DATA130008 Introduction to Artificial Intelligence

复旦大学大数据学院
School of Data Science, Fudan University

魏忠钰

Informed Search

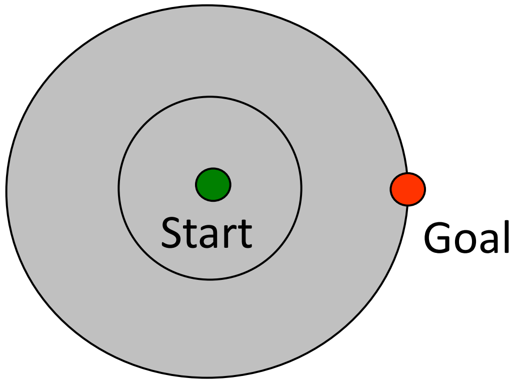
September 28th, 2021

Outline

- Heuristics Function

Informed Search

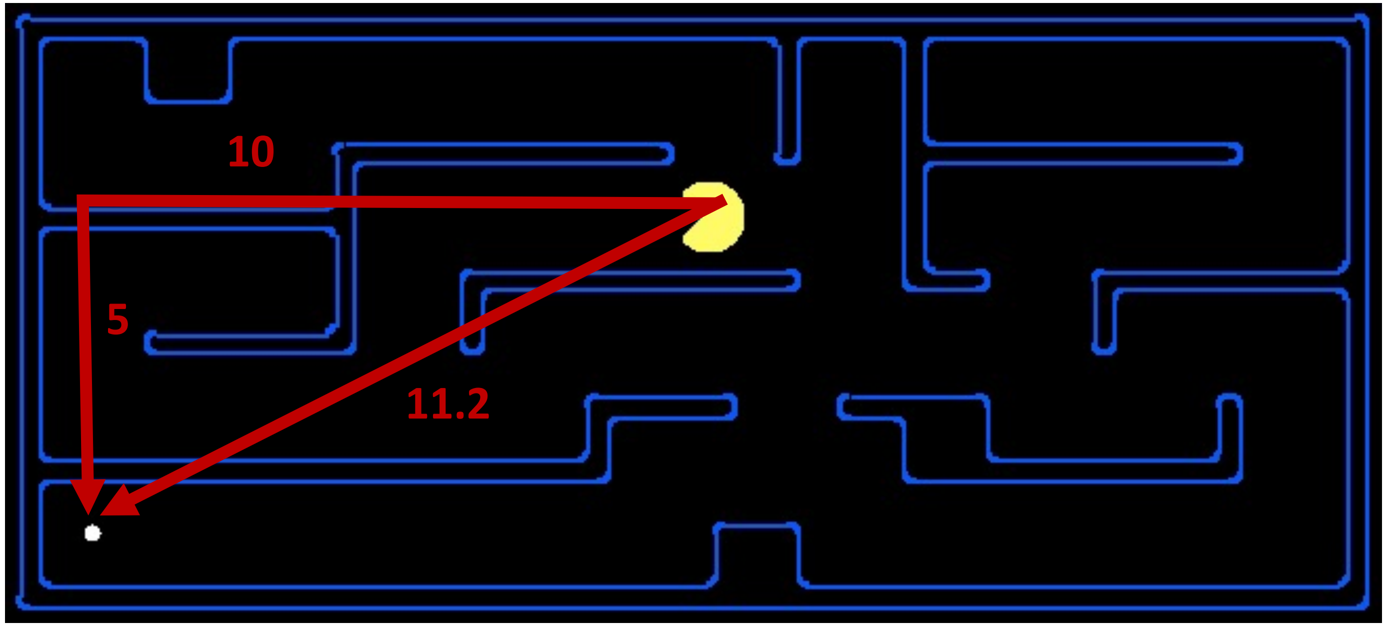
- In uninformed search, we only care about past, but never “look ahead”.
  - Consider the distance from the start to current node (cost in USC search), ignoring the future estimation.



- Often we have some other knowledge about nodes in the search tree.

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

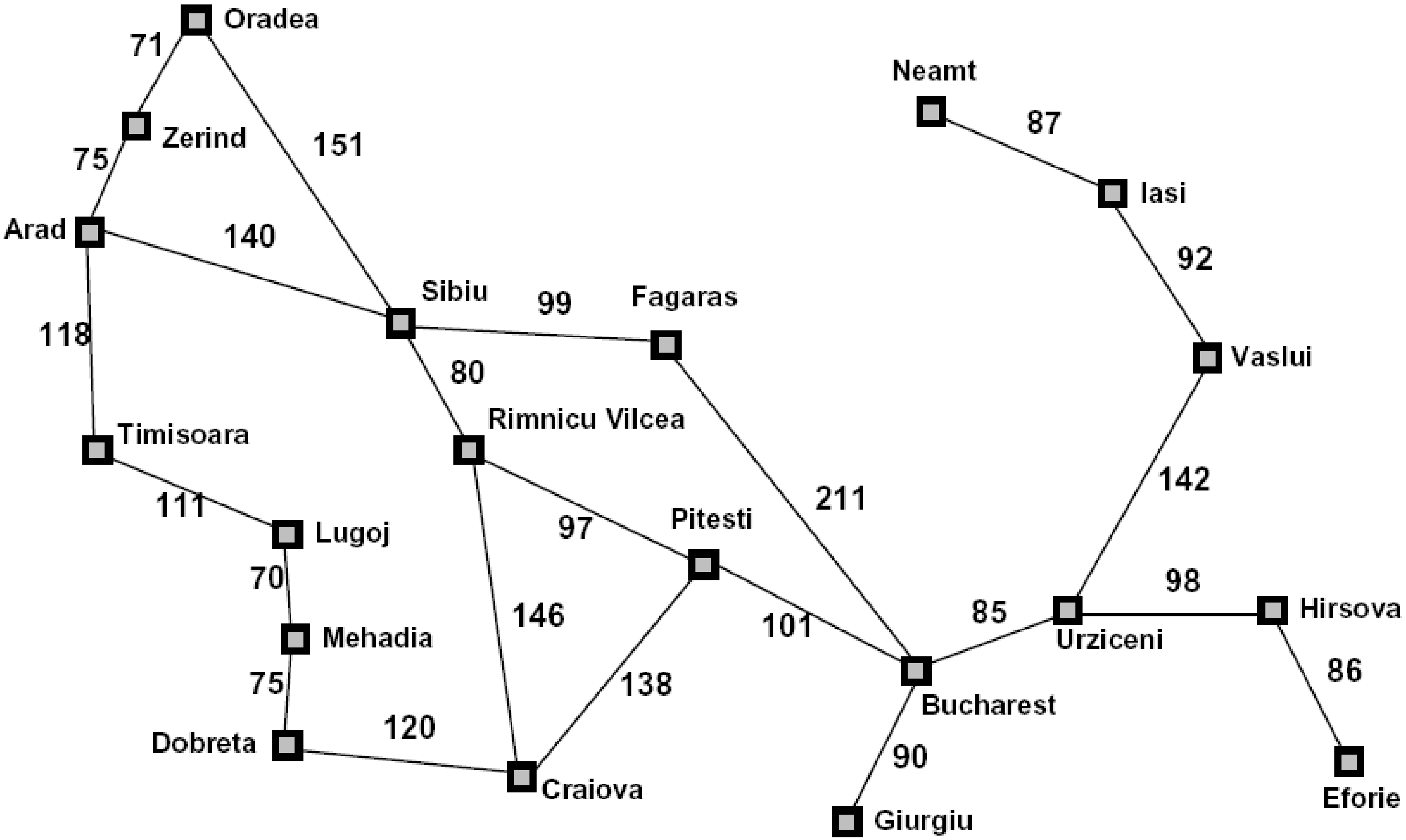


Manhattan distance: $|x_1 - x_2| + |y_1 - y_2|$

Euclidean distance: $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Heuristic Example: Romania Traveling

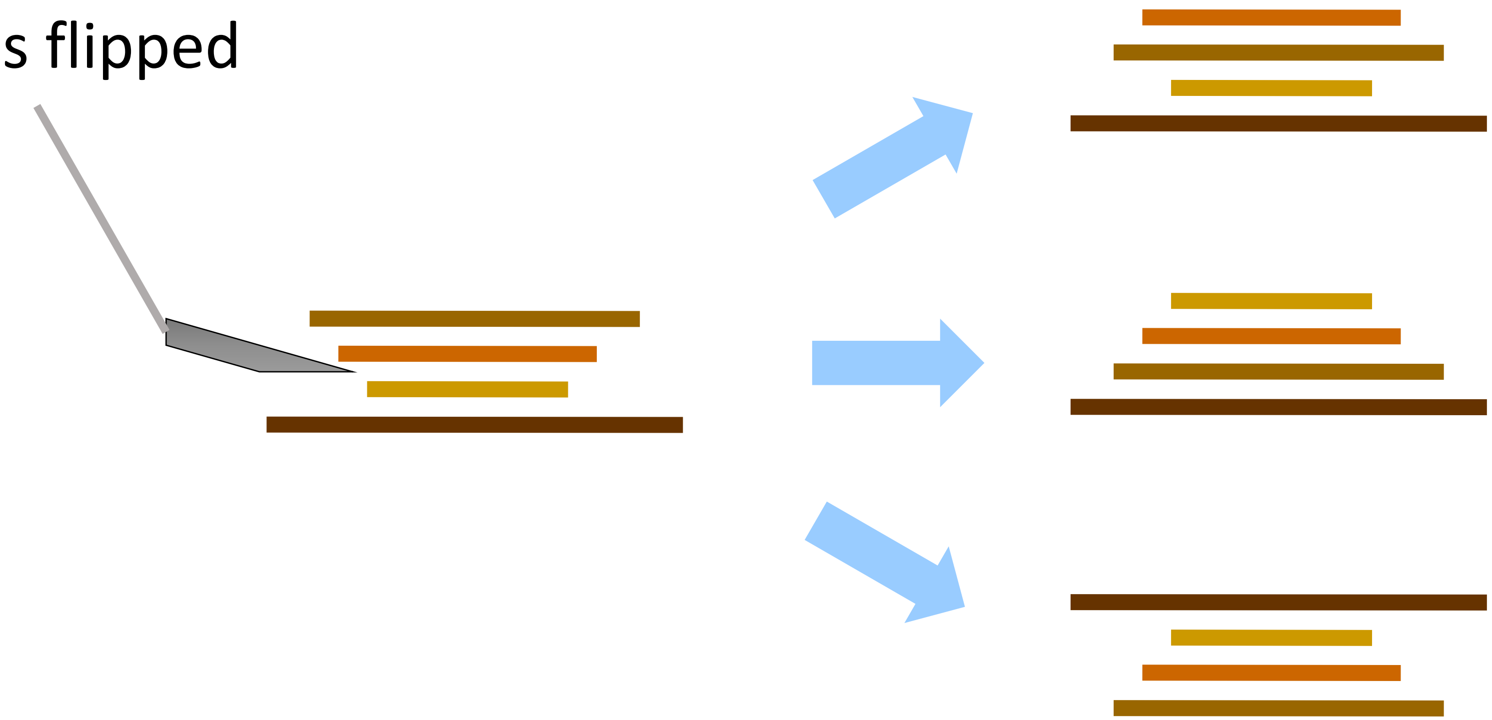
A heuristic is: straight-line distance to Bucharest.



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

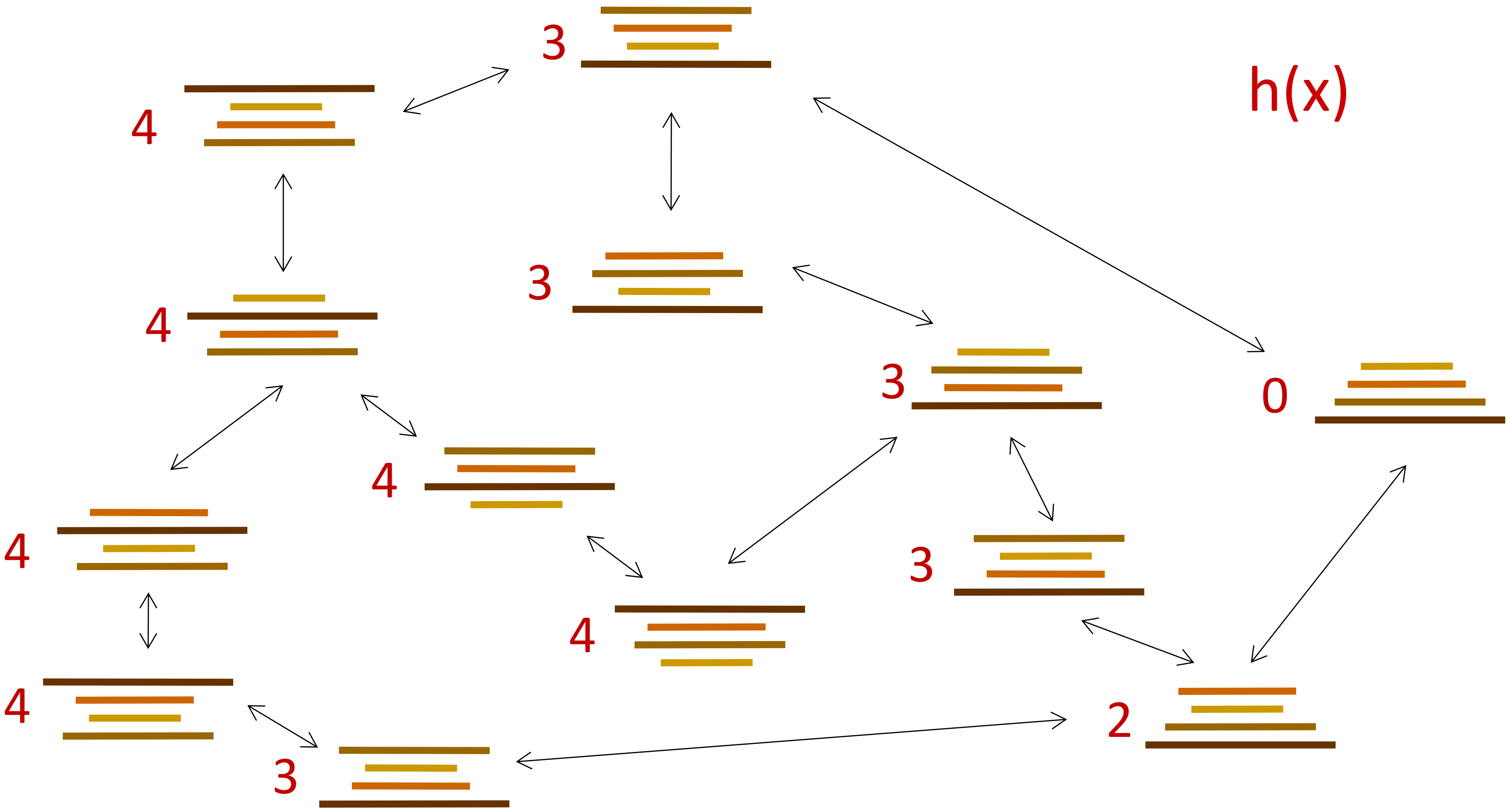
Heuristic Example: Pancake Problem

- Start state: a stack of disordered pancakes
- Goal state: an ordered pancake stack
- Result function: flip pancakes
  - Action: choose any pancake to flip and all pancakes on top the target one (included) will be reversed.
  - Cost: Number of pancakes flipped



Example: Heuristic Function

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



Outline

- Heuristics Function
- Greedy Search

Greedy Search for Romania Traveling

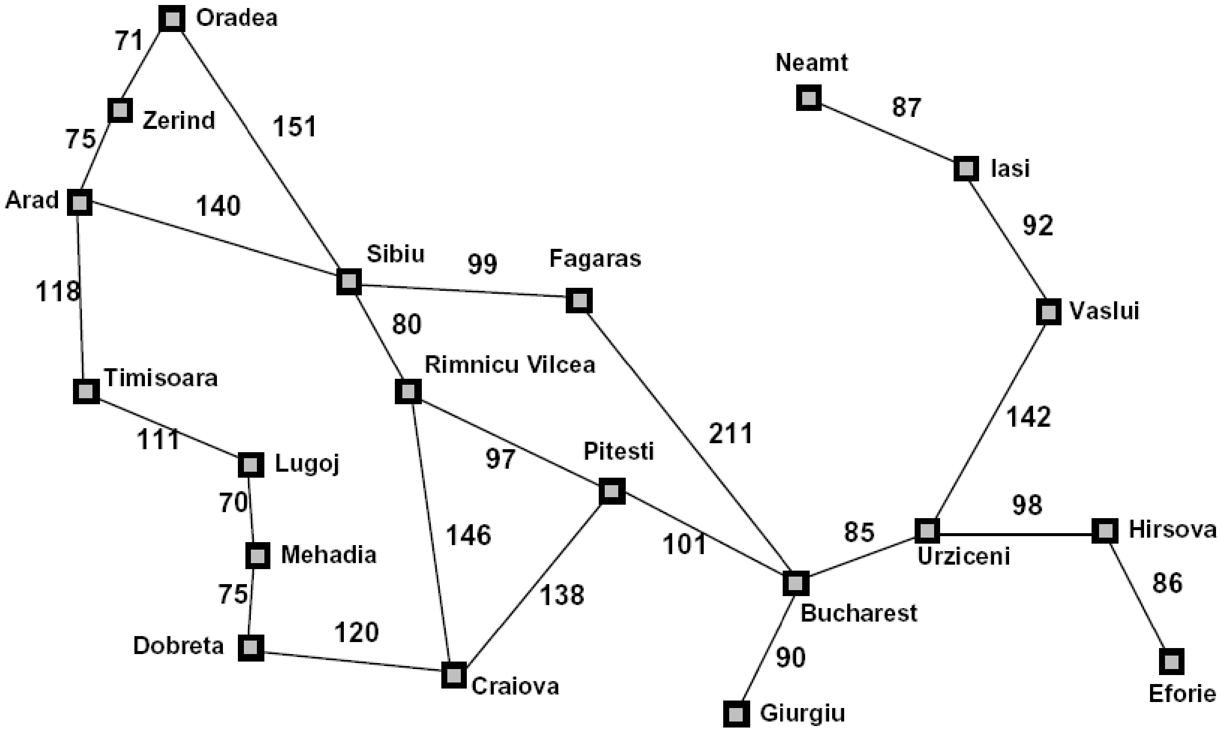
Fringe: *A data structure to store nodes observed but not visited*

Expand*: pop a node from the fringe*

Generate*: get neighbors of the expanded node and insert them into the fringe*

Strategy*: expand a node with lowest heuristic cost*

Implementation*: priority queue*

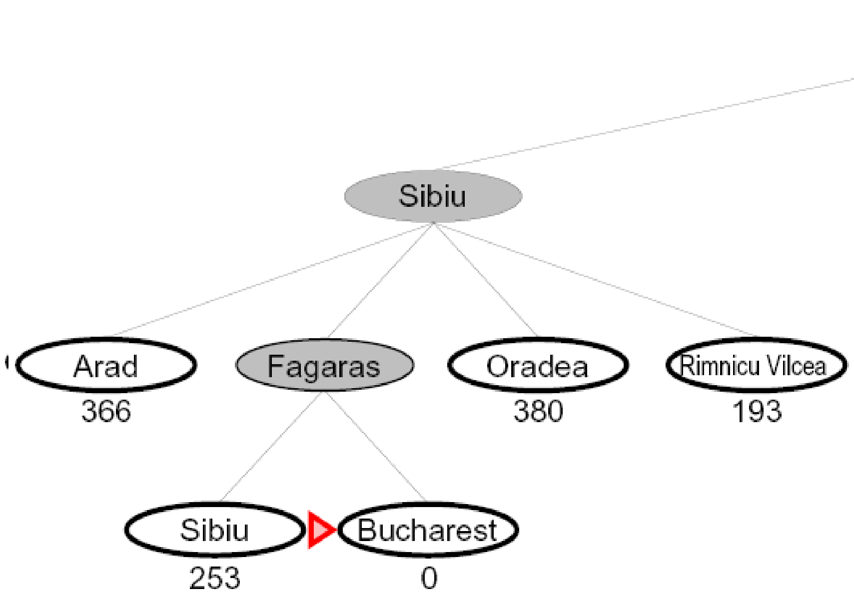


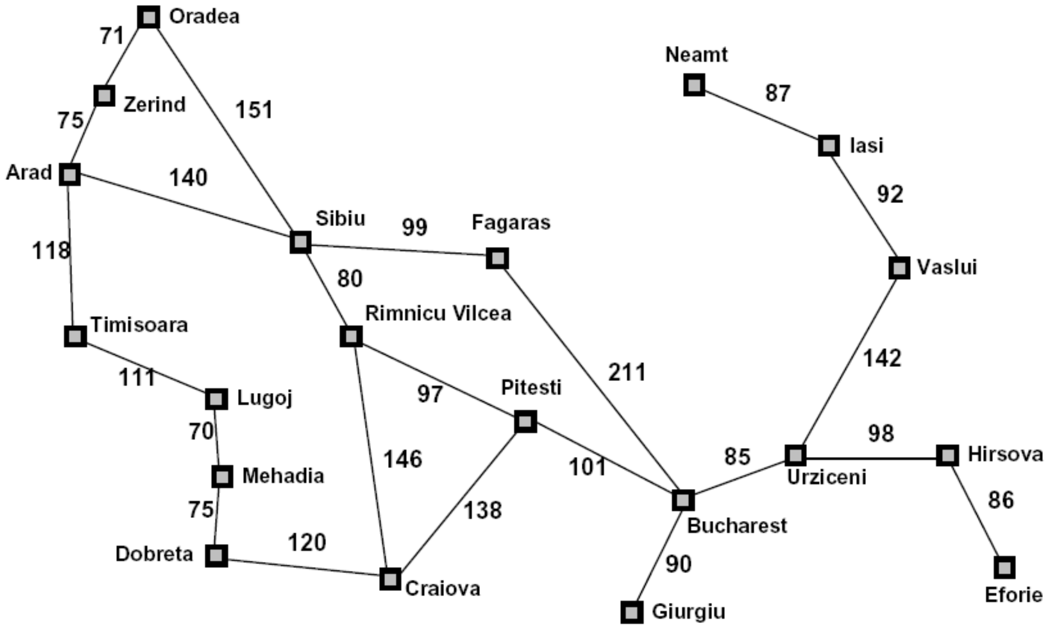
Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Tree Search for Romania Traveling

- Expand the node that seems closest (with lowest heuristic value)



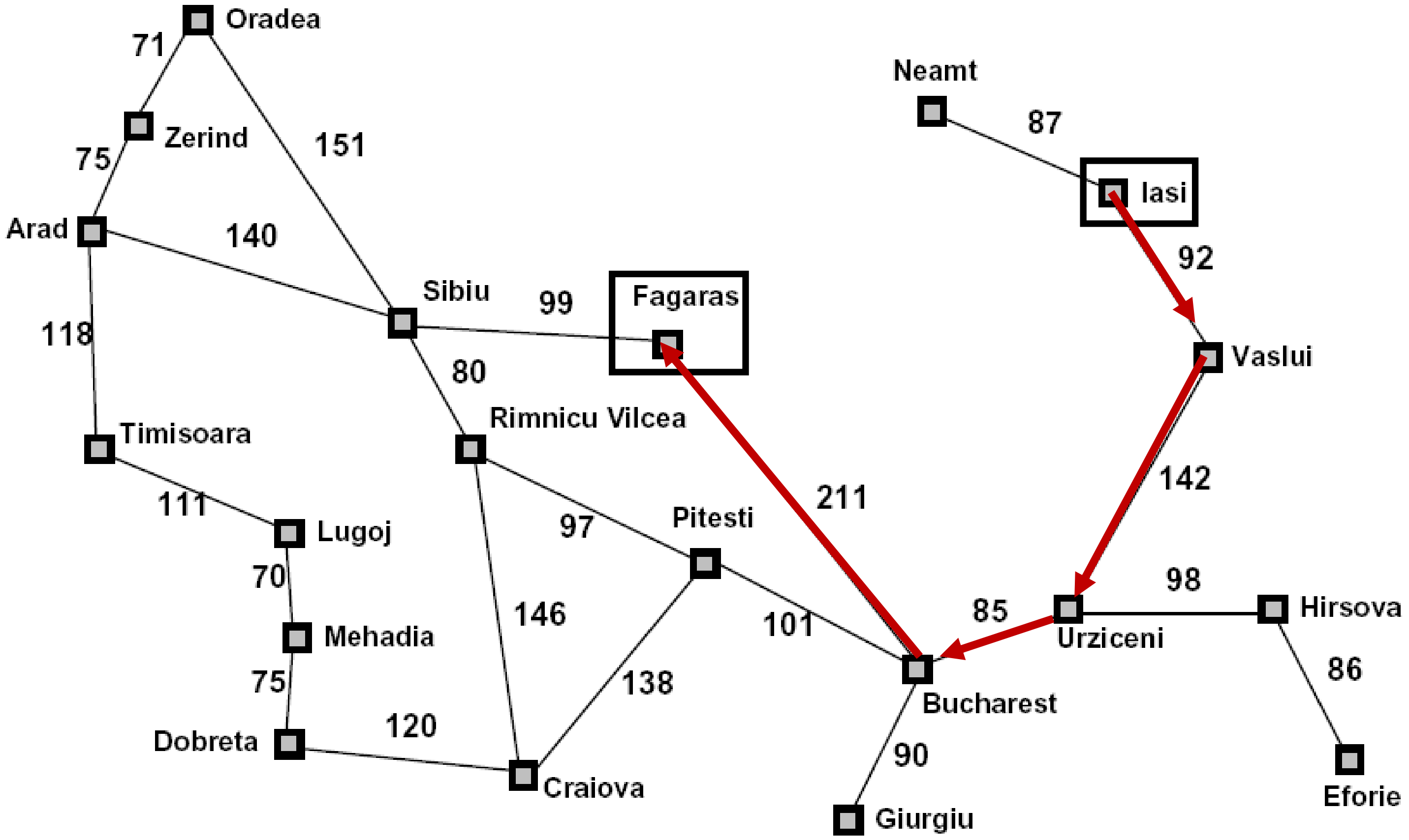


Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Tree Search

- What can go wrong?
  - From Lasi to Fagaras
  - Optimal solution: I→V→U→B→F

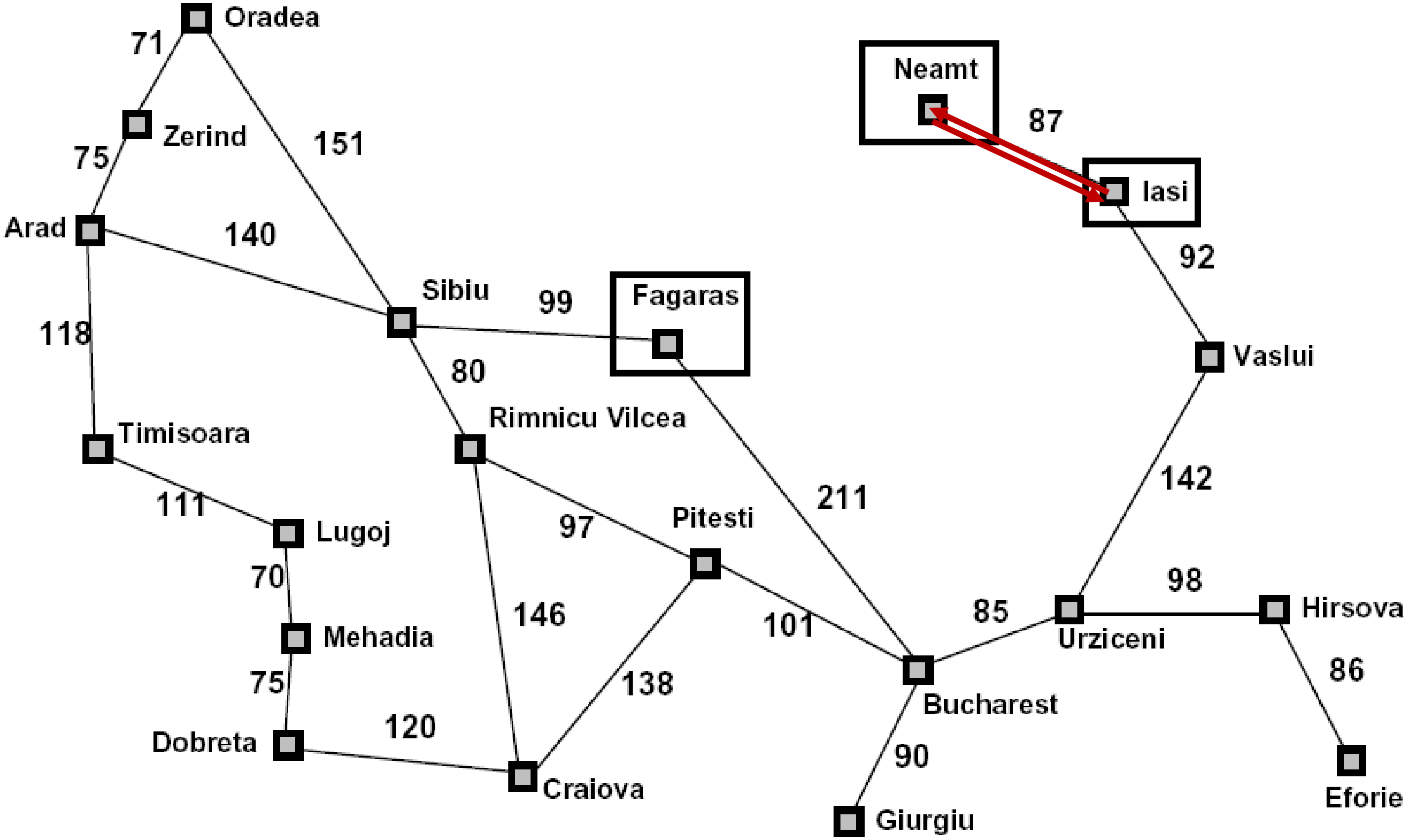


Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Tree Search

- What can go wrong?
  - Greedy Search does not care about the history cost at all.
  - Greedy solution: I→N→I→N→I→N ……

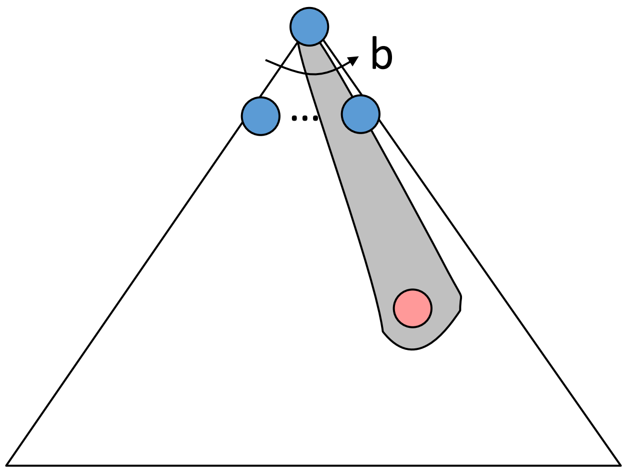


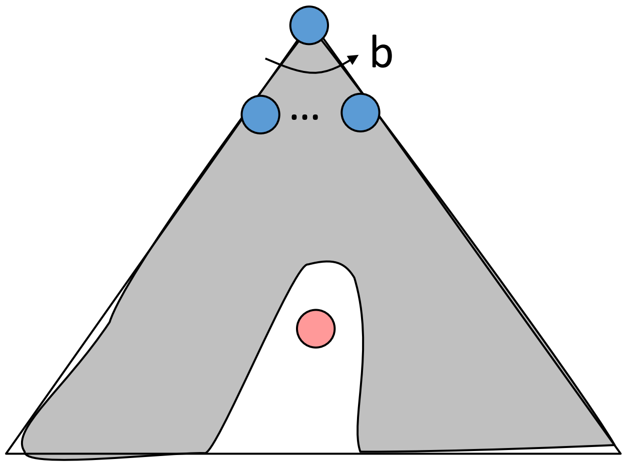
Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Search

- Strategy: expand a node that you think is closest to a goal state (lowest heuristic)
  - Heuristic: distance to nearest goal
  - Best-first takes you straight to the goal (can be wrong)





- Not Complete
- Not Optimal

Outline

- Heuristics Function
- Greedy Search
- A* Search

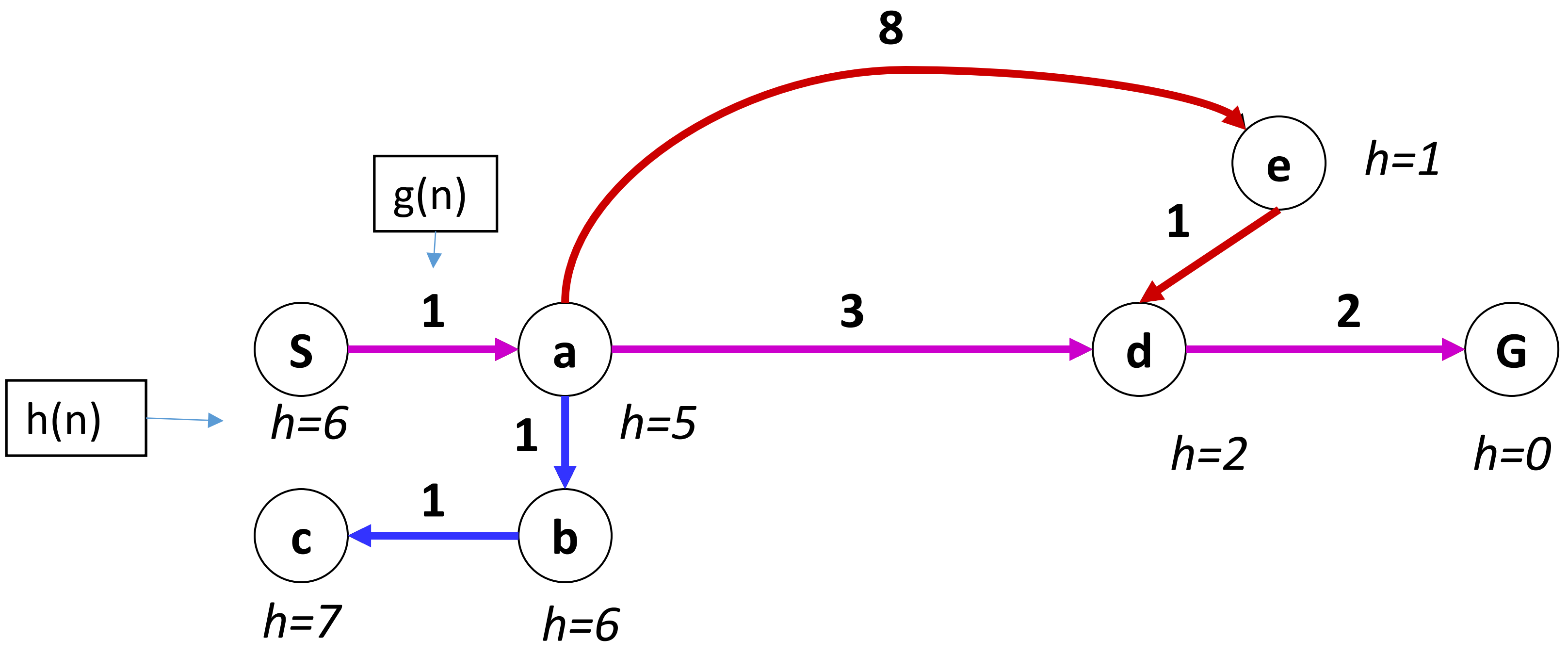
A* Search

- History：**cost from starting node**
- Future：**cost to the goal**
- A* considers both information of history and future.

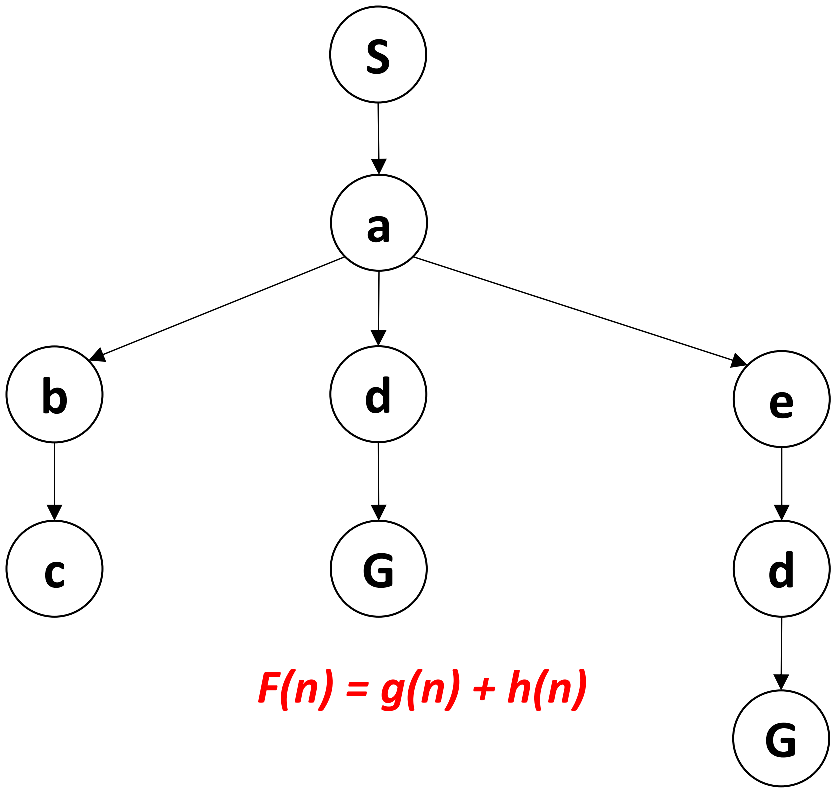
- Evaluation function f(n)
  - f(n) = g(n) + h(n)
  - g(n) is the cost of the path represented by node n (from root to n)
  - h(n) is the heuristic estimate of n (the cost of achieving the goal from n)

Example: USC vs Greedy vs A*

- Write down the search path for this problem and specify the path each algorithm chooses.
- Assume that: g(s) = 0 and h(G) = 0
- Uniform-cost orders by path cost, or *backward cost* g(n)
- Greedy orders by goal proximity, or *forward cost* h(n)
- A* orders by the sum: f(n) = g(n) + h(n). A Combination of USC and Greedy.

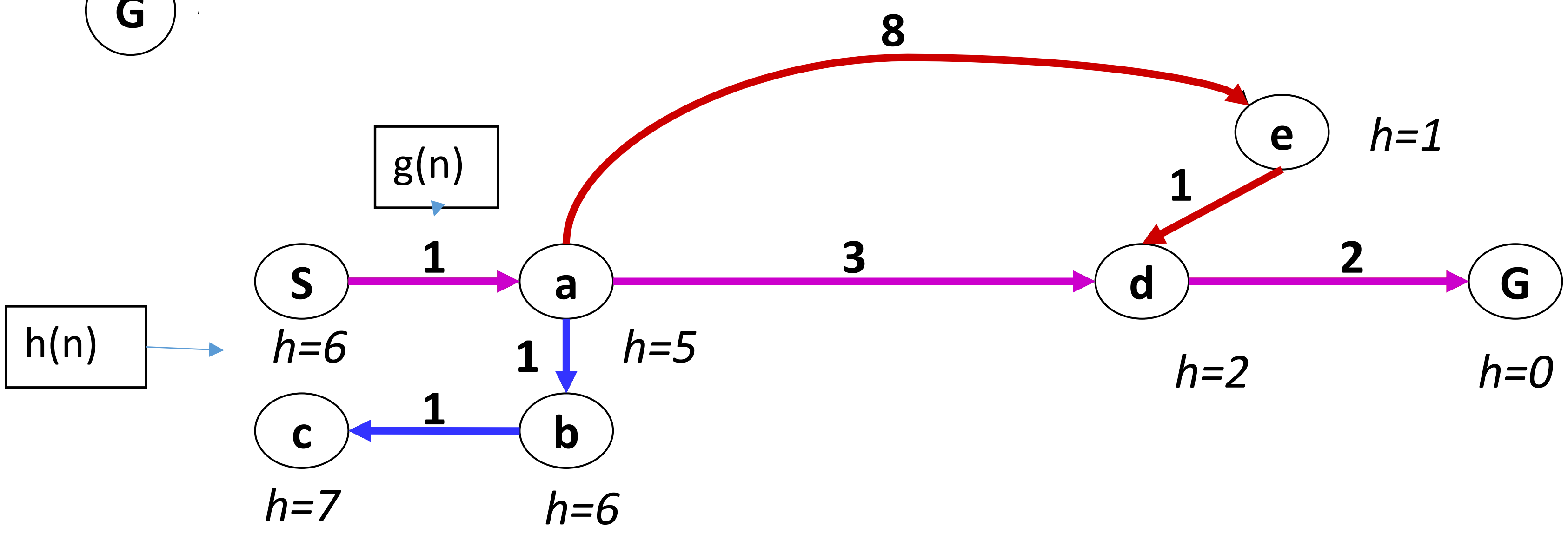


Example: USC vs Greedy vs A*

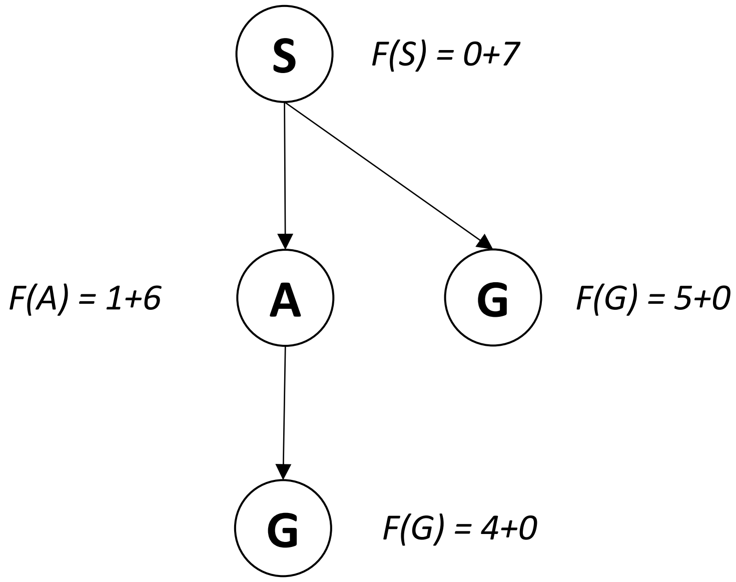


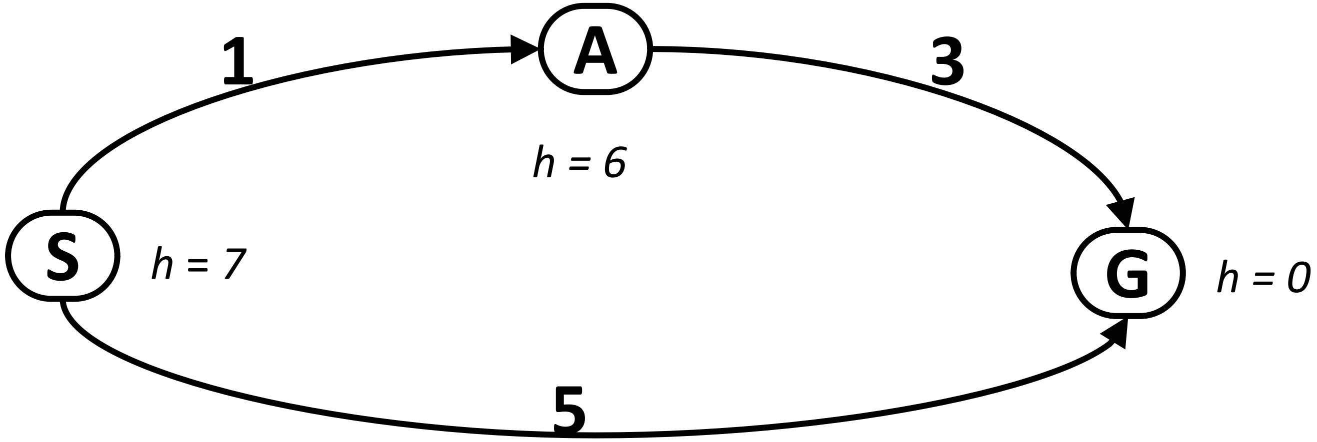
$F(n) = g(n) + h(n)$

Algorithm	Resulted path	Node visited
optimal	S → a → d→ G	S, a, d, G
UCS		
Greedy		
A*		



Is A* Optimal?





F(n) = g(n) + h(n)

Algorithm	Resulted path	Node visited
optimal	S → A → G	S, A, G
A*		
UCS		

- We need to pop node A before G to get the optimal solution.
- F(A) < F(G) → h(A) + g(A) < g(G) + h(G) → h(A) < g(G) – g(A)
- The estimated cost of nodes should be less than the real cost.

Outline

- Heuristics Function
- Greedy Search
- A* Search
- Admissible Heuristics Function

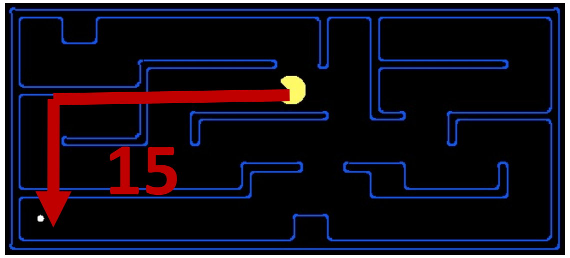
Admissible Heuristics

- 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:

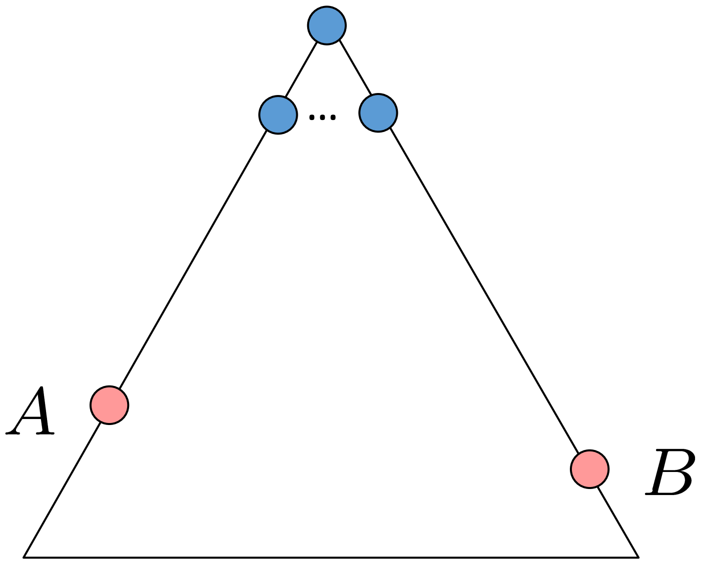


- Admissible heuristics is essential for using A* in practice.

Optimality of A* Tree Search

Assume:

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



Claim:

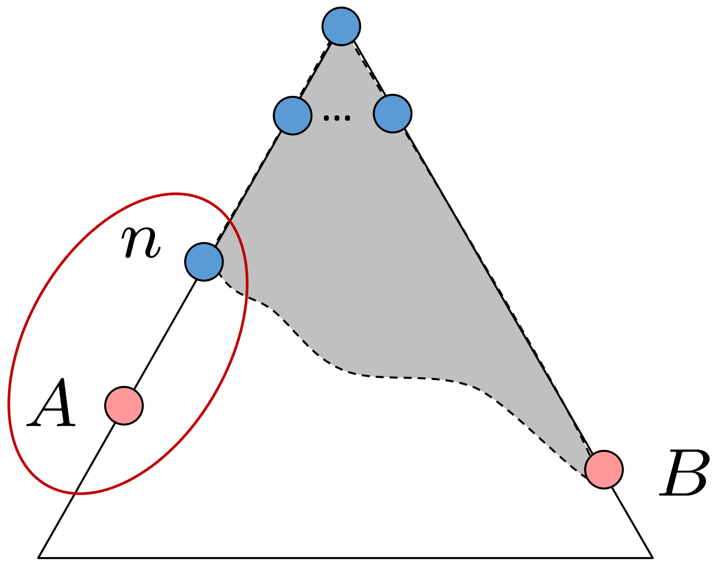
- A will be visited before B
- i.e. A is popped from fringe earlier than B

- Case 1: A and B are in the fringe together.
  - $f(A) = g(A) + h(A) = g(A) < g(B) = g(B) + h(B) = f(B)$
  - A will be visited earlier
- Case 2: B is inserted into the fringe before A
  - need to prove: all ancestors of A expand before B

Proof: Optimality of A* Tree Search

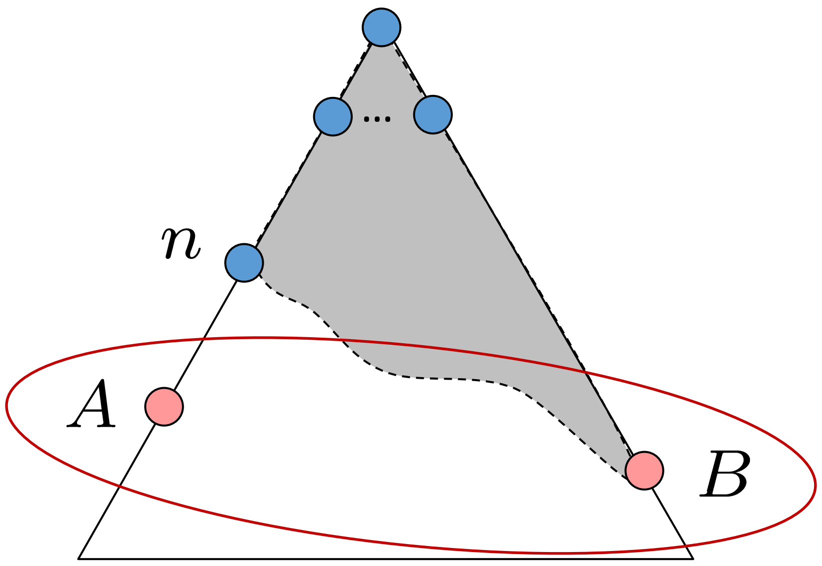
- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too
- Claim: *n* will be expanded before B
  - f(n) is less or equal to f(A)

$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	h = 0 at a goal



Proof: Optimality of A* Tree Search

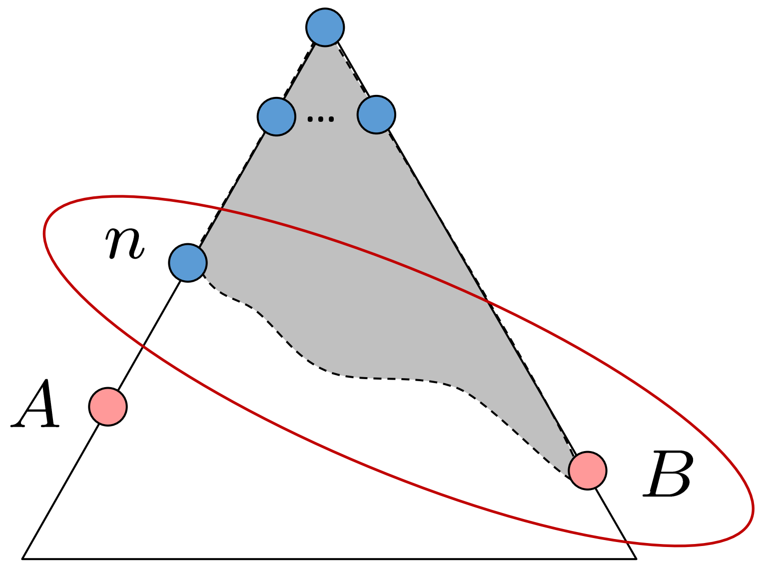
- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too
- Claim: *n* will be expanded before B
  - f(n) is less or equal to f(A)
  - f(A) is less than f(B)



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

Proof: Optimality of A* Tree Search

- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too
- Claim: *n* will be expanded before B
  - f(n) is less or equal to f(A)
  - f(A) is less than f(B)
  - *n* expands before B



- All ancestors of A expand before B
- A expands before B
- A* search is optimal

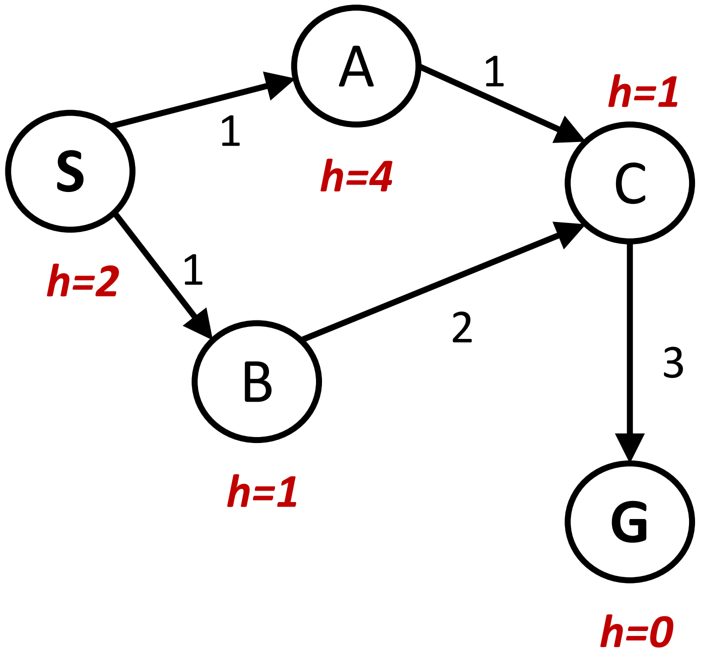
$$f(n) \leq f(A) < f(B)$$

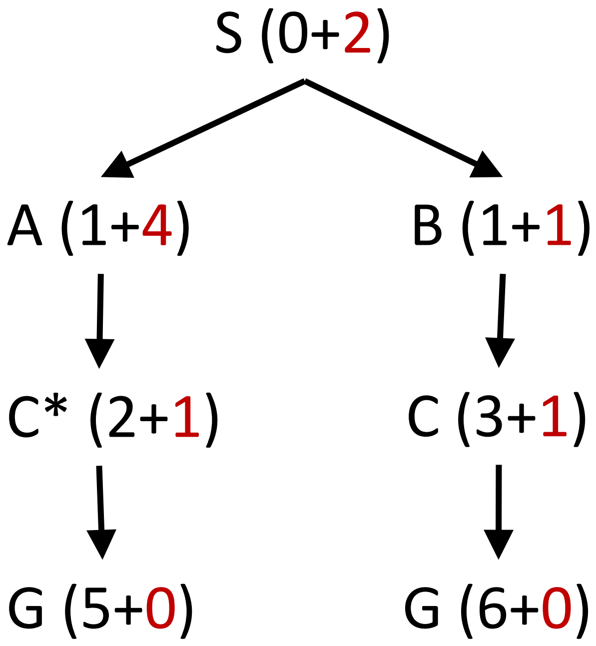
Outline

- Heuristics Function
- Greedy Search
- A* Search
- Admissible Heuristics Function
- Consistent Heuristics Function

A* Graph Search Gone Wrong with admissible function

Algorithm	Resulted path
optimal	
A* graph	





C need to be expanded optimally in the first visit.

← C* should be expanded before C

← ancestors of C* should be expanded before C.

f (A) < f(C), i.e., g(A) + h(A) < h(C) + g(C), i.e., h(A) – h(C) < g(C) – g(A)

The estimated cost of each edge should be less than the real cost.

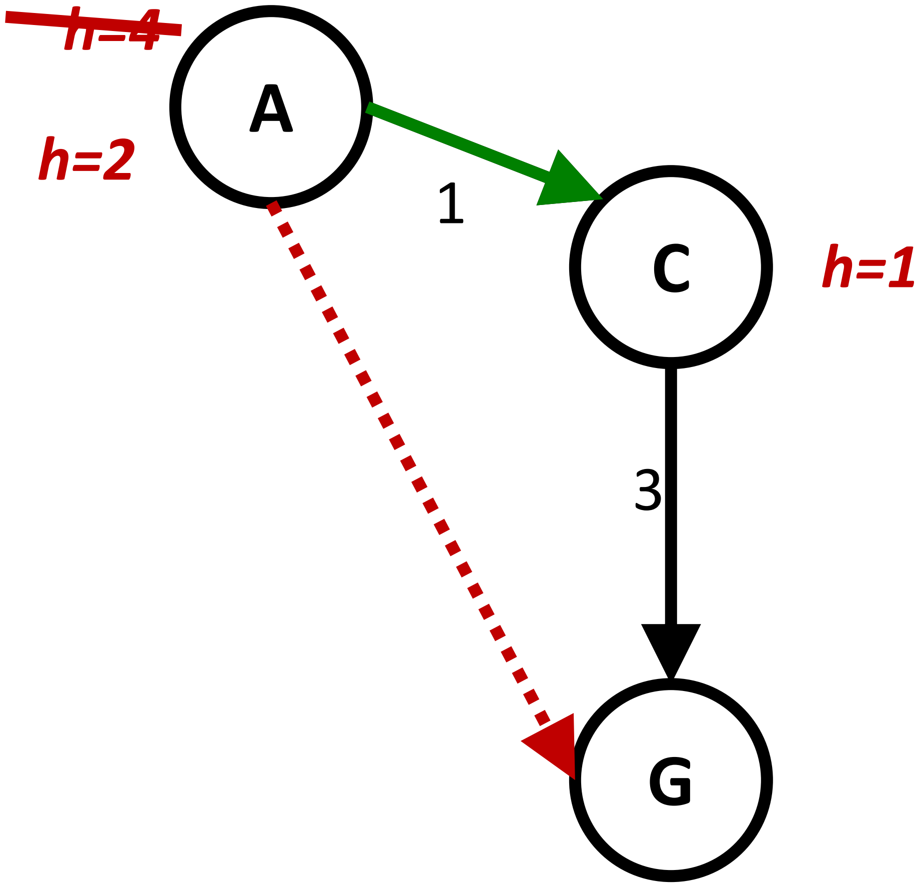
Consistency of Heuristics

- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

    h(A) ≤ actual cost from A to G

  - Consistency: heuristic "arc" cost ≤ actual cost for each arc

    h(A) – h(C) ≤ cost(A to C)



- Consequences of consistency:
  - The f value along a path never decreases

    h(A) ≤ cost(A to C) + h(C)

  - A* graph search is optimal

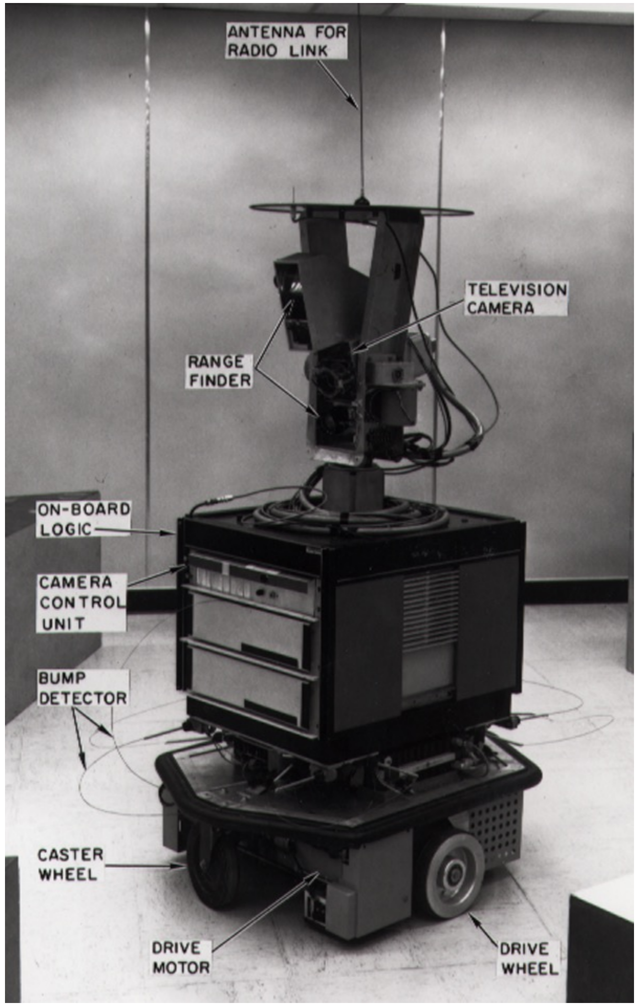
Optimality

- 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

A* History

- Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute (now SRI International) first described the algorithm in 1968.

- A1 -> A2 -> A*(Optimal)



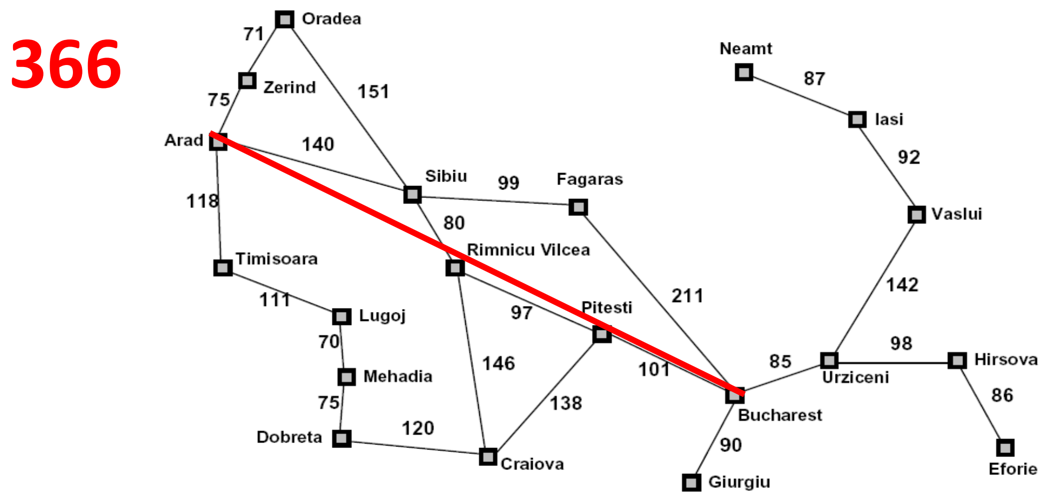
Outline

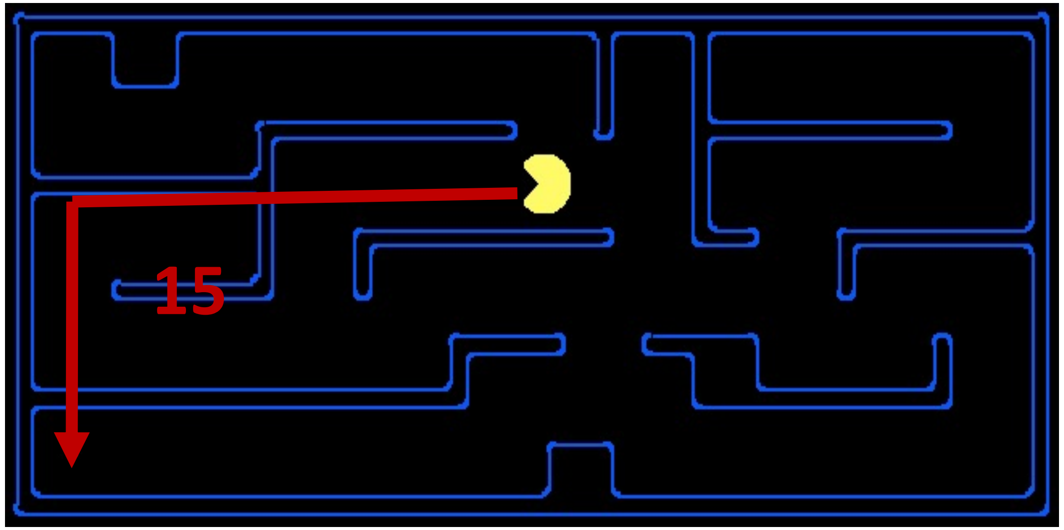
- Heuristics Function
- Greedy Search
- A* Search
- Admissible Heuristics Function
- Consistent Heuristics Function
- Creating Heuristics Function

Creating Admissible Heuristics

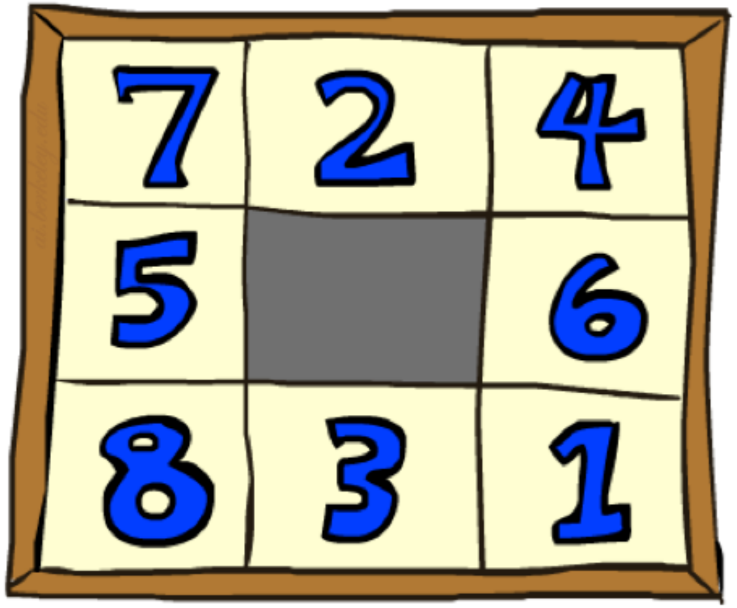
- Coming up admissible heuristics is crucial to use A*

- Find a relaxed problem, and solutions to that problem can be admissible heuristics.
- Relaxed problem has less or equal cost in every state compared to the original one.

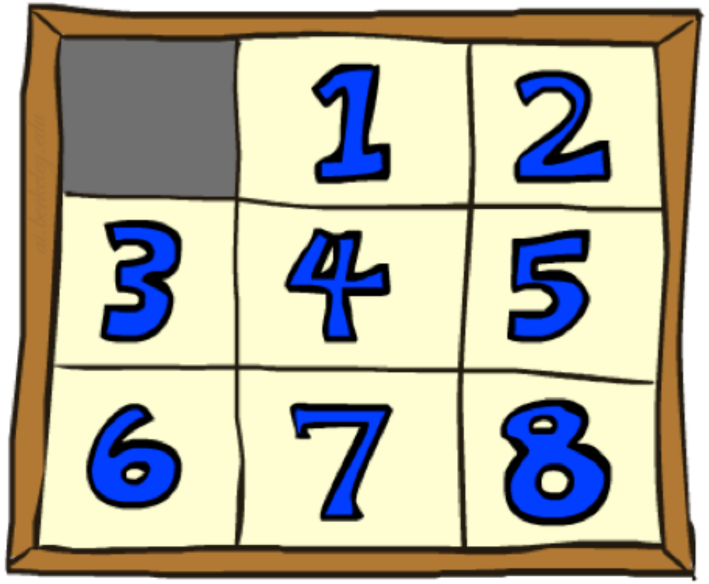




Example: 8 Puzzle



Start State



Goal State

From relaxed problem

A tile can move from square A to square B if **A is adjacent to B** and **B is blank**.

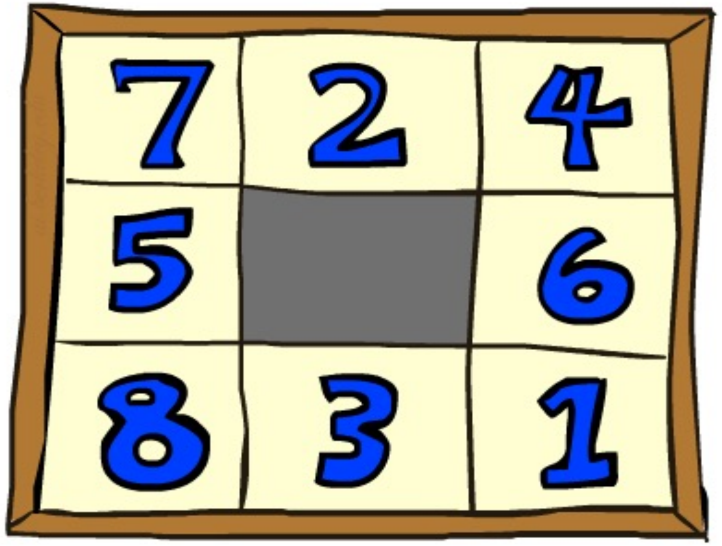
- Constraint 1: A and B is adjacent
- Constraint 2: B is blank

- Problem w/o C1: A tile can move from square A to square B if **B is blank**.
- Problem w/o C2: A tile can move from square A to square B if **A is adjacent to B**.
- Problem w/o C1 and C2: A tile can move from square A to square B.

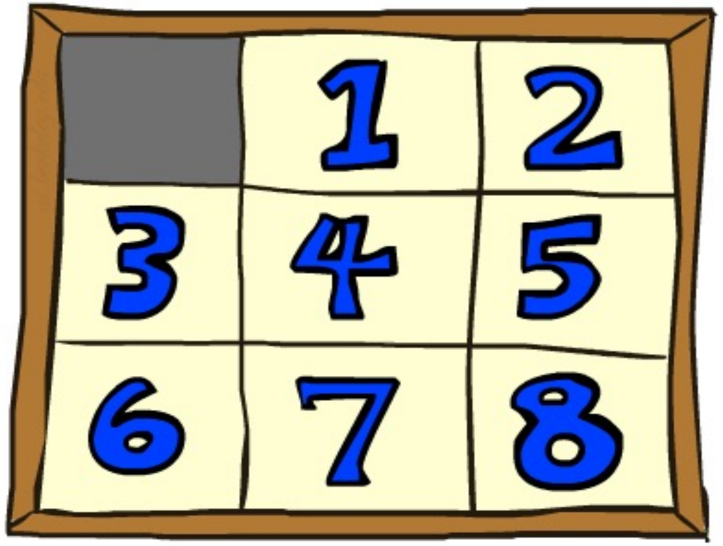
Relaxed problems can be identified automatically with formal expression of the original problem!

8 Puzzle I

- Heuristic: Number of tiles misplaced
- h (start) = 8
- Relaxed problem: a tile can move from square A to square B, and the cost of moving A to B always equals to one.



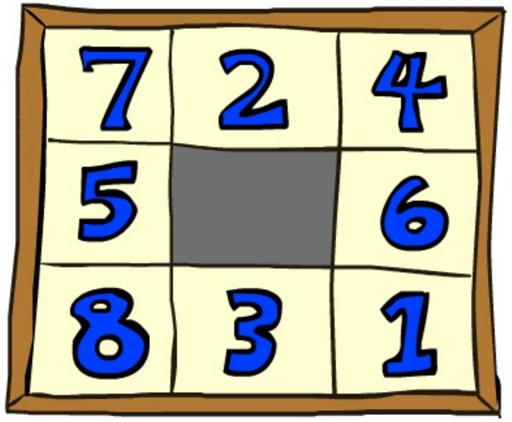
Start State



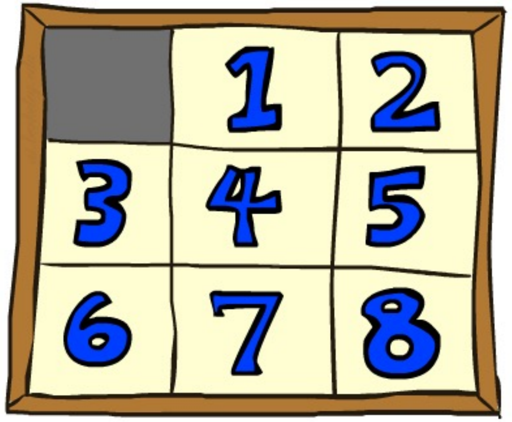
Goal State

8 Puzzle II

- Tiles can slide any direction at any time, ignoring other tiles?
- Heuristic: Total *Manhattan* distance
- h(start) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18
- Relaxed problem: A tile can move from square A to square B, and the cost of moving A to B is their Manhattan distance.



Start State

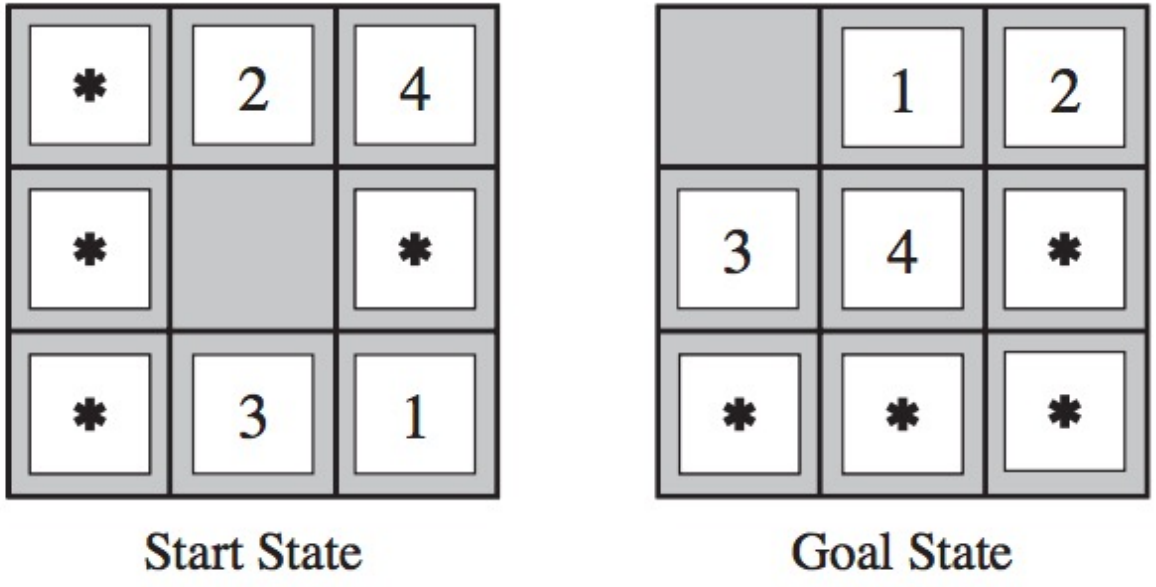


Goal State

From sub-problems

- Subproblem with less specified numbers:
  - Mask tiles 5, 6, 7, 8
  - The task is to get tiles 1, 2, 3, and 4 into their correct positions

- The cost of solving the sub-problem is no more than its original problem.



Learning for heuristic functions

- Learning heuristic functions

$$H(n) = c_1 x_1(n), \ldots, c_m x_m(n)$$

- How about optimality?
  - When it comes to learning-based approaches, optimality becomes complex.

Outline

- Heuristics Function
- Greedy Search
- A* Search
- Admissible Heuristics Function
- Consistent Heuristics Function
- Creating Heuristics Function
- Properties of Heuristics Function

Comparison of different heuristic functions

- Problem： 8 Puzzle
- IDS: Iterative Deepening Search (DFS + BFS)
- h1: # of tiles mis-placed
- h2: Total Manhattan distance

	Search Cost (nodes generated)			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

Figure 3.29 Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A^* algorithms with h_1, h_2. Data are averaged over 100 instances of the 8-puzzle for each of various solution lengths d.

- Effective Branching Factor (b^*) is computed based on the depth and # of nodes in the tree.

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

Heuristics of Dominance

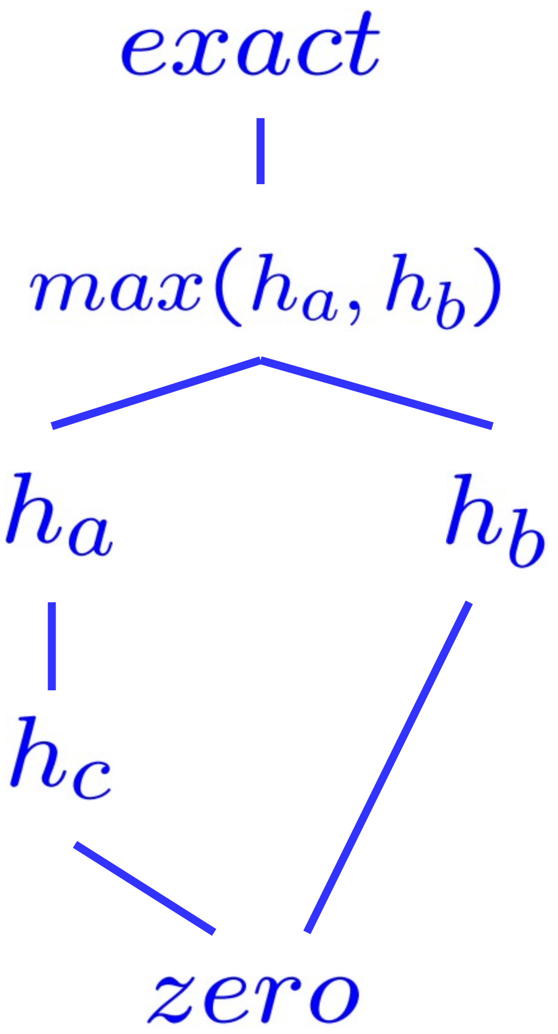
- Dominance: $h_a \geq h_c$ if

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

- 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
  - Top of lattice is the exact heuristic



Properties of Heuristics Function

- $h_1 \leq h_2$, heuristics function in dominance can generate better results.
- How about using the *actual cost* as a heuristic?
  - Would it be admissible? , Yes
  - Would we save on nodes expanded? , yes
  - What's wrong with it? More computational cost.
- 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

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