PURDUE CS47100

INTRODUCTION TO AI

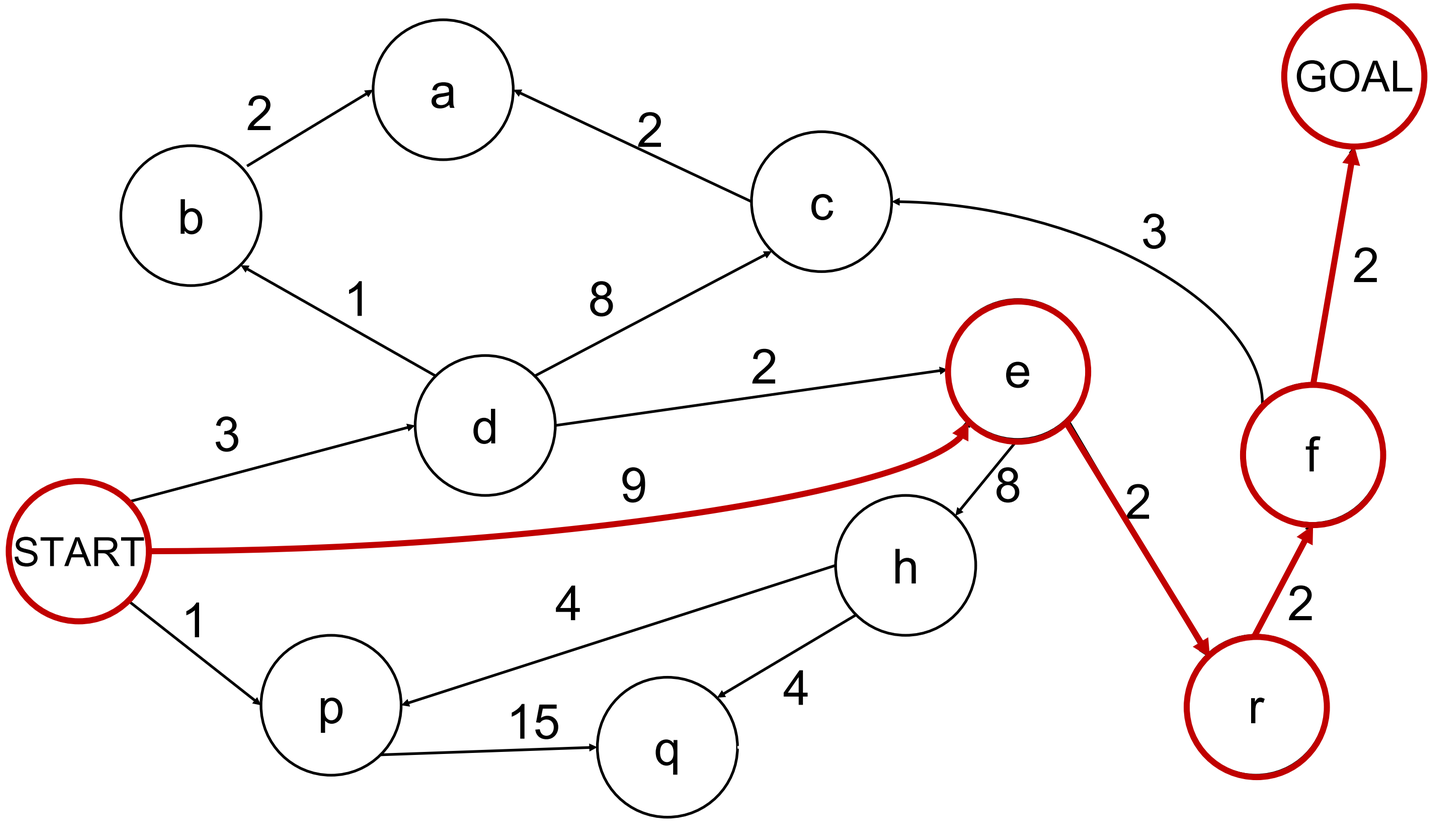
ANNOUNCEMENTS

- Assignment 1 is released on Brightspace
- Submit to GradeScope!
- Due date: 11:59pm, September 23 (Friday)
- 100 points + 28 bonus points
  - It can contributes up to 128/100*15%=19.2% of your final grades
- If you'd like to use extension days on this assignment, specify it clearly in your written pdf submission!

RECAP: UNINFORMED SEARCH STRATEGIES

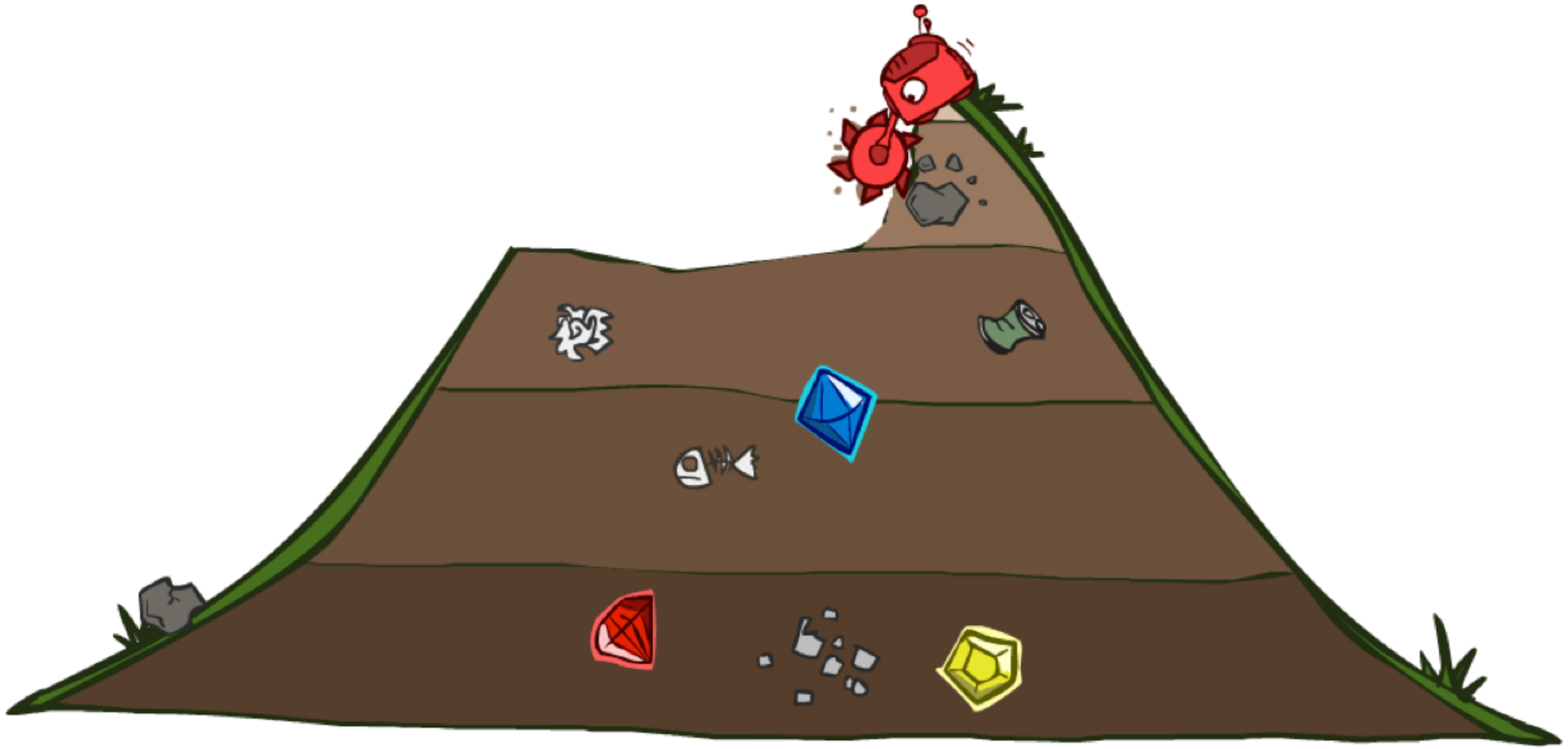
- The search algorithm has no additional information about states beyond what has been provided in the problem formulation
  - Depth-first search
  - Breadth-first search
  - Depth-limited search
  - Iterative deepening depth-first search
  - Bi-directional search

COST-SENSITIVE SEARCH



BFS finds the shortest path in terms of number of actions.
It does not find the least-cost path.

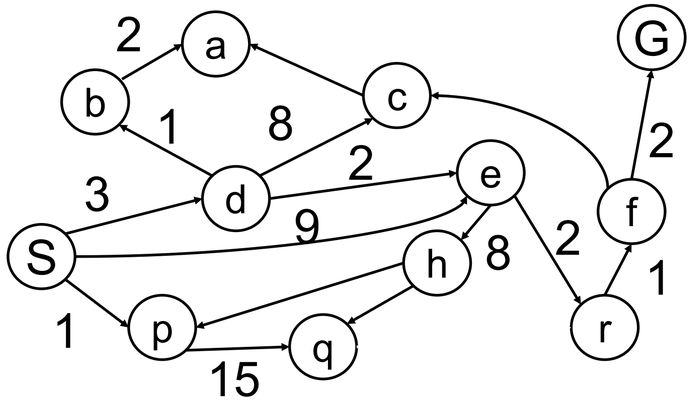
UNIFORM COST SEARCH



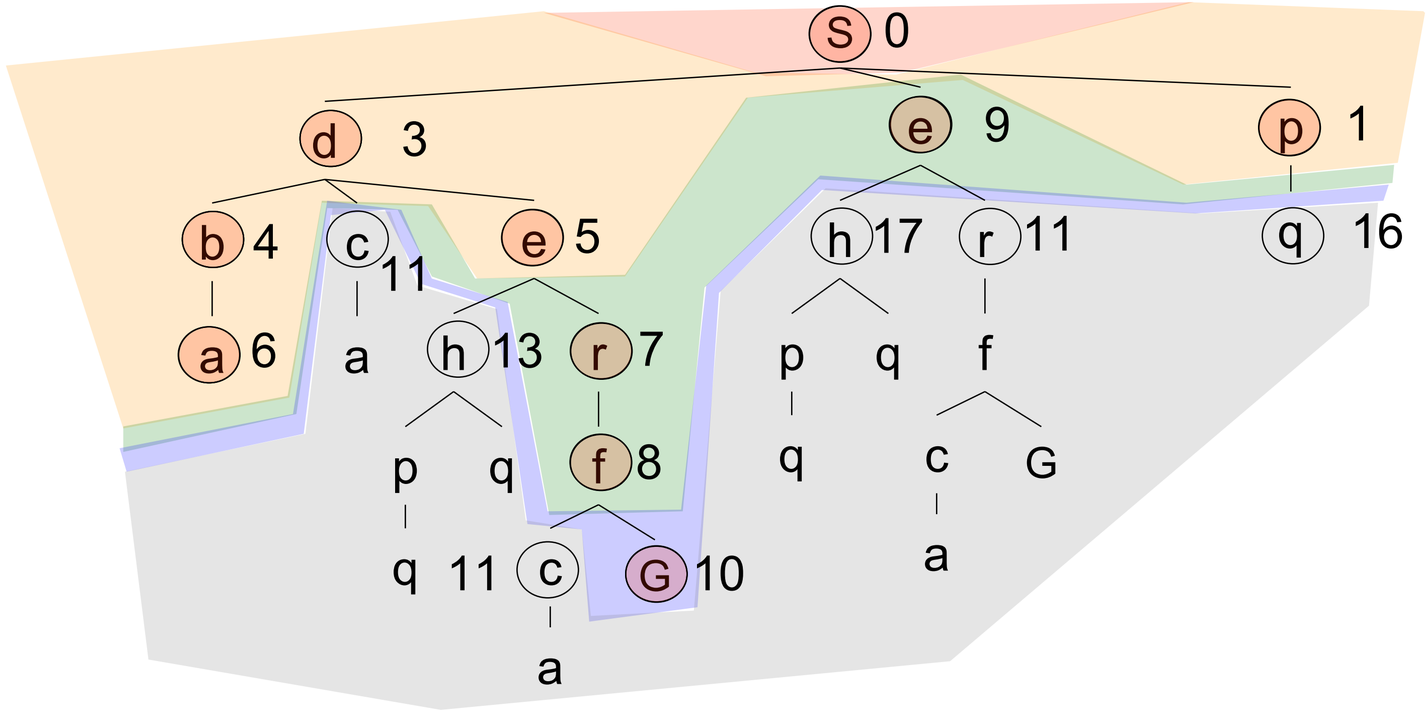
UNIFORM COST SEARCH

Strategy: expand cheapest node first:

Fringe is a priority queue (priority: cumulative cost)

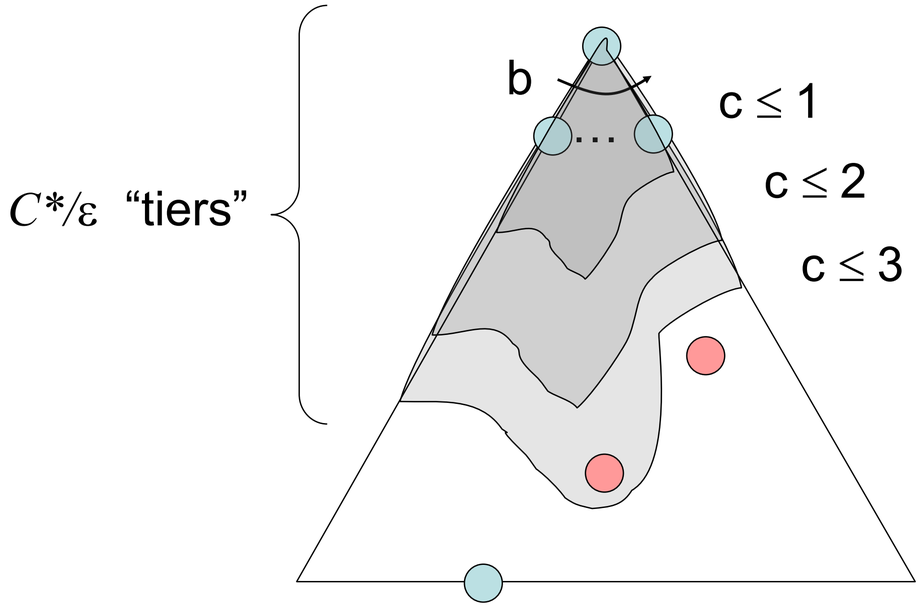


Cost contours



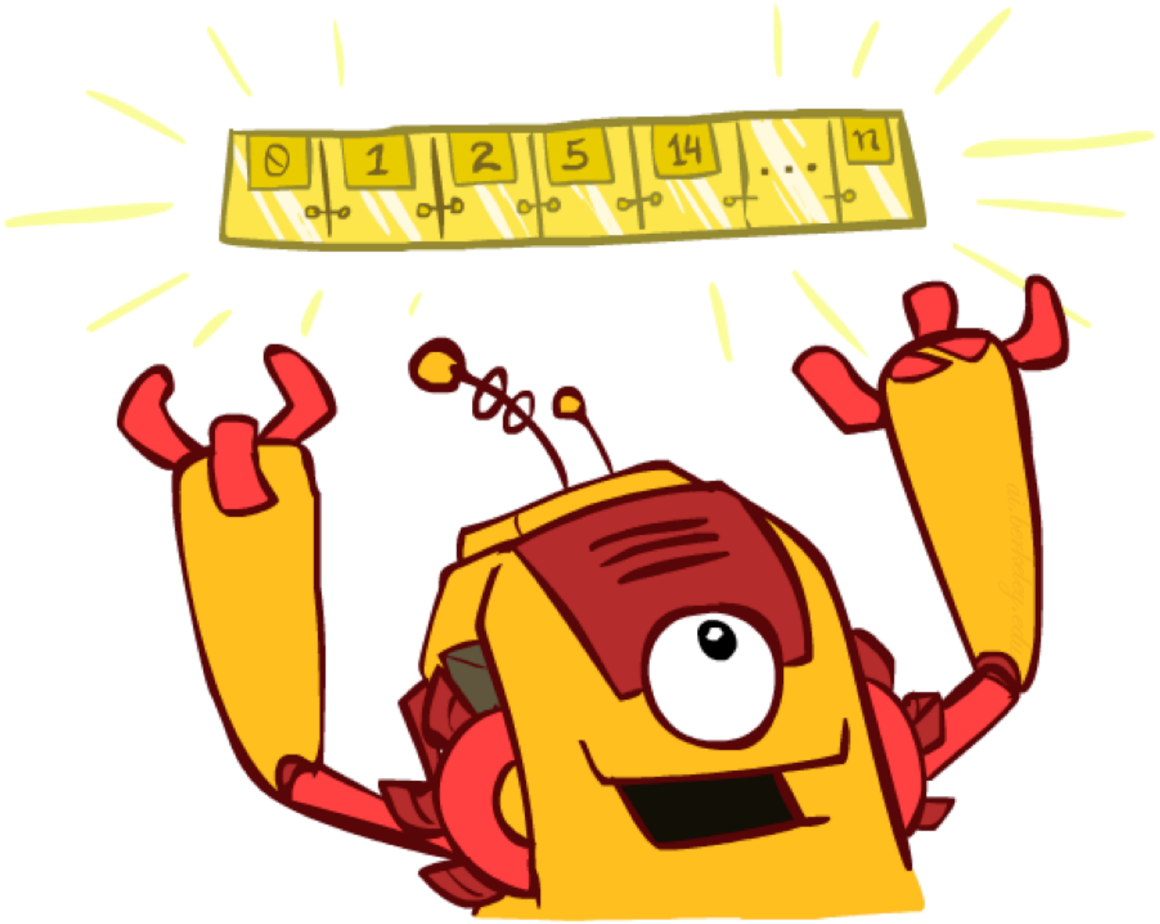
UNIFORM COST SEARCH (UCS) PROPERTIES

- Time complexity
  - Processes all nodes with cost less than cheapest solution
  - If that solution costs C^* and actions cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{1+C^*/\varepsilon})$
- Space complexity
  - Has roughly the successors of the last tier, so $O(b^{1+C^*/\varepsilon})$
- Is it complete?
  - Yes, assuming best solution has finite cost and min action cost is positive
- Is it optimal?
  - Yes



ONE SEARCH QUEUE

- All these search algorithms are the same except for fringe strategies
  - Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
  - Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
  - Can even code one implementation that takes a variable queuing object



SUMMARY — UNINFORMED SEARCH STRATEGIES

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

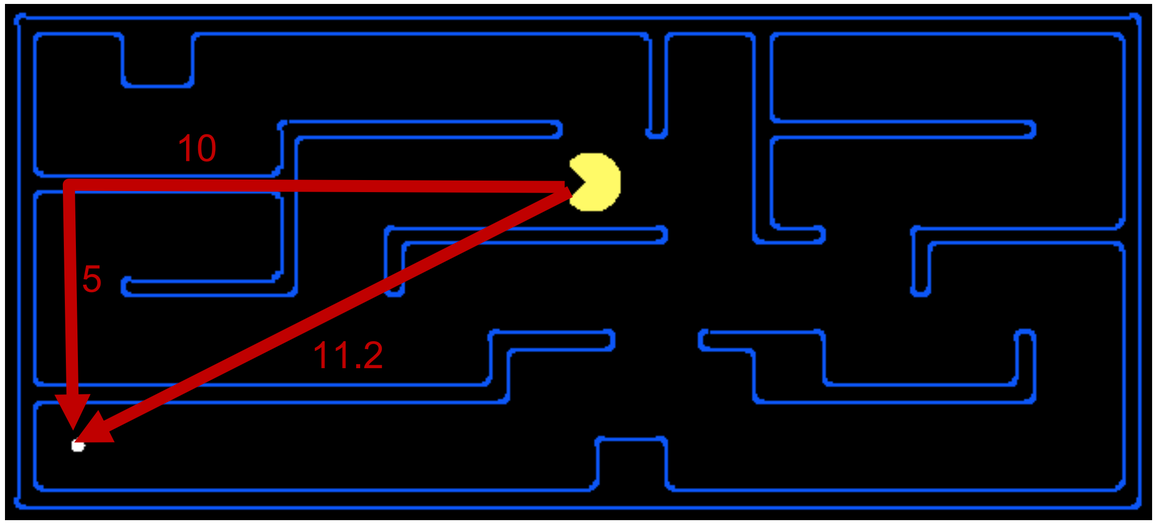
Figure 3.21 Evaluation of search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

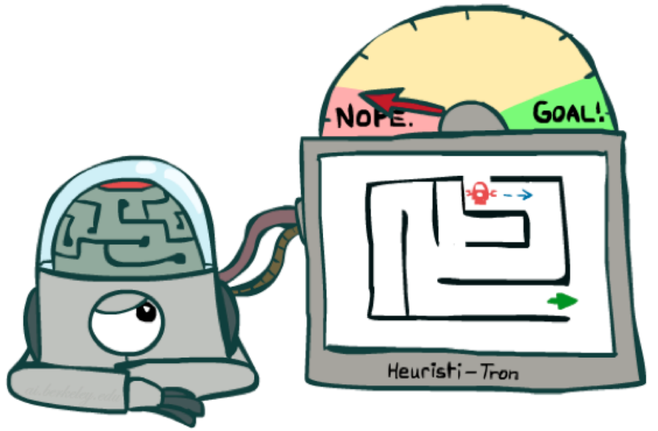
INFORMED SEARCH

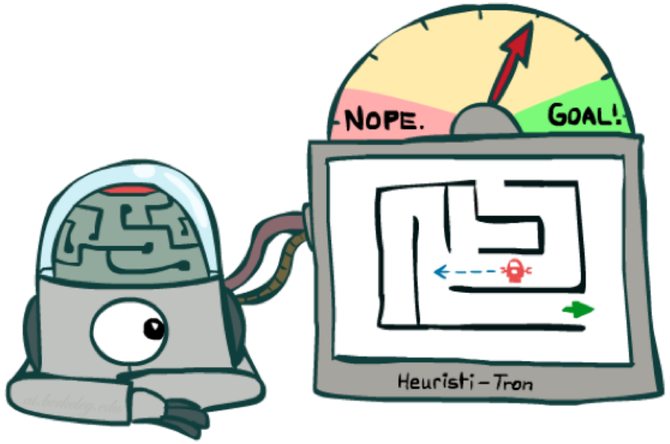
- Determine how to prioritize the expansion of different states based on information beyond what has been provided in the problem formulation
- **Key idea:** Uses an evaluation function *f(n)*, which estimate the "cost" of node *n*
  - *f(n)* usually contains a heuristic function *h(n)*, which is the estimated cost of the cheapest path from the state at node *n* to a goal state.
  - **Best-first search**: Select node with minimum *f(n)* for expansion

HEURISTIC FUNCTION

- A ***heuristic function*** is:
  - A non-negative function that estimates how close a state is to a goal, designed for a particular search problem
  - $h(n) = 0$ if n is a goal state
  - Examples: Manhattan distance, Euclidean distance for pathing



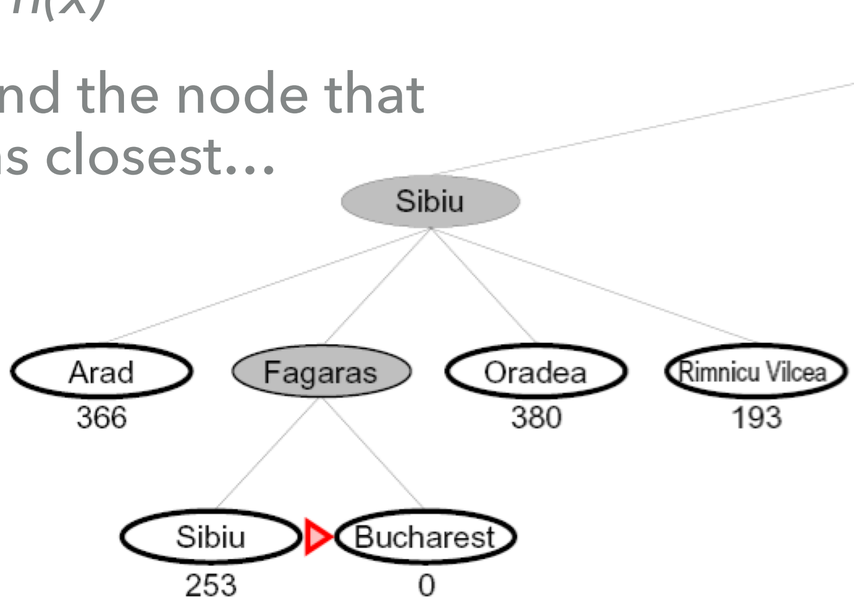




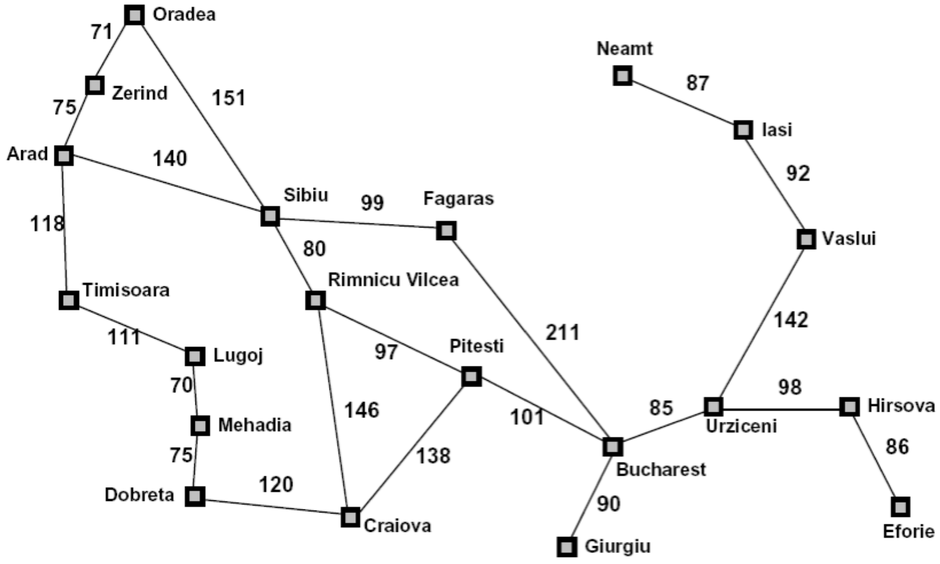
GREEDY SEARCH

GREEDY SEARCH

- $f(x) = h(x)$
- Expand the node that seems closest…



- What can go wrong?

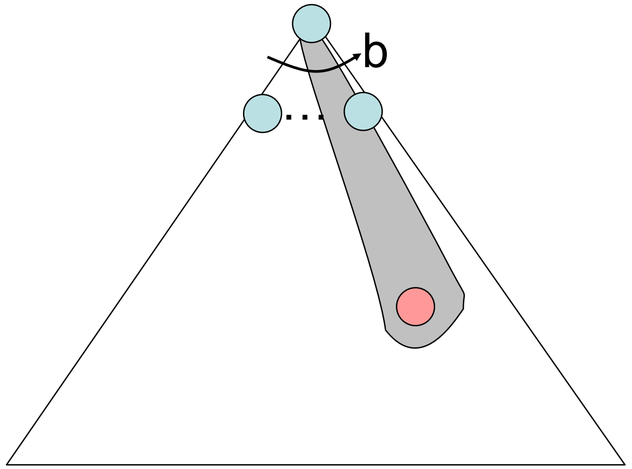


h(x)

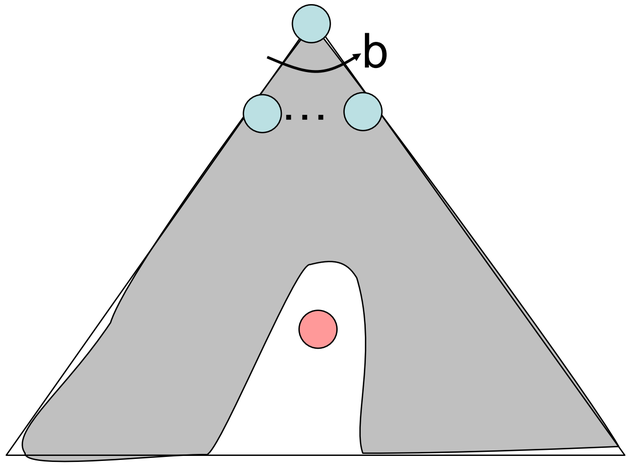
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

GREEDY SEARCH

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state



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



- Worst-case: like a badly-guided DFS

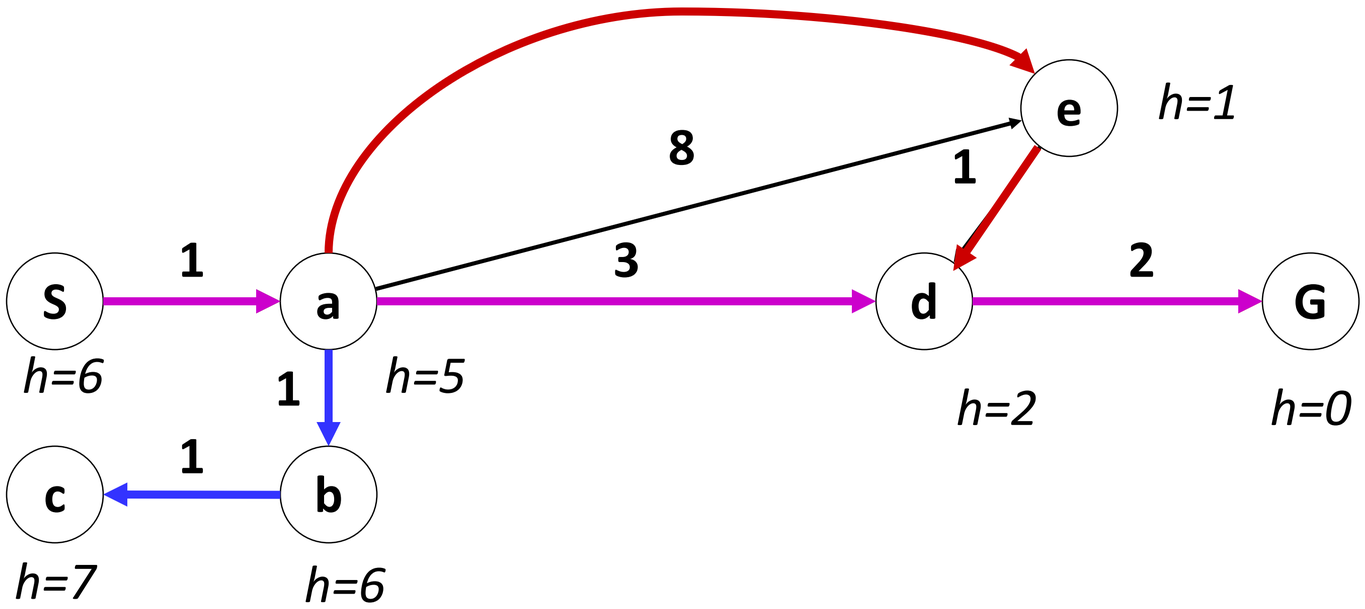
A* SEARCH

A* SEARCH

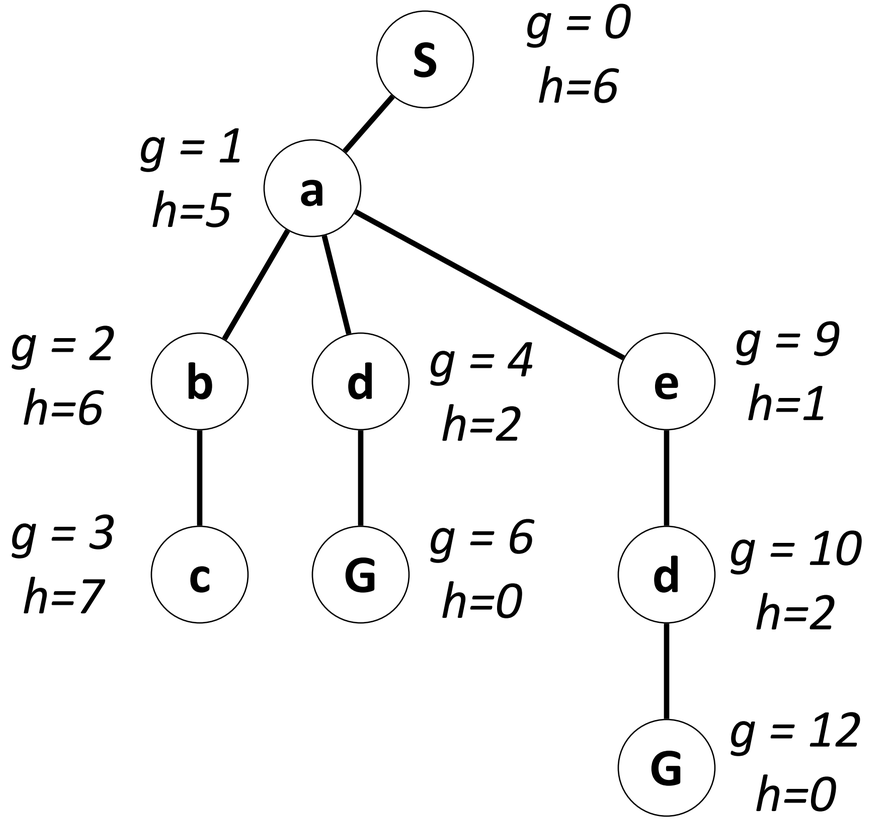
- $f(n) = g(n) + h(n)$
  - g(n) = path cost from start node to current node n — Uniform cost search
  - h(n) = estimate of path cost from n to the goal — Greedy best-first search
- Expands the node *n* if the estimated cost of the solution passing through it is the lowest
- Combines uniform cost search and greedy search
- A* search is optimal if *h(n)* is an **underestimate** of the actual cost from *n* to the goal

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)



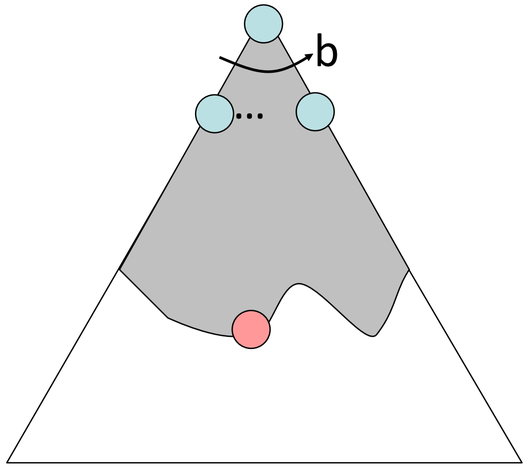
- A* Search orders by the sum: f(n) = g(n) + h(n)



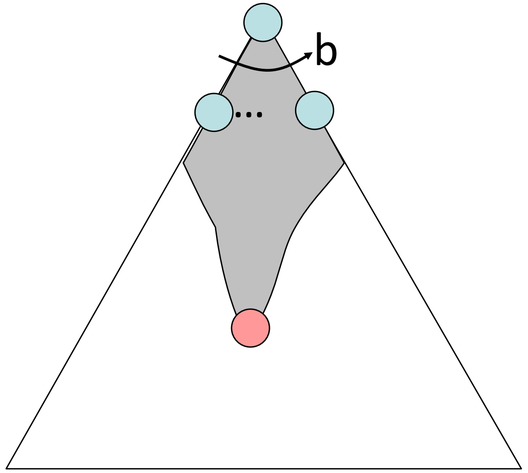
Example: Teg Grenager

UCS VS A*

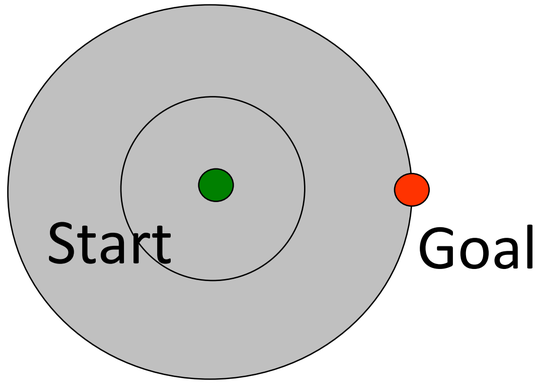
Uniform-Cost

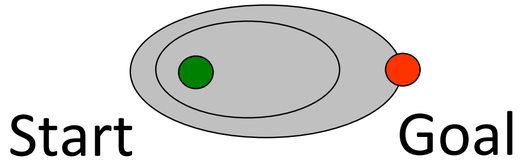


A*



Uniform-cost expands equally in all “directions”

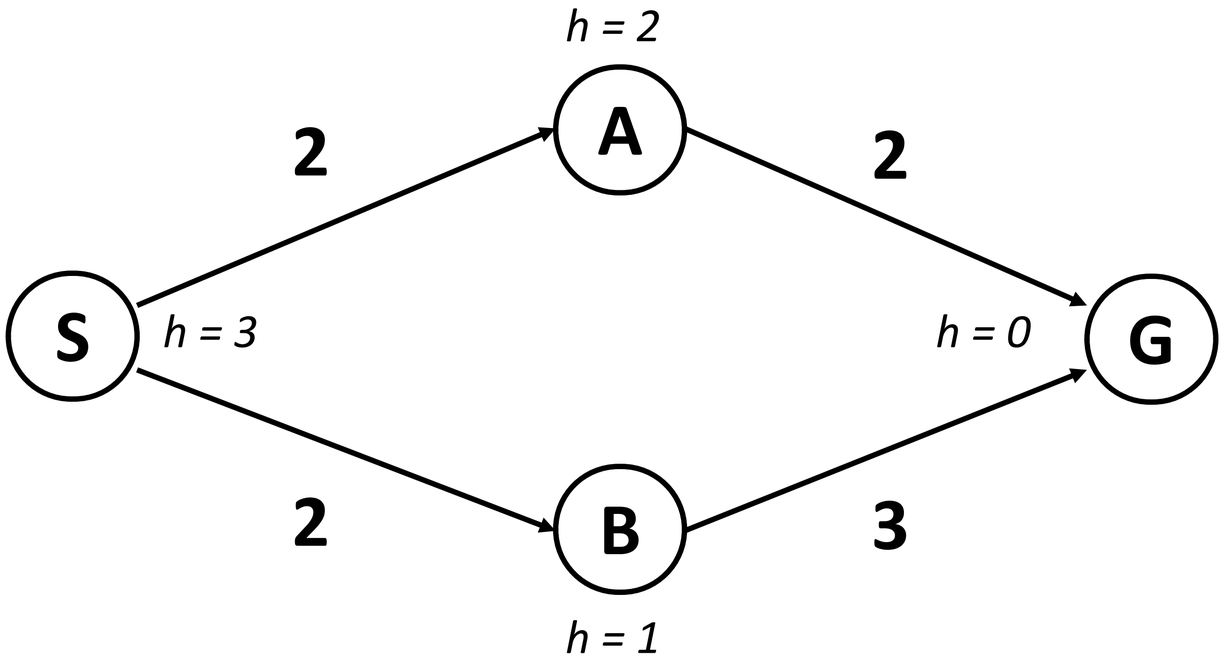




A* expands mainly toward the goal, but does hedge its bets to ensure optimality

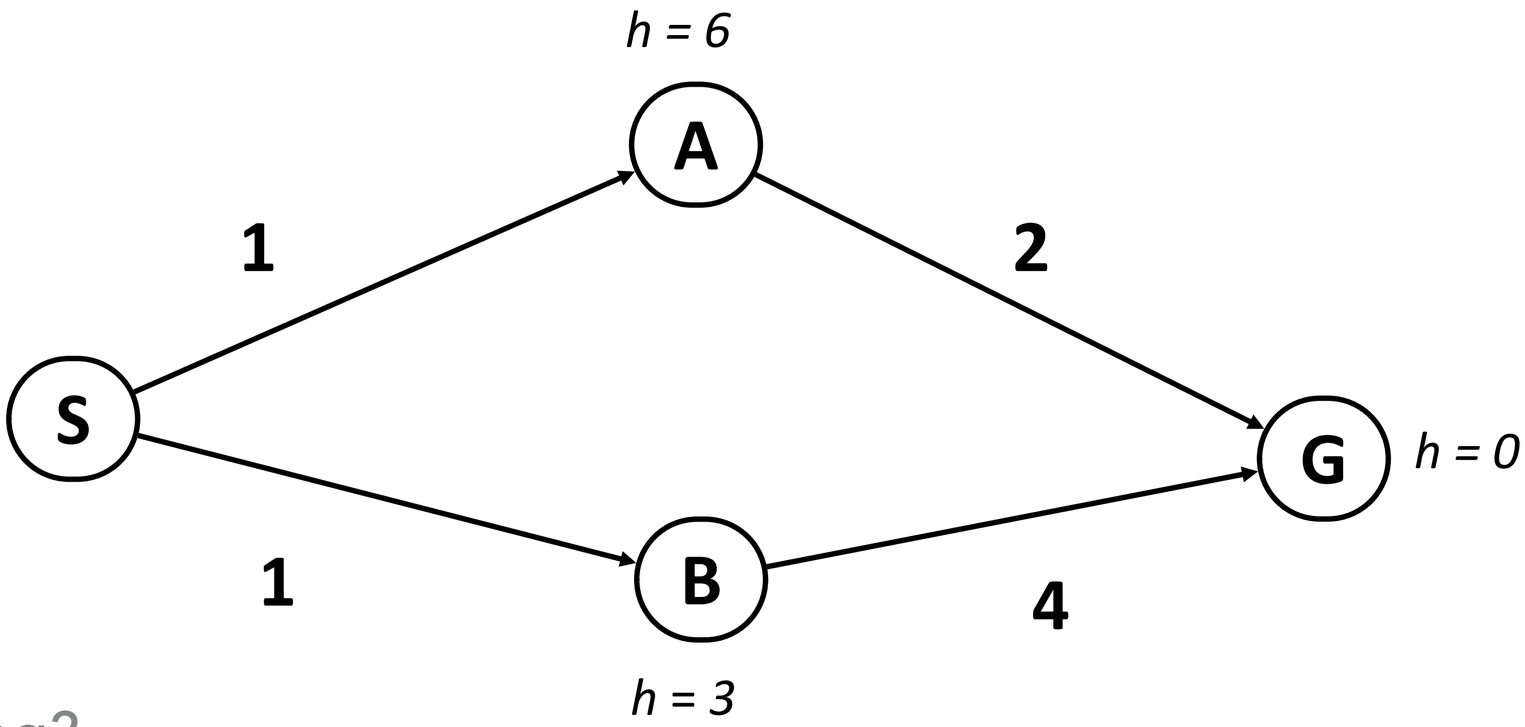
WHEN SHOULD A* TERMINATE?

- Should we stop when a goal state enters the fringe?



- **No**: only stop when we a goal state is expanded (i.e., exits the fringe)

IS A* OPTIMAL?

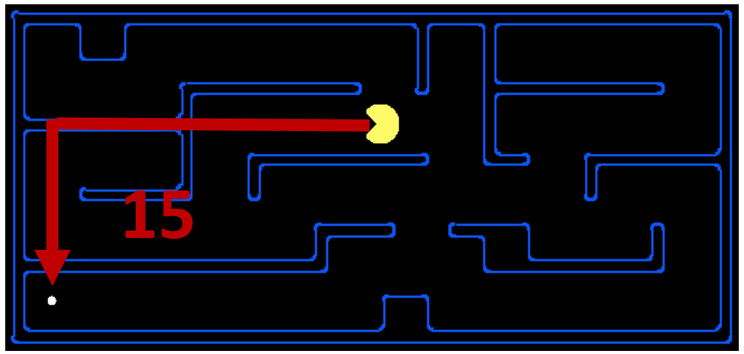


- What went wrong?
- We need **estimated costs *h(n)* to be less than actual costs**

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.

- Example:

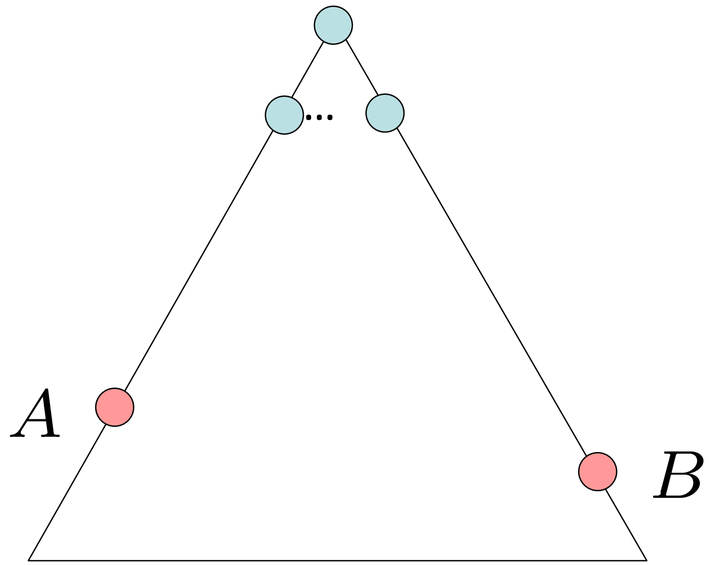


- Coming up with admissible heuristics is most of what's involved in using A* in practice

OPTIMALITY OF A* TREE SEARCH

Assume:

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



Claim:

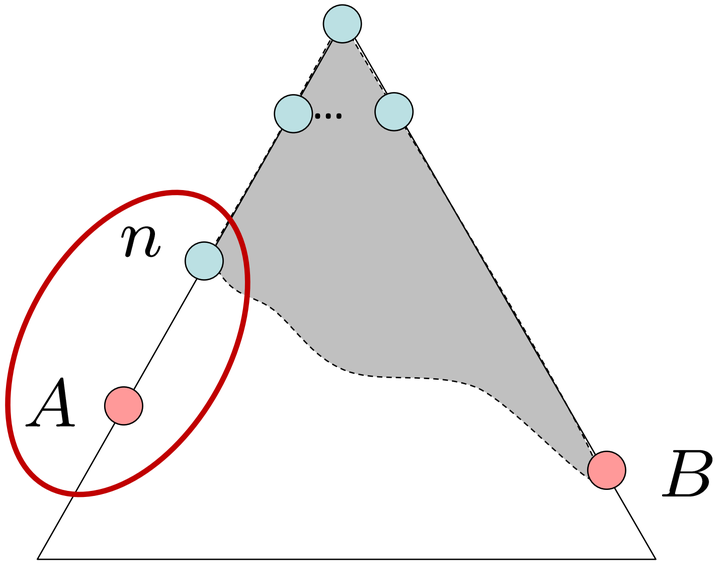
- A will exit the fringe before B

OPTIMALITY OF A* TREE SEARCH

Proof:

- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe
- Claim: *n* will be expanded before B

1. f(n) is less or equal to f(A)



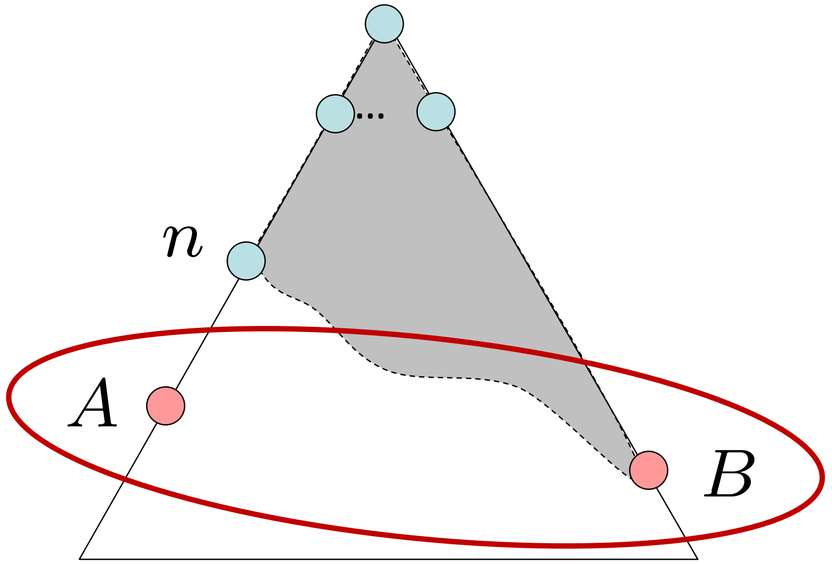
$$f(n) = g(n) + h(n) \quad \text{Definition of f-cost}$$
$$f(n) \leq g(A) \quad \text{Admissibility of h}$$
$$g(A) = f(A) \quad \text{h = 0 at a goal}$$

OPTIMALITY OF A* TREE SEARCH

Proof:

- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe
- 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

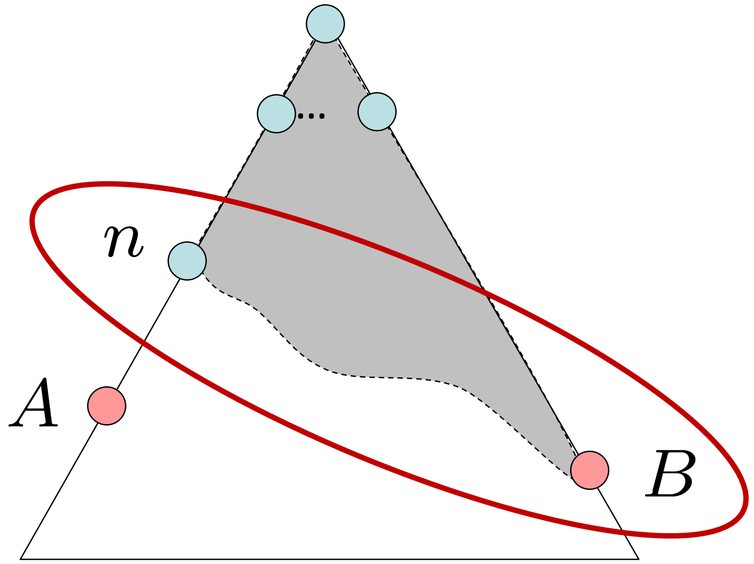
$f(A) < f(B)$ h = 0 at a goal

OPTIMALITY OF A* TREE SEARCH

Proof:

- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe
- 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

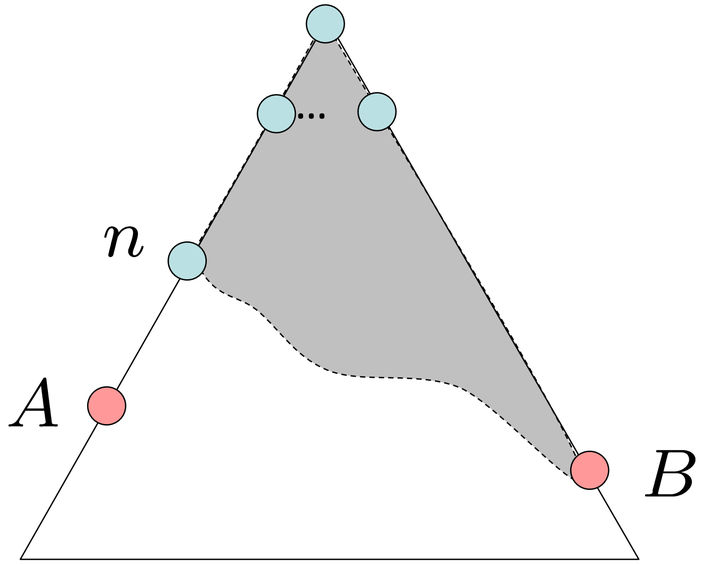


$$f(n) \leq f(A) < f(B)$$

OPTIMALITY OF A* TREE SEARCH

Proof:

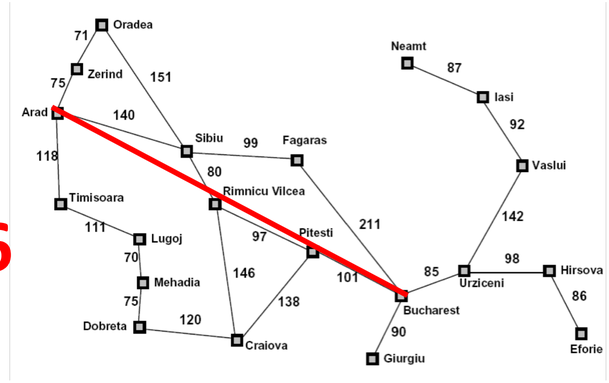
- Imagine B is on the fringe; some ancestor *n* of A is on the fringe
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)
  3. *n* expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal

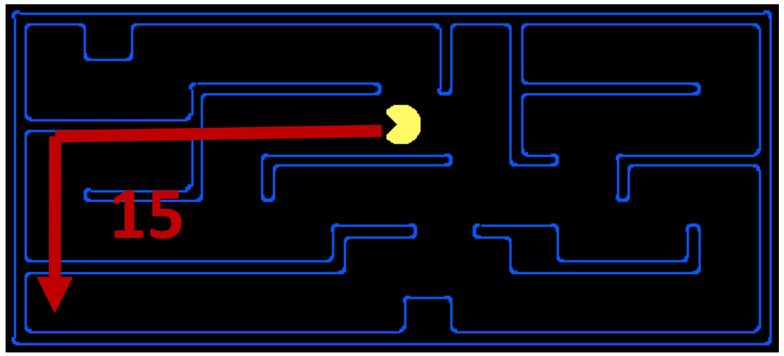


CREATING ADMISSIBLE HEURISTICS

- 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

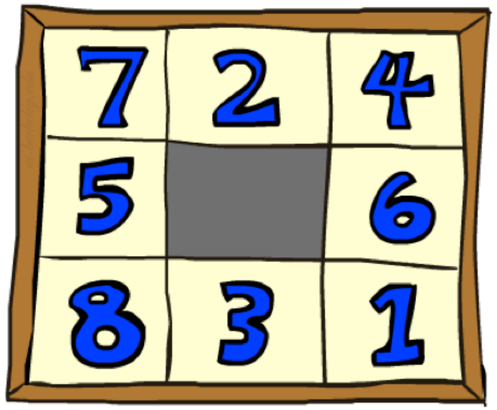
366



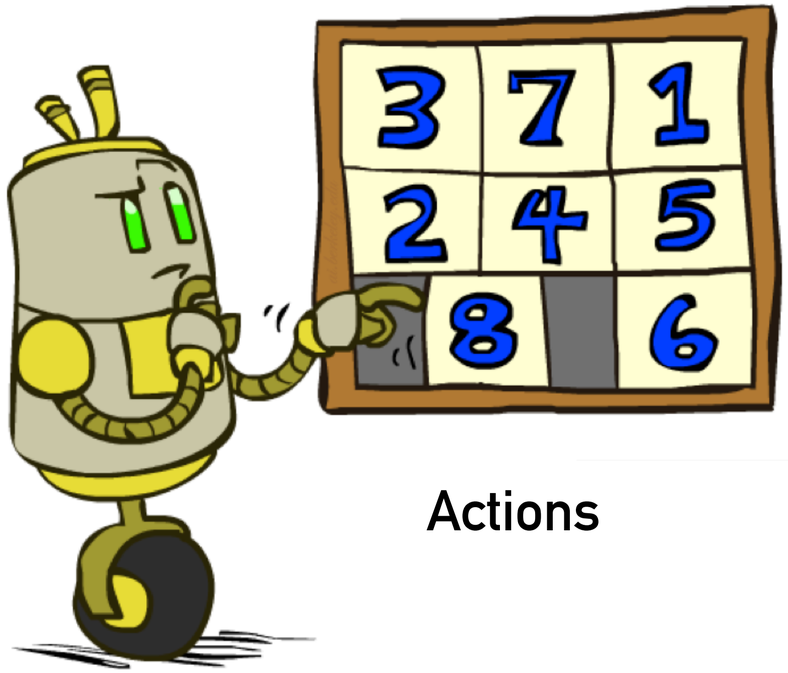


- Inadmissible heuristics are often useful too

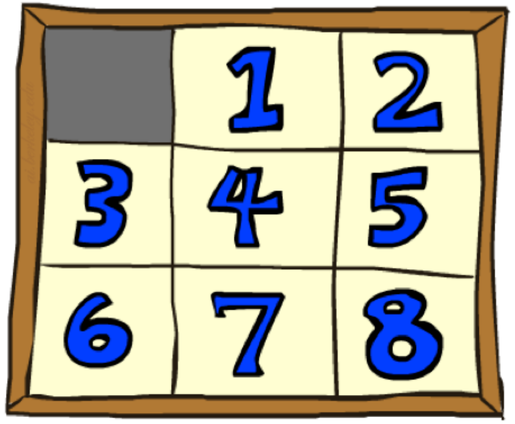
ADMISSIBLE HEURISTICS EXAMPLE: 8 PUZZLE



Start State



Actions



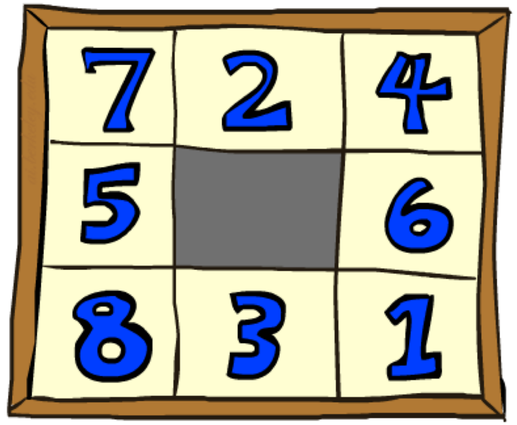
Goal State

What could be an admissible heuristic function for 8-puzzle problems?

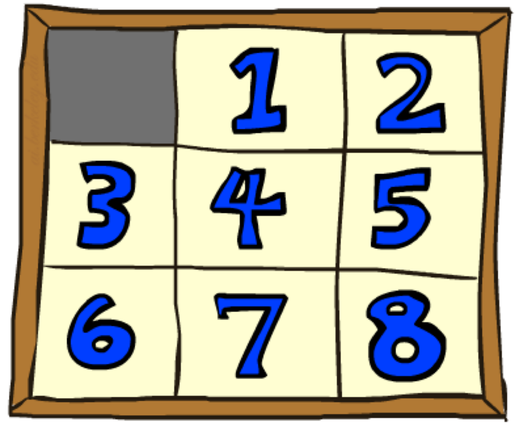
Hint: A tile can move from Location A to B if A and B is adjacent and B is blank…Relax it?

8 PUZZLE HEURISTIC I

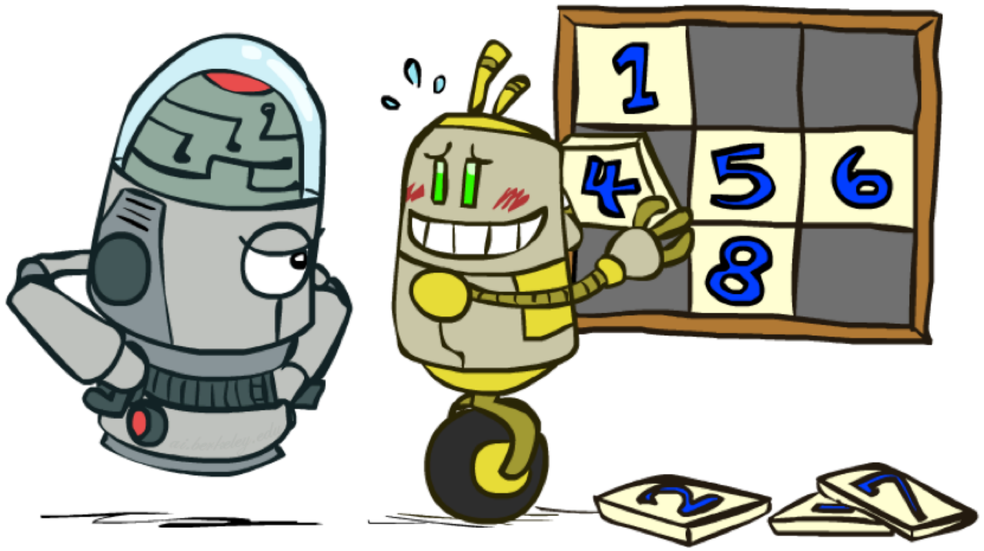
- Heuristic: Number of misplaced tiles
- Why is it admissible?
- h(start) = 8
- This is a *relaxed-problem* heuristic



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