Artificial Intelligence

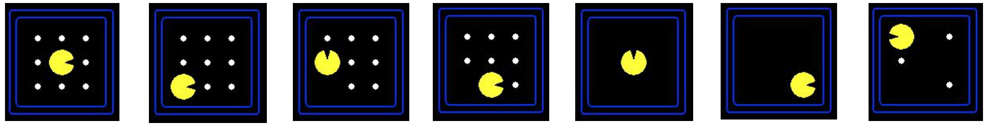
CS4365 --- Fall 2022

Informed Search

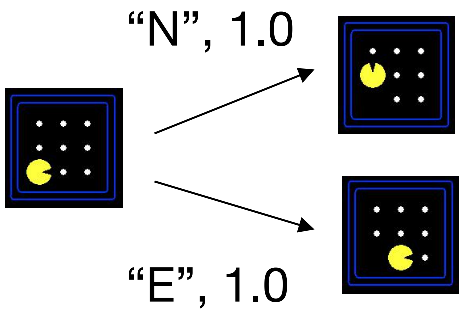
Instructor: Yunhui Guo

Define a Search Problem

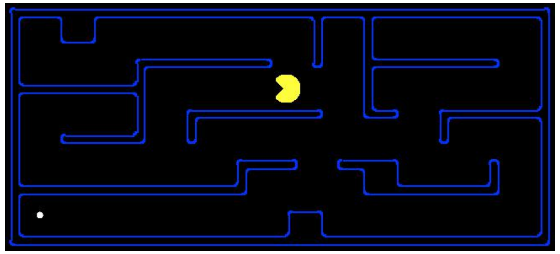
- A search problem consists of:
  - State space:



  - A successor function: (action + cost)



  - A start state and a goal test



Summary

	Complete?	Optimal?	Time	Space
BFS	Finte b	Yes	$O(b^d)$	$O(b^d)$
DFS	Finte d	No	$O(b^m)$	O(bm)
UCS	Finite b , step cost $\geq \epsilon$	Yes	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$
IDS	Finte b	Yes	$O(b^d)$	O(bd)

b: branching factor d: depth of the shallowest solution m: maximum depth

Limitations

- What are the problems of all the methods? Slow!
- The search is blind in the sense that the information of the goal state is not used
- Informed search:
  - with the guidance of the goal state

Informed Methods: Heuristic Search

- Informed methods use problem-specific knowledge
  - The location of the goal
- Humans rely on informed search!

Informed Methods: Heuristic Search

- We want to have some estimate of the distance from the states in the frontier to the goal state.

- Why estimate? Because the states we can reach are based on the actions we can take.

Informed Methods: Heuristic Search

- We use an evaluation function f(n) as our estimate
- Best-first search:
  - Nodes are selected for expansion based on the evaluation function, f(n).
  - Expand the node with the lowest evaluation
  - The evaluation function can be complex: $f(n) = f_1(n) + f_2(n) + \ldots$

Greedy Best-First Search

- One natural component of f(n):
  - Heuristic function:
    - h(n) = estimated cost of the cheapest path from the state at node n to a goal state

Heuristic search is an attempt to search the most promising paths first

Greedy Best-First Search

- Greedy Best-First Search
  - Expands the node that is “closest” to the goal as measured by h(n)

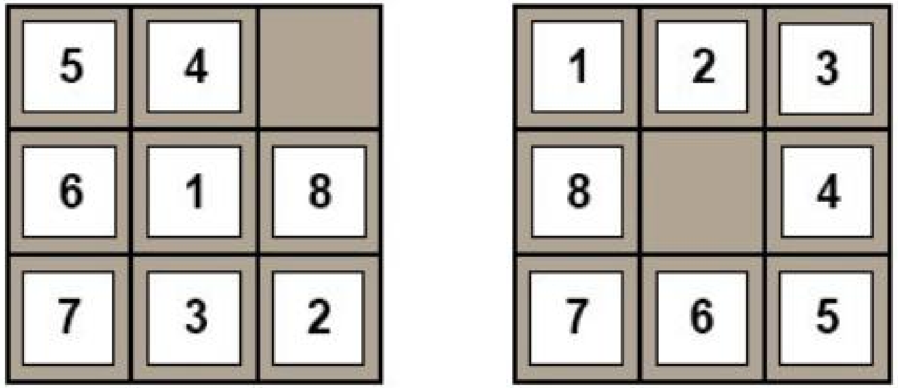
- A common case:
  - Best-first takes you straight to the (wrong) goal

- Worst-case: like a badly-guided DFS

Example: 8-puzzle problem

- Design of heuristic function is important

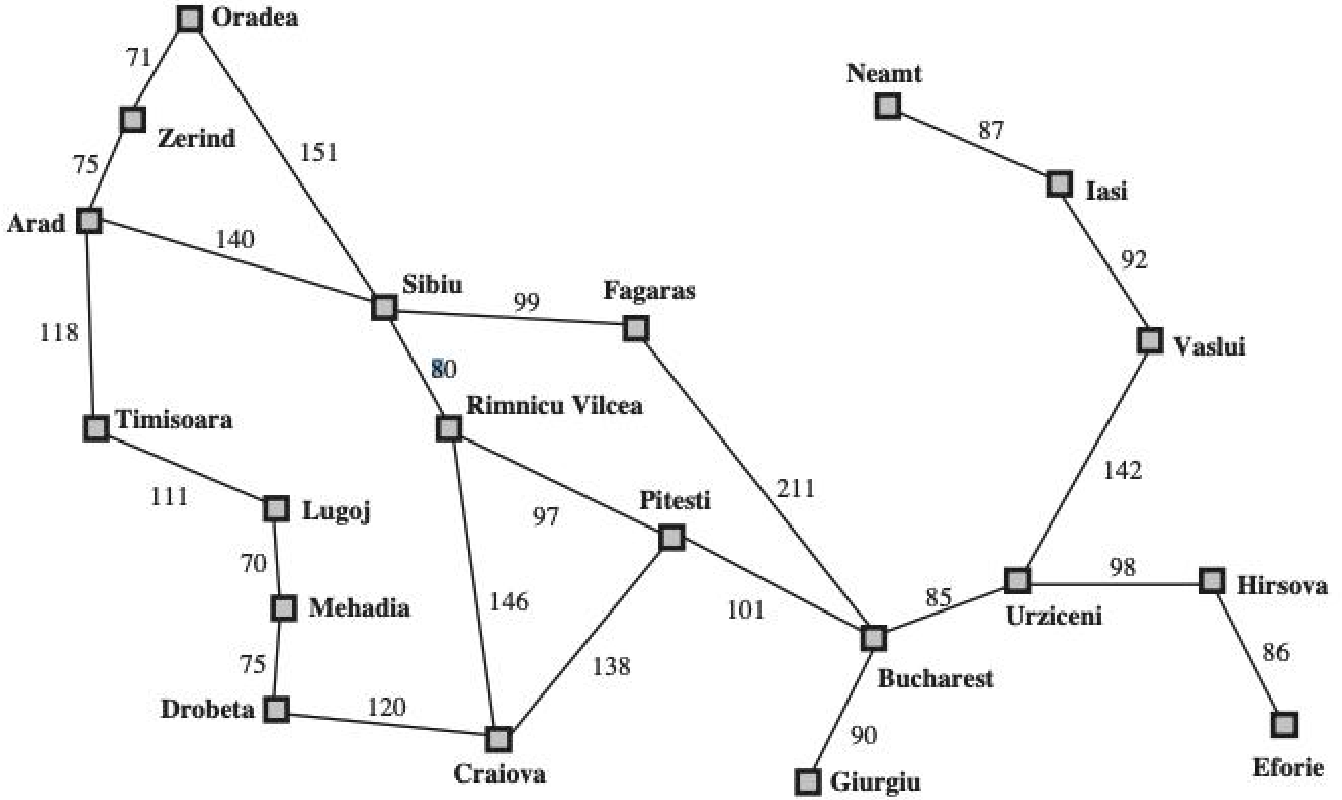
- One possible heuristic function:
    - The number of tiles misplaced



Start State Goal State

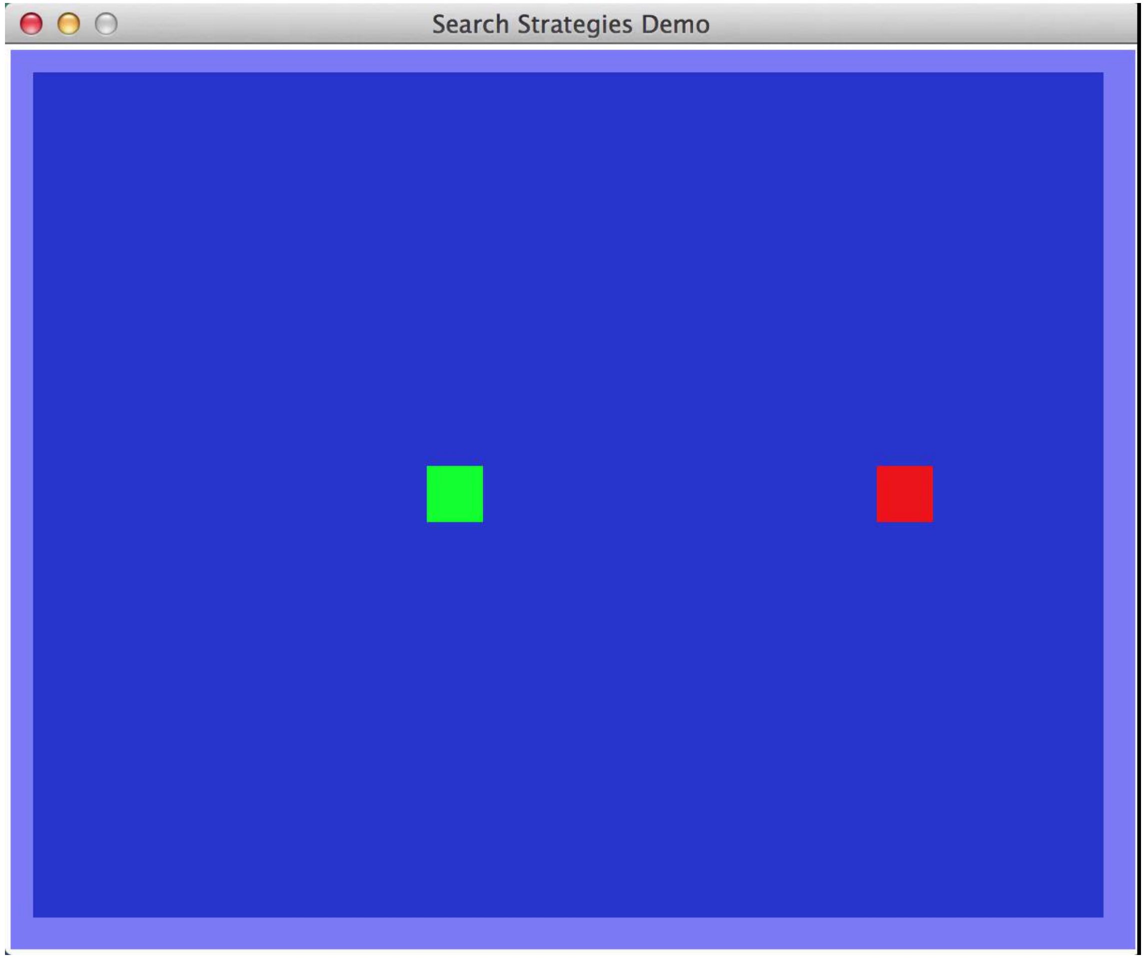
Example: Find a Path from Arad to Bucharest

- One possible heuristic function:
  - the straight-line distance to Bucharest



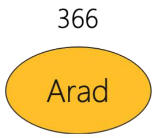
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Greedy Best-First Search



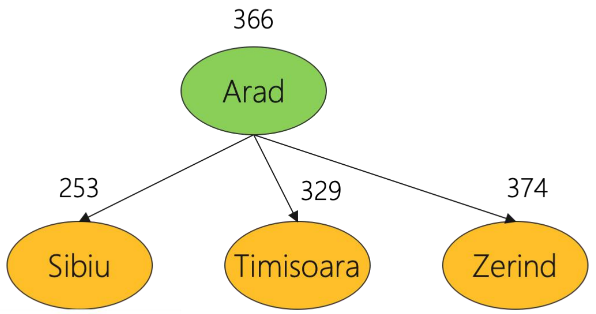
Example: Find a Path from Arad to Bucharest

- One possible heuristic function:
  - the straight-line distance to Bucharest
- Expand the node seems "closest"



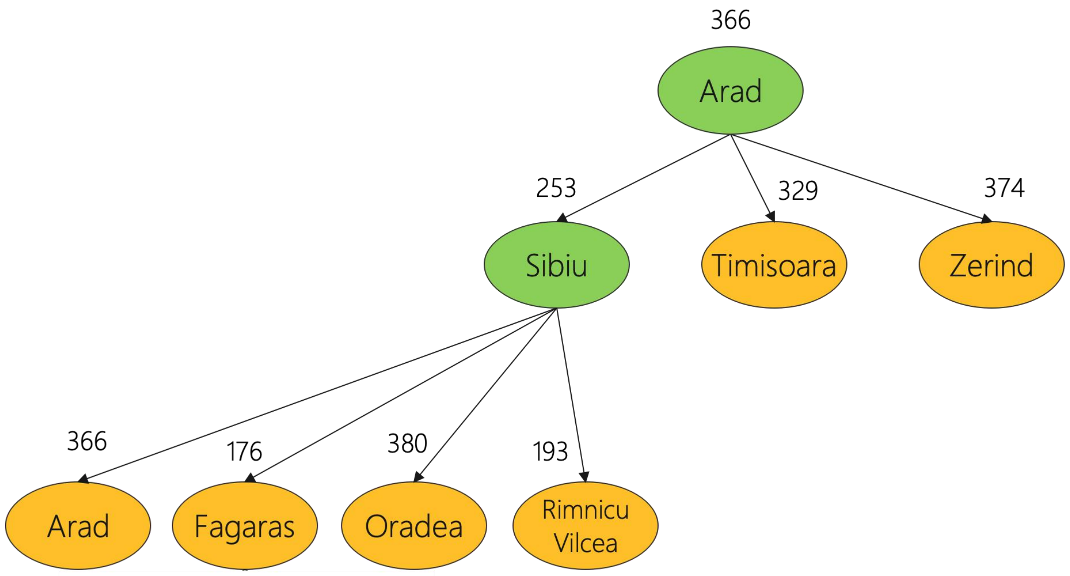
Example: Find a Path from Arad to Bucharest

- One possible heuristic function:
  - the straight-line distance to Bucharest
- Expand the node seems “closest”



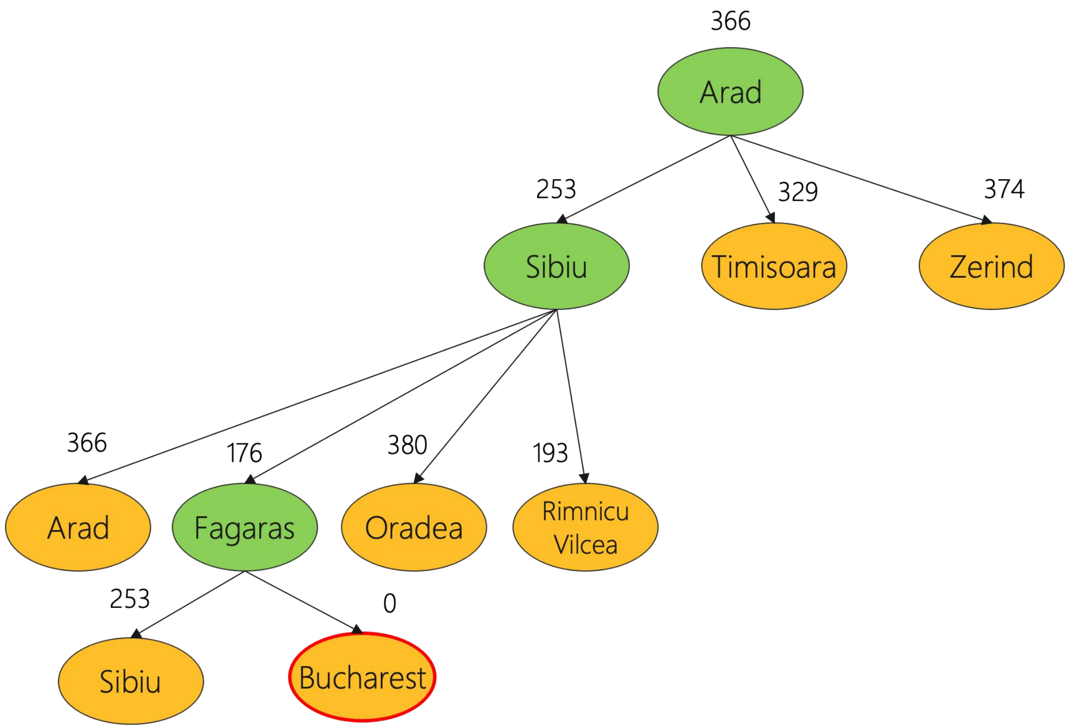
Example: Find a Path from Arad to Bucharest

- One possible heuristic function:
  - the straight-line distance to Bucharest
- Expand the node seems "closest"



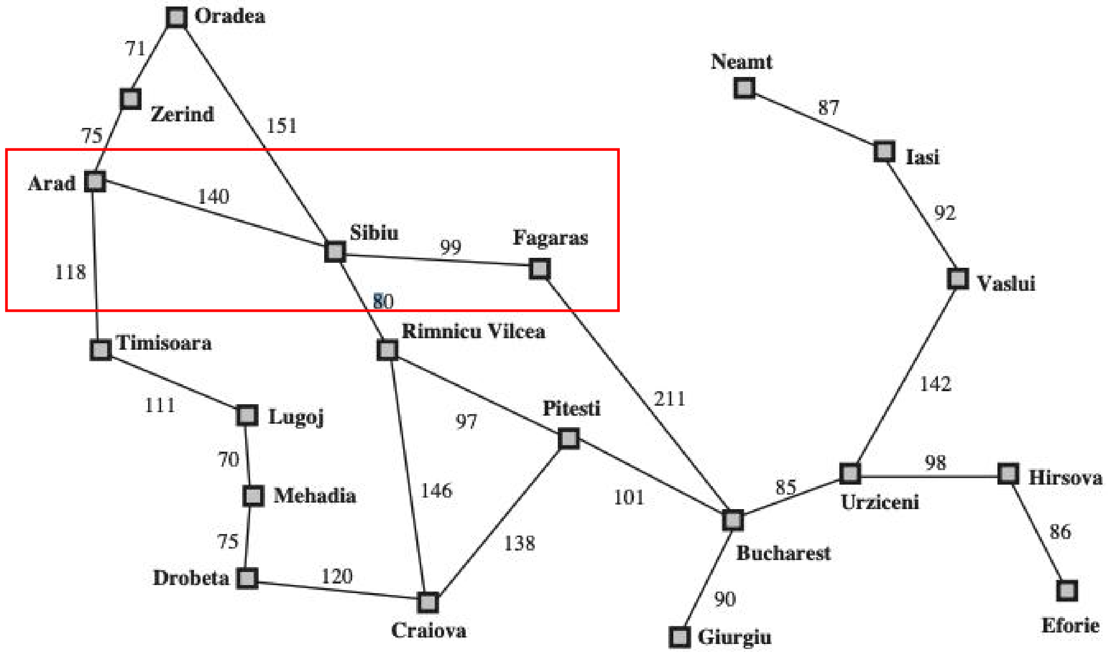
Example: Find a Path from Arad to Bucharest

- One possible heuristic function:
  - the straight-line distance to Bucharest
- Expand the node seems “closest”



Greedy Best-First Search Can be Suboptimal

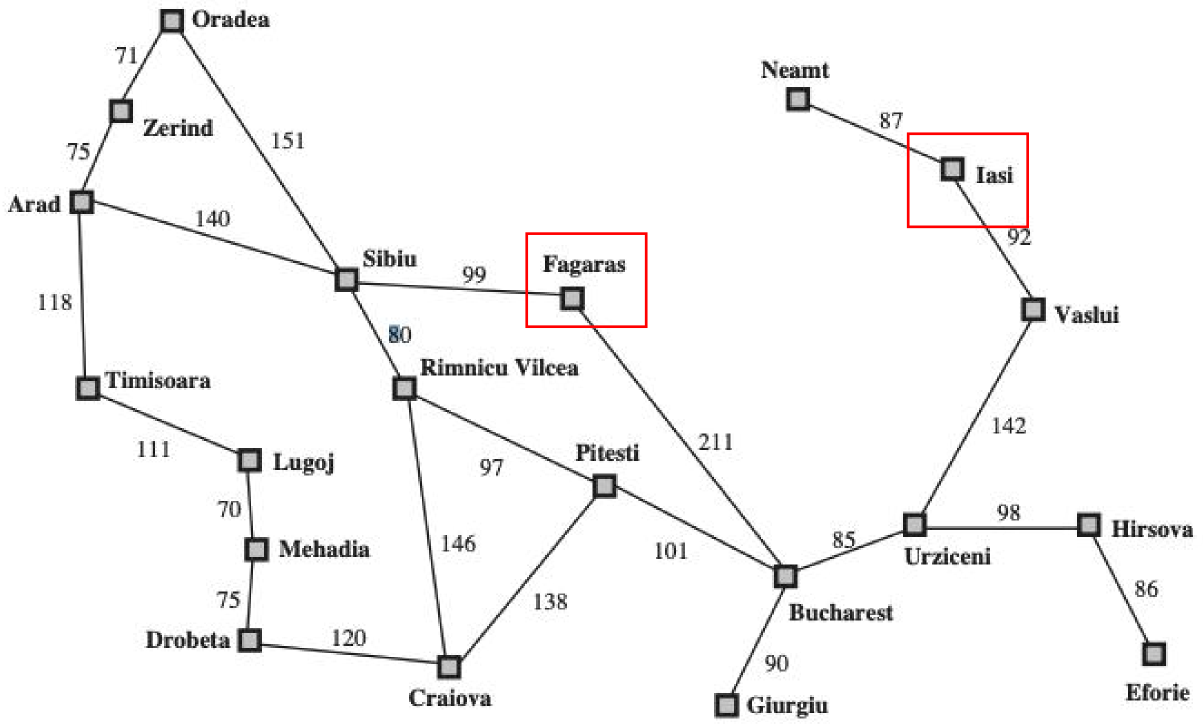
- From Arad to Sibiu to Fagaras -- but to Rimnicu would have been better



- What is missing?
    - The cost of getting from the start node (Arad) to intermidiate nodes!

Greedy Best-First Search is Incomplete

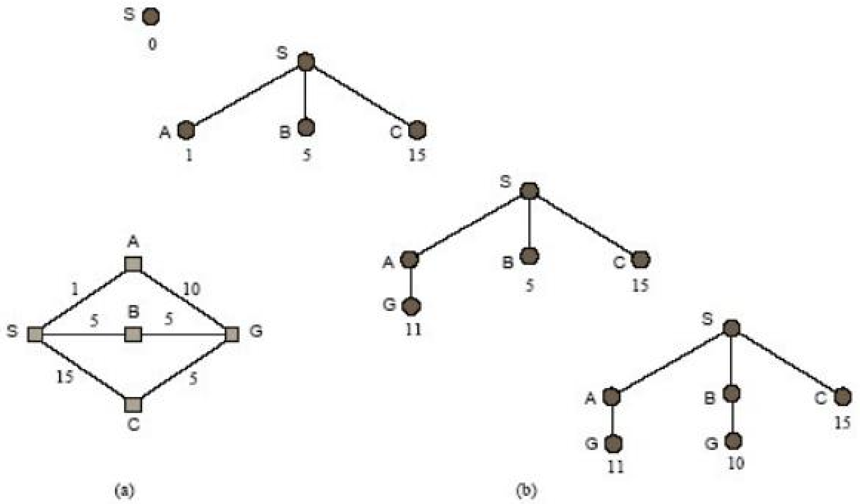
- Start state: Iasi
- Goal state: Fagaras



A* Search

- Proposed in 1968 by Peter Hart, Nils Nilsson and Bertram Raphael
- Most widely known form of Best-First Search.
- Combining Uniform-Cost Search and Greedy Best-First Search

UCS



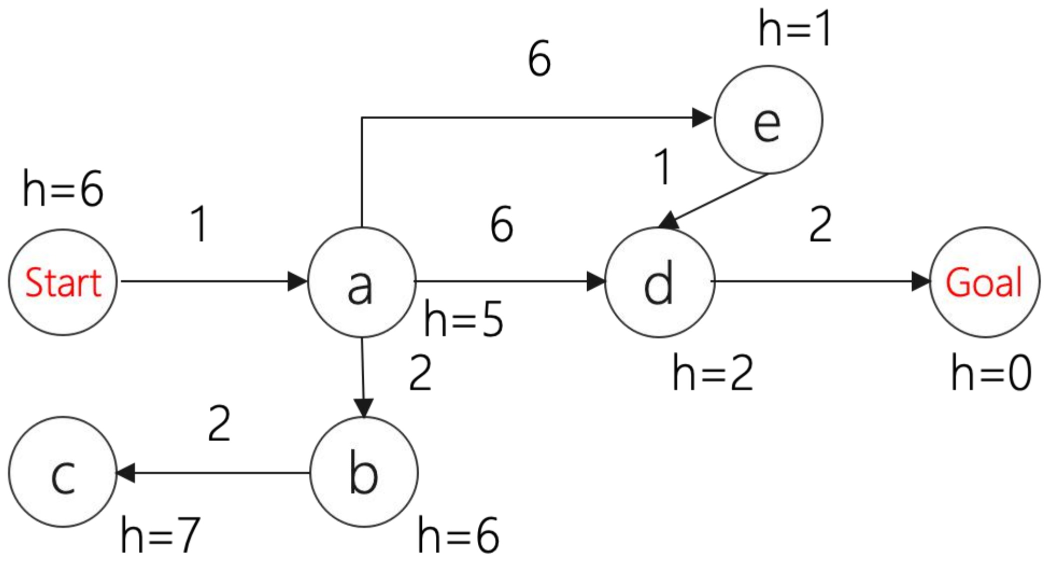
A* Search

- Combining Uniform-Cost Search and Greedy Best-First Search

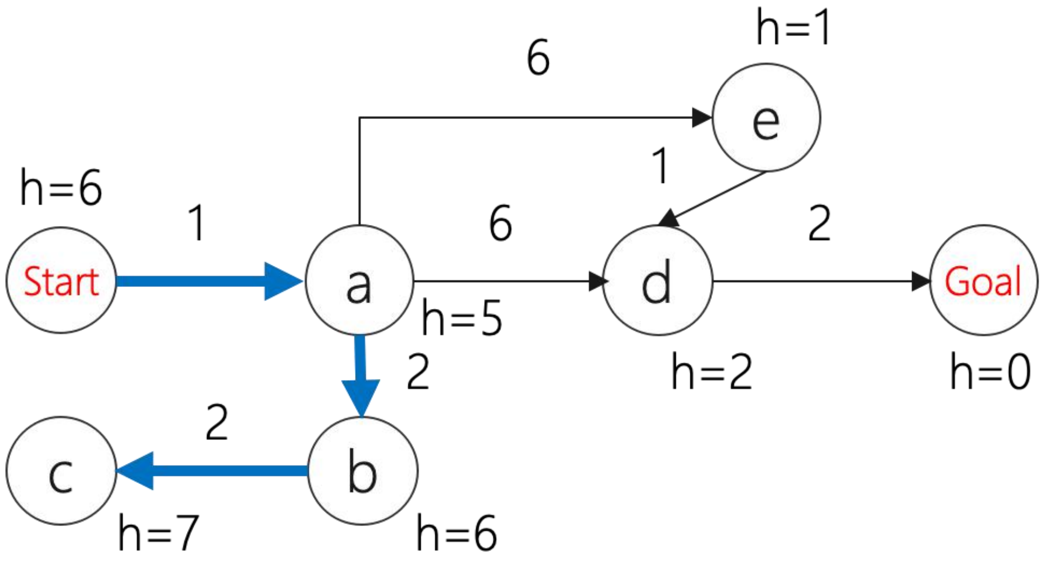
- $f(n) = g(n) + h(n)$
  - $g(n)$: the path cost from the start node to node n
  - $h(n)$: the estimated cost of the cheapest path from node n to the goal node

- When $h(n)$ satisfies certain properties, A* is both complete and optimal!

A* Search

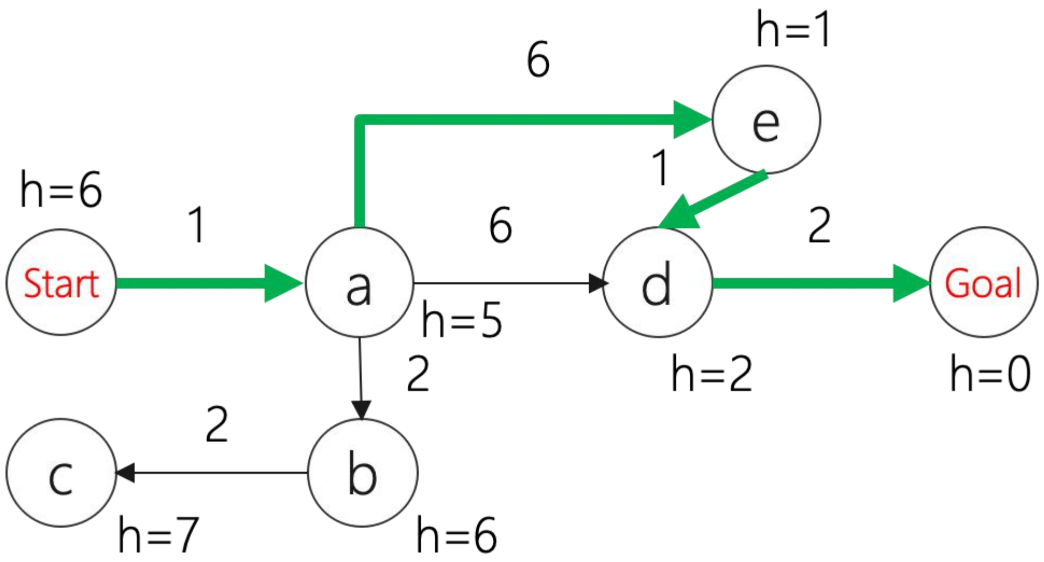


A* Search



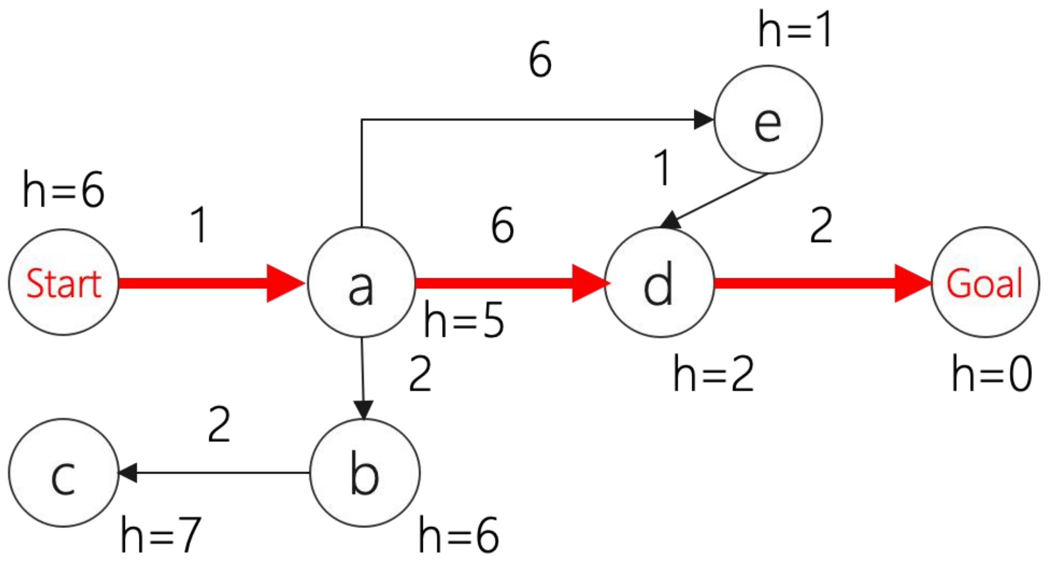
- Uniform-cost orders by path cost g(n)

A* Search



- Uniform-cost orders by path cost g(n)
- Greedy best first search orders by estimated goal proximity h(n)

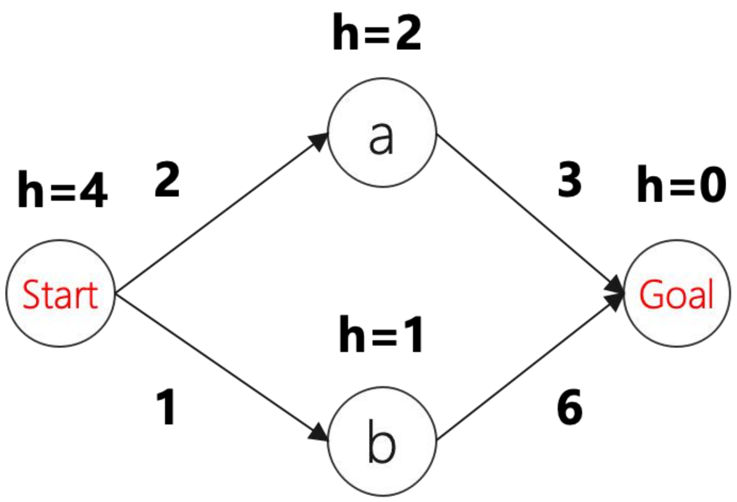
A* Search



- Uniform-cost orders by path cost g(n)
- Greedy best first search orders by estimated goal proximity h(n)
- A* search combines g(n) and h(n)

When to terminate in A* Search

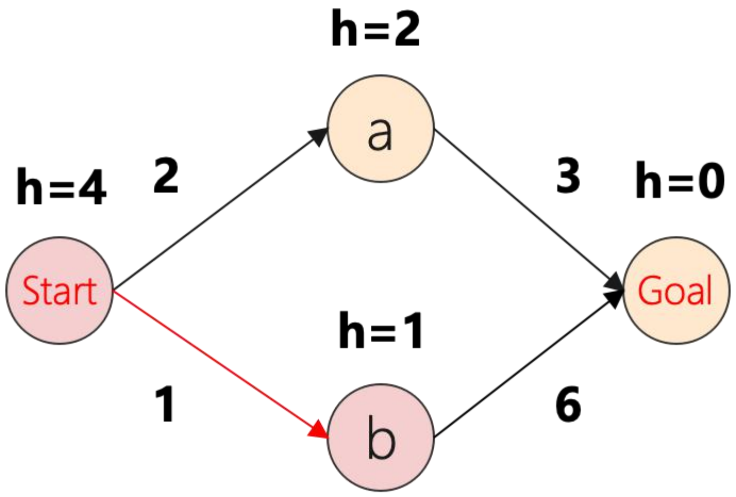
- Similar to Uniform Cost Search, the goal test is applied to a node when it is selected for expansion,



- The path cost to the goal state may get updated.

When to terminate in A* Search

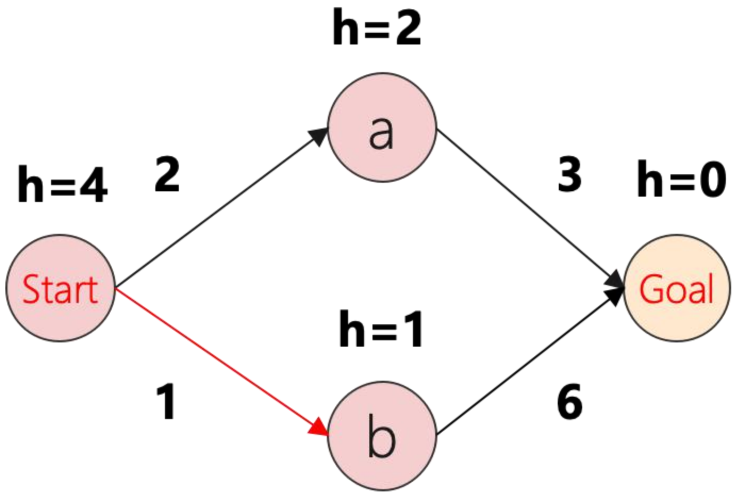
- Similar to Uniform Cost Search, the goal test is applied to a node when it is selected for expansion,



- The path cost to the goal state may get updated.

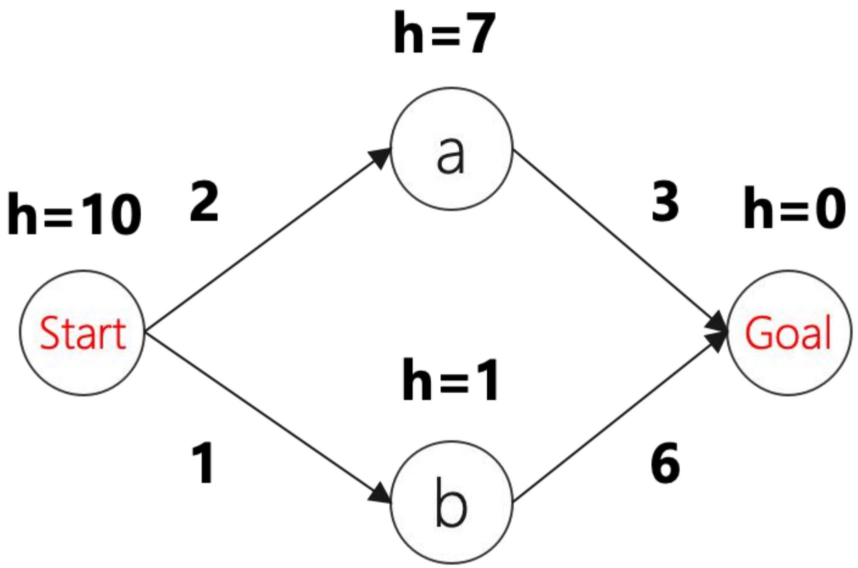
When to terminate in A* Search

- Similar to Uniform Cost Search, the goal test is applied to a node when it is selected for expansion,

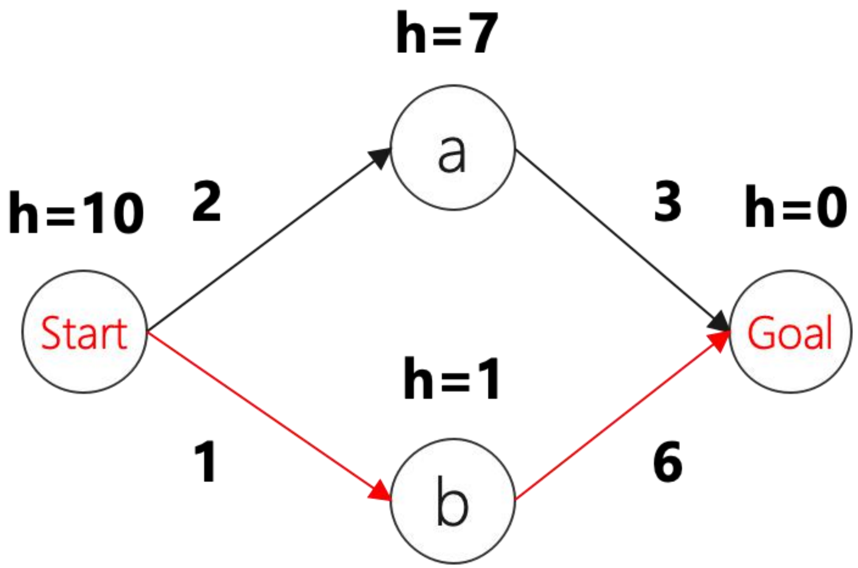


- The path cost to the goal state may get updated.

Is A* Search Optimal?



Is A* Search Optimal?



- What went wrong?
- Actual goal cost < estimated goal cost
- Solution?

Need Some Conditions

- To guarantee that A* finds an optimal solution, we need that h(n) never overestimates the cost of reaching the goal

- Called an admissible heuristic

- Transfer to f, i.e., f also doesn't overestimate.

Formal Definition of Admissibility

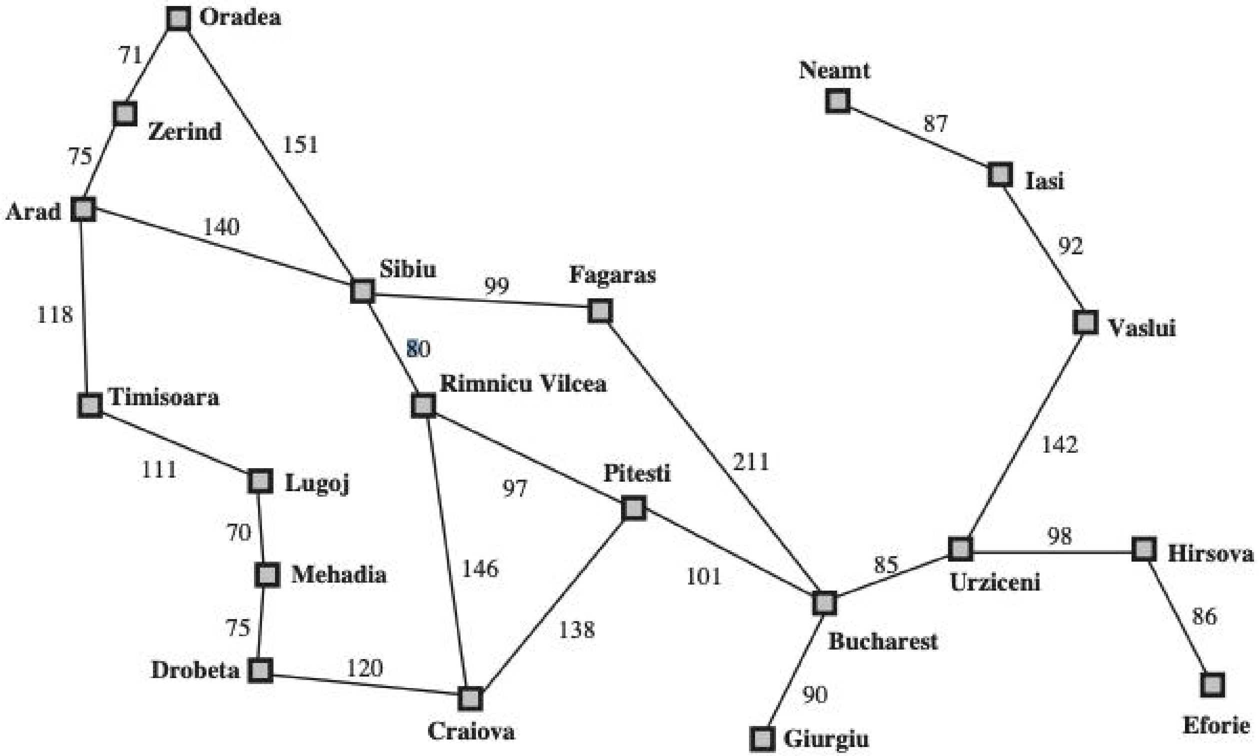
- Let $h^*(n)$ be the actual cost to reach a goal from n.

- A heuristic function h is optimistic or admissible if $0 \leq h(n) \leq h^*(n)$ all nodes n.

- If h is admissible, then the A* tree search will never return a sub-optimal goal node.

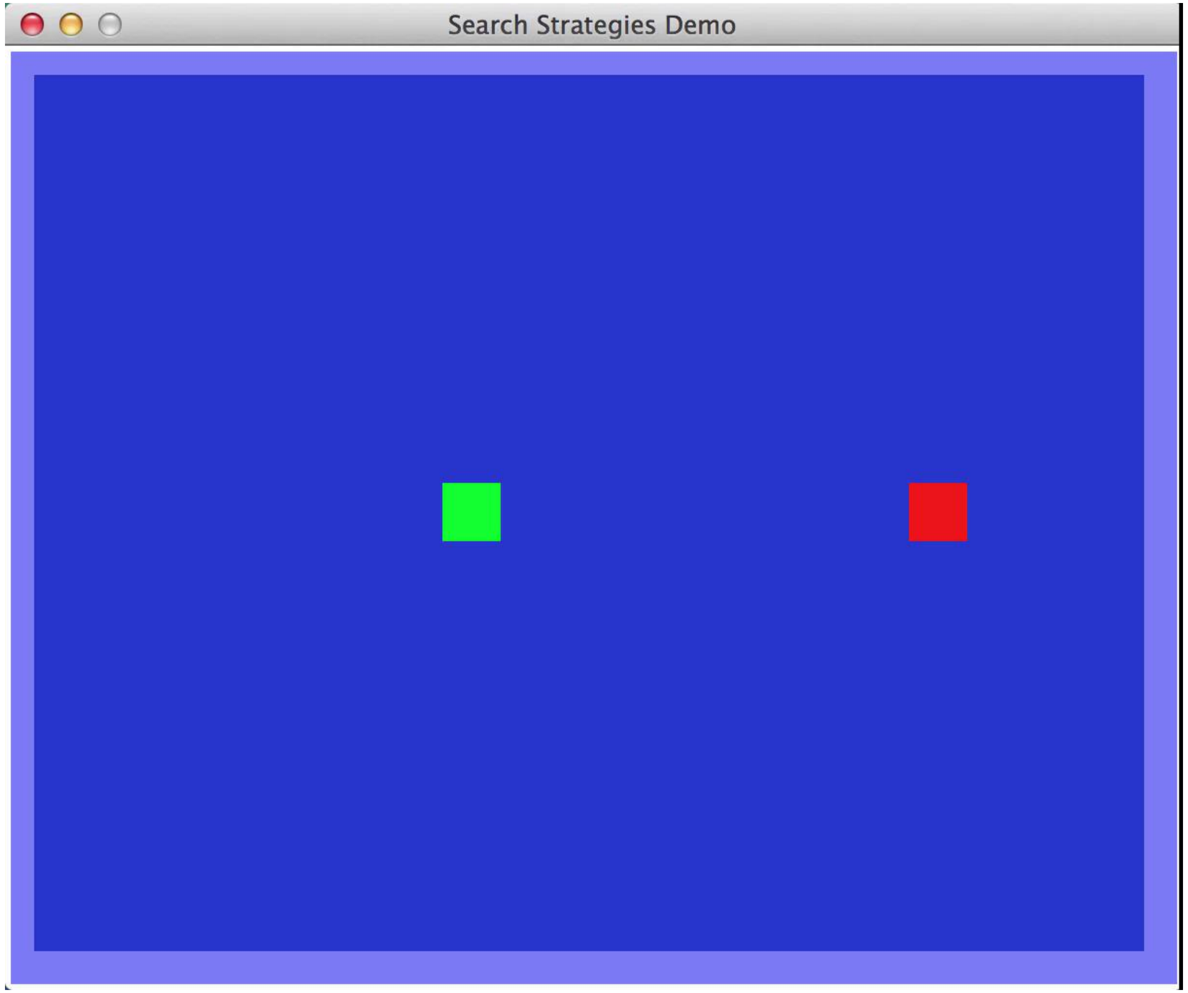
Example: Admissible Heuristic

- Path finding:
  - the straight-line distance to Bucharest



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

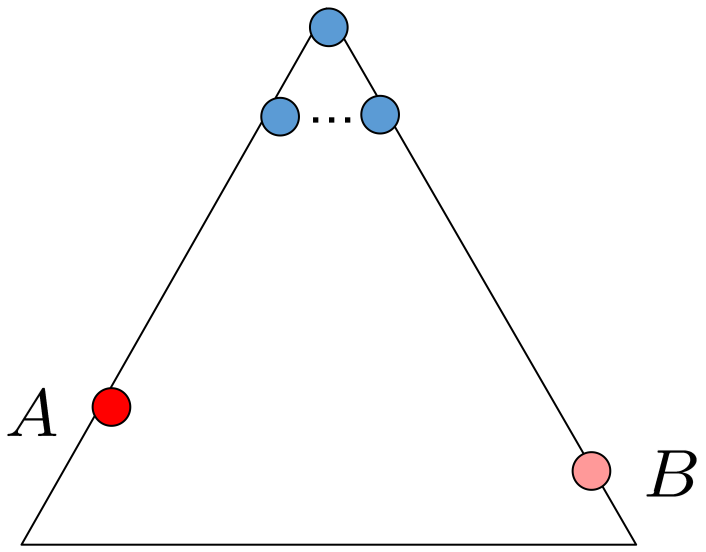
A* Search



Optimality of A* Tree Search

Assume:

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



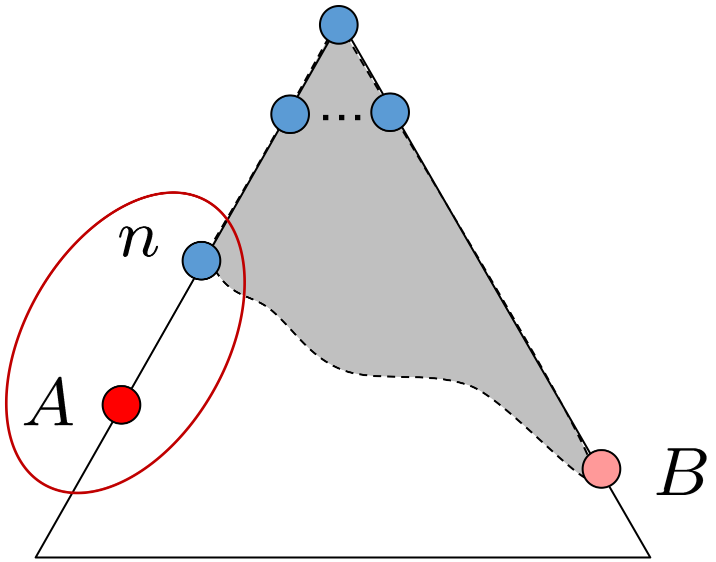
Claim:

A will be expanded before B

Optimality of A* Tree Search

Proof:

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

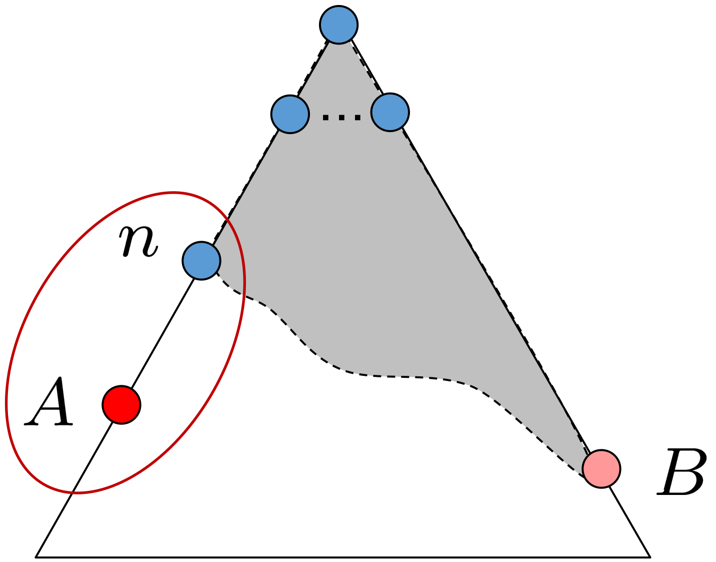


$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	h = 0 at a goal

Optimality of A* Tree Search

Proof:

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



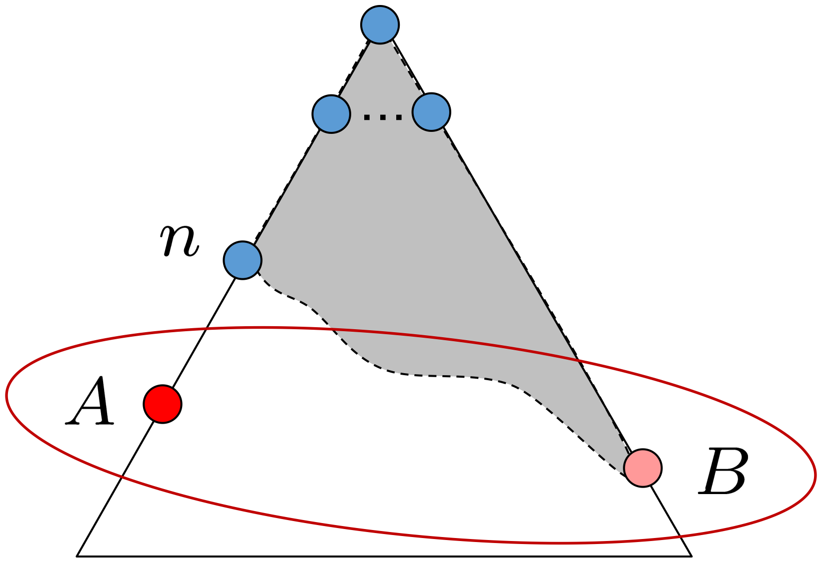
$$f(n) = g(n) + h(n)$$ Definition of f-cost

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

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

- Let h*(n) be the cheapest cost of getting to A from n
- h is admissible -> h(n) ≤ h*(n)
- h*(n) = g(A) - g(n) -> h(n) ≤ g(A) - g(n)
- -> f(n) ≤ f(A)

Optimality of A* Tree Search



Proof:

- Imagine B is on the frontier
- Some ancestor n of A is on the frontier, 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)

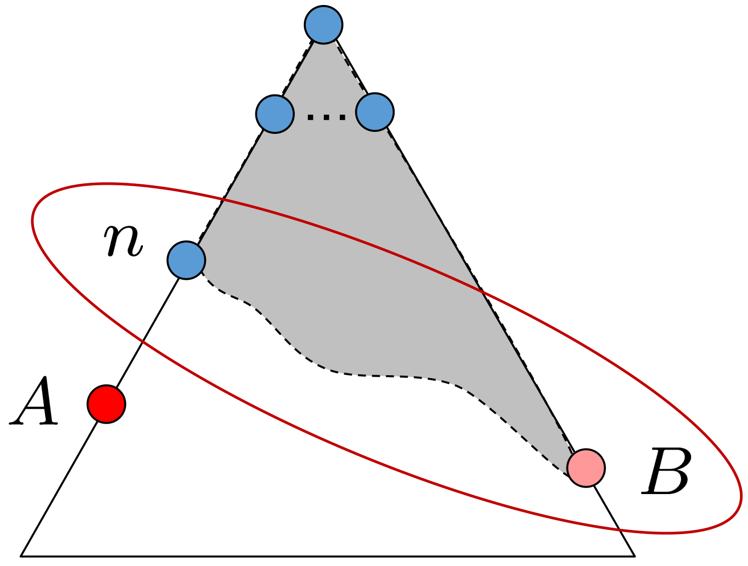
$g(A) < g(B)$ B is sub-optimal

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

Optimality of A* Tree Search

Proof:

- Imagine B is on the frontier
- Some ancestor n of A is on the frontier, 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

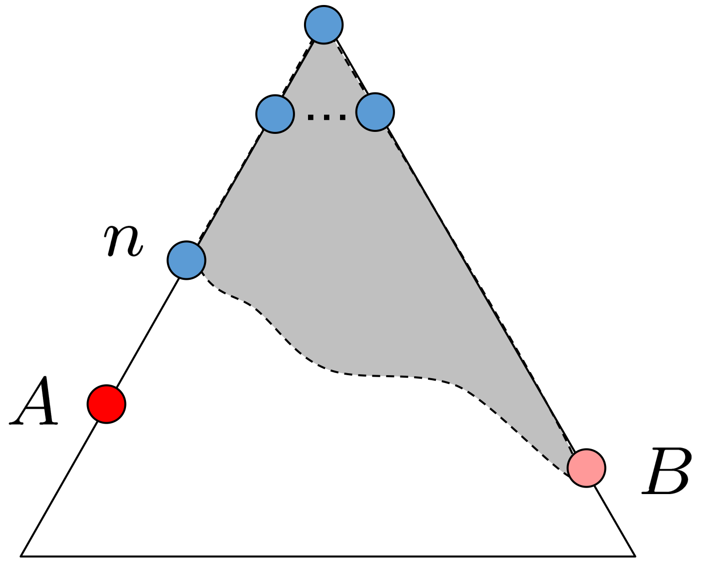


$$f(n) \leq f(A) < f(B)$$

Optimality of A* Tree Search

Proof:

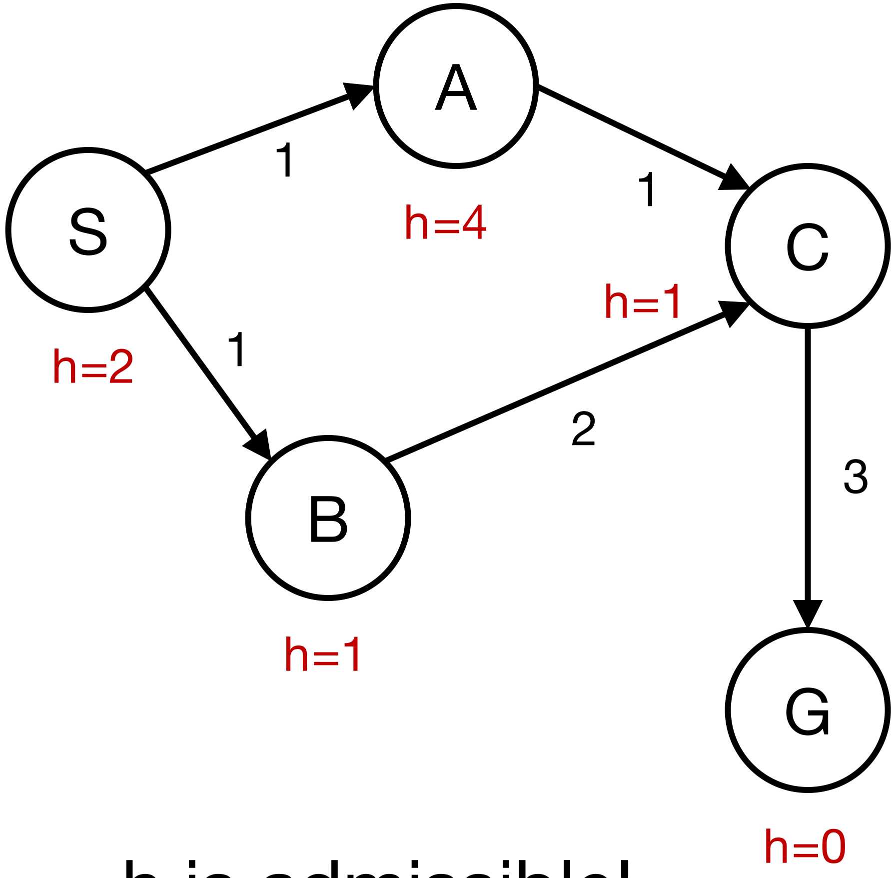
- Imagine B is on the frontier
- Some ancestor n of A is on the frontier, too (maybe A!)
- Claim: n will be expanded before B
    1. f(n) is less or equal to f(A)
    2. f(A) is less than f(B)
    3. n expands before B
- All ancestors of A expand before B --> A expands before B



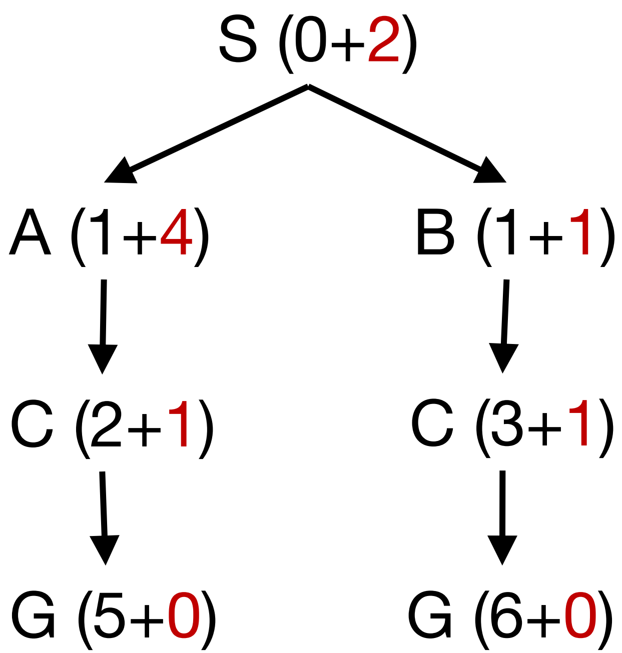
$$f(n) \leq f(A) < f(B)$$

A* Graph Search Gone Wrong?

State space graph

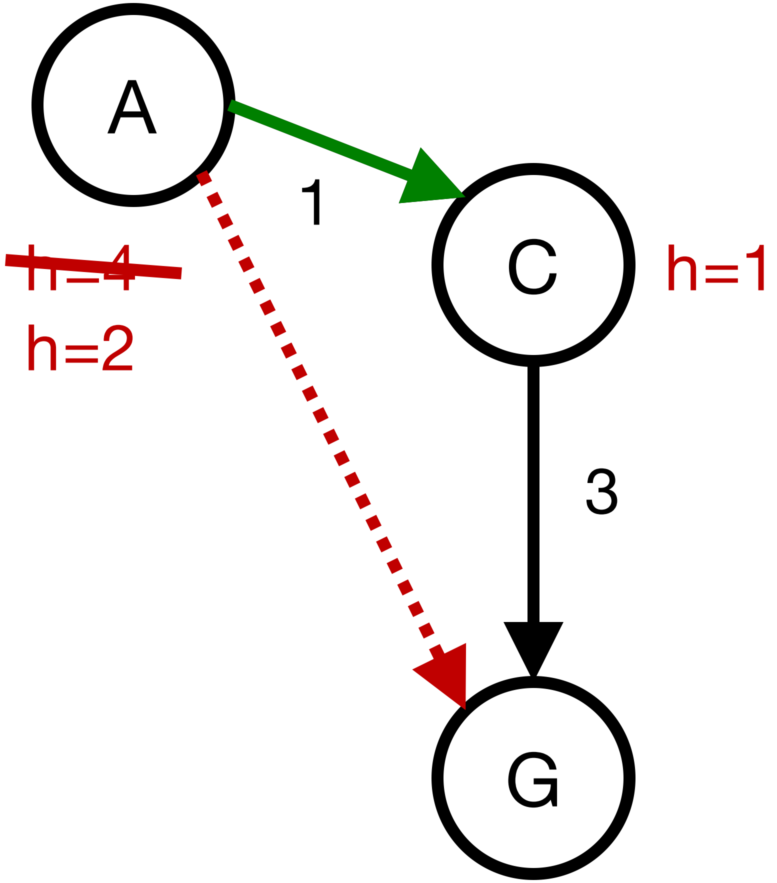


Search tree



h is admissible!

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

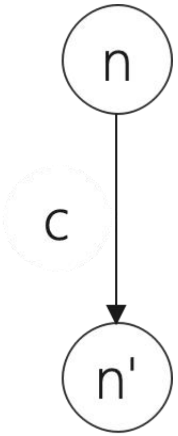
- Consistency: heuristic “arc” cost ≤ actual cost for each arc

    h(A) – h(C) ≤ cost(A to C)

Consequence of Consistency

- The f value along a path never decreases

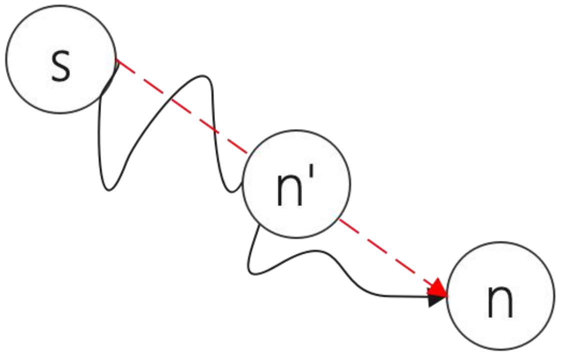
$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c + h(n') \\
&\geq g(n) + h(n) \qquad \text{consistency} \\
&= f(n)
\end{aligned}
$$



Consequence of Consistency

- When A* selects a node for expansion, the optimal path to that node has been found
- Proof:

1. Assume $g(n) > g^*(n)$
2. Let n' be the shallowest node in frontier on the optimal path from s to n
3. $g(n') = g^*(n')$ and $f(n') = g^*(n') + h(n')$
4. We have $f(n') \leq g^*(n') + c(n', n) + h(n)$ consistency
5. $f(n') \leq g^*(n) + h(n)$
6. $f(n') < f(n)$ contradiction



Optimality of A* Graph Search

- Consider what A* does with a consistent heuristic:
  - Fact 1: A* expands nodes in nondecreasing total f value
  - 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

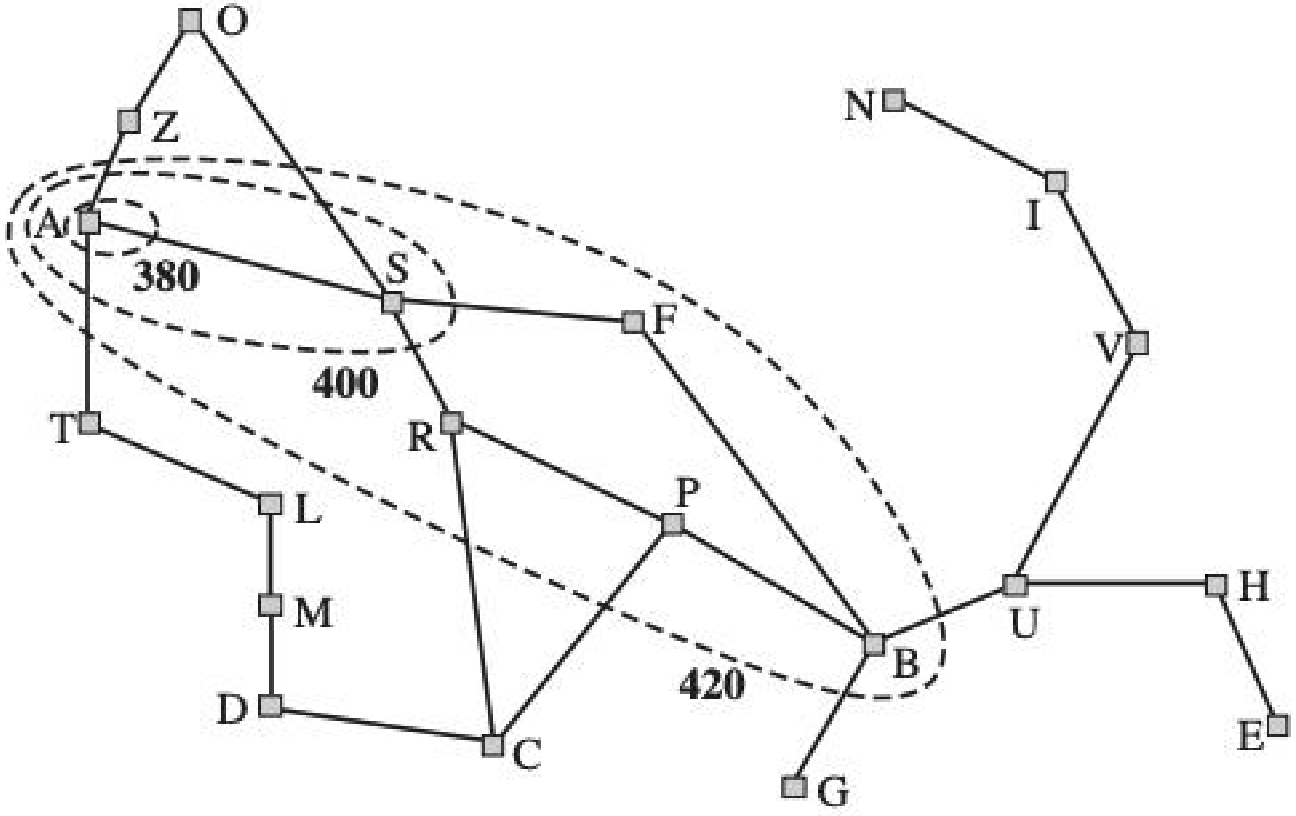
Summary

- 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.

Intuition A*

- Let f* be the cost of optimal solution path.
- We have
  - A* expands all nodes with $f(n) < f^*$
  - A* may then expand some nodes right on “goal contour”, with $f(n) = f^*$ before selecting a goal node.

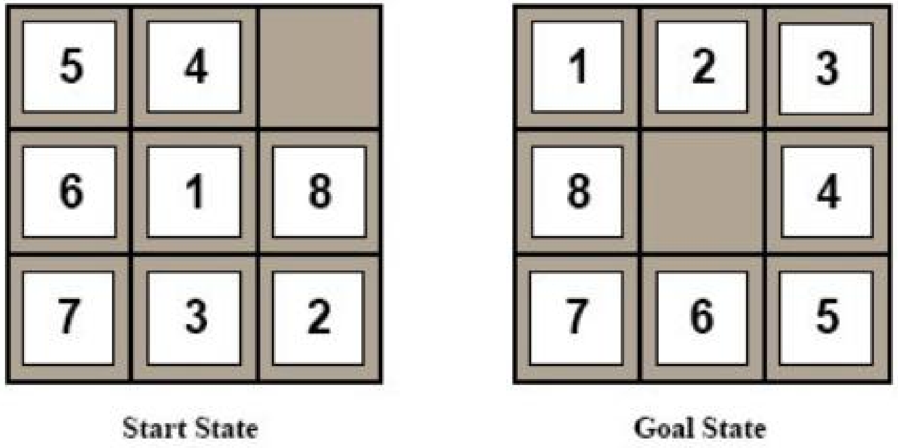
Intuition A*



- A* gradually adds "f-contours" of nodes
- Contour i has all nodes with $f=f_i$, where $f_i < f_{i+1}$

Design of Heuristics

1. h_1 = Number of misplaced tiles
2. h_2 = Manhattan distance
   - the sum of the distances of the tiles from their goal positions



Which one should we use?

$$h_1 \leq h_2 \leq h^*$$

Importance of h(n)

$$h_1 \leq h_2 \leq h^*$$

- Prefer h_2
- Note: Expand all nodes with $f(n) = g(n) + h(n) < f^*$
- Every node with $h(n) < f^* - g(n)$ will be expanded
- Since $h_1 \leq h_2$, every node expanded with h_2 will be expanded by h_1
- Aside. How would we get an h_{opt}?

Comparison of Search Costs on 8-Puzzle

	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

- Effective branch factor b*: characterize the quality of a heuristic

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

Where Do the Heuristics Come From?

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

- Admissible heuristics:

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

- But if $h^*(n)$ is unknown, how can we verify the condition?

Where Do the Heuristics Come From?

- Often, admissible heuristics are solutions to relaxed problems, where new actions are available -> fewer constraints

- The cost of an optimial solution to a relaxed problem is an admissible heuristic for the original problem

- The models obtained by such constraint-deletion processes are called relaxed models.

Example: 8 Queen

- Description of actions:

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

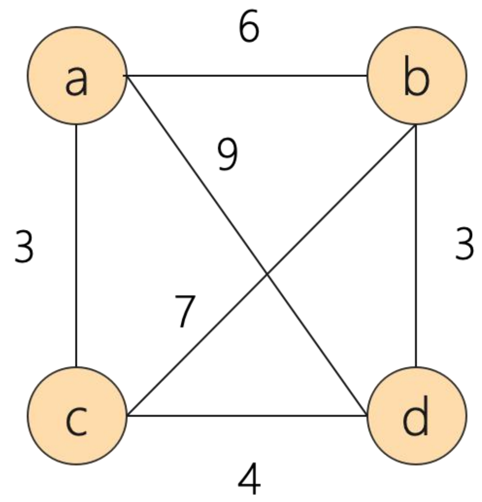
(a) A tile can move from square A to square B if A is adjacent to B.

(b) A tile can move from square A to square B if B is blank

(c) A tile can move from square A to square B

Example: Traveling Salesman Problem

- Find the shortest route that visits each city exactly once and returns to the origin city



- Find an estimate for the cheapest path that starts at a city X, ends at city Y, and go through each unvisited city.

Example: Traveling Salesman Problem

- Find an estimate for the cheapest path that starts at a city X, ends at city Y, and go through each unvisited city.

- Three conditions for the path:
  - Being a graph
  - Being connected
  - Being degree 2

Example: Traveling Salesman Problem

- Find an estimate for the cheapest path that starts at a city X, ends at city Y, and go through each unvisited city.

- Three conditions for the path:
  - Being a graph

    -> Optimal Assignment Problem

  - Being degree 2

Example: Traveling Salesman Problem

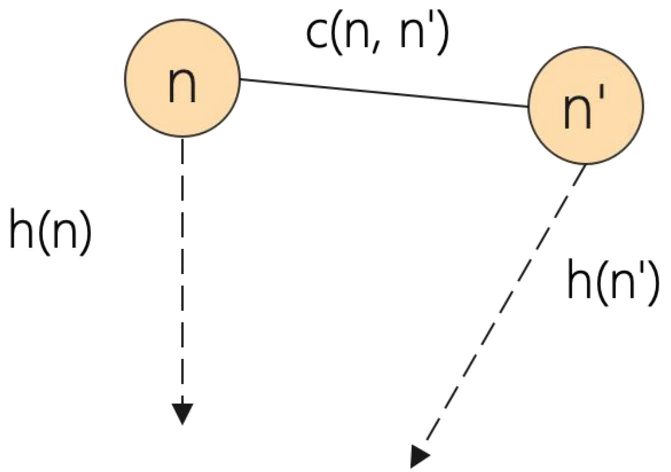
- Find an estimate for the cheapest path that starts at a city X, ends at city Y, and go through each unvisited city.

- Three conditions for the path:
  - Being a graph
  - Being connected

-> Minimum Spanning Tree Problem

Consistency

- Heuristics come from relaxed problems are guaranteed to be consistent.



- h(n) and h(n') stand for the minimum cost of finding solution for some relaxed problem. We have

$$h(n) \leq c'(n, n') + h(n') \quad \text{relaxed cost}$$

$$\rightarrow \quad h(n) \leq c(n, n') + h(n)$$

What if We Have Multple Heuristics

- Suppose we have heuristics $h_1(n)$, $h_2(n)$, $h_3(n)$

- Every node with $h(n) < f^* - g(n)$ will be expanded

- The final heuristic $h(n) = \max(h_1(n), h_2(n), h_3(n))$

A* Applications

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

A*: Summary

Optimal: Yes

A* is optimally efficient: given the information in h, no other optimal search method can expand fewer nodes

Complete: Unless there are infinitely many nodes with $f(n) < f^*$. Asussme locally finite: (1) finite branching, (2) every operator costs at least $\epsilon > 0$.

Complexity (time and space): still exponential because of breadth-first nature. Unless $|h(n) - h^*| <= O(\log(h^*(n))$, with h true cost of getting to goal, the time complexity is polynomial.