هوش مصنوعی پیشرفته

Advanced Artificial Intelligence

سهیلا اشکذری طوسی

فصل چهارم

مروری بر الگوریتم های جستجوی آگاهانه

جستجوی ناآگاهانه در بسیاری از موارد ناکارا هستند. در این فصل روش هایی را بررسی می کنیم که از دانش مسئله برای یافتن یک راه مؤثرتر استفاده می کنند.

جستجوی آگاهانه/ تعریف کلی

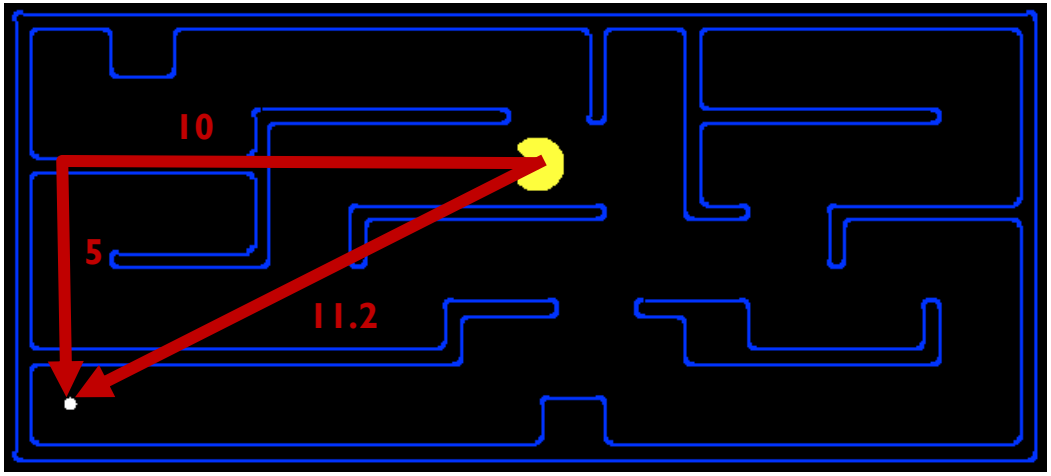
- روش جستجوی آگاهانه با استفاده از دانش خاص مسئله که فراتر از تعریف مسئله است (که در جستجوی ناآگاهانه استفاده می شد)، می تواند راه حل های بهتری پیدا نماید.
- در این روش ها تابعی به نام ارزیاب (Evaluation Function) تعریف می شود که فاصله تا هدف را اندازه می گیرد.
- در هر مرحله گره ای که کمترین مقدار ارزیابی را دارد، برای گسترش انتخاب می شود.
- نام کلی این روشها، **جستجوی اول بهترین** می باشد.
- در این روشها، لبه (fringe) یک صف است.

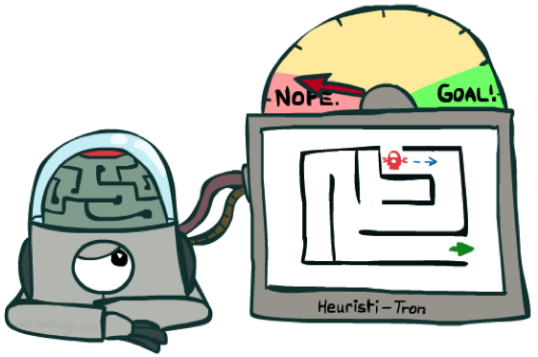
الگوریتم های جستجوی آگاهانه

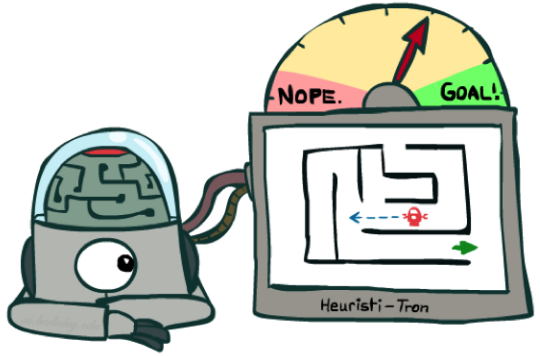
- الگوریتم جستجوی حریصانه (Greedy Search)
- الگوریتم A*
- الگوریتم IDA*
- الگوریتم RBFS
- الگوریتم MA*

جستجوی آگاهانه – تابع هیوریستیک

- 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

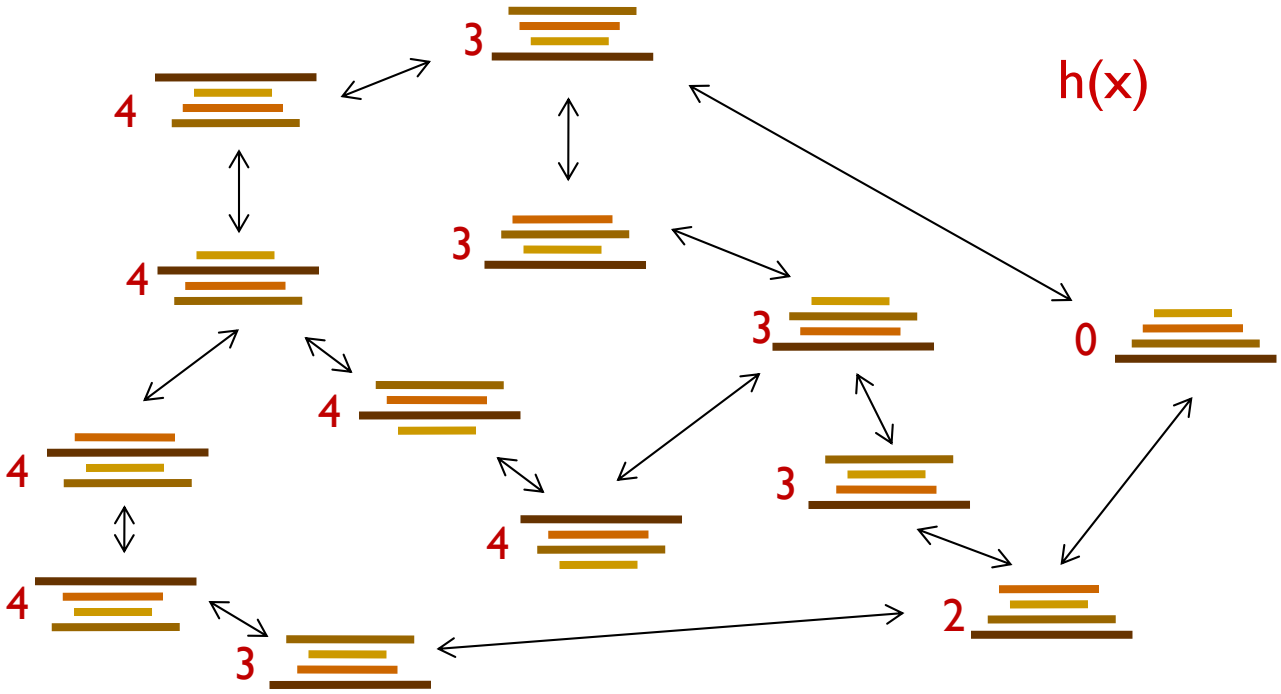






جستجوی آگاهانه – تابع هیوریستیک

Heuristic: the number of the largest pancake that is still out of place



جستجوی آگاهانه / اول بهترین حریصانه

- سعی می کند نزدیک ترین گره به هدف را گسترش دهد زیرا احتمال دارد سریع ترین راه حل باشد.
- از تابع هیوریستیک (heuristic) h(n) برای ارزیابی گره ها استفاده می کند.
- تابع h(n) هزینه کم هزینه ترین مسیر تا هدف را تخمین می زند.
- اگر گره n، گره هدف باشد، h(n)=0 خواهد بود.

Greedy Search

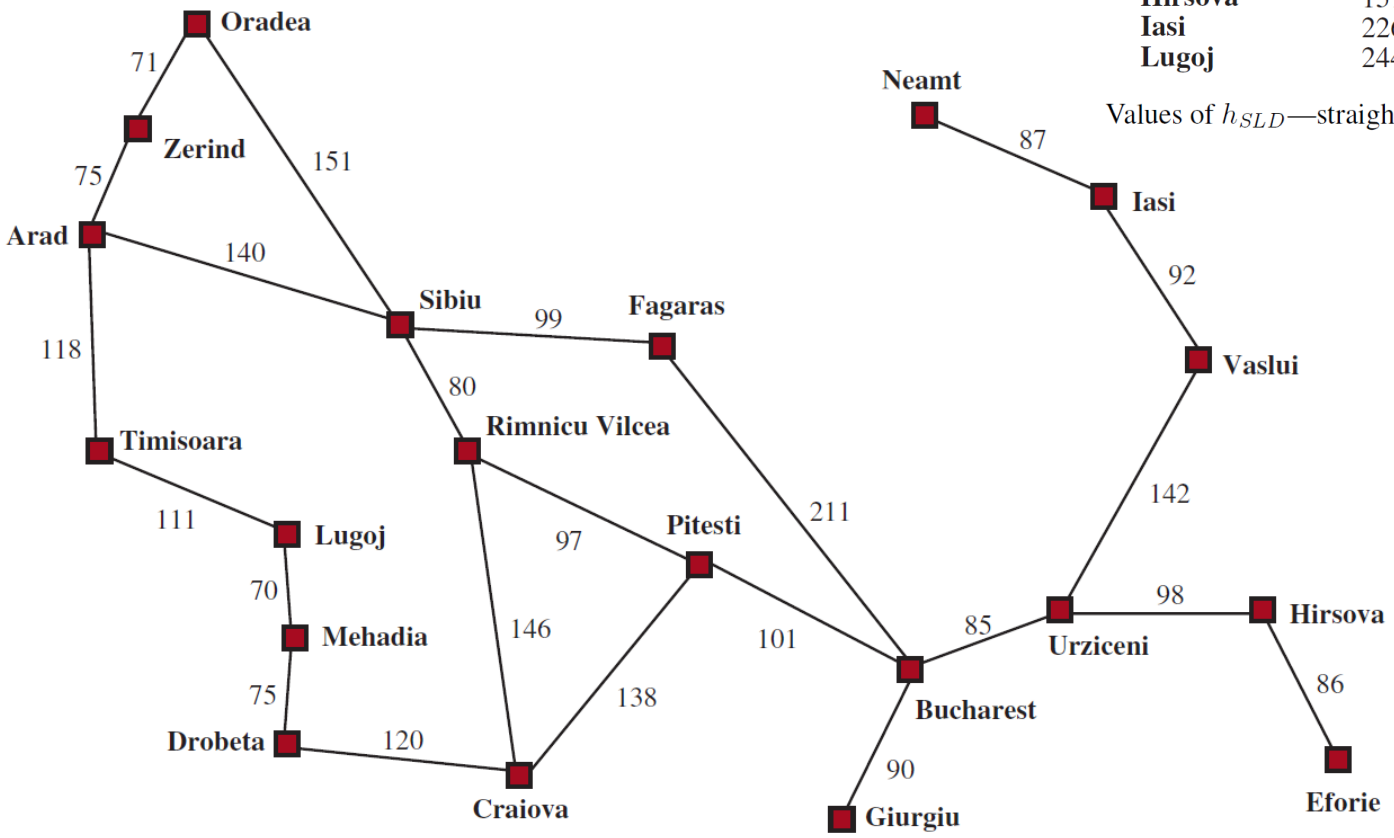
جستجوی آگاهانه / اول بهترین حریصانه/مثال

- مثال: مسئله مسیریابی در نقشه رومانی
- تابع هیوریستیک : تخمین فاصله مستقیم هر شهر تا بخارست

جستجوی آگاهانه / اول بهترین حریصانه/مثال

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

Values of h_{SLD}—straight-line distances to Bucharest.



جستجوی آگاهانه / اول بهترین حریصانه/مثال

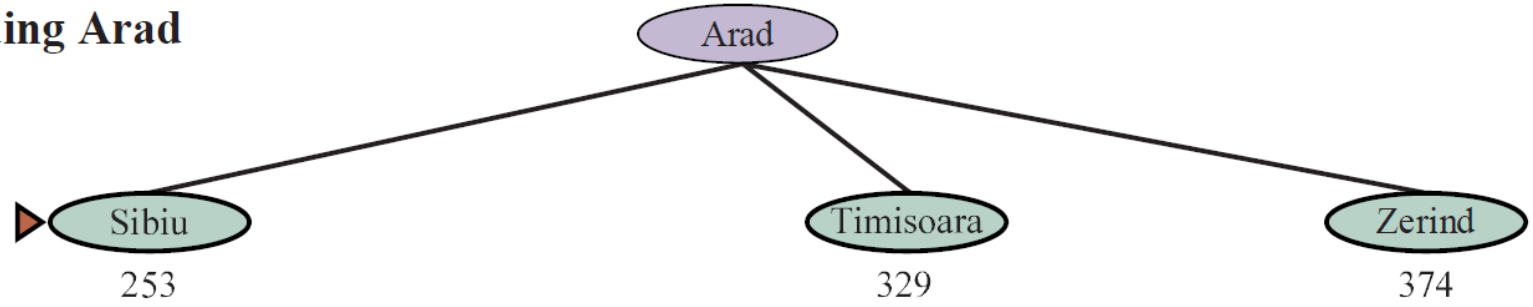
(a) The initial state

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

Values of h_{SLD}—straight-line distances to Bucharest.

جستجوی آگاهانه / اول بهترین حریصانه/مثال

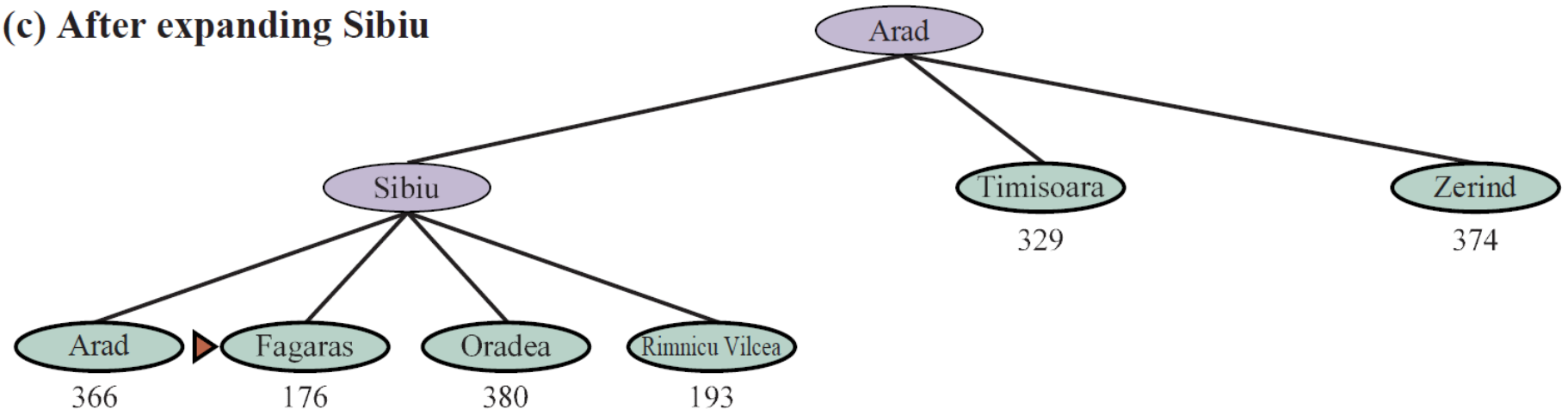
(b) After expanding Arad



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

Values of h_{SLD}—straight-line distances to Bucharest.

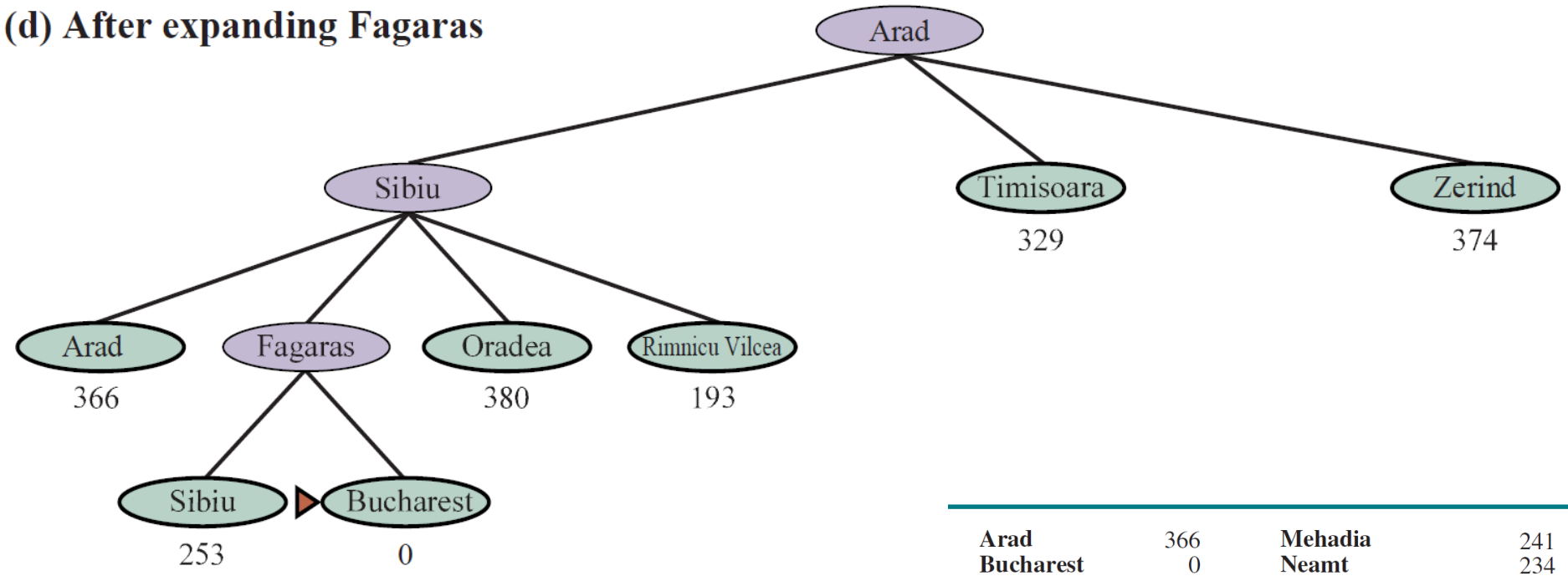
جستجوی آگاهانه / اول بهترین حریصانه/مثال



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

Values of h_{SLD}—straight-line distances to Bucharest.

جستجوی آگاهانه / اول بهترین حریصانه/مثال



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

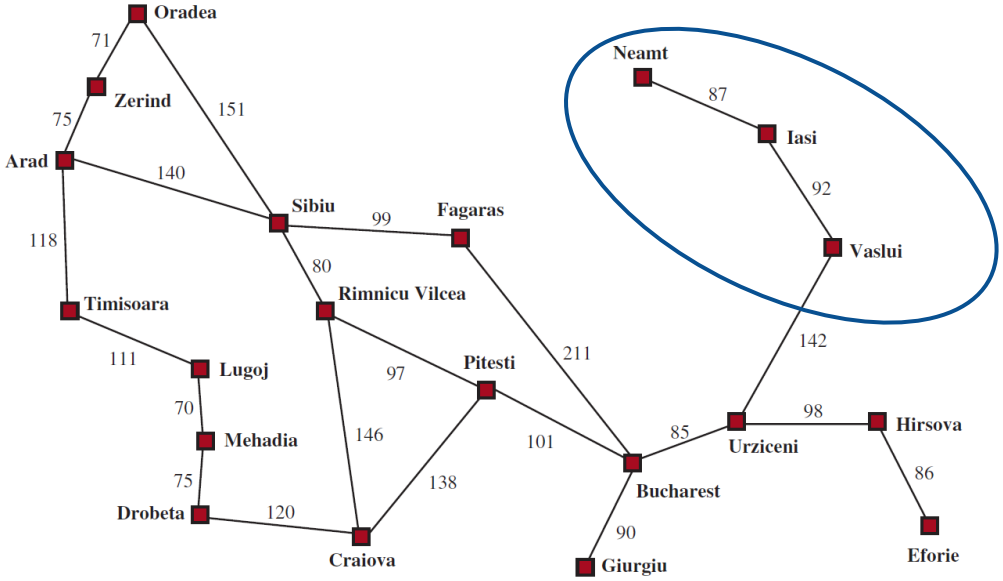
Values of h_{SLD}—straight-line distances to Bucharest.

جستجوی آگاهانه / اول بهترین حریصانه

- در این روش هزینه جستجو حداقل است.
- اما بهینه نیست: مسیری که انتخاب می شود ۳۲ کیلومتر طولانی تر از مسیری است که از Rimnicu Vileca و Pitesti می گذرد.
- به همین علت به این روش حریصانه می گویند. زیرا:
  - در هر **گام**، سعی می کند تا حد امکان به هدف نزدیک شود.
- این روش بیشتر مشابه جستجوی اول عمق می باشد:
  - ترجیح می دهد یک مسیر را تا هدف دنبال کند.
  - اگر به بن بست رسید، به عقب بازمی گردد.

جستجوی آگاهانه / اول بهترین حریصانه

- این روش کامل نیست:
  - امکان وجود گره های تکراری و مسیر بی انتها
  - کمینه کردن h(n) می تواند منجر به شروع های نادرست شود.
  - مثلا در نقشه رومانی، اگر بخواهیم از Iasi به فاگارس برویم، تابع هیوریستیک در Iasi پیشنهاد می دهد که به Neamt برویم در حالیکه بن بست است!

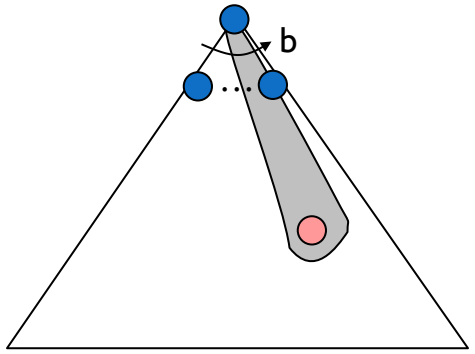


- پیچیدگی زمانی و فضایی:

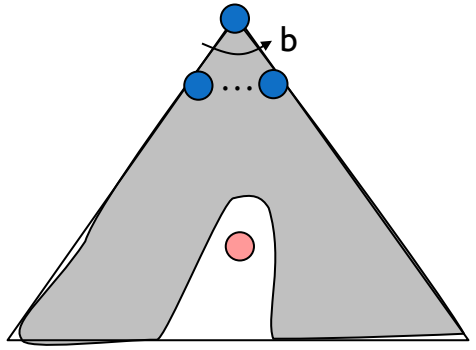
$$O(b^m)$$

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



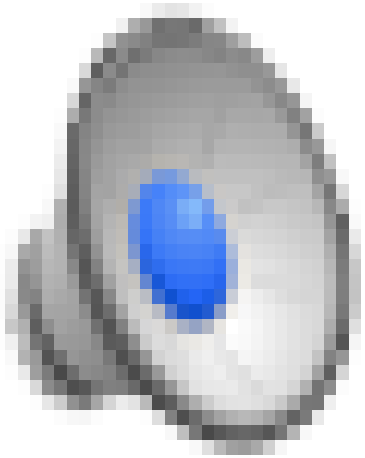
- A common case:
  - Best-first takes you straight to the (wrong) goal



- Worst-case: like a badly-guided DFS

[Demo: contours greedy empty (L3D1)]
[Demo: contours greedy pacman small maze (L3D4)]

Video of Demo Contours Greedy (Pacman Small Maze)



جستجوی آگاهانه / جستجوی A*

- شناخته شده ترین روش جستجوی اول بهترین است.
- در این روش گره ها با ترکیب
  - هزینه رسیدن به گره (g(n)) و
  - هزینه رسیدن از آن گره به هدف (h(n))

  ارزیابی می شوند.

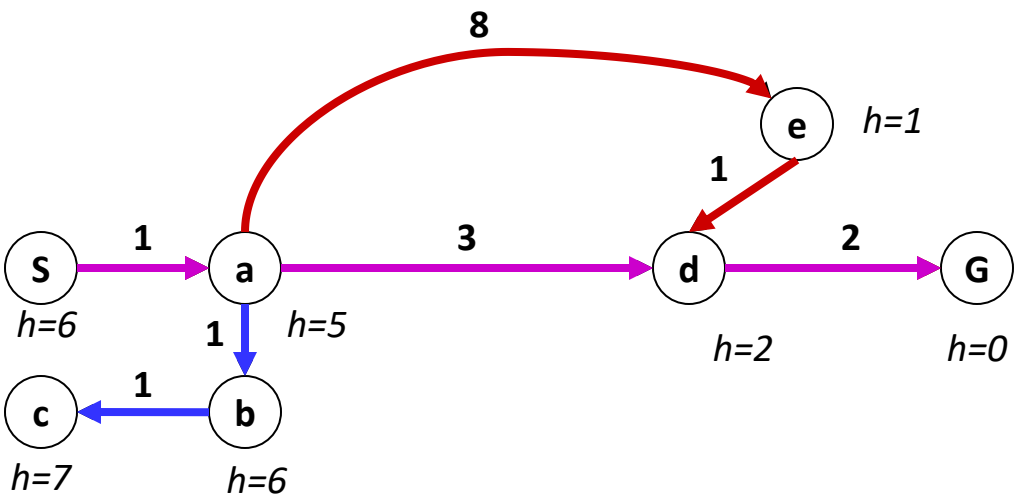
$$f(n) = g(n)+h(n)$$

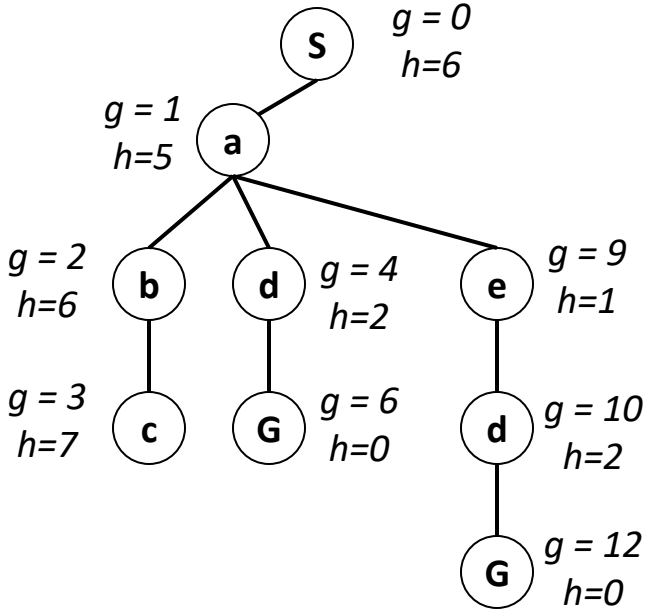
f(n) : هزینه برآورد ارزان ترین راه حل از طریق n

جستجوی آگاهانه / جستجوی A*

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)





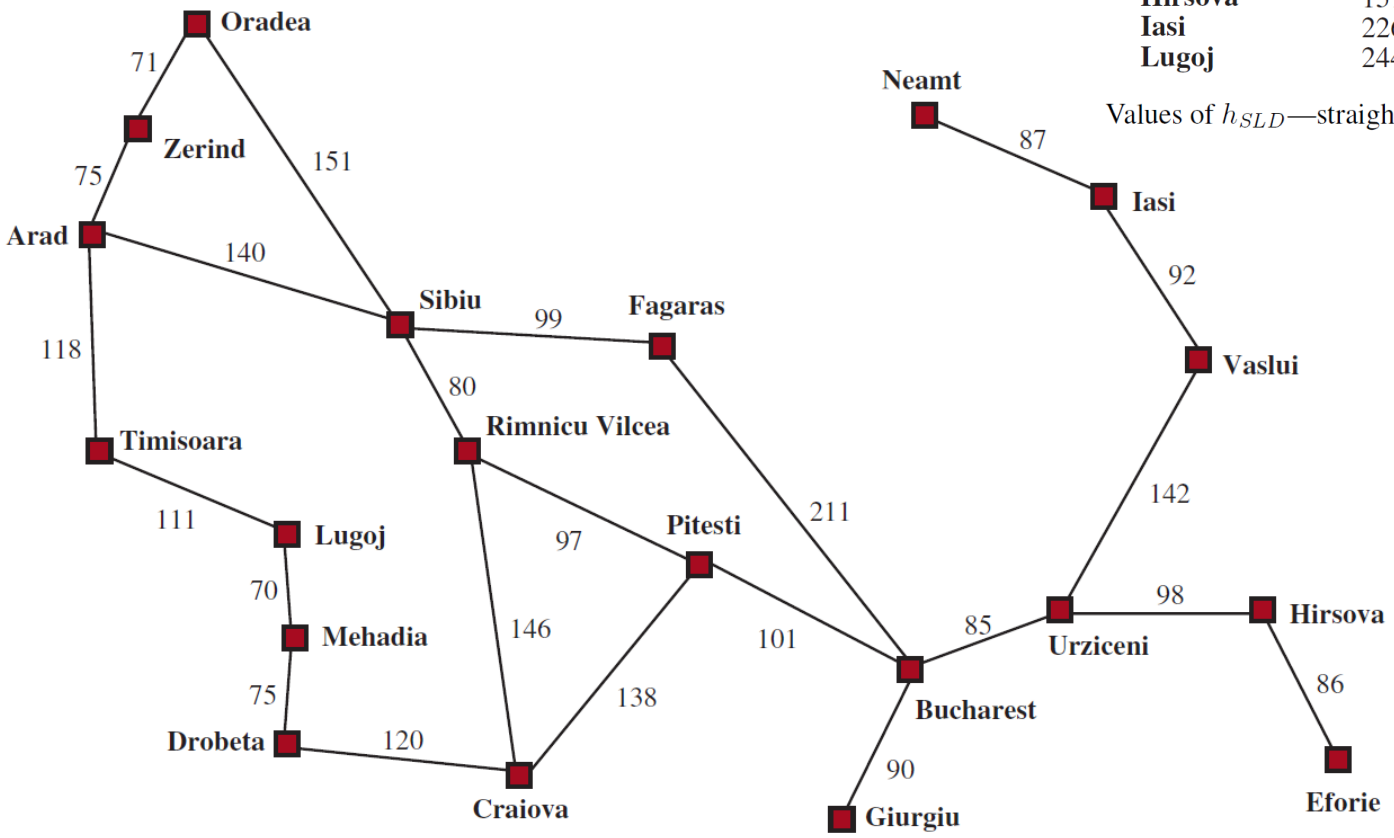
- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

جستجوی آگاهانه / جستجوی A*

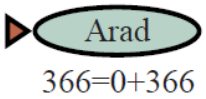
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Values of h_{SLD}—straight-line distances to Bucharest.

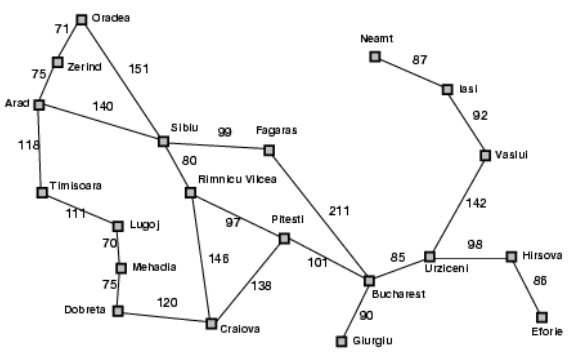


جستجوی آگاهانه / جستجوی A*

(a) The initial state

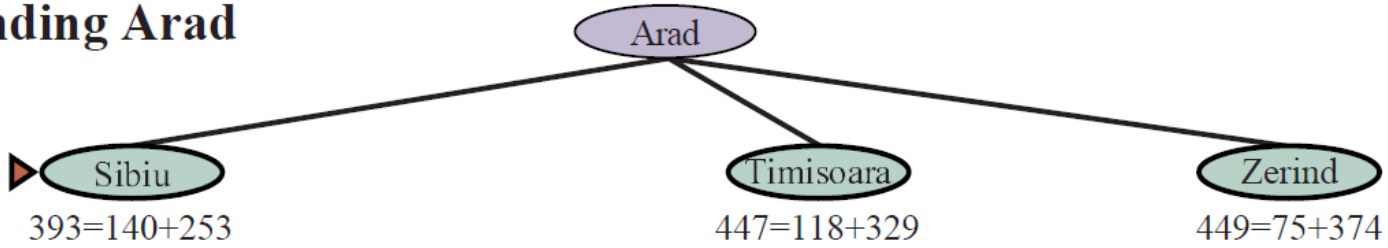


- Find Bucharest starting at Arad
  - f(Arad) = c(??,Arad)+h(Arad)=0+366=366

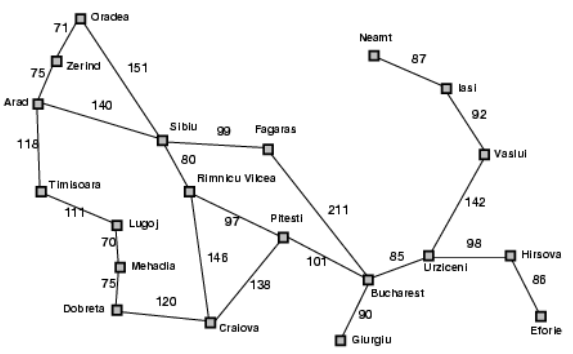


جستجوی آگاهانه / جستجوی A*

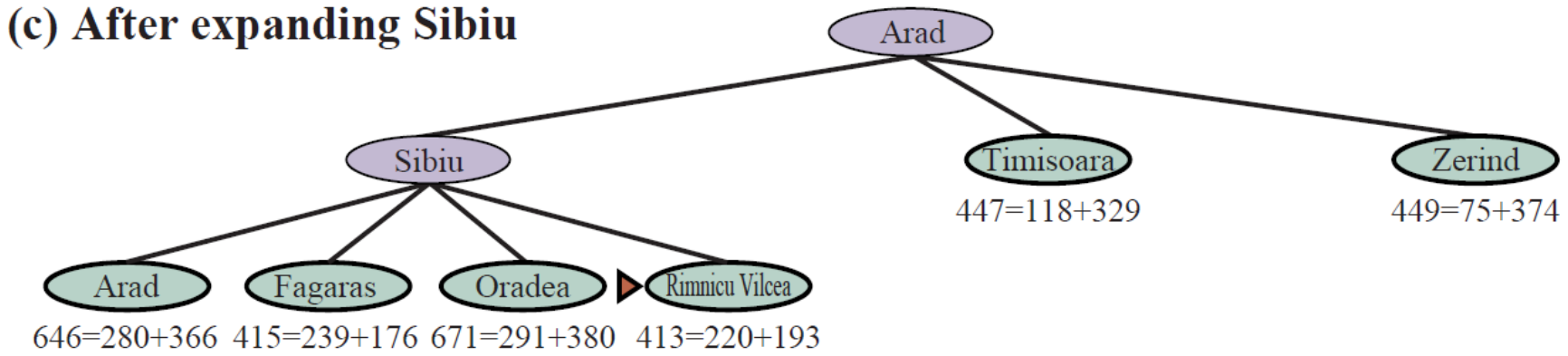
(b) After expanding Arad



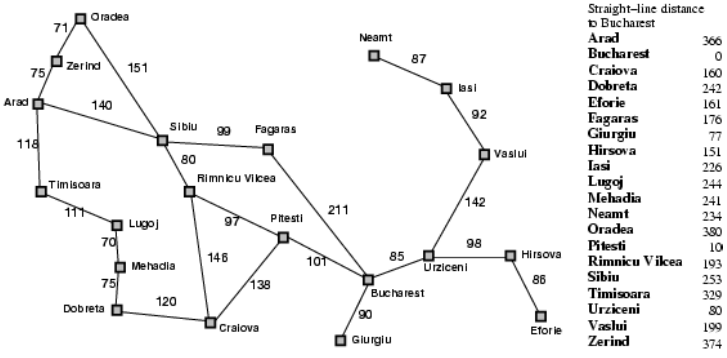
- Expand Arrad and determine *f(n)* for each node
  - f(Sibiu)=c(Arad,Sibiu)+h(Sibiu)=140+253=393
  - f(Timisoara)=c(Arad,Timisoara)+h(Timisoara)=118+329=447
  - f(Zerind)=c(Arad,Zerind)+h(Zerind)=75+374=449
- Best choice is Sibiu



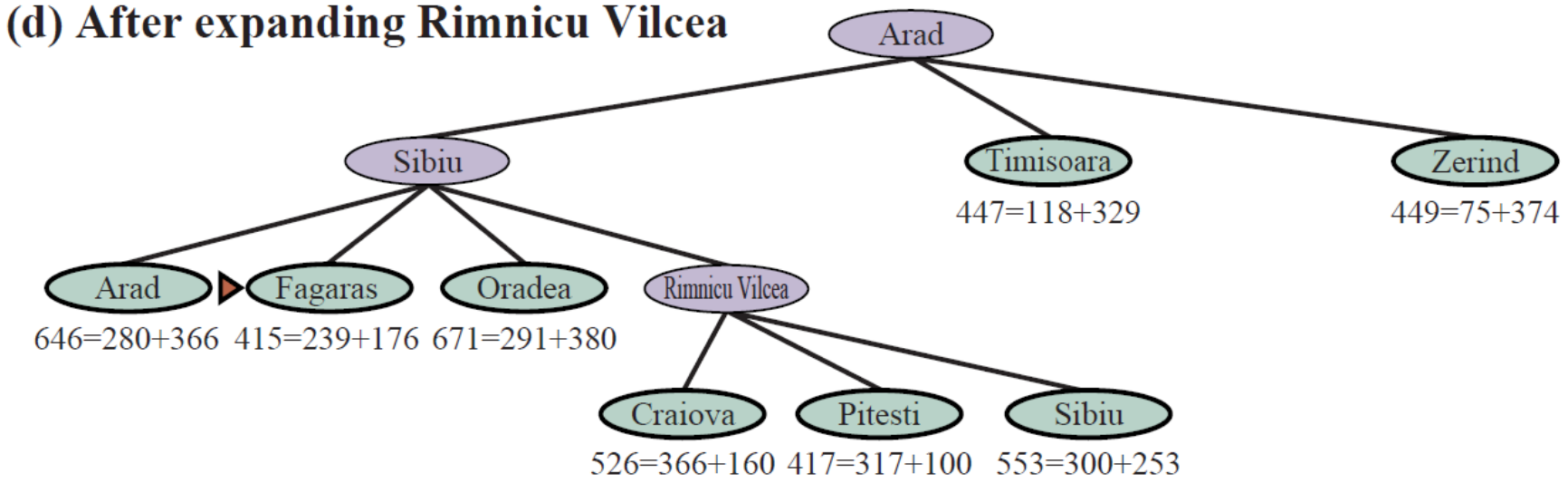
جستجوی آگاهانه / جستجوی A*



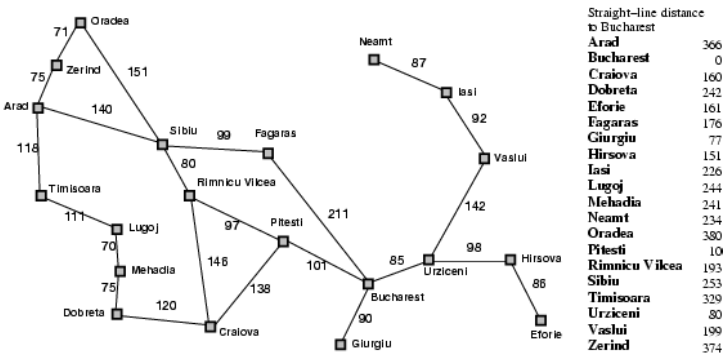
- Expand Sibiu and determine *f(n)* for each node
  - f(Arad)=c(Sibiu,Arad)+h(Arad)=280+366=646
  - f(Fagaras)=c(Sibiu,Fagaras)+h(Fagaras)=239+176=415
  - f(Oradea)=c(Sibiu,Oradea)+h(Oradea)=291+380=671
  - f(Rimnicu Vilcea)=c(Sibiu,Rimnicu Vilcea)+h(Rimnicu Vilcea)=220+192=413
- Best choice is Rimnicu Vilcea



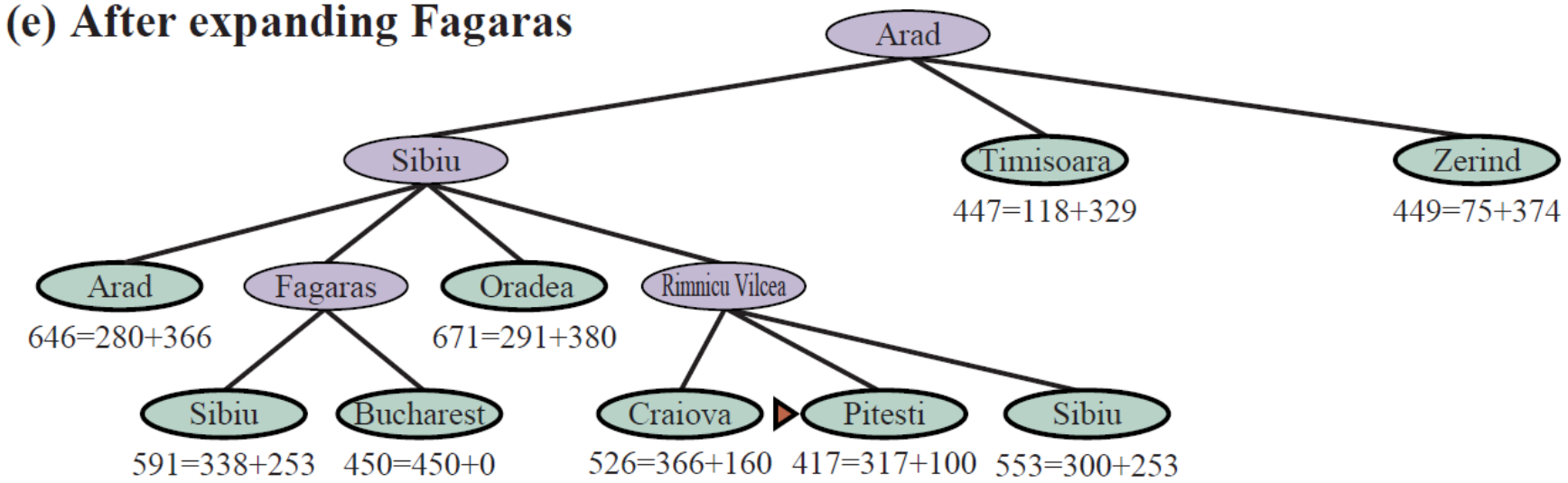
جستجوی آگاهانه / جستجوی A*



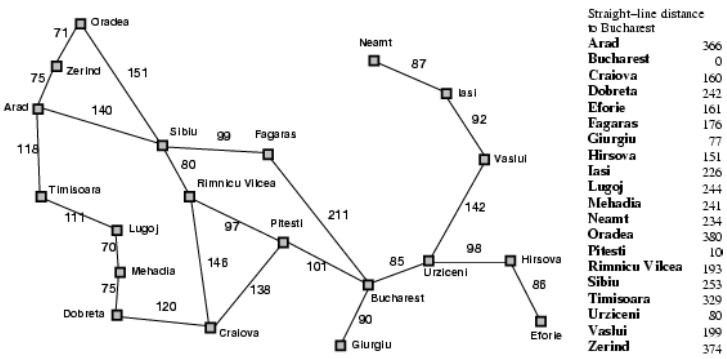
- Expand Rimnicu Vilcea and determine *f(n)* for each node
  - f(Craiova)=c(Rimnicu Vilcea, Craiova)+h(Craiova)=360+160=526
  - f(Pitesti)=c(Rimnicu Vilcea, Pitesti)+h(Pitesti)=317+100=417
  - f(Sibiu)=c(Rimnicu Vilcea,Sibiu)+h(Sibiu)=300+253=553
- Best choice is Fagaras



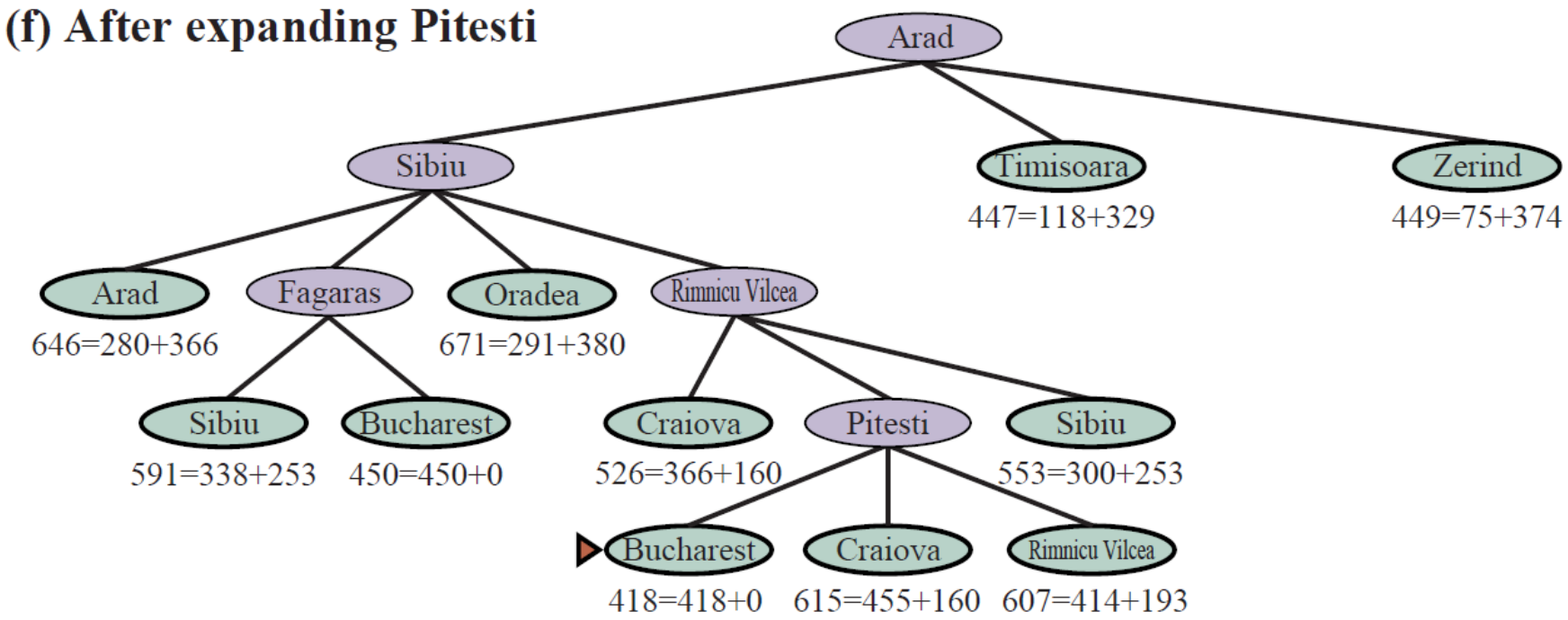
جستجوی آگاهانه / جستجوی A*



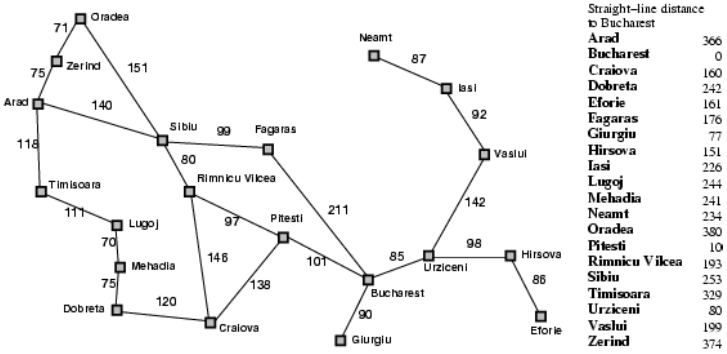
- Expand Fagaras and determine *f(n)* for each node
  - f(Sibiu)=c(Fagaras, Sibiu)+h(Sibiu)=338+253=591
  - *f(Bucharest)=c(Fagaras,Bucharest)+h(Bucharest)=450+0=450*
- Best choice is Pitesti !!!



جستجوی آگاهانه / جستجوی A*



- Expand Pitesti and determine *f(n)* for each node
  - *f(Bucharest)=c(Pitesti,Bucharest)+h(Bucharest)=418+0=418*
- Best choice is Bucharest !!!
  - Optimal solution?
  - Yes, only if *h(n)* is admissible



جستجوی آگاهانه / جستجوی A*

- جستجوی A* کامل است.
- بهینه است اگر تابع هیوریستیکی که استفاده می شود، قابل قبول (admissible) باشد.
- **تابع هیوریستیک قابل قبول** : هرگز هزینه رسیدن به هدف را بیش از آنچه که واقعا هست برآورد نکند.
- برای مثال قبل، استفاده از فاصله مستقیم بین دو شهر، تخمین مناسبی از فاصله واقعی بین آنها روی نقشه رومانی است، زیرا کوتاه ترین مسیر بین دو نقطه، یک خط مستقیم است و بنابراین هیوریستیک خط مستقیم نمی تواند فاصله را بیشتر از مقدار واقعی تخمین بزند.
- اگر هیورستیک استفاده شده قابل قبول باشد، آنگاه روش A* با جستجوی درختی بهینه است.

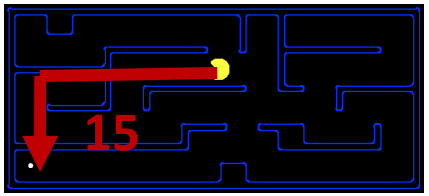
Admissible Heuristics

- 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:



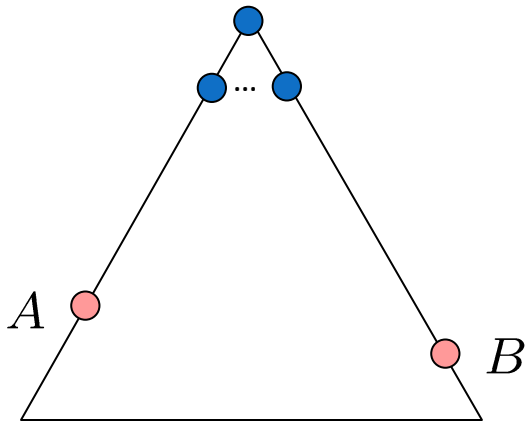
4

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

Optimality of A* Tree Search

Assume:

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



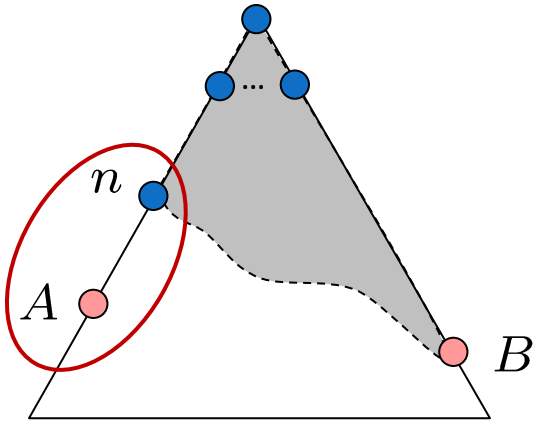
Claim:

- A will exit the fringe before B

Optimality of A* Tree Search

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)



$$f(n) = g(n) + h(n)$$ Definition of f-cost

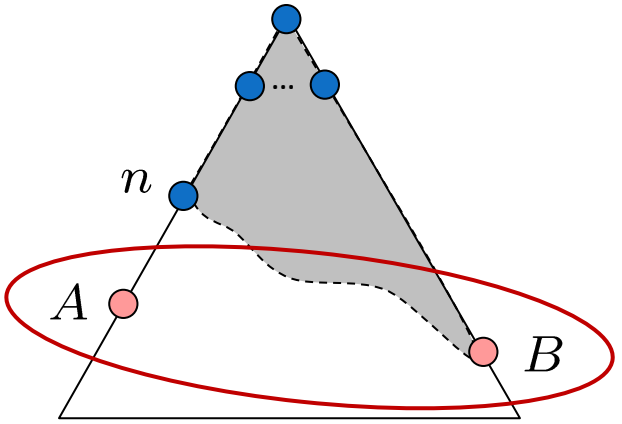
$$f(n) \leq g(A)$$ Admissibility of h

$$g(A) = f(A)$$ h = 0 at a goal

Optimality of A* Tree Search

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)



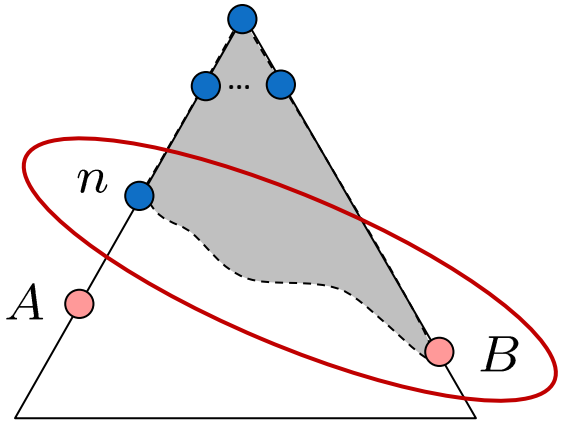
$g(A) < g(B)$ B is suboptimal

$f(A) < f(B)$ h = 0 at a goal

Optimality of A* Tree Search

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



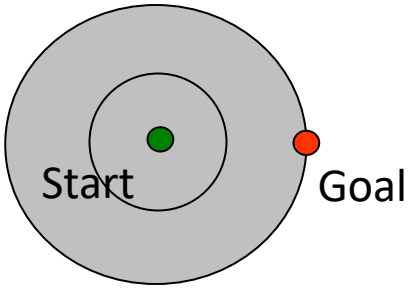
$$f(n) \leq f(A) < f(B)$$

جستجوی آگاهانه / جستجوی A*

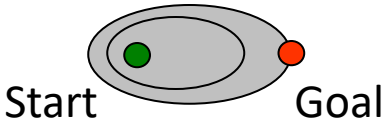
- کامل بودن : بله
- بهینه بودن : بله
- پیچیدگی زمانی : تعداد نودهایی که توسعه می یابد نسبت به طول مسیر، نمایی است.
- پیچیدگی مکانی : همه نودهای تولید شده در حافظه ذخیره می شوند.
- این روش بیشتر با مشکل حافظه رو به رو است تا زمان و اغلب قبل از آنکه وقت کم بیاورد، حافظه کم می آورد.

UCS vs A* Contours

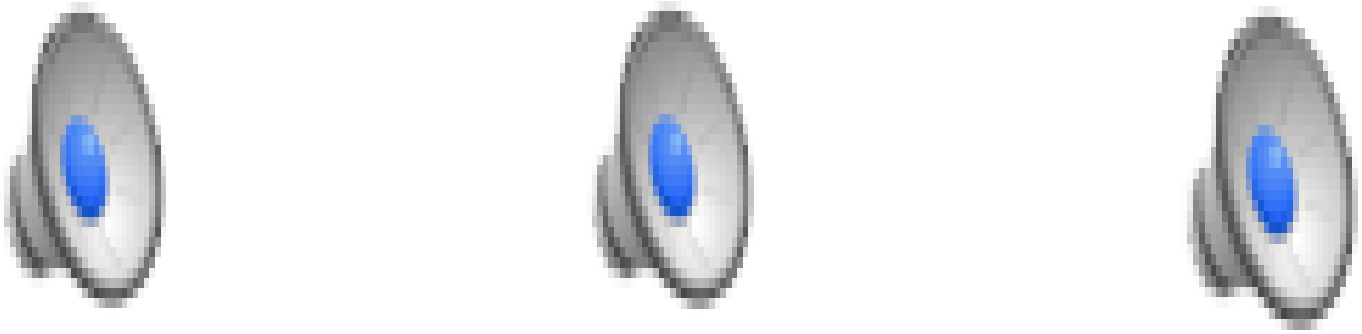
- Uniform-cost expands equally in all “directions”



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

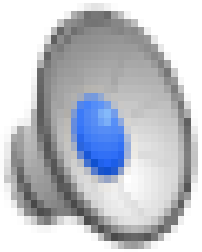


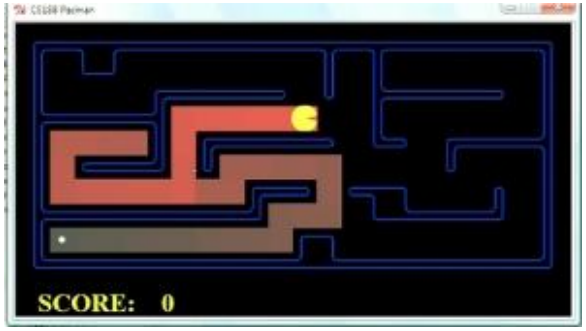
UCS , Greedy and A* Contours

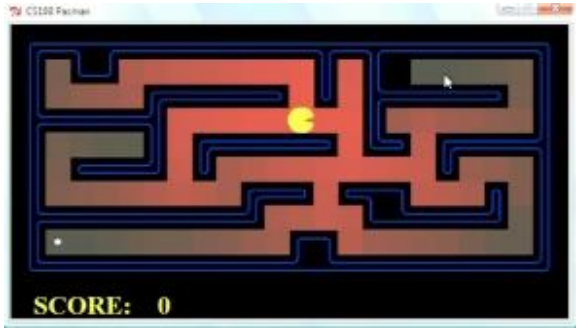


USC Greedy A*

Demo Contours (Pacman Small Maze) – A*







Greedy

Uniform Cost

A*

جستجوی آگاهانه – مثال معمای ۸

5		8
4	2	1
7	3	6

STATE(N)

1	2	3
4	5	6
7	8	

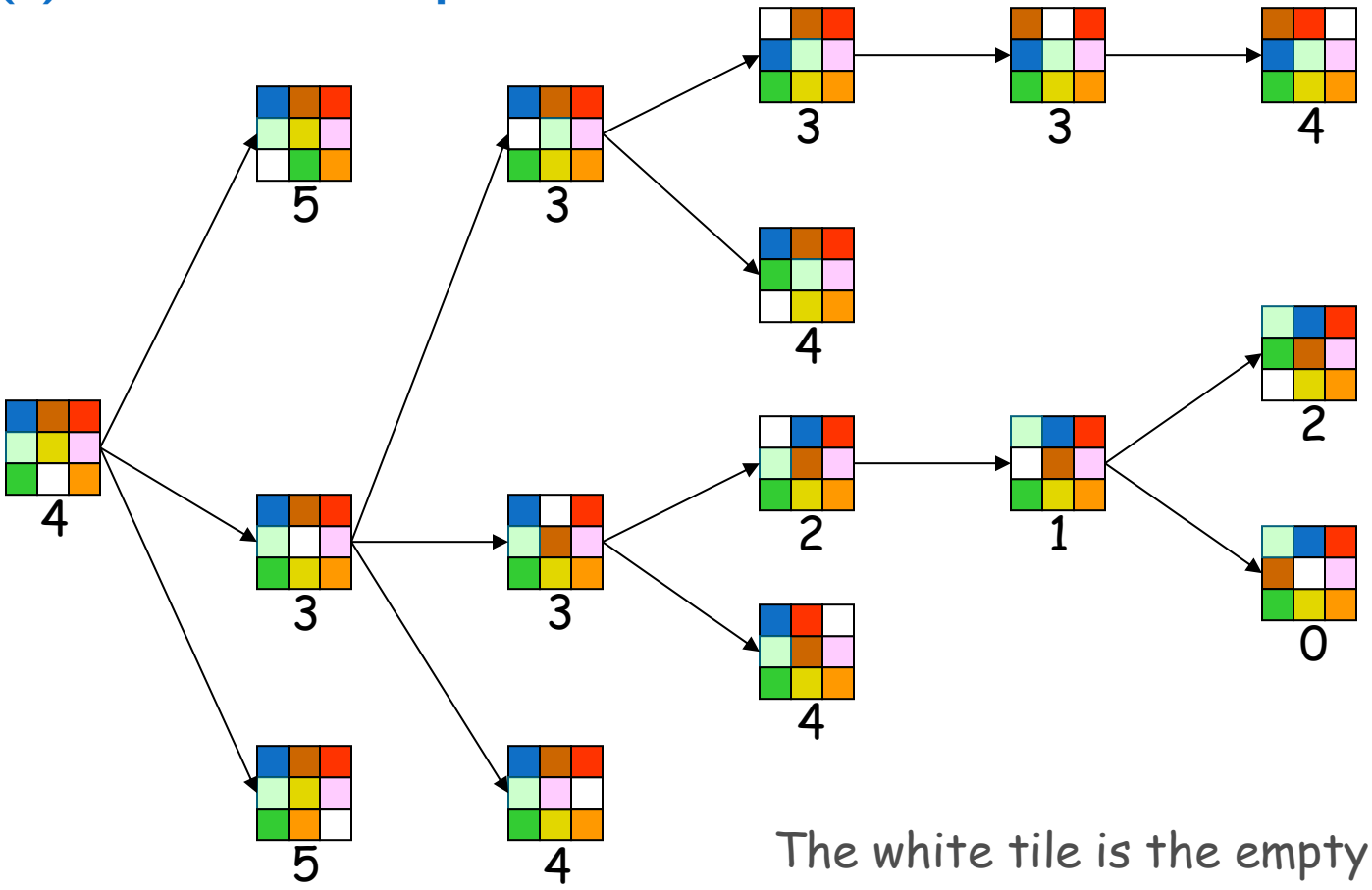
Goal state

- $h_1(N)$ = number of misplaced numbered tiles = 6
- $h_2(N)$ = sum of the (Manhattan) distance of every numbered tile to its goal position
  = 3 + 1 + 3 + 0 + 2 + 1 + 0 + 3 = 13

Manhattan distance : مجموع فواصل عمودی و افقی

جستجوی آگاهانه – مثال معمای ۸

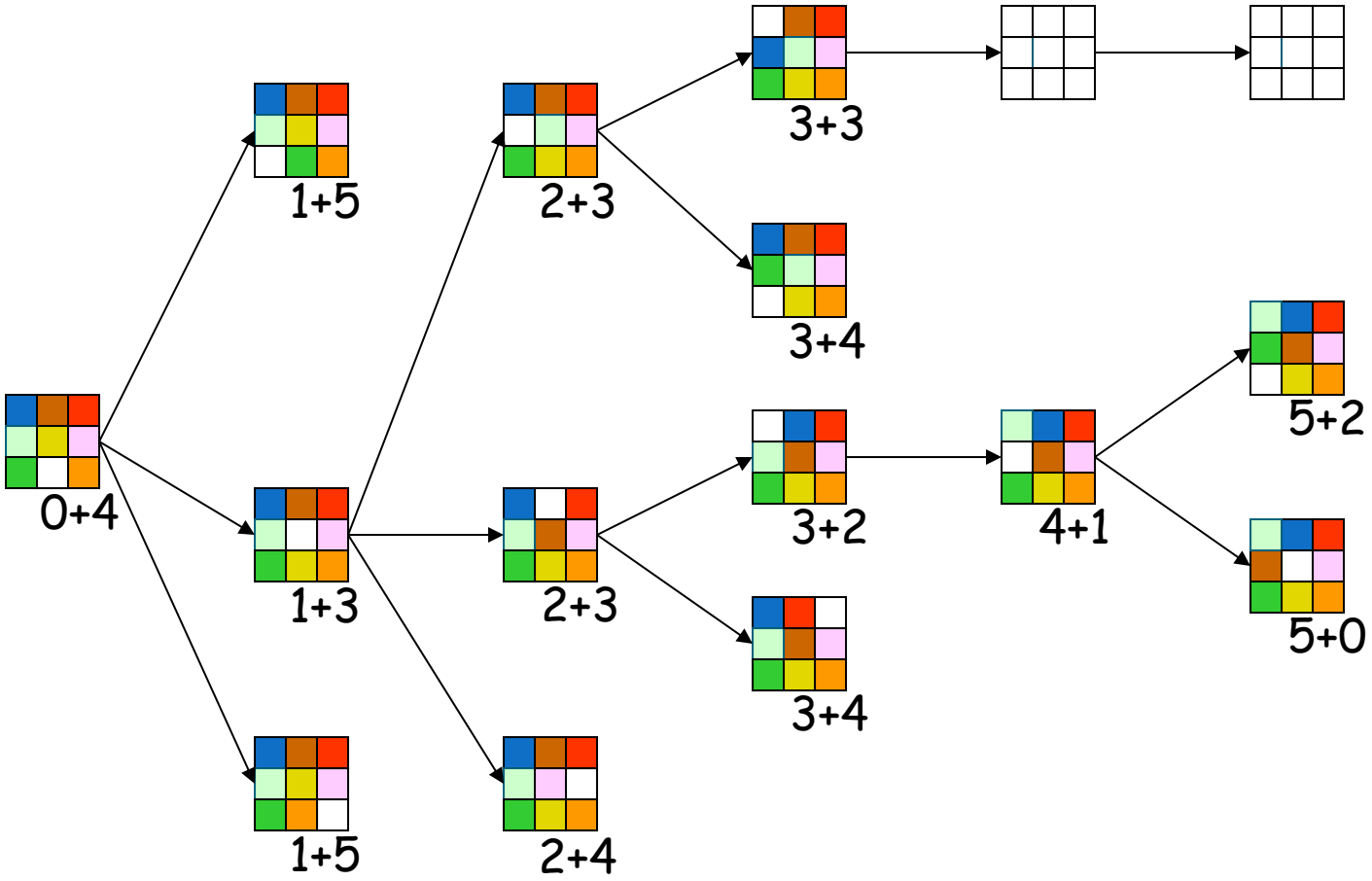
f(n) = h(n) = number of misplaced numbered tiles



The white tile is the empty tile

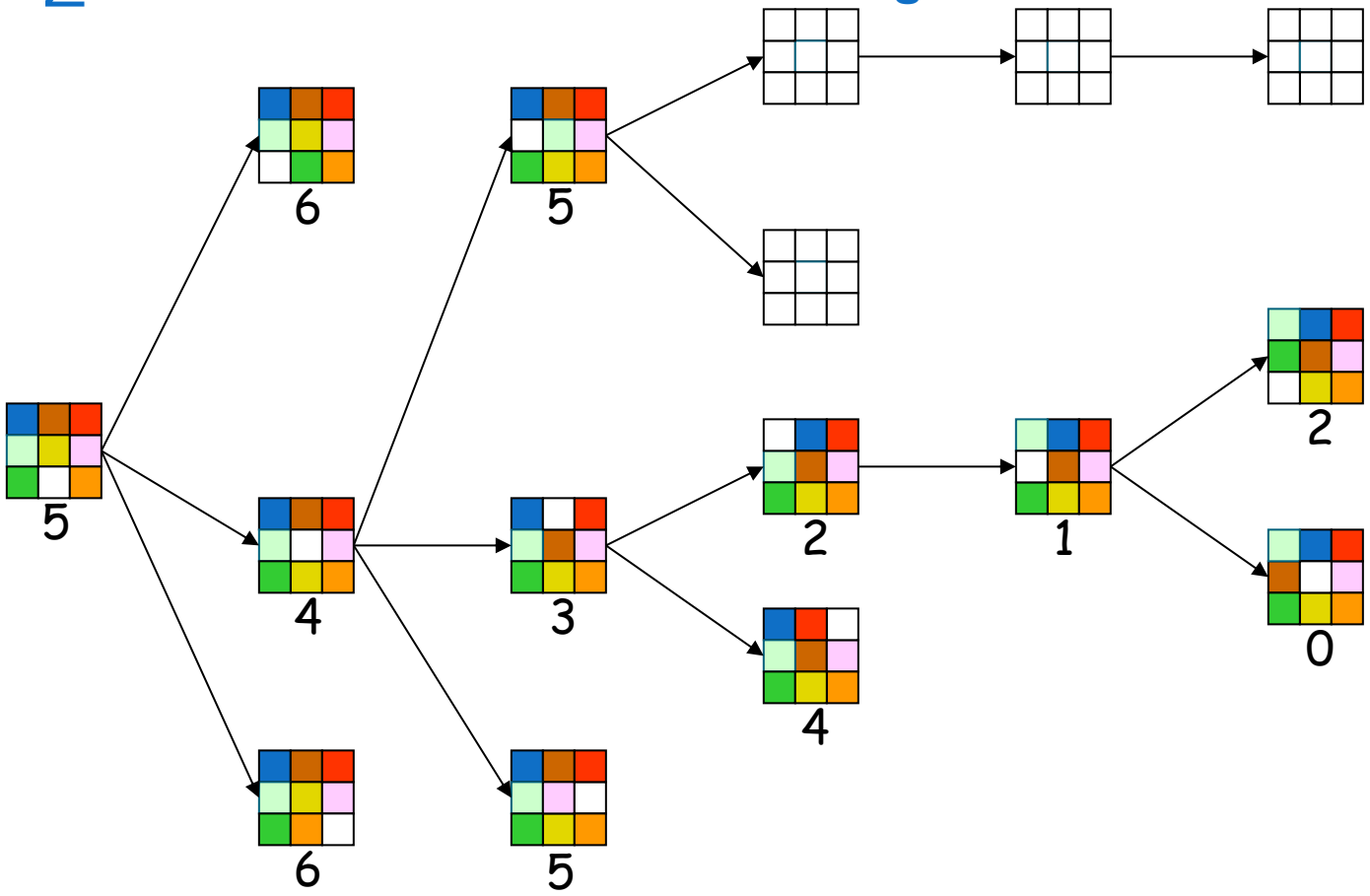
جستجوی آگاهانه – مثال معمای ۸

f(n) = g(n) + h(n) with h(n) = number of misplaced numbered tiles



جستجوی آگاهانه – مثال معمای ۸

f(n) = h(n) = ∑ distances of numbered tiles to their goals



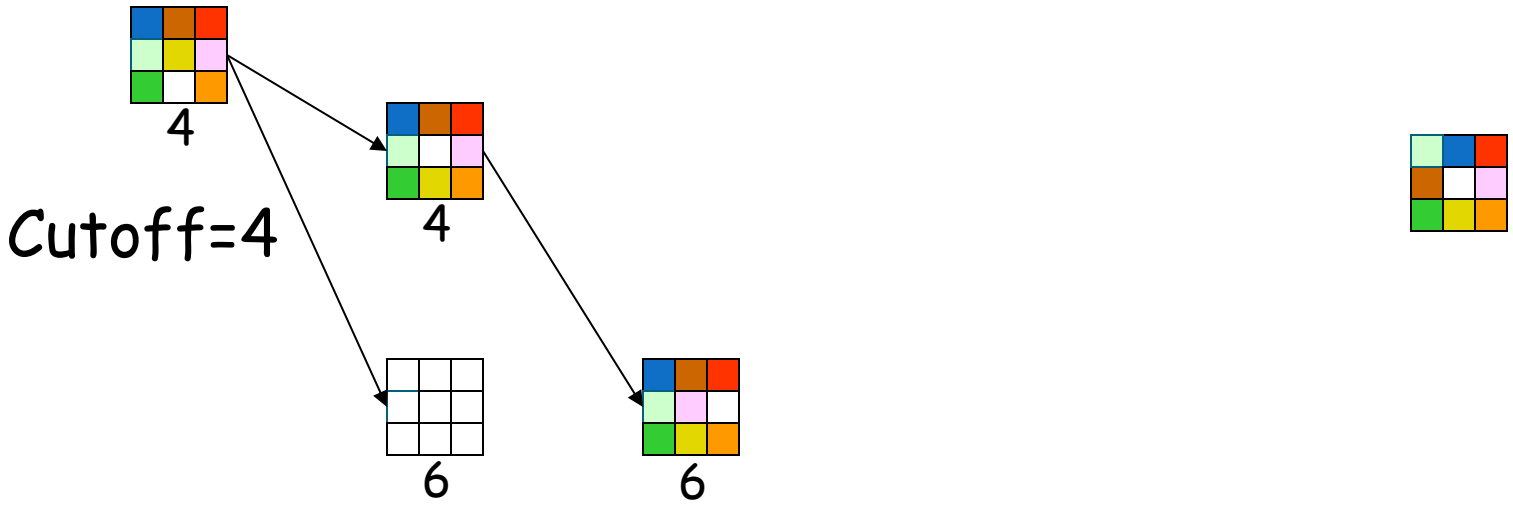
- بدنبال روش هایی هستیم که مشکل کمبود حافظه در روش A* را برطرف نماید:
  - الگوریتم A* عمیق شونده تکراری (Iterative-deepening A* (IDA*
  - الگوریتم جستجوی اول بهترین بازگشتی Recursive best-first search(RBFS)
  - الگوریتم A* با حافظه محدود ((S)MA* (simple) Memory-bounded A*

- در الگوریتم استاندارد عمیق شونده تکراری، برش (cutoff) بر اساس عمق بود.
- در این روش، برش بر اساس کوچک ترین هزینه f (یعنی g+h) است.
- در هر مرحله نودهایی گسترش می باند که هزینه آنها f(n) کمتر یا مساوی مقدار برش باشد.

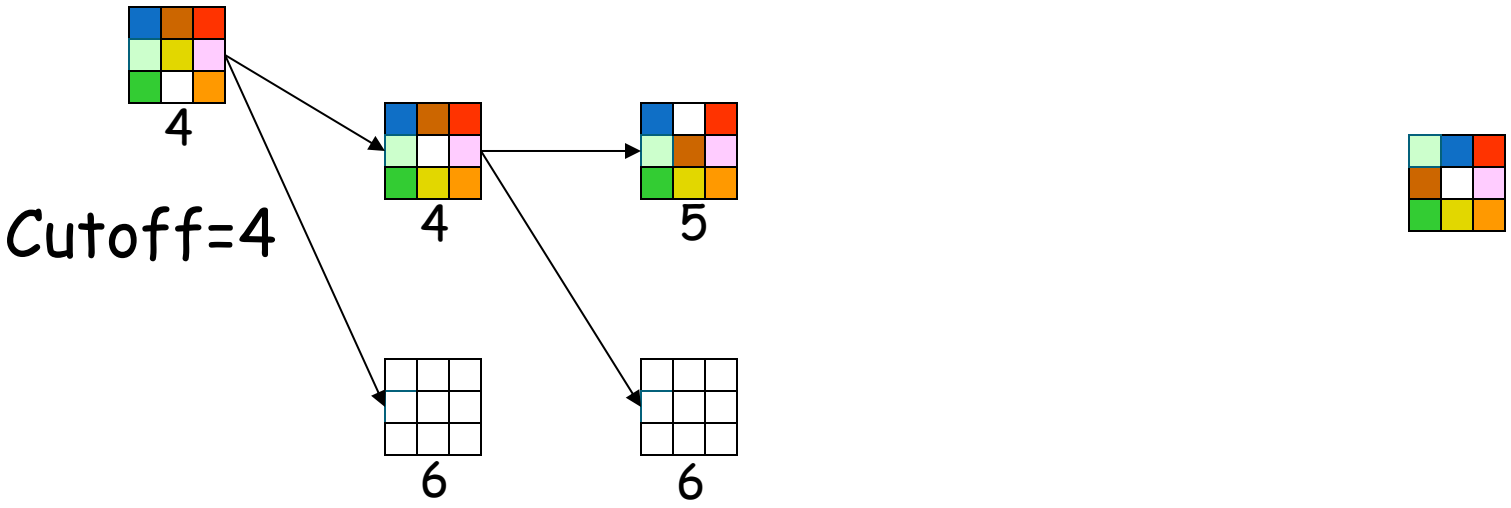
$$f(n) \leq \text{cutoff}$$

f(N) = g(N) + h(N) , with h(N) = number of misplaced tiles

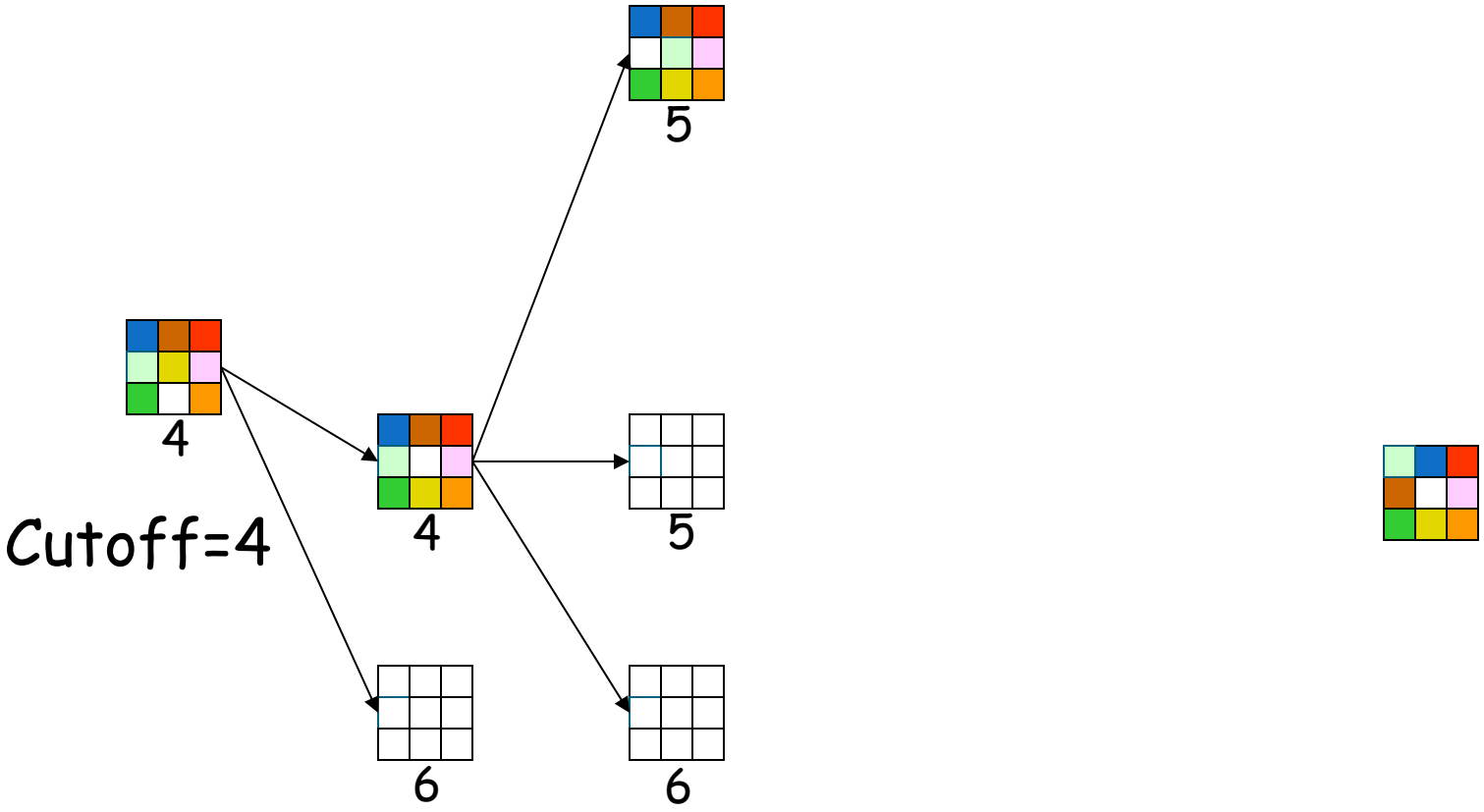
f(N) = g(N) + h(N) , with h(N) = number of misplaced tiles



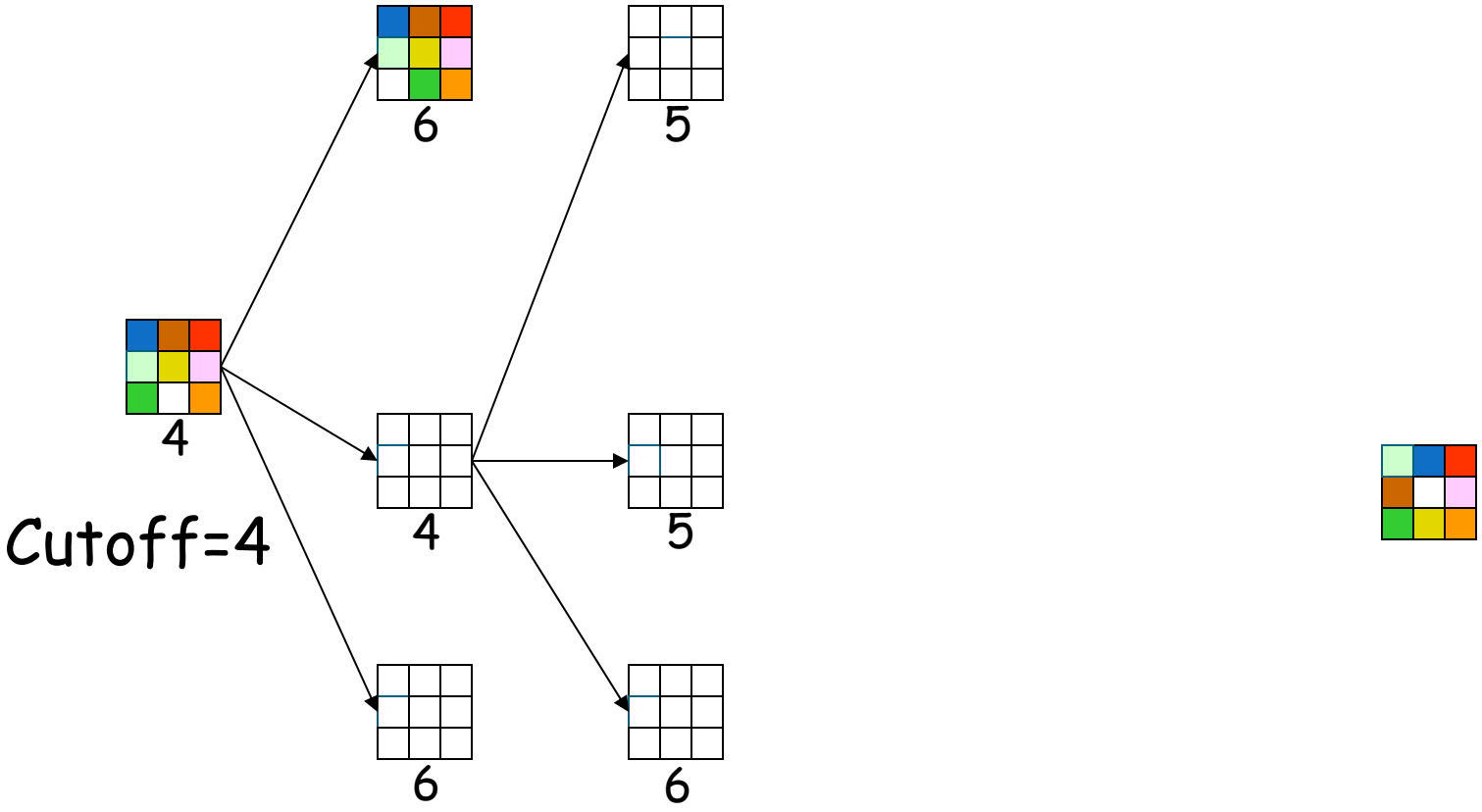
f(N) = g(N) + h(N) , with h(N) = number of misplaced tiles



f(N) = g(N) + h(N) , with h(N) = number of misplaced tiles

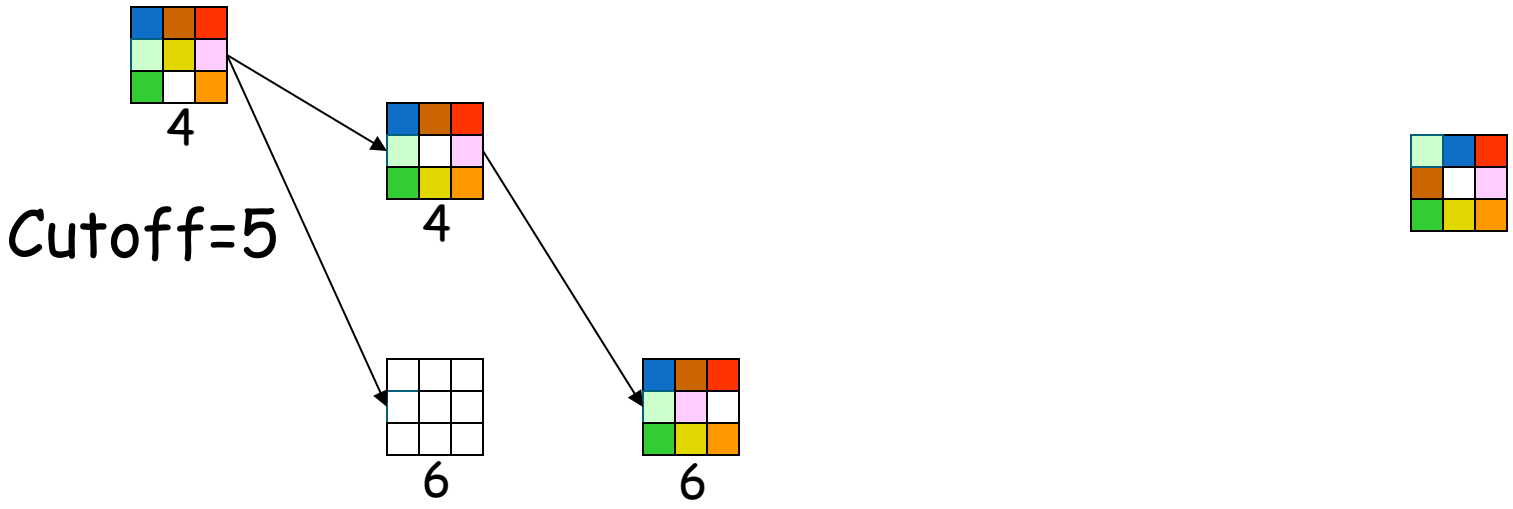


f(N) = g(N) + h(N) , with h(N) = number of misplaced tiles

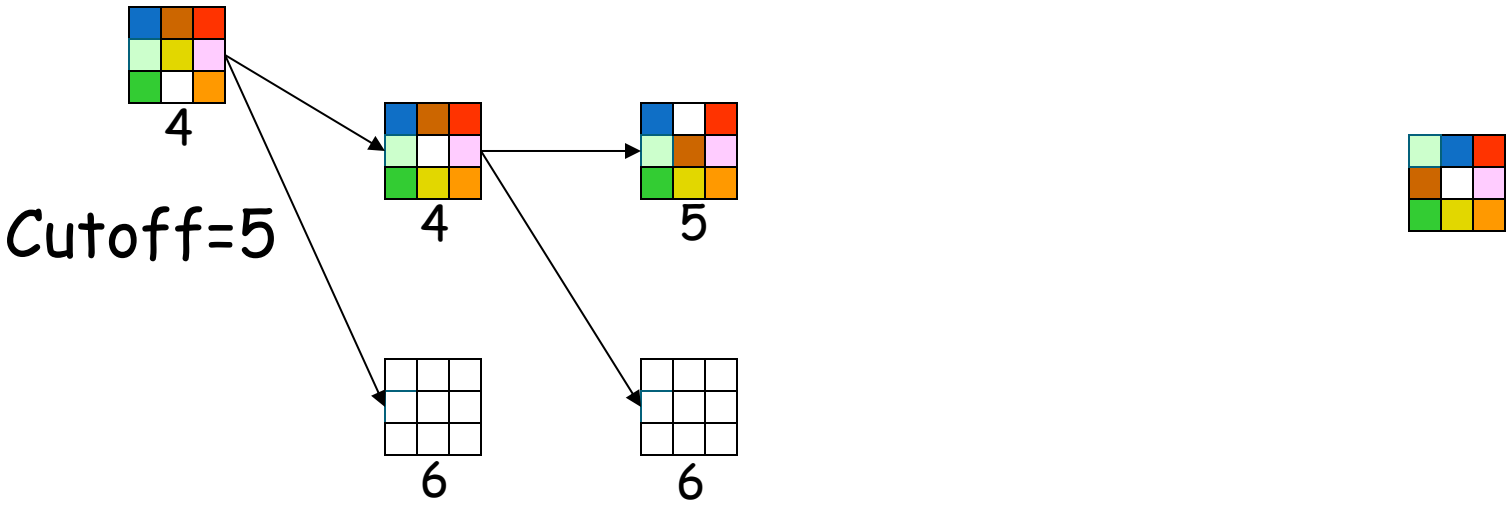


f(N) = g(N) + h(N) , with h(N) = number of misplaced tiles

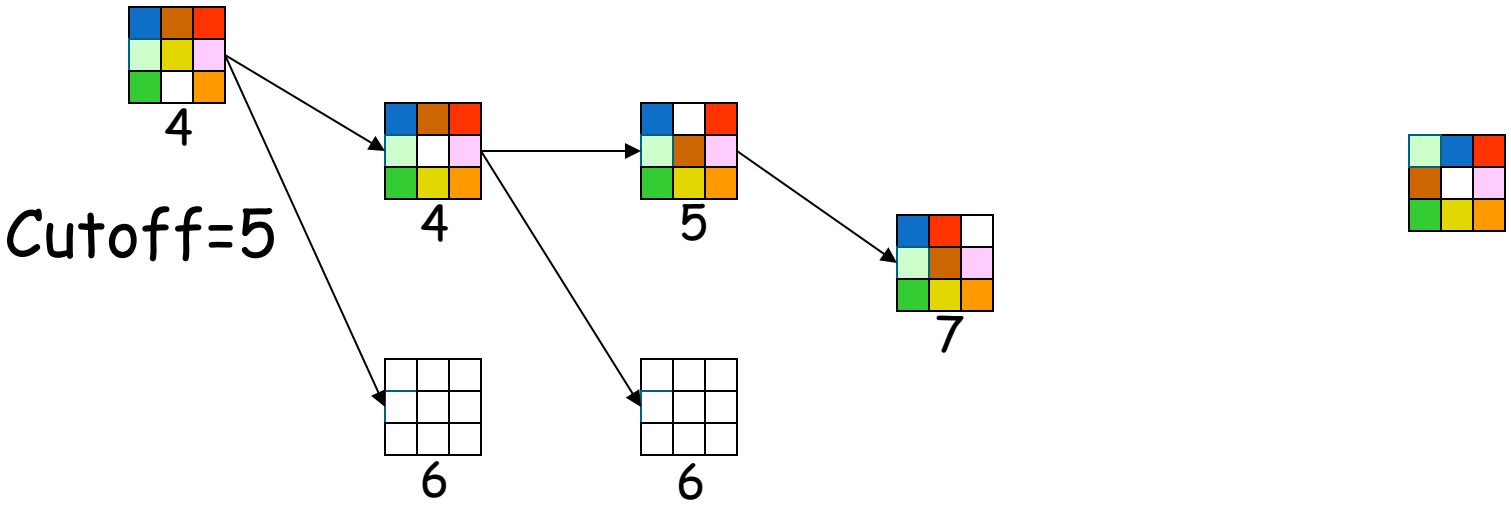
f(N) = g(N) + h(N) , with h(N) = number of misplaced tiles



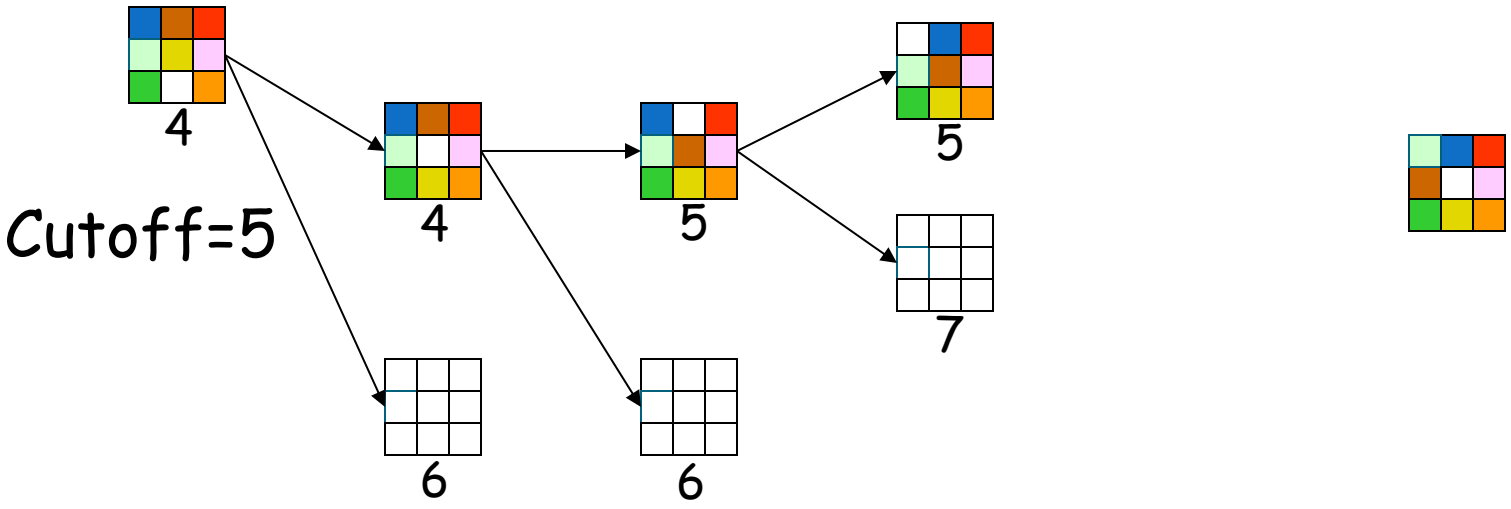
f(N) = g(N) + h(N) , with h(N) = number of misplaced tiles



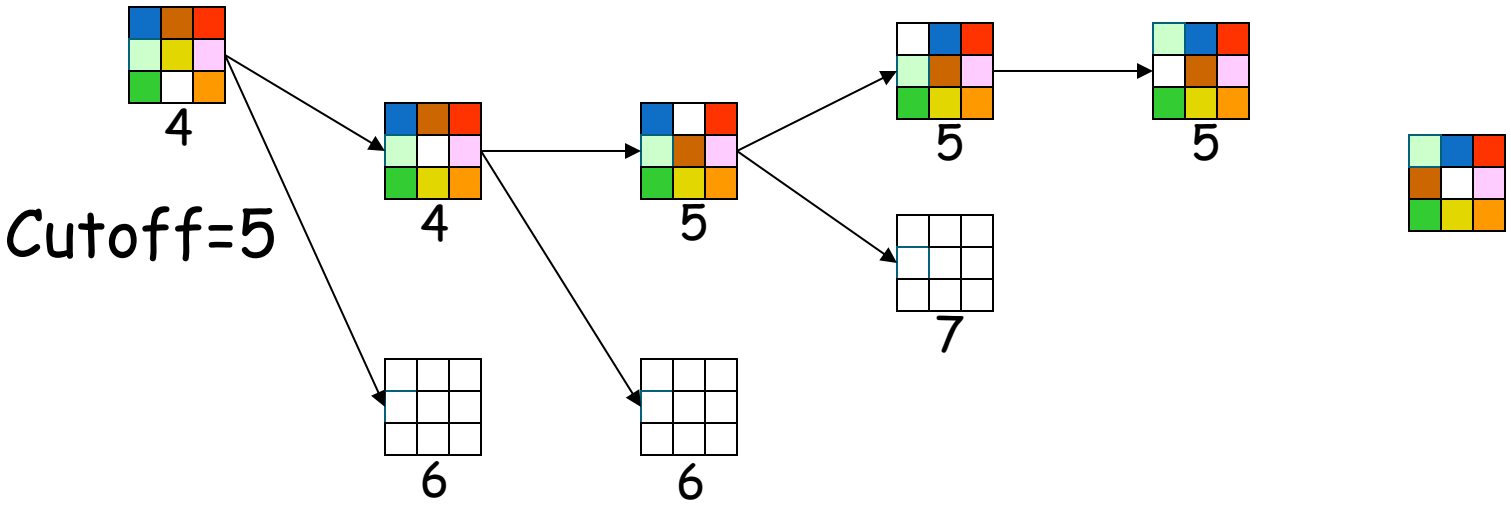
f(N) = g(N) + h(N) , with h(N) = number of misplaced tiles



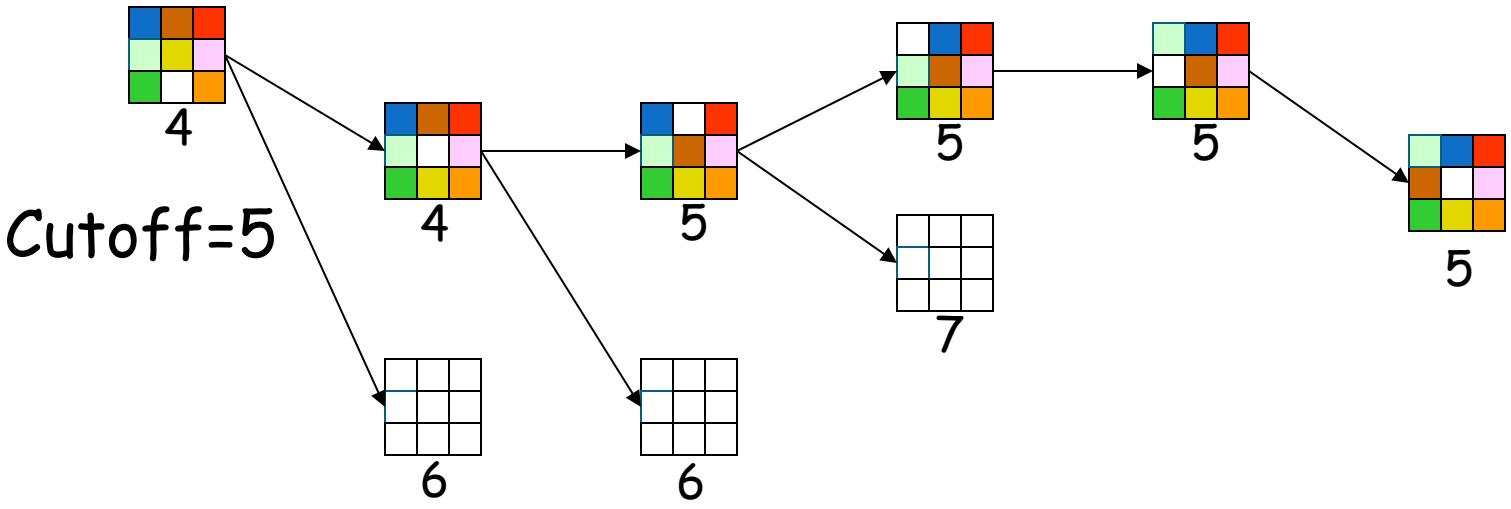
f(N) = g(N) + h(N) , with h(N) = number of misplaced tiles



f(N) = g(N) + h(N) , with h(N) = number of misplaced tiles

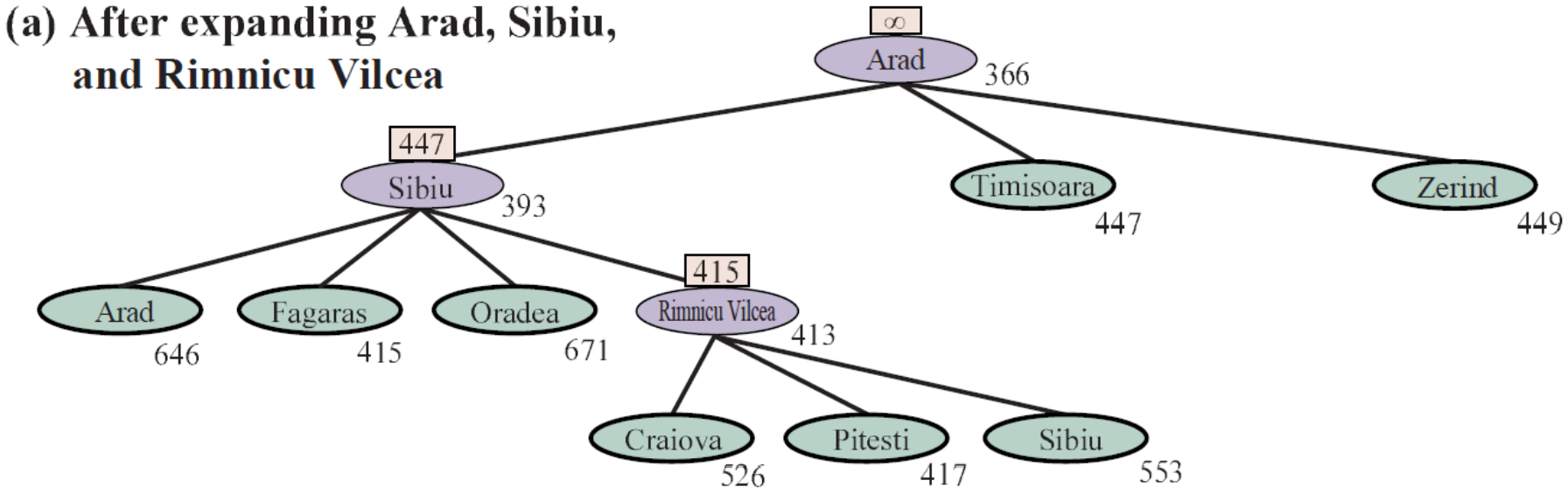


f(N) = g(N) + h(N) , with h(N) = number of misplaced tiles

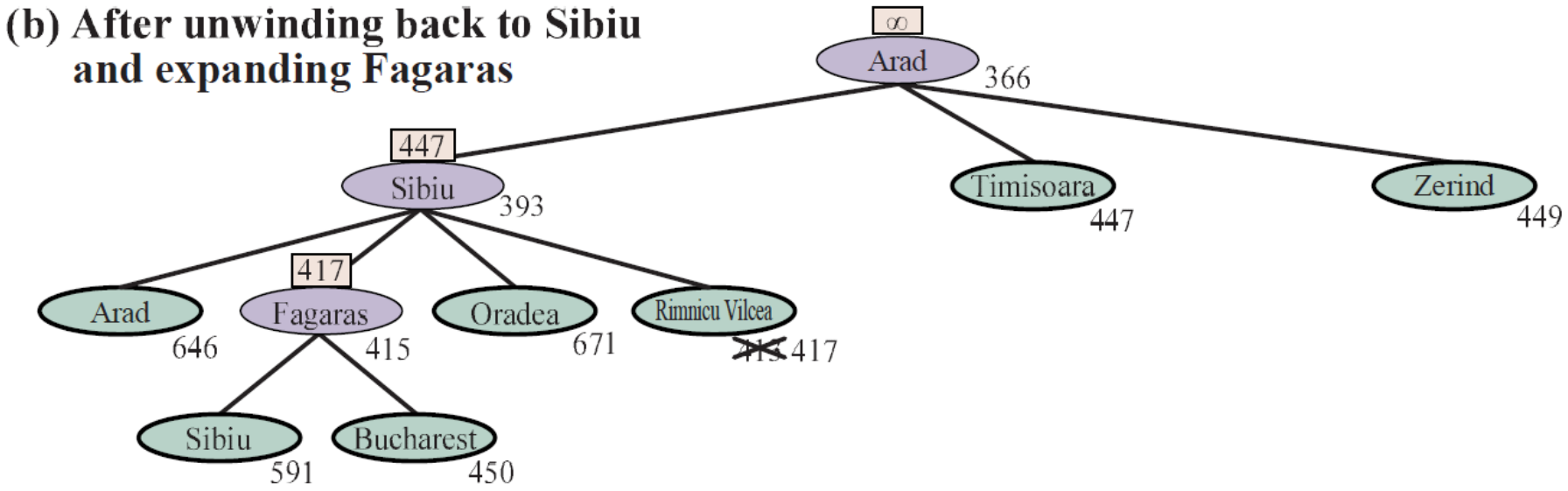


- ساختار روش جستجوی اول بهترین بازگشتی مشابه اول عمق است.
- اما به جای ادامه مسیر فعلی، خود را از مقدار f بهترین مسیر جایگزین قابل دسترس از همه اجداد گره فعلی، مطلع نگه می دارد.
- اگر گره فعلی از این مقدار فراتر برود، تابع بازگشتی آن را به عقب بر می گرداند تا در یک مسیر جایگزین دیگر قرار بگیرد.
- در حال برگشت به عقب، مقدار f برای هر گره در مسیر، به بهترین مقدار f فرزندانش تغییر می کند.
- بنابراین، RBFS مقدار f بهترین برگ در زیرگره فراموش شده را به خاطر می آورد و می تواند در آینده در صورت لزوم از آن برای توسعه زیر درخت مربوطه استفاده نماید.

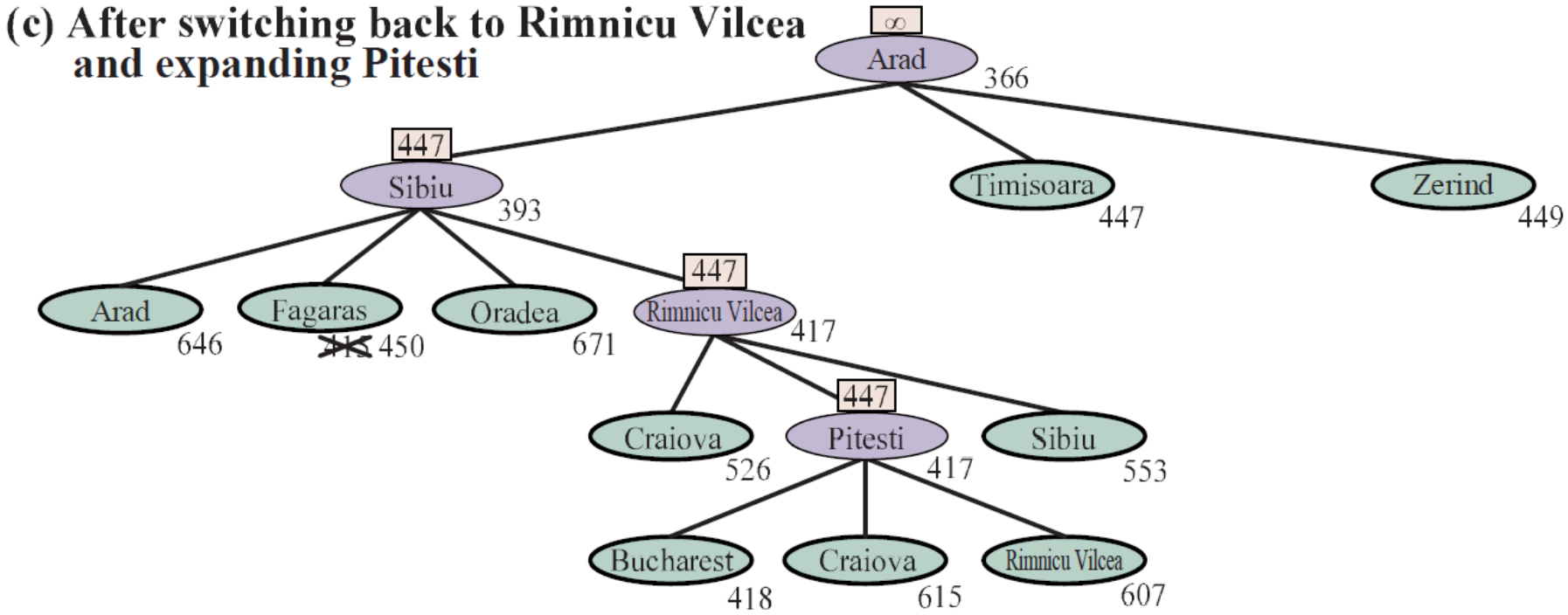
جستجوی آگاهانه / جستجوی هیوریستیک با حافظه محدود/ RBFS/ مثال



جستجوی آگاهانه / جستجوی هیوریستیک با حافظه محدود/ RBFS/ مثال



جستجوی آگاهانه / جستجوی هیوریستیک با حافظه محدود/ RBFS/ مثال



ارزیابی RBFS و IDA*

- هر دو روش کامل و بهینه هستند.
- پیچیدگی مکانی : O(bd)
- هر دو ممکن است یک حالت را چندین بار تکرار کنند.
- هر دو مشکل حافظه بسیار اندک دارند و حتی اگر حافظه بیشتری در اختیار داشته باشند نمی توانند از آن استفاده نمایند.

جستجوی آگاهانه / جستجوی هیوریستیک با حافظه محدود/SMA*

- جستجوی A* با حافظه محدود ساده شده، از همه حافظه‌ی در دسترس استفاده می کند.
- مشابه A* تا وقتی حافظه باشد، بهترین گره را گسترش می دهد.
- بعد از پر شدن حافظه، بدترین گره برگی (بیشترین مقدار f) حذف می شود.
- مشابه RBFS مقدار گره فراموش شده به والدینش برگردانده می شود.
- SMA* زمانی کامل و بهینه است که راه حل دست یافتنی وجود داشته باشد.

مقایسه برخی از الگوریتم های جستجو در بازی های کامپیوتری

Table 3. Result of time

Years	Authors	Result (Time)
2007	[17]	GA better than A*
2011	[4]	GA better than A*
2012	[11]	GA better than First Search
2016	[5]	ACO better than GA
2016	[2]	A* better than Dijkstra
2017	[19]	Improved A* better than A*
2017	[20]	IDA* better than A*
2018	[3]	A* better than Bee (in the simple map)
2018	[21]	Hybrid A* better than A*

Table 3 shows the results of pathfinding algorithms used for game development based on the performance of time. For heuristic techniques, the A* algorithm is faster than Dijkstra for calculating and searching a path [2]. However, a few researchers improved the A* algorithm. In 2017, Firmansyah [19], Primanita [20] and Sazaki [21] improved the A* algorithm and the result was much better than the basic A* algorithm. Overall, for heuristic techniques, the A* algorithm has a good result compared to others, but it can be bettered by the improved A* algorithm.

Table 4. Result of memory

Year	Author	Result(Memory)
2017	[20]	IDA* better than A* on an empty map
2018	[3]	Bee algorithm better than A* on a complex map
2018	[21]	Hybrid A* gets better results compared to A* algorithm on an empty track

Table 4 shows the results of pathfinding algorithms used for game development based on the performance of memory usage. Three of 10 papers investigated the performance of pathfinding algorithms in terms of memory. For heuristic techniques, the improved A* algorithms like IDA* [10] and Hybrid A* [31] have better results compared to the basic A* algorithm. From the result, we can conclude that improving the algorithm results in reduced memory usage. For a complex map, metaheuristic techniques are very good compared to heuristic techniques. For example, the Bee

Warcraft

IOP Conf. Series: Materials Science and Engineering769 (2020) 012021 IOP Publishing doi:10.1088/1757-899X/769/1/012021

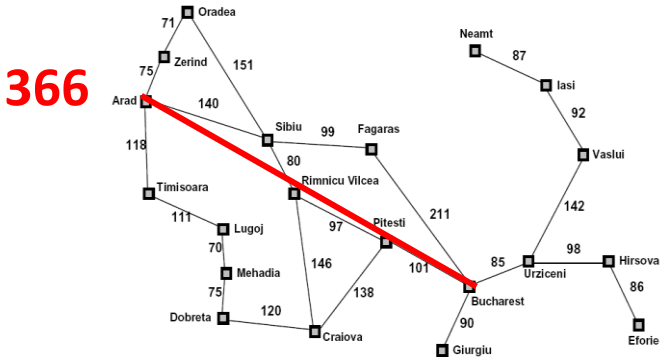
How does Google Maps determine routes?

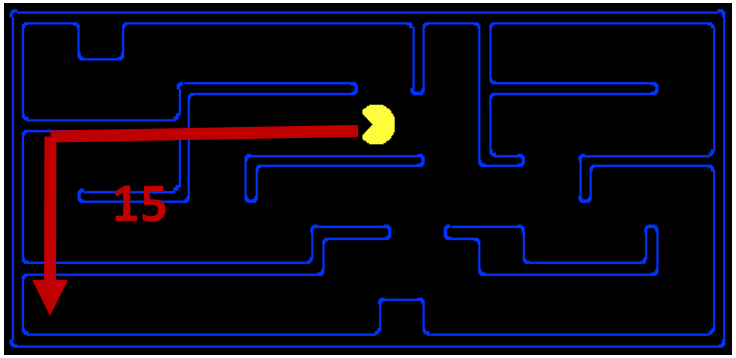
- **Google Maps uses a variety of algorithms to determine the shortest path between two points:**
- Here are some of the algorithms that may be used:
- **Dijkstra's algorithm:** This is a classic algorithm for finding the shortest path between two nodes in a graph. It works by starting at the source node and progressively exploring the graph, adding nodes to the shortest path as it goes.
- **A* search algorithm:** This is another popular algorithm for finding the shortest path between two points. It works by combining the benefits of Dijkstra's algorithm with a heuristic function that helps guide the search toward the destination node.
- It's worth noting that Google Maps may use a combination of these algorithms, as well as other specialized algorithms, to determine the shortest path between two points. The specific algorithms used may vary depending on the specifics of the route, such as the distance, the number of turns, and the type of terrain.

https://www.geeksforgeeks.org/how-does-google-map-works/
More: https://stackoverflow.com/questions/6937459/which-algorithm-does-google-maps-use-to-compute-the-direction-between-2-points

طراحی تابع هیوریستیک

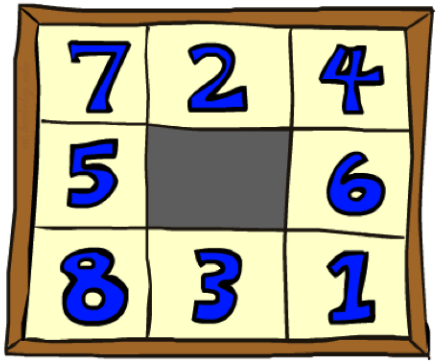
- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to *relaxed problems*, where new actions are available



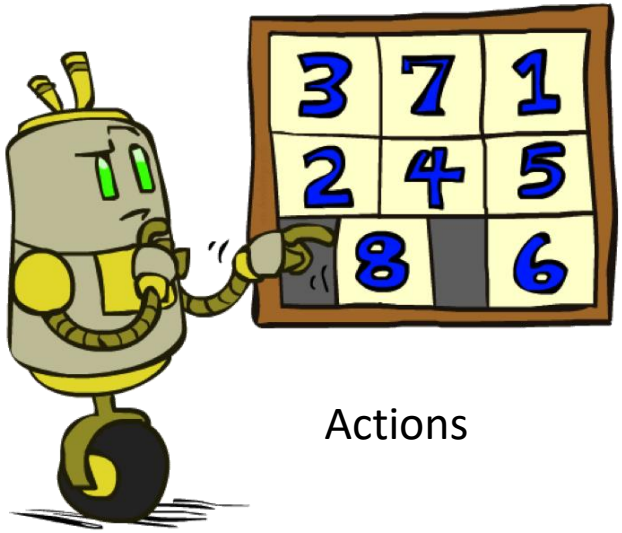


- Inadmissible heuristics are often useful too

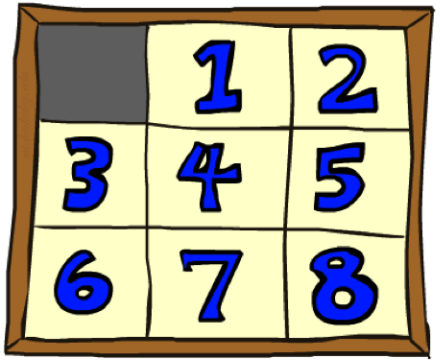
Example: 8 Puzzle



Start State



Actions

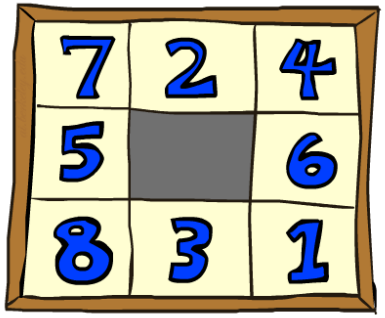


Goal State

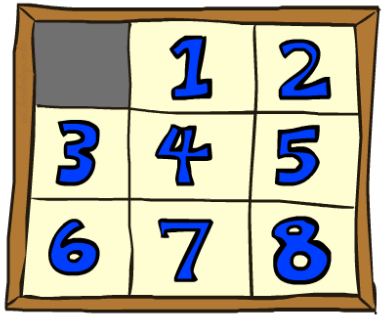
- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

8 Puzzle I

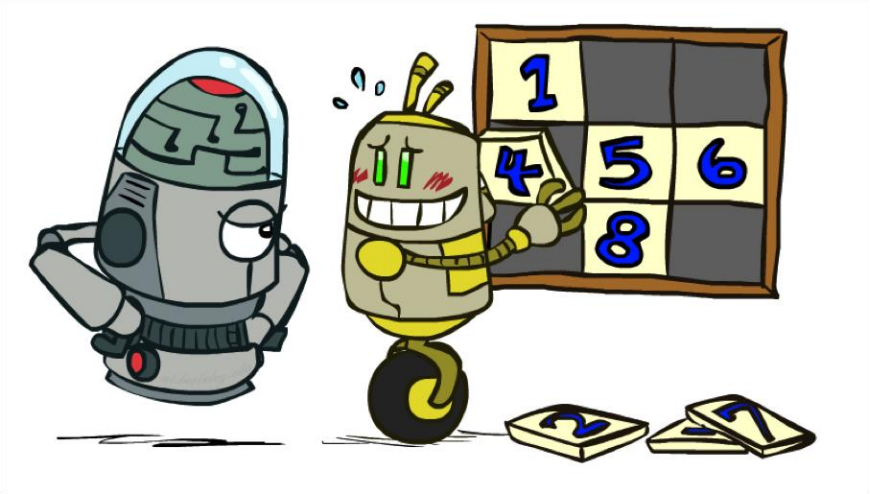
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8
- This is a *relaxed-problem* heuristic



Start State



Goal State

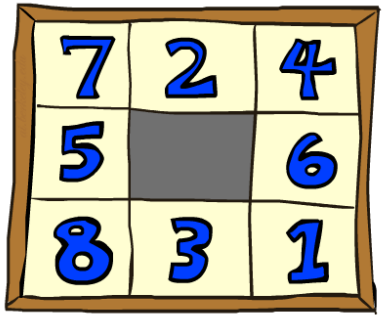


	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

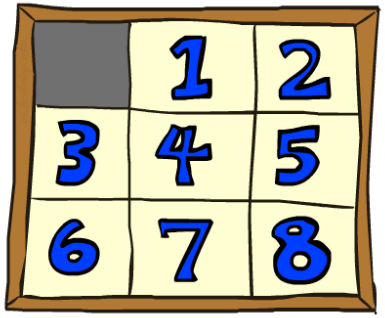
Statistics from Andrew Moore

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(start) = 3+1+2+2+2+3+3+2 = 18



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

جستجوی آگاهانه / طراحی تابع هیوریستیک

- ضریب انشعاب مؤثر (b*)
  - معیاری برای ارزیابی کیفیت یک تابع هیوریستیک است.
  - اگر تعداد کل گره های تولید شده توسط A* برابر N و عمق راه حل d باشد، آنگاه b* ضریب انشعابی است که یک درخت یکنواخت به عمق d باید داشته باشد تا دارای N+1 گره باشد:

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

- اگر تابع هیورستیکی خوب تعریف شده باشد مقدار b* آن حدودا نزدیک به ۱ است.

جستجوی آگاهانه / طراحی تابع هیوریستیک

- برای آزمایش توابع هیورستیک h1 و h2 ۱۲۰۰ مسئله تصادفی با طول راه حل ۲ تا ۲۴ تولید و با IDS و نیز A* با استفاده از h1 و h2 حل شده است.
- نتایج نشان می دهد h2 بهتر از h1 و A* خیلی بهتر از IDS می باشد.

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

جستجوی آگاهانه / طراحی تابع هیوریستیک

- Comparison of the search costs and effective branching factors for 8-puzzle problems using breadth-first search, A∗ with h1 (misplaced tiles), and A∗ with h2 (Manhattan distance). Data are averaged over 100 puzzles for each solution length d from 6 to 28.

	Search Cost (nodes generated)			Effective Branching Factor		
d	BFS	A*(h_1)	A*(h_2)	BFS	A*(h_1)	A*(h_2)
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17270	1683	364	1.74	1.48	1.32
18	41558	4102	751	1.72	1.49	1.34
20	91493	9905	1318	1.69	1.50	1.34
22	175921	22955	2548	1.66	1.50	1.34
24	290082	53039	5733	1.62	1.50	1.36
26	395355	110372	10080	1.58	1.50	1.35
28	463234	202565	22055	1.53	1.49	1.36

جستجوی آگاهانه / طراحی تابع هیوریستیک

- بنابراین اگر برای دو هیورستیک h1 و h2 برای هر گره n، h2(n)≥h1(n) باشد می گوییم h2 نسبت به h1 برتری دارد.
- یعنی A* با استفاده از h2 هرگز گره های بیشتری نسبت به A* با استفاده از h1 تولید نخواهد کرد.
- بنابراین بهتر است همیشه از توابع هیورستیک با مقدار h بالاتر استفاده کنیم به شرطی که بیشتر از اندازه تخمین نزند.

جستجوی آگاهانه / طراحی تابع هیوریستیک

- مسئله تعدیل شده (relaxed problem)
  - مسئله ای که تعداد محدودیت های کمتری برای اقدامات دارد.
- هزینه یک راه حل بهینه برای یک مسئله تعدیل شده، یک تابع هیورستیک قابل قبول برای مسئله اصلی است.

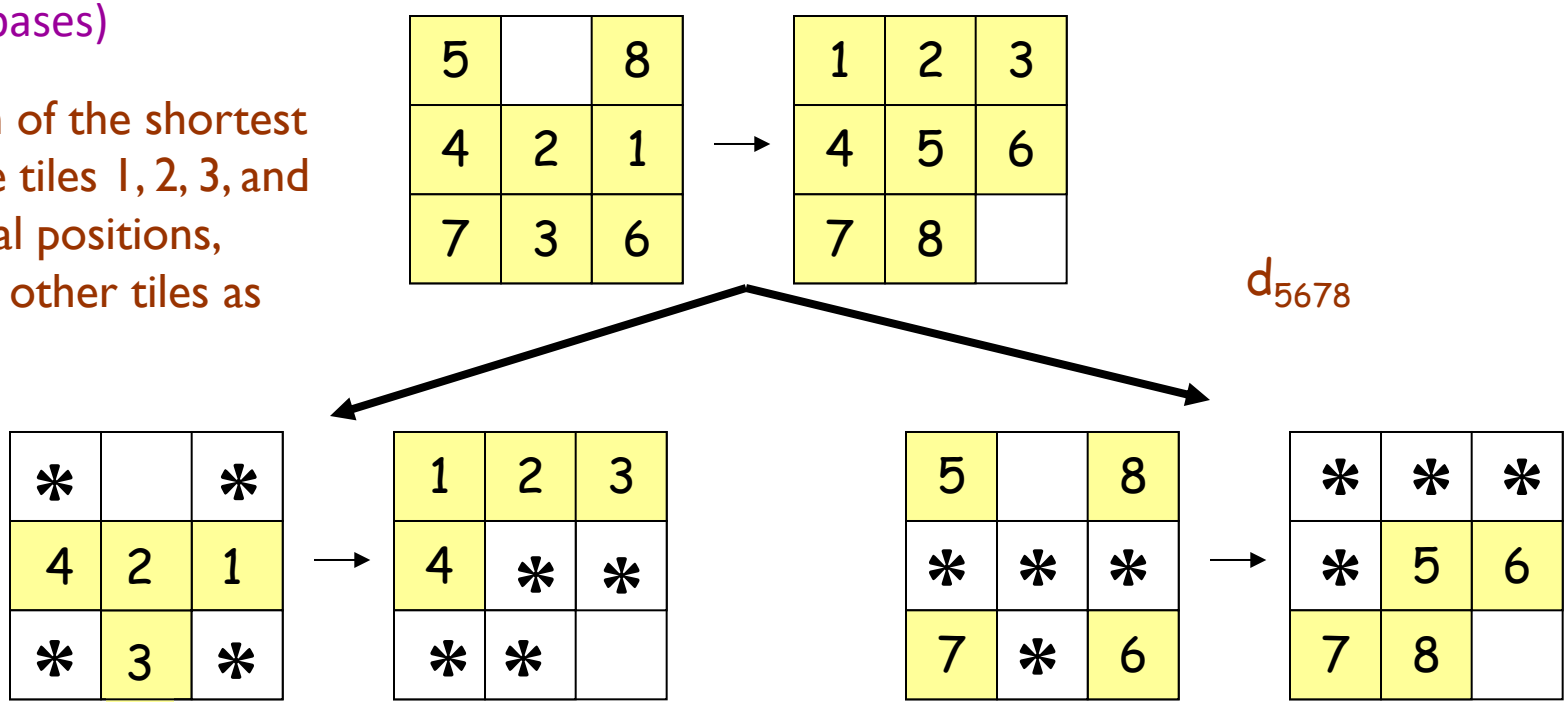
جستجوی آگاهانه / طراحی تابع هیوریستیک/مثال

- برای معمای ۸، اقدامات به صورت زیر بود:
  - یک کاشی می تواند از مربع A به B برود اگر B مجاور عمودی یا افقی A بوده و خالی باشد.
- با حذف هر یک یا همه شروط می توان مسائل تعدیل شده زیر را داشت:
  - یک کاشی می تواند از مربع A به B برود اگر B مجاور عمودی یا افقی A باشد. ← هیورستیک h2 امتیاز مناسبی خواهد داشت.
  - یک کاشی می تواند از مربع A به B برود . ← هیورستیک h1 امتیاز مناسبی خواهد داشت.
  - می خواهیم مسائل تعدیل شده آنقدر ساده باشند که بدون جستجو بتوان آنها را حل نمود تا بتوان تخمین های خوبی بدست آورد.

جستجوی آگاهانه / طراحی تابع هیوریستیک/مثال

Generating heuristics from subproblems:
(Pattern databases)

d_{1234} = length of the shortest path to move tiles 1, 2, 3, and 4 to their goal positions, assuming the other tiles as identical



$h = d_{1234} + d_{5678}$

d_{1234} and d_{5678} are pre-computed and stored

$$h(n) = \max\{h1(n), h2(n), \ldots, hm(n)\}$$

Admissibility and consistency

- an admissible heuristic Admissible heuristic is one that *never overestimates* the cost to reach a goal. (An admissible heuristic is therefore *optimistic*.)
- A slightly stronger property is called consistency. A heuristic $h(n)$ is consistent if, for every node n and every successor n' of n generated by an action a, we have:

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

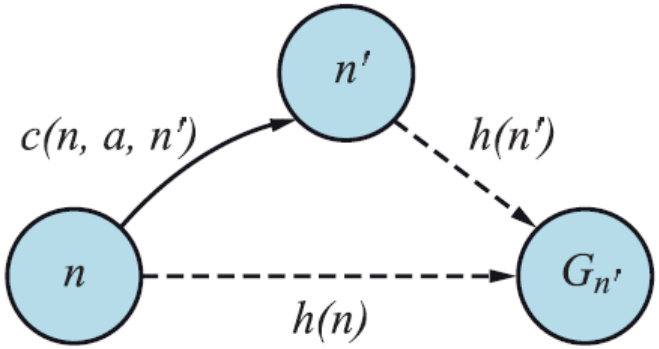


Figure 3.19 Triangle inequality: If the heuristic h is **consistent**, then the single number $h(n)$ will be less than the sum of the cost $c(n,a,a')$ of the action from n to n' plus the heuristic estimate $h(n')$.

Admissibility and consistency

- Every consistent heuristic is admissible (but not vice versa), so with a consistent heuristic, A∗ is cost-optimal.
- In addition, with a consistent heuristic, the first time we reach a state it will be on an optimal path, so we never have to re-add a state to the frontier, and never have to change an entry in reached
- A∗ with a consistent heuristic is optimally efficient in the sense that any algorithm that extends search paths from the initial state, and uses the same heuristic information, must expand all nodes that are surely expanded by A∗ (because any one of them could have been part of an optimal solution).