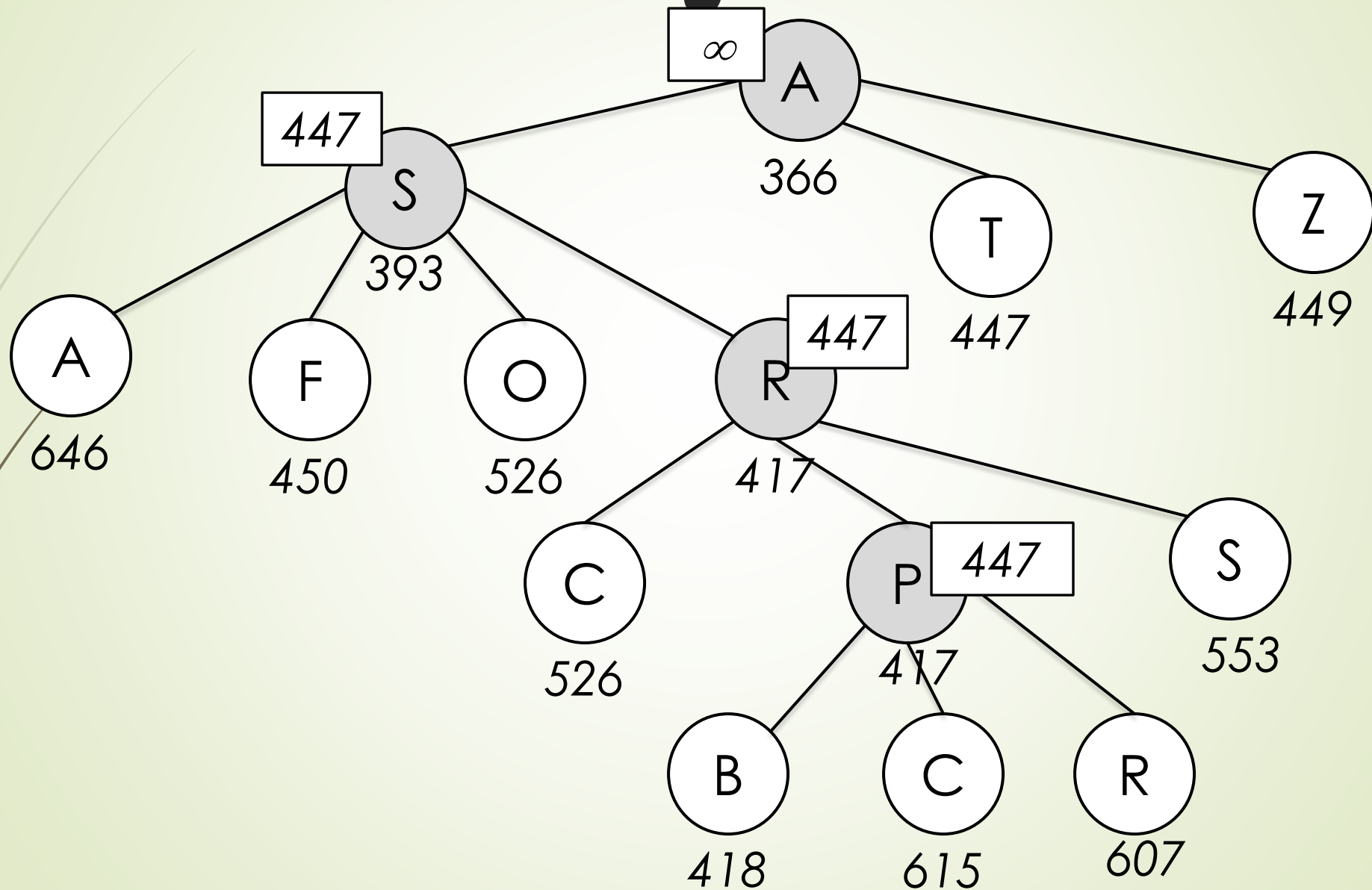
RBFS – مثال



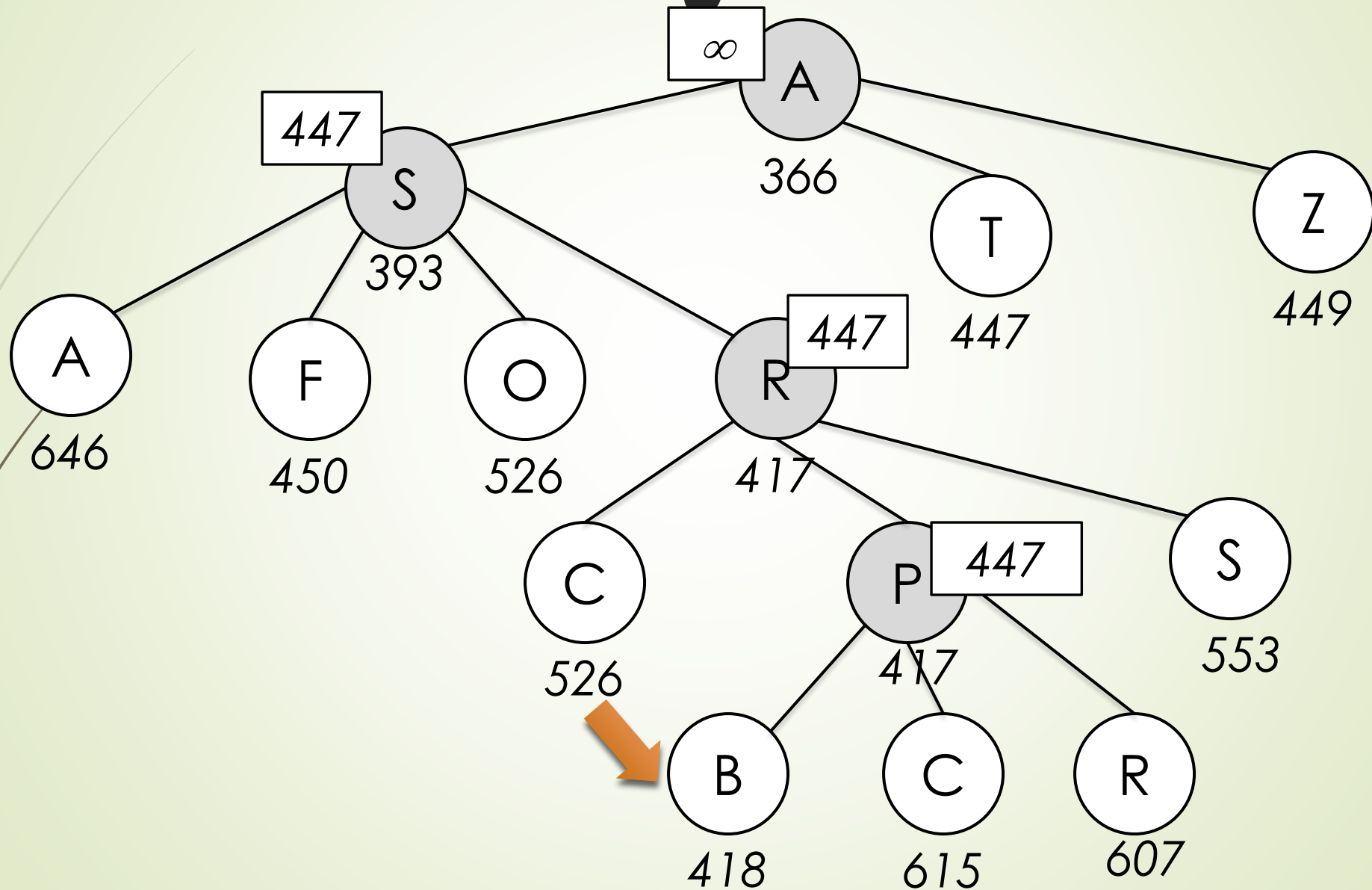
RBFS – مثال



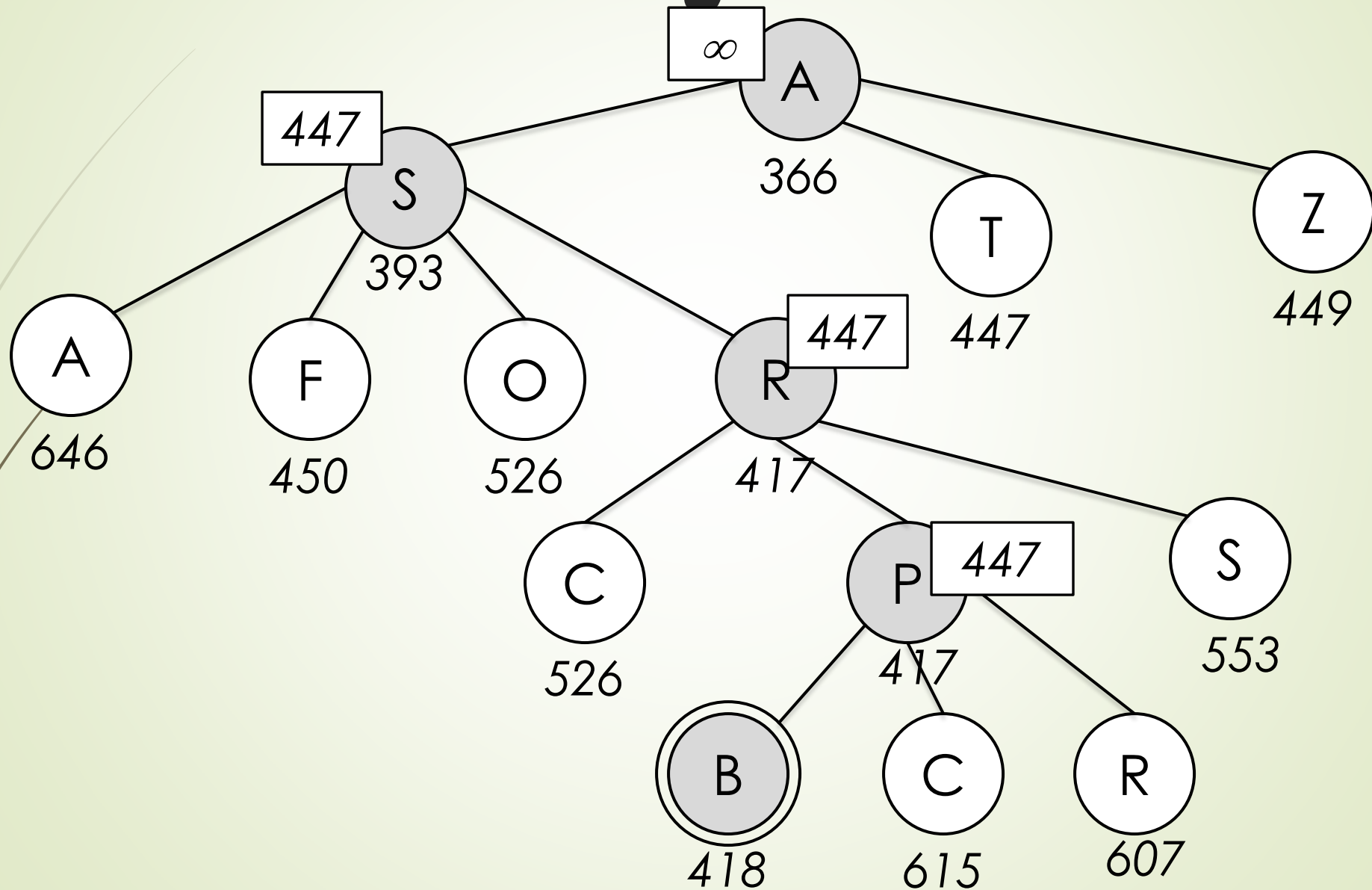
RBFS – مثال



RBFS – مثال



RBFS – مثال



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 \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, ∞

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*

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!!