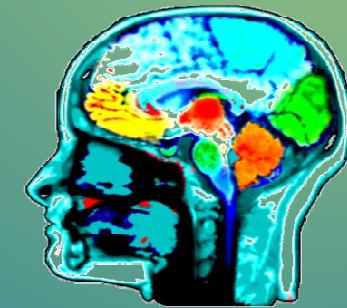




Introduction To Artificial Intelligence

Isfahan University of Technology (IUT)
1402



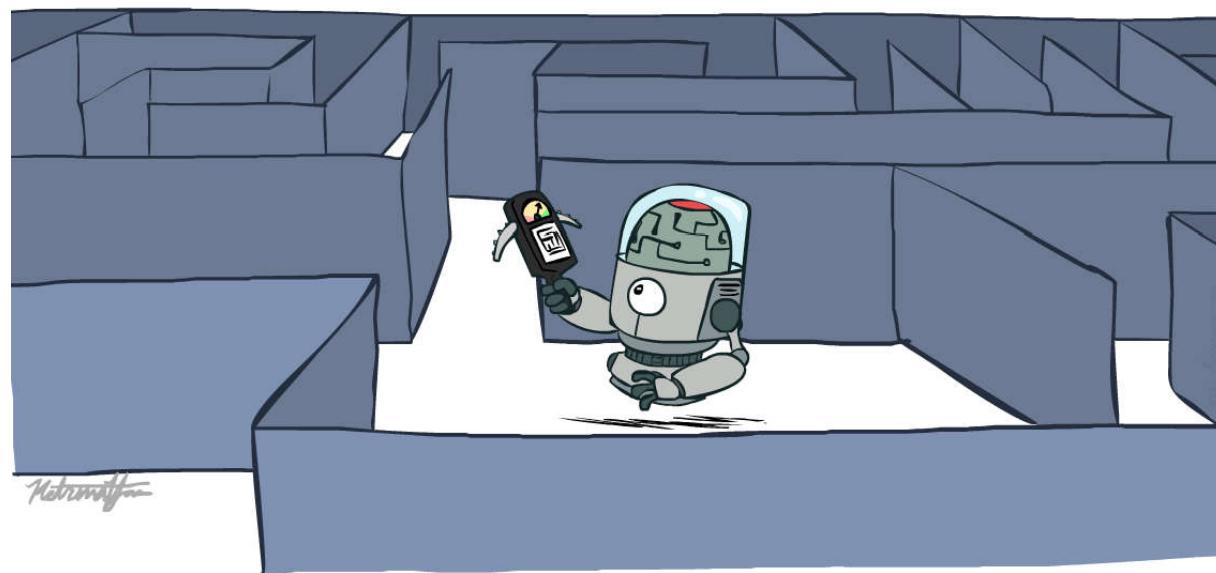
Informed Search

Dr. Hamidreza Hakim
hamid.hakim.u@gmail.com

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley.]

بِسْمِ اللّٰهِ الرَّحْمٰنِ الرَّحِيْمِ

Informed Search



یک دسته دیگه ای از روش های جستجو وجود داره به اسم روش جستجوی اگاهانه ینی جدا از اطلاعاتی که می ریم خودمون کسب میکنیم یکسری اطلاعاتی از هدف هم می تونیم به خودمون اضافه کنیم ینی یکسری اطلاعاتی از مسئله رو میخوایم اضافه بکنیم به خود اون عامل که اون عامل هوشمندانه تر عمل بکنه

Today

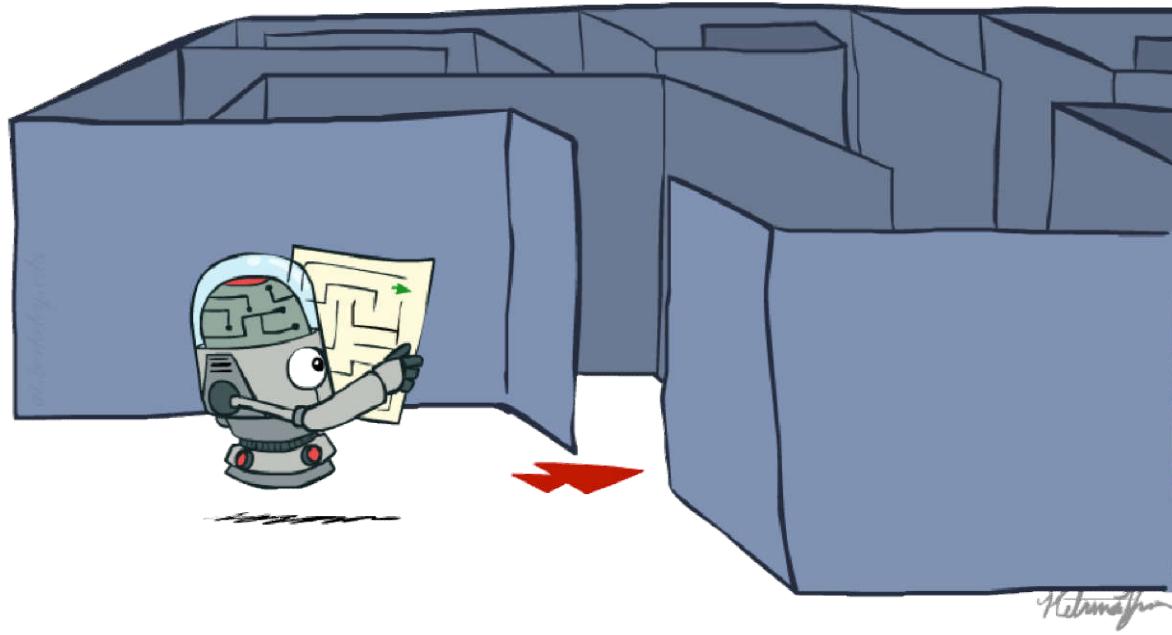
- Informed Search
 - Heuristics
 - Greedy Search
 - A* Search

- Graph Search



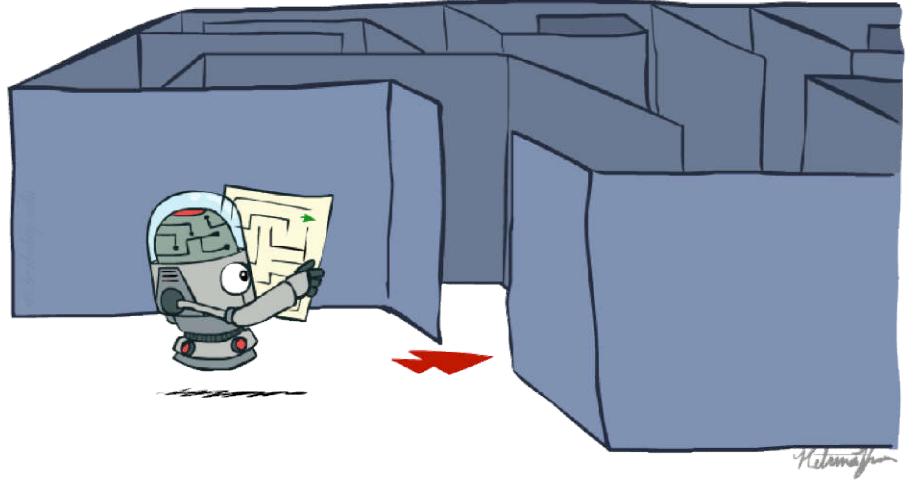
ایا همه مسائل رو می تونیم به صورت Informed Search حل بکنیم؟ نه

Recap: Search



Recap: Search

- Search problem:
 - States (configurations of the world)
 - Actions and costs
 - Successor function (world dynamics)
 - Start state and goal test
- Search tree:
 - Nodes: represent plans for reaching states
 - Plans have costs (sum of action costs)
- Search algorithm:
 - Systematically builds a search tree
 - Chooses an ordering of the fringe (unexplored nodes)
 - Optimal: finds least-cost plans

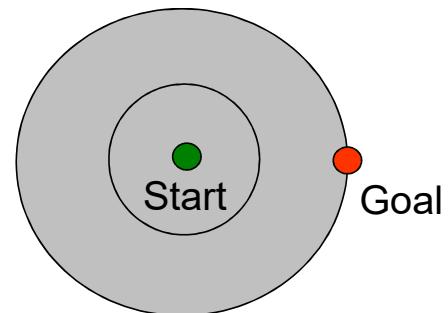
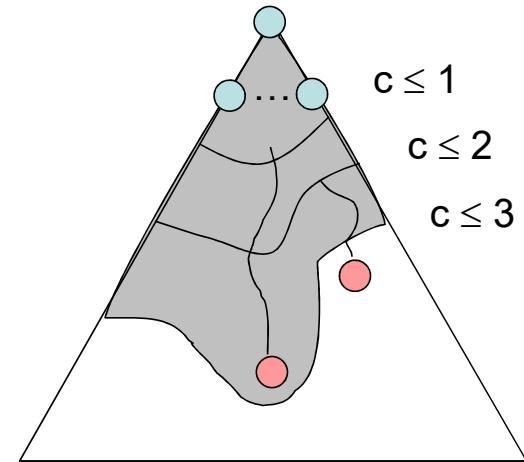


Uninformed Search



Uniform Cost Search

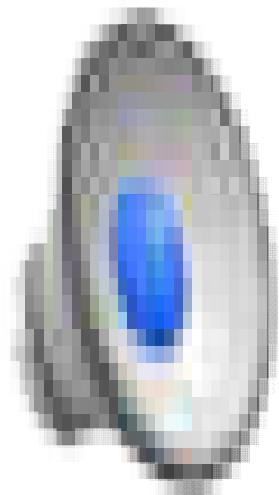
- Strategy: expand lowest path cost
- The good: UCS is complete and optimal!
- The bad:
 - Explores options in every “direction”
 - No information about goal location



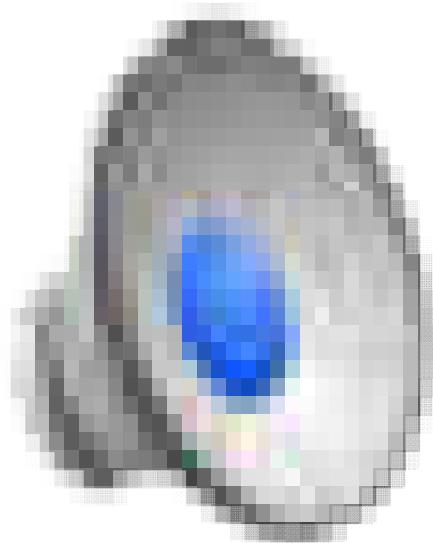
[Demo: contours UCS empty (L3D1)]

[Demo: contours UCS pacman small maze (L3D3)]

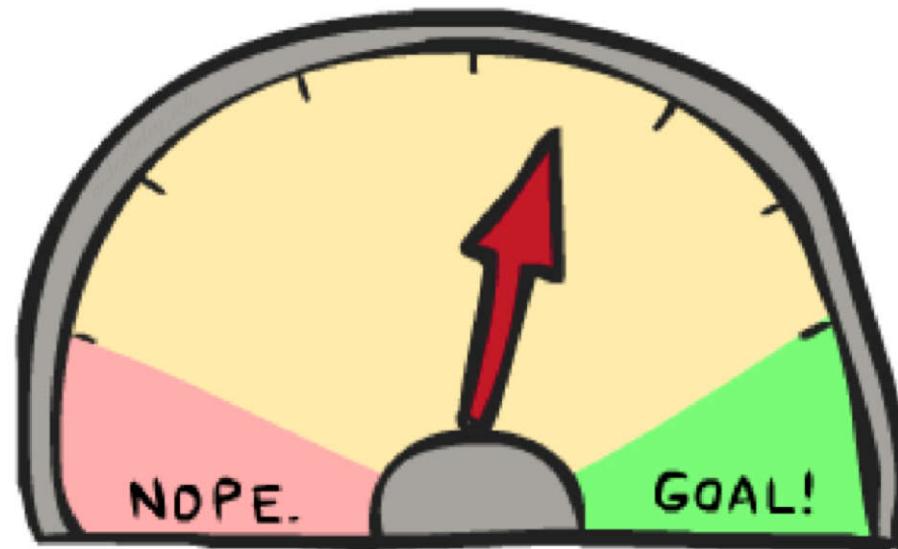
Video of Demo Contours UCS Empty



Video of Demo Contours UCS Pacman Small Maze



Informed Search



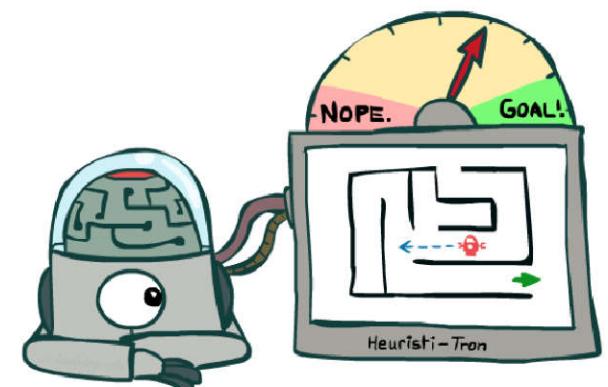
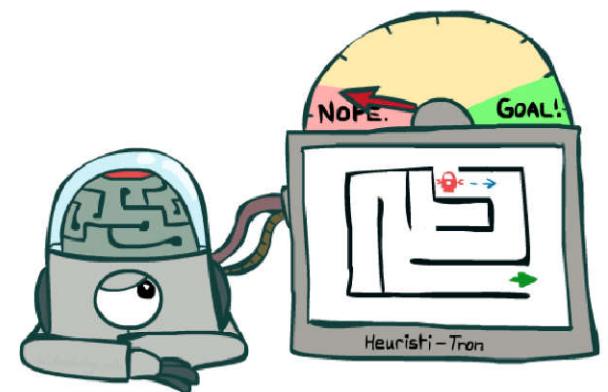
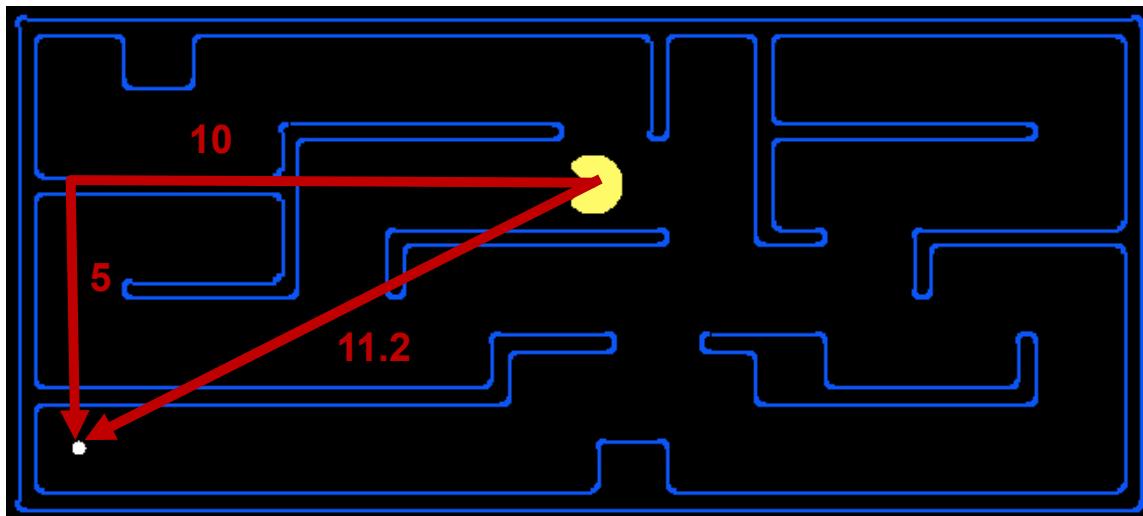
اتفاقی که اینجا می افته:

توی این روش جدا از اینکه تا کجا او مدی ما بہت میگیم تا هدف چقدر فاصله داری

Search Heuristics

- A heuristic is:

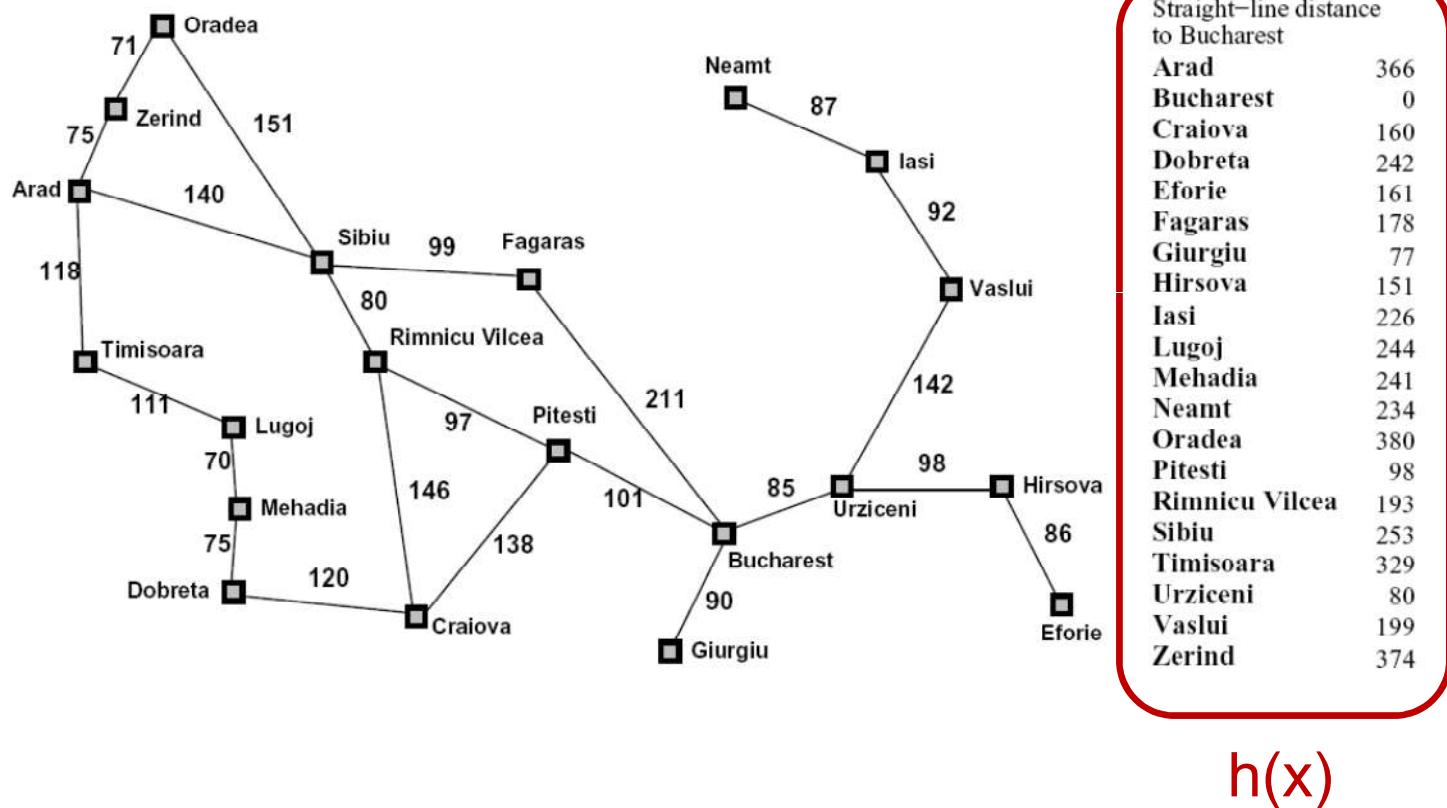
- A function that *estimates* how close a state is to a goal
- Designed for a particular search problem
- Examples: Manhattan distance, Euclidean distance for pathing



اون دستگاهه که داره فاصله ما رو تا هدف اندازه گیری میکنه بهش میگن Heuristics function که این Heuristics یکسری ویژگی هایی داره که جلوتر می گیم و هر مسئله ای هم Heuristics خودش رو داره

ما لوکیشن عامل و هدف رو داریم می تونیم فاصله این دو تا رو حساب کنیم الان ؟؟ نفهمیدم خوب چیه؟

Example: Heuristic Function



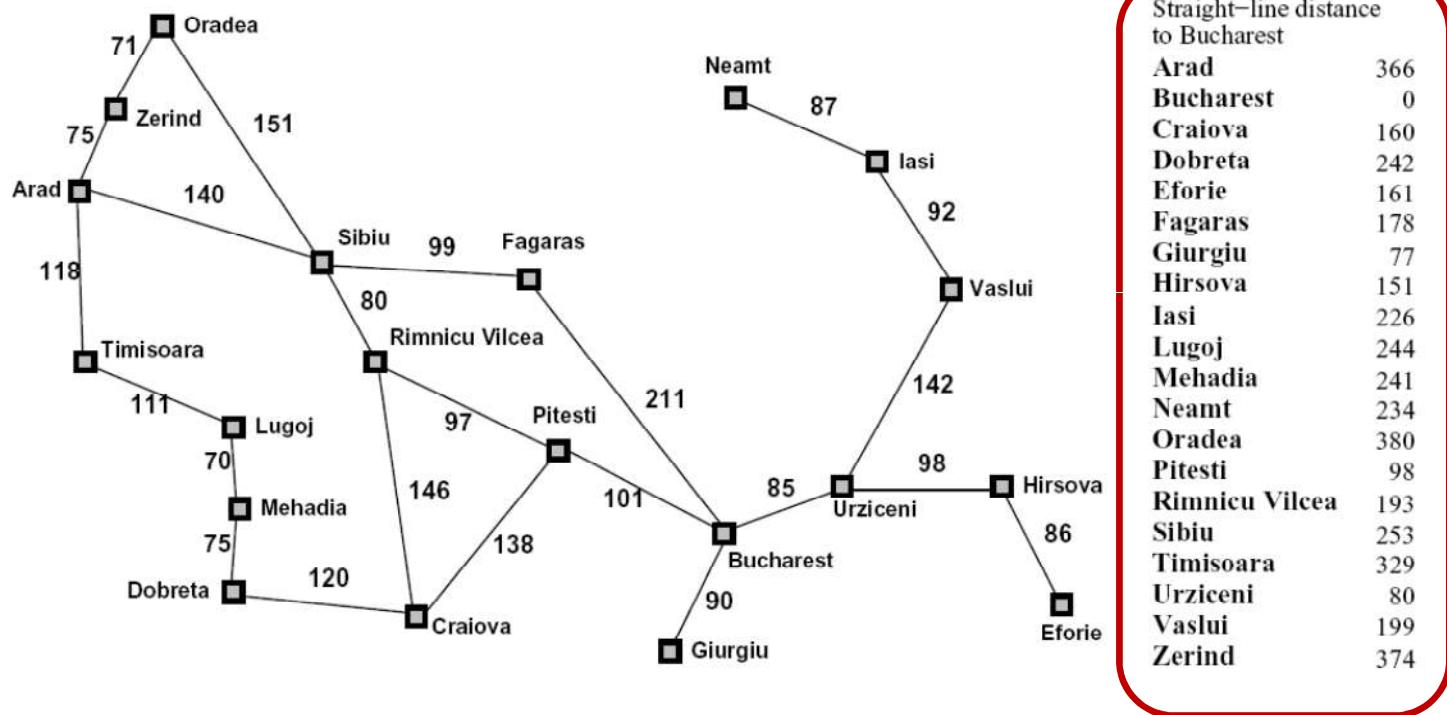
اینجا Heuristic Function چی می تونه باشه؟
فاصله رو بسنجیم
بازم نفهمیدم://

Greedy Search



این Greedy Search از همون Heuristic Function کمک میگیره و میگه که هر موقع خواستی نودی رو انتخاب بکنی یک نودی رو انتخاب بکن که فاصله اش به هدف نزدیکتره

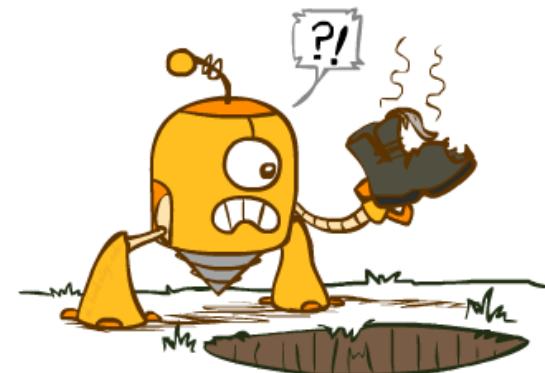
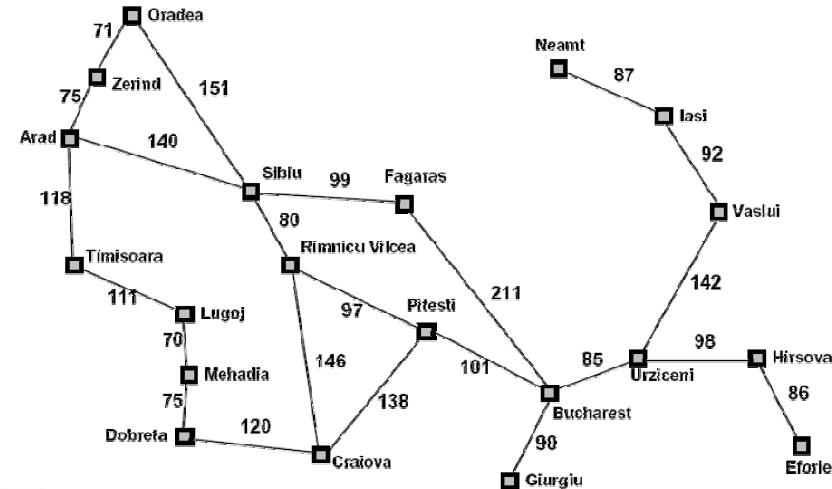
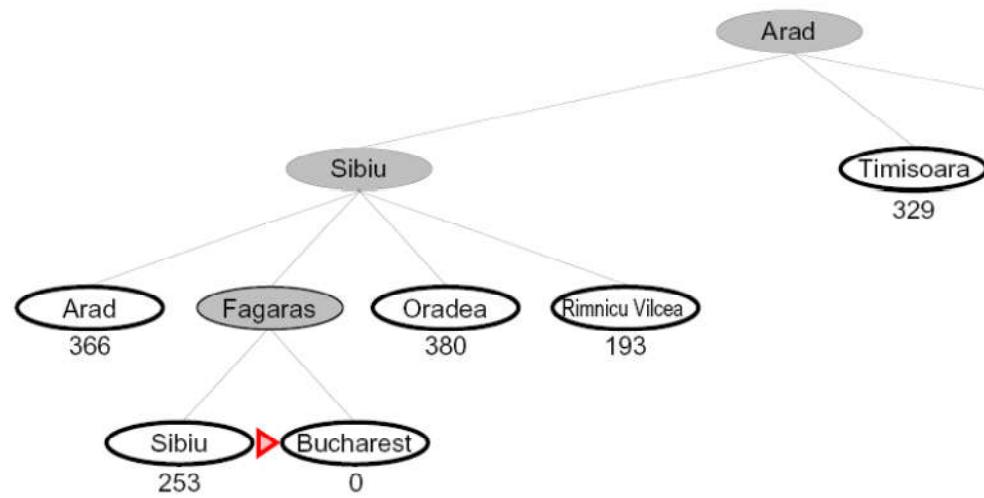
Example: Heuristic Function



$h(x)$

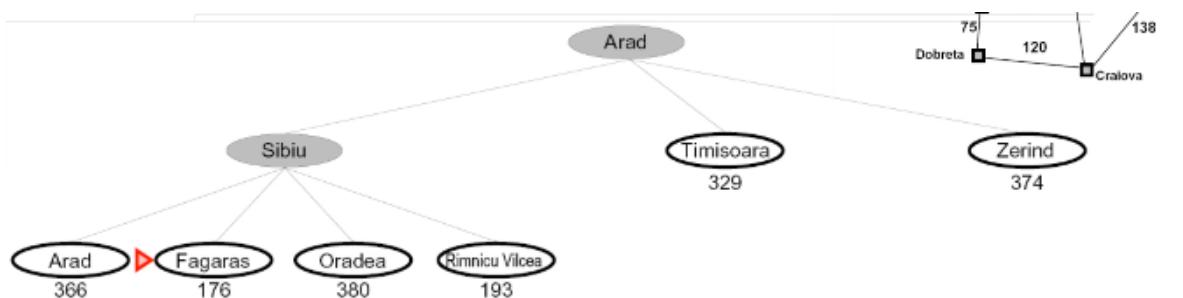
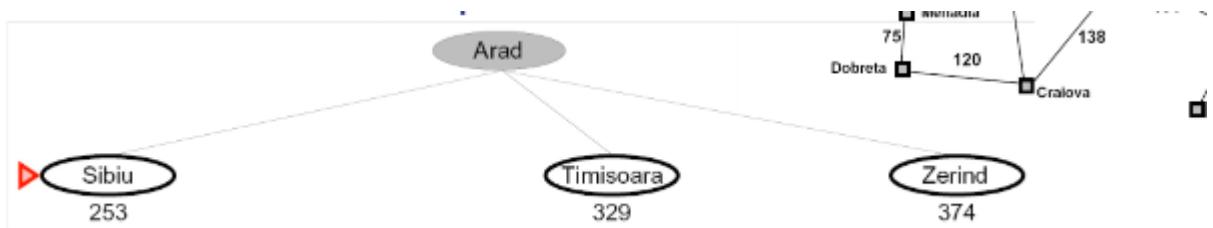
Greedy Search

- Expand the node that seems closest...
- lowest heuristic value for expansion



- What can go wrong?

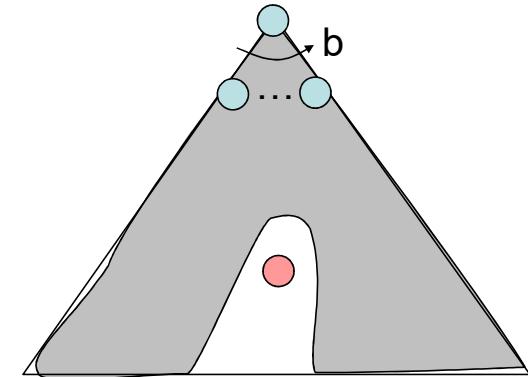
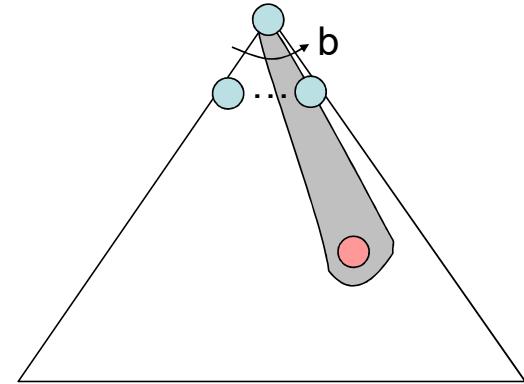
هر کدام یکی از این سه تا شهر یک فاصله ای از هدف دارد:



حالا این مسیر، مسیر خوبیه و بهینه است؟ نه چون مسیر کمینه ای نیست؟؟

Greedy Search

- Strategy: expand a node that you think is closest to a goal state
 - Heuristic: estimate of distance to nearest goal for each state
 - computed backward cost vs estimated forward cost
- A common case:
 - Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS



[Demo: contours greedy empty (L3D1)]

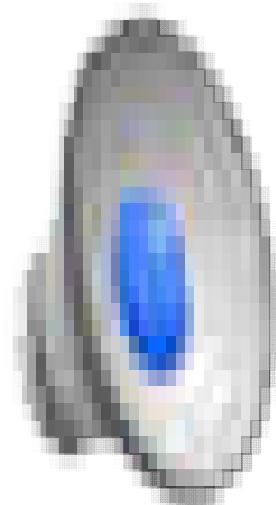
[Demo: contours greedy pacman small maze (L3D4)]

رویه ای که توی گرید سرچ داریم اینه که نگاه میکنه نودها توی چه جهتی از استیت هدف هستن و توی همون جهت عمیق میشه و بهترین ها میشن همون هایی که به هدف نزدیکترن

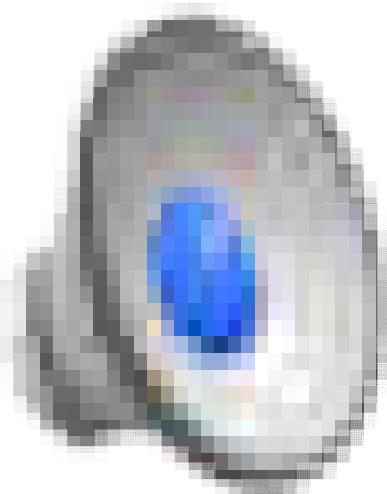
واقعیتی که هست اینه که بعضی وقتا که بخوایم با این فرض حرکت بکنیم ممکنه به یک شرایطی برسیم که اصلا نرسیم یا مثلًا جاده ای اونجا نباشه و ممکنه برگردیم

- بهینه؟
- نه، مسیر ساوه بهتر است.
- کامل؟
- به علت مشکلات گفته شده در فضای حالت نامحدود کامل نیست.
- رفتاری همانند عمق نخست
- پیچیدگی زمان و حافظه در بدترین حالت $O(b^m)$
- m حداکثر عمق فضای جستجو

Video of Demo Contours Greedy (Empty)

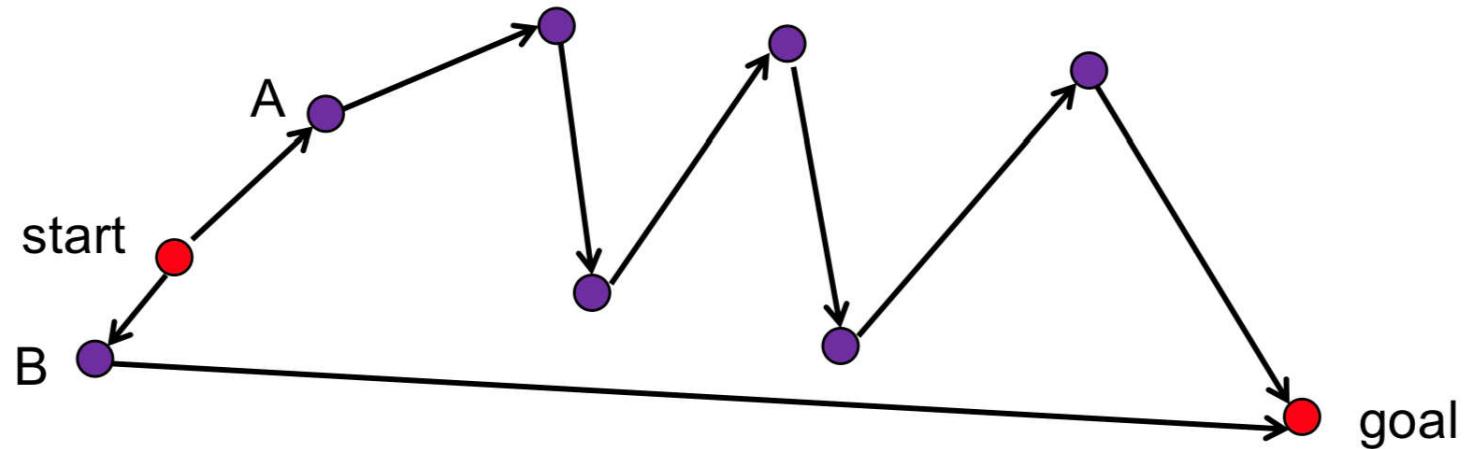


Video of Demo Contours Greedy (Pacman Small Maze)



Greedy Search

- Expand the node that seems closest...

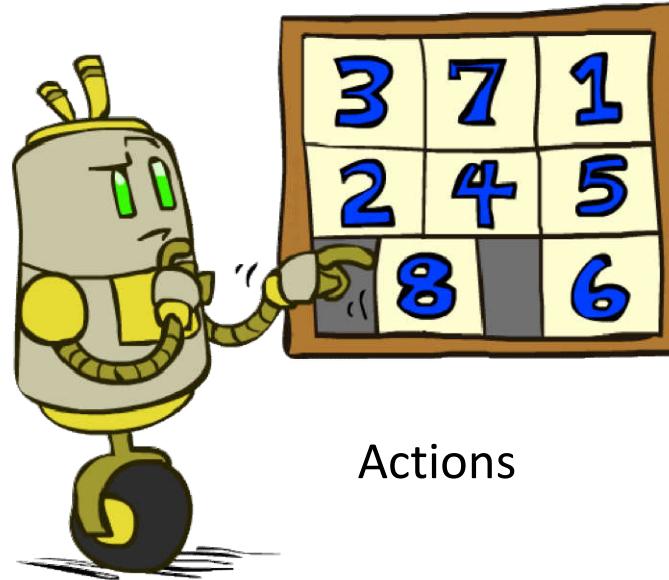


توی حالت استارت یا توی حالت های بعد از استارت انگار با یک شرایطی طرف میشیم که باید دورتر بشیم از هدف ینی باید نودی رو انتخاب بکنیم که دورتر است و توی این حالت ها گرید سرچ میاد نودی رو انتخاب میکنه که هی داریم در اون جهت رسیدن به هدف ما رو کمک میکنه

Example 2: Eight Puzzle

7	2	4
5		6
8	3	1

Start State



Actions

	1	2
3	4	5
6	7	8

Goal State

- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

مثال دیگری از گرید سرچ:

هشت پازل: میخوایم این پازل ها رو مرتب کنیم

استیت میشه چیش این خونه ها --> تعداد حالت هاش میشه؟

اکشن ها:

هزینش: هر دفعه که حرکت میکنیم

Example 2: Eight Puzzle

- The task is to transform the initial puzzle

2	8	3
1	6	4
7		5

این وضعیت اسکارت ما است

- into the goal state

1	2	3
8		4
7	6	5

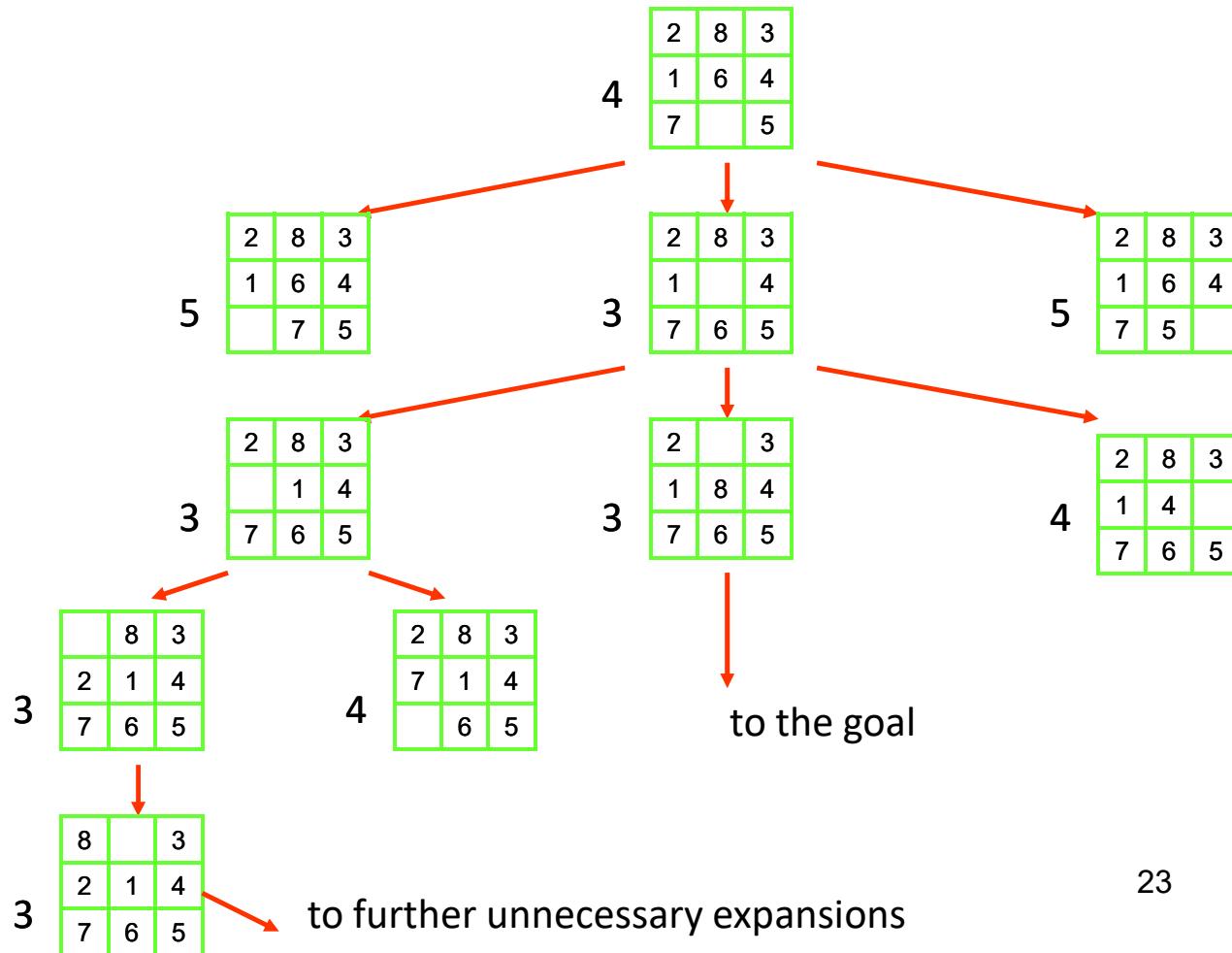
و این وضعیت هدف ما است

- by shifting numbers up, down, left or right.

حالا میخوایم یک Heuristic برای این پیدا بکنیم:

Heuristic Function خوب برای این میشه که اگر قرار باشه ما به این حالت هدف بررسیم اونایی که جاشون درست است رو کاری نداریم و اونایی که جاشون درست نیست رو باید درست کنیم پس می تونیم به تعداد شماره هایی که سر جای خودشون نیستن می تونیم امتیاز بهشون بدیم و بگیم چقدر به هدف نزدیک شدیم مثلا اگر همه اینا سر جای خودشون نباشن ما 9 امتیاز رو میگیرن

Example 2: Eight Puzzle



اینجا سه تا اکشن داریم که این خونه رو کجا ببریم: ببریم چپ یا بالا یا راست

اتفاقی که می افته: توی این مسیری که داریم می ریم توی اون دور سه تایی هی گیر می افته ینی هر چقدر اون حالت رو توسعه بدیم توی وضعیت دور سه تایی گیر میکنه ینی کمتر از 3 هیچ وقت پیدا نمیکنه

Properties of Greedy Search

- **Complete:**
 - Generally No, (Yes, in finite space with repeated-state checking)
- **Time:**
 - $O(b^m)$, but a good heuristic can give dramatic improvement
- **Space:**
 - $O(b^m)$ —keeps all nodes in memory!(find best heuristic)
- **Optimal:**
 - No

ایا کامله: اگر توی اون لوپ بیوشه می تونه نرسه و یا اینکه از اون استیت دور بشه و مارو بندازه توی اون لوپه پس در کل نمیشه

از لحاظ زمانی و حافظه: b به توان m^{--} همه گره ها رو داره انالیز میکنه ینی توی بدترین حالت اینه که همه نودها رو انالیز بکنه و از اونجایی که همه نودها رو باید نگه داره توی حافظه بخاطر اینکه منطق خود الگوریتم بود چون نیاز داریم حداکثر همه نودها رو داشته باشیم که ببینیم مقدار **Heuristic** کدوم کمتره و اونو برداریم

است؟ وقتی که کامل نیست پس **Optimal** هم نیست

A* Search



A* Search

1

ایده پشت این الگوریتم: میخواد از محاسن الگوریتم های دیگه کمک بگیره و ما رو برسونه به اون هدف

گریدی خیلی سریع عمل می کرد
و **Uninformed Search** خیلی اهسته
حالا میخوایم این دوتا رو با هم ترکیب کنیم چجوری؟ صفحه بعد

A*: the core idea

- Expand a node n most likely to be on an optimal path
- Expand a node n with lowest value of $g(n) + h^*(n)$
 - $g(n)$ is the cost from root to n
 - $h^*(n)$ is the optimal cost from n to the closest goal
- We seldom know $h^*(n)$ but might have a heuristic approximation $h(n)$
- A* = tree search with priority queue ordered by $f(n) = g(n) + h(n)$

Uninformed Search یک تابعی داشت و براساس اون تابع انتخاب می کرد و گریدی هم یک تابعی داشت و براساس اون تابع انتخاب می کرد پس حالا این دوتا رو با هم جمع میکنیم:

توی این a^* وقتی یک نودی رو میخوایم انتخاب بکنیم برای **expand** کردن بیانودها رو اینجوری مرتب بکن که هر نودی و هر استیتی فاصله اش تا حالت شروع چدره که اینو با g نشون میدیم که هزینه ما میشه که به اون حالت رسیدیم

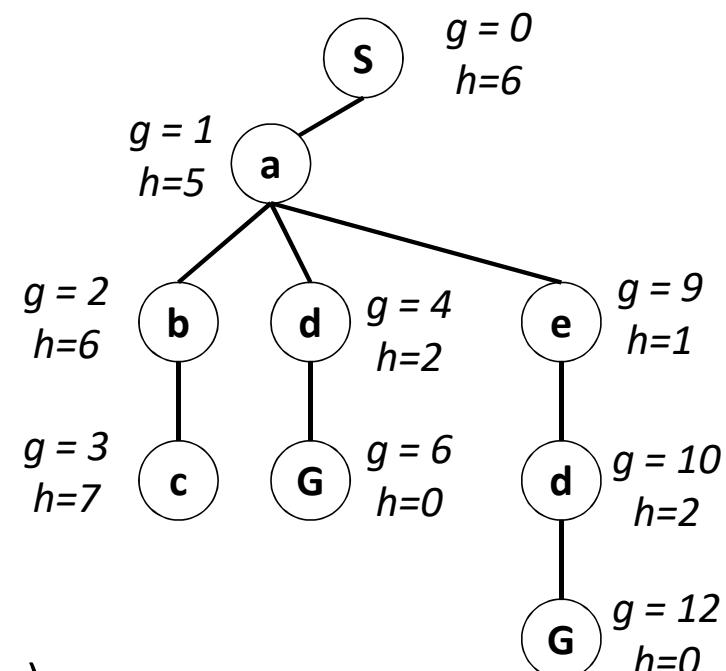
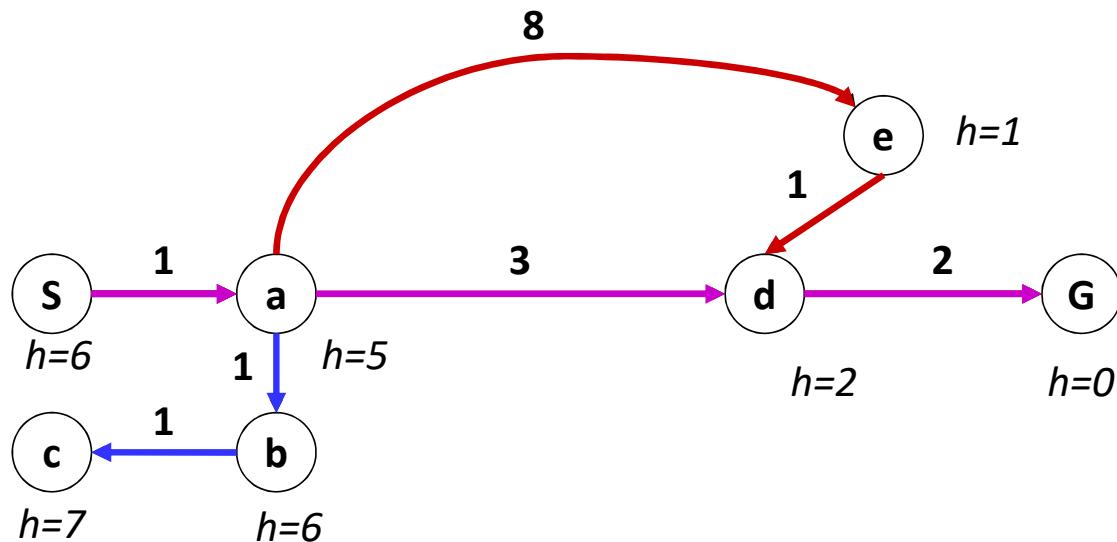
در عین حال هم یک هیوریستکی داریم که این نود اگر بخواهد به هدف برسه چقدر فاصله داره که میشه h^* حالا بیا این دوتا رو با هم جمع کن نکته: در واقع این h^* کمترین هزینه یا همون بهینه ترین حالته که از نود n می تونیم ببریم به نود هدف

مسئله ای که اینجا پیش میاد اینه که ما همیشه هزینه دقیق رو نداریم یعنی توی خیلی از مسئله ها هزینه دقیق رو نداشتیم و اونو تخمین می زدیم یعنی h^* رو باید تخمین بزنیم که این کارو با یک **heuristic approximation** انجام میدیم

پس الگوریتم a^* میاد از هزینه ای که طی شده توی هر نود + تخمینی از فاصله تا جواب و این رو به عنوان یک مبنای قرار میده که نودها رو انتخاب بکنه

Combining UCS and Greedy

- Uniform-cost orders by path cost, or *backward cost* $g(n)$
- Greedy orders by goal proximity, or *forward cost* $h(n)$



- A* Search orders by the sum: $f(n) = g(n) + h(n)$

Example: Teg Grenager

هدف اینه که از s بریم به g :

ینی کوتاه ترین مسیر رو پیدا بکنیم که از s بریم به g

از روش Uniform-cost بریم:

g رو در نظر می گیریم:

$s \rightarrow a \rightarrow b \leftarrow c \rightarrow d \rightarrow g$

الان به هدف رسیدیم پس مسئله حل شده و دیگه ادامه نمی دیم

از روش گریدی سرچ بریم:

اینجا h رو در نظر میگیریم:

$s \rightarrow a \rightarrow e \rightarrow d \rightarrow g$

تموم شد

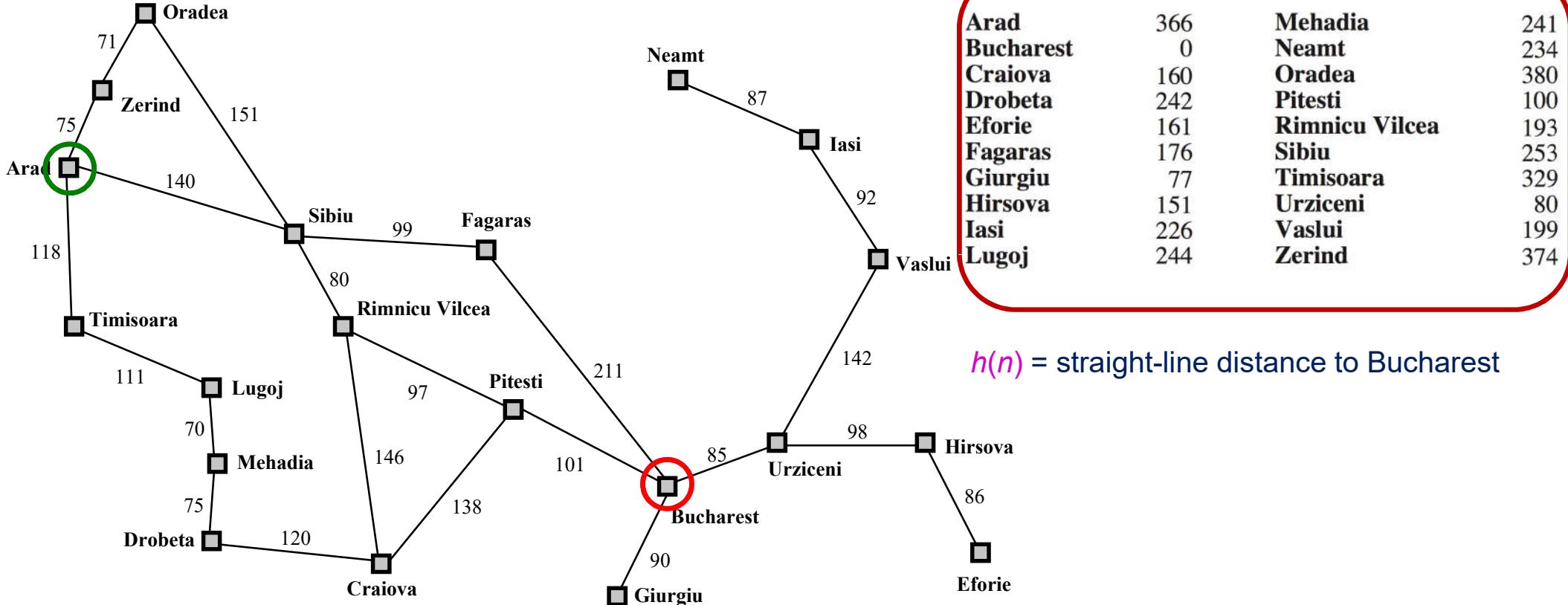
ترکیب این دو تا:

کم باشن: g, h

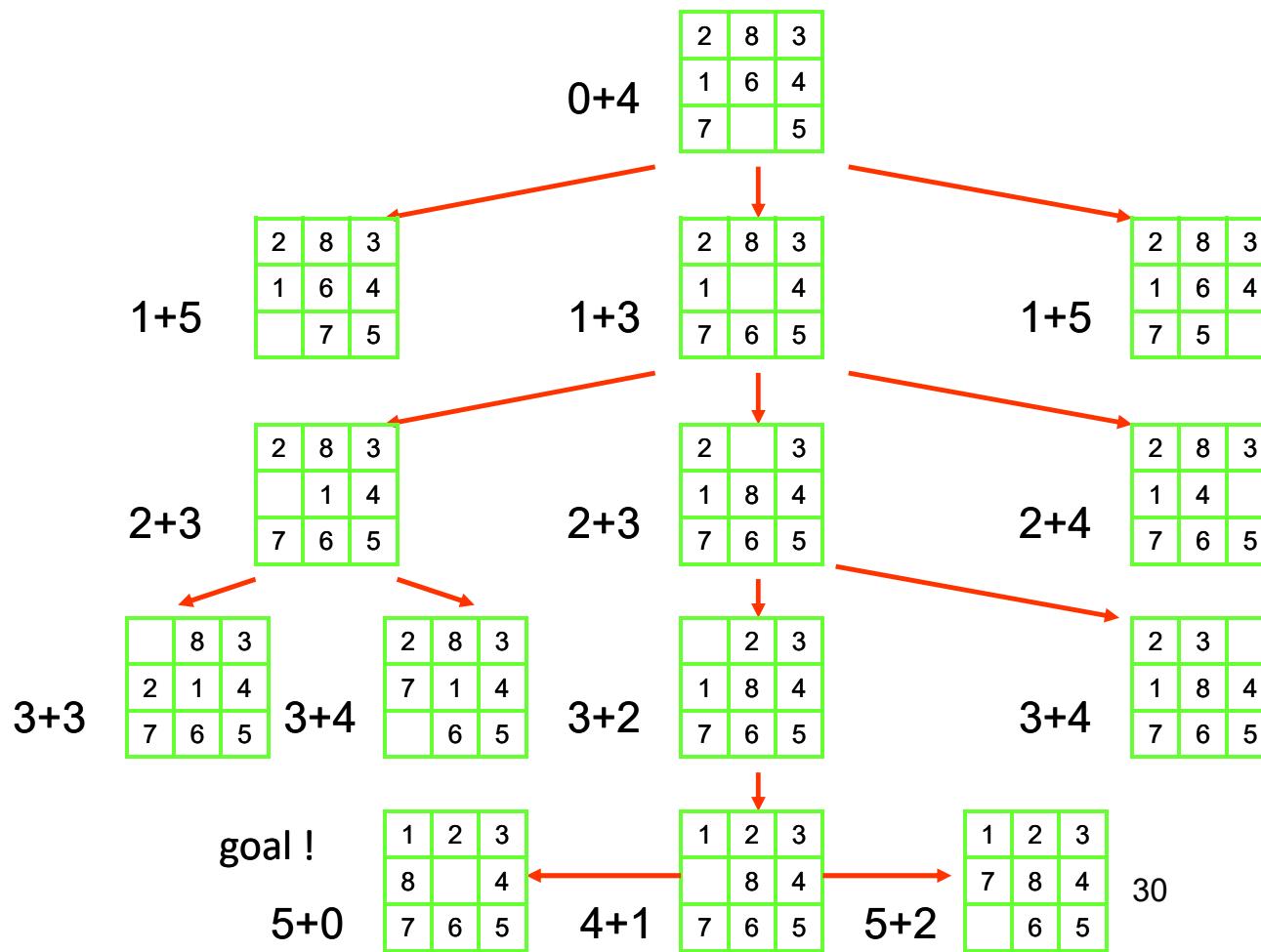
$s \rightarrow a \rightarrow d \rightarrow g$

تموم شد

Example: route-finding in Romania

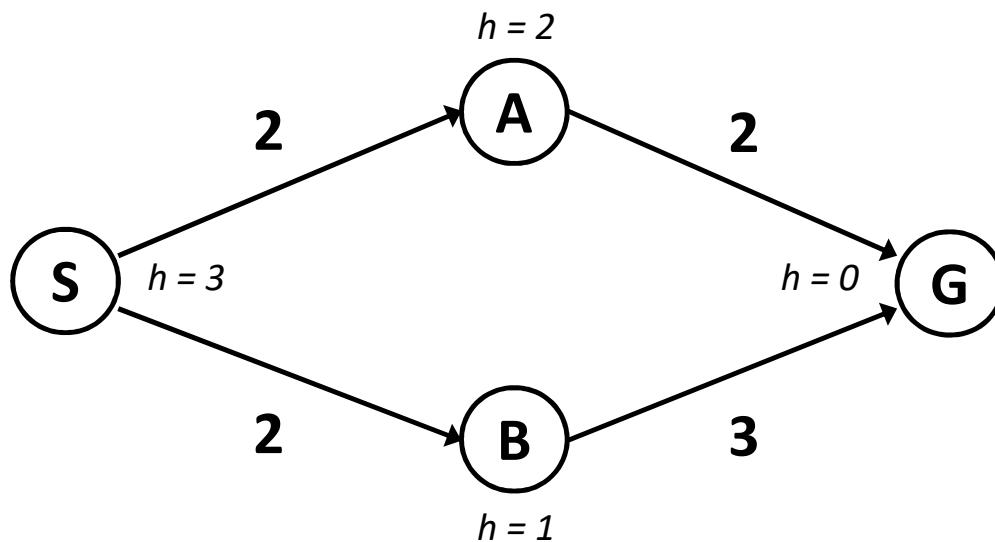


Example 2: Eight Puzzle



When should A* terminate?

- Should we stop when we enqueue a goal?



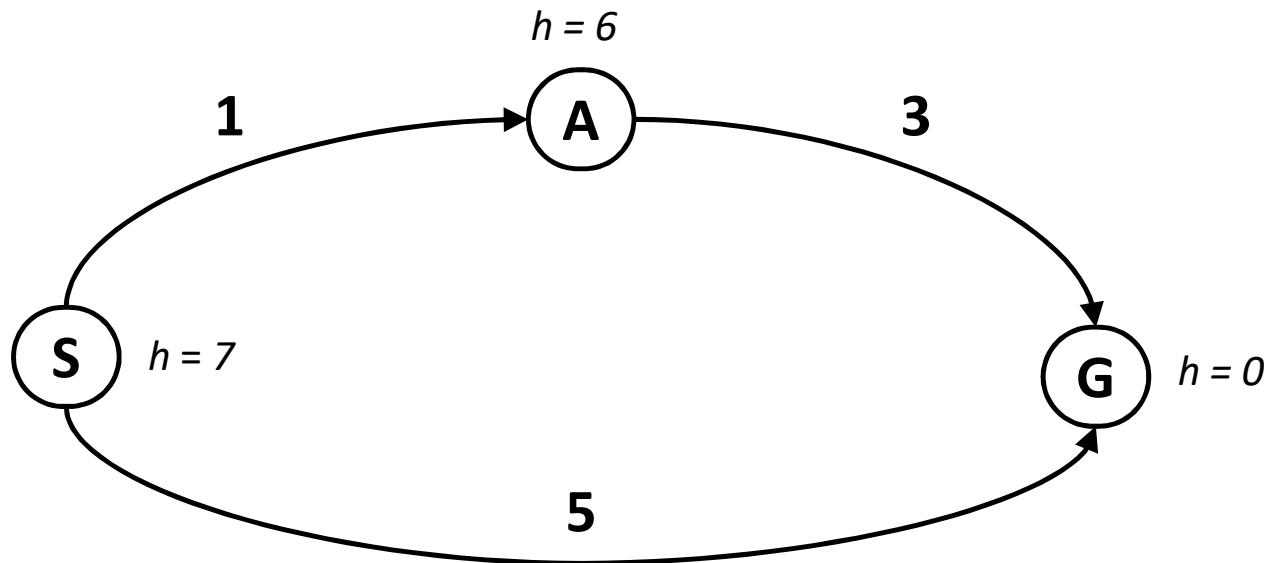
- No: only stop when we dequeue a goal

همچین گرافی داریم و میخوایم با a^* روش حرکت کنیم و به جواب بررسیم چه اتفاقی می‌افته اینجا؟
به محض اینکه استیت هدف رو دیدیم بگیم ما رسیدیم به هدف یا نه اجازه بدیم که اون هم حساب
بکنه برای مرحله بعدی و اگه وجود نداشت بگیم رسیدیم به هدف؟؟؟؟ نفهمیدم ؟؟؟؟

جواب: ما حتما باید به استیت هدف بررسیم

من باز نفهمیدم ولی مهمه بفهمم اینو
؟؟

Is A* Optimal?



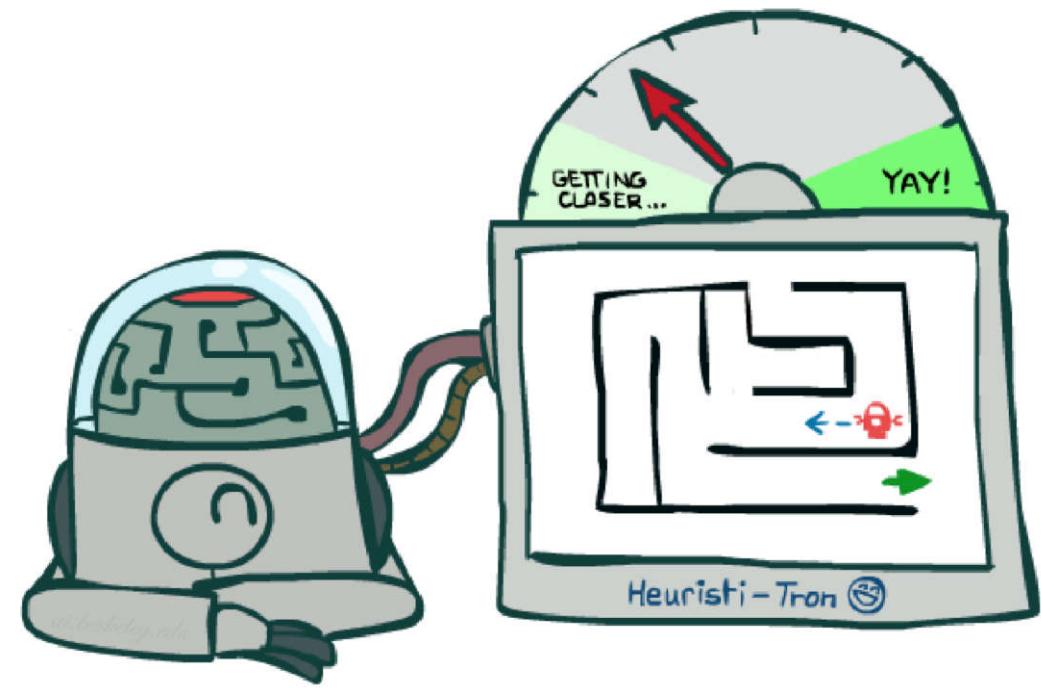
- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

مثال:

از s حرکت میکنیم دو تا نود داریم یکی a و یکی g الان:
تابع g میشه ۵ و a میشه ؟؟

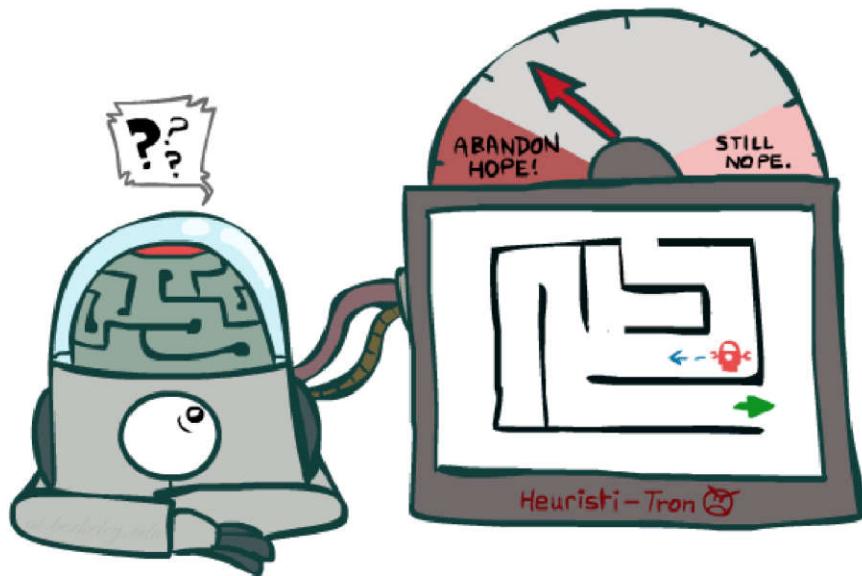
الان اینجا شرایط terminate شدن رو داریم ینی به نودی رسیدیم که هم f اش کمینه است و هم g هست

ADMISSIBLE HEURISTICS

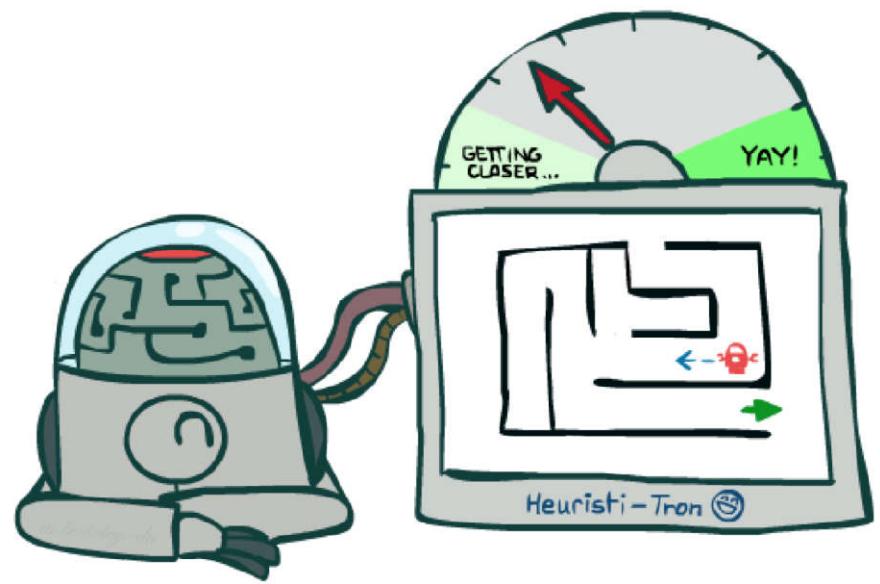


نکته: a^* به شرطی اینکه HEURISTICS ما یک شرایطی داشته باشند می تونه به جواب ما رو برسونه و حتی optimistic به جواب برسونه

Idea: Admissibility



Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe



Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs

و اگر ما heuristics بد داشته باشیم اتفاقی که می افته اینه که رفتار الگوریتم مثل گردید سرج میشه

اگر heuristics خوب باشه ما توی بهینه ترین حالت به جواب می رسیم از لحاظ اینکه مسیر بهینه باشه کاری الان به مموری اینا نداریم

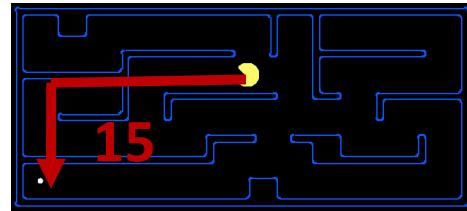
Admissible Heuristics

- A heuristic h is *admissible* (optimistic) if:

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

where $h^*(n)$ is the true cost to a nearest goal

- Examples:



- Coming up with admissible heuristics is most of what's involved in using A* in practice.

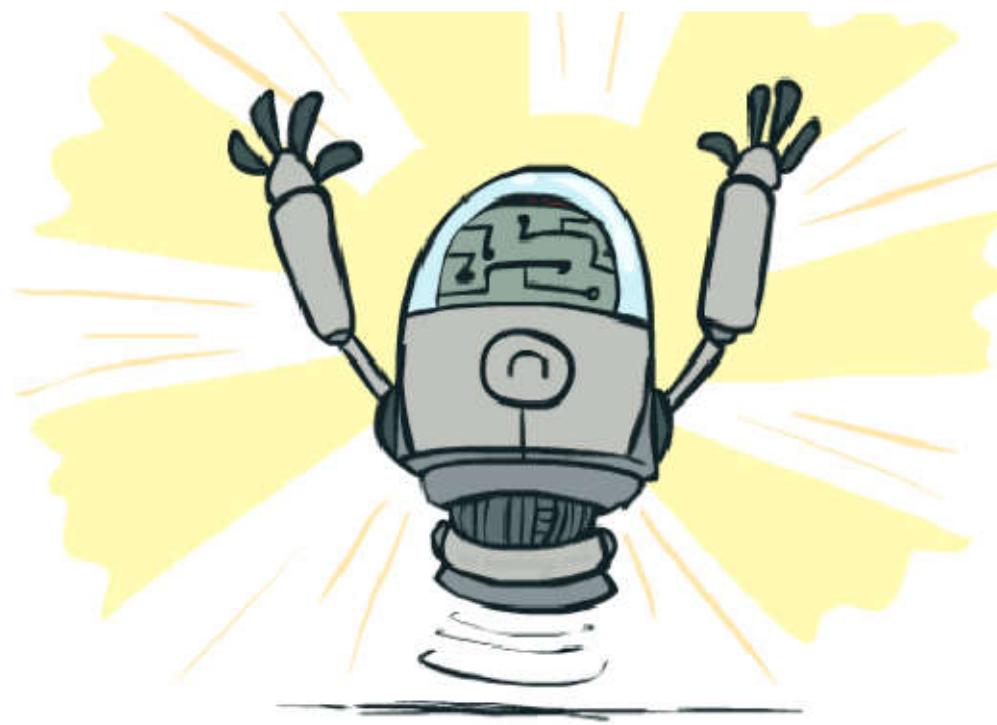
اگر هزینه واقعی هر نود برای رسیدن به نود هدف رو داشته باشیم هر heuristics که پیشنهاد بدیم این مقدار heuristics باید از مقدار واقعی همیشه کمتر باشد

مثلًا برای این مثال ما تابع heuristics رو جمع فاصله طول و عرضی که از مقصد داره در نظر گرفتیم ینی $15 - \lceil \frac{x}{10} \rceil$ به شرط اینکه هیچ دیواری بین این ها نباشه اره با ۱۵ تا می رسه ولی اگر یک دیوار باشه توی این مسیر اون هزینه ما برای رسیدن به هدف بیشتر از ۱۵ تا میشه و همین باعث شد این تابع heuristics که اینجا استفاده کردیم یک heuristics قابل پذیرفته شدن یا admissible باشد

این تابع heuristics رو چجوری پیدا میکنیم؟ توی مسئله این کار طراح مسئله است

نکته: هر چقدر به صفر نزدیک بشیم انگار داریم از قدرت A^* کم میکنیم

Optimality of A* Tree Search



میخوایم اثبات بکنیم الگوریتم a^* بهینه است:

جواب بهینه ینی چی؟ ینی اگر ما به یک نود هدف رسیدیم و الگوریتم ترمینیت شد واقعا مسیر دیگه ای که به این نود هدف بررسیم وجود نداشته باشه

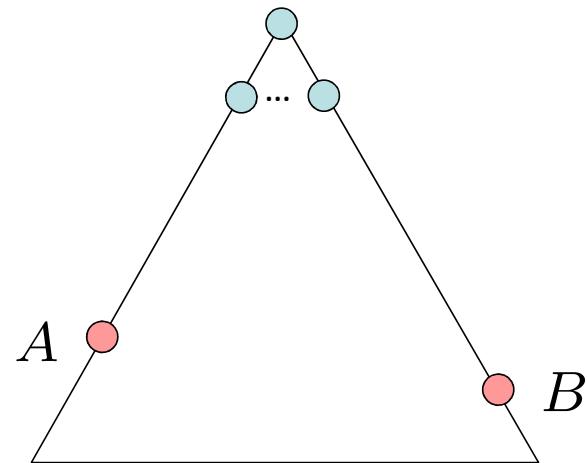
Optimality of A* Tree Search

Assume:

- A is an optimal goal node
- B is a suboptimal goal node
- h is admissible

Claim:

- A will exit the fringe before B



اگر همچین درخت سرچی داشته باشیم و دو تا نود هدف هم داشته باشه:
نود A هزینه کمتری داره و مسیر بهینه اینه که ما از ریشه زودتر بررسیم به A با هزینه کمتری
ولی اگر الگوریتم بهینه نباشه ما رو می رسوونه به B

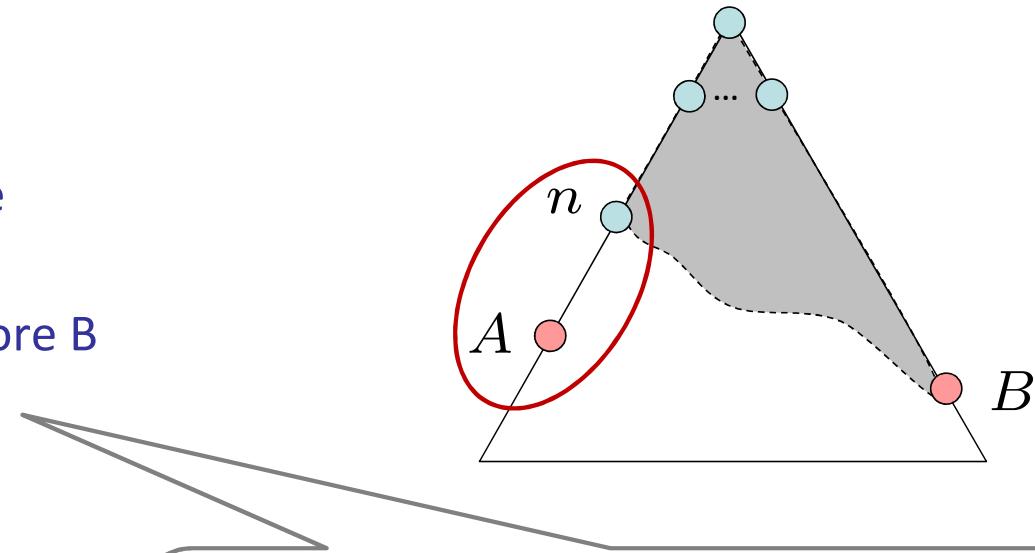
با برهان خلف اینو اثبات میکنیم یعنی فرض میکنیم که بررسه به یک نود دیگه ای به نام B و این نود B از نوع خودش یک نود هدف است که الگوریتم توش ترمینیت شده و برهان خلف نشون میدن که تحت این شرایط با این فرض هایی که ما کردیم منطبق نیست

فرض میکنیم می رسه به نود B این فرضمون است و نشون میدیم که با a^* نمی تونیم به حالت B
بررسیم

Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded before B
 1. $f(n)$ is less or equal to $f(A)$



$$\begin{array}{ll} f(n) = g(n) + h(n) & \text{Definition of f-cost} \\ f(n) \leq g(A) & \text{Admissibility of } h \\ g(A) = f(A) & h = 0 \text{ at a goal} \end{array}$$

$g(A)$ هزینه واقعی است و $f(n)$ همیشه از $g(A)$ کمتر است

1. $f(n) \leq f(A)$

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

$$\begin{aligned} f(n) &= g(n) + h(n) \\ f(n) &\leq g(A) \end{aligned}$$

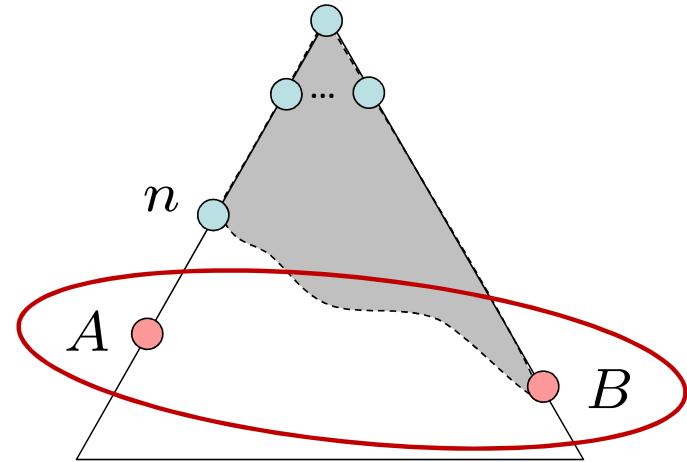
Definition of f -cost
Admissibility of h



Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded before B
 1. $f(n)$ is less or equal to $f(A)$
 2. $f(A)$ is less than $f(B)$



$$g(A) < g(B)$$

$$f(A) < f(B)$$

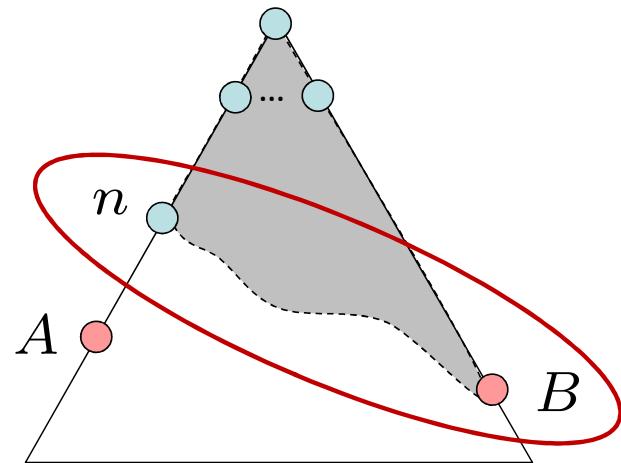
B is suboptimal

$h = 0$ at a goal

Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded before B
 1. $f(n)$ is less or equal to $f(A)$
 2. $f(A)$ is less than $f(B)$
 3. n expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal

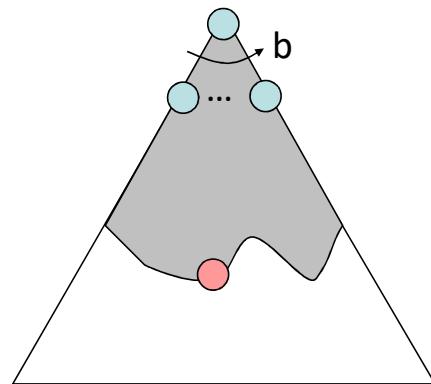


$$f(n) \leq f(A) < f(B)$$

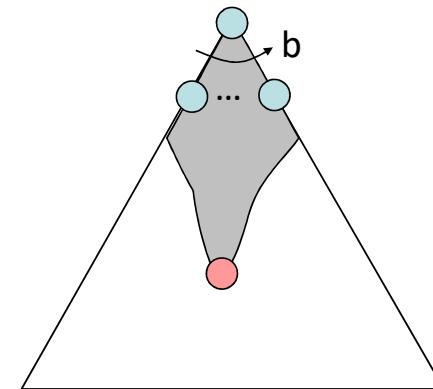
Properties of A*

Properties of A*

Uniform-Cost

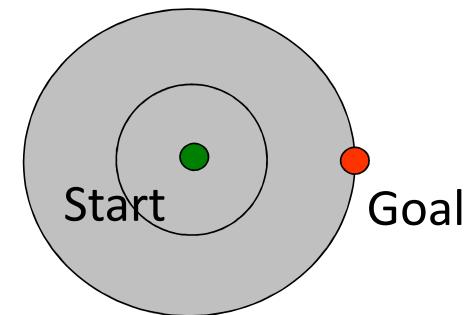


A*

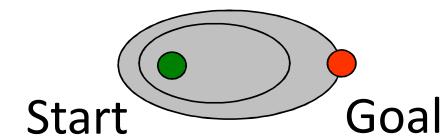


UCS vs A* Contours

- Uniform-cost expands equally in all “directions”



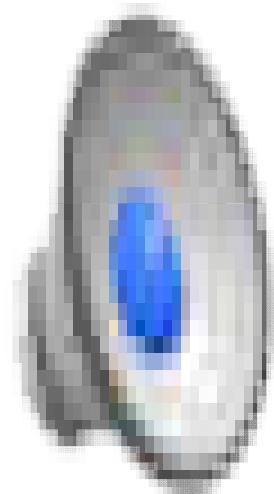
- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



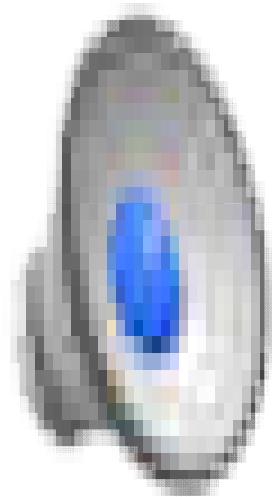
[Demo: contours UCS / greedy / A* empty (L3D1)]
[Demo: contours A* pacman small maze (L3D5)]

توی UCS حول این نود اولیه جستجو میکنه و این رو گسترش میده تا به نود هدف برسه ولی توی A^* این متمایز میشه به سمت هدف و همین باعث میشه سرعتش بالاتر بشه و از لحاظ optimality هم A^* و هم UCS جفتشون جواب رو پیدا می کن

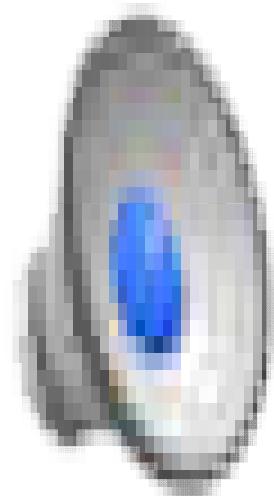
Video of Demo Contours (Empty) -- UCS



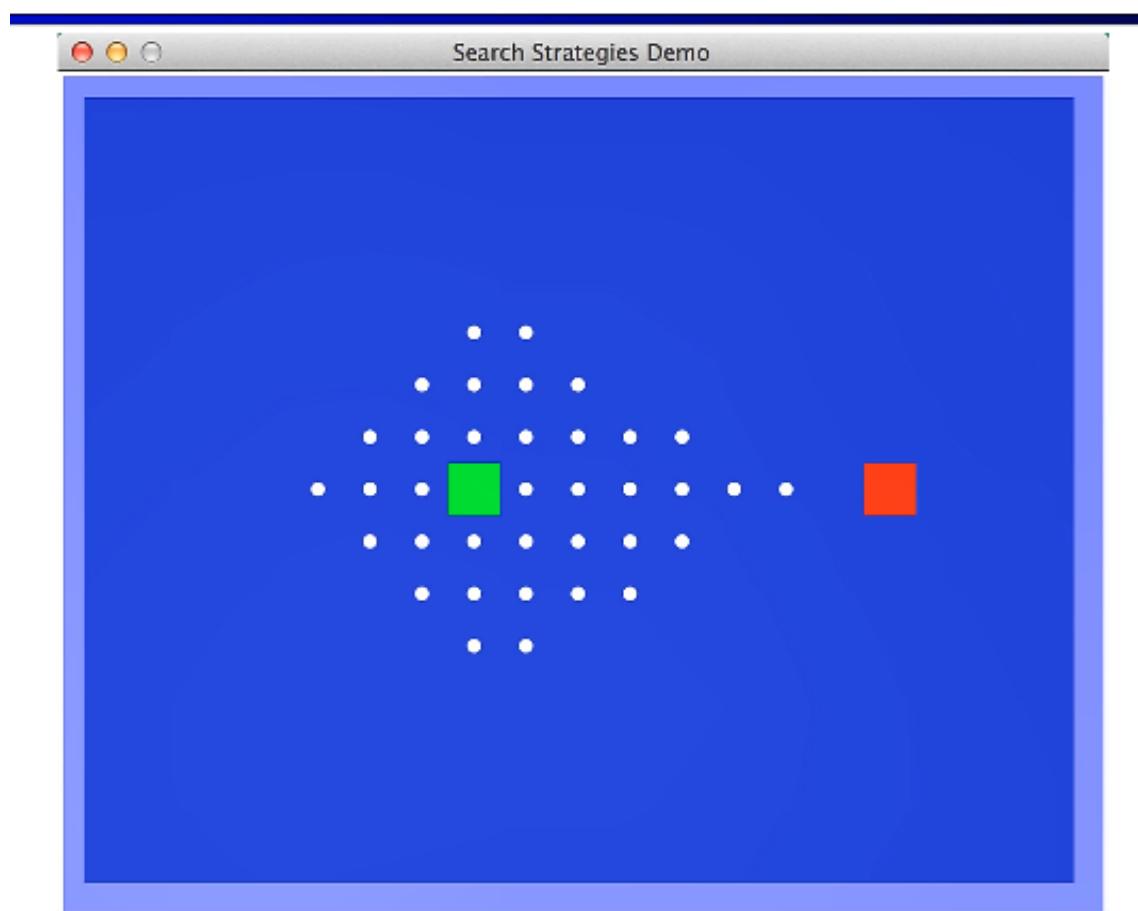
Video of Demo Contours (Empty) -- Greedy



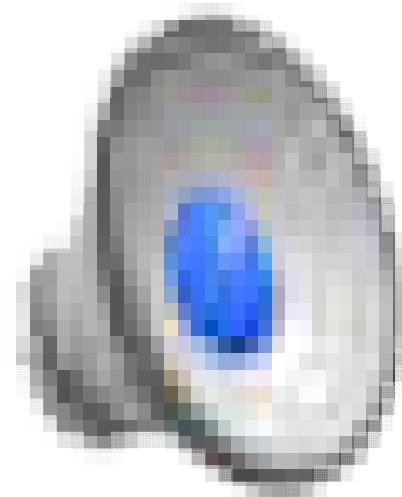
Video of Demo Contours (Empty) – A*



توی این روش ابتدای شروع کارش مشابه UCS است ولی هر چه جلوتر می‌ره اون heuristics خودش رو بیشتر نشون میده و در جهت نود هدف نودهای بیشتری رو expand میکنه و در نهایت مسیر optimality رو پیدا میکنه



Video of Demo Contours (Pacman Small Maze) – A*



Comparison



Greedy

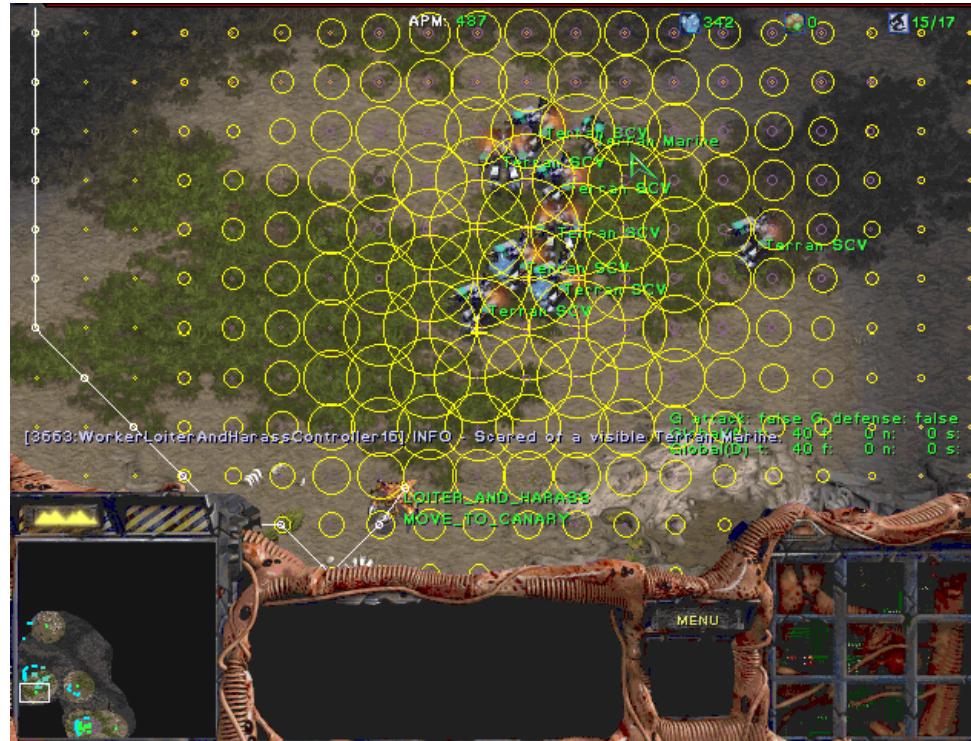


Uniform Cost



A*

A* Applications



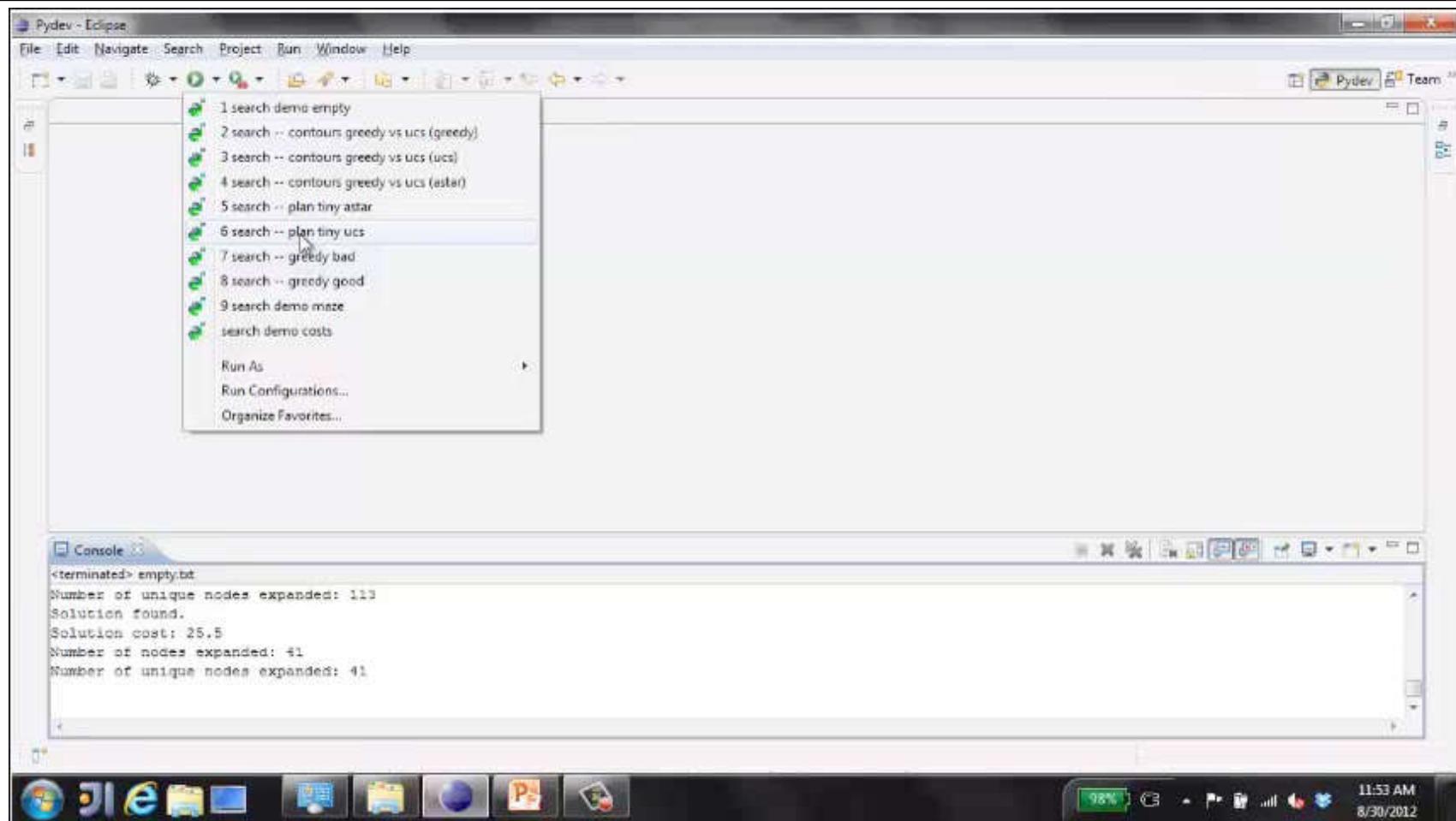
A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- ...

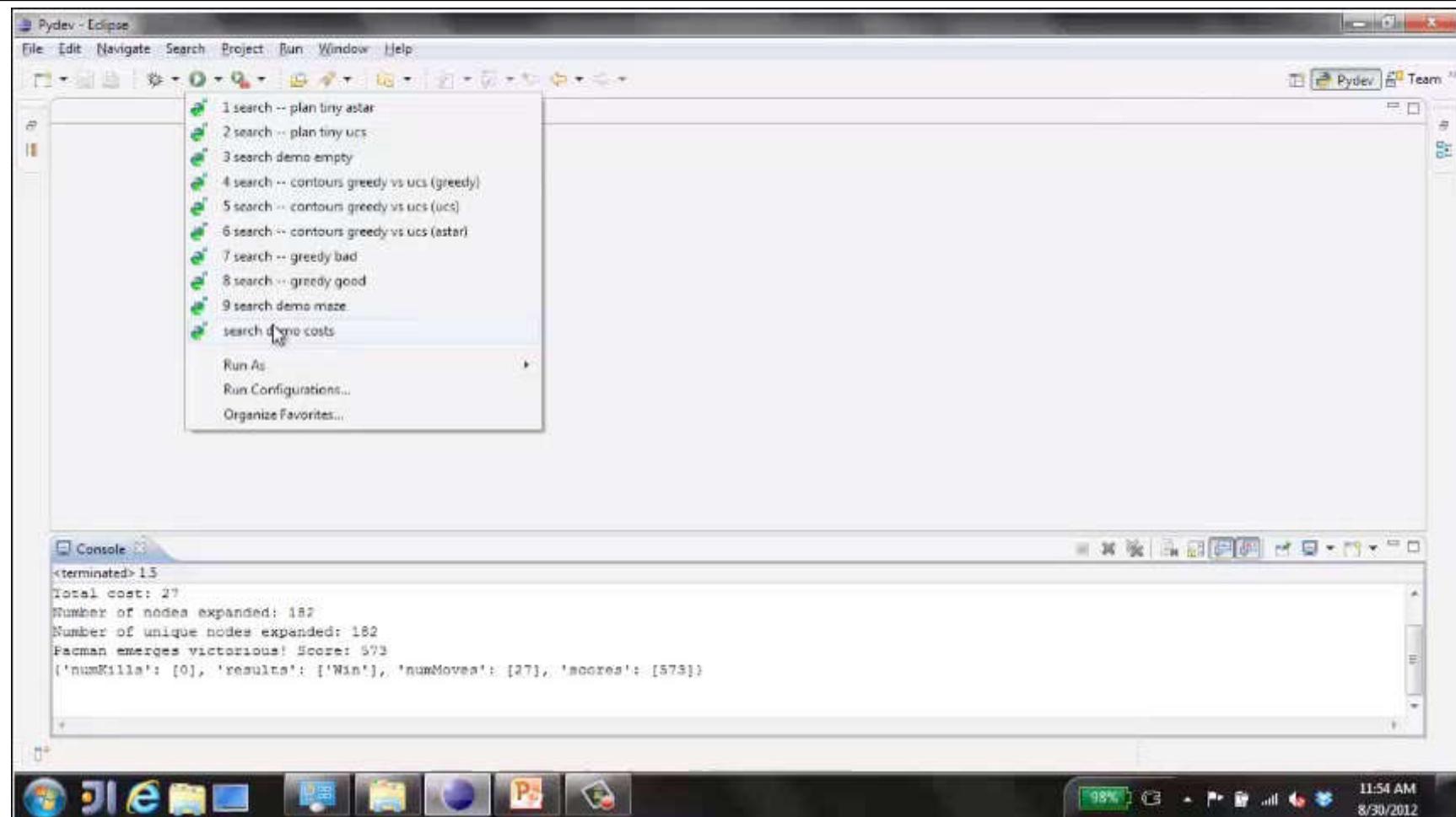


[Demo: UCS / A* pacman tiny maze (L3D6,L3D7)]
[Demo: guess algorithm Empty Shallow/Deep (L3D8)]

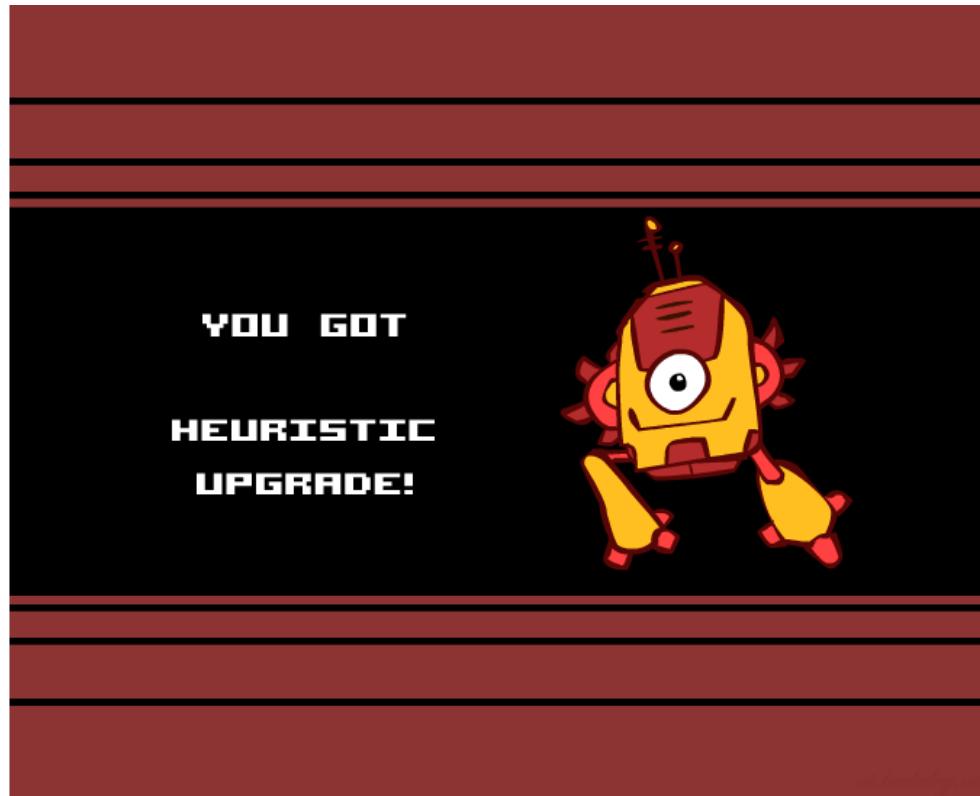
Video of Demo Pacman (Tiny Maze) – UCS / A*



Video of Demo Empty Water Shallow/Deep – Guess Algorithm

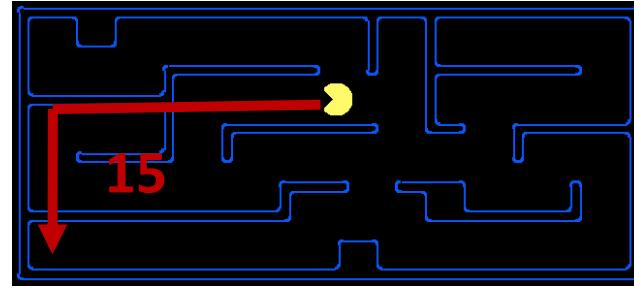
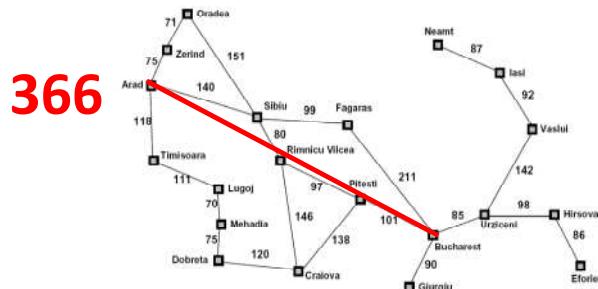


Creating Heuristics



Creating Admissible Heuristics

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to *relaxed problems*, where new actions are available



- Inadmissible heuristics are often useful too

اینجا برای تعریف Heuristics یک داستانی وجود داره به نام ریلکس کردن مسئله فانکشن ما چه شرایطی باید داشته باشه که پذیرفتی باشه؟ توی مسئله ریلکس میگن ما میتوونیم یکسری قیود رو حذف بکنیم و Heuristics جدید ایجاد بکنیم

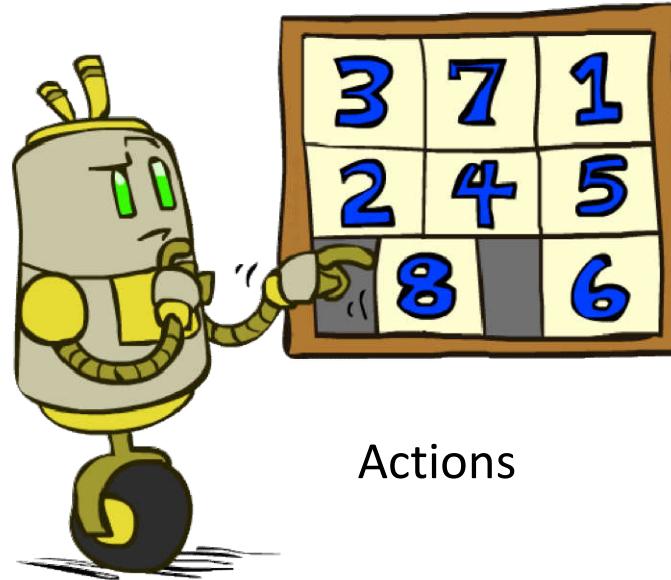
بعضی وقتا ما یک شرایطی داریم که این Heuristics فانکشن !!!

وقتی که داریم برای یک مسئله فضای حالتی در نظر میگیریم معمولاً داریم اینو با یک قیدهایی تعریفش میکنیم اگر این قیدها رو حذف بکنیم و بباییم توی یک فضای بزرگتری جستجو بکنیم لزوماً توی این فضای بزرگتری که داریم جستجو میکنیم جواب بهینه هم قرار داره بینی قرار نیست ما با حذف کردن قیدها به یک شرایطی برسیم که جواب بهینه نباشه

Example: 8 Puzzle

7	2	4
5		6
8	3	1

Start State



Actions

	1	2
3	4	5
6	7	8

Goal State

- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

مثال راجع به ریلکس کردن:

هزینه ای که میخوایم طی بکنیم که به نود هدف بررسیم باید چندتا کار انجام بدیم:

اگر قراره یک خونه ای رو حرکت بدیم حتما اون خونه رو به سمتی حرکت بدیم که یک خونه خالی باشه این یک قید است توی کاست ما پس ما خونه هایی رو می تونیم حرکت بدیم که اون خونه ها در مجاورت یک خونه خالی هستن

حتی اکشن های ما هم محدود میشه در واقع اکشن هایی رو می تونیم انجام بدیم که در مجاورت اون خونه خالی هستن

و نودهایی که مجاور هم هستن اصلا نمی تونن جابه جا بشن
پس سه تا شرط داریم اینجا

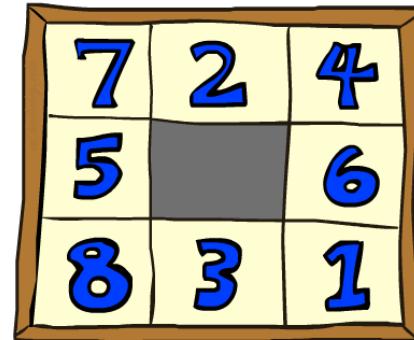
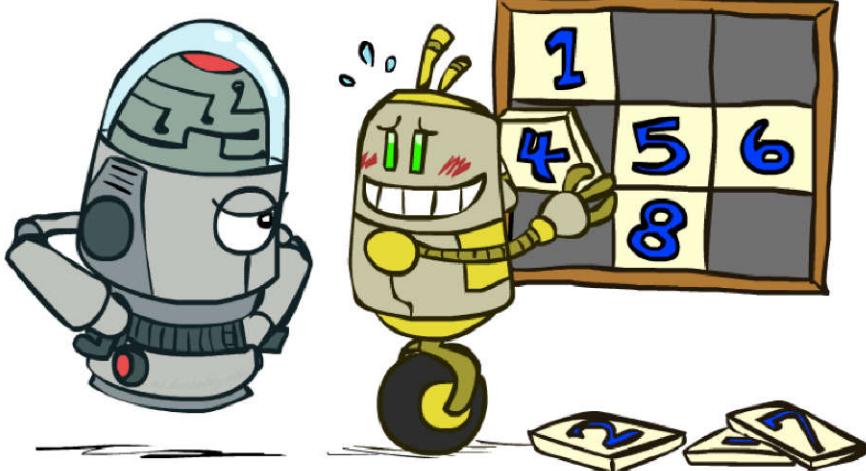
اگر این قیدها رو دقیق دقیق بکنیم اون فانکشن ایده ال رو برآمون درمیاره ولی چون یکم کار سخته
یه کاری کرد؟؟؟

اگر فانکشن 2 رو حذف بکنیم: اگر A رو بخوایم حرکت بدیم ببریم به یک نقطه دیگه اون نقطه باید خالی باشه حالا این فرض خالی بودن رو در نظر نگیریم

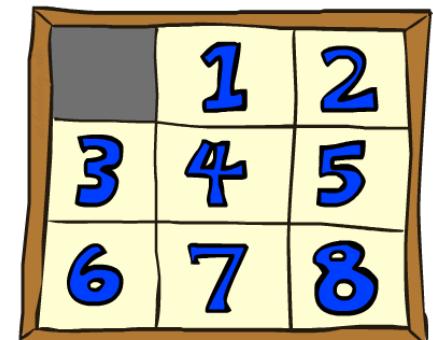
در واقع ما میخوایم الان برای این مسئله یک Heuristics فانکشن دیگه طراحی بکنیم:
صفحه بعد..

8 Puzzle I

- Heuristic: Number of tiles misplaced
- Why is it admissible?
- $h(\text{start}) = 8$
- This is a *relaxed-problem* heuristic



Start State



Goal State

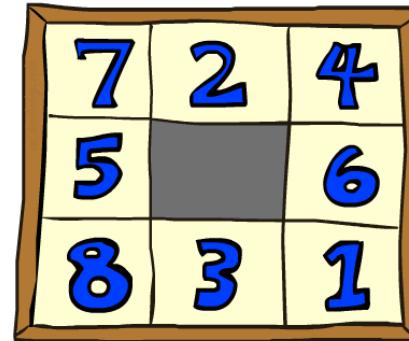
Average nodes expanded when the optimal path has...		
	...4 steps	...8 steps
UCS	112	6,300
TILES	13	39
		3.6×10^6
		227

Statistics from Andrew Moore

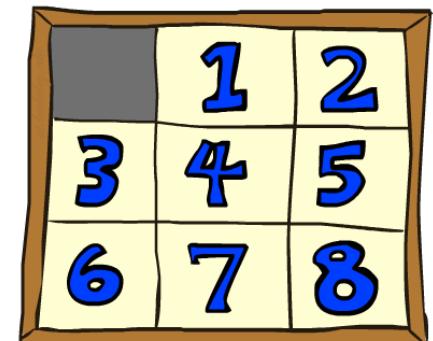
8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total *Manhattan* distance
- Why is it admissible?
- $h(\text{start}) = 3 + 1 + 2 + \dots = 18$

این میشه فاصله الان هر نود از اون نودی که توی استیت goal داره مثلای 7 برای 3 میشه



Start State



Goal State

	Average nodes expanded when the optimal path has...		
	...4 steps	...8 steps	...12 steps
TILES	13	39	227
MANHATTAN	12	25	73

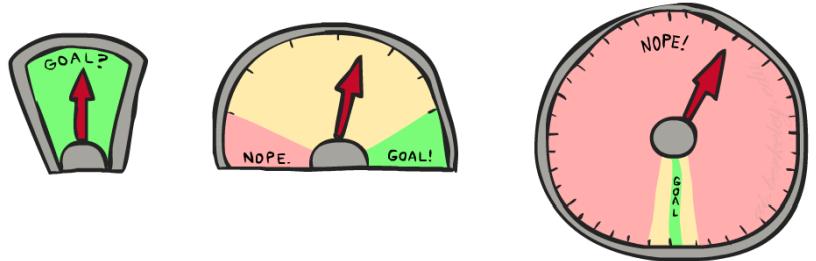
ما ببایم خونه هایی که سر جای خودشون نیستن رو از اونجایی که هستن رو بسنجیم:
که به این میگیم Manhattan distance

ایا این روش **admissible** هست؟؟؟

8 Puzzle III

- How about using the *actual cost* as a heuristic?

- Would it be admissible?
- Would we save on nodes expanded?
- What's wrong with it?



- With A*: a trade-off between quality of estimate and work per node

- As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

?????????????????????????????????

برای اینکه بخوایم heuristic های مختلف رو با هم مقایسه بکنیم یک شاخصی هست به نام :effective branch factor or EBF

این بهمون میگه که این الگوریتم چقدر داره کارا عمل میکنه توی سرچ یک الگوریتم رو میایم اجرا میکنیم به طور تصادفی چندین بار یک مقداری زمان می بره که این به حالت استیت goal برسه اون رو میان به کمک این دنباله حل میکن و b^* رو به دست میارن

Effective Branch factor_EBF : b^*

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

Example:

Depth 5

$N = 52$

$$53 = 1 + b^* + (b^*)^2 + \dots + (b^*)^5$$

$$b^* = 1.92$$

Lower B (Near 1) is more desirable !!

Semi-Lattice of Heuristics

Trivial Heuristics, Dominance

- Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

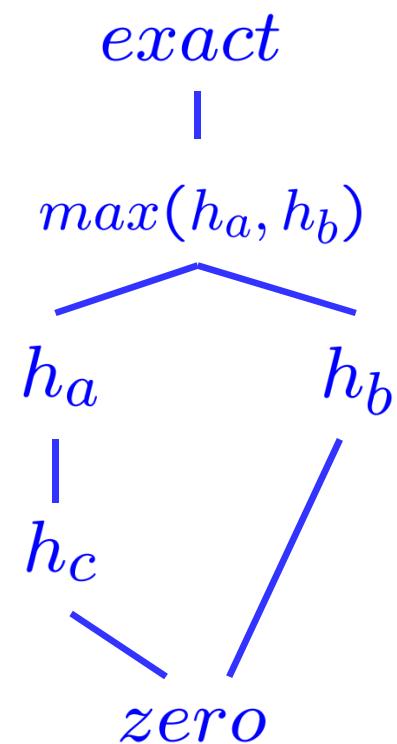
- Heuristics form a semi-lattice:

- Max of admissible heuristics is admissible

$$h(n) = \max(h_a(n), h_b(n))$$

- Trivial heuristics

- Bottom of lattice is the zero heuristic (what does this give us?)
- Top of lattice is the exact heuristic



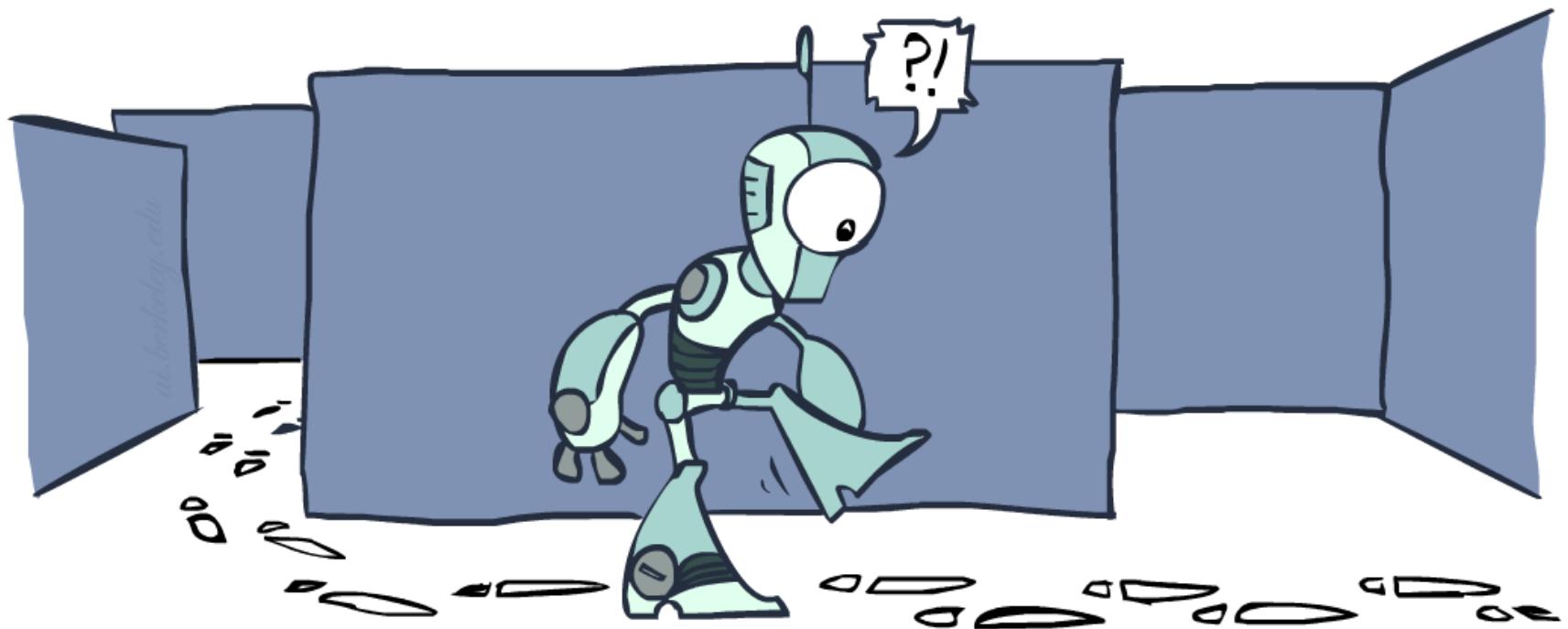
یک تکنیک دیگه که برای ایجاد Heuristics فانکشن داریم اینه که یه جور ایی Heuristics فانکشن های مختلف رو با هم مقایسه بکنیم یا ترکیب بکنیم:

یک مسئله ای هست به نام Dominance بودن یک Heuristics نسبت به یک دیگه

اگر یک Heuristics ما پیدا بکنیم که توی همه حالت ها مقدار Heuristics که داره میده توی این تساوی صدق بکنه ما میگیم این Dominance فانکشن Heuristics میکنه اون یکی رو

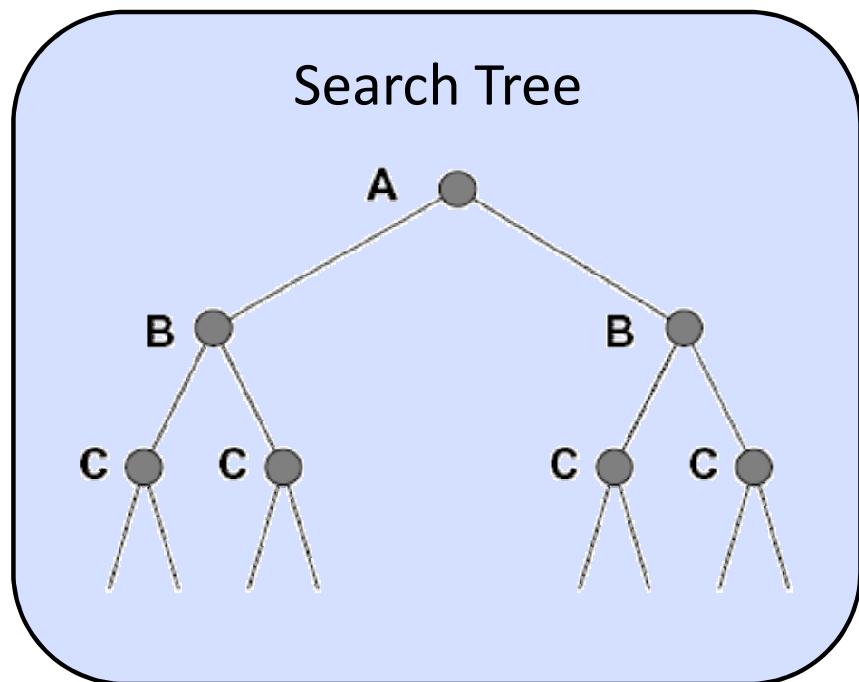
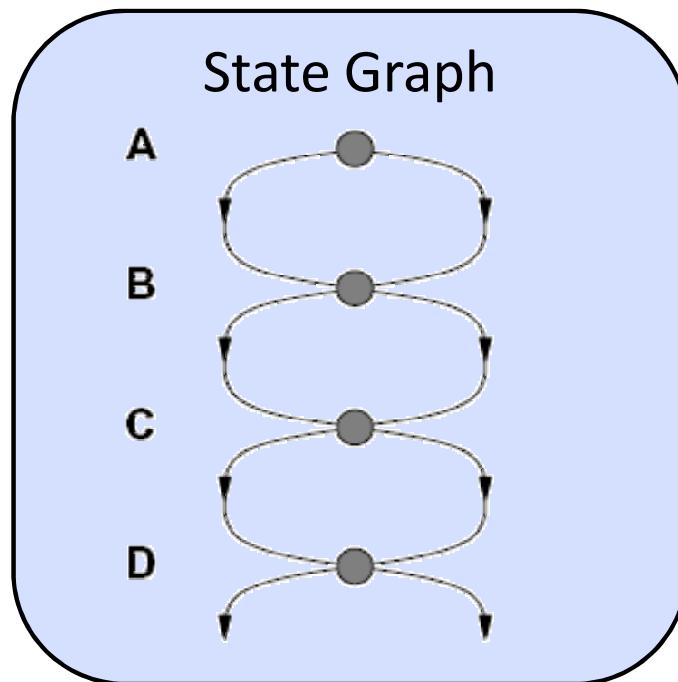
توی واقعیت وقتی که میایم Heuristics فانکشن های مختلف رو در نظر میگیریم نمیتونیم ما بگیم این Heuristics فانکشن از اون Heuristics فانکشن بهتره توی همه حالت ها بنابراین حالت هایی پیش میاد که توی 60 یا 70 درصد این Heuristics از اون یکی بهتره توی این حالت ها یک پیشنهادی وجود داره اینکه ما بیایم بین این Heuristics ها ماقرزیم بگیریم و اون مقدار ماقرزیم رو انتخاب بکنیم

Graph Search



Tree Search: Extra Work!

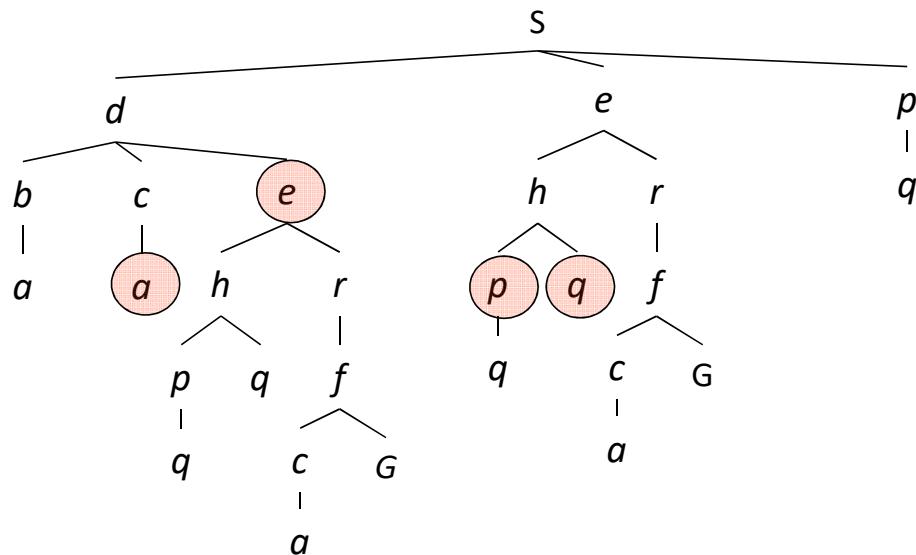
- Failure to detect repeated states can cause exponentially more work.



بعضی از مسائل وجود دارن که شرایط به نحوی است که ما با گراف سر و کار داریم
اگر اون تکنیک های قبلی رو روی این حالت اجرا بکنیم شکست می خوره

Graph Search

- In BFS, for example, we shouldn't bother expanding the circled nodes (why?)



:BFS برای مثلا

اینجا یکسری نودها هستن که این نودها رو تکراری داره می بینه مثل p , q و این تکراری ها هزینه سرچ رو افزایش میده چی کار باید کرد توی این حالت؟

Graph Search

- Idea: never **expand** a state twice
- How to implement:
 - Tree search + set of expanded states (“closed set”)
 - Expand the search tree node-by-node, but...
 - Before expanding a node, check to make sure its state has never been expanded before
 - If not new, skip it, if new add to closed set
- Important: **store the closed set as a set**, not a list
- Can graph search wreck completeness? Why/why not?
- How about optimality?

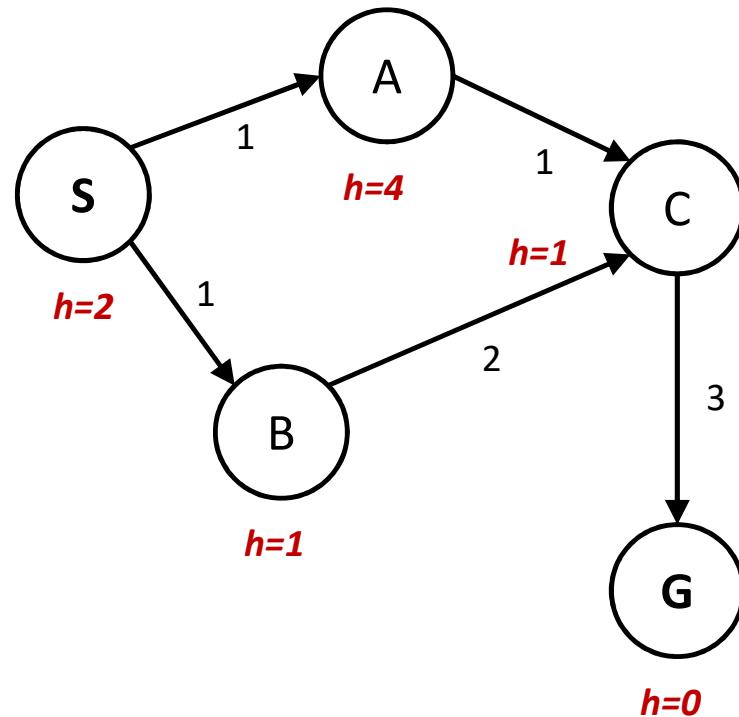
اولین پیشنهاد:

ما ببایم نودهایی که یکبار دیدیم یه کاری بکنیم دیگه نبینیم شون برای همین ما باید اونارو توی یک لیستی نگه داریم به نام **closed set** که ما بدونیم این نودها رو قبلا **expand** کردیم مموری اینجا خیلی افزایش پیدا میکنه --> همشون رو باید یکجا داشته باشیم و توی همشون هم سرچ بکنیم ینی نودی که الان بهش رسیدیم جز نودهایی هست که توی **closed set** ما هست اگر توی **closed set** ما هست دیگه **expand** نکنیم و اگر نیست اون نود رو **expand** بکنیم

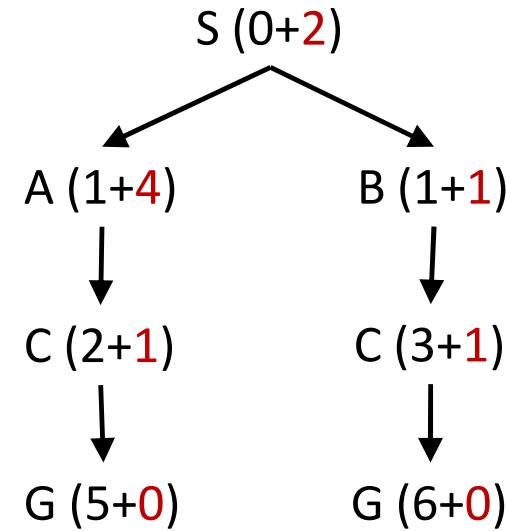
توی این حالت گراف سرچ ما **completeness** میشه ایا؟؟؟؟

A* Graph Search Gone Wrong?

State space graph



Search tree



برای A^* توی این حالت:

توی اینجا چون نود B رو یکبار دیده ینی از بین B ، A ما B رو انتخاب کردیم و این رو $expand$ کردیم پس این نود توی لیست $closed$ الان قرار میگیره بعد C و بعد G

حتی اگر بخواهد از نود A هم ادامه پیدا بکنه و بره به نود C با این حالت روبه رو میشه که ما C رو یکبار $expand$ کردیم و این C الان توی لیست $closed$ هست پس از این مسیر نمی تونیم بریم و اینجا الگوریتم ناکارامد میشه و نمی تونه کار بکنه

چجوری میشه الان اینو حلش کرد؟

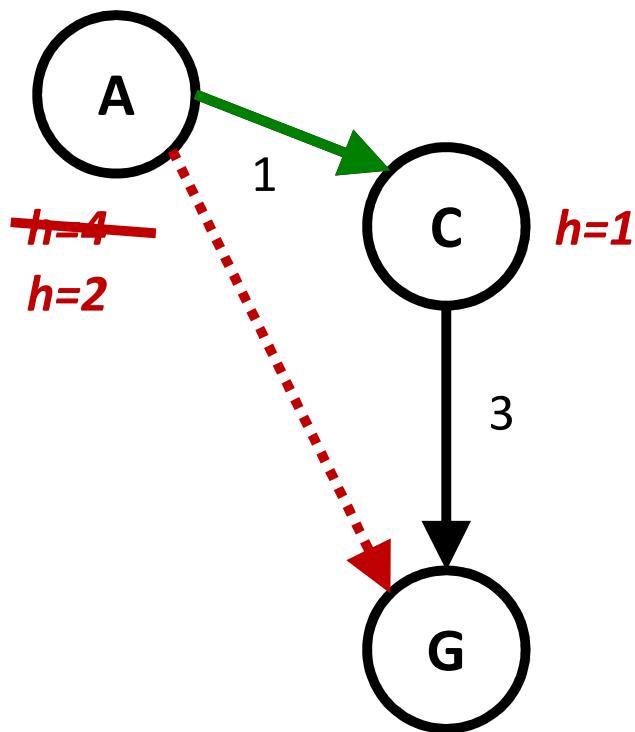
راهکار اول:

ما هزینه ها هم نگه داریم

راهکار دوم:

ما ببایم این A^* رو یه کاری بکنیم که اگر قرار شد به یک نود تکراری برسه حتما اون نود تکراری هزینش بیشتر باشه ینی اون C بهتره اول بره توی لیست $closed$ و بعد برسیم به اون C بدتره پس می خوایم یه بلای سر این $heuristic$ فانکشن بیاریم که این $heuristic$ فانکشن ما رو نبره به این شرایط و ما رو ببره به یک سمتی که اول C خوب رو ببینیم و بعد C بدتر رو ببینیم

Consistency of Heuristics



- Main idea: estimated heuristic costs \leq actual costs
 - Admissibility: heuristic cost \leq actual cost to goal
$$h(A) \leq \text{actual cost from } A \text{ to } G$$
 - Consistency: heuristic “arc” cost \leq actual cost for each arc
$$h(A) - h(C) \leq \text{cost}(A \text{ to } C)$$
- Consequences of consistency:
 - The f value along a path never decreases
$$h(A) \leq \text{cost}(A \text{ to } C) + h(C)$$
 - A* graph search is optimal

این اتفاق رو بهش میگن سازگار بودن heuristic فانکشن و یک شرطی رو باید طی بکنه:
اون بحث Admissibility بودنش که هست

علاوه بر این باید یک شرط دیگه هم داشته باشیم: که ایدش از نامساوی مثلثی گرفته میشه

اینجا اگر این Heuristics فانکشن مقدارش به جای اینکه 4 باشه 2 باشه --> هزینه ای که داریم

از مسیر 1 می ریم با مسیر 2 می ریم Heuristics ما رو به اشتباه می ندازه چون داره حداقل

هزینه که ما باید طی بکنیم که از یک مسیر بریم رو خوب نشون نمیده اما برای اون 4 همزمان یک

مسیر دیگه ای هم وجود داره که اون مسیر با هزینه کمتر ما رو می تونه به استیت هدف برسونه و

بحث ناسازگاری پیش میاد --> توی مثلث ما یک ایده ای داشتیم و می گفتیم همیشه فاصله یک نود

تا راس دیگه همیشه باید از مجموعه دوتای دیگه کمتر باشه الان هم ما باید همچین اتفاقی رو اینجا

حاکم بکنیم --> پس Heuristics مقدار 4 ما رو به اشتباه میندازه و Heuristics مقدار 2 درسته

اینجا الان --> ما باید توی گراف هرچی داریم حرکت میکنیم در جهت استیت هدف یک هزینه ای

پرداخت میکنیم--> Heuristics هر دوتا نود مجاور باید اختلافشون از Heuristics فانکشن

کمتر باشه

Heuristics میشه فاصله این نود تا نود هدف و یک دامنه هم داشتیم برای مقادیر و این دامنه مقادیر

می تونست از صفر باشه تا مقدار خود h^* حالا می ریم به نود بعدی و وقتی به نود بعدی رفتم

که توی نود بعدی می بینیم باید این هزینه ای که ما الان طی کردیم و به اون رسیدیم

توش لاحظ کرده باشیم چون Heuristics ما همش داشت به هدف نگاه میکرد و به هزینه گذشته

نگاه نمی کرد و ما داریم با این بحث سازگاری یک شرایطی از گذشته هم توی Heuristics ها

وارد میکنیم تا اون کاستی که رسیده تا اونجا هم یک شرطی حاکم بشه توی این Heuristics ها

انگار داریم محدودتر میکنیم این فضای Heuristics رو

با این روش همیشه نودهای تکراری رو اول از همه می بینیم ینی اون خوبش رو اول می بینیم

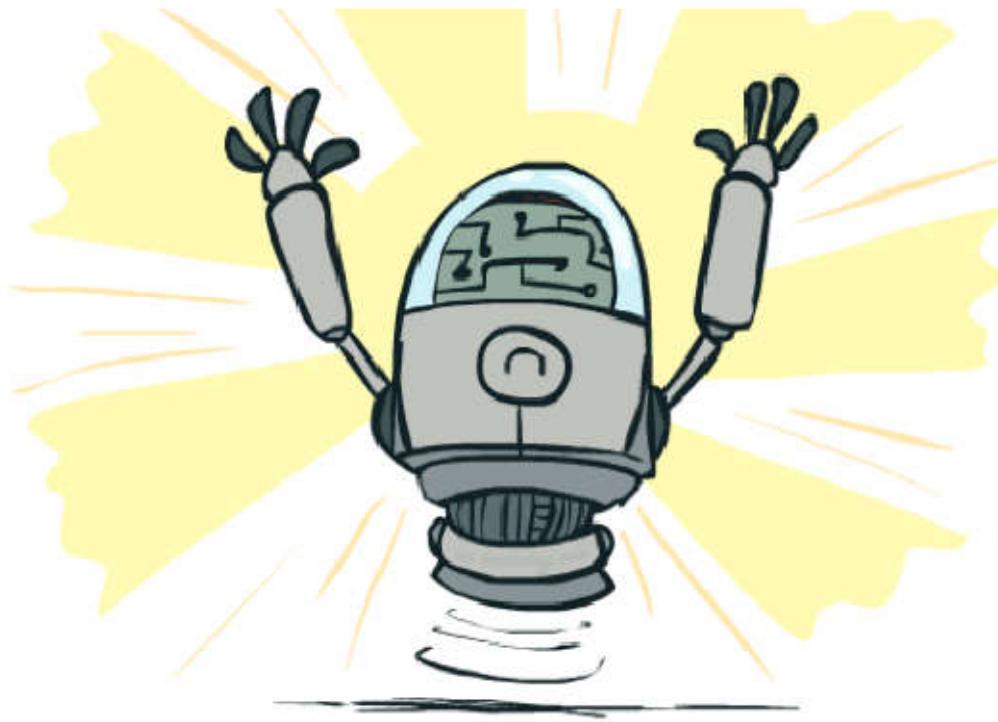
پس ما هر Heuristics که داریم به عنوان نود پدر باید مقدار Heuristics اش از نود فرزندash

کمتر باشه چرا؟ چون حداقل یک هزینه ای داریم: 1-هزینه خود اونو داریم 2- هزینه رفتن پدر تا

پسر 3- یک Heuristics اون مقدار داره

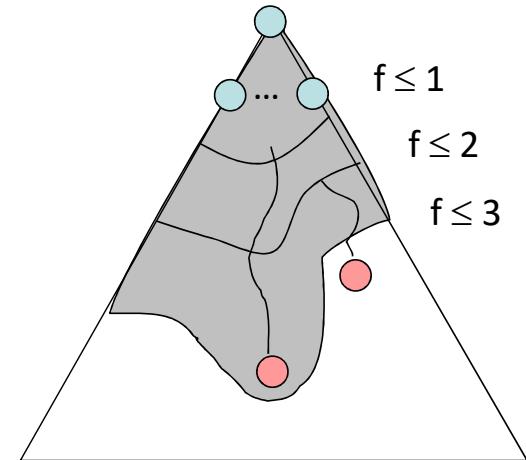
این شرایط اگر برقرار باشه A^* ما بھینه میشه ینی نودهای C رو اول از همه می بینیم

Optimality of A* Graph Search



Optimality of A* Graph Search

- Sketch: consider what A* does with a consistent heuristic:
 - Fact 1: In tree search, A* expands nodes in increasing total f value (f-contours)
 - Fact 2: For every state s, nodes that reach s optimally are expanded before nodes that reach s suboptimally
 - Result: A* graph search is optimal

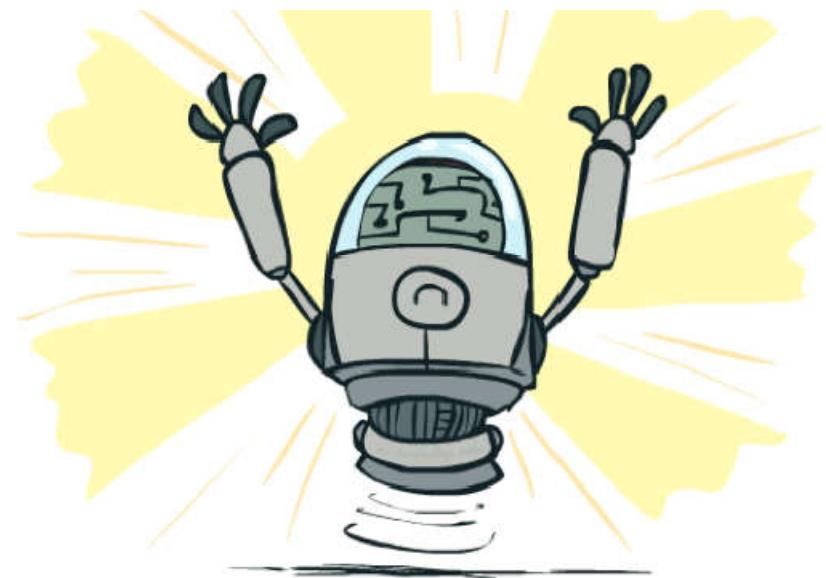


:A*

چون اینجا با گراف سر و کار داریم یکسری لیست closed داریم و نودهایی رو باید اول از همه ببینیم که توی مسیر بهینه هستن و اون نودها باید اول از همه expand بشن

Optimality

- Tree search:
 - A* is optimal if heuristic is admissible
 - UCS is a special case ($h = 0$)
- Graph search:
 - A* optimal if heuristic is consistent
 - UCS optimal ($h = 0$ is consistent)
- Consistency implies admissibility
- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems



توی گرید سرچ اگر heuristic فانکشن رو صفر بدیم میشه UCS و این UCS فاصله زمانی تا رسیدن به نود هدف خیلی زمان بر بود

؟؟

ایا هر heuristic فانکشن admissibility سازگار است یا نه؟

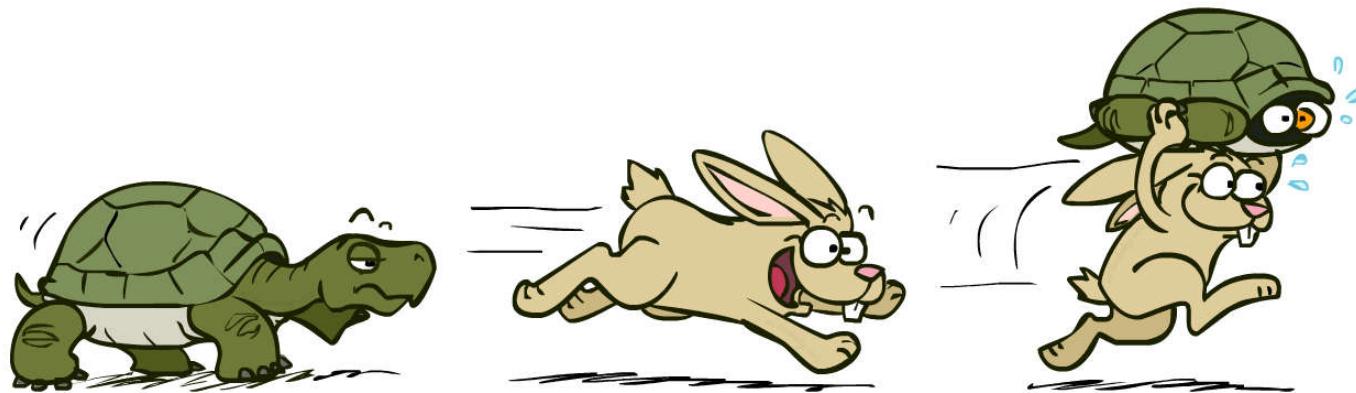
اگر یک heuristic فانکشن داشته باشیم که سازگار باشه اون admissibility بودن رو برآش ایجاد میکنه چون ما یک نود هدف داریم و در طرف اون نود هدف که داریم حرکت میکنیم مقادیر کاهش پیدا میکنه براساس هزینه هامون ولی بر عکش اینطوری نیست یعنی اون admissibility بودن یک فضای بزرگتری است و لزوما توی نامساوی مثلث نمی تونه صدق بکنه پس اگر ما یک heuristic فانکشن سازگار پیاده سازی بکنیم لزوما admissibility هم هست ولی بر عکش رو نداریم

A*: Summary



A*: Summary

- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible / consistent heuristics
- Heuristic design is key: often use relaxed problems



A^* توی ابتدای کار با سرعت کم مثل UCS داره عمل می کنه ولی به محض اینکه یک دید خوبی راجع به استیت هدف گرفت با سرعت بیشتری حالت گردید سرچ عمل میکنه

Tree Search Pseudo-Code

```
function TREE-SEARCH(problem, fringe) return a solution, or failure
  fringe  $\leftarrow$  INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node  $\leftarrow$  REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    for child-node in EXPAND(STATE[node], problem) do
      fringe  $\leftarrow$  INSERT(child-node, fringe)
    end
  end
```

Graph Search Pseudo-Code

```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    if STATE[node] is not in closed then
      add STATE[node] to closed
      for child-node in EXPAND(STATE[node], problem) do
        fringe ← INSERT(child-node, fringe)
    end
  end
```
