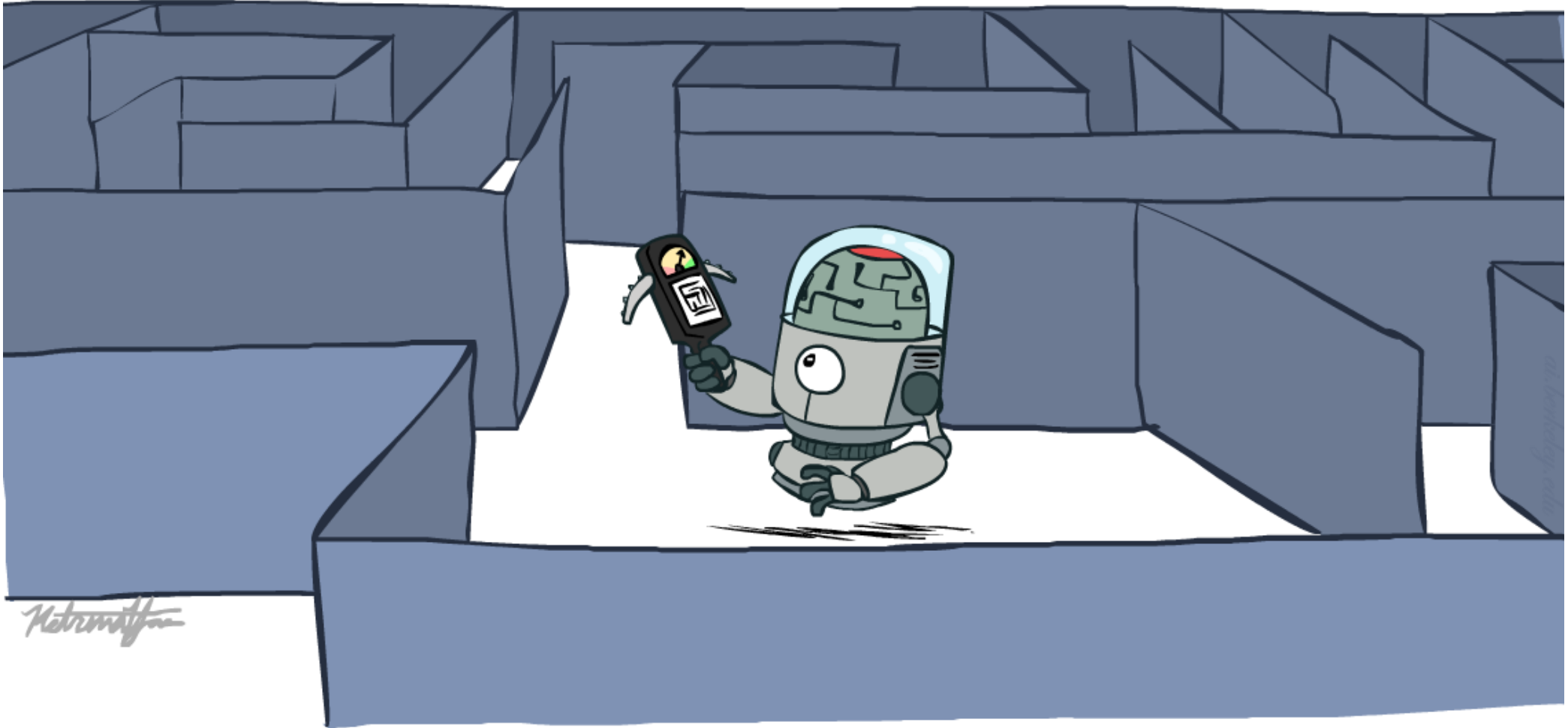


Informed Search



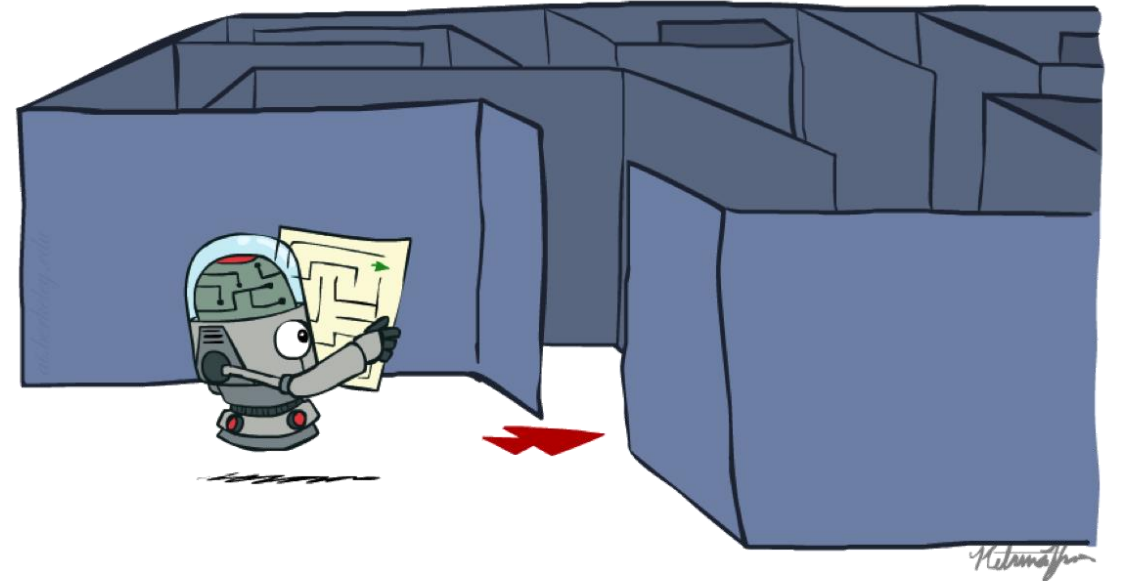
Today

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

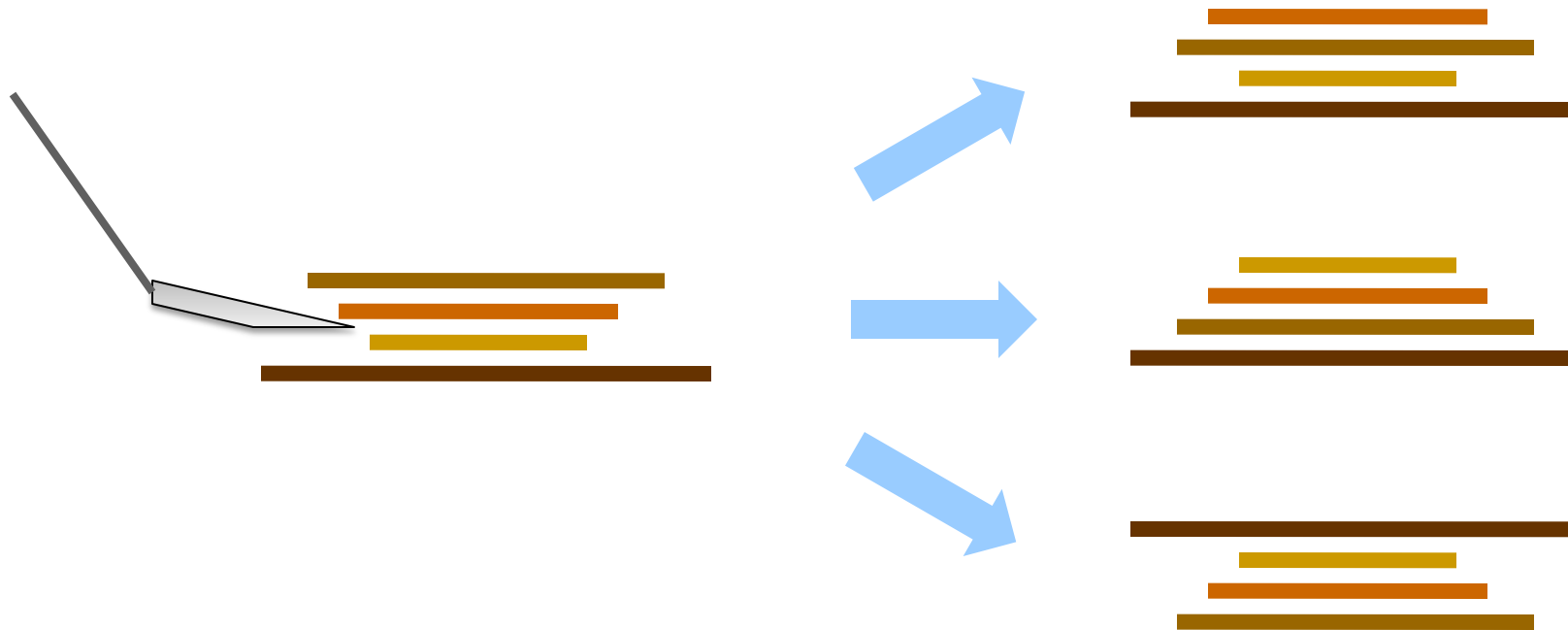


Recap: Search

- **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



Example: Pancake Problem



Cost: Number of pancakes flipped

Example: Pancake Problem

BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES

Microsoft, Albuquerque, New Mexico

Christos H. PAPANIMITRIOU*†

Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

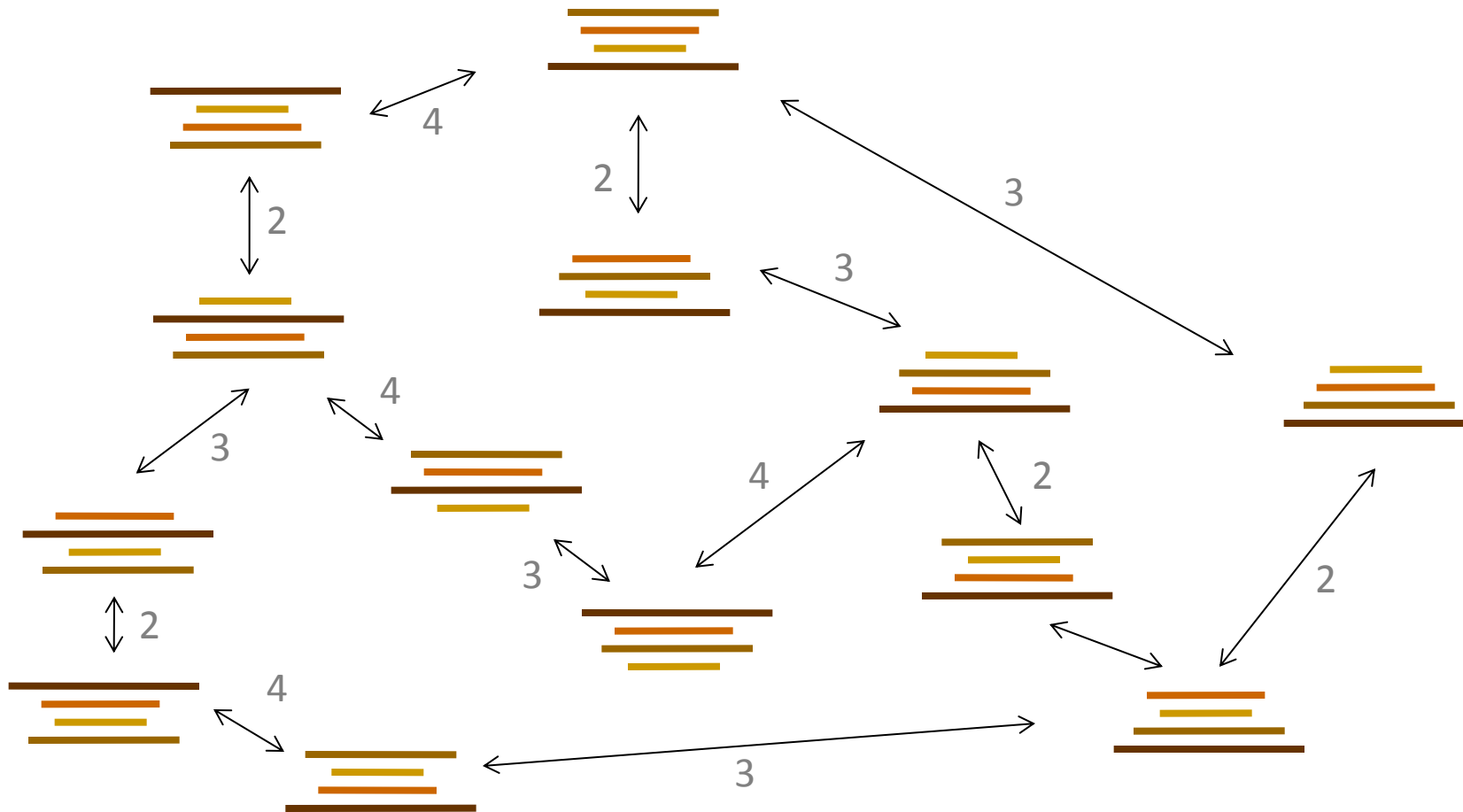
Received 18 January 1978

Revised 28 August 1978

For a permutation σ of the integers from 1 to n , let $f(\sigma)$ be the smallest number of prefix reversals that will transform σ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all σ in (the symmetric group) S_n . We show that $f(n) \leq (5n+5)/3$, and that $f(n) \geq 17n/16$ for n a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2 - 1 \leq g(n) \leq 2n + 3$.

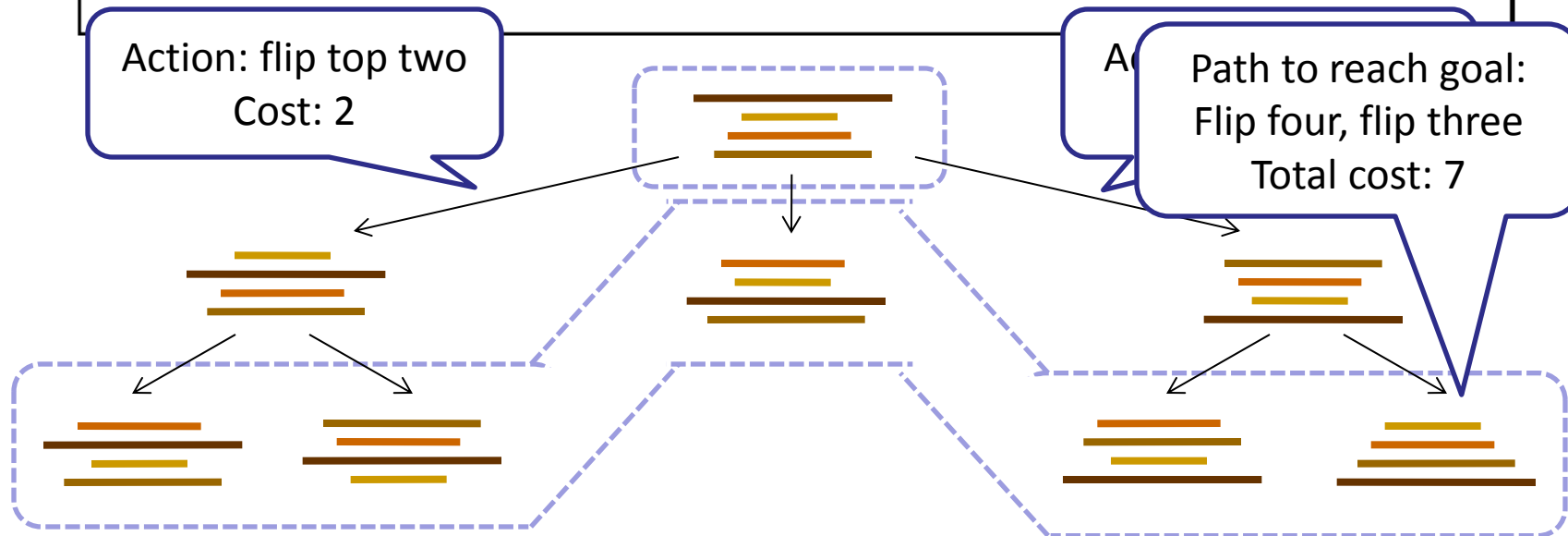
Example: Pancake Problem

State space graph with costs as weights



General Tree Search

```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
  end
```

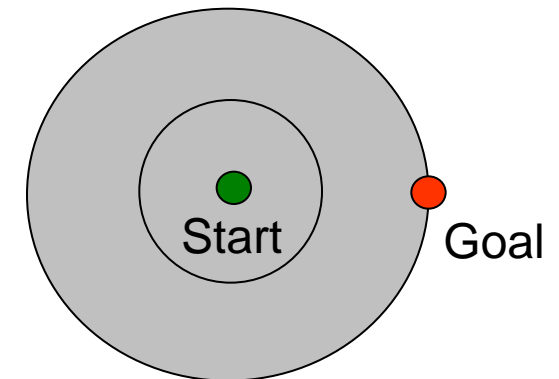
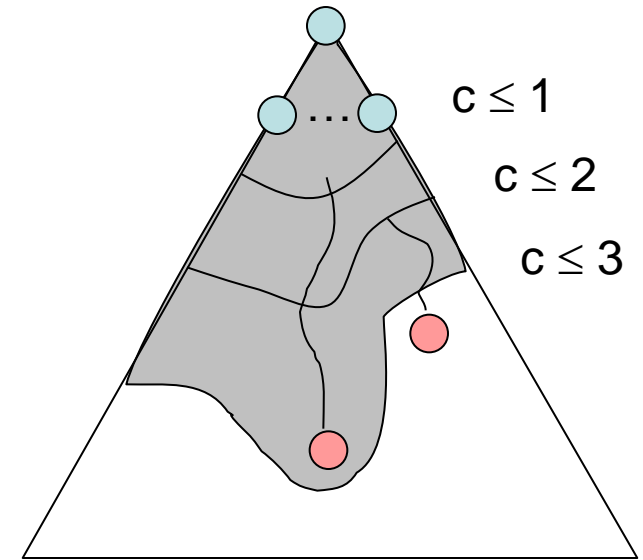


Recap: Uniform Cost Search



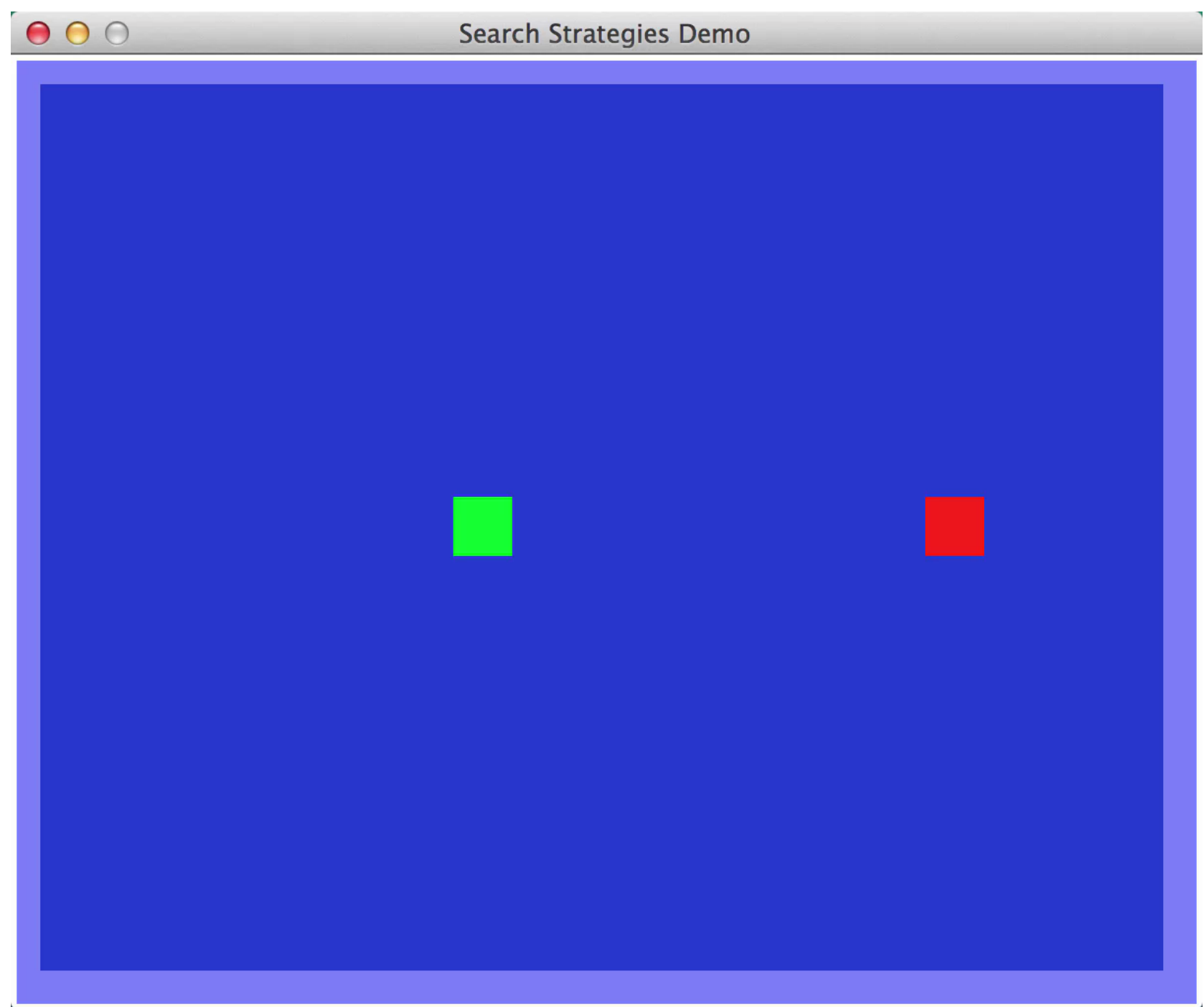
Uniform Cost Search

- Strategy: expand lowest path cost
- The good: UCS is complete and optimal!
- The bad:
 - Explores options in every “direction”
 - No information about goal location



Uniform Cost
Search (UCS):
Pathing in an
empty world

Notice: UCS
explores in all
directions

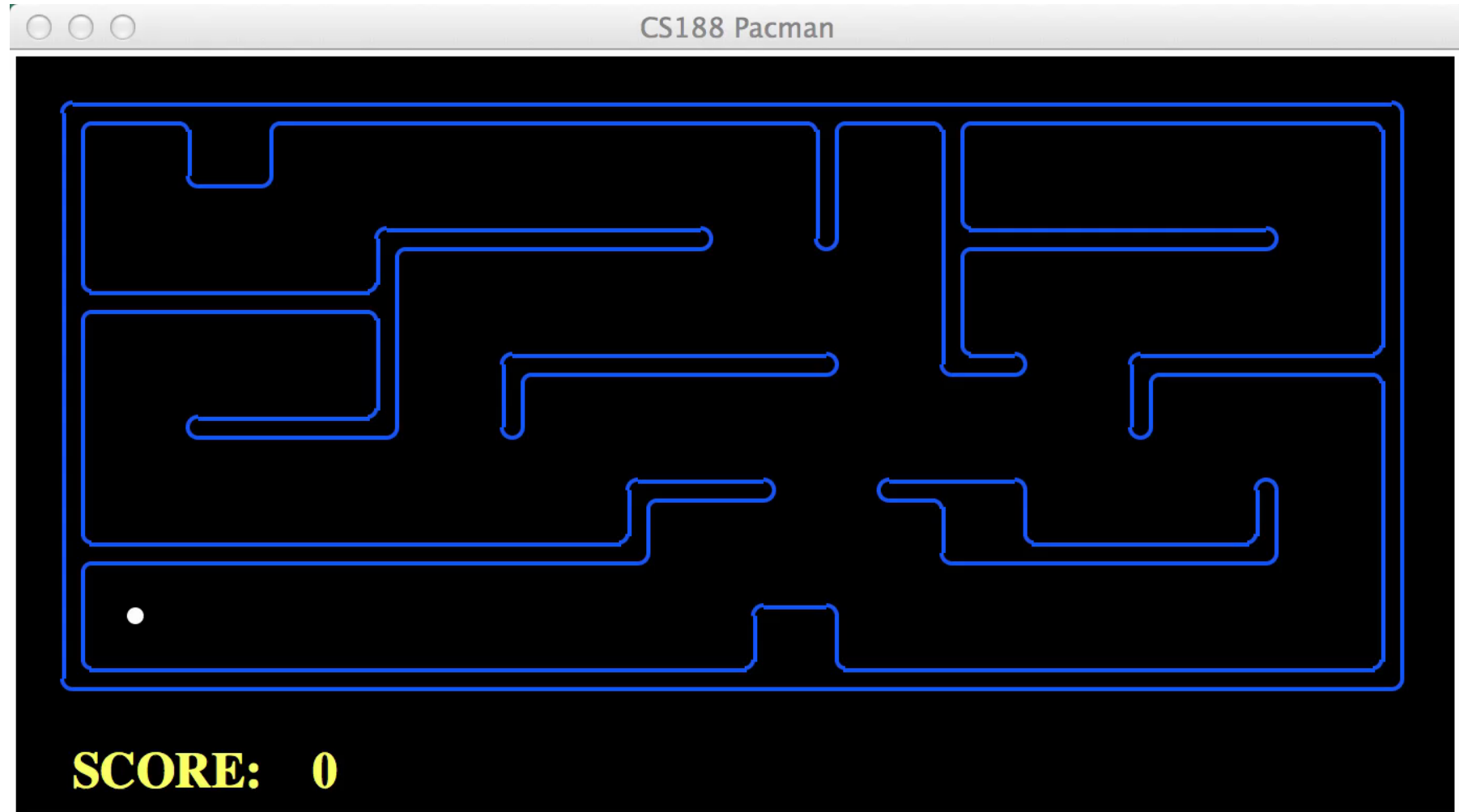


Uniform Cost Search (UCS): Pathing in Pac-Man world

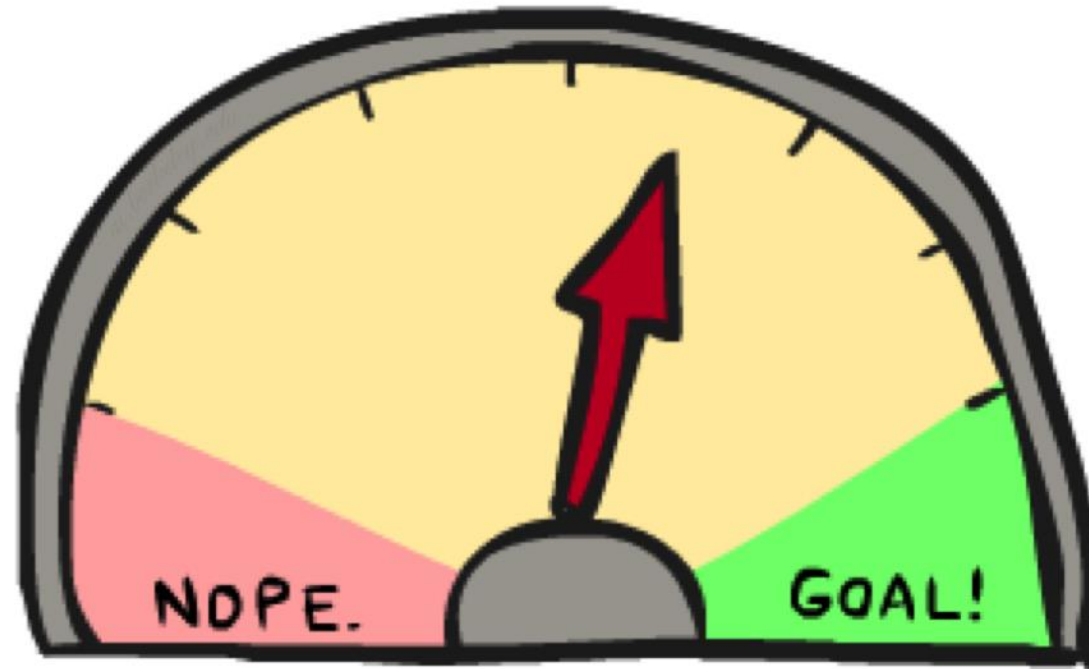
Color indicates when state was expanded during search.

Red = first

black = never

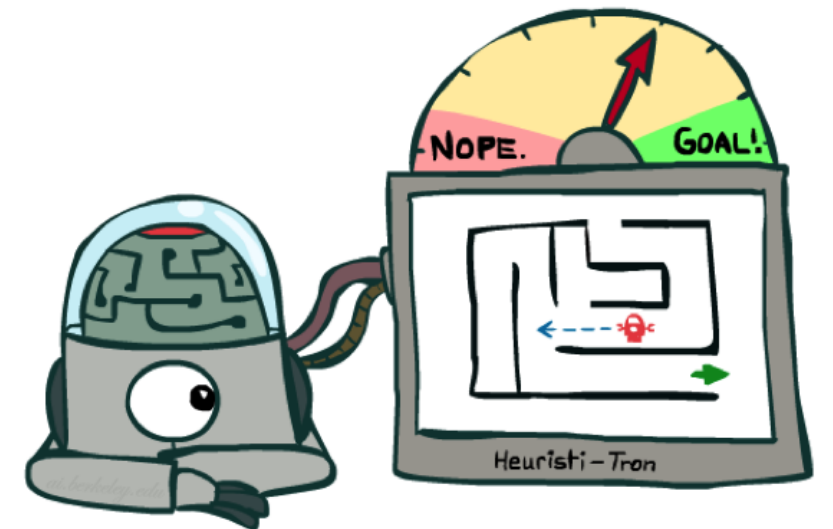
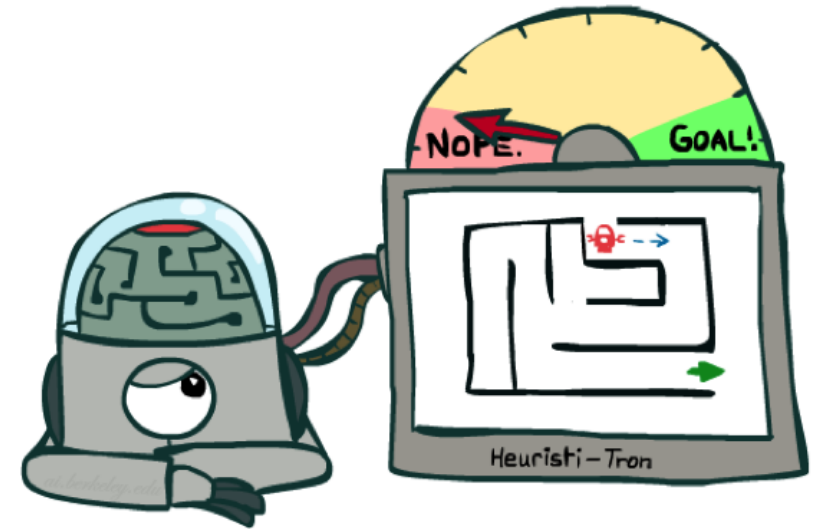
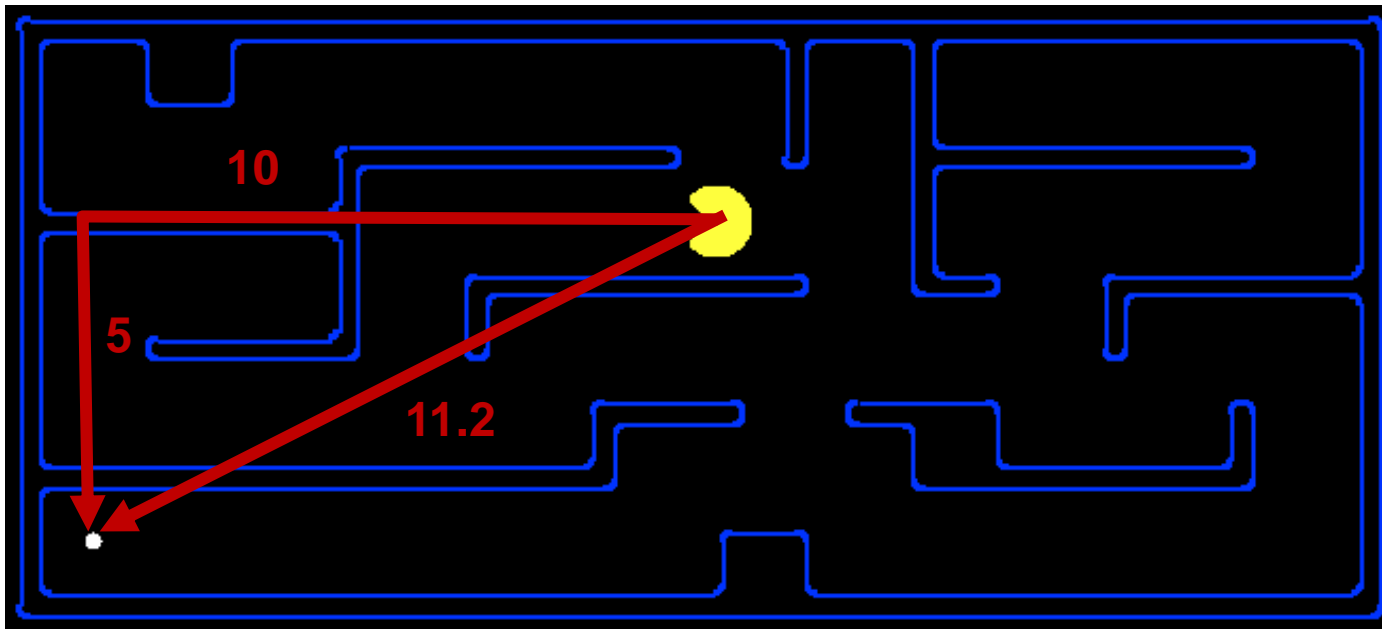


Informed Search

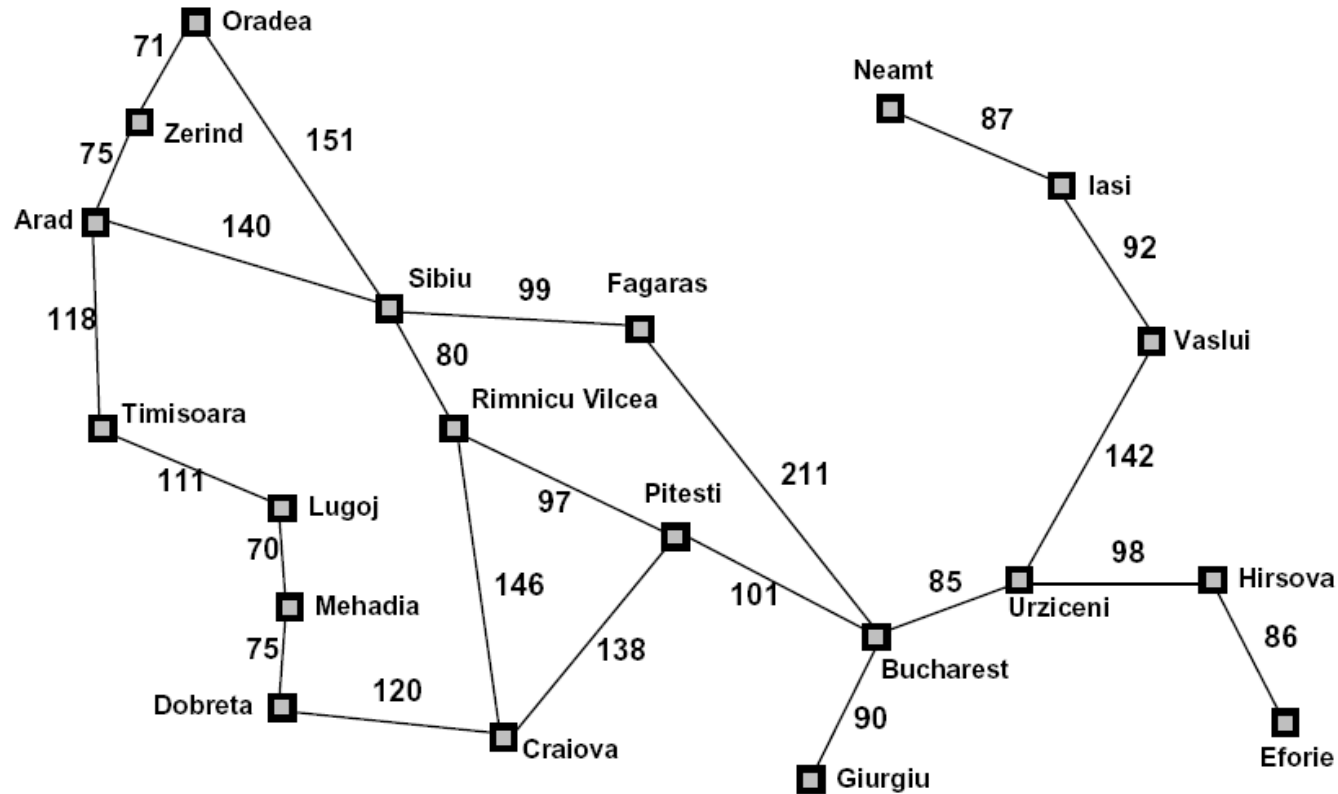


Search Heuristics

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



Example: Heuristic Function

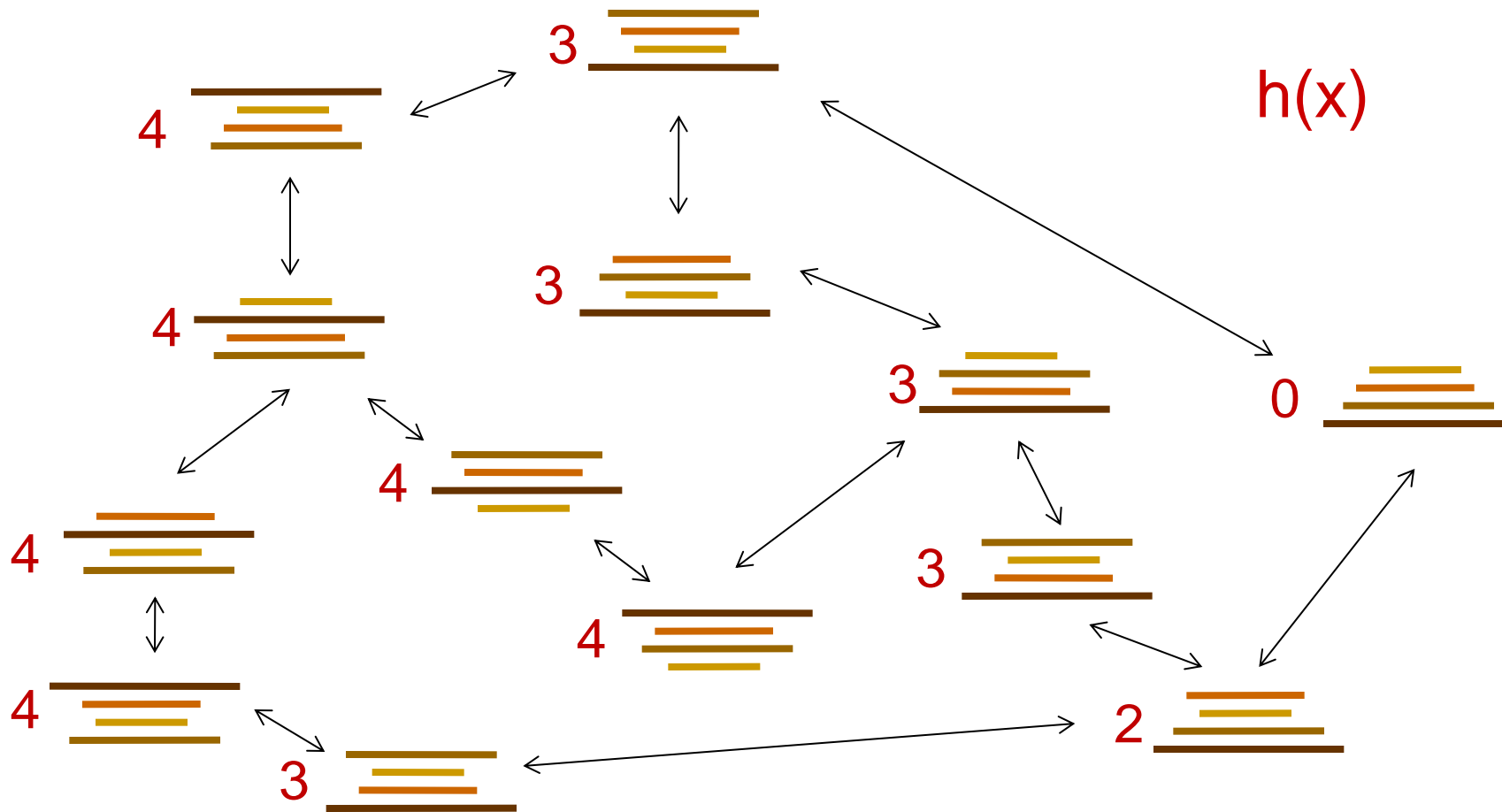


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

$h(x)$

Example: Heuristic Function

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

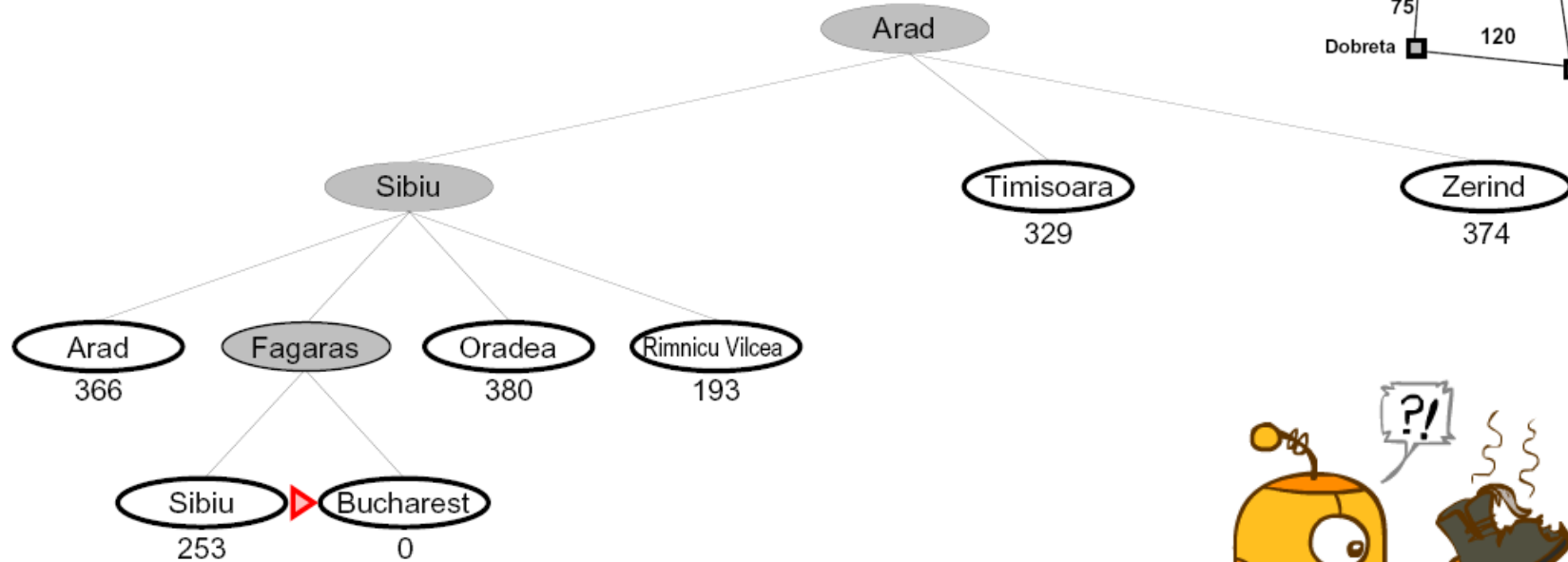


Greedy Search

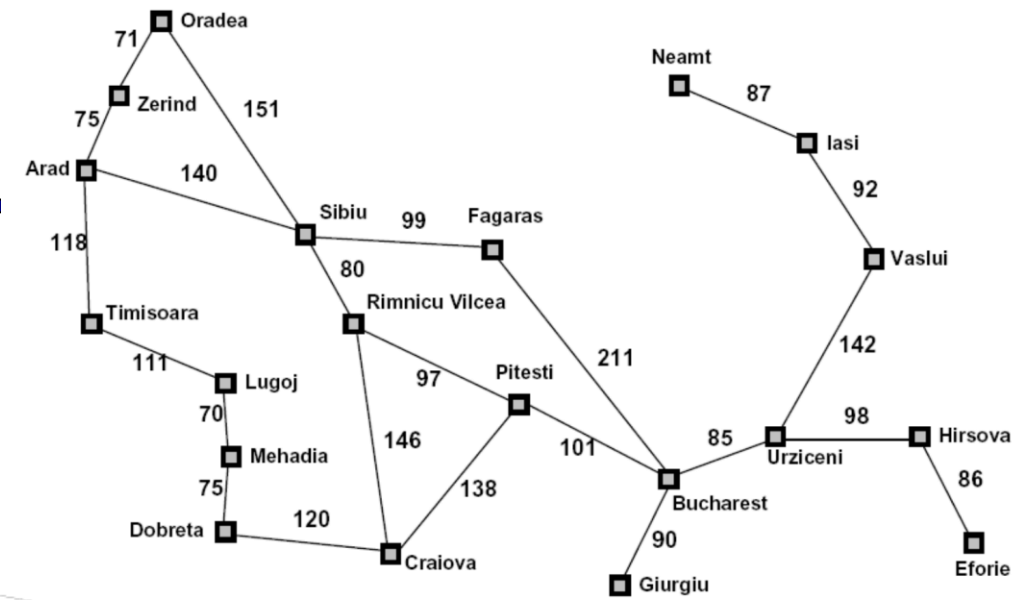


Greedy Search

- Expand the node that seems closest...

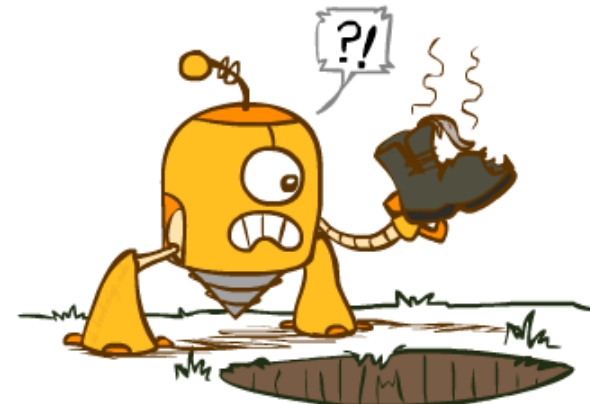


- What can go wrong?
 - You can get a path that is not optimal



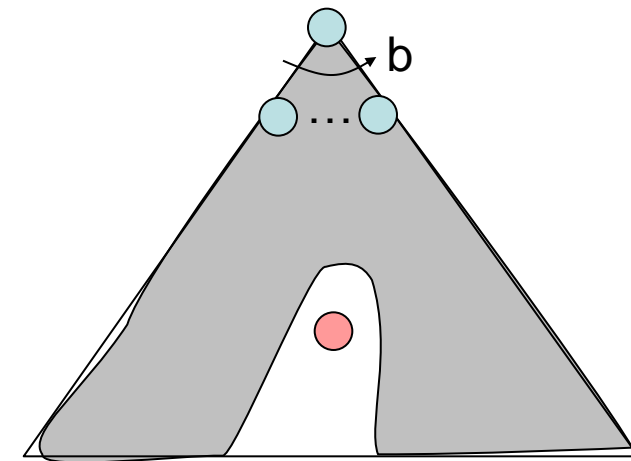
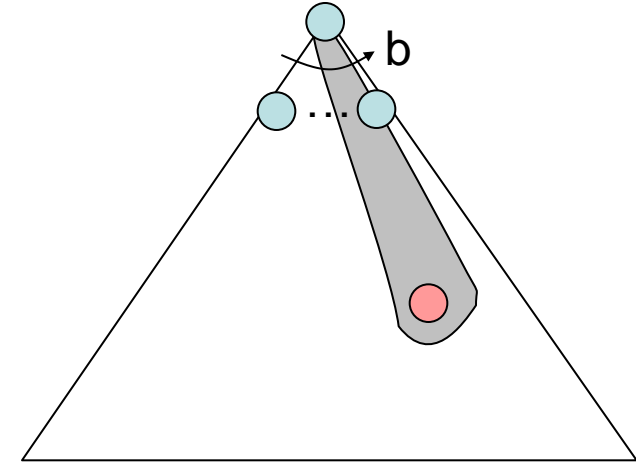
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

$h(x)$



Greedy Search

- 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



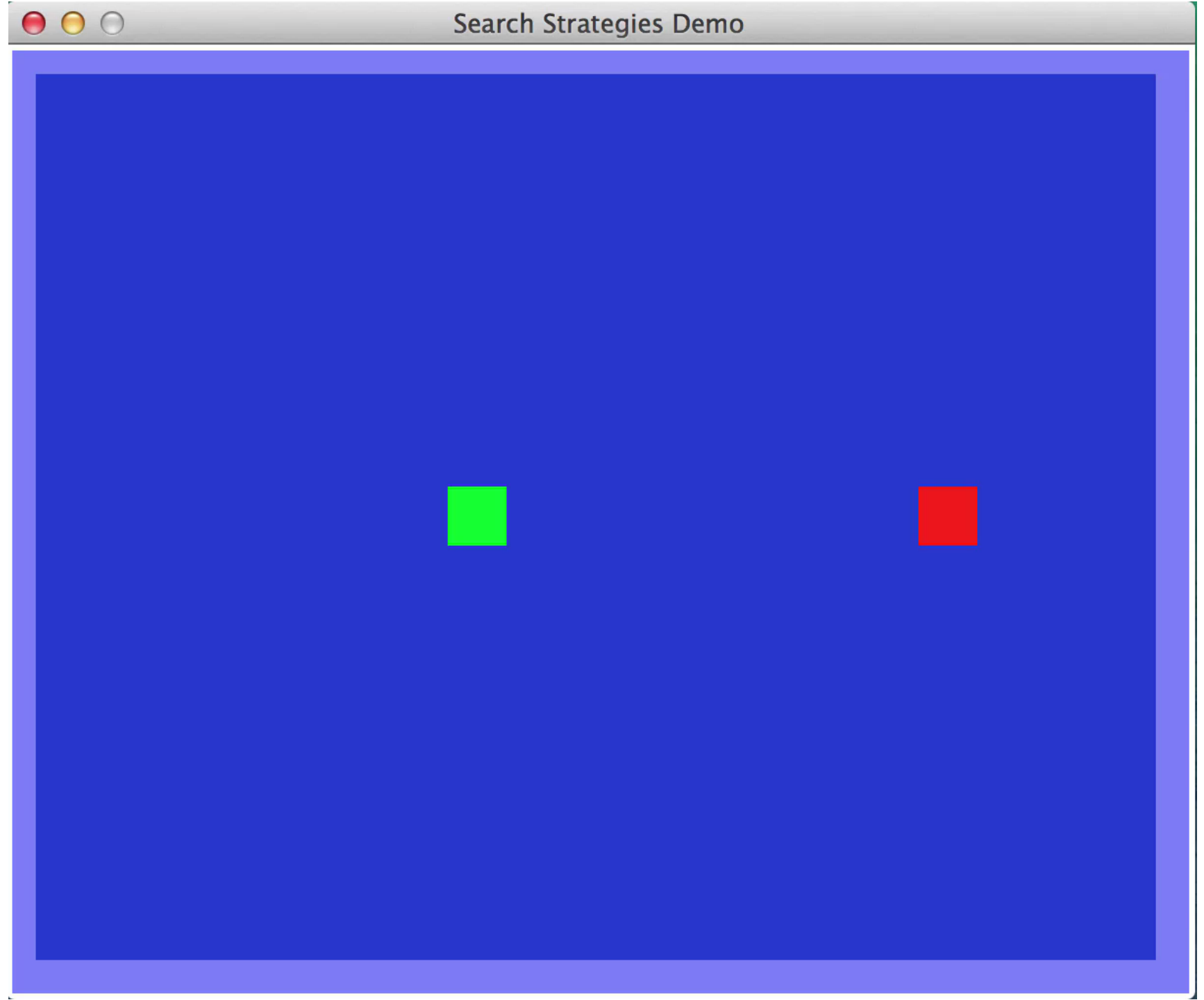
What search strategy is this?

Breadth-First Search (BFS)

-or-

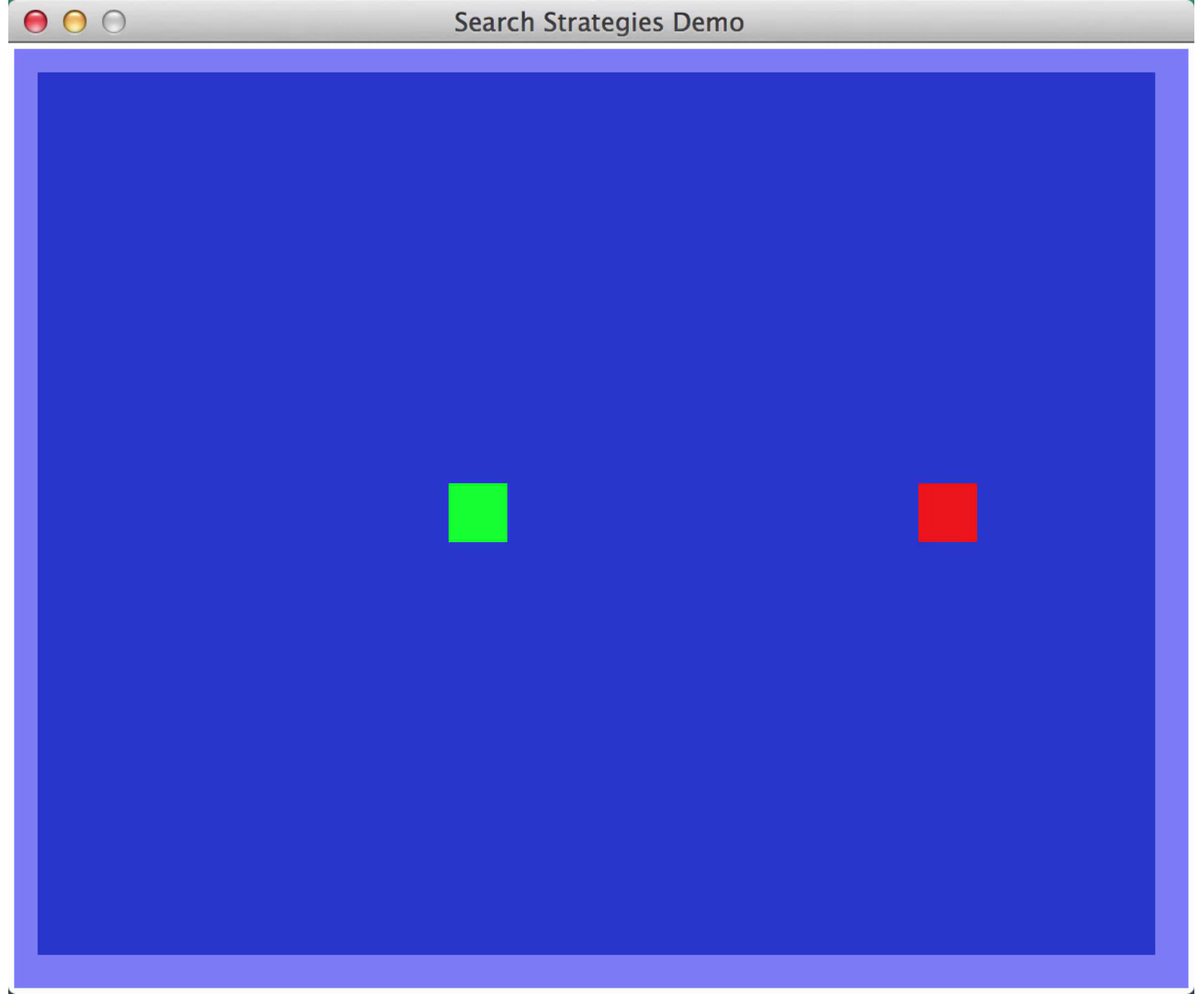
Uniform Cost Search (UCS)

Note: since all costs 1, behaves the same as BFS



What search strategy is this?

Depth-First Search (DFS)



What search strategy is this?

Greedy search



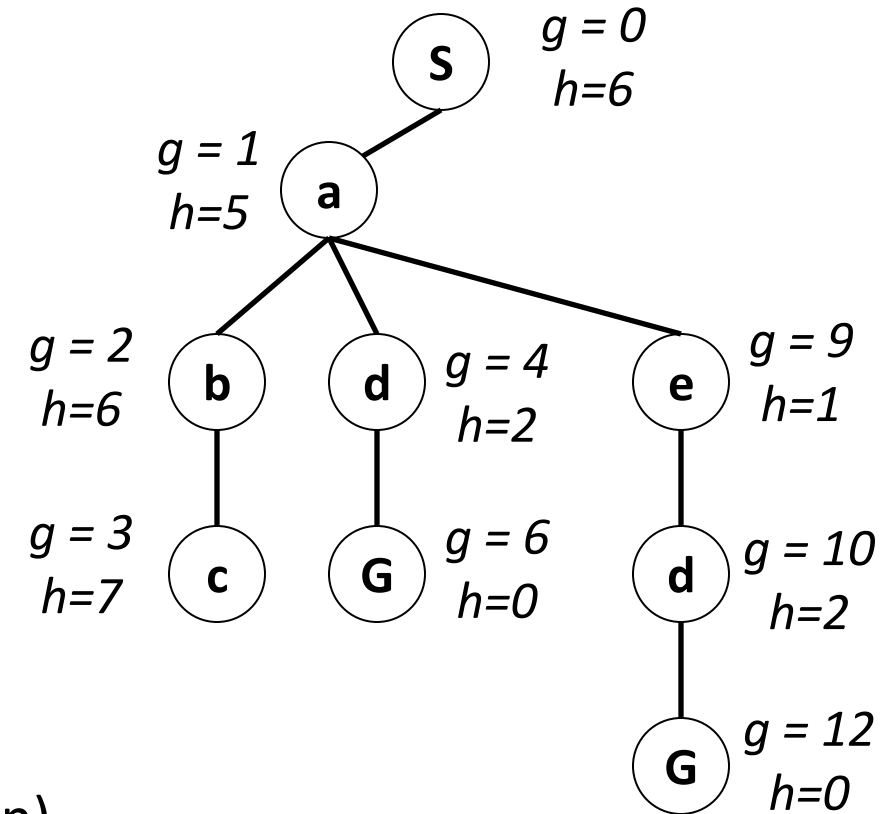
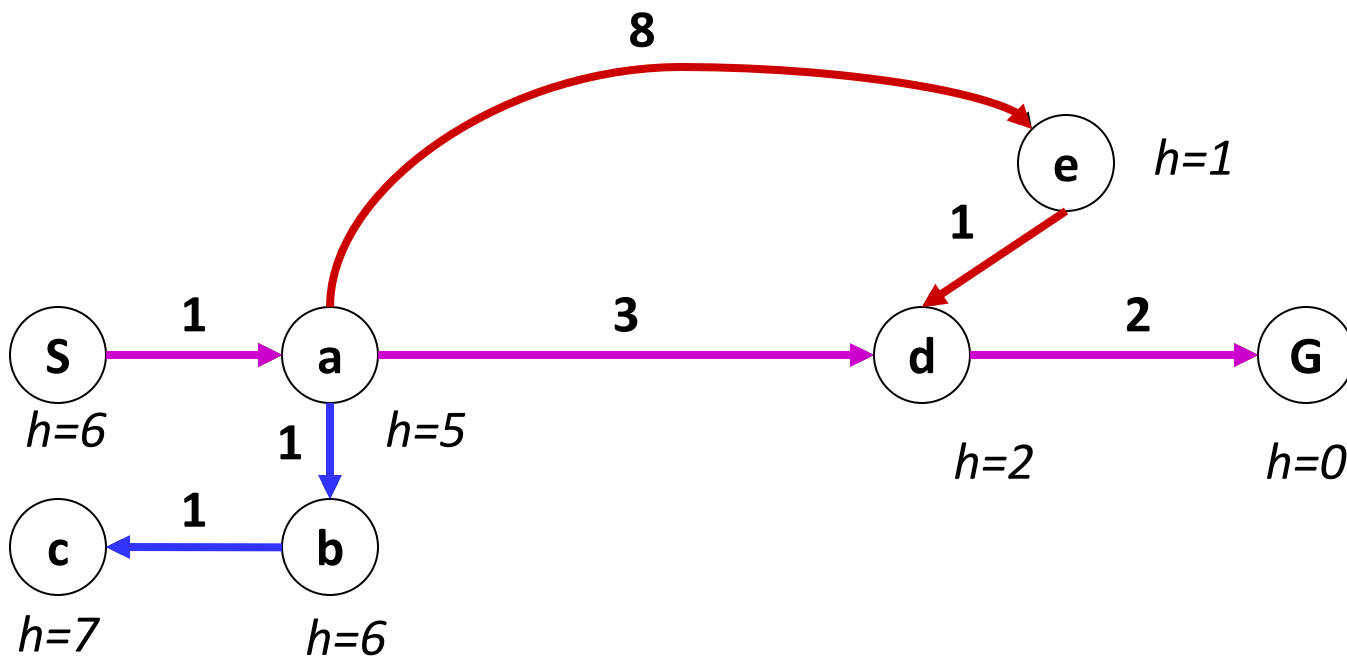
A* Search



A* Search

Combining UCS and Greedy

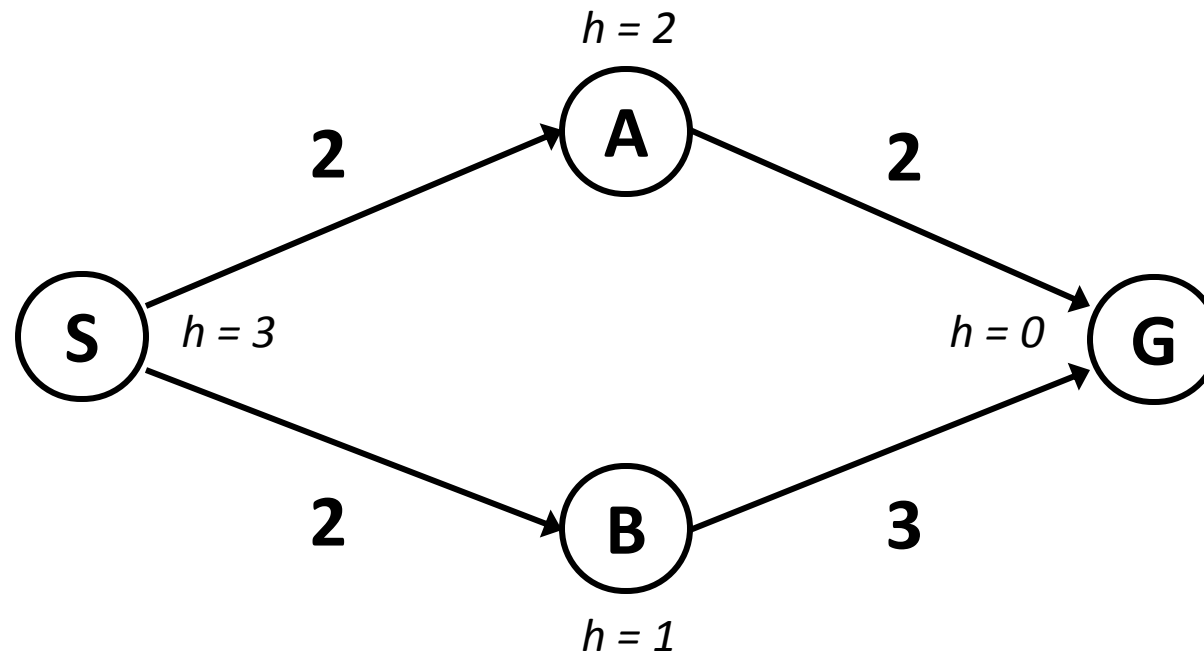
- Uniform-cost orders by path cost, or *backward cost* $g(n)$
- Greedy orders by goal proximity, or *forward cost* $h(n)$



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

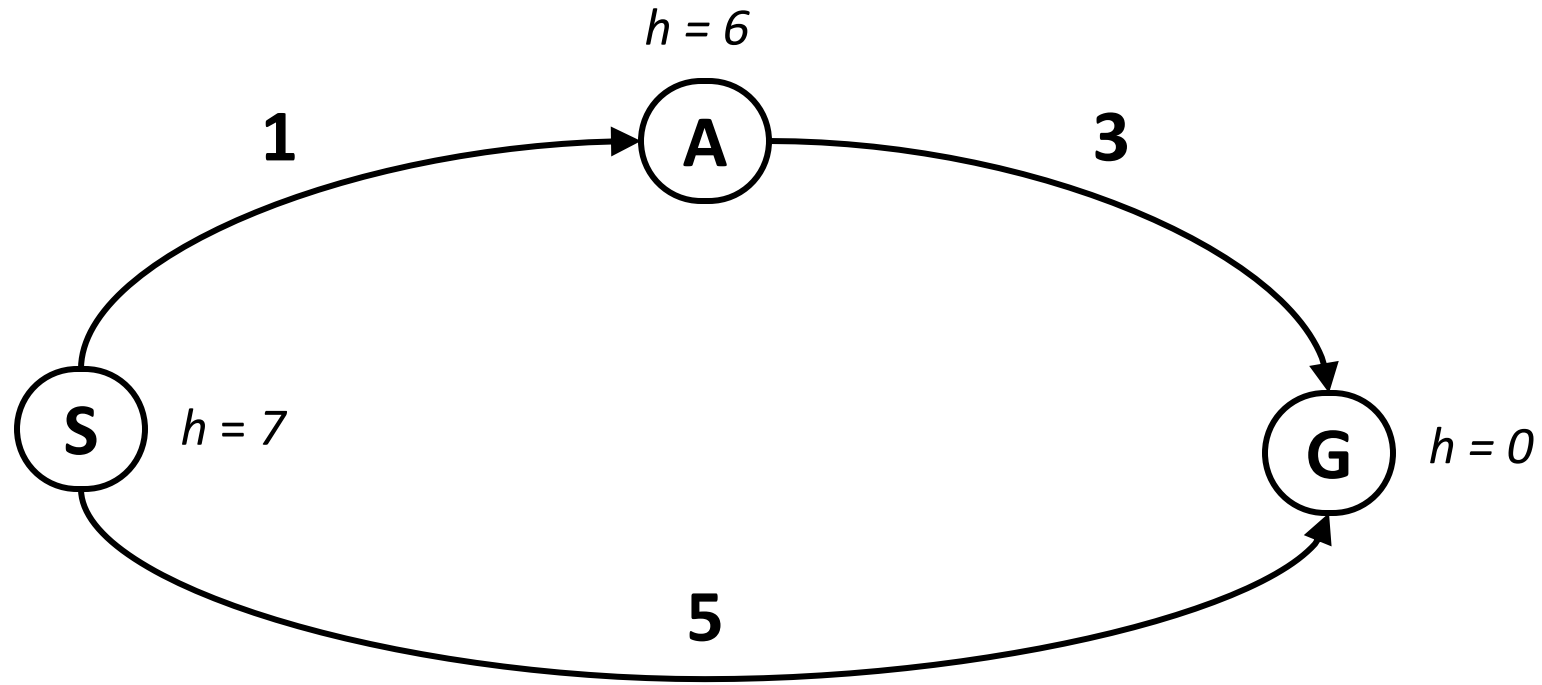
When should A* terminate?

- Should we stop when we enqueue a goal?



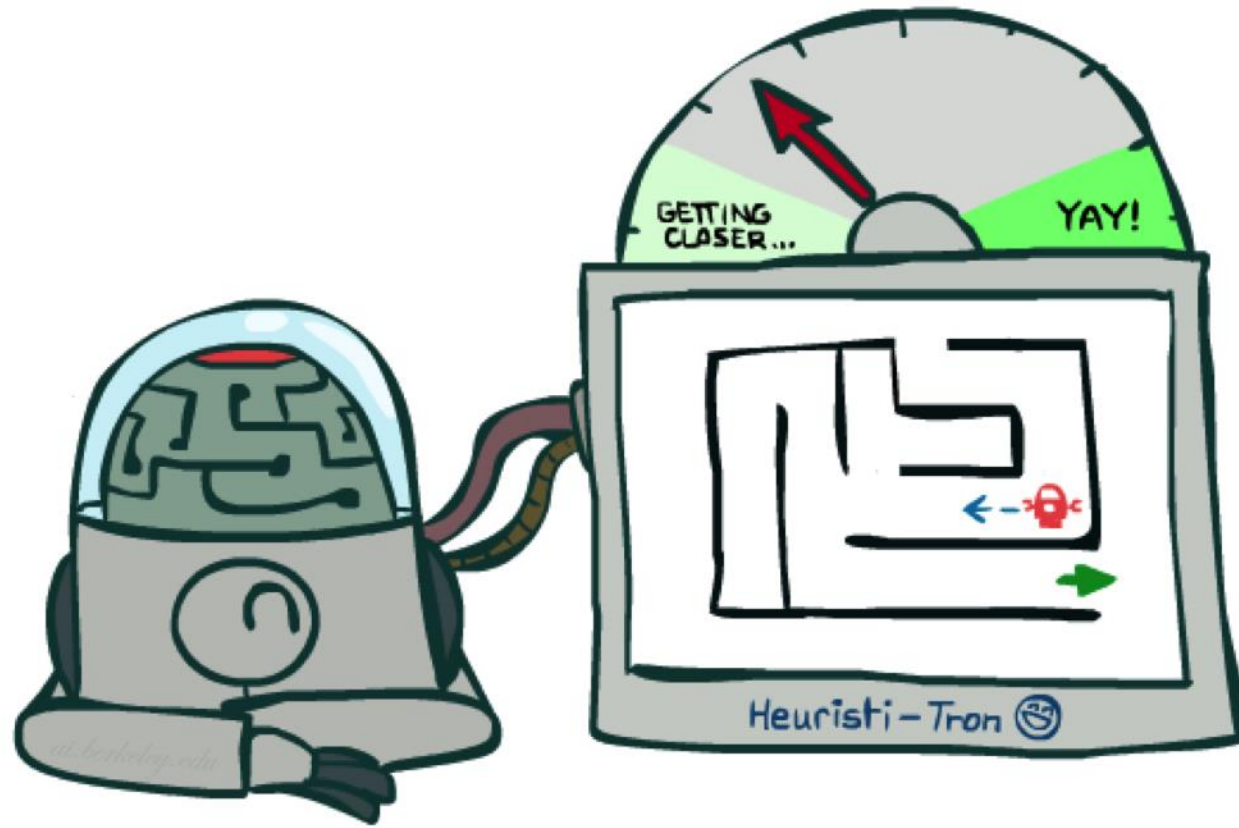
- No: only stop when we dequeue a goal

Is A* Optimal?

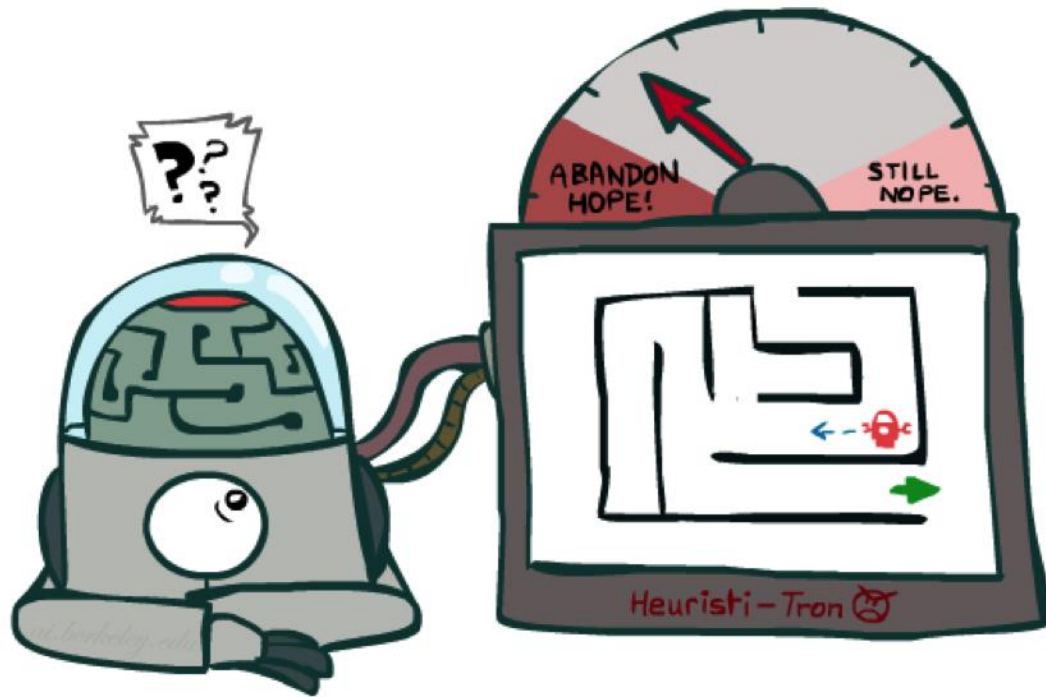


- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

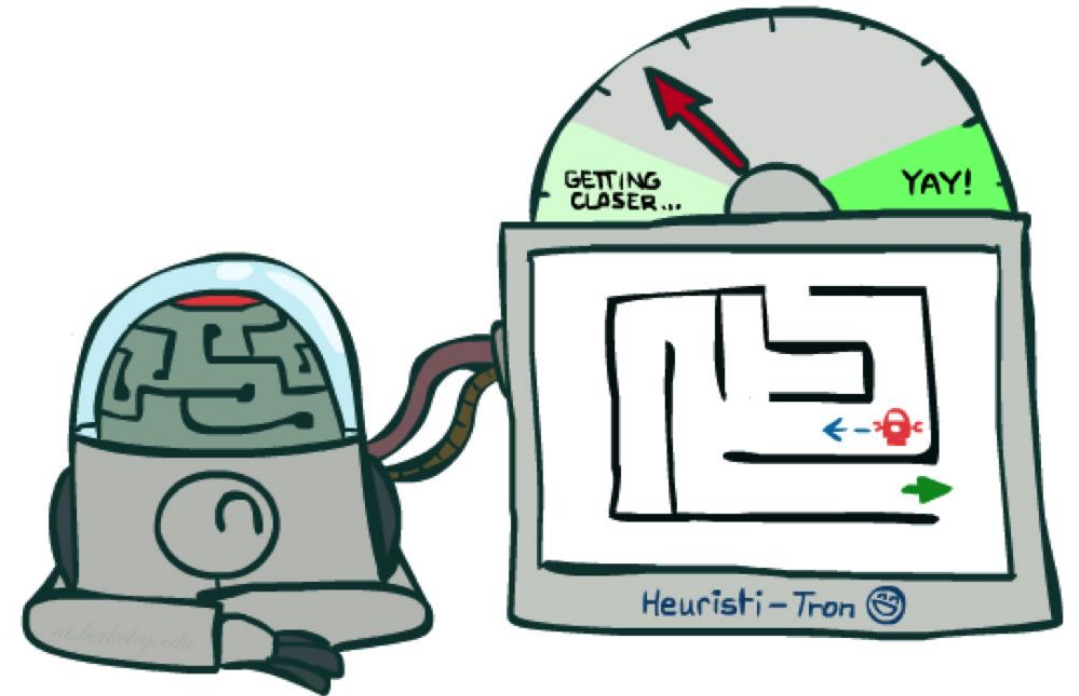
Admissible Heuristics



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

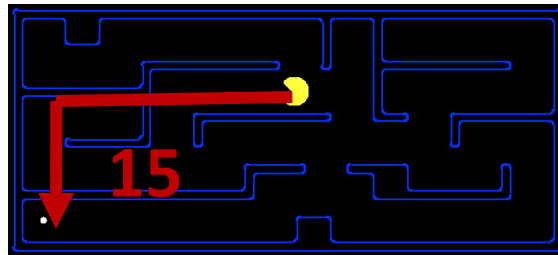
Admissible Heuristics

- 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:



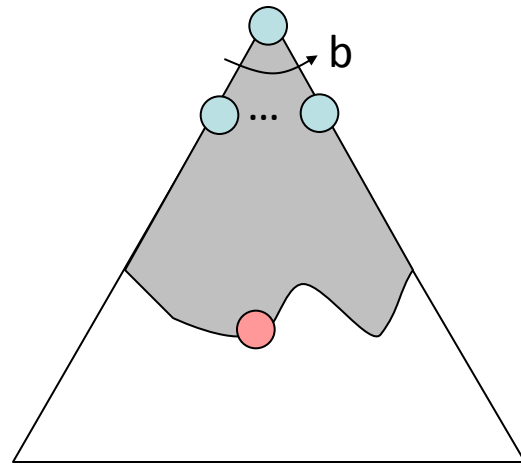
4



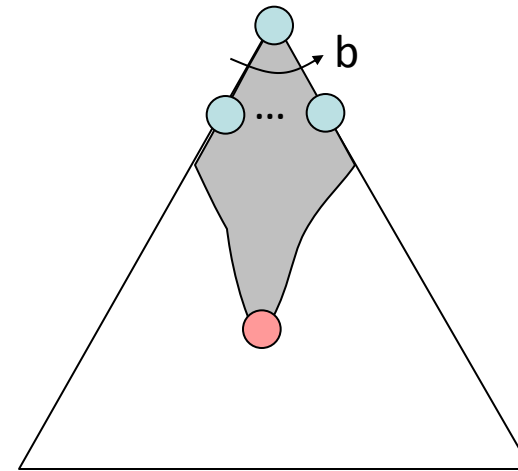
- Coming up with admissible heuristics is most of what's involved in using A* in practice.

Properties of A^*

Uniform-Cost

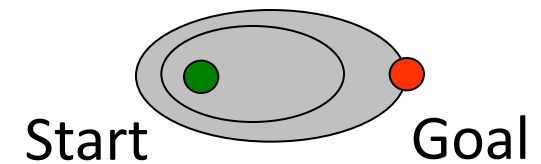
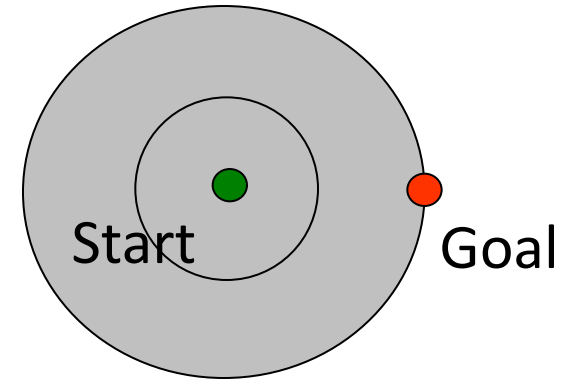


A^*



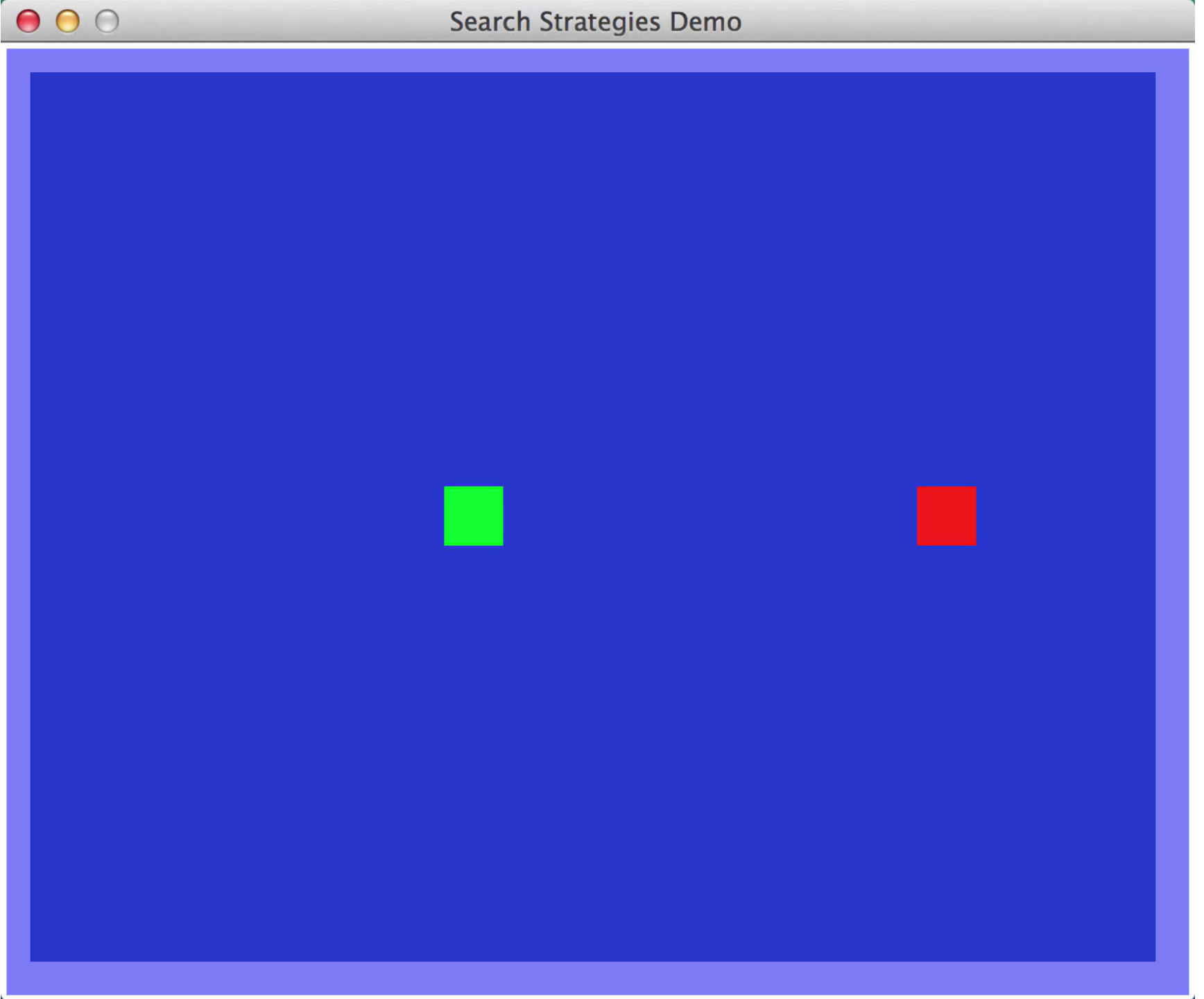
UCS vs A* Contours

- Uniform-cost expands equally in all “directions”
- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



What search strategy is this?

A* search



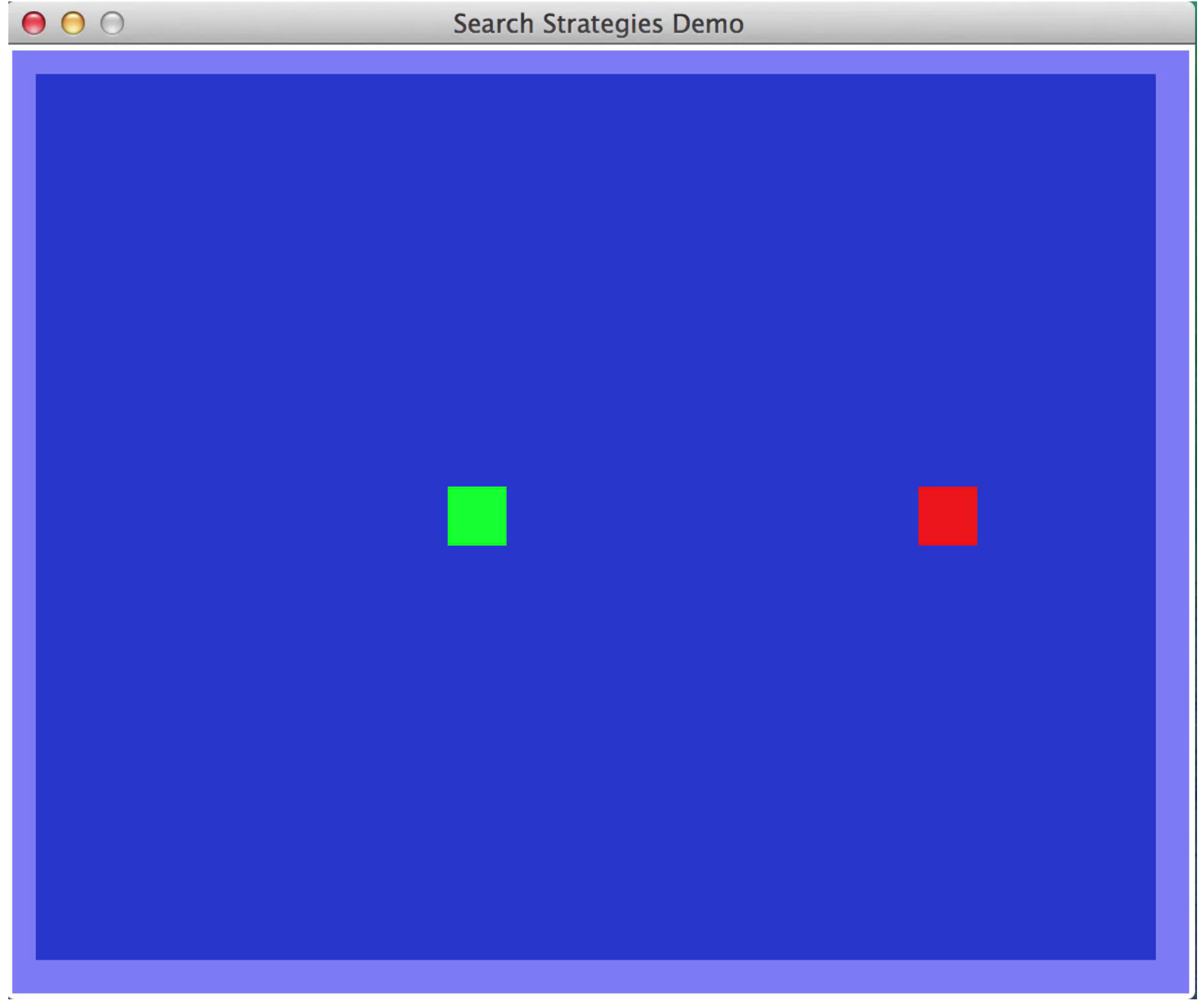
What search strategy is this?

Breadth-First Search (BFS)

-or-

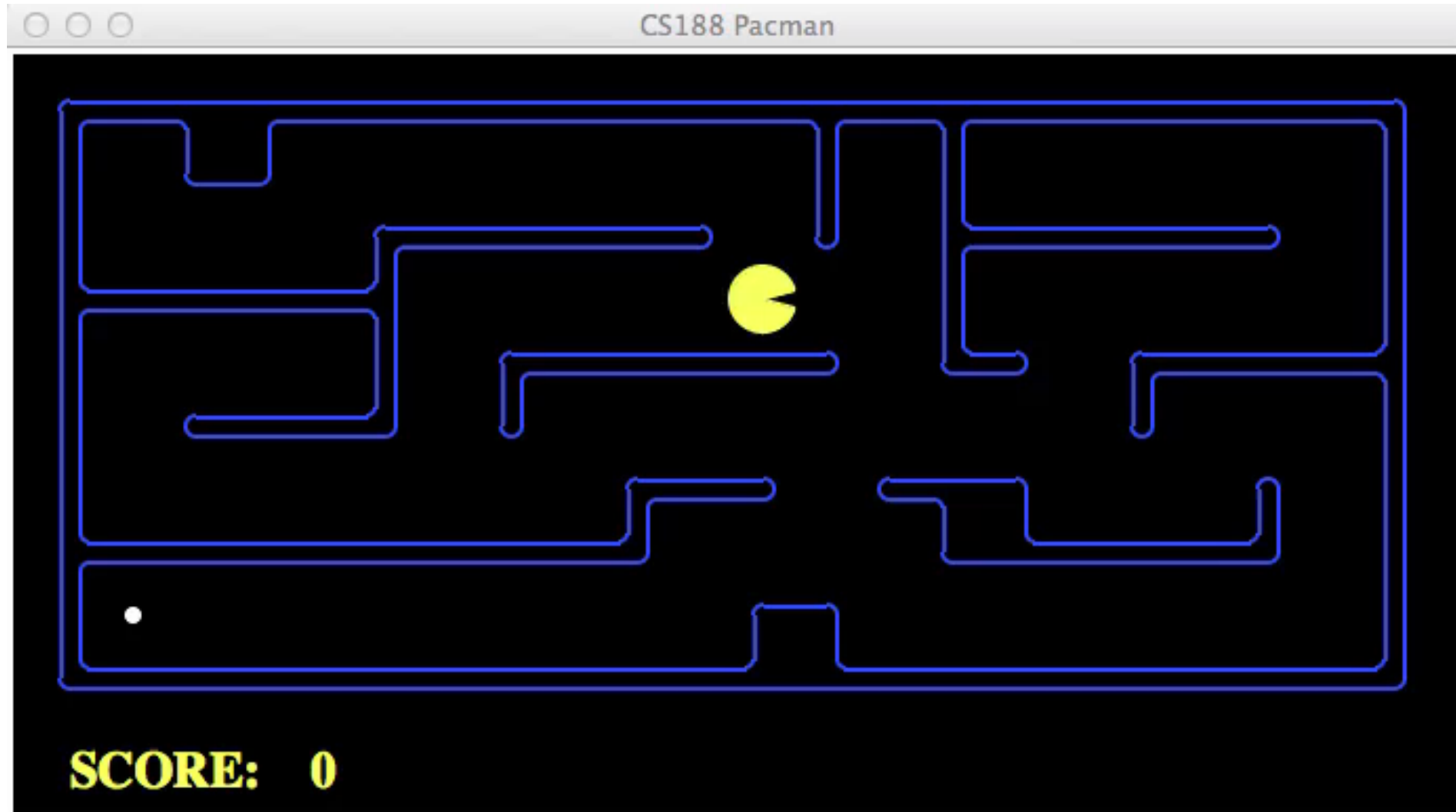
Uniform Cost Search (UCS)

Note: since all costs 1, behaves the same as BFS



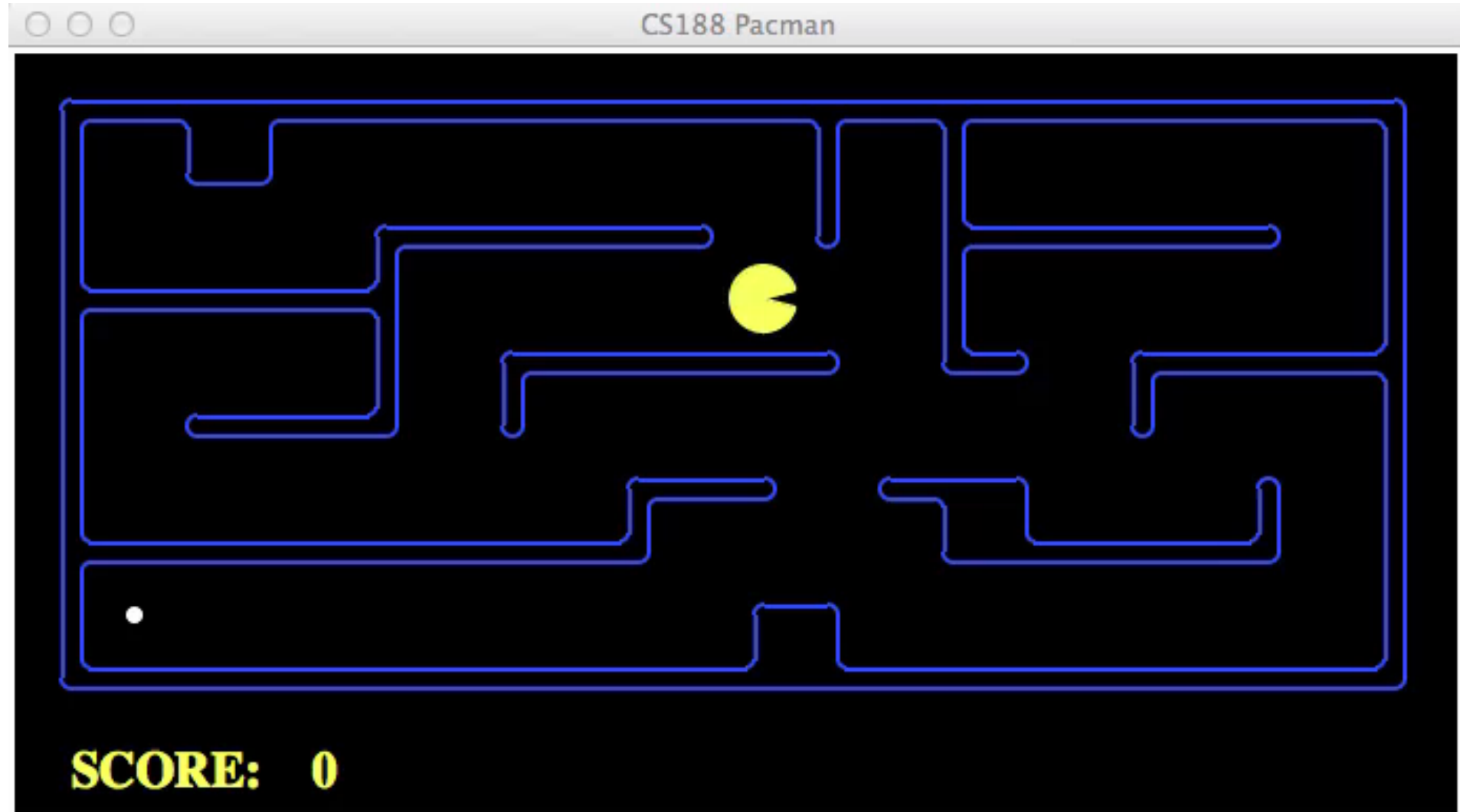
What search strategy is this?

Greedy search



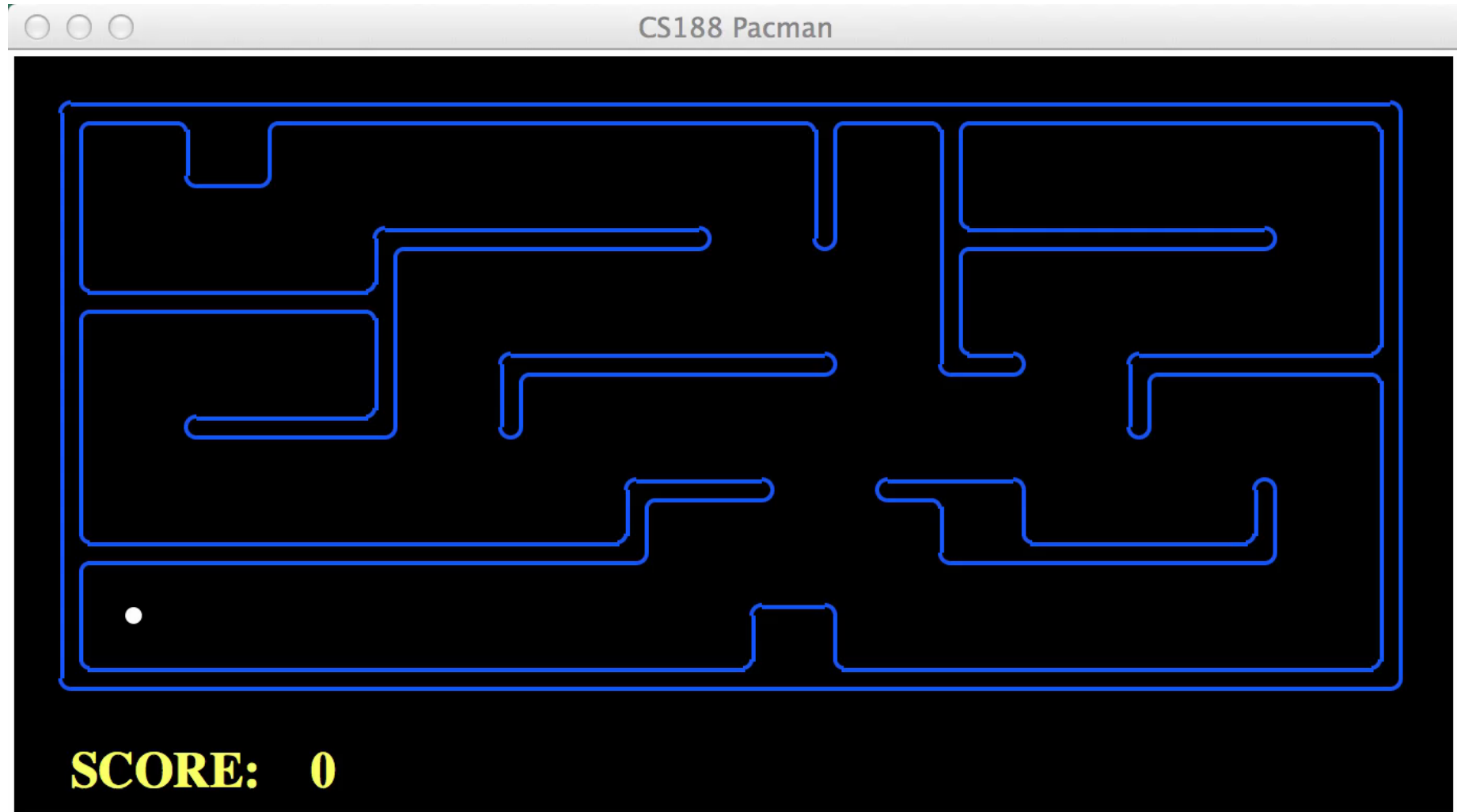
What search strategy is this?

A* search



What search strategy is this?

Uniform Cost Search (UCS)

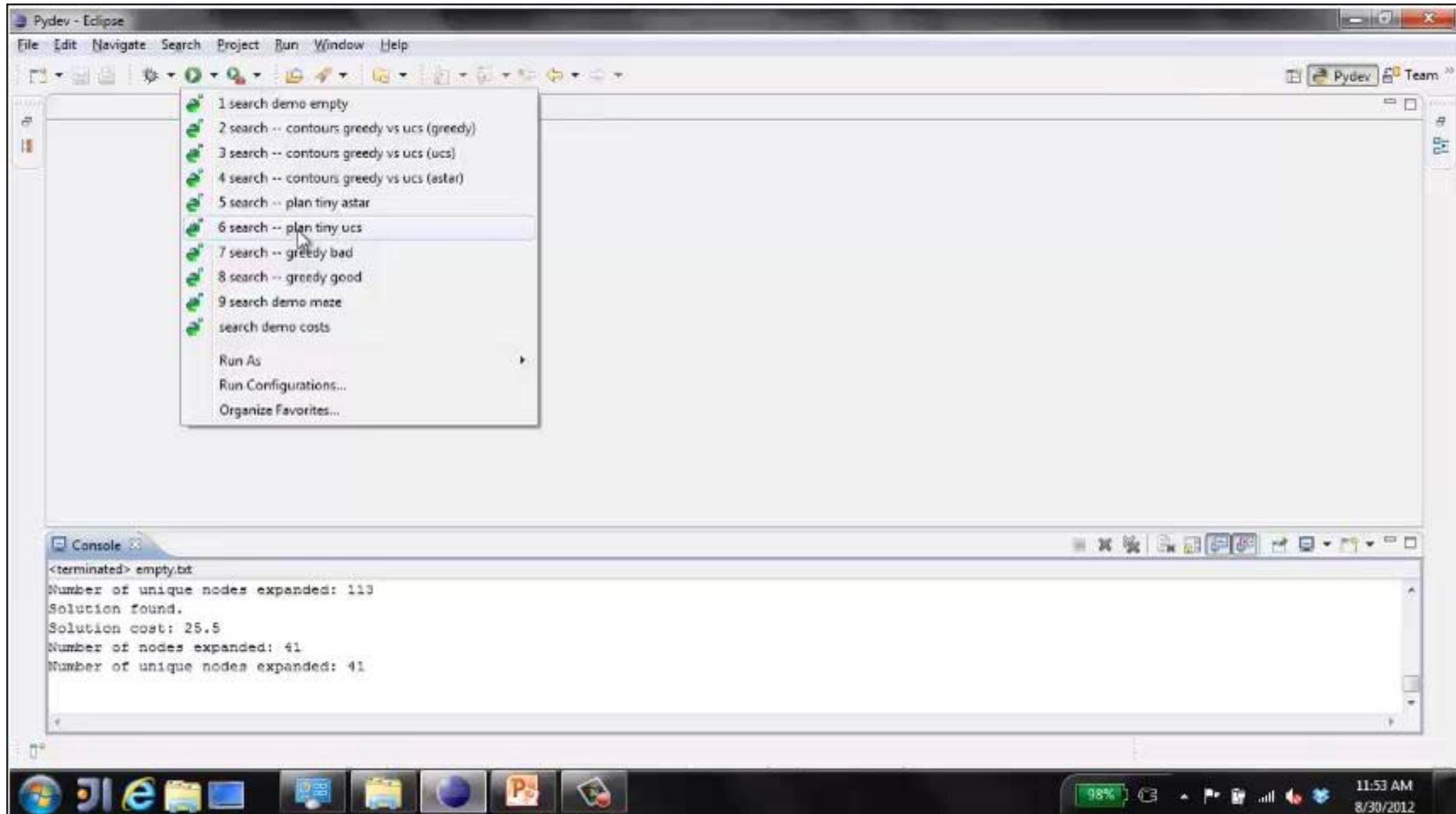


A* Applications

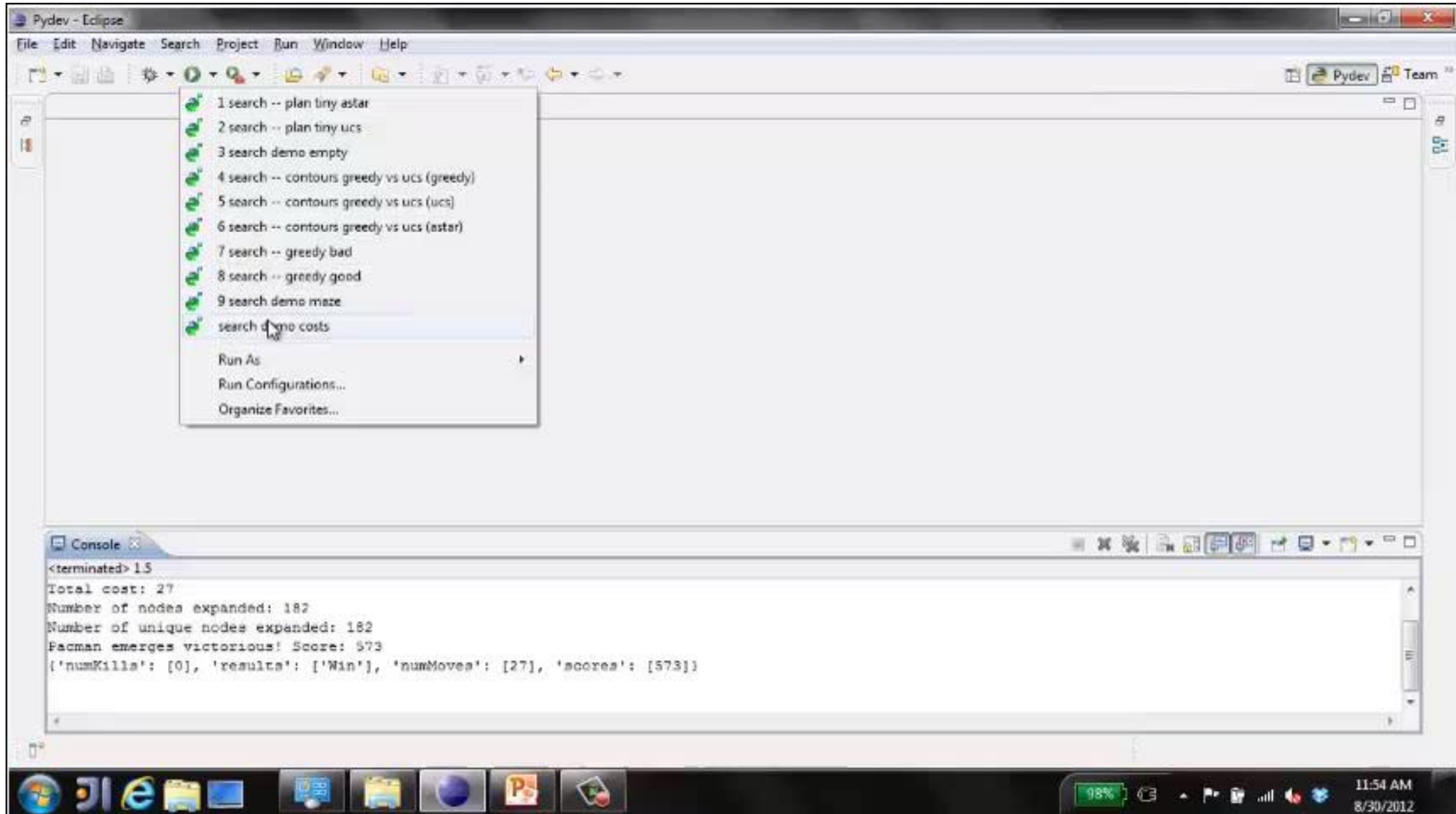
- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- ...



Video of Demo Pacman (Tiny Maze) – UCS / A*



Video of Demo Empty Water Shallow/Deep – Guess Algorithm

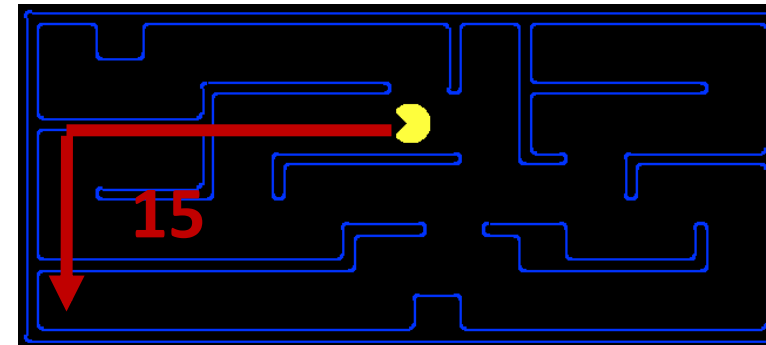
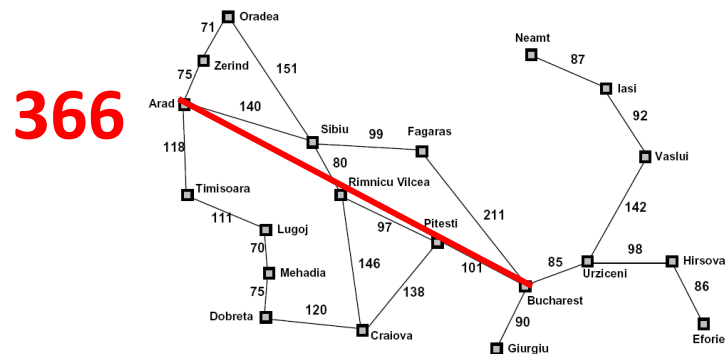


Creating Heuristics



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

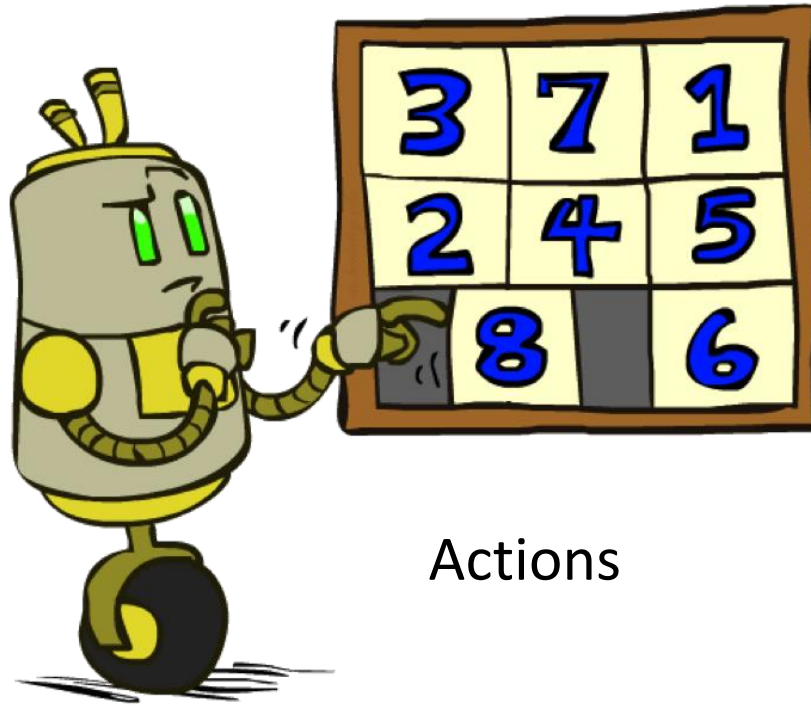


- Inadmissible heuristics are often useful too

Example: 8 Puzzle

7	2	4
5		6
8	3	1

Start State



Actions

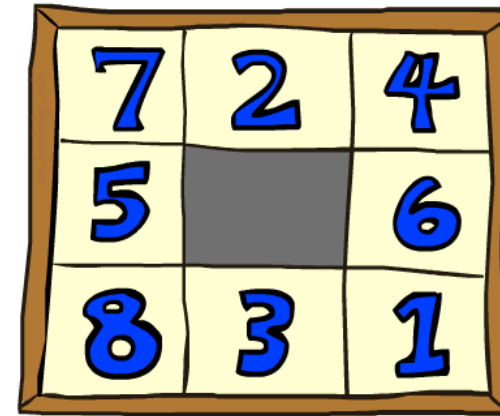
	1	2
3	4	5
6	7	8

Goal State

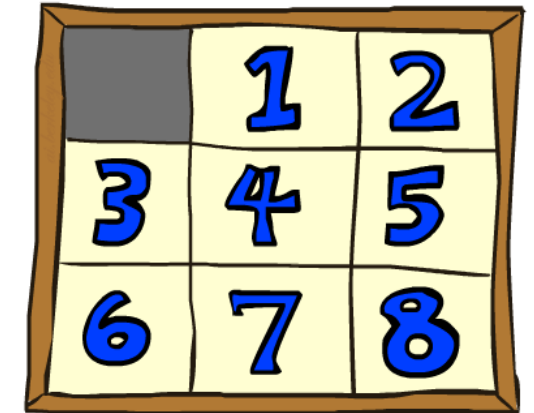
- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

8 Puzzle I

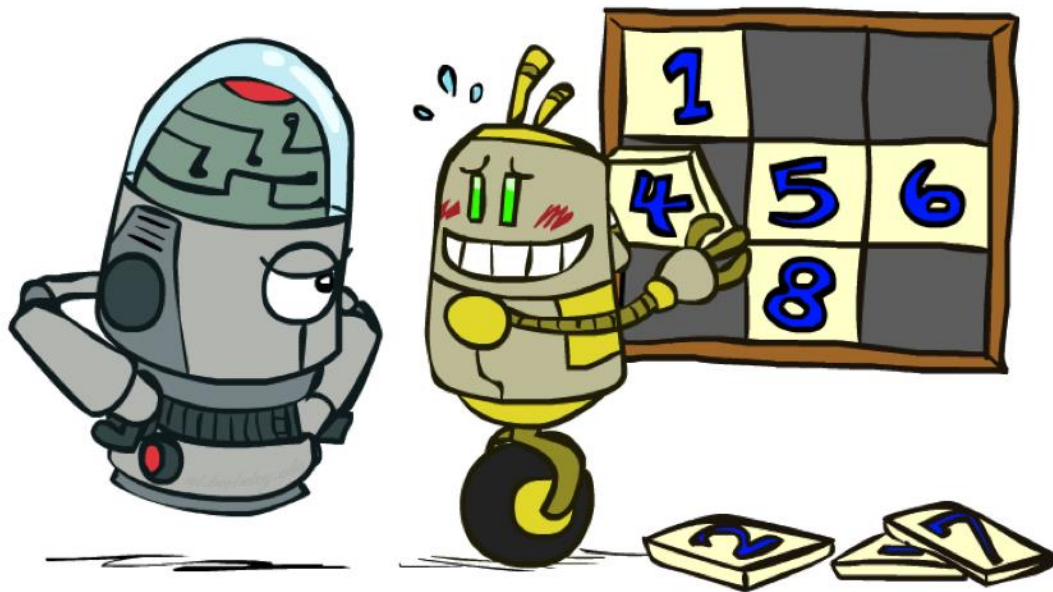
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- $h(\text{start}) = 8$
- This is a *relaxed-problem* heuristic



Start State



Goal State

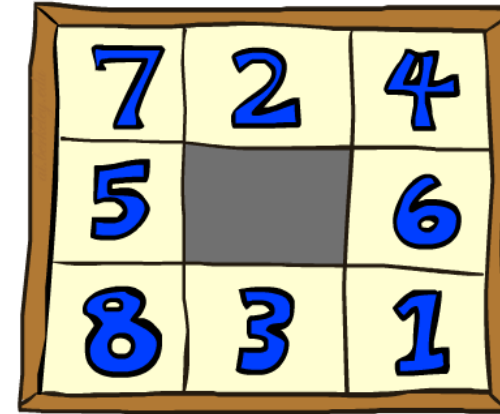


Average nodes expanded
when the optimal path has...

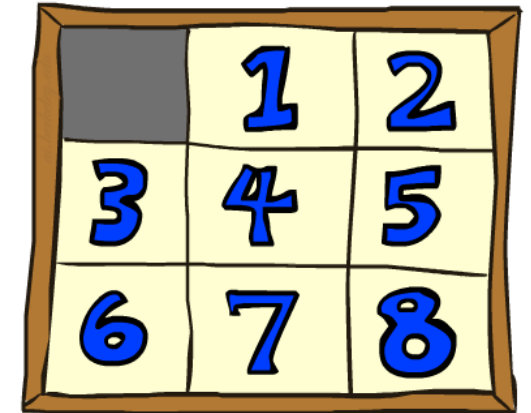
	...4 steps	...8 steps	...12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

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?
- $h(\text{start}) = 3 + 1 + 2 + \dots = 18$



Start State

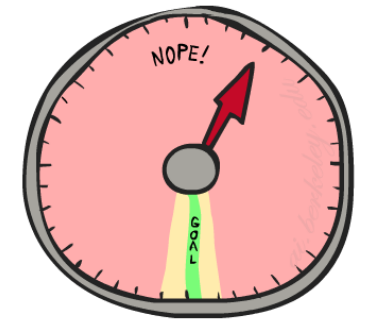
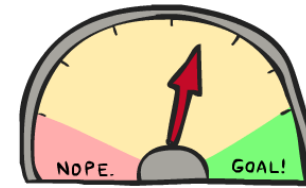


Goal State

	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

Trivial Heuristics, Dominance

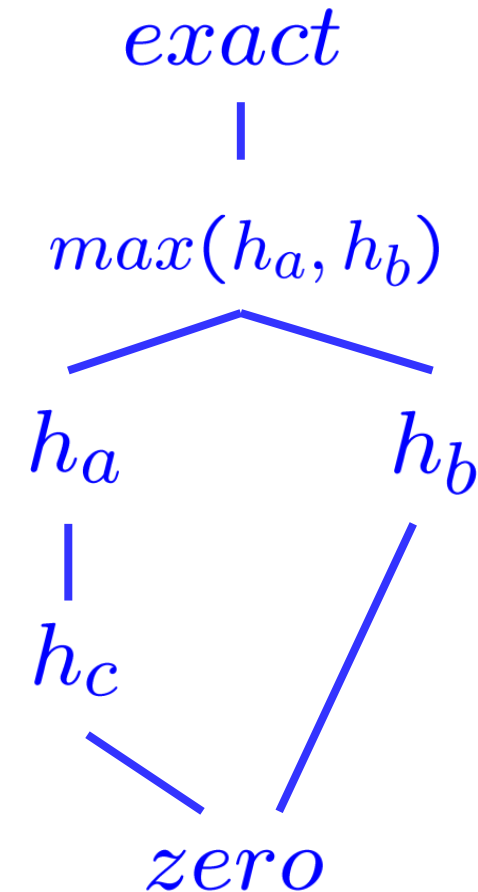
- Dominance: $h_a \geq h_c$ if

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

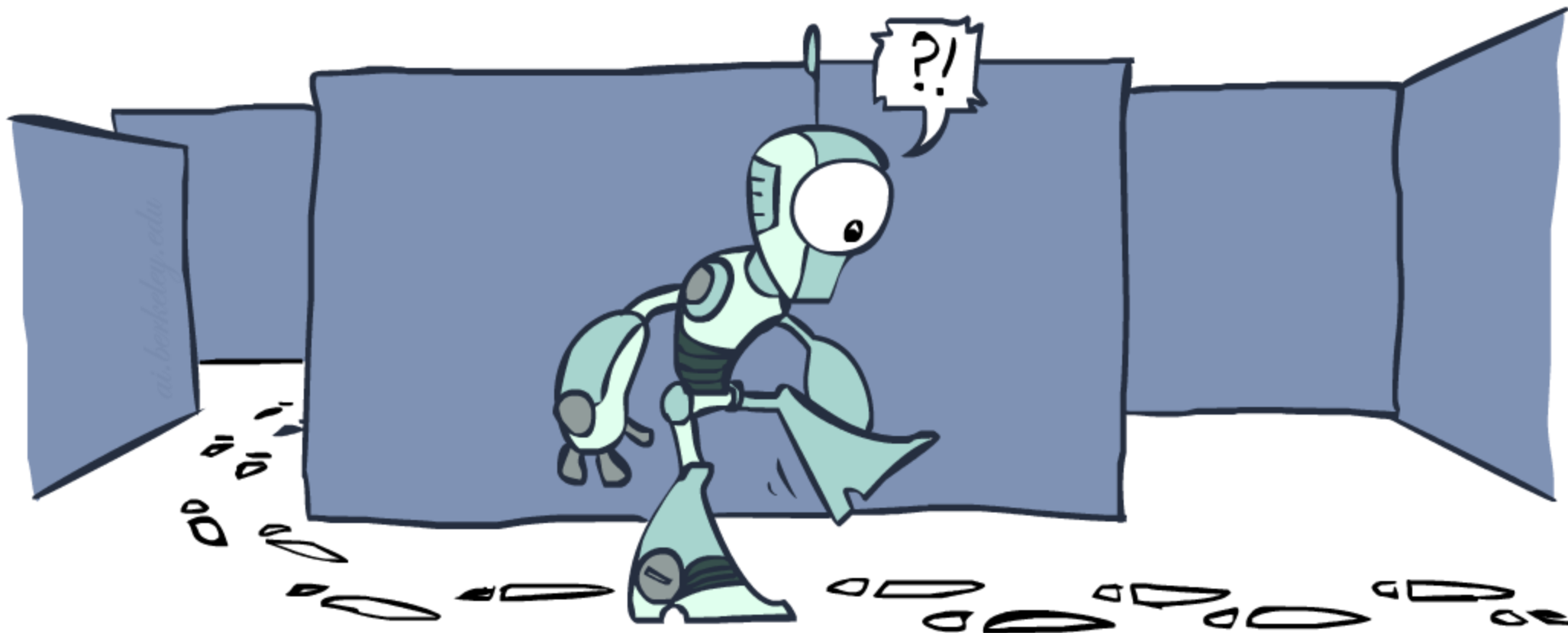
- Heuristics form a semi-lattice:
 - Max of admissible heuristics is admissible

$$h(n) = \max(h_a(n), h_b(n))$$

- Trivial heuristics
 - Bottom of lattice is the zero heuristic (what does this give us?)
 - Top of lattice is the exact heuristic

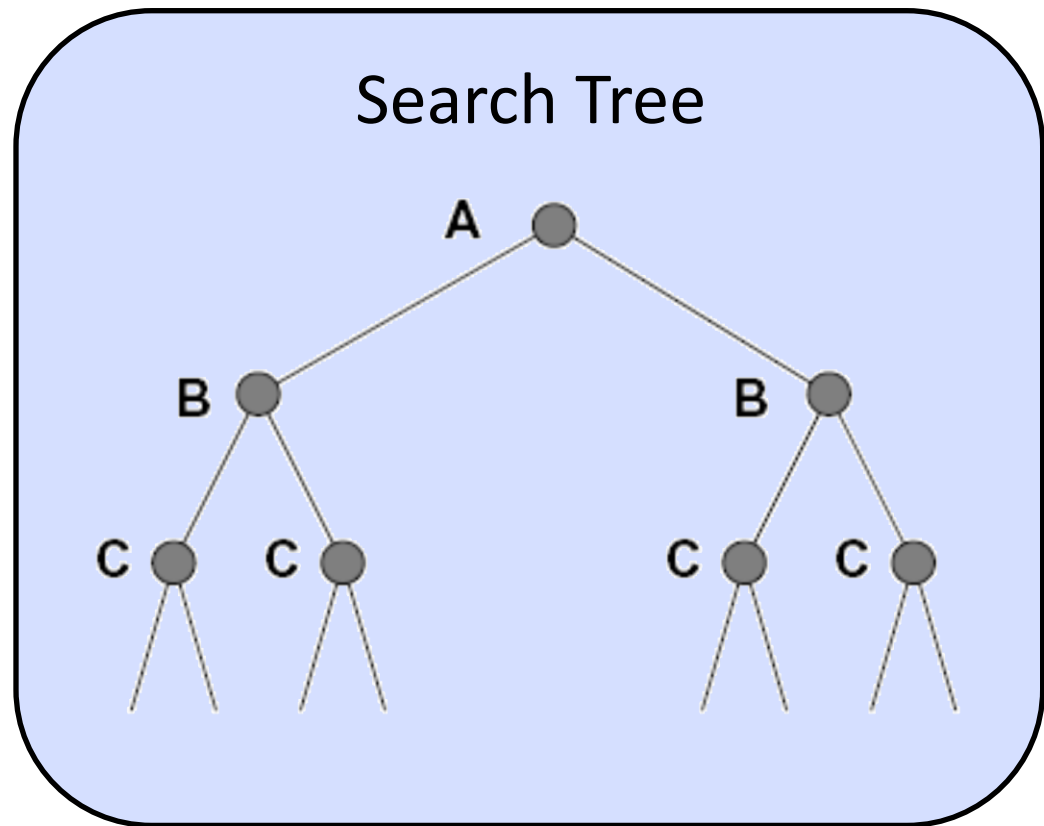
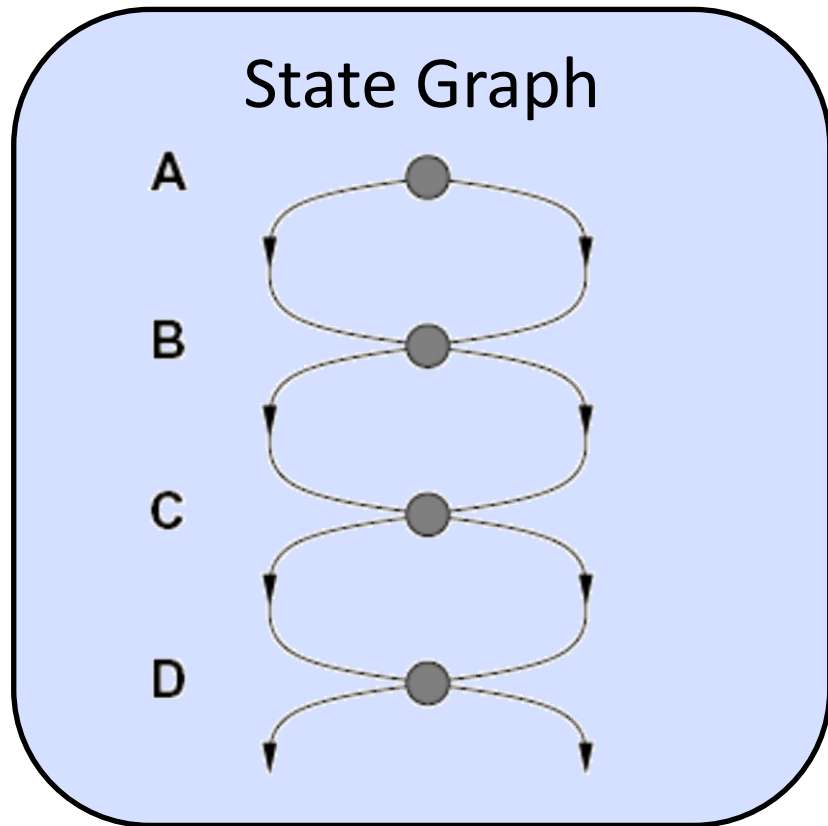


Graph Search



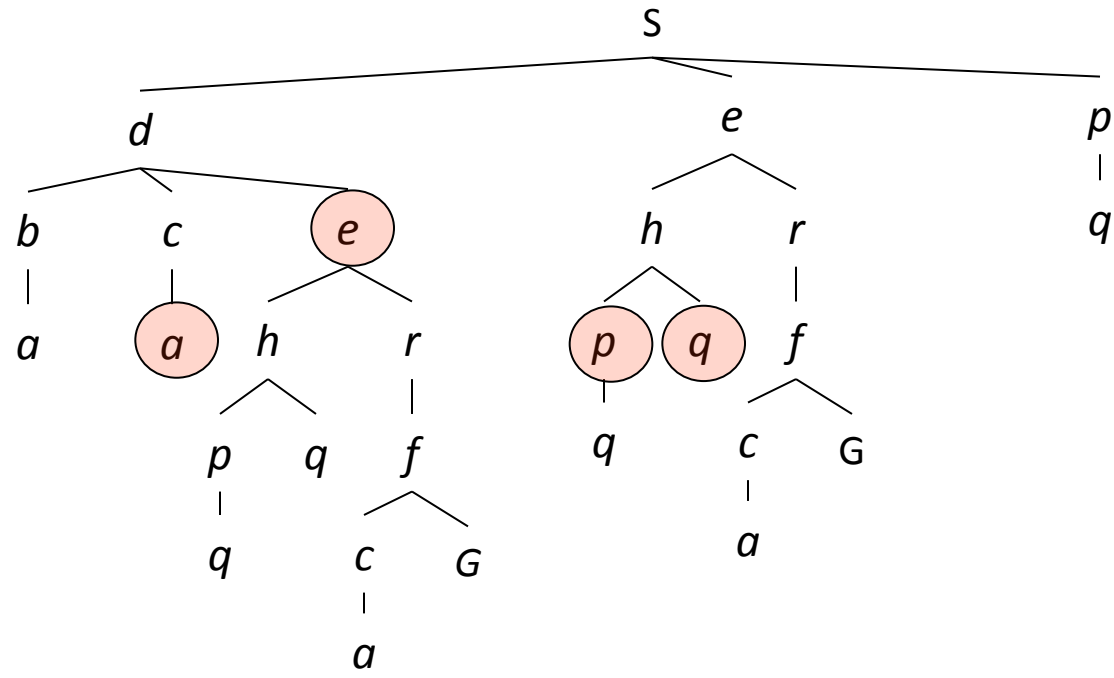
Tree Search: Extra Work!

- 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?)

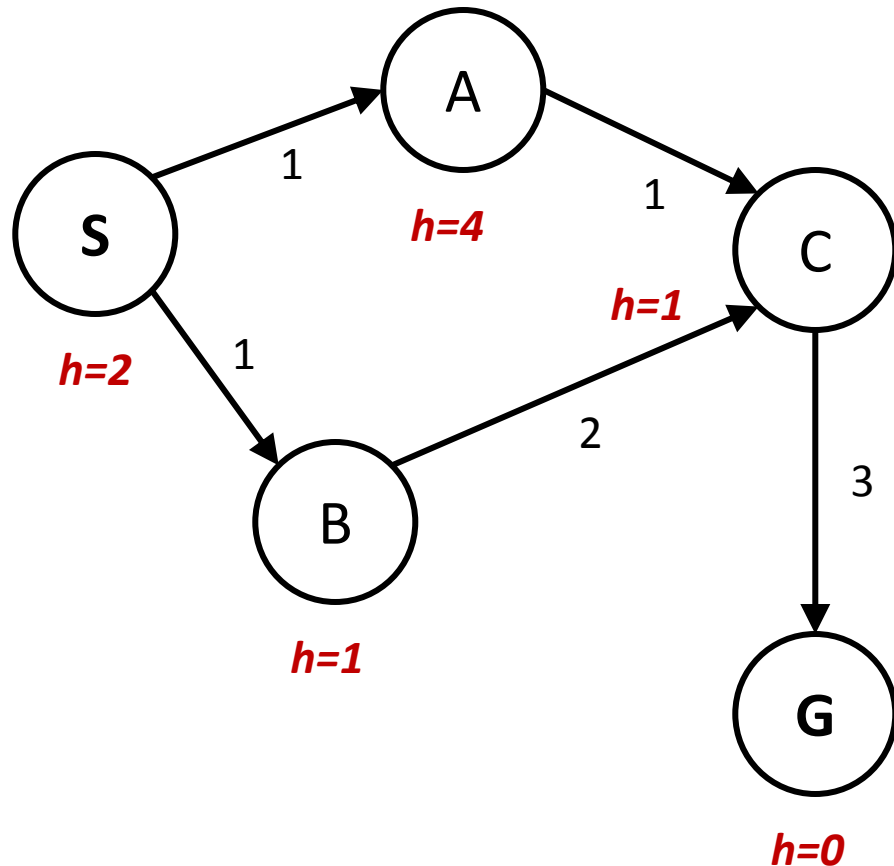


Graph Search

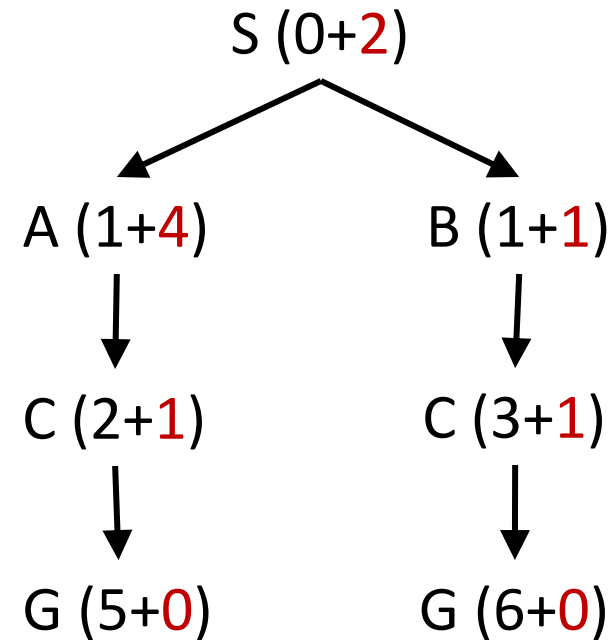
- Idea: never **expand** a state twice
- How to implement:
 - Tree search + set of expanded states (“closed set”)
 - Expand the search tree node-by-node, but...
 - Before expanding a node, check to make sure its state has never been expanded before
 - If not new, skip it, if new add to closed set
- Important: **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?

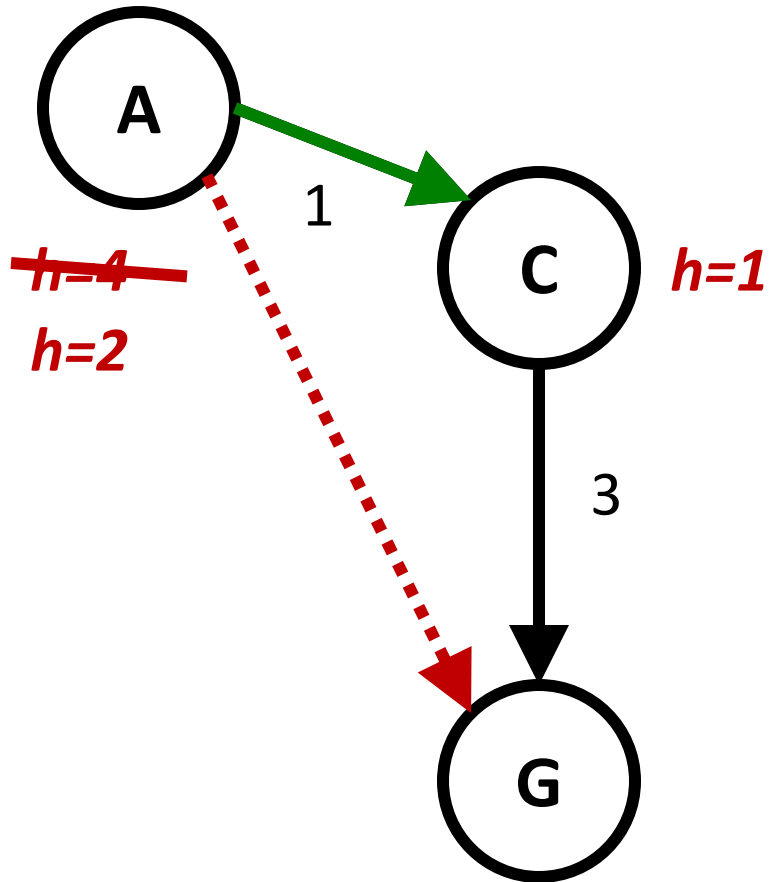
State space graph



Search tree



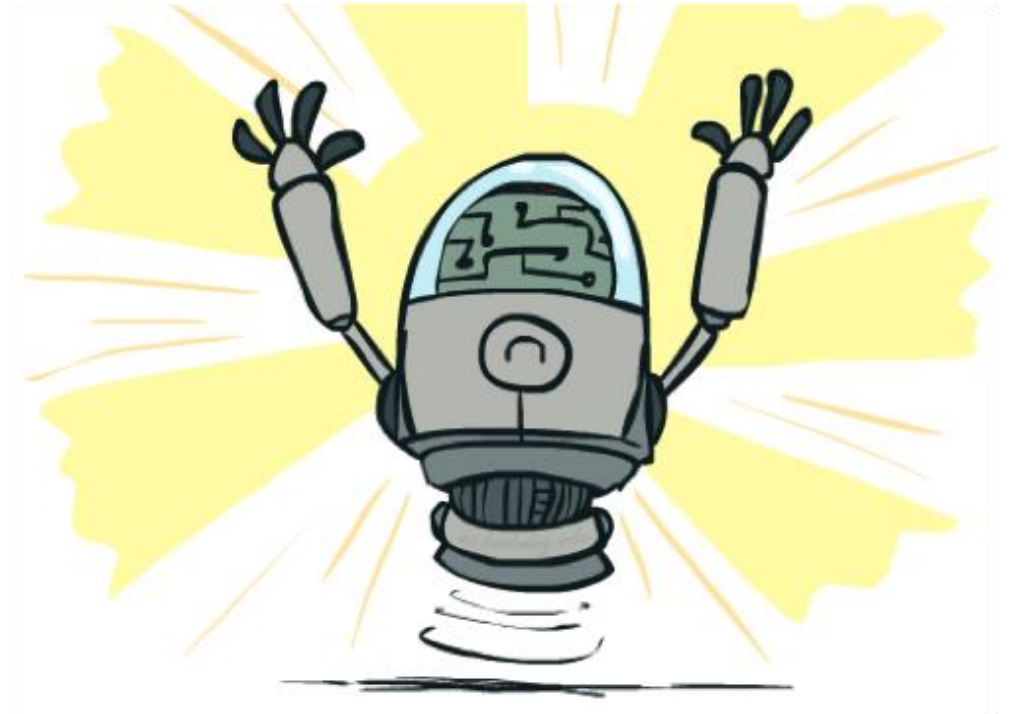
Consistency of Heuristics



- Main idea: estimated heuristic costs \leq actual costs
 - Admissibility: heuristic cost \leq actual cost to goal
$$h(A) \leq \text{actual cost from A to G}$$
 - Consistency: heuristic “arc” cost \leq actual cost for each arc
$$h(A) - h(C) \leq \text{cost}(A \text{ to } C)$$
- Consequences of consistency:
 - The f value along a path never decreases
$$h(A) \leq \text{cost}(A \text{ to } C) + h(C)$$
 - A* graph search is optimal

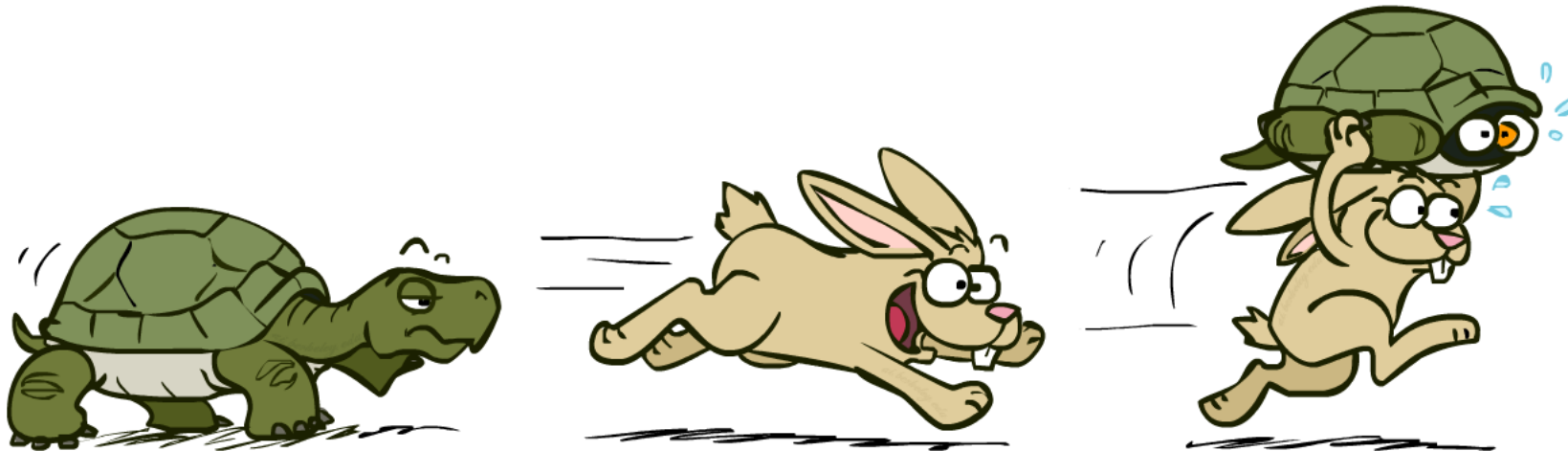
Optimality

- 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
- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems



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
  end
```

Some Hints for P1

- Graph search is almost always better than tree search (when not?)
- Implement your closed list as a dict or set!
- Nodes are conceptually paths, but better to represent with a state, cost, last action, and reference to the parent node