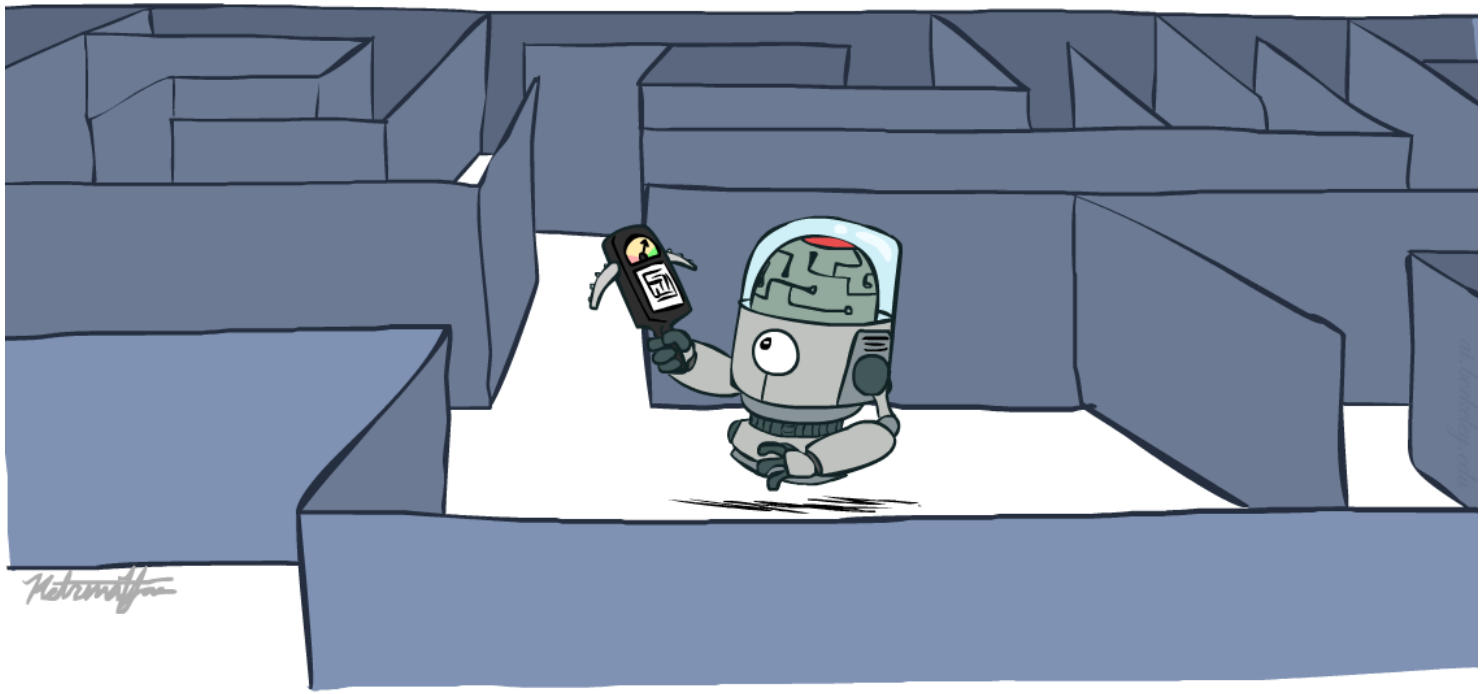
CS 161: Fundamentals of Artificial Intelligence

Informed Search

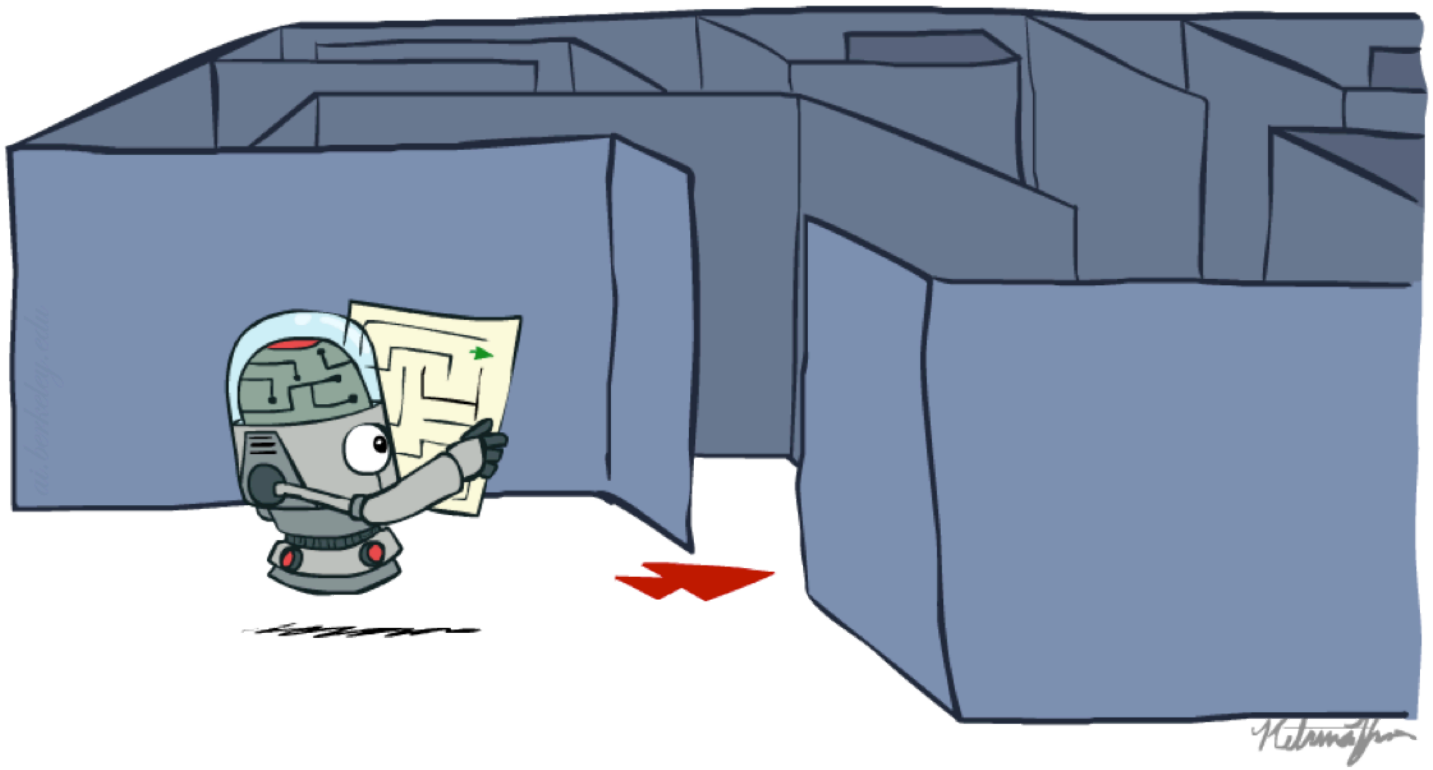


University of California Los Angeles

Today

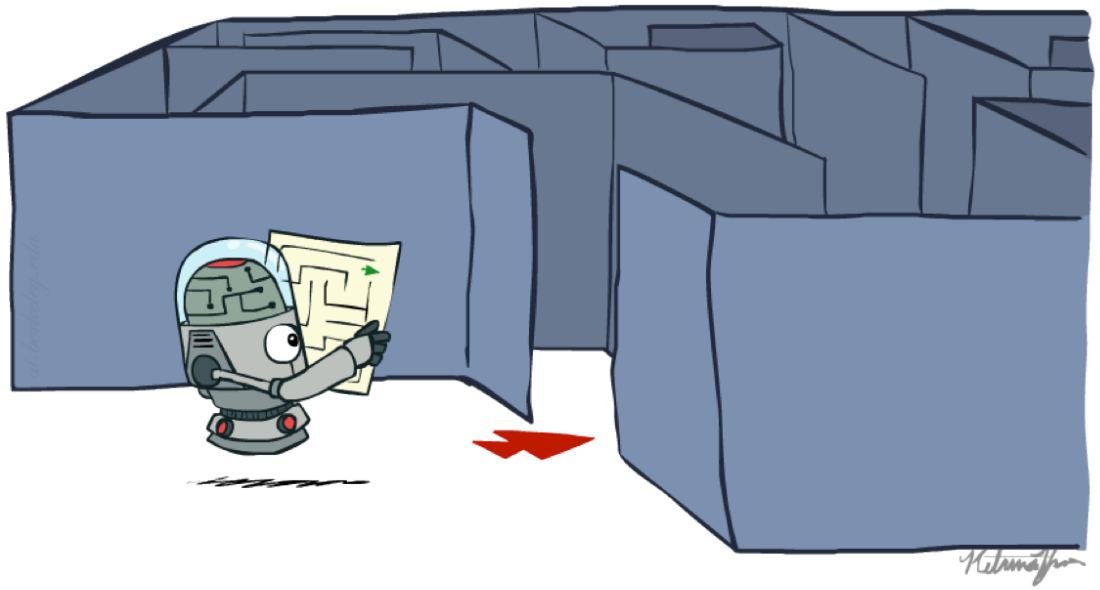
- Informed Search
  - Uniform Cost Search
  - Heuristics
  - Greedy Search
  - A* Search
- Graph Search

Recap: Search



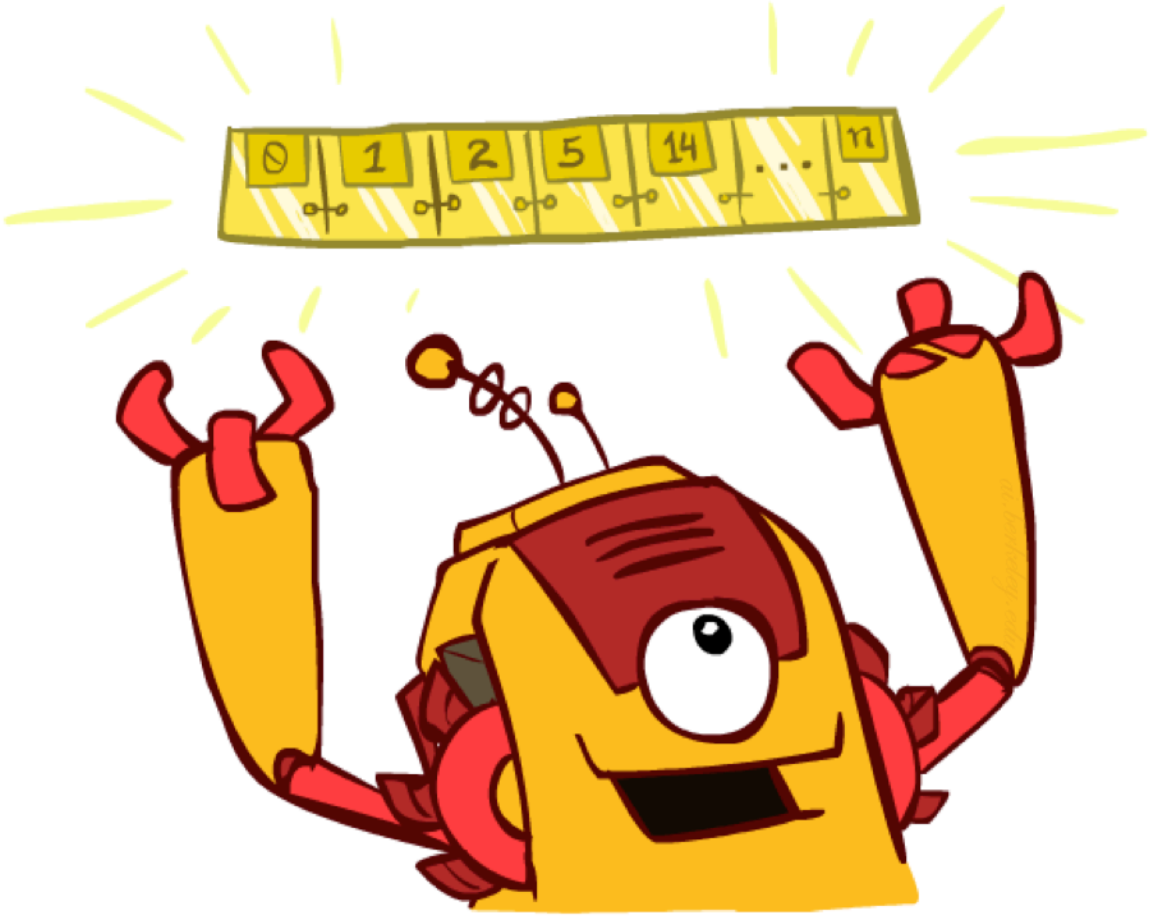
Recap: Search

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

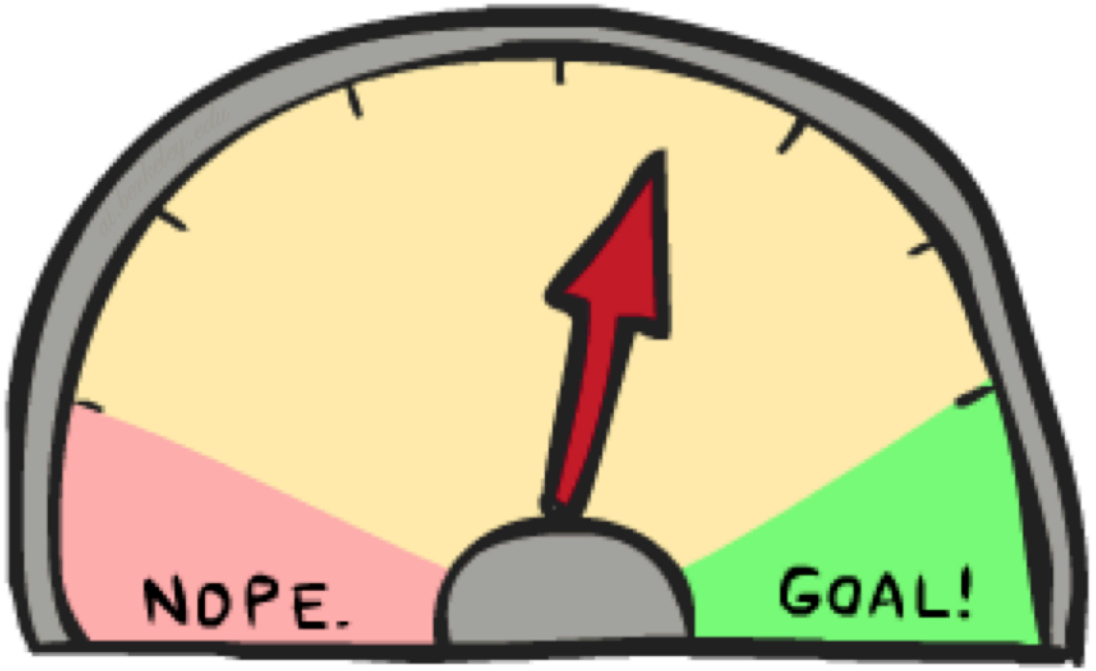


The One Queue

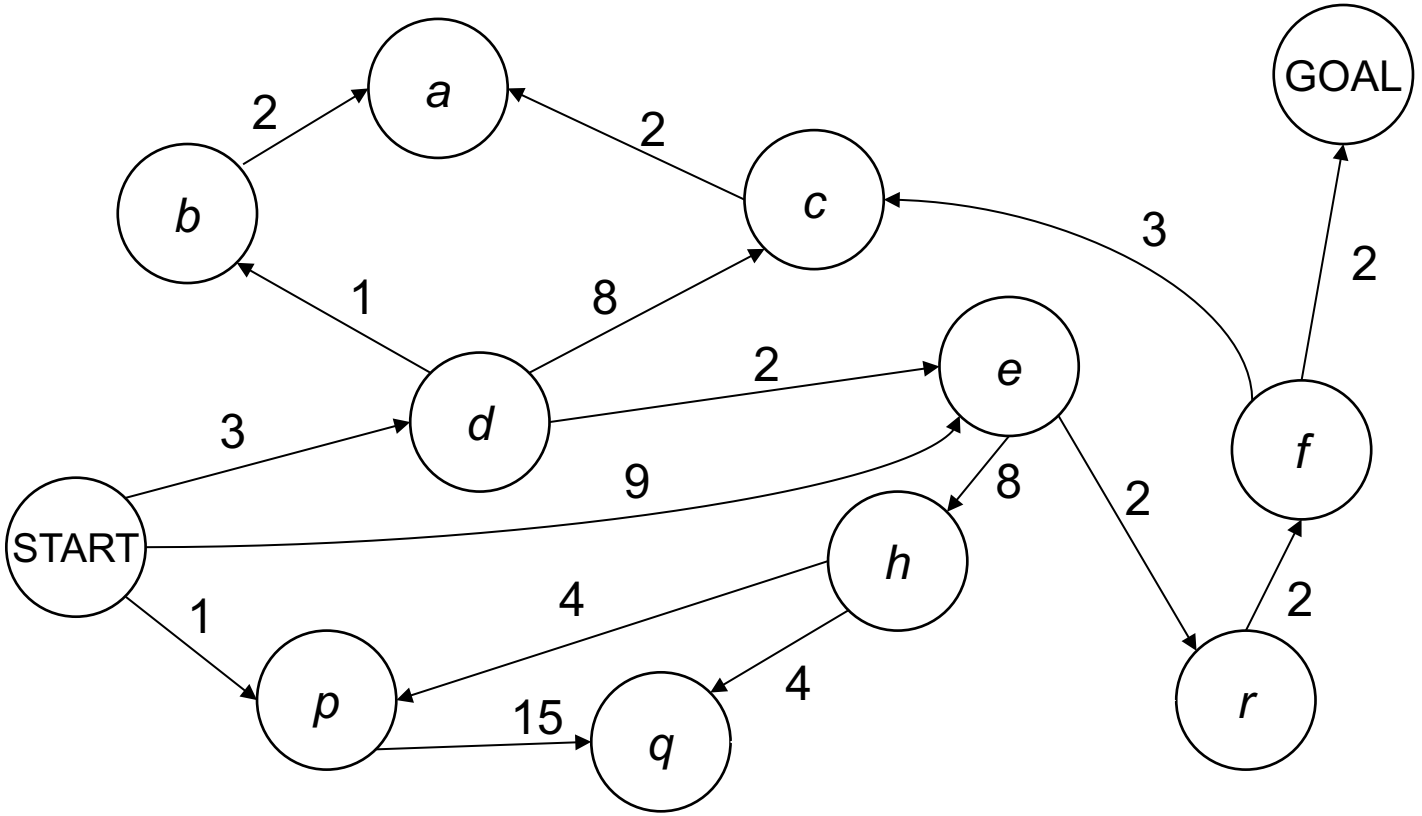
- All these search algorithms are the same except for fringe strategies
  - Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
  - Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues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



Informed Search

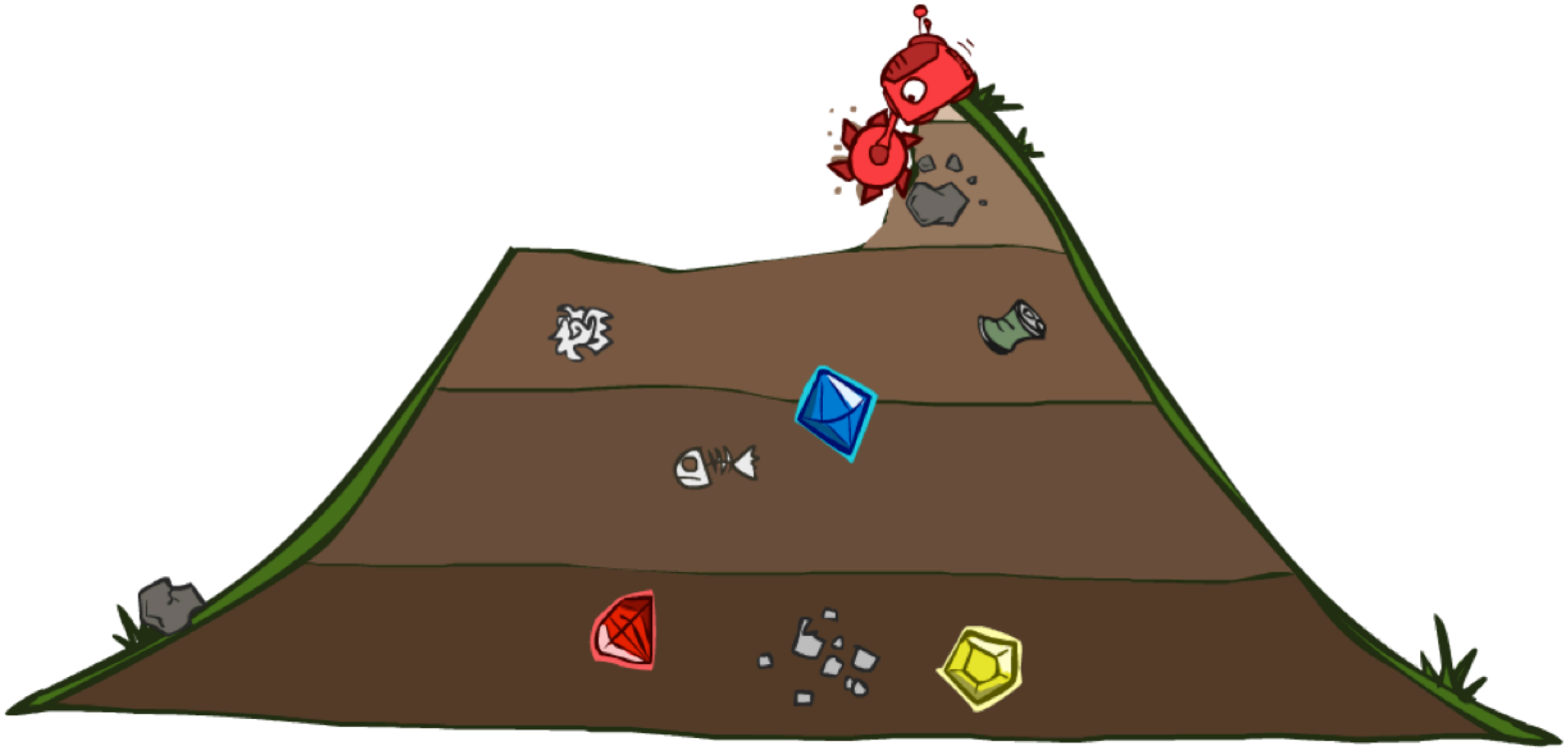


Cost-Sensitive Search



BFS finds the shortest path in terms of number of actions.
It does not find the least-cost path. We will now cover
a similar algorithm which does find the least-cost path.

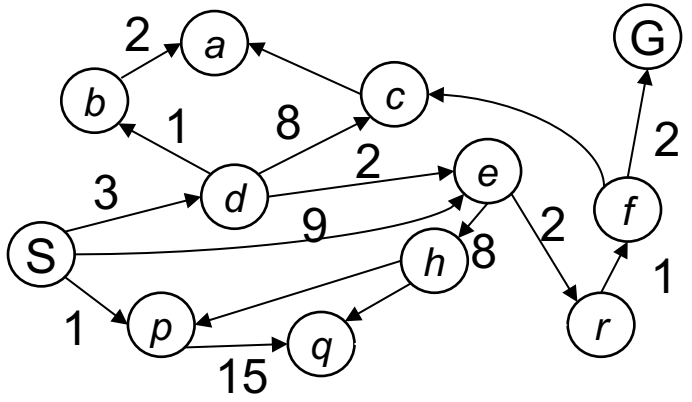
Uniform Cost Search



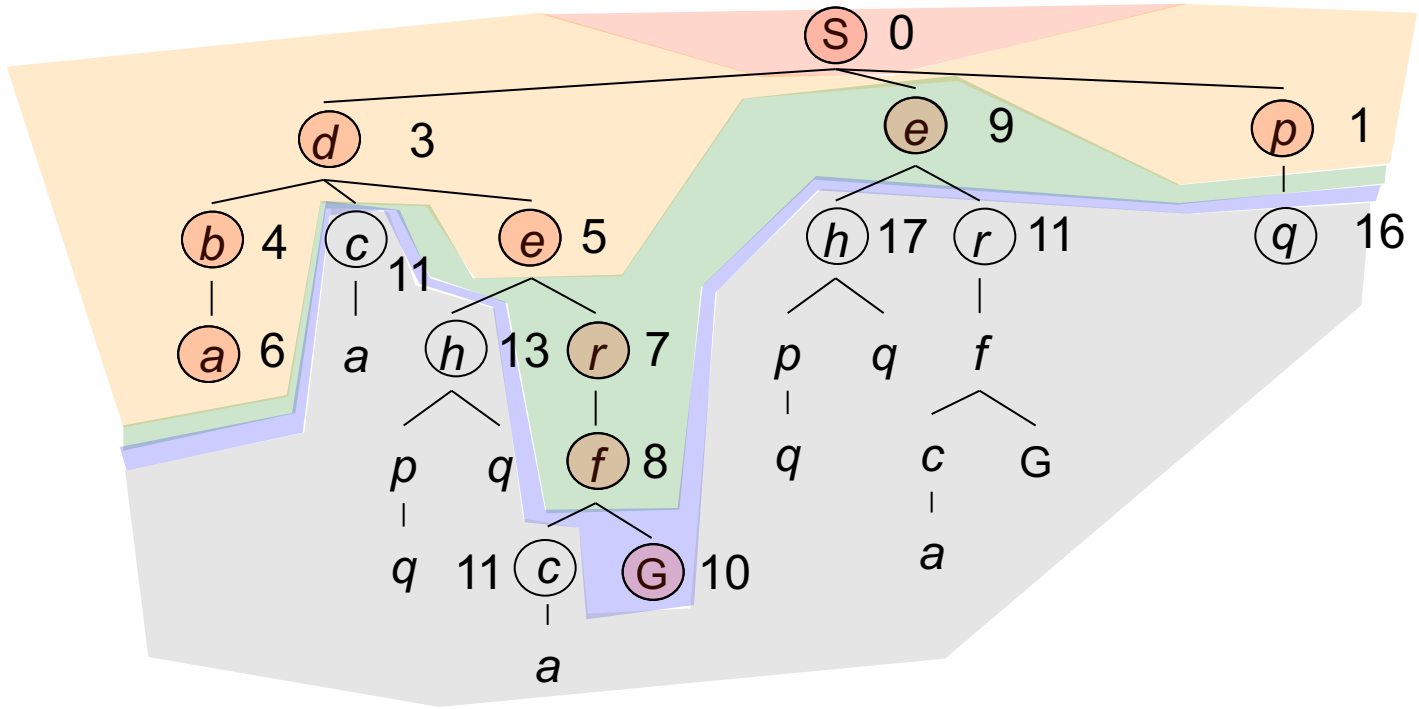
Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)

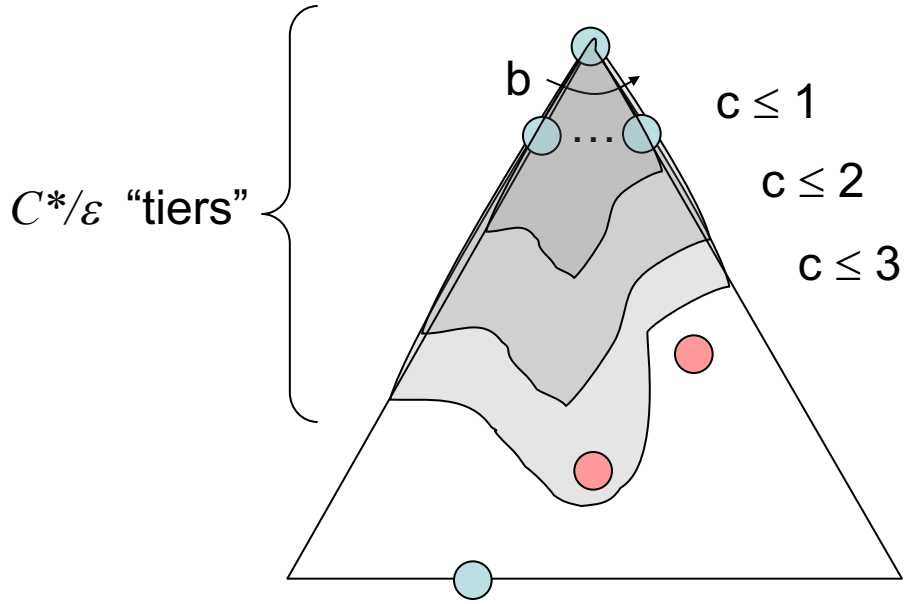


Cost contours



Uniform Cost Search (UCS) Properties

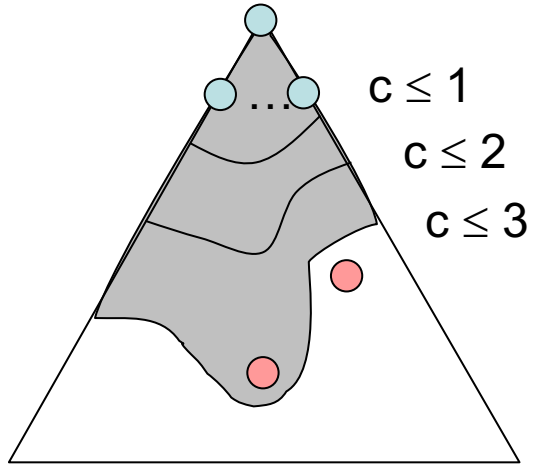
- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)



- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!
- Is it optimal?
  - Yes! (Proof next lecture via A*)

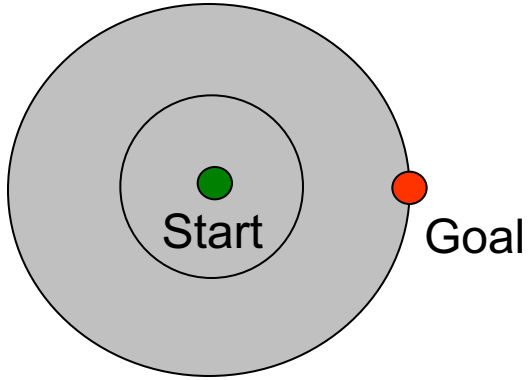
Uniform Cost Issues

- Remember: UCS explores increasing cost contours



- The good: UCS is complete and optimal!

- The bad:
  - Explores options in every “direction”
  - No information about goal location

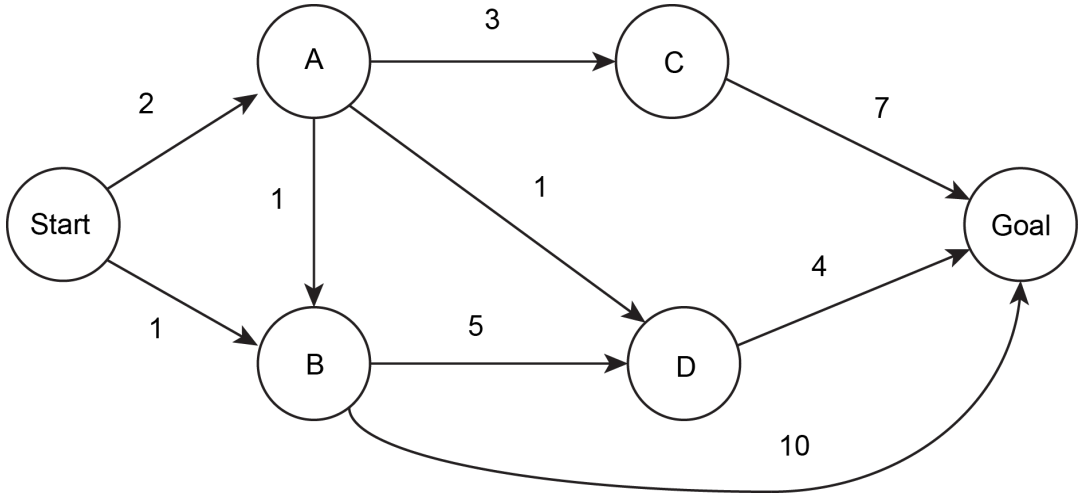


- We’ll fix that soon!

[Demo: empty grid UCS (L2D5)]
[Demo: maze with deep/shallow water DFS/BFS/UCS (L2D7)]

Exercise

Consider the graph on right hand side. Arcs are labeled with their weights. Assume that ties are broken alphabetically (so a partial plan S->X->A would be expanded before S->X->B and S->A->Z would be expanded before S->B->A.



Q1: In what order are states expanded by Uniform Cost Search? You may find it helpful to execute the search on scratch paper.

Start, B, A, D, C, Goal

Q2: What path does uniform cost search return?

Start-A-D-Goal

Solution for exercise

Step 1: Expand S
Fringe: (S-A, 2), (S-B, 1)
Closed Set: S

Step 2: Expand S-B
Fringe: (S-A, 2), (S-B-D, 6), (S-B-G, 11)
Closed Set: S, B

Step 3: Expand S-A
Fringe: (S-B-D, 6), (S-B-G, 11), (S-A-D, 3), (S-A-C, 5)
Closed Set: S, B, A

Step 4: Expand S-A-D
Fringe: (S-B-D, 6), (S-B-G, 11), (S-A-C, 5), (S-A-D-G, 7)
Closed Set: S, B, A, D

Step 5: Expand S-A-C
Fringe: (S-B-D, 6), (S-B-G, 11), (S-A-D-G, 7), (S-A-C-G, 13)
Closed Set: S, B, A, D, C

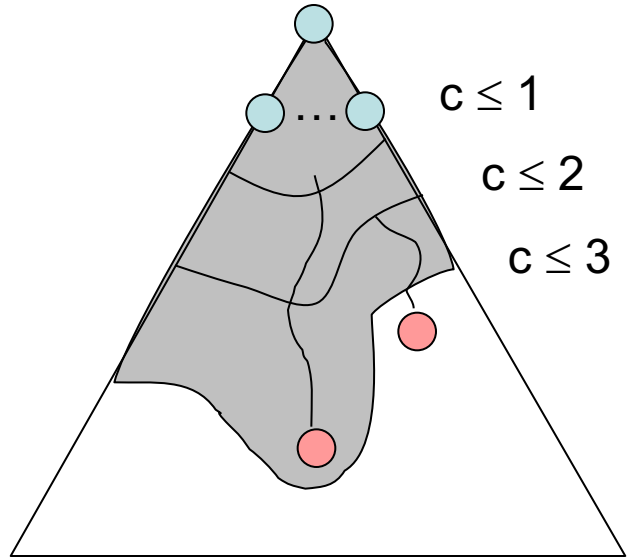
Step 6: Expand S-B-D
Fringe: (S-B-G, 11), (S-A-D-G, 7), (S-A-C-G, 13), (S-B-D-G, 10)
Closed Set: S, B, A, D, C

Step 7: Expand S-A-D-G, finding the goal
Closed Set: S, B, A, D, C, G
Return: Start-A-D-Goal

Uninformed Search

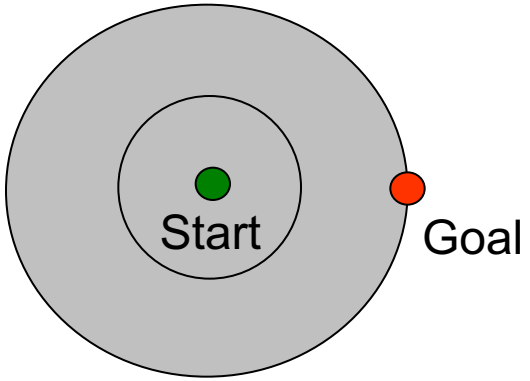
Uniform Cost Search

- Strategy: expand lowest path cost



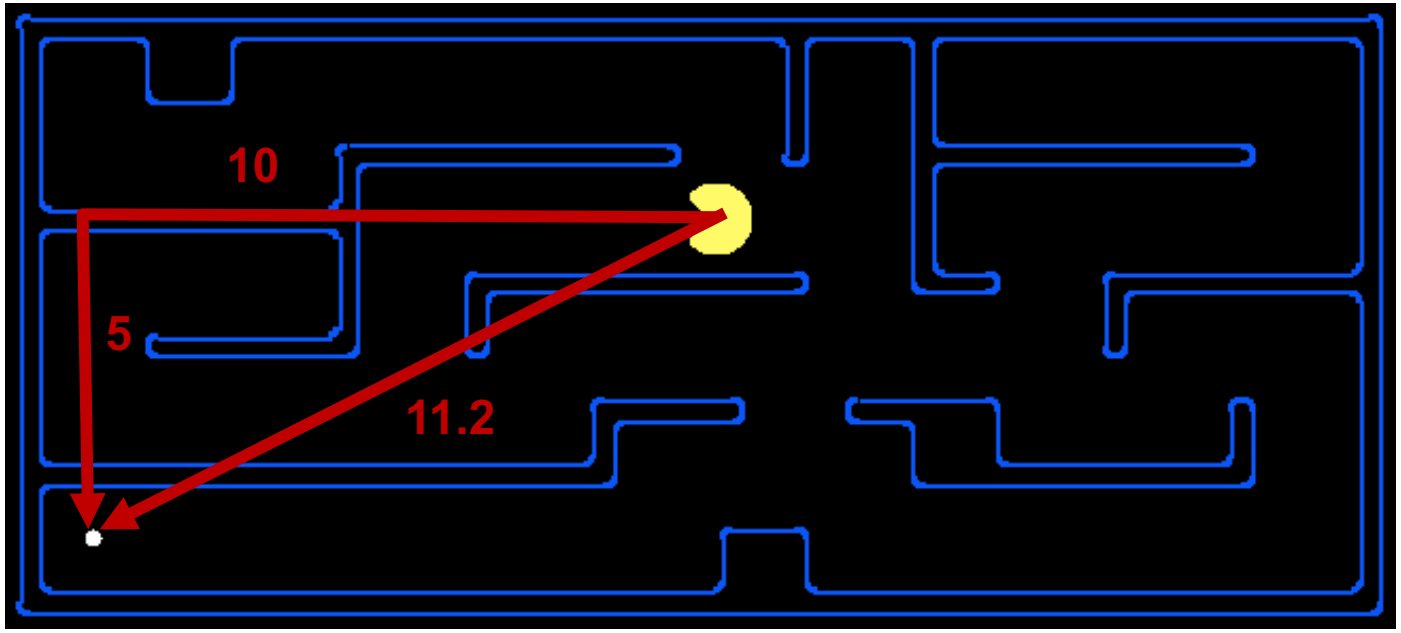
- The good: UCS is complete and optimal!

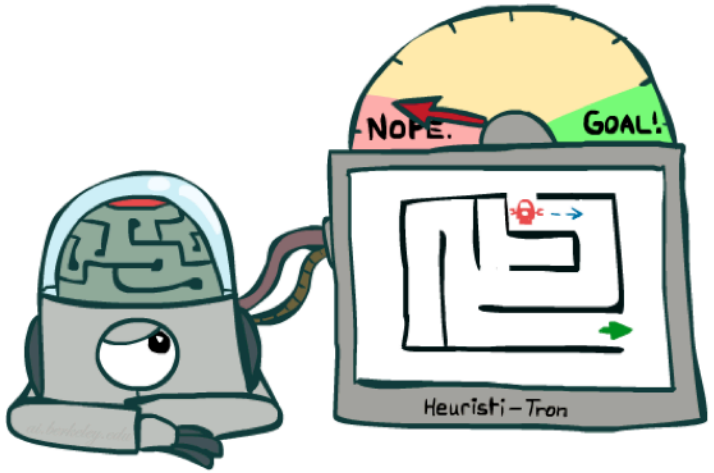
- The bad:
  - Explores options in every “direction”
  - No information about goal location

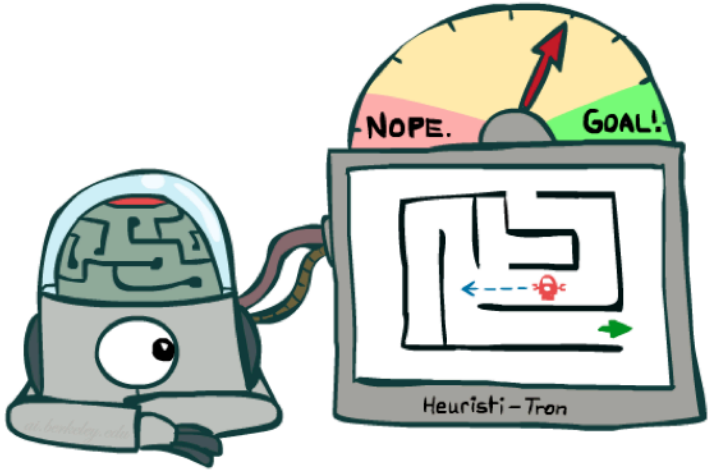


Search Heuristics

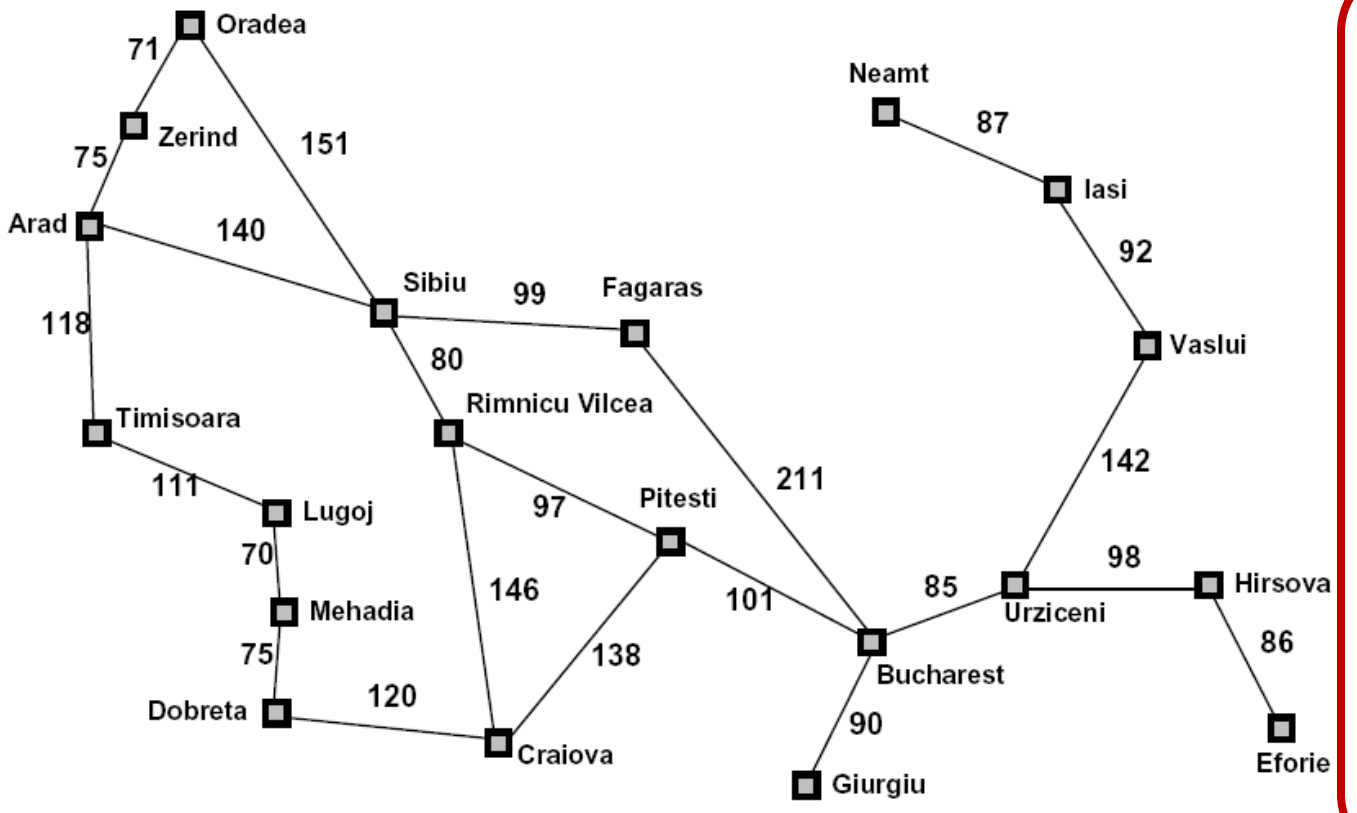
- A heuristic is:
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing







Example: Heuristic Function

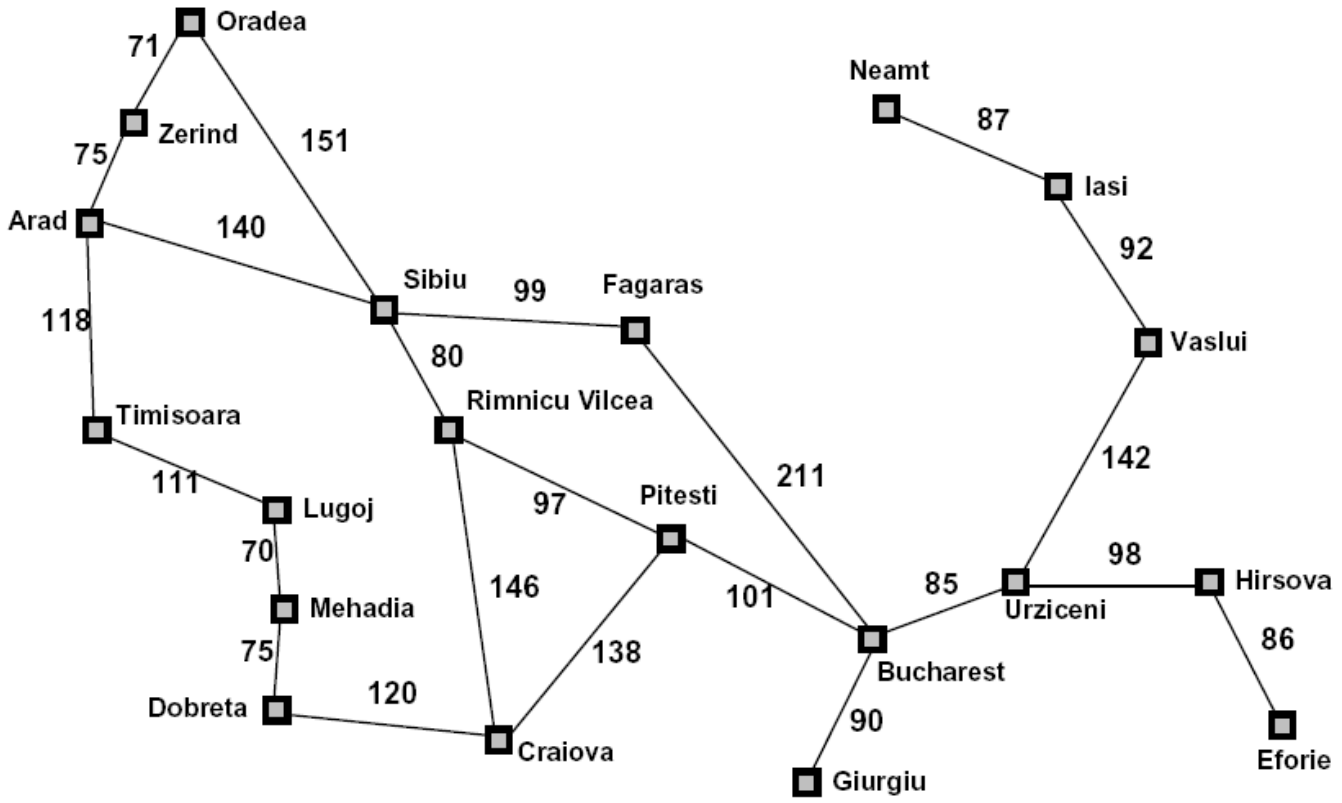


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

Greedy Search

Example: Heuristic Function

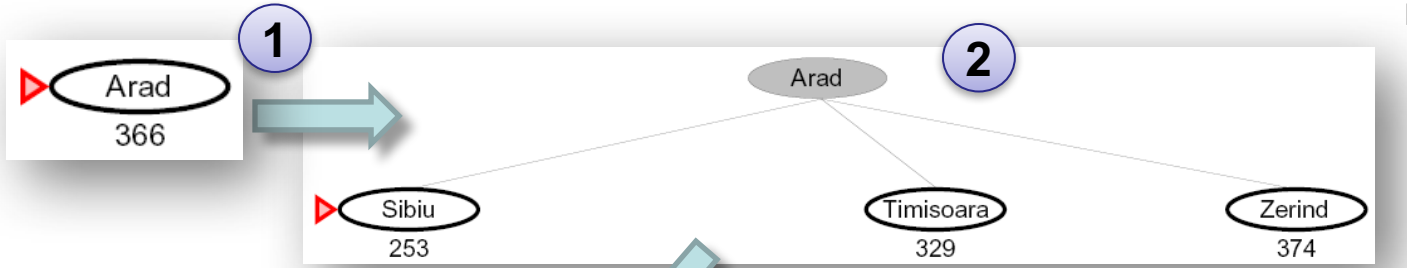


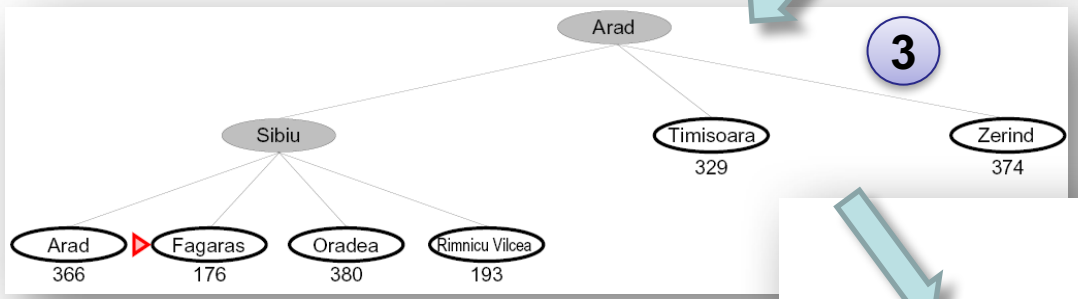
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

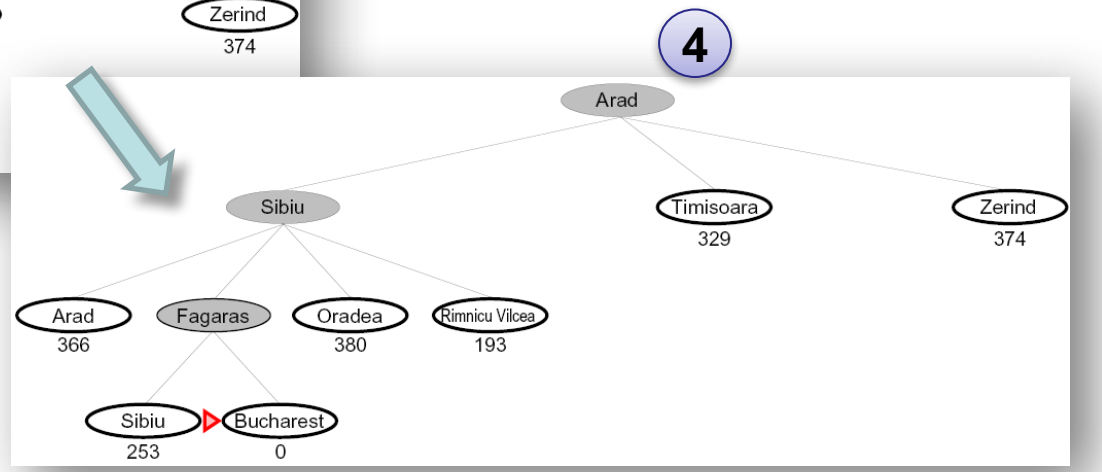
h(x)

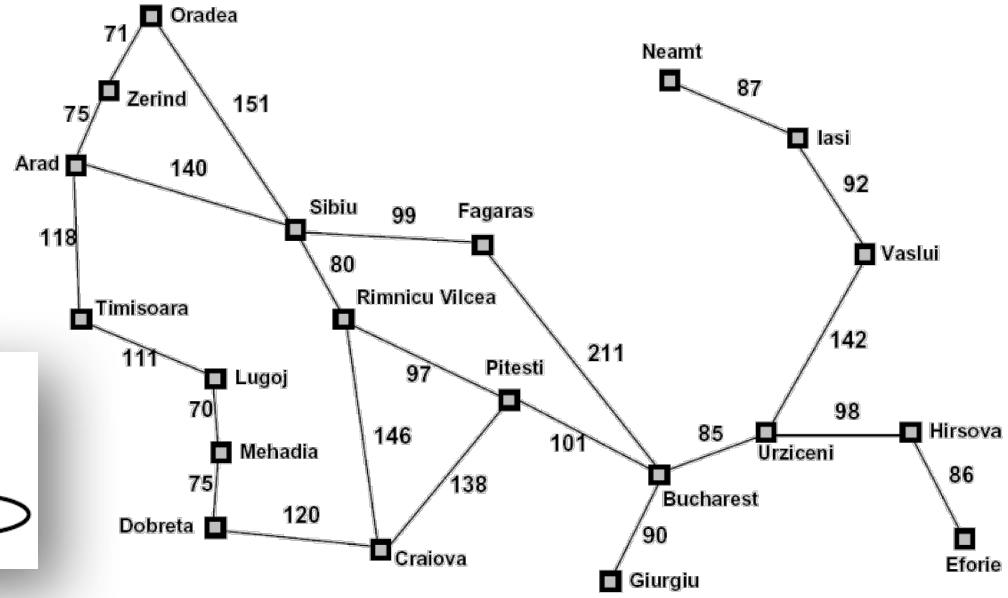
Greedy Search

- Expand the node that seems closest…

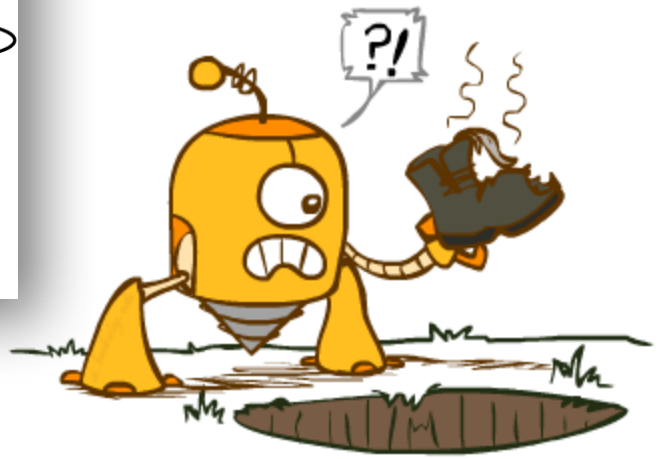






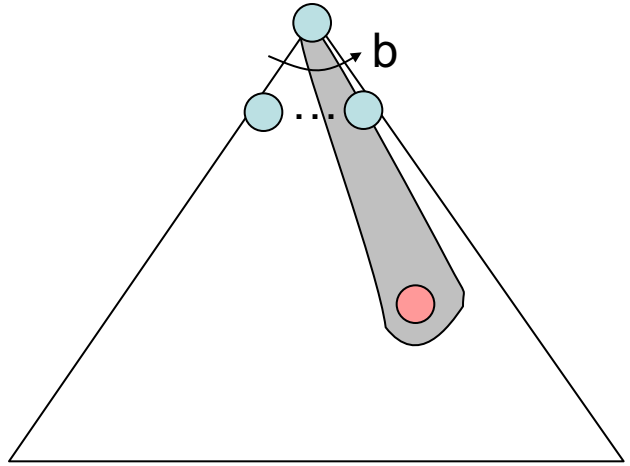


- What can go wrong?

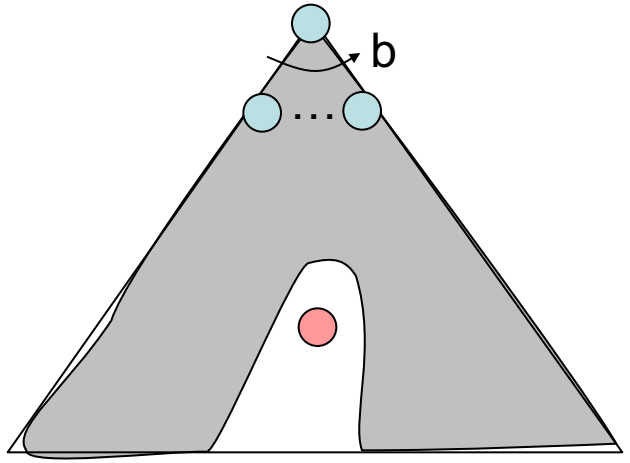


Greedy Search

- 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)]

A* Search

A* Search

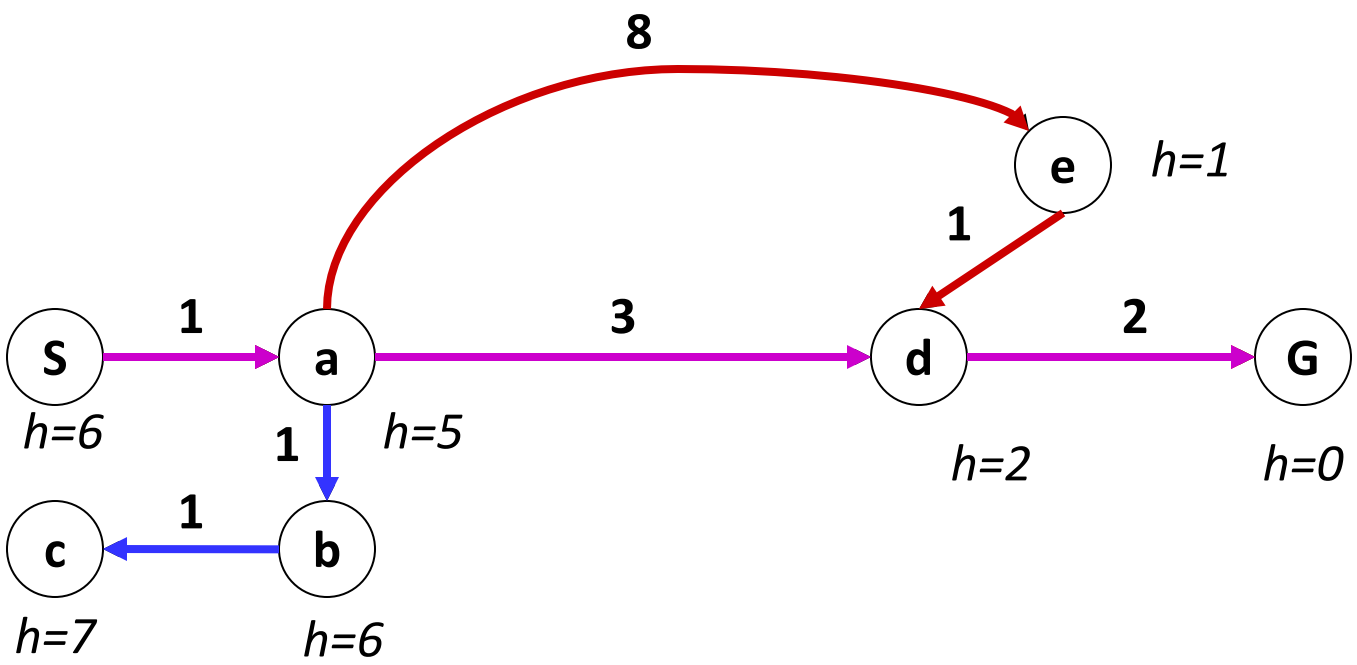
UCS

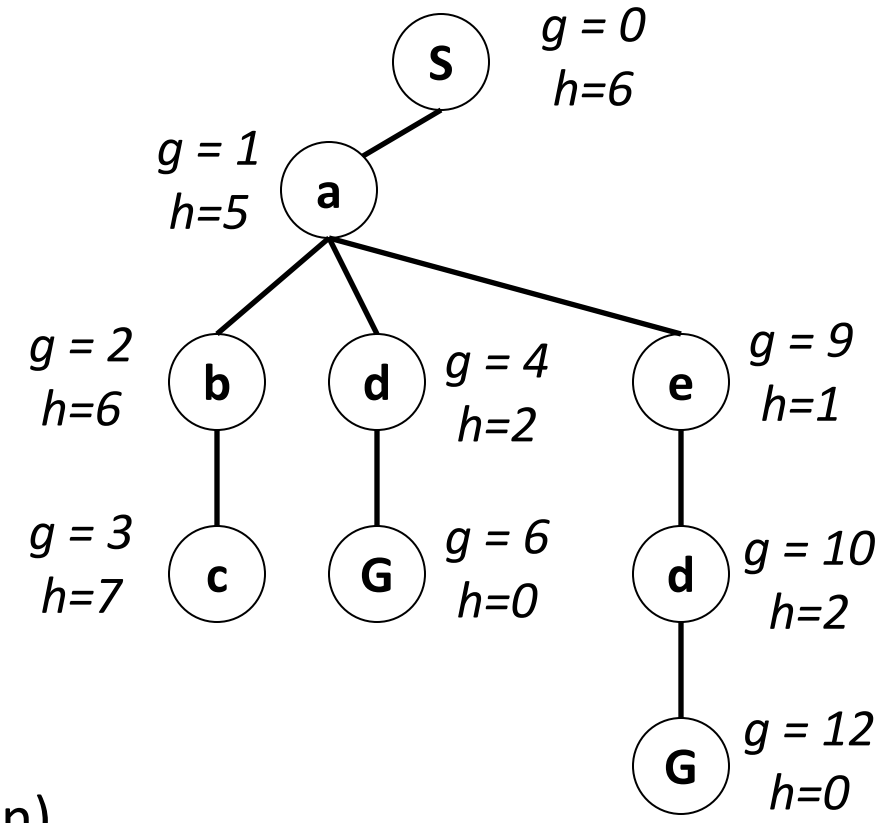
Greedy

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)



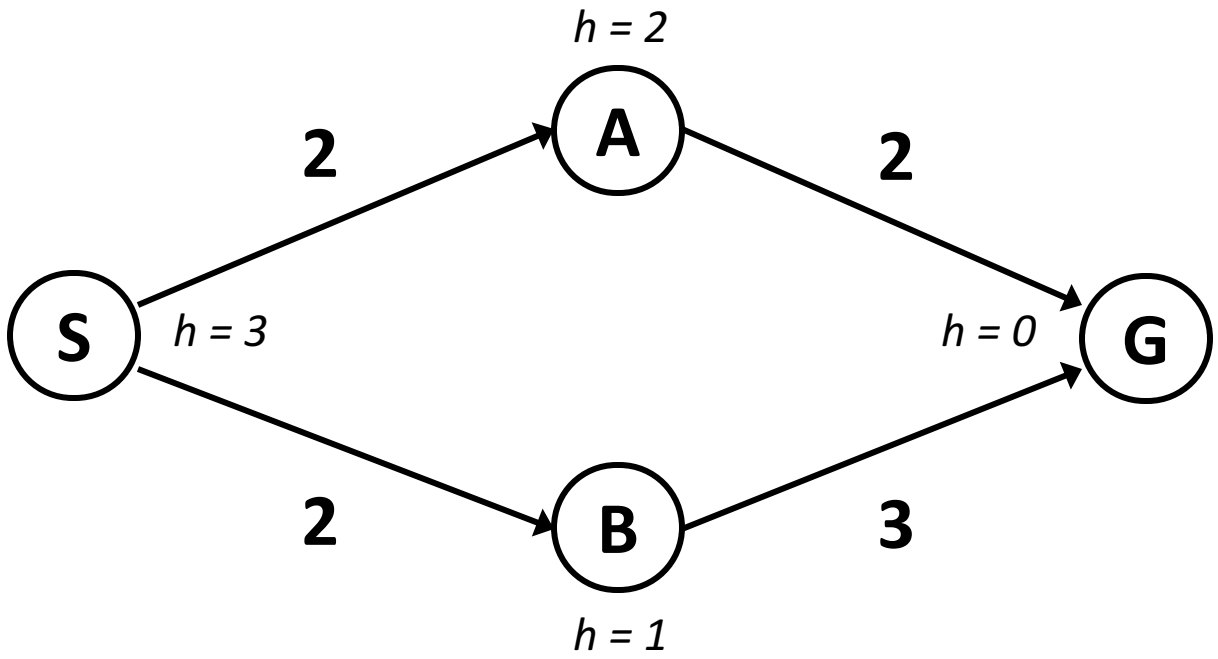


- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

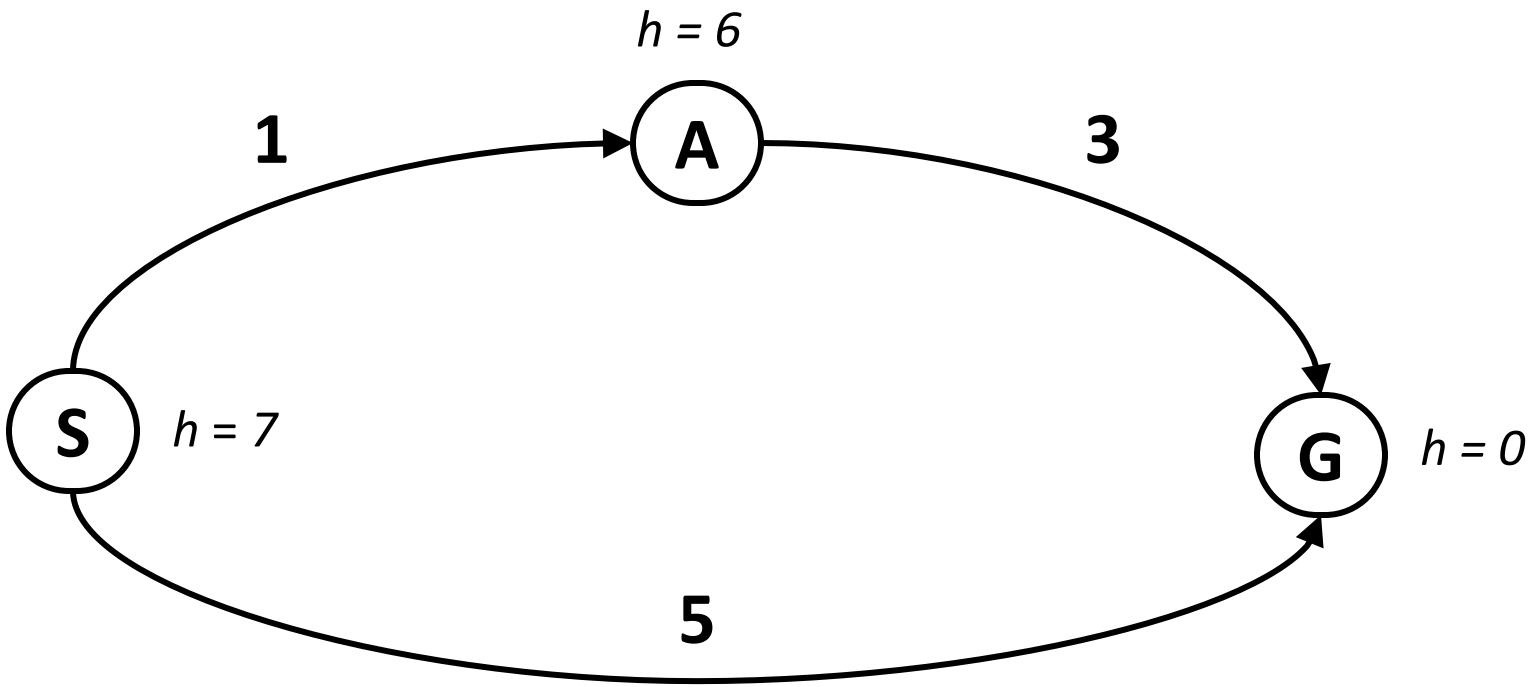
When should A* terminate?

- Should we stop when we enqueue a goal?



- No: only stop when we dequeue a goal

Is A* Optimal?



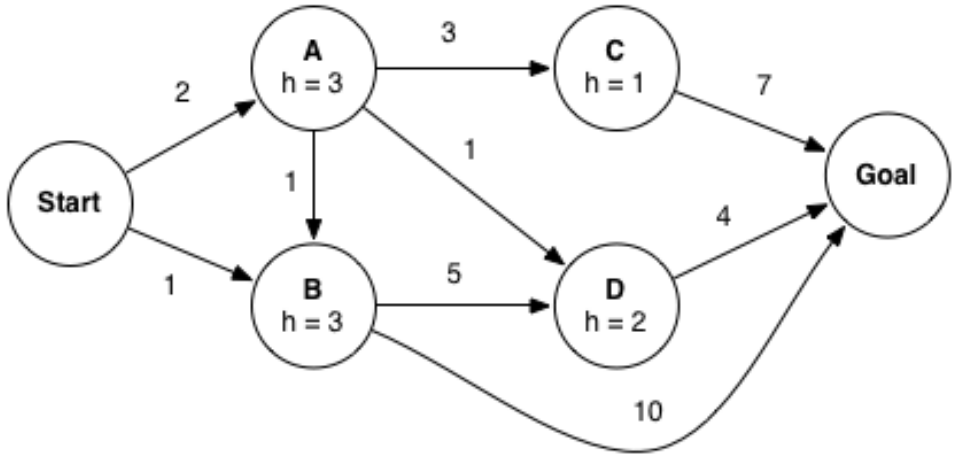
- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

Exercise

Consider A* graph search on the graph below. Arcs are labeled with action costs and states are labeled with heuristic values. Assume that ties are broken alphabetically (so a partial plan S->X->A would be expanded before S->X->B and S->A->Z would be expanded before S->B->A).

In what order are states expanded by A* graph search? You may find it helpful to execute the search on scratch paper.

Start, B, A, D, C, Goal



What path does A* graph search return?

Start-A-D-Goal

Solution to exercise

Step 1: Expand S
Fringe: (S-A, 5), (S-B, 4)
Closed Set: S

Step 2: Expand S-B
Fringe: (S-A, 5), (S-B-D, 8), (S-B-G, 11)
Closed Set: S, B

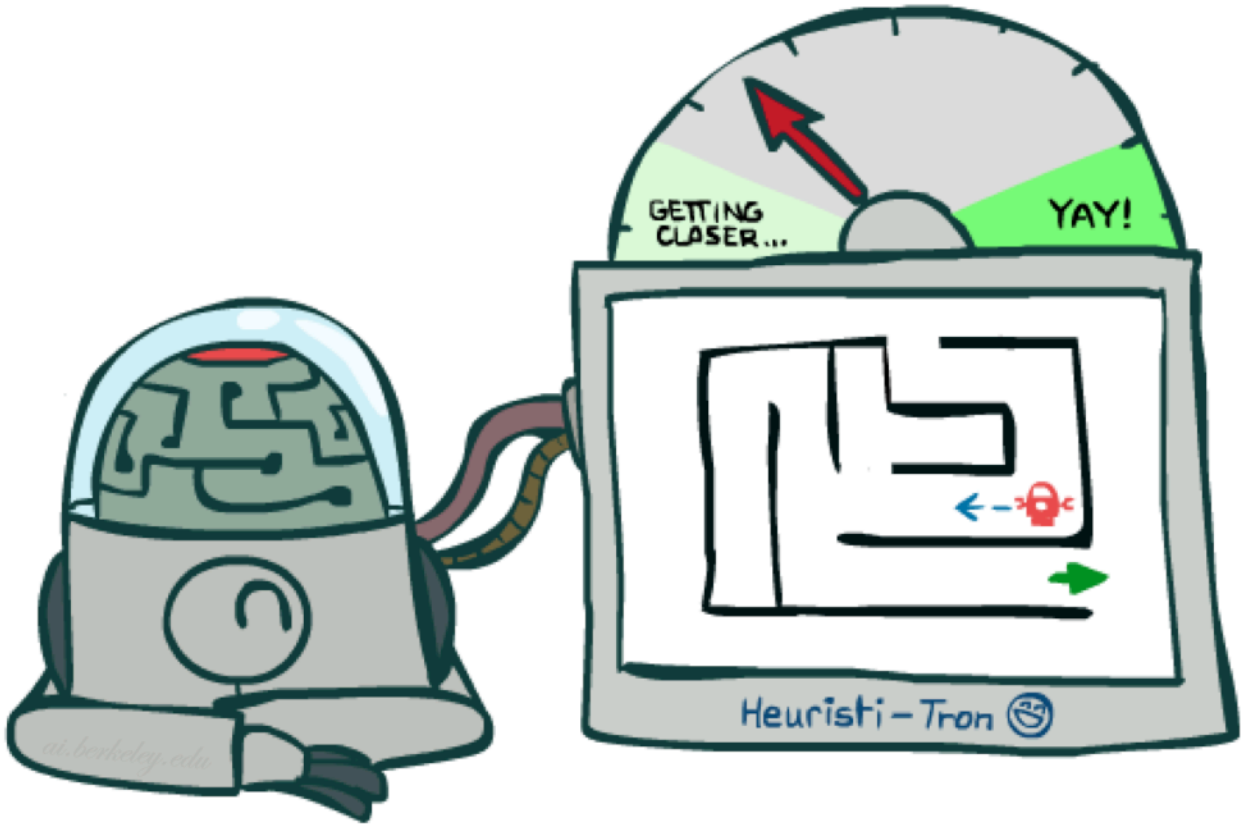
Step 3: Expand S-A
Fringe: (S-B-D, 8), (S-B-G, 11), (S-A-D, 5), (S-A-C, 6)
Closed Set: S, B, A

Step 4: Expand S-A-D
Fringe: (S-B-D, 8), (S-B-G, 11), (S-A-C, 6), (S-A-D-G, 7)
Closed Set: S, B, A, D

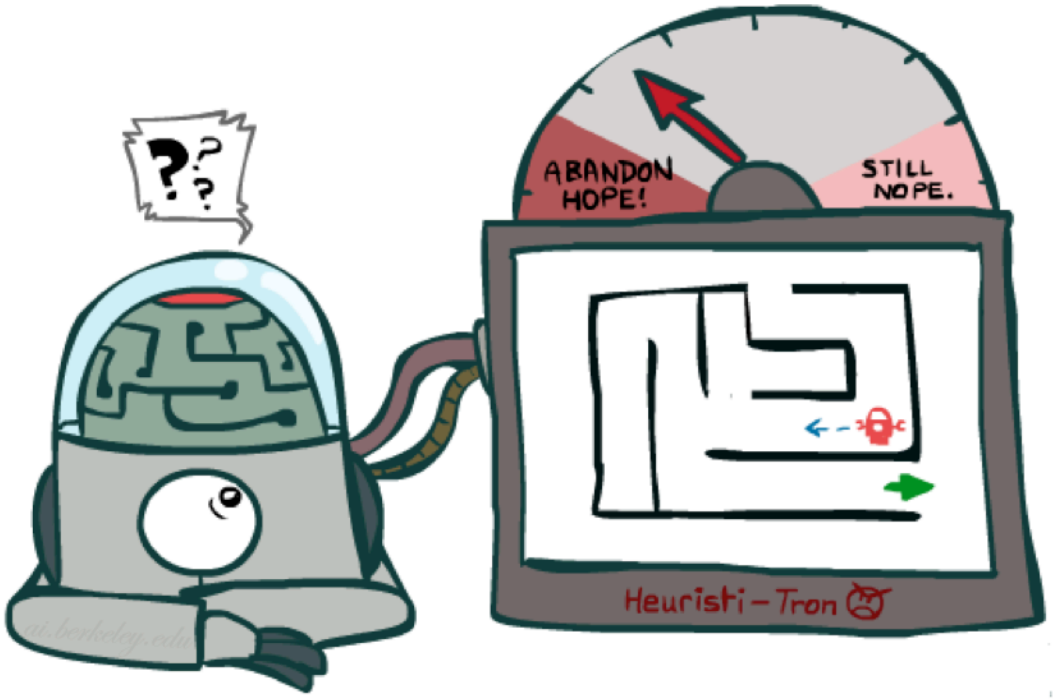
Step 5: Expand S-A-C
Fringe: (S-B-D, 8), (S-B-G, 11), (S-A-D-G, 7), (S-A-C-G, 13)
Closed Set: S, B, A, D, C

Step 6: Expand S-A-D-G, finding the goal
Closed Set: S, B, A, D, C, G
Return: Start-A-D-Goal

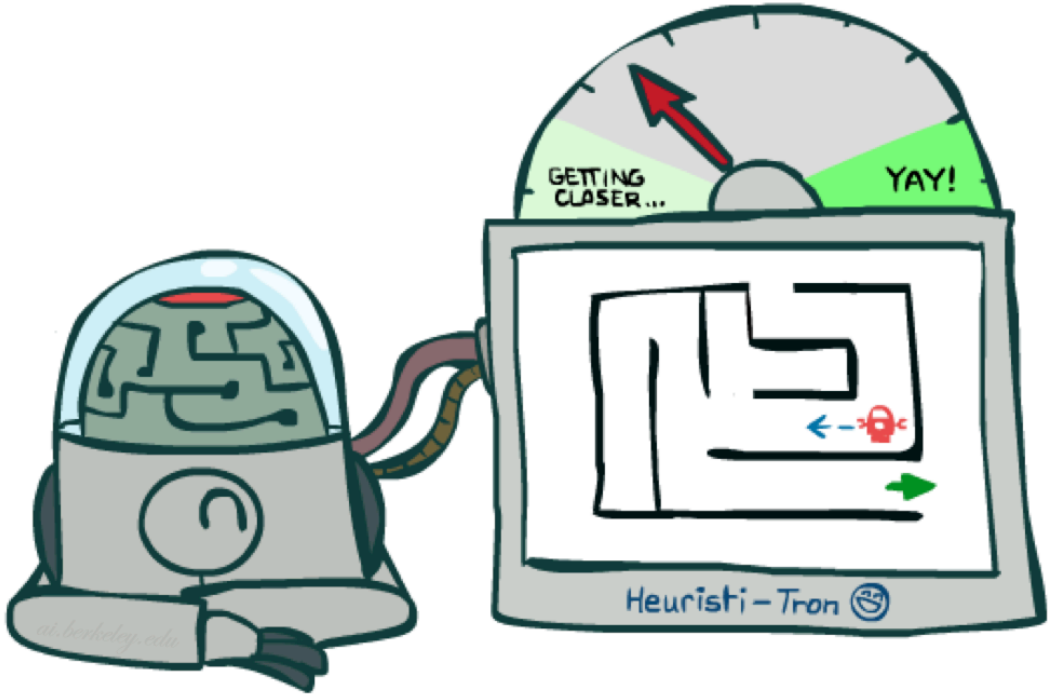
Admissible Heuristics



Idea: Admissibility



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



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

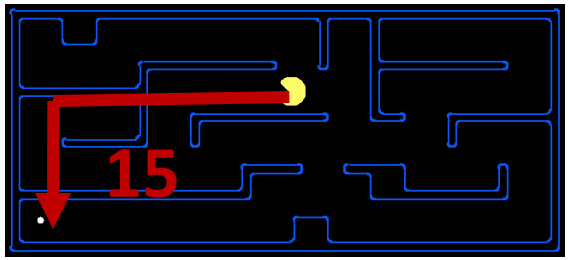
Admissible Heuristics

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

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

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

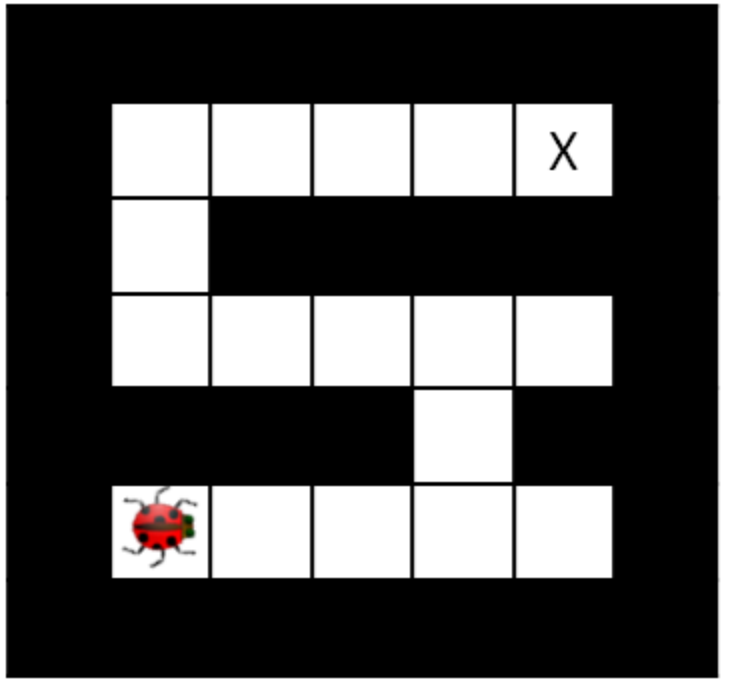
- Examples:



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

Exercise

You control a single insect as shown in the maze below, which must reach a designated target location X, also known as the hive. There are no other insects moving around.



Which of the following is a minimal correct state space representation?

An integer d encoding the Manhattan distance to the hive.

A tuple (x,y) encoding the x and y coordinates of the insect.

A tuple (x,y,d) encoding the insect's x and y coordinates as well as the Manhattan distance to the hive.

What is the size of the state space? MN

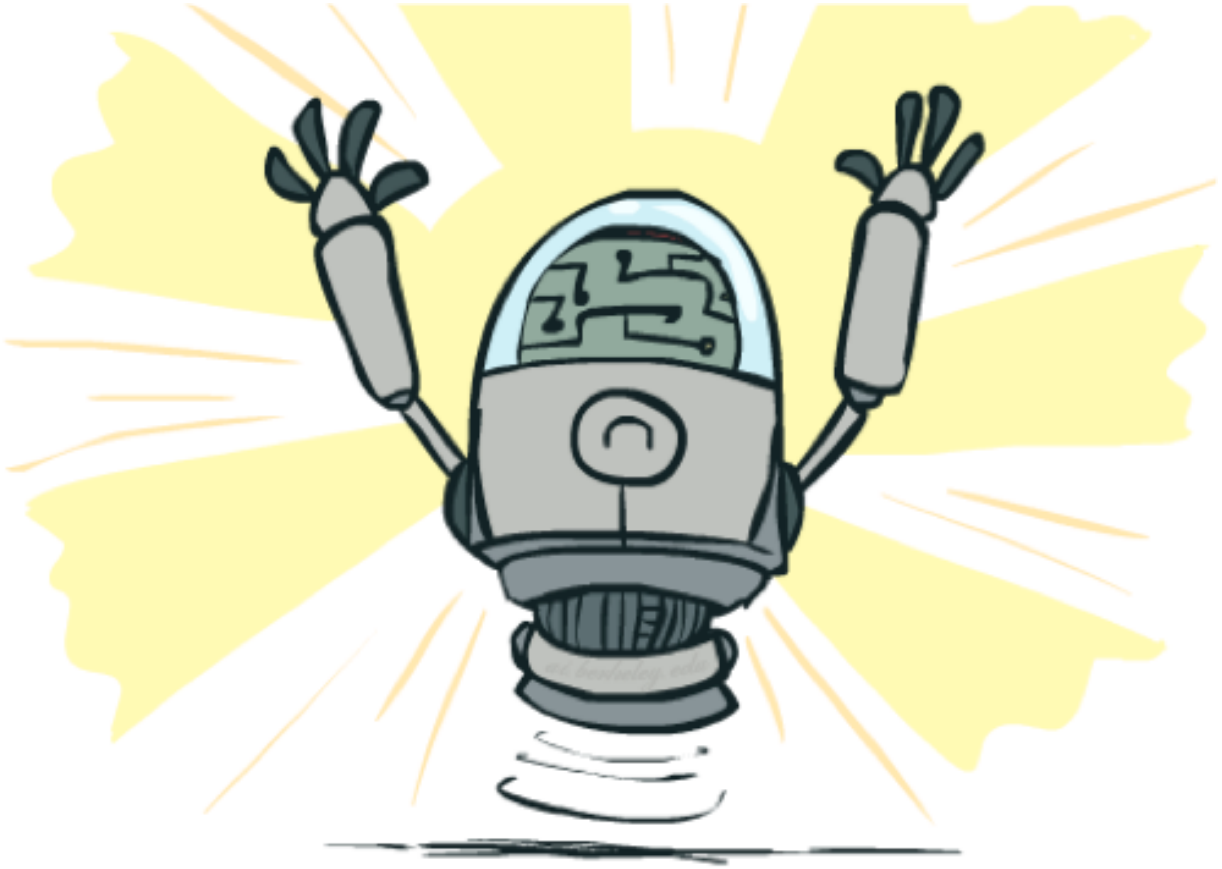
Which of the following heuristics are admissible (if any)?

Manhattan distance from the insect's location to the hive.

Number of steps taken by the insect.

Euclidean distance from the insect's location to the hive.

Optimality of A* Tree Search



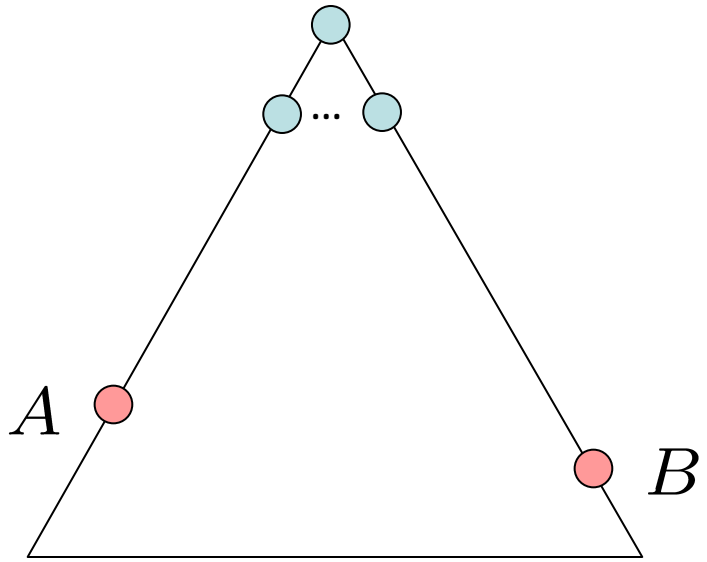
Optimality of A* Tree Search

Assume:

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

Claim:

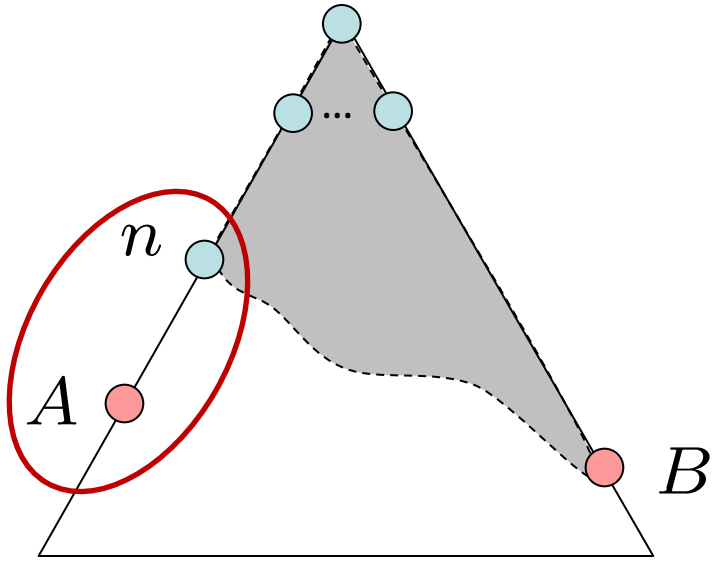
- A will exit the fringe before B



Optimality of A* Tree Search: Blocking

Proof:

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

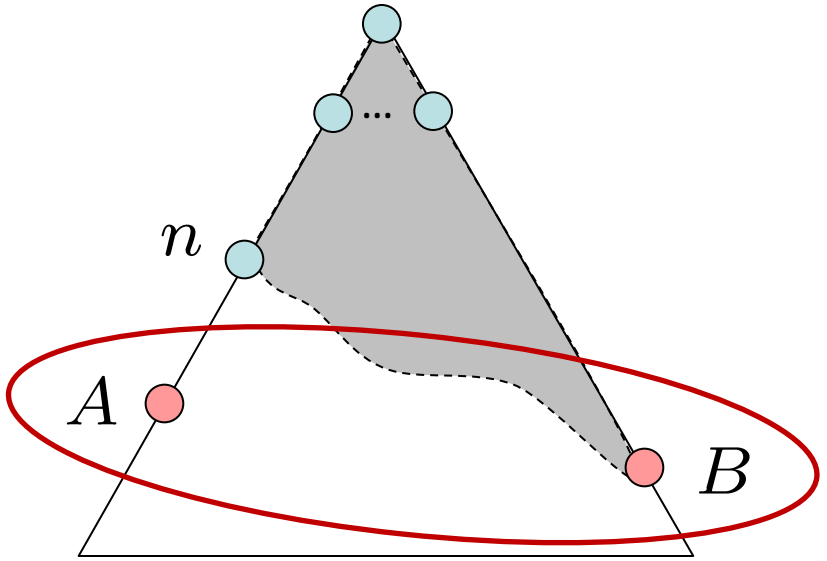


$$f(n) = g(n) + h(n) \quad \text{Definition of f-cost}$$
$$f(n) \leq g(A) \quad \text{Admissibility of h}$$
$$g(A) = f(A) \quad \text{h = 0 at a goal}$$

Optimality of A* Tree Search: Blocking

Proof:

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



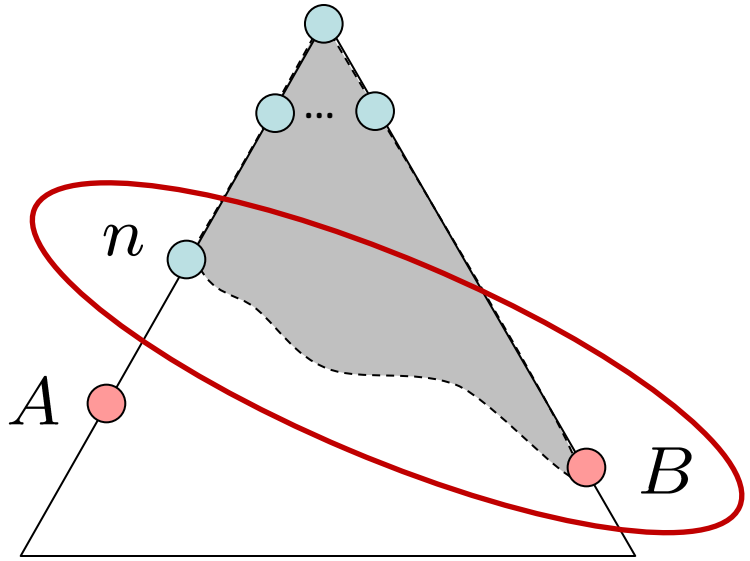
$g(A) < g(B)$ B is suboptimal

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

Optimality of A* Tree Search: Blocking

Proof:

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

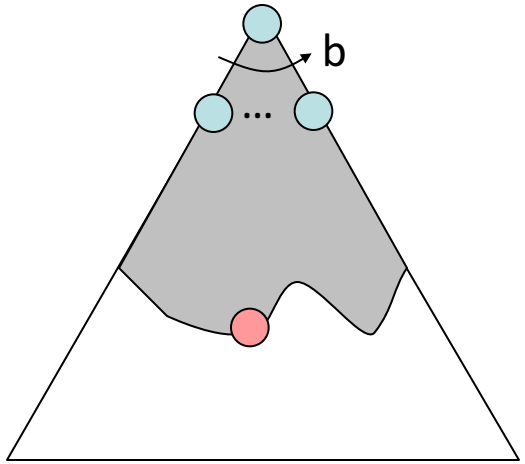


$$f(n) \leq f(A) < f(B)$$

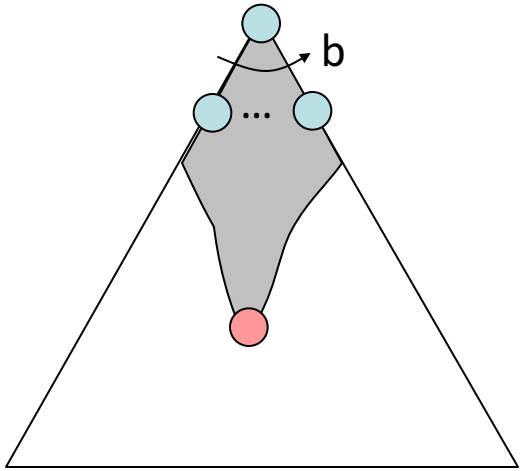
Properties of A*

Properties of A*

Uniform-Cost

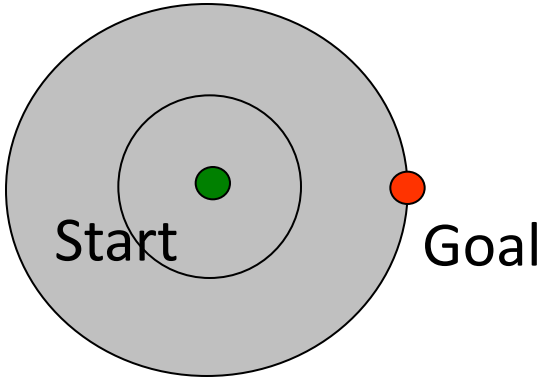


A*

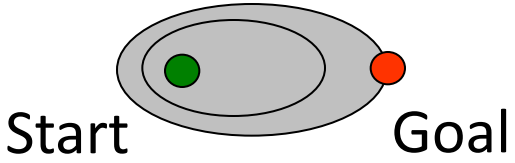


UCS vs A* Contours

- Uniform-cost expands equally in all "directions"

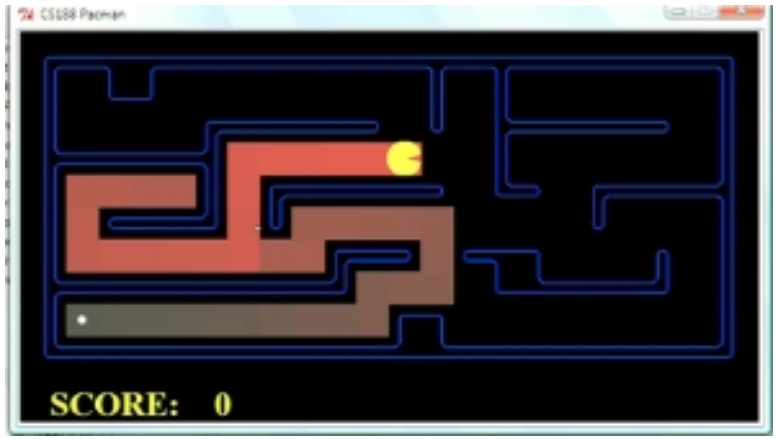


- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



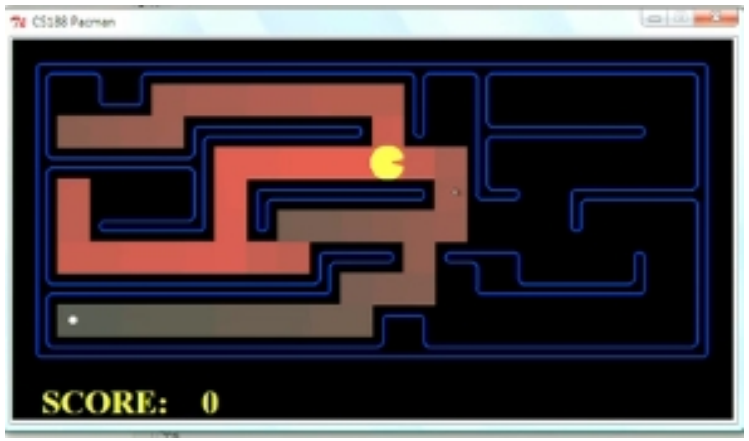
[Demo: contours UCS / greedy / A* empty (L3D1)]
[Demo: contours A* pacman small maze (L3D5)]

Comparison



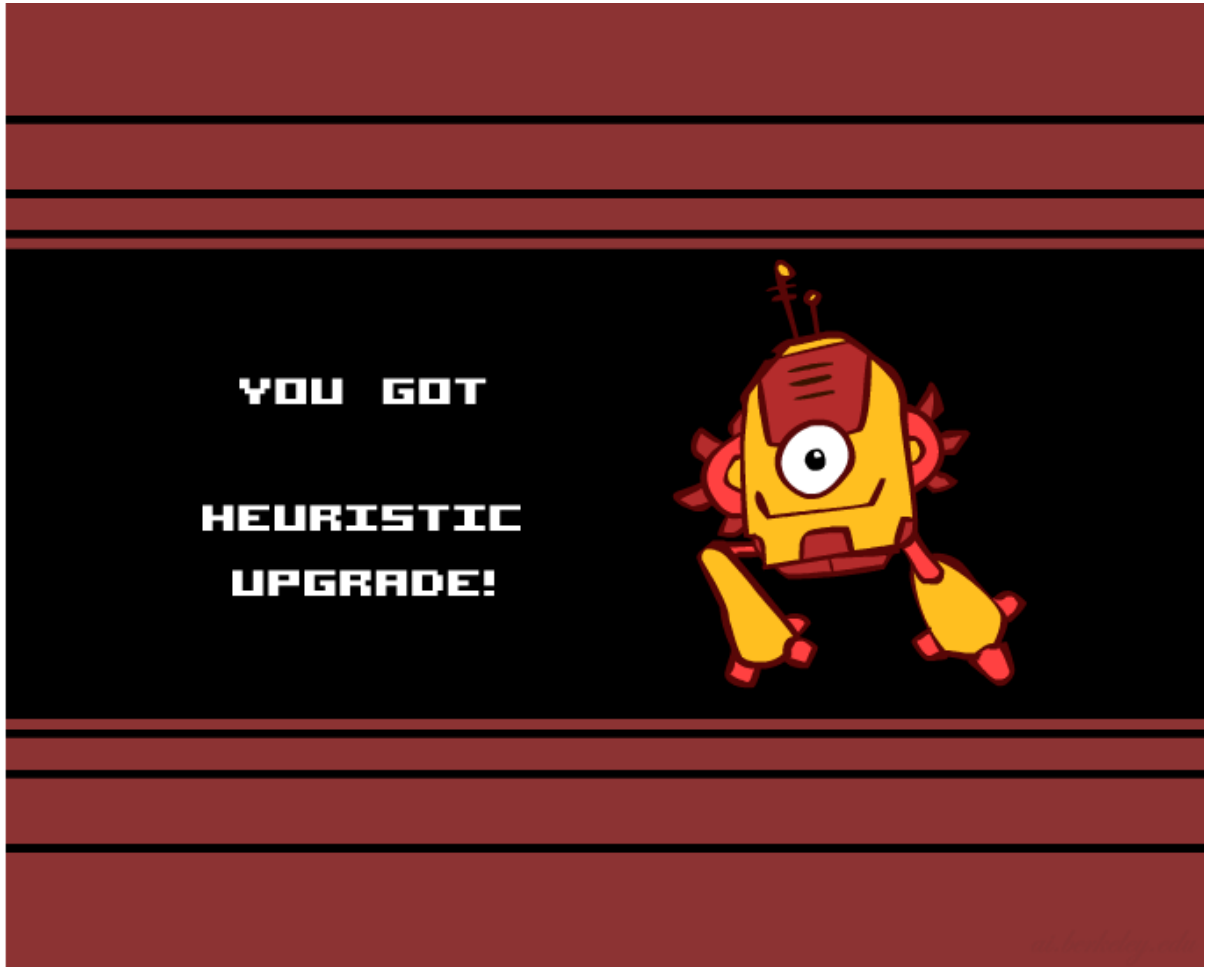
Greedy

Uniform Cost



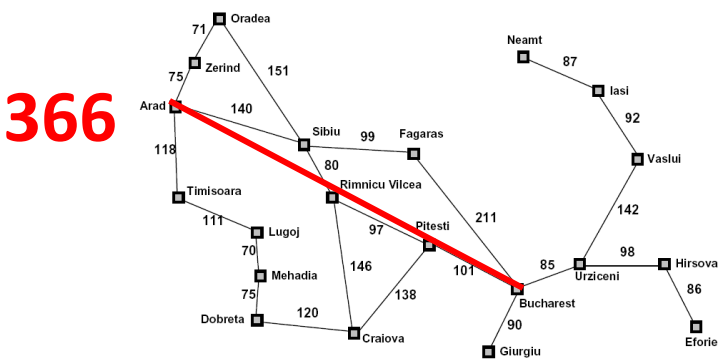
A*

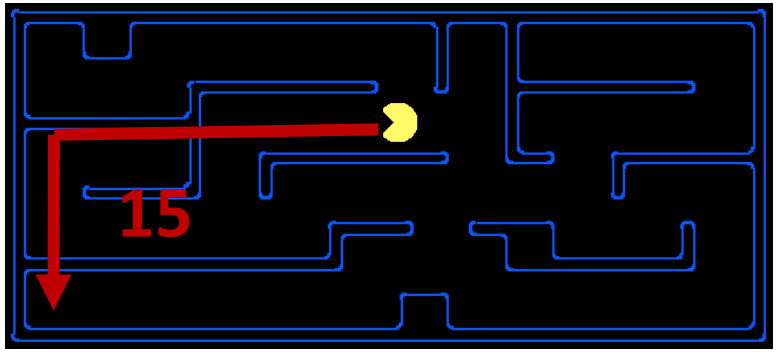
Creating Heuristics



Creating Admissible Heuristics

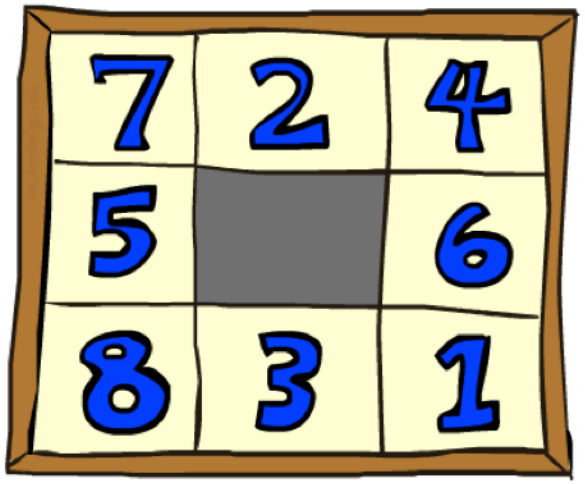
- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available



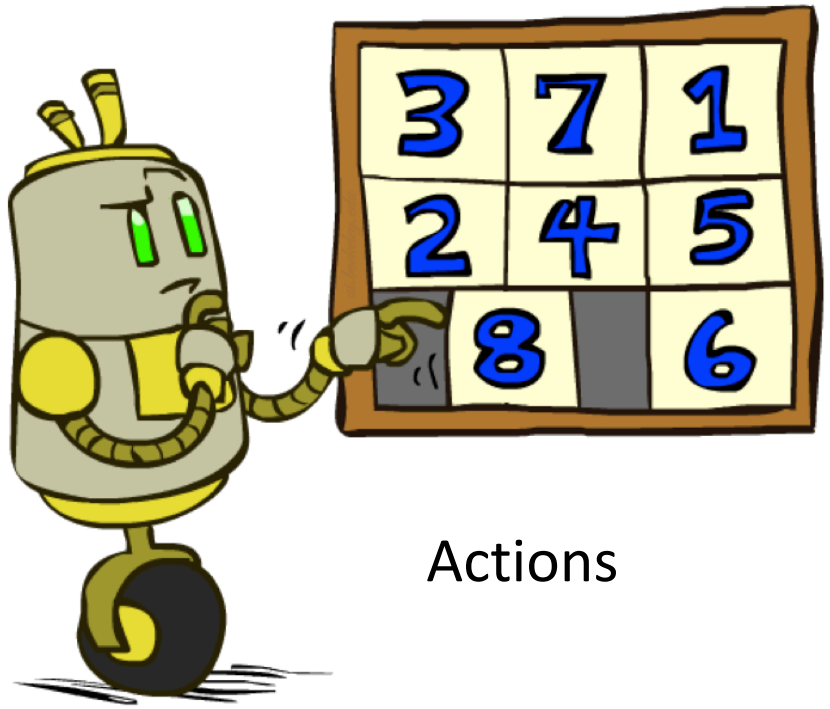


- Inadmissible heuristics are often useful too

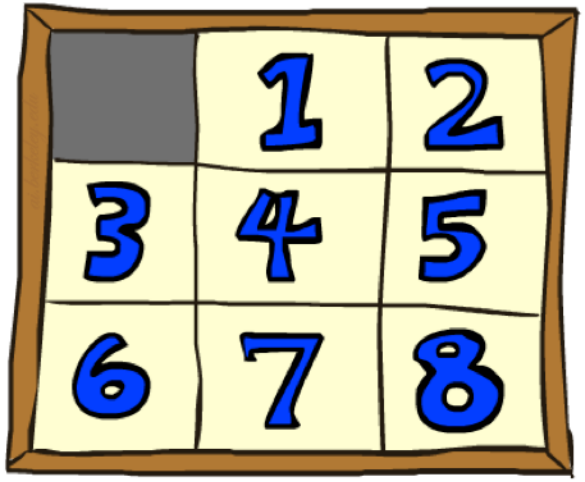
Example: 8 Puzzle



Start State



Actions

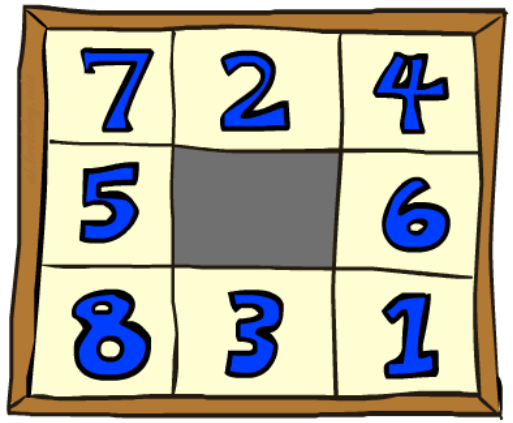


Goal State

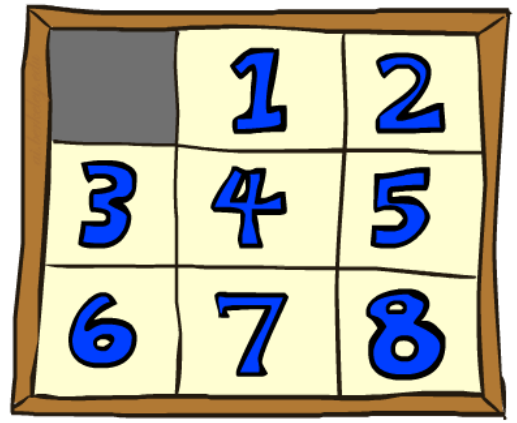
- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

8 Puzzle I

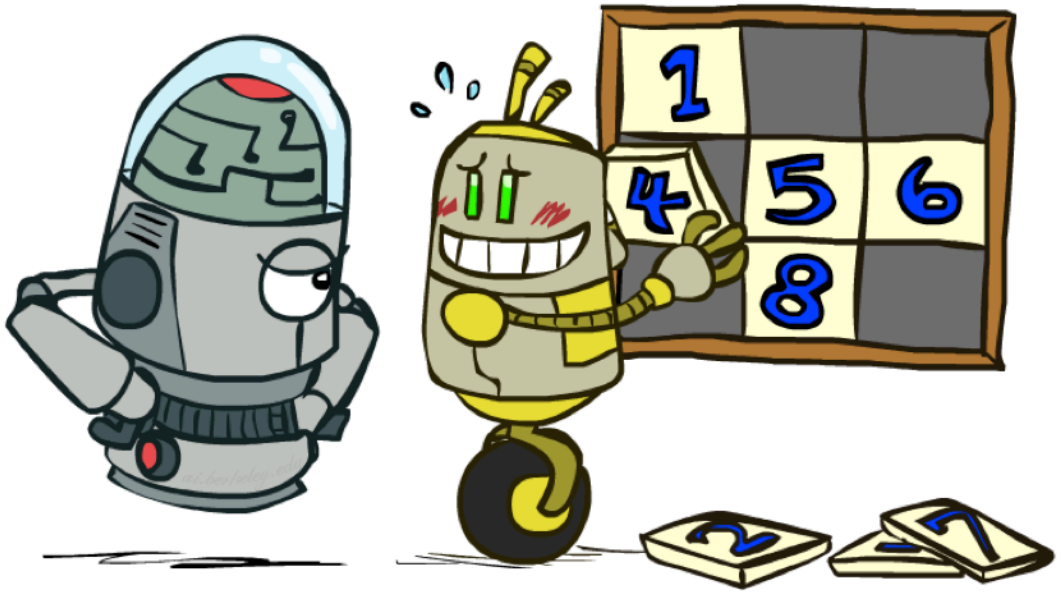
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8
- This is a *relaxed-problem* heuristic



Start State



Goal State

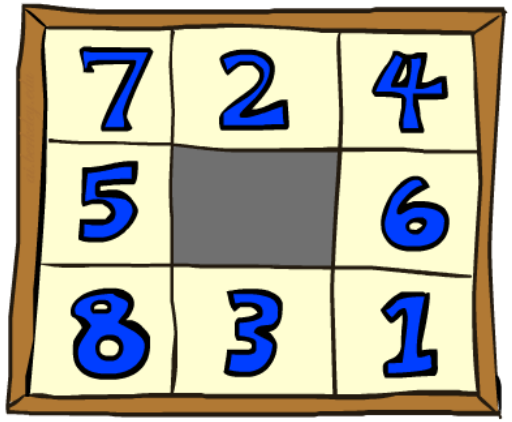


	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

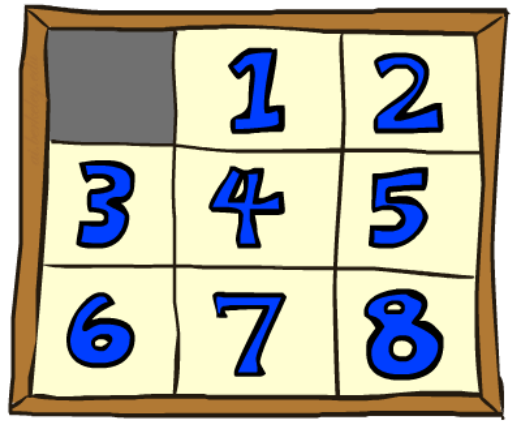
Statistics from Andrew Moore

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 + … = 18



Start State

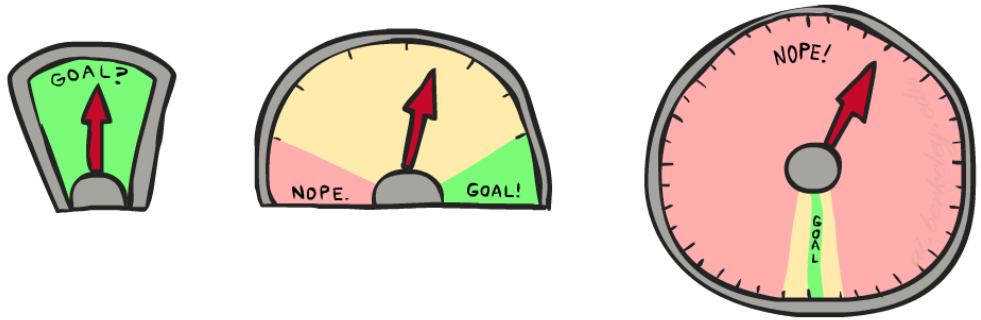


Goal State

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

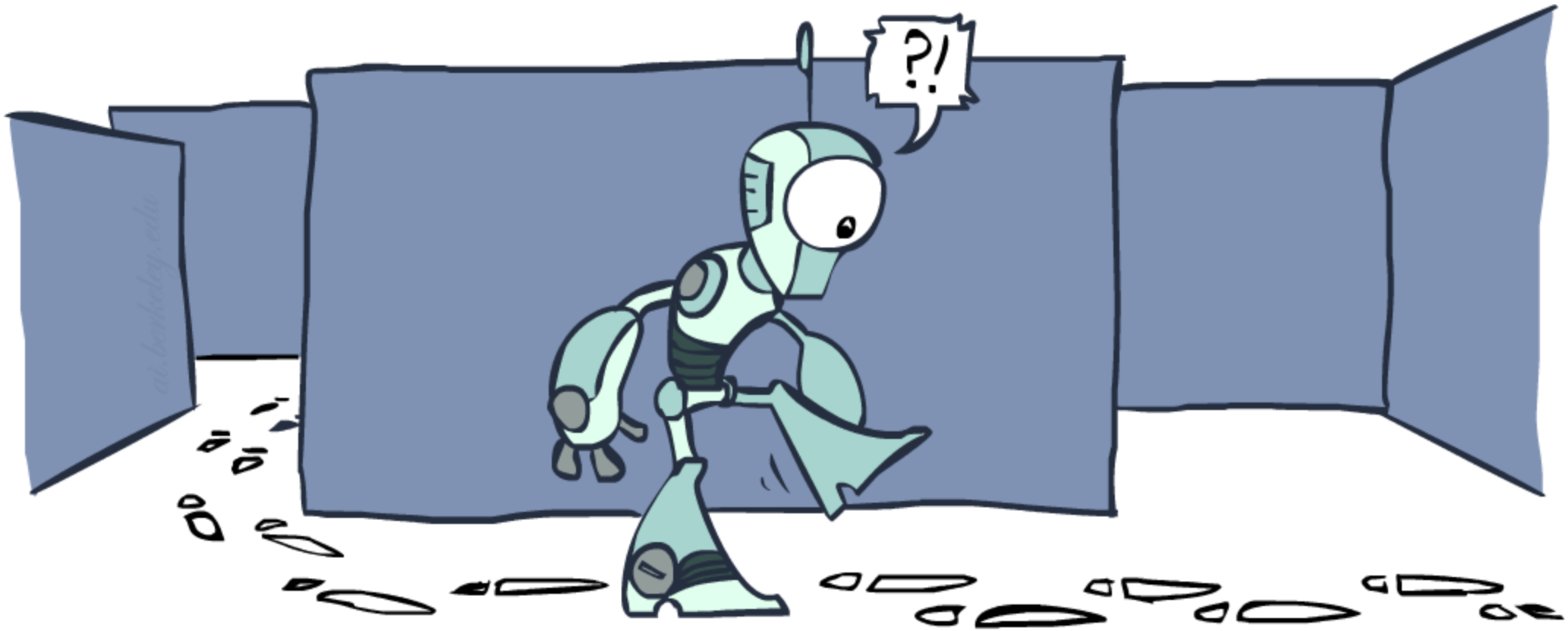
8 Puzzle III

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?



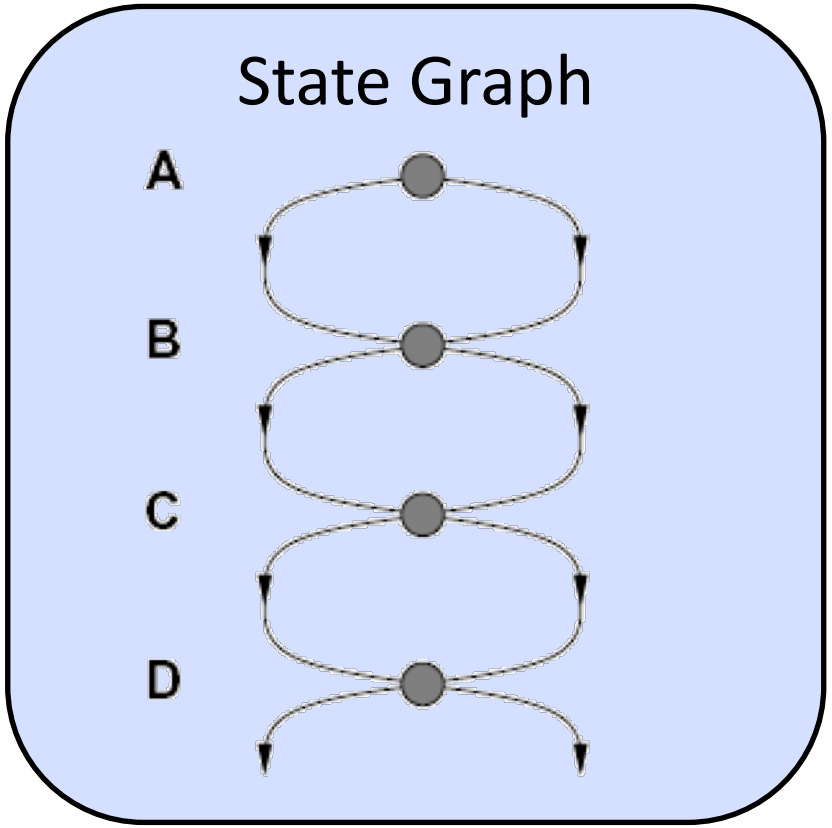
- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

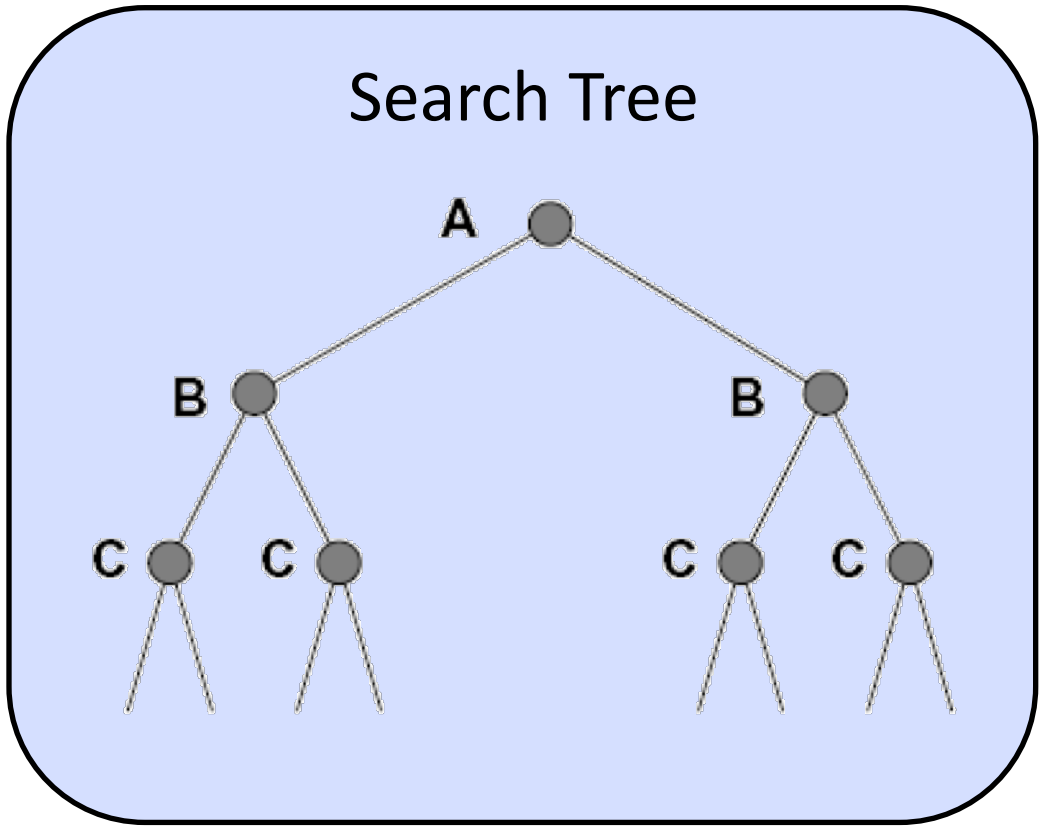
Graph Search



Tree Search: Extra Work!

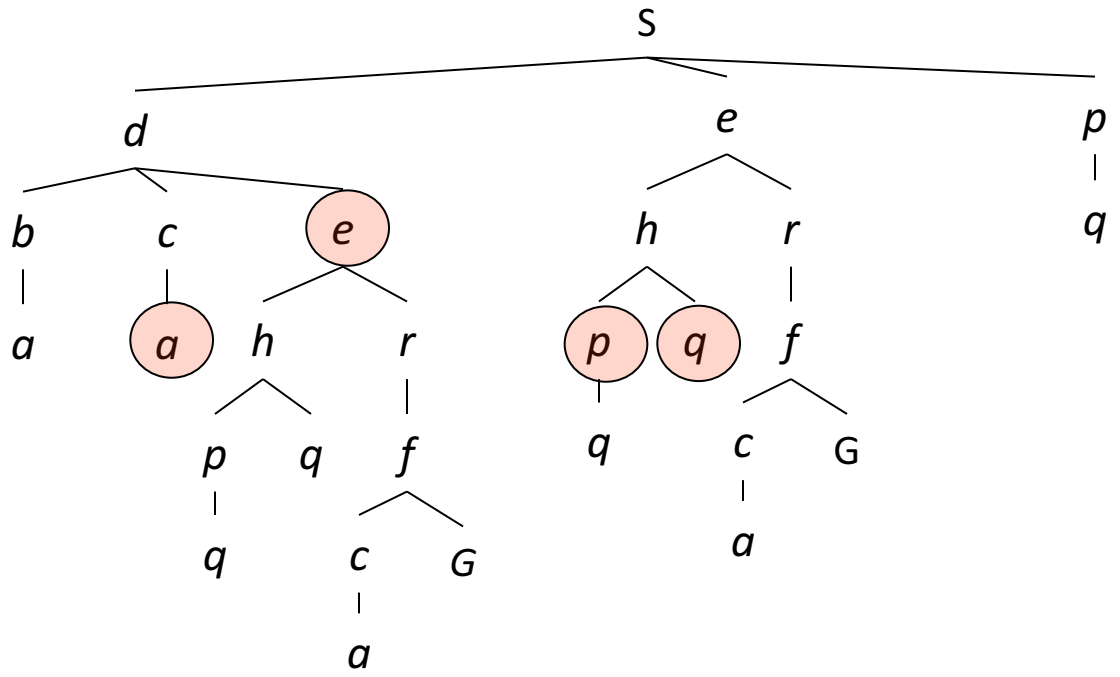
- Failure to detect repeated states can cause exponentially more work.





Graph Search

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

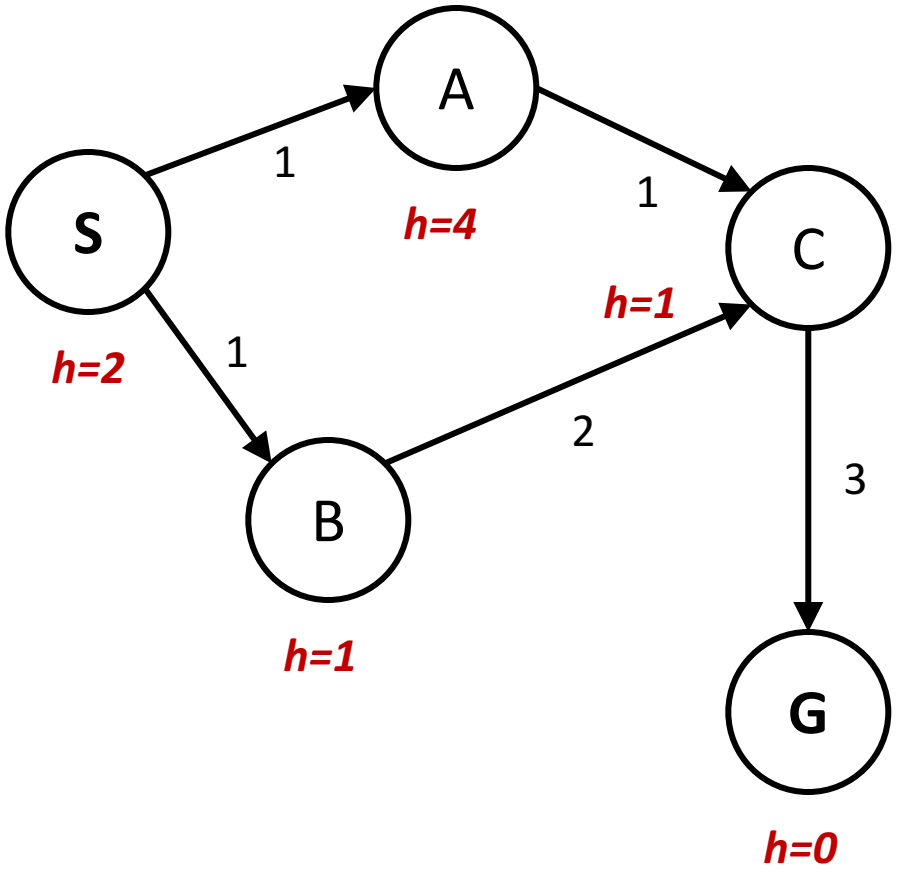


Graph Search

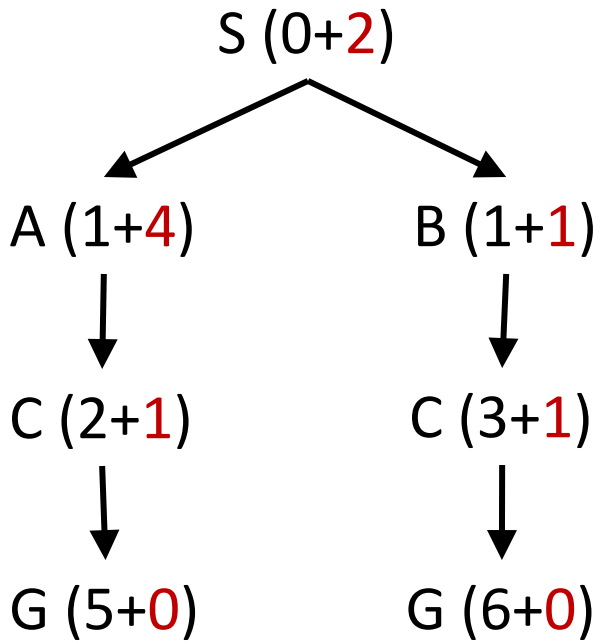
- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set
- Important: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

A* Graph Search Gone Wrong?

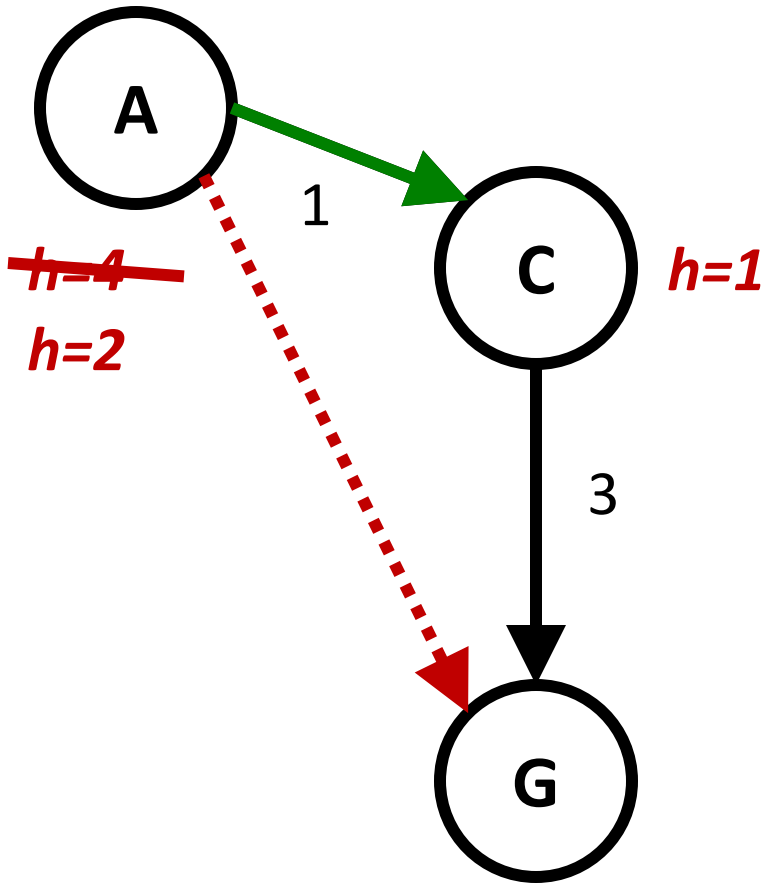
State space graph



Search tree



Consistency of Heuristics



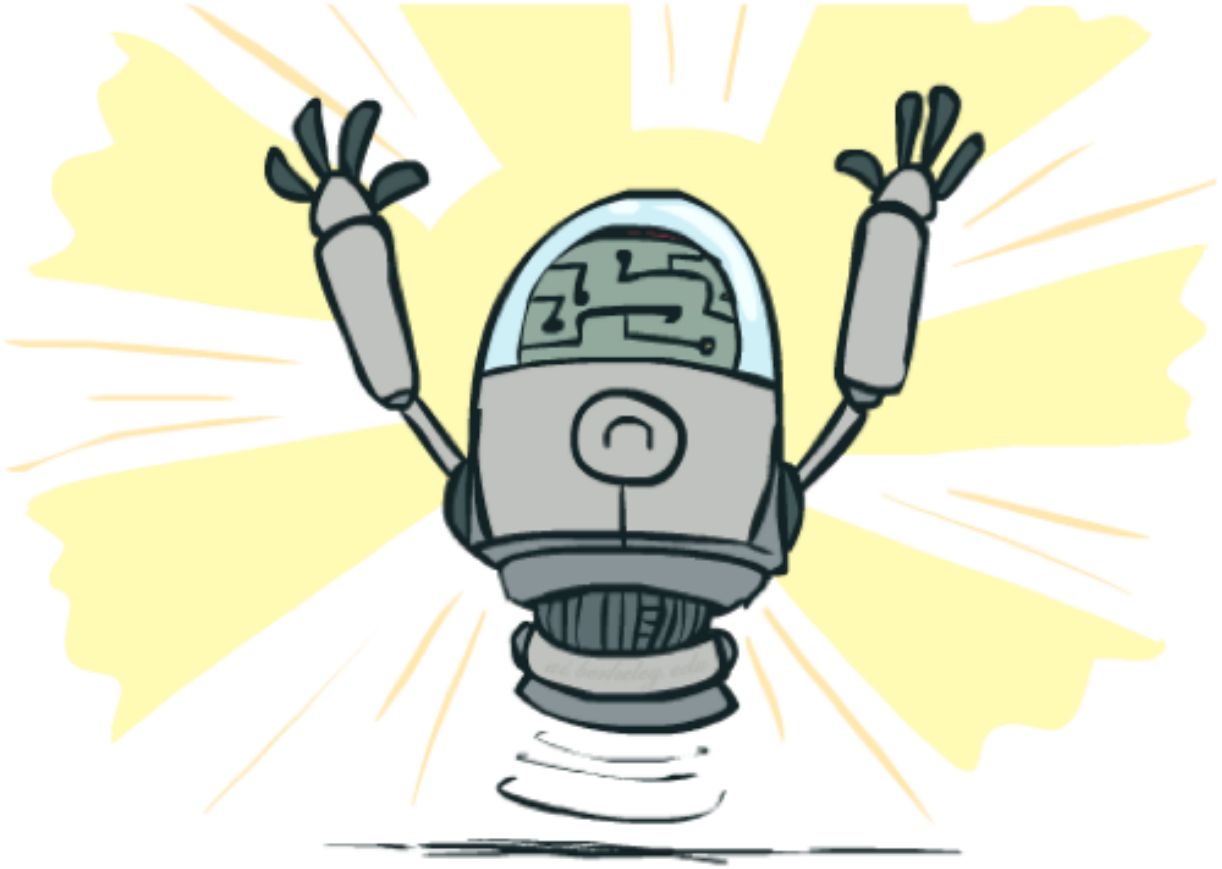
- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

    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

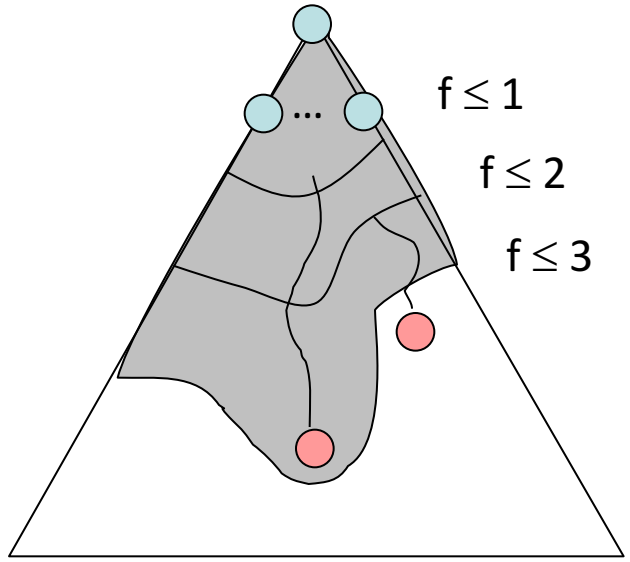
    h(A) ≤ cost(A to C) + h(C)
  - A* graph search is optimal

Optimality of A* Graph Search



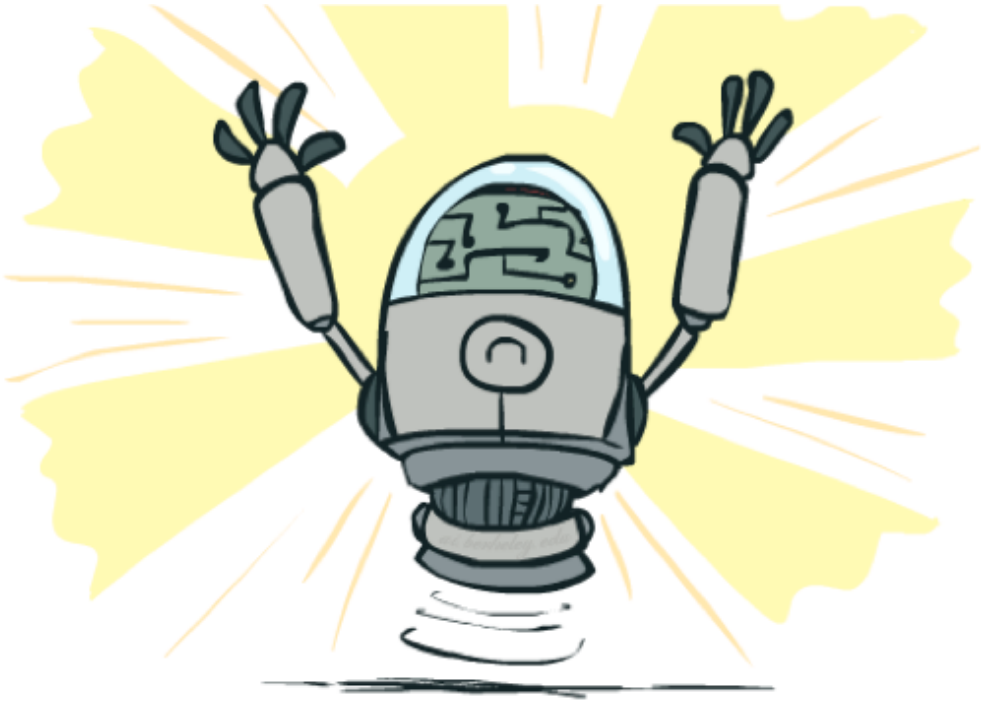
Optimality of A* Graph Search

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



Optimality

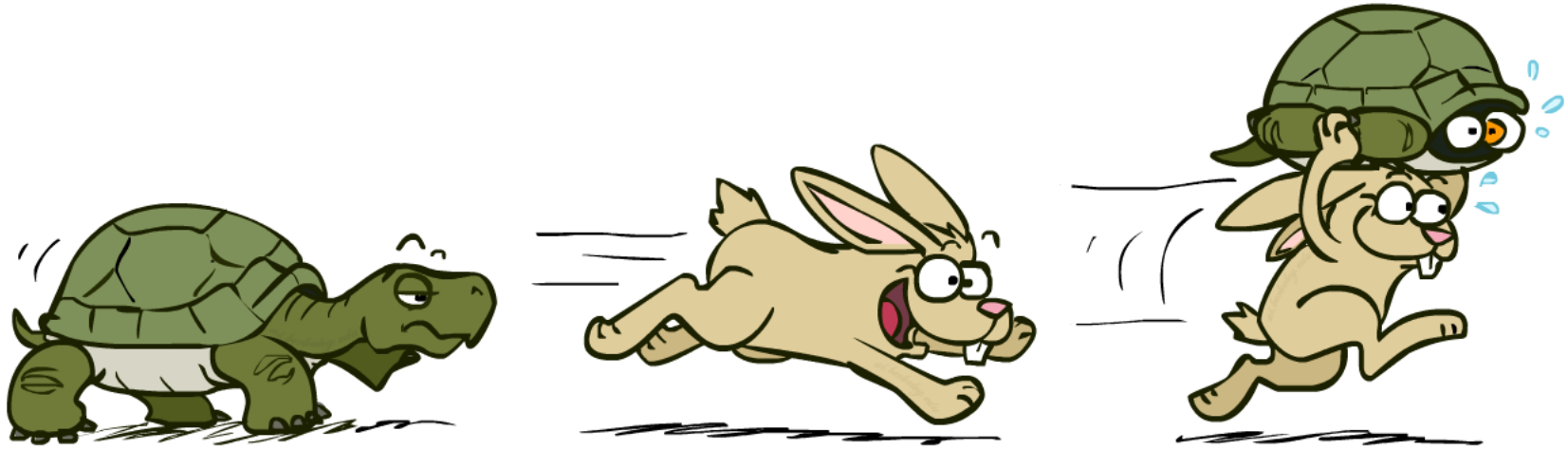
- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case (h = 0)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)
- Consistency implies admissibility
- In general, most natural admissible heuristics tend to be consistent, especially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



A*: Summary

A*: Summary

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



Tree Search Pseudo-Code

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

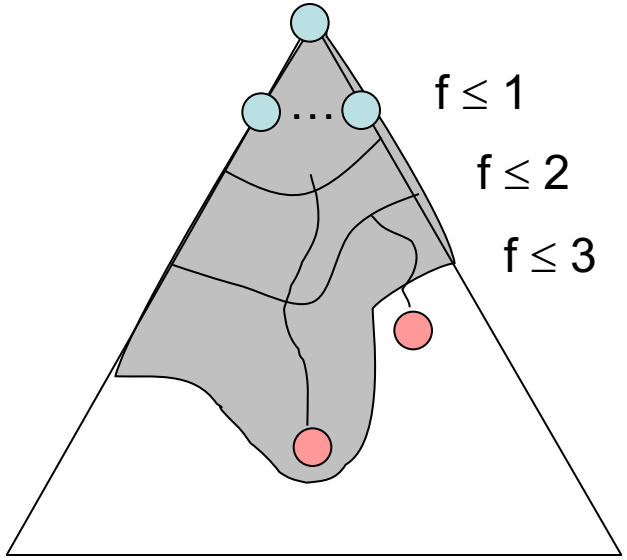
Graph Search Pseudo-Code

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

Optimality of A* Graph Search

- Consider what A* does:
  - Expands nodes in increasing total f value (f-contours)
    Reminder: f(n) = g(n) + h(n) = cost to n + heuristic
  - Proof idea: the optimal goal(s) have the lowest f value, so it must get expanded first

There's a problem with this argument. What are we assuming is true?



Optimality of A* Graph Search

Proof:

- New possible problem: some *n* on path to G* isn't in queue when we need it, because some worse *n'* for the same state dequeued and expanded first (disaster!)
- Take the highest such *n* in tree
- Let *p* be the ancestor of *n* that was on the queue when *n*' was popped
- *f(p) < f(n)* because of consistency
- *f(n) < f(n')* because *n'* is suboptimal
- *p* would have been expanded before *n*'
- Contradiction!