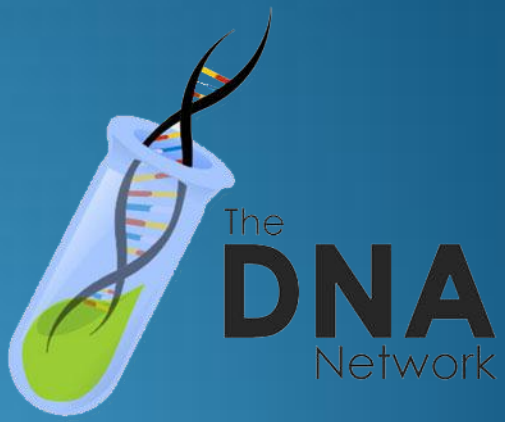

# هوش مصنوعی

مدرس : احمد ابدالی

# فصل چهارم

## جستجوی آگاهانه یا جستجوی هیورستیک




Ahmad Abdali Mohamadi

# فصل چهارم

جستجوی آگاهانه چیست ؟

تابع ارزیابی و تابع هیورستیک

الگوریتم های جستجوی آگاهانه

الگوریتم های جستجوی محلی

الگوریتم پرندگان

الگوریتم مورچگان

الگوریتم ژنتیک

الگوریتم زنبورها

# توابع مهم این فصل

تابع هزینه مسیر $g(n)$ : هزینه مسیر از گره اولیه تا گره $n$

تابع اکتشافی $h(n)$ : هزینه تخمینی ارزان ترین مسیر از گره $n$ به گره هدف

تابع بهترین مسیر $h^*(n)$ : ارزان ترین مسیر از گره $n$ تا گره هدف

تابع ارزیابی $f(n)$ : هزینه تخمینی ارزان ترین مسیر از طریق $n$

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

$f^*(n)$ : هزینه ارزان ترین مسیر از طریق $n$

$$f^*(n): g(n) + h^*(n)$$

# جستجوی آگاهانه

استراتژي جستجوي آگاهانه از دانش مسئله استفاده مي كند و در انتخاب گره ، گرهي را انتخاب مي كنند كه شانس رسيدن به هدف در آن بيشتر باشد يا به نظر برسد كه به هدف نزديك تراست .

براي اينكه تخمين بزنيم كه گره فرزند چقدر به هدف نزديك تر است از تابع ارزيابي استفاده مي كنيم. اين تابع هزينه رسيدن به گره هدف را تخمين مي زند و به عبارت ديگر ميزان مفيد بودن گره فعلي را باز مي گرداند.

تابع ارزيابي را با $f(n)$ نشان می دهند

تابع هيورستيک را با $h(n)$ نشان می دهند

# انواع استراتژی های جستجوی آگاهانه

جستجوی اول بهترین
- ۱) جستجوی حریصانه
- ۲) جستجوی A*

جستجوی حافظه محدود شده
- ۱) IDA*
- ۲) MA*
- ۳) SMA*

جستجوی محلی
- ۱) تپه نوردی
- ۲) سرد و گرم
- ۳) ژنتیک
- ۴) پرتو

# جستجوی اول بهترین

در این روش در هر مرتبه گره اي كه بهترين ارزيابي را داشته باشد ابتدا بسط داده مي شود به عبارت ديگر گرهي انتخاب مي شود كه تابع ارزيابي بهترين مقدار را براي آن باز گرداند.

جستجوی اول بهترین با صف اولویت پیاده سازی می شود .

در جستجوی اول بهترین اگر تابع ارزیابی غیر صحیح باشد می تواند باعث گمراه شدن جستجو شود .

جزء کلیدی جستجوی اول بهترین تابع هیورستیک ( اکتشافی ) می باشد

# تابع هیورستیک در الگوریتم جستجوی اول بهترین

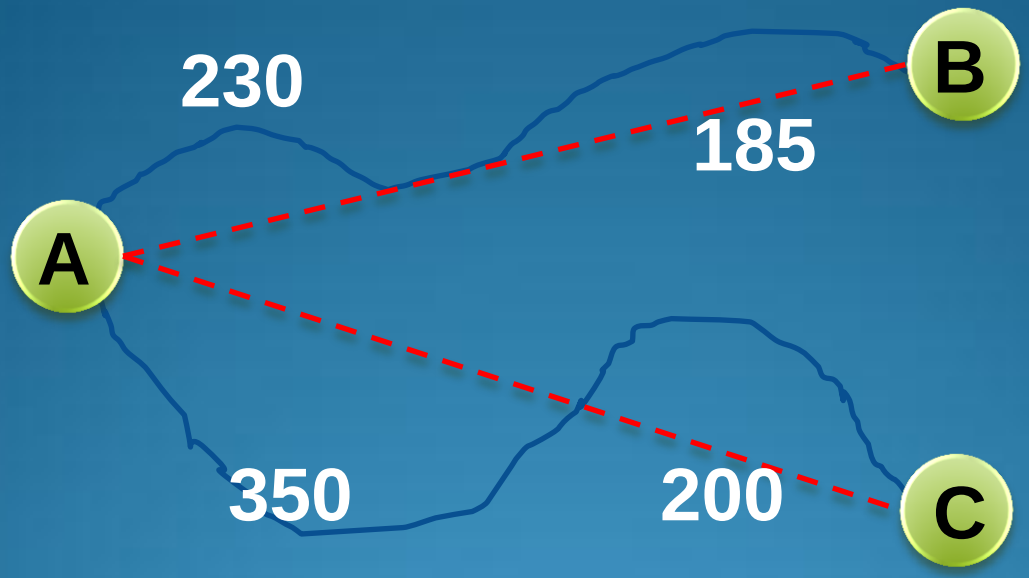
جزء کلیدی جستجوی اول بهترین تابع هیورستیک ( اکتشافی ) می باشد

هزینه تخمینی ارزانترین مسیر از حالت $n$ تا نود هدف $= h(n)$

اگر $n$ هدف باشد داریم : $h(n) = 0$

# یک مثال از تابع هیورستیک

در یک مسله مسیر یابی داریم :




| State | h(n) |
|---|---|
| B | 185 |
| C | 200 |

# جستجوی اول بهترین با استراتژی حریصانه (Greedy Search)

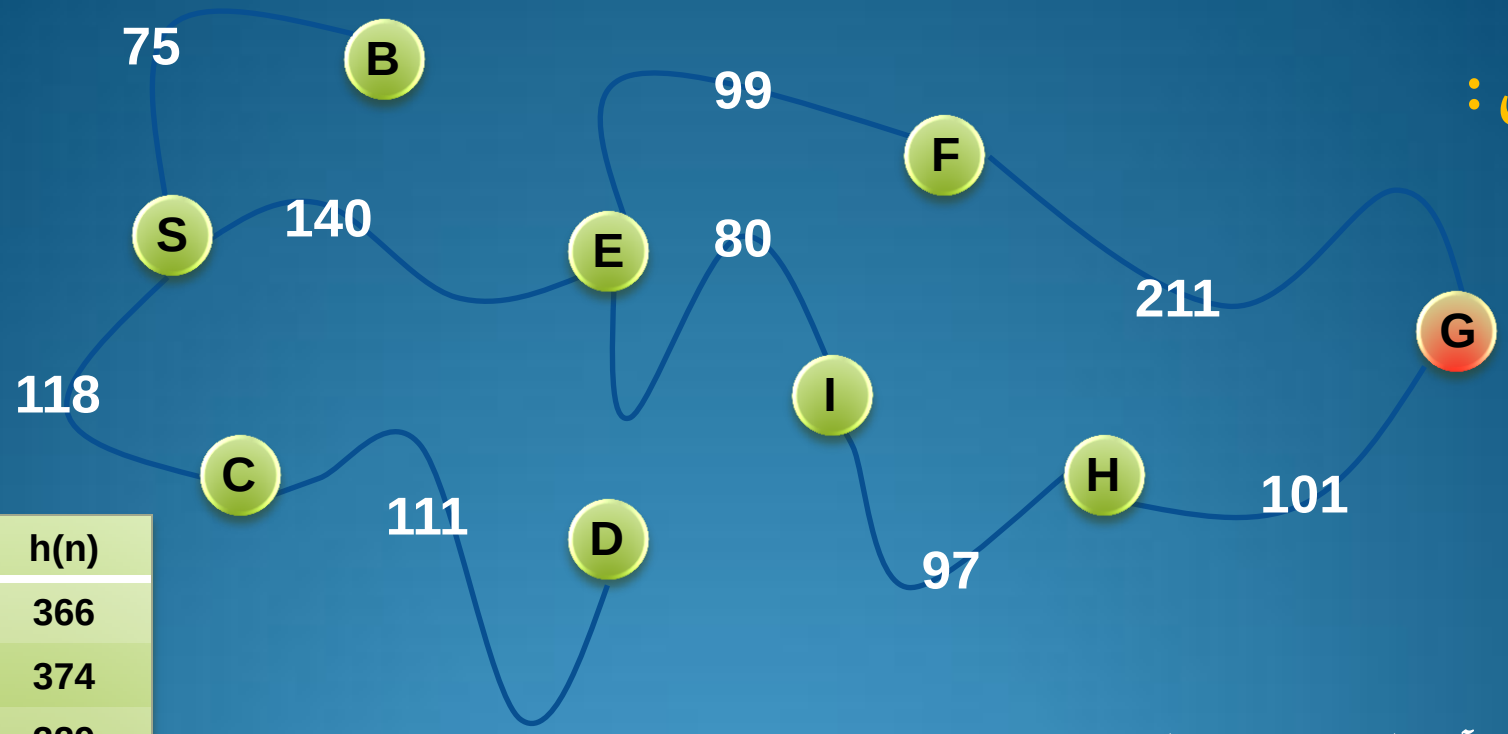
جستجوي حريصانه يكي از روش هاي جستجوي اول بهترين است در اين روش هدف به حداقل رساندن هزينه رسيدن به هدف با استفاده از تابع تخمين ( هيورستيک ) مي باشد در اين استراتژی گرهي كه به هدف نزديكتر است ابتدا بسط داده مي شود

در جستجوی حریصانه داریم :

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

# مسیر یابی با استراتژی حریصانه

مثال :




| State | h(n) |
|---|---|
| S | 366 |
| B | 374 |
| C | 329 |
| D | 244 |
| E | 253 |
| F | 178 |
| I | 193 |
| H | 98 |
| G | 0 |

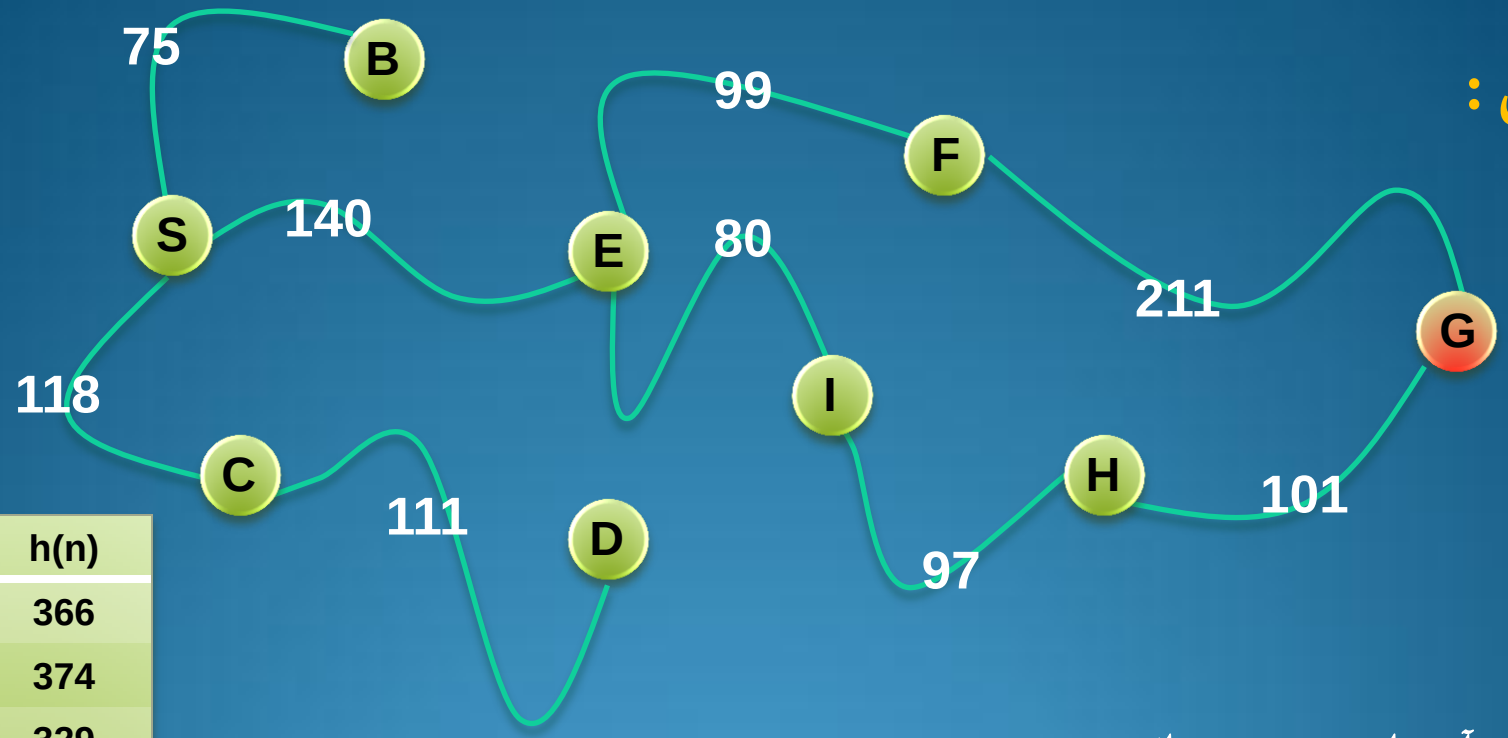
مسیر بدست آمده با جستجوی حریصانه :

S-E-F-G = 140 + 99 + 211= 450

Ahmad Abdali Mohamadi

# مسیر یابی با استراتژی حریصانه بهینه نیست

مثال :




| State | h(n) |
|---|---|
| S | 366 |
| B | 374 |
| C | 329 |
| D | 244 |
| E | 253 |
| F | 178 |
| I | 193 |
| H | 98 |
| G | 0 |

مسیر بدست آمده با جستجوی حریصانه :

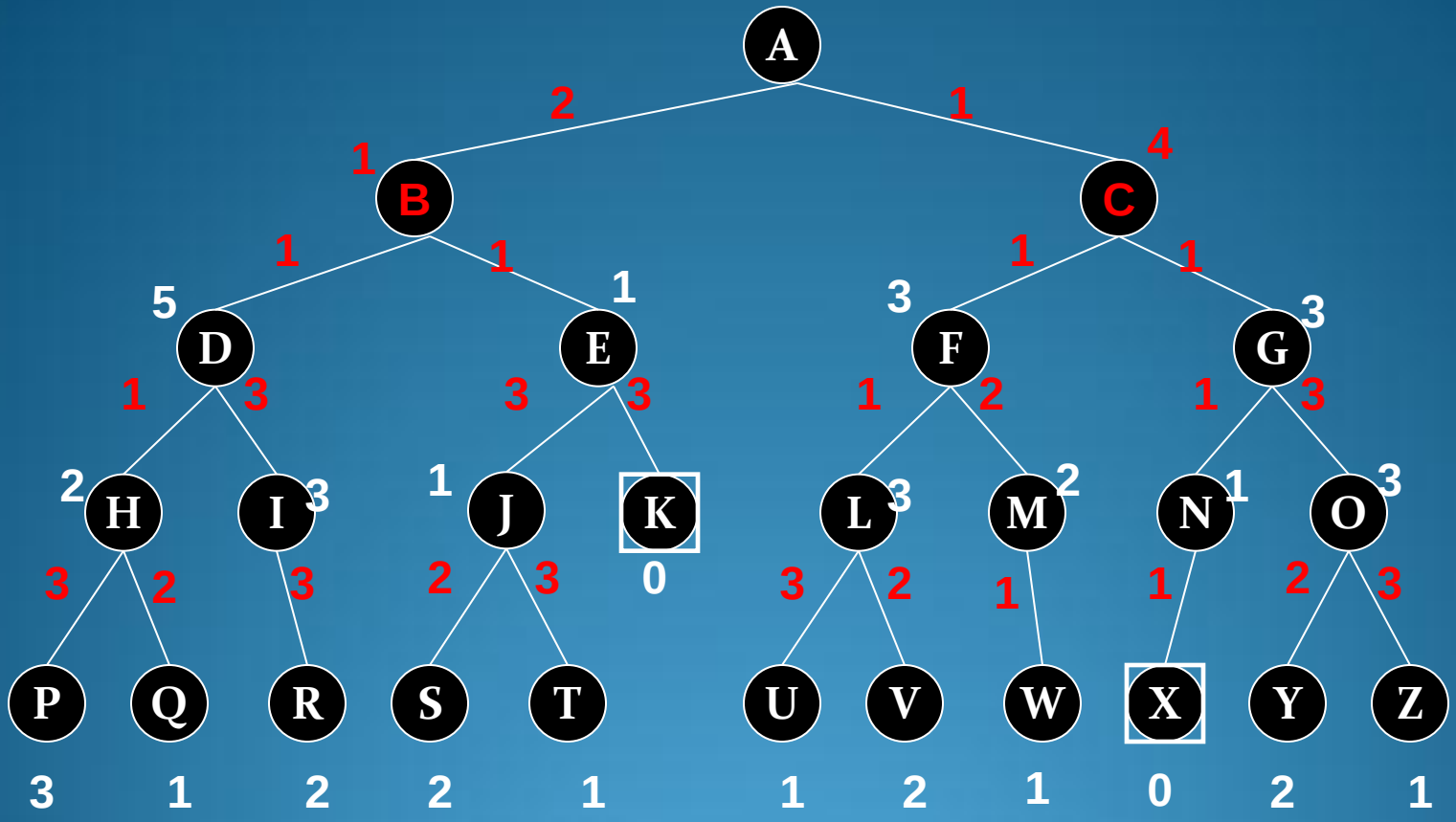
S-E-F-G = 140 + 99 + 211= 450

مسیر بدست آمده بهینه :

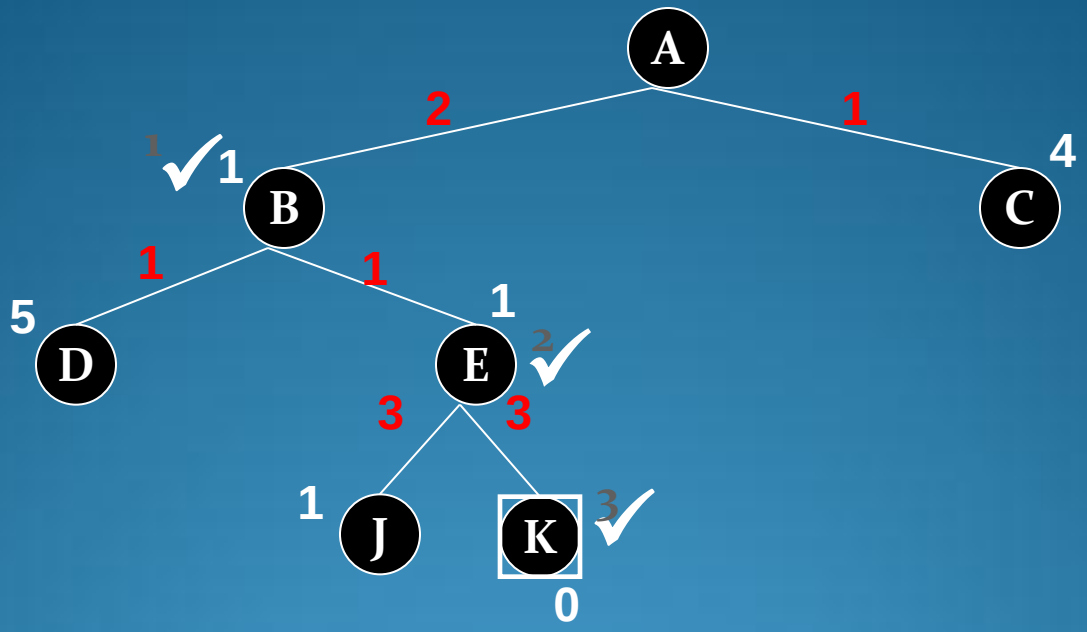
S-E-I-H-G = 140 + 80 + 97+101= 418

Ahmad Abdali Mohamadi

# مثال :



# ادامه مثال :



# ویژگی های استراتژی حریصانه

۱ ) کامل نیست

۲ ) بهینه نیست

۳ ) پیچیدگی زمانی $O(b^m)$

۴ ) پیچیدگی مکانی $O(b^m)$

۵ ) کارآیی این روش به دقت تابع هیورستیک بستگی دارد

۶ ) پیچیدگی زمانی و مکانی با انتخاب یک تابع هیورستیک خوب کاهش می یابد.

# حالات خاص در الگوریتم حریصانه

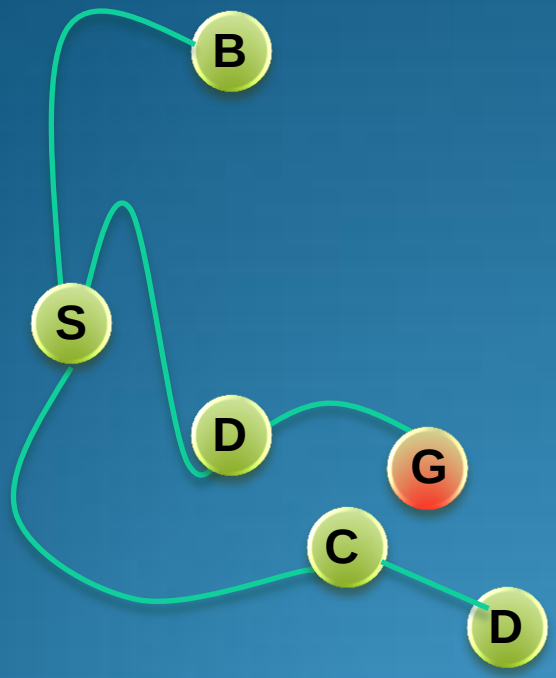
اگر $h = h^*$ آنگاه جستجو کامل میشود

اگر $h = h^*$ آنگاه جستجو بهینه میشود

اگر $h = h^*$ آنگاه پیچیدگی زمانی و مکانی $O(bd)$ می شود .

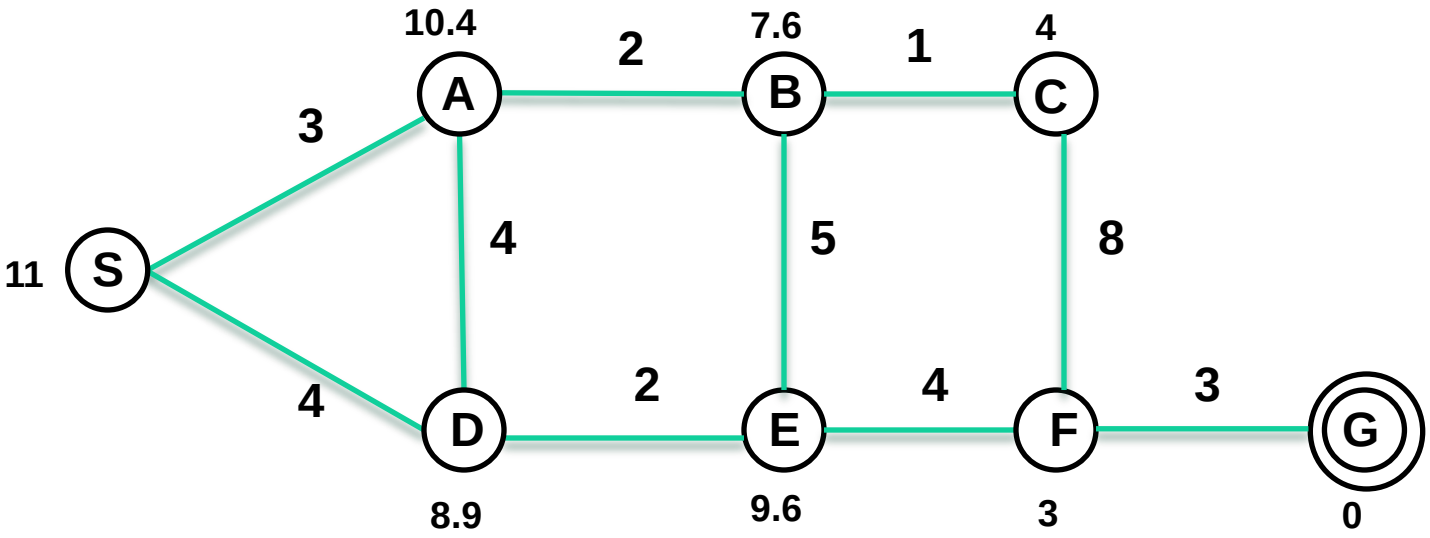
# ایراد های استراتژی حریصانه

ممکن است در یک بن بست گرفتار شود




مانند جستجوی عمقی مسیر را برای رسیدن به هدف دنبال می کند و ممکن است یک مسیر نامتناهی به پایین را دنبال کند که هیچ وقت به هدف نمی رسد.

# مثال :



# جستجوی اول بهترین با استراتژی الگوریتم A*

روش جستجوی A* تلفیقی از روش جستجوی هزینه یکنواخت ( UCS ) و روش جستجوی حریصانه است. در جستجوی هزینه یکنواخت بر اساس هزینه تا گره فعلی ، کم هزینه ترین گره را انتخاب کرده و گسترش می دهیم. جستجوی هزینه یکنواخت بهینه است ، یعنی جواب بهینه مسئله را پیدا می کند ولی در مقابل بسیاز زمانبر است. جستجوی حریصانه نیز بر اساس هزینه تا مقصد ، کم هزینه ترین گره را برای گسترش انتخاب می کند. یافتن جواب با استفاده از جستجوی حریصانه به سرعت انجام می گیرد. ولی این روش نیز از مشکلاتی همچون بهینه نبودن جواب رنج می برد. روش جستجوی A* ، سرعت روش حریصانه در رسیدن به جواب و بیهنگی روش هزینه یکنواخت در پیدا کردن جواب را باهم ترکیب کرده و به جستجوی هدف خود می پردازد.

Ahmad Abdali Mohamadi

# جستجوی اول بهترین با استراتژی الگوریتم A*

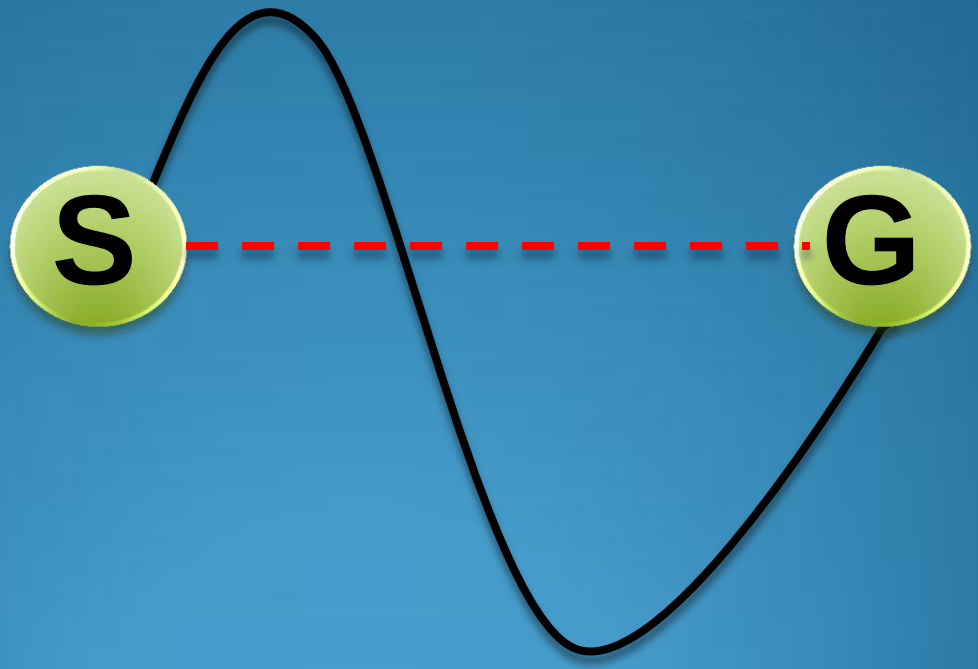
جستجوی A* سعی می کند مجموع هزینه پرداخت شده تا گره فعلی و هزینه باقی مانده از گره فعلی تا هدف را مینیمم کند. تخمین هزینه باقی مانده تا هدف را هیوریستیک مسئله می گویند. طراحی هیوریستیک مسئله در روش جستجوی A* از اهمیت بسزایی برخوردار است و بهینگی روش A* تحت تاثیر طراحی هیوریستیک مسئله قرار دارد.

هیوریستیک در روش A* هرینه قابل پرداخت از نقطه فعلی تا نقطه هدف را تخمین می زند. با این توصیف هیوریستیک قابل قبول را چنین تعریف می کنیم : هیوریستیکی قابل قبول است که هزینه تخمینی آن از نقطه فعلی تا نقطه هدف ، از هزینه واقعی قابل پرداخت از نقطه فعلی تا نقطه هدف کمتر باشد. هیوریستیکی که این شرط را برآورده نکند ، هیوریستیک غیرقابل قبول می نامند.

بعبارت دیگر : یک تابع هیورستیک در صورتی قابل قبول است که هزینه رسیدن به هدف را بیشتر از هزینه واقعی تخمین نزند .

Ahmad Abdali Mohamadi

سوال : چرا در مسله مسیر یابی هیورستیک فاصله مستقیم قابل قبول است ؟

# جستجوی اول بهترین با استراتژی الگوریتم A*

در این الگوریتم با ترکیب دو تابع داریم:

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

**g(n):** هزینه مسیر از گره آغازین به گره n را به ما می‌دهد.

**h(n):** هزینه تخمین زده شده از ارزانترین مسیر از n به هدف است

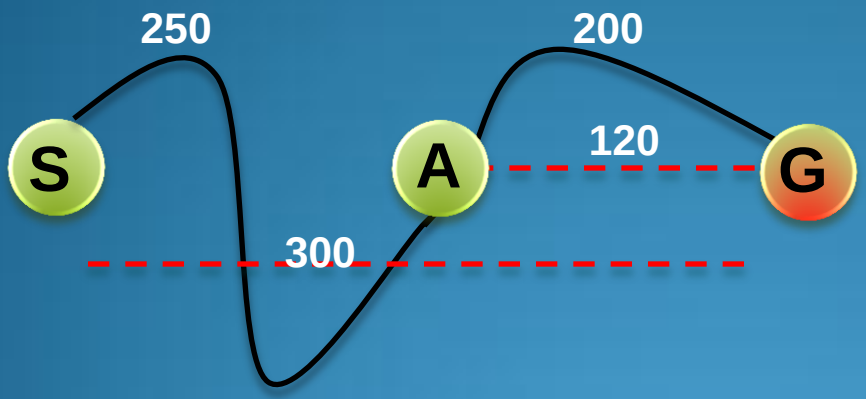
هزینه تخمین زده شده ارزانترین راه حل از طریق n = f(n)

# جستجوی اول بهترین با استراتژی الگوریتم A*

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

هزینه تقریبی رفتن از گره n به گره هدف

هزینه واقعی رسیدن از گره شروع به گره n




محاسبه تابع f(n) در گره S

$f(n) = 0 + 300 = 300$

محاسبه تابع f(n) در گره A

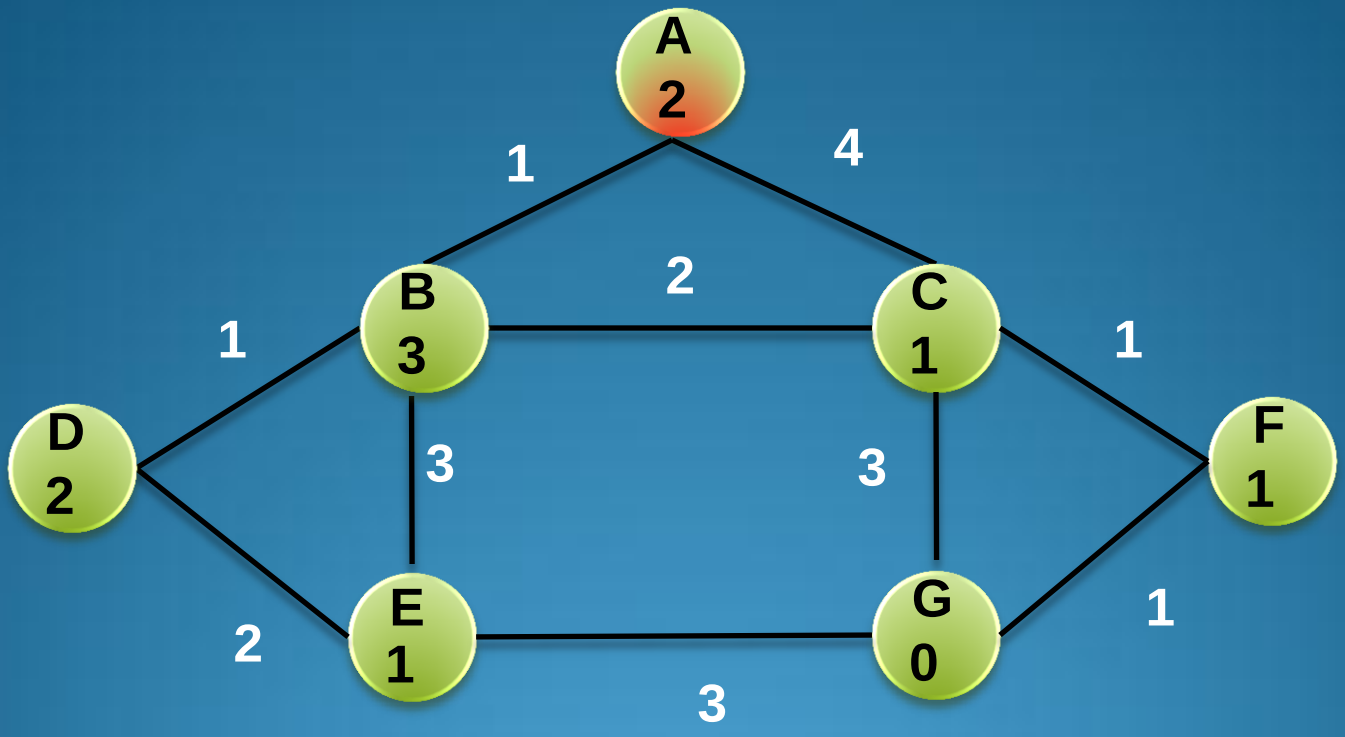
$f(n) = 250 + 120 = 370$

محاسبه تابع f(n) در گره G

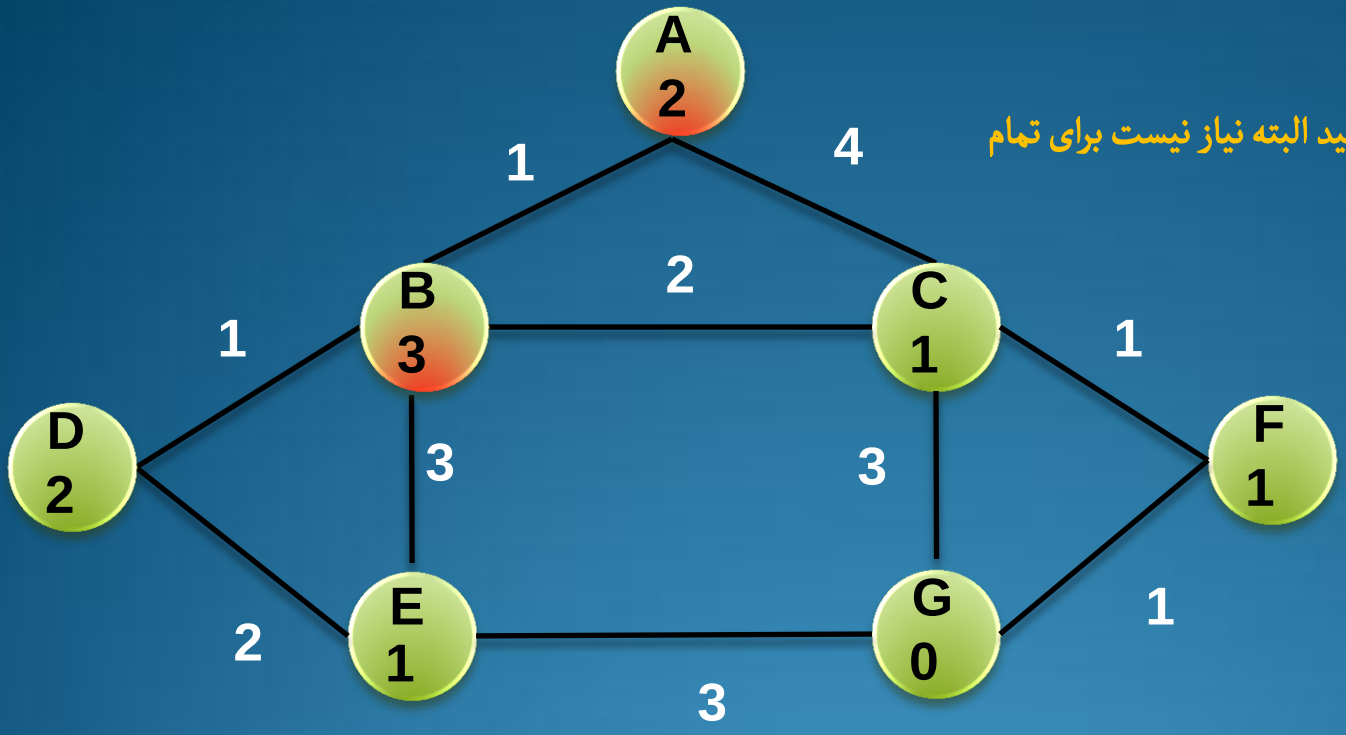
$f(n) = (250+200) + 0 = 450$

Ahmad Abdali Mohamadi

**مثال :** استراتژی A * را روی گراف زیر پیاده سازی کنید و مسیری که توسط این الگوریتم تولید می شود را محاسبه کنید ؟



# جواب :




برای هر گره f (n ) را محاسبه کنید البته نیاز نیست برای تمام گره ها محاسبه شود

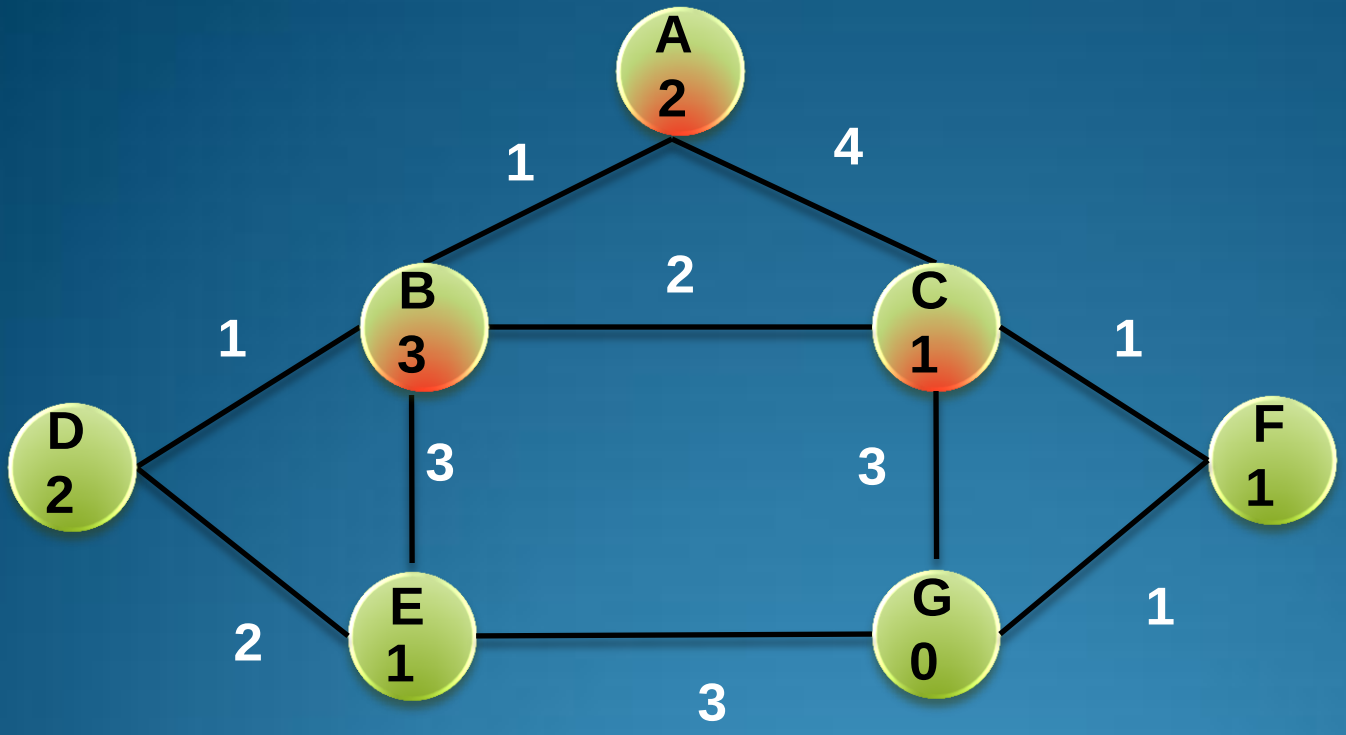
f(n) = g(n) + h(n)

چون f(B) کوچکتر از f(C) هست پس گره B گسترش می یابد .

f(B) = g(B) + h(B) =1+ 3 = 3

f(C) = g(C) + h(C) =4+ 1 = 5

جواب :



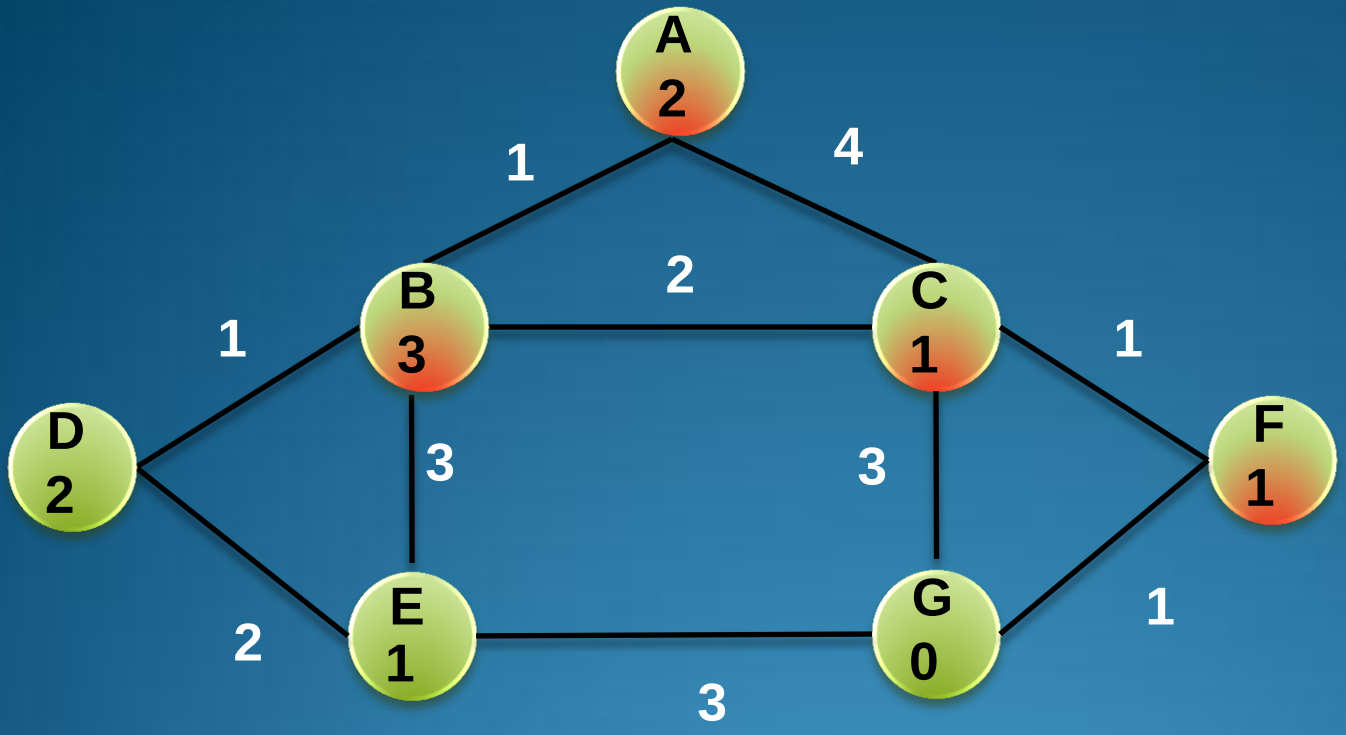

f(D) = g(D) + h(D) =(1+1) + 3 = 5

f(C) = g(C) + h(C) =( 1 + 2 ) + 1 = 4

f(E) = g(E) + h(E) =( 1 + 3 ) + 1 = 5

چون f(C) کوچکتر از ما بقی هست پس گره C گسترش می یابد .

جواب :

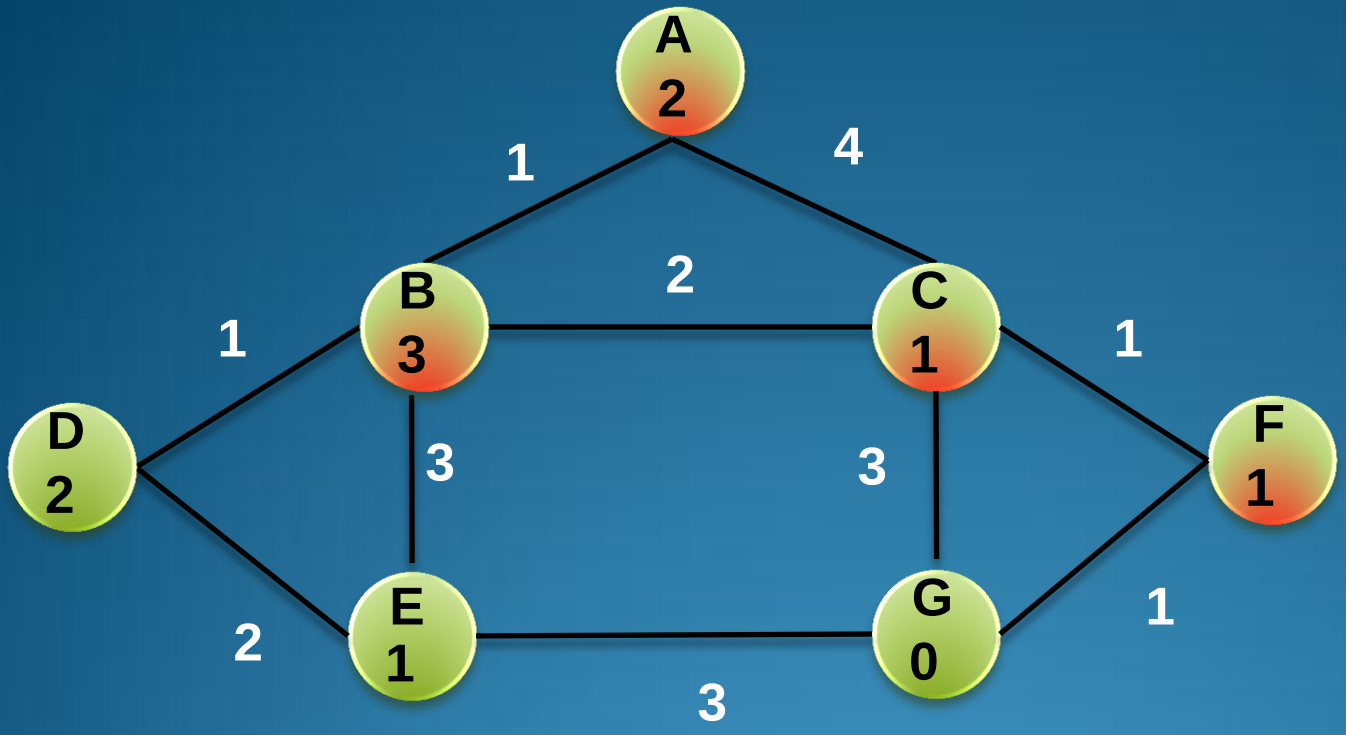



$f(F) = g(F) + h(F) = (1+2+1) + 1 = 5$

$f(G) = g(G) + h(G) = ( 1 + 2 + 3 ) + 0 = 6$

چون f(F) کوچکتر از f(G) هست پس گره F گسترش می یابد .

# جواب :




$f(G) = g(G) + h(G) = ( 1 + 2 + 3 ) + 0 = 6$

انتخابی دیگر نداریم و به هدف رسیدیم با مسیر زیر :

A B C F G هزینه آن ۵ می شود

البته چند مسیر دیگر داریم :

A C G هزینه آن ۷ می شود

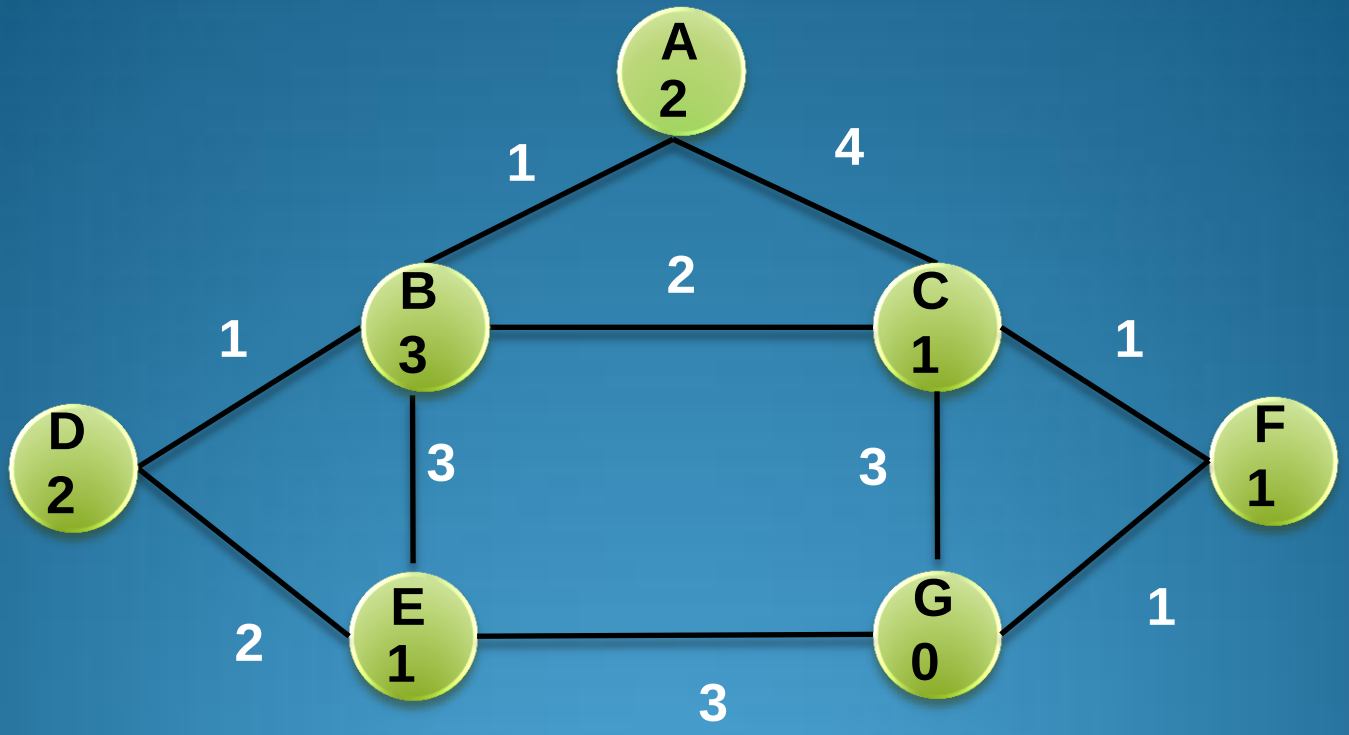
A B C G هزینه آن ۶ می شود

A B E G هزینه آن ۷ می شود

آیا الگوریتم A* همیشه جواب بهینه را می دهد ؟

A* بهینه است به شرط اینکه تابع h(n) قابل قبول باشد

سوال : چرا تابع h(n) زیر قابل قبول است ؟




سوال : چرا تابع h(n) در مسله مسیر یابی قابل قبول است ؟

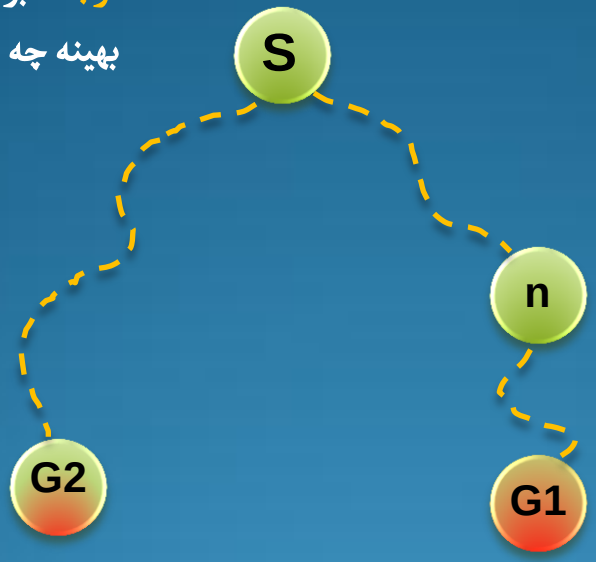
با توجه به درس طراحی الگوریتم دو الگوریتم زیر چه تفاوتی دارند :

**TREE – SEARCH & GRAPH SEARCH**

قضیه : ثابت کنید که A* با استفاده از الگوریتم TREE – SEARCH بهینه است

توجه : برای هر گره هدف چه بهینه چه نیمه بهینه بهینه داریم :

h(n)=0




C* هزینه واقعی مسیر بهینه است

G2 یک گره نیمه بهینه است

G1 یک گره بهینه است

برای گره نیمه بهینه G2 داریم :

1 $f(G2) = g(G2) + h(G2) = g(G2) + 0 > C^*$

برای گره n که در مسیر بهینه قرار دارد داریم :

2 $f(n) = g(n) + h(n) = g(n) + h(n) \leq C^*$

1 , 2 ==> $f(n) \leq C^* < h(G2)$

این نشان می دهد که G2 گسترش نمی یابد بلکه n گسترش می یابد

توجه : این اثبات در الگوریتم های GRAPH- SEARCH به شکست میرسد .

Ahmad Abdali Mohamadi

مفاهیم زیر توسط دانشجویان بررسی شوند :

اگر در A* از الگوریتم GRAPH – SEARCH استفاده کنیم الگوریتم A* بهینه است ؟

کانتور چیست ؟

چرا با وجو د بهینه بودن الگوریتم A* این الگوریتم کارآمد نیست ؟

# نکات مهم الگوریتم A*

**نکته ۱:** مهمترین شرط برای بهینه بودن الگوریتم A* این است که برای هر گره داشته باشیم

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

یعنی تابع اکتشافی همیشه کمتر مساوی مقدار واقعی تخمین بزند $h(n)$ هزینه تخمینی براي رسیدن از نود $n$ به هدف است

$h(n)$ هزینه تخمینی براي رسیدن از نود $n$ به هدف است

$h^*(n)$ هزینه واقعی براي رسیدن از نود $n$ به هدف است

# نکات مهم الگوریتم A*

نکته ۲: اگر $f^*$ هزینه واقعي مسير بهينه باشد و $h$ قابل قبول باشد ، آنگاه در الگوريتم $A^*$ تمام گره هايي را كه شرايط $f(n)<f^*$ را دارند بسط مي يابند.

بعبارت دیگر : یعنی هزینه تخمینی هر گره $f(n)$ در مسیر بهینه به سمت جواب همواره از هزینه بهینه واقعی کل مسیر کمتر است

# نکات مهم الگوریتم A*

نکته ۳: اگر برای (جستجو با هزینه یکسان). nها ۰ =h(n) باشد جستجو تبدیل به جستجوی غیر هوشمند خواهد شد.
حال اگر g (n) برابر با یك باشد جستجو تبدیل به جستجوي اول سطح مي شود.

# نکات مهم الگوریتم A*

نکته ۴: اگر $h(n)= h^*(n)$ باشد جستجو ی صورت نمیگیرد و فقط گره های واقع در مسیر بهینه بسط داده می شوند .

# نکات مهم الگوریتم A*

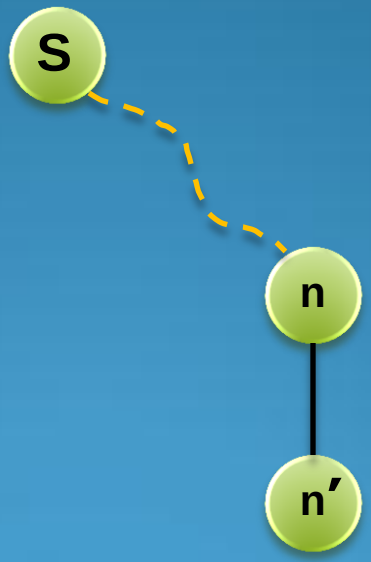
**نکته ۵ :** اگر A1 و A2 دو نسخه از A* باشند که با h1 و h2 بسط داده می شوند و داشته باشیم که h*(n)>h2(n)>h1(n) آنگاه الگوریتم A2 هوشمند تر از الگوریتم A1 عمل می کند .

در ضمن تعداد گره های که A2 بسط می دهد حداقل به تعداد گر ه های A1 است .

# نکات مهم الگوریتم A*

نکته ۶ : تابع $h(n)$ را یکنوا می گویند اگر رابطه زیر در مورد آن صادق باشد :

$$h(n) < C(n , a , n') + h(n')$$

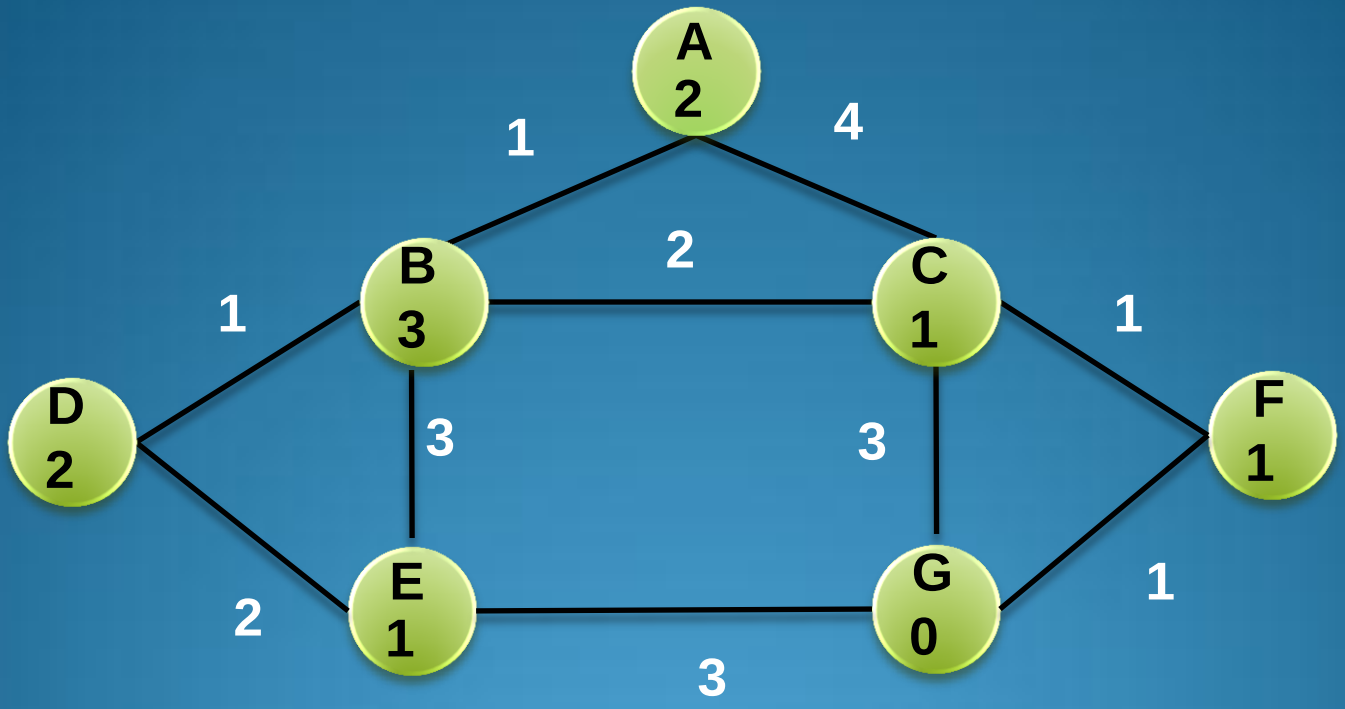



$C(n , a , n')$ یعنی $n'$ با استفاده از عمل $a$ از حالت $n$ ساخته شده

# نکات مهم الگوریتم A*

مثال : در گراف زیر تابع $h(n)$ یکنوا است زیرا با توجه به رابطه زیر :

$$h(n) < C(n , a , n') + h(n')$$



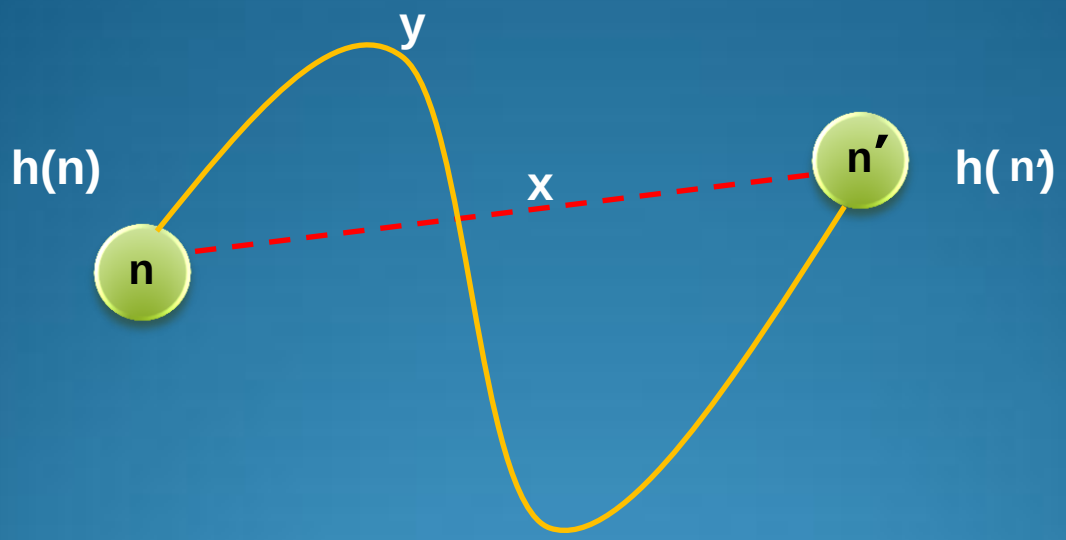

h(A) < C(A , a , B) +h(B) => 2 < 1 + 3 => 2 < 4 ≡ True

h(A) < C(A , a , C) +h(C) => 2 < 4 + 1 => 2 < 5 ≡ True

Ahmad Abdali Mohamadi

# نکات مهم الگوریتم A*

مثال : ثابت کنید در مسله مسیر یابی تابع هیورستیک $h_{SLD}$ یکنواست ؟



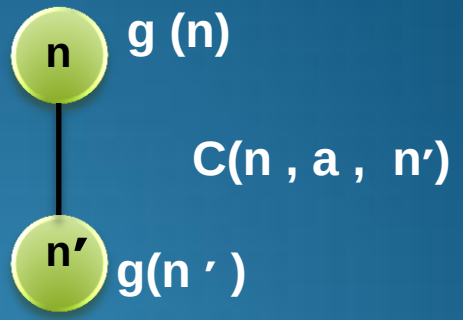

$h(n) < C(n , a , n') + h( n' )$

$x < y + h( n' )$ $\xrightarrow{h( n' ) >=0}$

چون روابط بازگشتی هستند اثبات می شود

# نکات مهم الگوریتم A*

مثال : ثابت کنید اگر $h(n)$ یکنوا باشد مقادیر $f(n)$ در مسیر غیر کاهشی قرار دارند ؟



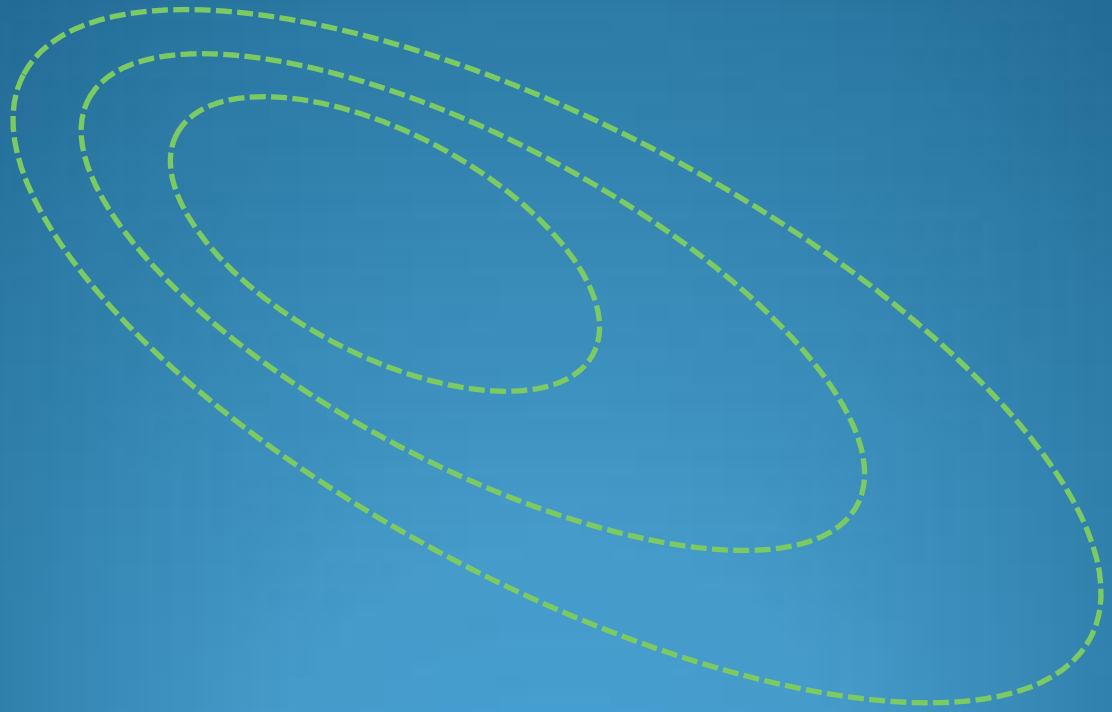

1. $g(n') = g(n) + C(n, a, n')$
2. $h(n) < C(n, a, n') + h(n')$

1 2 ⟶ $f(n') = g(n') + h(n') = g(n) + C(n, a, n') + h(n') > g(n) + h(n) = f(n)$

نکته مهم : خصوصیت غیر کاهشی تابع $f$ این امکان را به ما می دهد که کانتور را تعریف کنیم

# نکات مهم الگوریتم A*

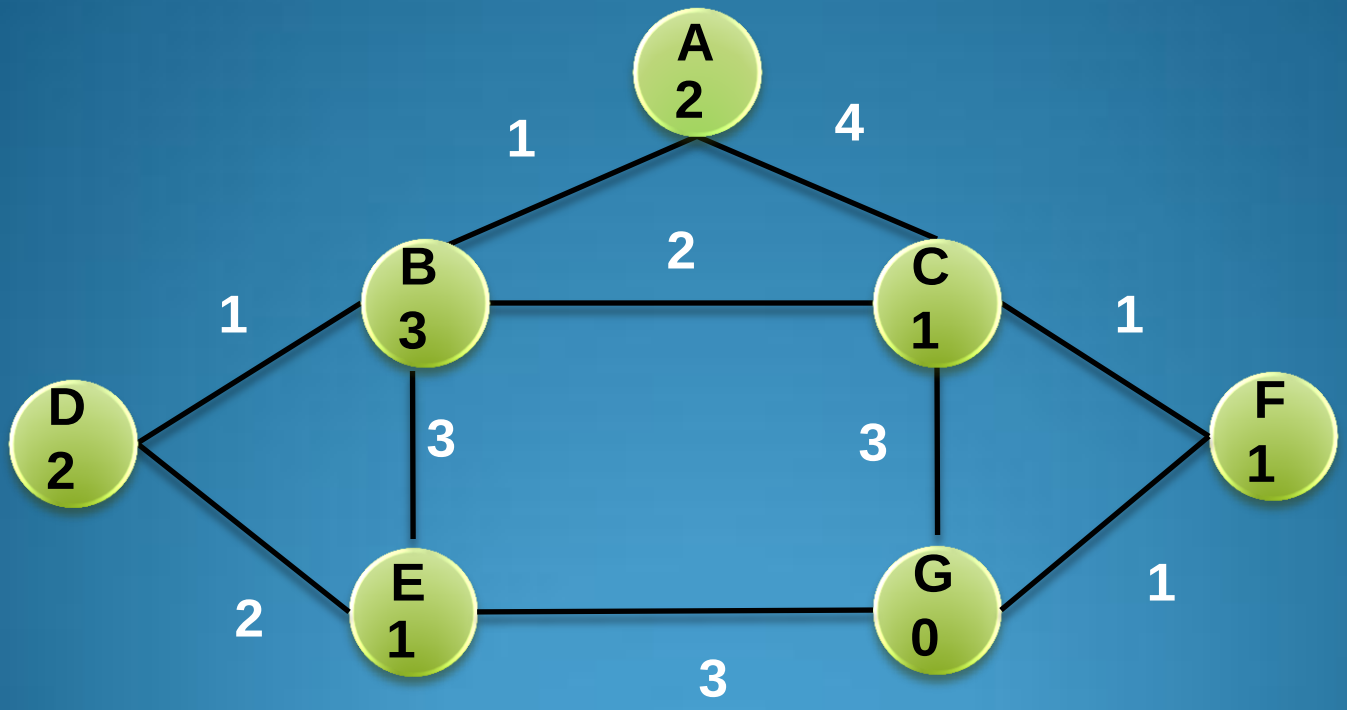
سوال : کانتور های مسله مسیر یابی در رومانی را رسم کنید ؟



# نکات مهم الگوریتم A*

سوال : با توجه به گراف زیر موضوع ذیل را بررسی کنید :

A* تمام گره های را که $f(n) < C^*$ باشد را بسط می دهد



با توجه به مفهوم کانتور نشان دهید $A^*$ کامل است

# نکات تکمیلی الگوریتم A*

A* تمام گره های را که $f(n) < C^*$ باشد را بسط می دهد

A* ممکن است برخی از گره های که روی کانتور $f(n) = C^*$ هستند را قبل از گره هدف بسط دهد

A* تمام گره های $f(n) < C^*$ را بسط می دهد

A* هیچ گره ای را که $f(n) > C^*$ را بسط نمی دهد

هیچ الگوریتمی بهینه دیگری تضمین نمی کند که تعداد گره کمتری نسبت به A* تولید کنند زیرا اگر الگوریتمی تمام گره های که در محدوده $f(n) < C^*$ را بسط ندهد با خطر از دست دادن راه حل بهینه مواجه میشود

# نکات تکمیلی الگوریتم A*

توابع هیورستیک که یکنوا نیستند به معادله زیر تبدیل به یکنوا می شوند

معادله pathmax :

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

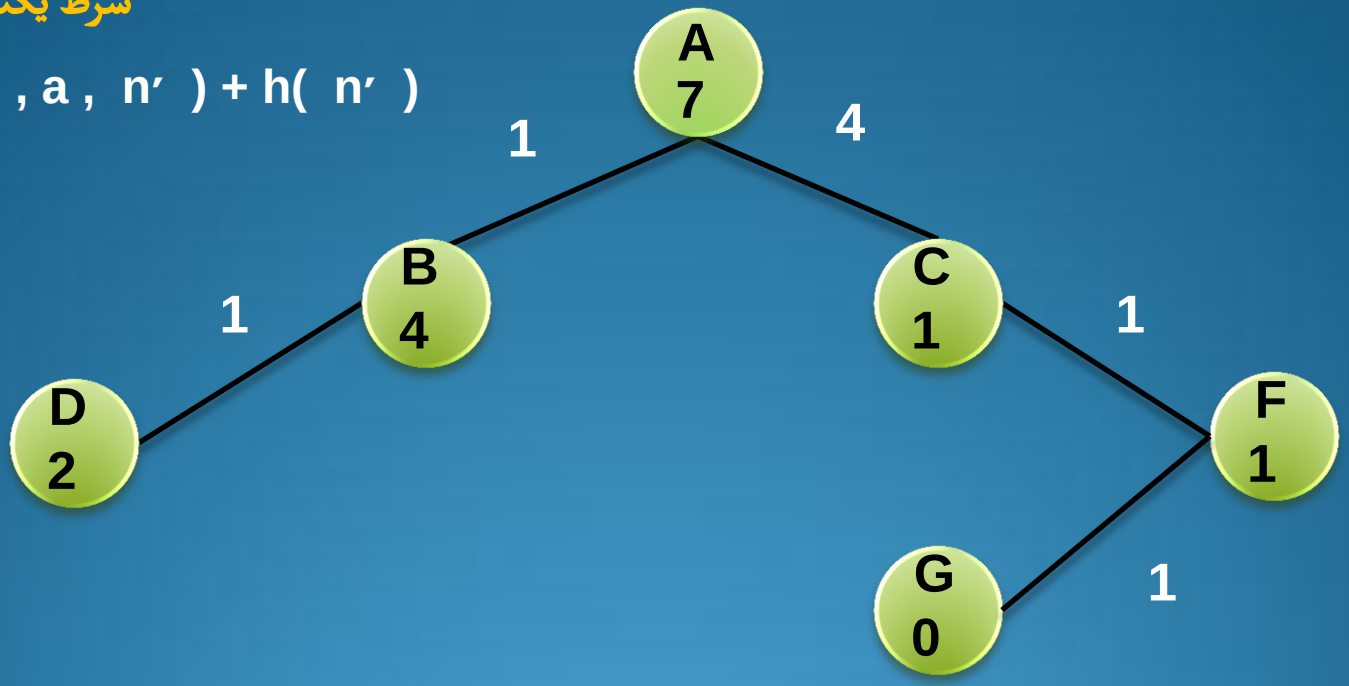

پدر

فرزند

**مثال :** تابع هیورستیک غیر یکنوای زیر را تبدیل به یک تابع هیورستیک یکنوا کنید ؟

**شرط یکنوایی :**

$$h(n) < C(n, a, n') + h(n')$$




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

# ایرادات اصلی الگوریتم A*

تعداد گره های که در کانتور هدف قرار دارد نمایی هستند( تعداد زیاد)

اثبات می شود اگر رابطه زیر برقرار باشد تعداد گر ه ها در کانتور هدف نمایی نیست :

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

متاسفانه در عمل :

1. $|h(n) - h^*(n)| \approx h^*(n)$
2. $h^*(n) \geq O(\log h^*(n))$

1 2 $|h(n) - h^*(n)| \geq O(\log h^*(n))$

نتیجه : در نهایت رشد نمایی تعداد گره ها کل حافظه کامپیوتر را اشغال می کند

# ایرادات اصلی الگوریتم A*

هم از نظر زمانی و هم از نظر مکانی الگوریتم A* مرتبه نمایی دارد .

به علت مصرف زیاد الگوریتم A* برای مسائلی با فضای حالت گسترده کاربرد ندارد .

# رفع مشکل الگوریتم A*

برای رفع مشکل الگوریتم A* باید سعی کنیم بر فضای حالت بزرگ غلبه کرد برای این کار سه روش داریم :

۱) IDA *
۲) MA *
۳) SMA *

جستجوی حافظه محدود شده

# الگوريتم IDA *

ديديم كه جستجوي عميق كننده تكراري تكنيكي مفيد براي كاهش مصرف حافظه است ( فصل قبل ) حال مي توانيم از اين تكنيك استفاده نموده و هر بار يك جستجوي عمقي تا هزينه f-limit را انجام دهيم . اگر هدف پيدا نشد مقداری به هزينه f-limit اضافه می کنیم و دوباره جستجو را تکرار می کنیم . البته در این الگوریتم بجای عمق از محدودیت هزینه استفاده می کنیم .

در این الگوریتم مقدار اولیه f-limit را برابر مقدار f ریشه قرار می دهیم . در هر مرحله گره ای که f آن از f-limit کمتر باشد گسترش می یابد . اگر هدف پیدا شد الگوریتم تمام می شود وگرنه کمترین مقدار f گسترش نیافته را درون متغیر f-limit می ریزیم .

# ویژگی های الگوریتم IDA *

هدف در تکراری پیدا می شود که f-limit = C*

ثابت میشو د این الگوریتم کامل و بهینه است

در این الگوریتم در هر مرحله فقط گره های که f<f-limit است گسترش می یابد .

اگر تعداد تکرار زیاد نباشد این الگوریتم از نظر کارآیی مانند A * است .

محدوديت هزينه در هر مرحله به گونه اي انتخاب مي شود كه در مراحل قبلي ثابت شده است كه جوابي با هزينه كمتر از اين مقدار وجود ندارد.

# ویژگی های الگوریتم IDA *

مي توان هزينه مرحله جديد را برابر با كمترين هزينه نودي كه در مرحله قبلي بسط داده نشده قرار داد از آنجا كه اين روش به صورت عمقي است پس
پيچيدگي فضايي آن در بدترين حالت $S(b.d)$ است.

پيچيدگي زماني بستگي به تابع اكتشافي دارد.

# نقطه ضعف الگوریتم IDA *

نقطه ضعف اصلی این الگوریتم دوباره کاری اضافی است.

# حل مسله فروشنده دوره گرد با $IDA^*$

در این روش تعداد تکرار زیاد نیست

در این روش کارآیی مانند $A^*$ است .

در این روش در هر تکرار فقط یک گره اضافه می شود .

اگر در $A^*$ تعداد n گره گسترش یابد پس $IDA^*$ n بار تکرار می شود .

پیچیدگی این روش :

$$1 + 2 + 3 + \ldots + n = O(n^2)$$

برای بهینه کردن مسله فروشنده دوره گرد با IDA * چه روشی وجود دارد ؟

# الگوریتم بازگشتی RBFS

ساختار آن شبیه جستجوی عمقي بازگشتي است ، اما به جای اینکه دائما به طرف پایین مسیر حرکت کند ، مقدار f مربوط به بهترین مسیر از هر جد گره فعلی را نگهداری میکند ، اگر گره فعلی از این حد تجاوز کند ، بازگشتی به عقب برمیگردد تا مسیر دیگري را انتخاب کند.

این جستجو اگر تابع اکتشافی قابل قبولی داشته باشد ، بهینه است.

پیچیدگي فضایي آن $O(bd)$ است

تعیین پیچیدگی زمانی آن به دقت تابع اکتشافی و میزان تغییر بهترین مسیر در اثر بسط گره ها بستگی دارد.

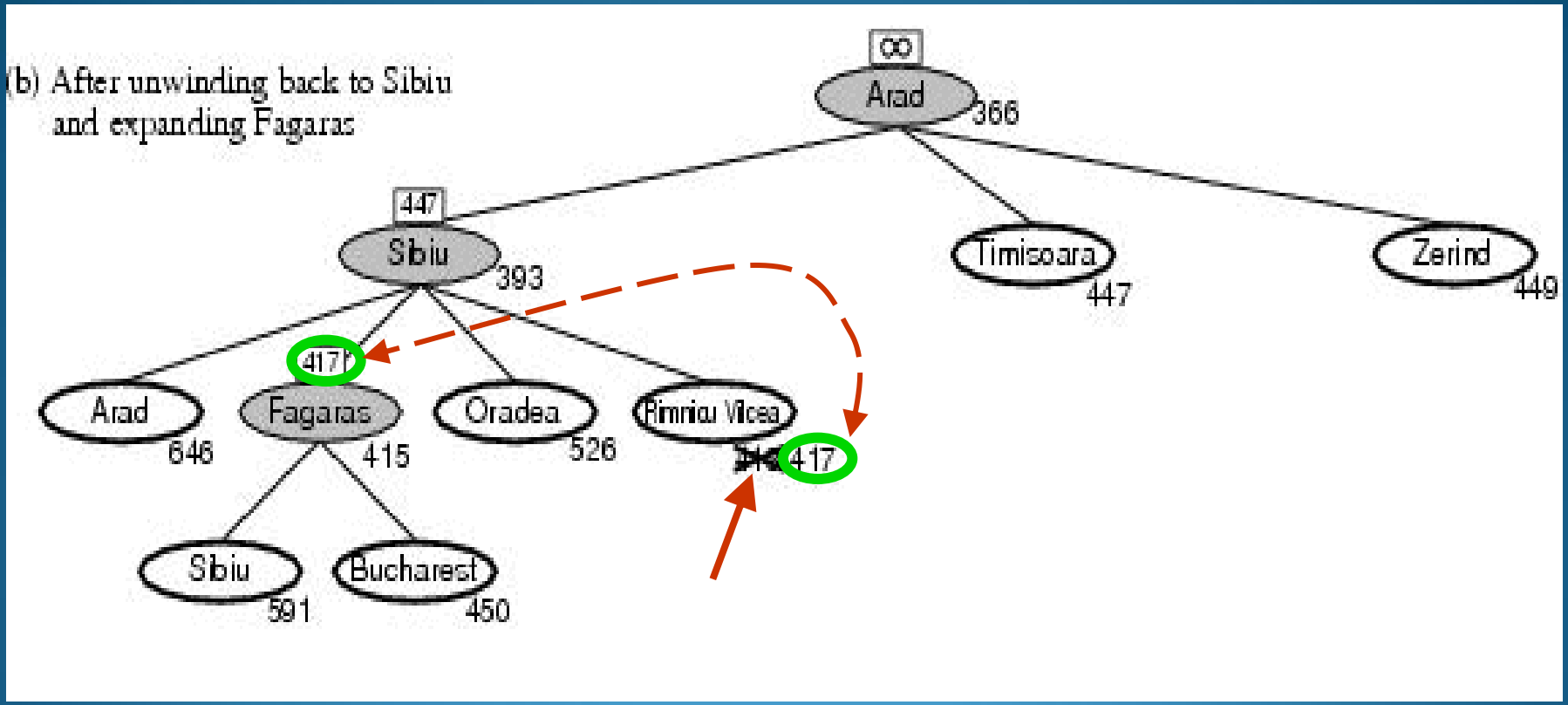
# ویژگی های الگوریتم بازگشتی RBFS

RBFS تا حدی از IDA* کارآمدتر است ، اما گره های زیادی تولید میکند.

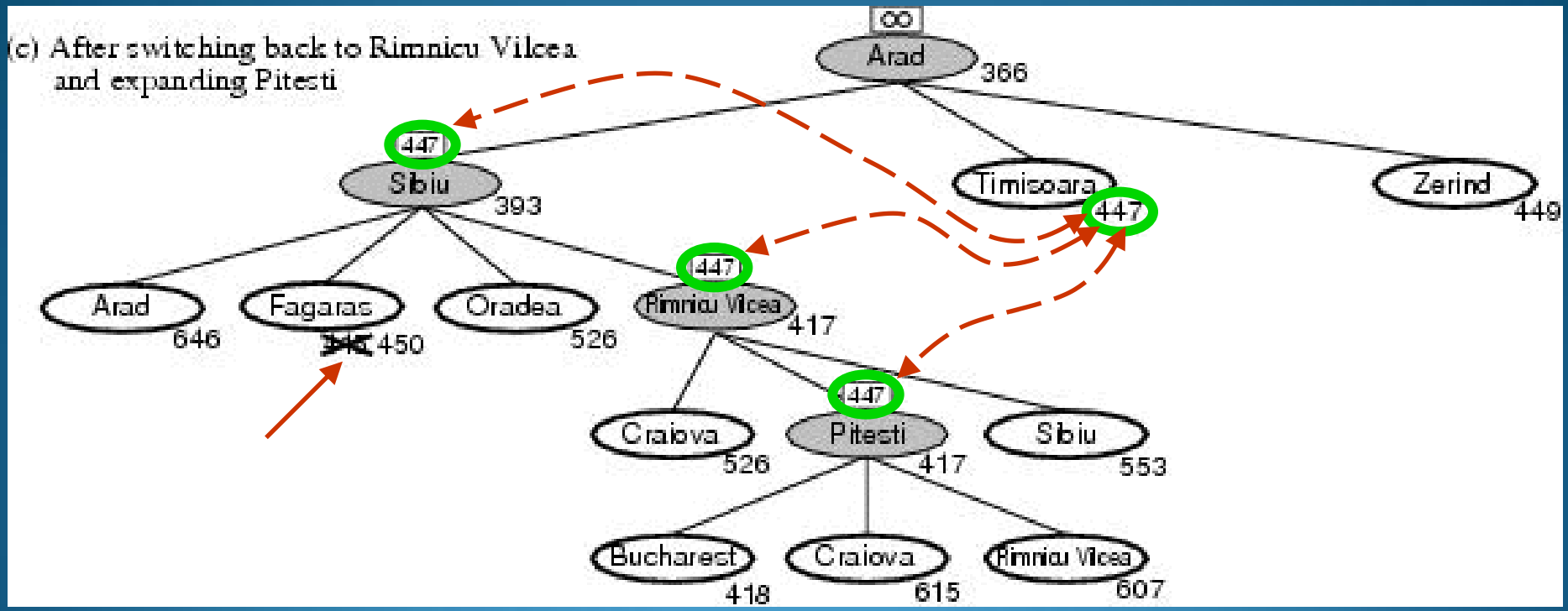
IDA* و RBFS در معرض افزايش تواني پيچيدگي قرار دارند که در جستجوی گرافها مرسوم است ، زیرا نمیتوانند حالتهای تکراری را در غیر از مسیر فعلي بررسي کنند. لذا ، ممکن است یک حالت را چندین بار بررسي کنند.

IDA* و RBFS از فضای اندکي استفاده میکنند که به آنها آسیب میرساند. IDA* بین هر تکرار فقط یک عدد را نگهداری میکند که فعلي هزينه f است. RBFS اطلاعات بیشتری در حافظه نگهداری میکند

# مسله مسیر یابی در رومانی توسط RBFS



# مساله مسیر یابی در رومانی توسط RBFS



# جستجوی حافظه محدود ساده SMA*

SMA* بهترین برگ را بسط میدهد تا حافظه پر شود. در این نقطه بدون از بین بردن گره های قبلي نمیتواند گره جدیدی اضافه کند

SMA* همیشه بدترین گره برگ را حذف میکند و سپس از طریق گره فراموش شده به والد آن بر میگردد. پس جد زیر درخت فراموش شده ، کیفیت بهترین مسیر را در آن زیر درخت میداند

اگر عمق سطحی ترین گره هدف کمتر از حافظه باشد, کامل است.

SMA* بهترین الگوریتم همه منظوره برای یافتن حلهای بهینه میباشد

# جستجوی حافظه محدود ساده SMA*

اگر مقدار f تمام برگها یکسان باشد و الگوریتم یک گره را هم برای بسط و هم برای حذف انتخاب کند ، SMA* این مسئله را با بسط بهترین برگ جدید و حذف بهترین برگ قدیمی حل میکند

ممکن است SMA* مجبور شود دائما بین مجموعه ای از مسیرهای حل کاندید تغییر موضع دهد ، در حالی که بخش کوچکی از هر کدام در حافظه جا شود

محدودیتهای حافظه ممکن است مسئله ها را از نظر زمان محاسباتی ، غیر قابل حل کند.

# سرگرمی یا هوش !

هواپیمایی بین مرز کانادا و امریکا سقوط می کند بنظر شما بازماندگان این حادثه کجا دفن می شوند ؟

# مباحثی در مورد توابع اکتشافی

توابع اکتشافی متداول معمای هشت

کیفیت توابع اکتشافی و تاثیر آن بر کارآیی مسله

ابداع توابع اکتشافی قابل قبول

ساخت توابع اکتشافی به وسیله تجربه


# توابع اکتشافی متداول معمای هشت

## پیشنهاد اول :

در این روش تعداد خانه های که در سر جای خود نیستند را به عنوان مقدار تابع اکتشافی در نظر میگیریم


h(این حالت) = 8


h(این حالت) = 7


# توابع اکتشافی متداول معمای هشت

## پیشنهاد دوم :

در این روش مجموع فاصله بین خانه افقی و عمودی را به عنوان مقدار تابع اکتشافی در نظر میگیریم


حالت اولیه


حالت هدف

h(حالت اولیه) = 3+1+2+2+2+3+3+2 =18

# کیفیت و کارآیی تابع اکتشافی

**فاکتور انشعاب موثر :** روشی برای تعین کارآیی و کیفیت تابه اکتشافی است

فاکتور انشعاب موثر را با $b^*$ نشان می دهند .

تعداد گره های تولیدی توسط $A^*$ را برابر $N$ باشد و عمق راه حل باشد $d$

داریم :

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

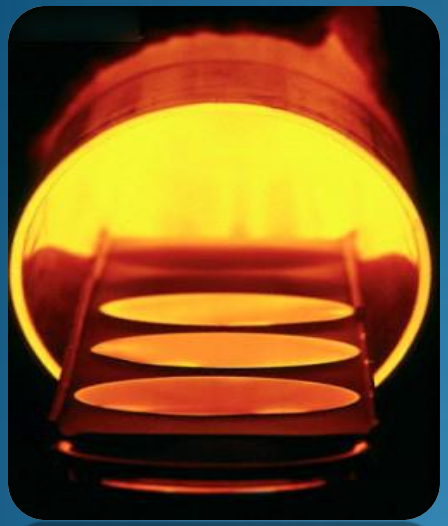
**مثال :** اگر با $A^*$ راه حلی به عمق ۵ و با گسترش ۵۲ گره تولید شده باشد فاکتور انشعاب تابع اکتشافی را بدست آورید ؟

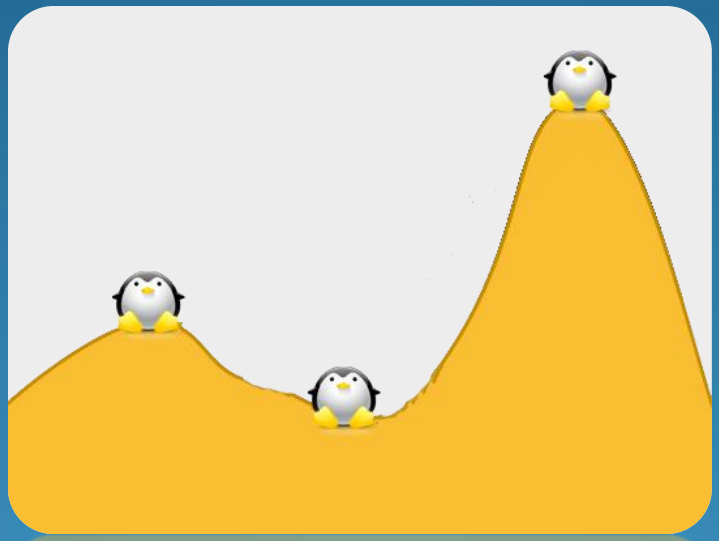
$$N+1 = 1+b^*+ (b^*)^2+ (b^*)^3+(b^*)^4+(b^*)^5$$

$$۵۲+1 = 1+b^*+ (b^*)^2+ (b^*)^3+(b^*)^4+(b^*)^5$$

$$b^* \approx 1.94$$

# الگوریتم های جست و جوی محلی و بهینه سازی





Ahmad Abdali Mohamadi

# الگوريتم هاي اصلاح تكراري

دسته اي از مسائل وجود دارد كه در آنها مسير رسيدن به جواب مهم نيست بلكه فقط جواب نهايي اهميت دارد  همانند مسئله ۸ وزير. ايده اصلی در این الگوریتم ها این است که وقتي از حالت قبلي به حالت فعلي برسيم ديگر حالت قبلي را فراموش كنيم لذا در این مسائل درخت نداريم بلكه تنها يك وضعيت فعلي داريم كه خودش شامل تمام اطلاعات مورد نياز مسئله است که در این الگوریتم ها از يك ساختار كامل شروع شده و سپس با انجام تغييراتي در آن سعي در اصلاح كيفيت آن ساختار داريم.

**دو مزیت اصلی این الگوریتم ها :**

۱) از حافظه کمی استفاده می کنند .

۲) در فضای بزرگ بر خلاف الگوریتم های سیستماتیک راه حل خوبی پیدا می کنند .

Ahmad Abdali Mohamadi

# الگوریتم تپه نوردی

الگوریتم تپه نوردی بدین صورت است که ابتدا جوابی به شکل تصادفی برای مساله تولید
می شود. سپس در یک حلقه و تا زمانی که شرط توقف الگوریتم برقرار نشده است ، به طور مکرر تعدادی همسایه برای حالت فعلی تولید شده و از میان حالت های همسایه بهترین آن ها انتخاب شده و جایگزین حالت فعلی می شود .

در حالت کلی بهینگی جوابی که این الگوریتم پیدا می کند محلی است. لازم بذکر است که منظور از بهینگی مینیمم سازی یا ماکزیمم سازی یک مسئله بهینه سازی می باشد.

Ahmad Abdali Mohamadi

# مراحل الگوریتم تپه نوردی

برای پایان ترم

Ahmad Abdali Mohamadi

# Question ?

# Thank You

Ahmad Abdali Mohamadi