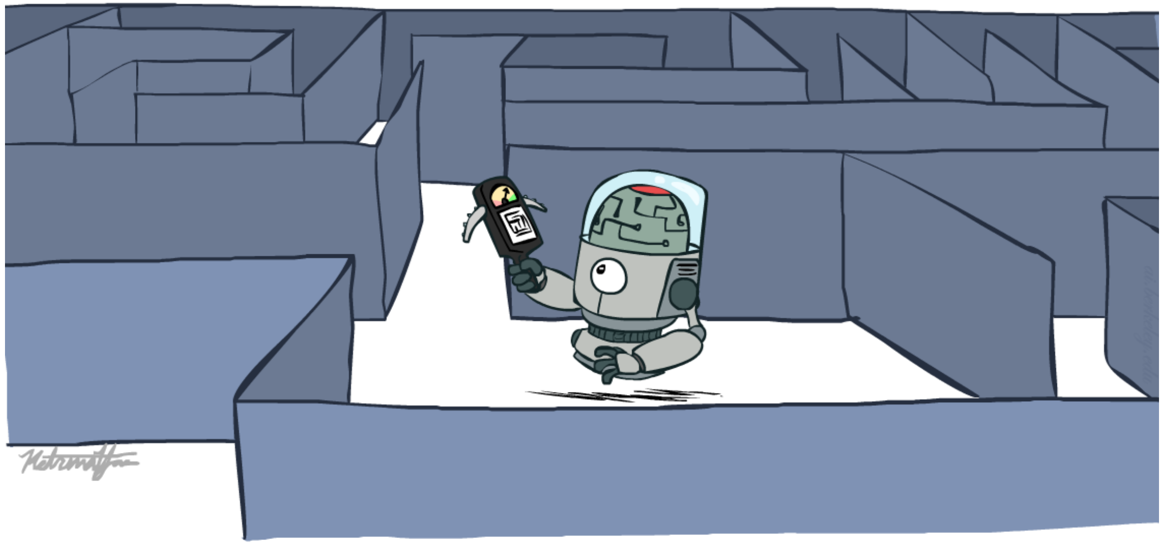
COMS W4701: Artificial Intelligence

Lecture 4: Informed Search



Instructor: Tony Dear

*Lecture materials derived from UC Berkeley's AI course at ai.berkeley.edu

Announcements

- HW 0 due today!
- HW 1 out now, due in two weeks (Sept 27)

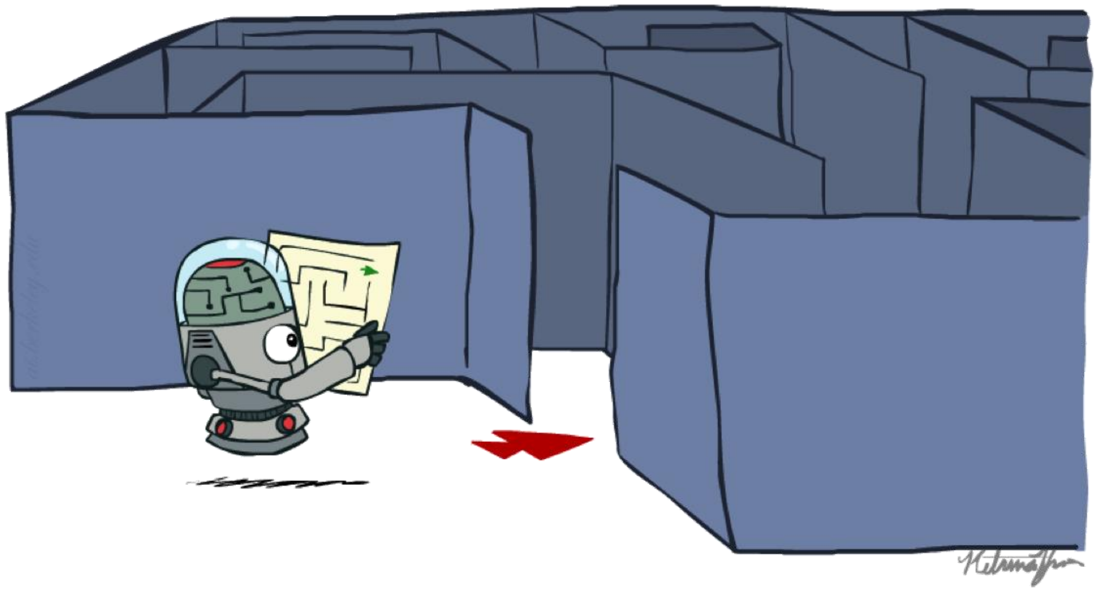
- Weekly review sessions
  - Friday 1-2pm, 214 Pupin
  - Friday 4-5pm, 420 Pupin

Today

- Recap uninformed search
- Heuristics
- Greedy search
- A* search
  - Optimality
- Admissibility and consistency

Recap: Search

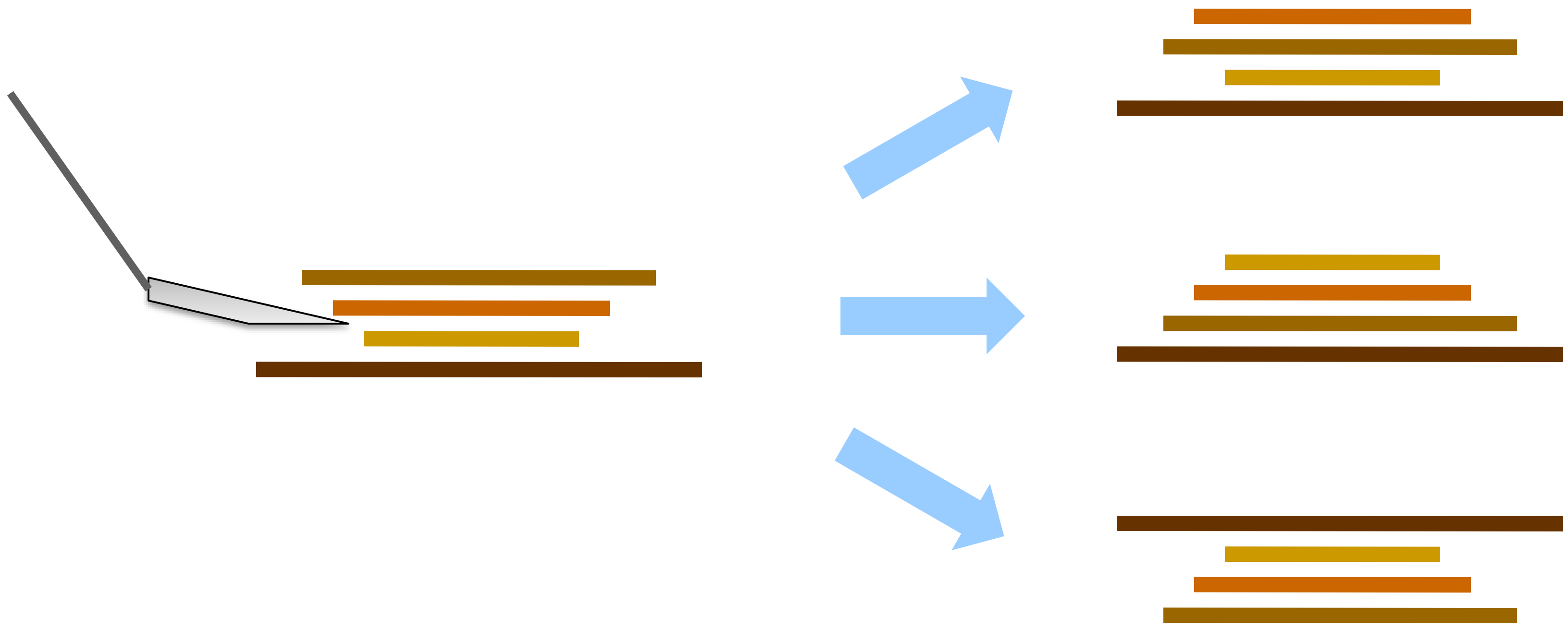
- Search problem:
  - States (configurations of the world)
  - Actions and costs
  - Transition model (world dynamics)
  - Start state and goal test



- Search tree:
  - Nodes: 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

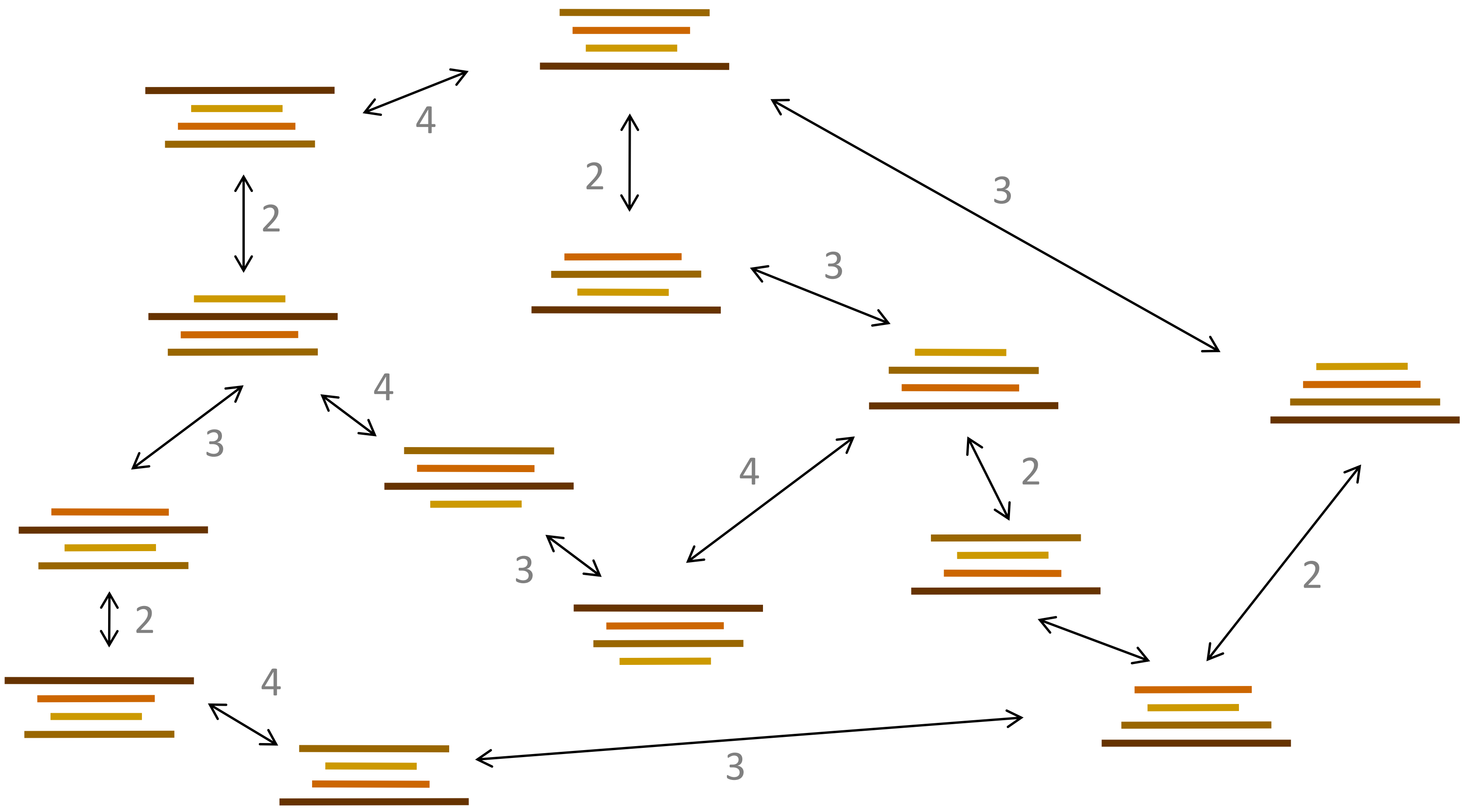
Example: Pancake Problem



Cost: Number of pancakes flipped

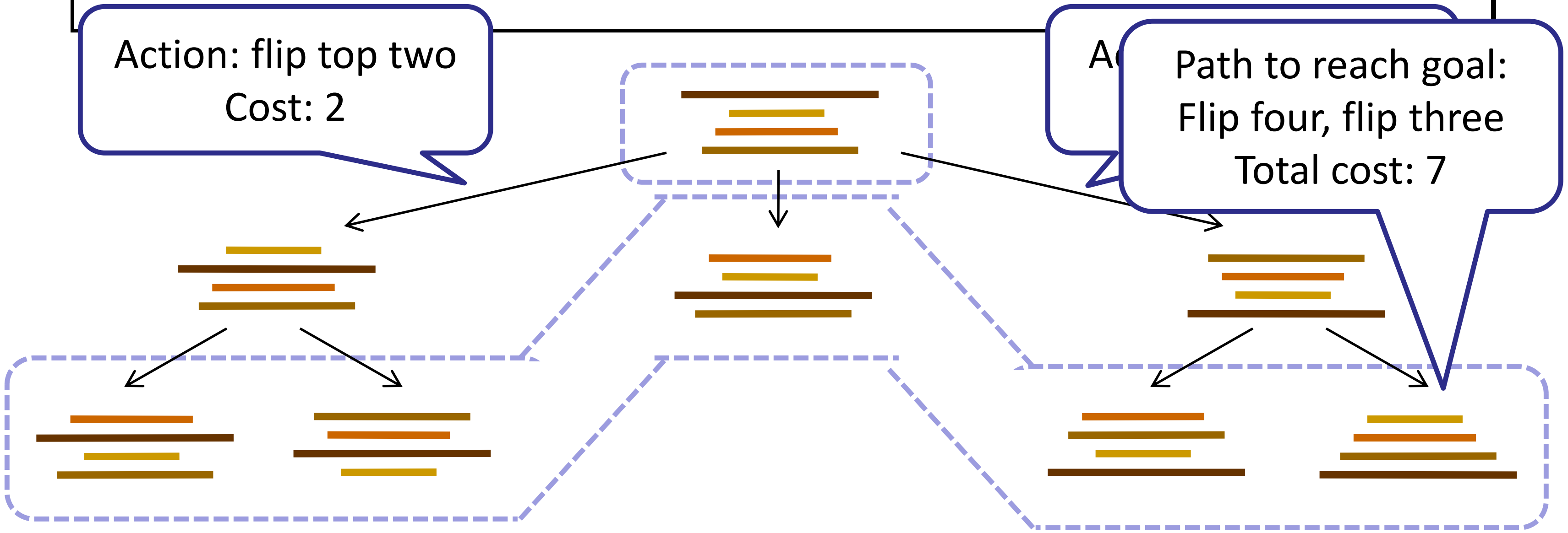
Example: Pancake Problem

State space graph with costs as weights



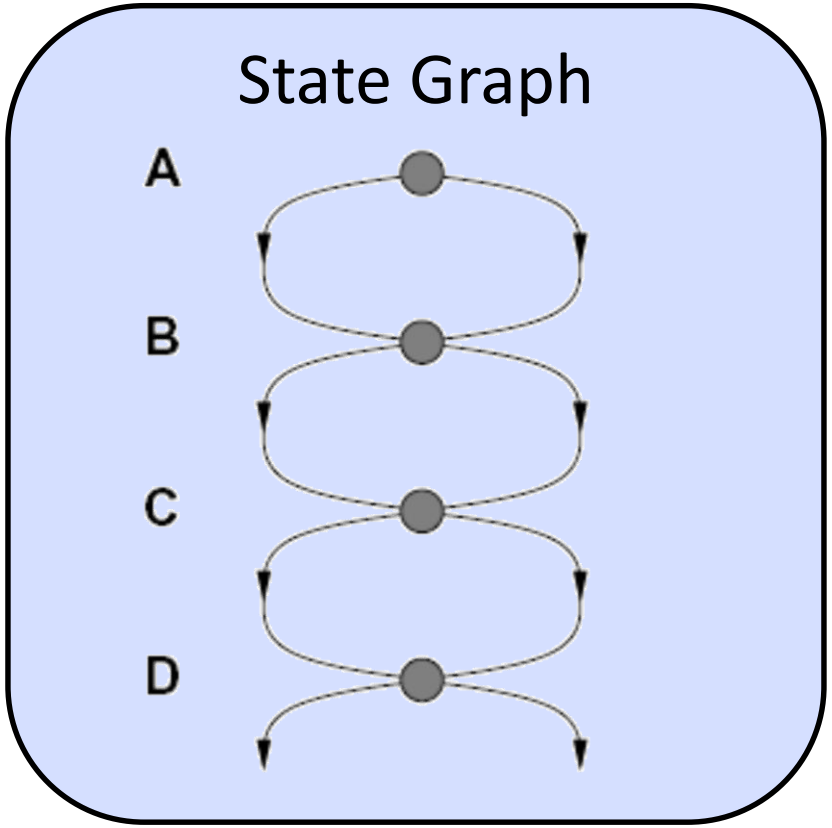
General Tree Search

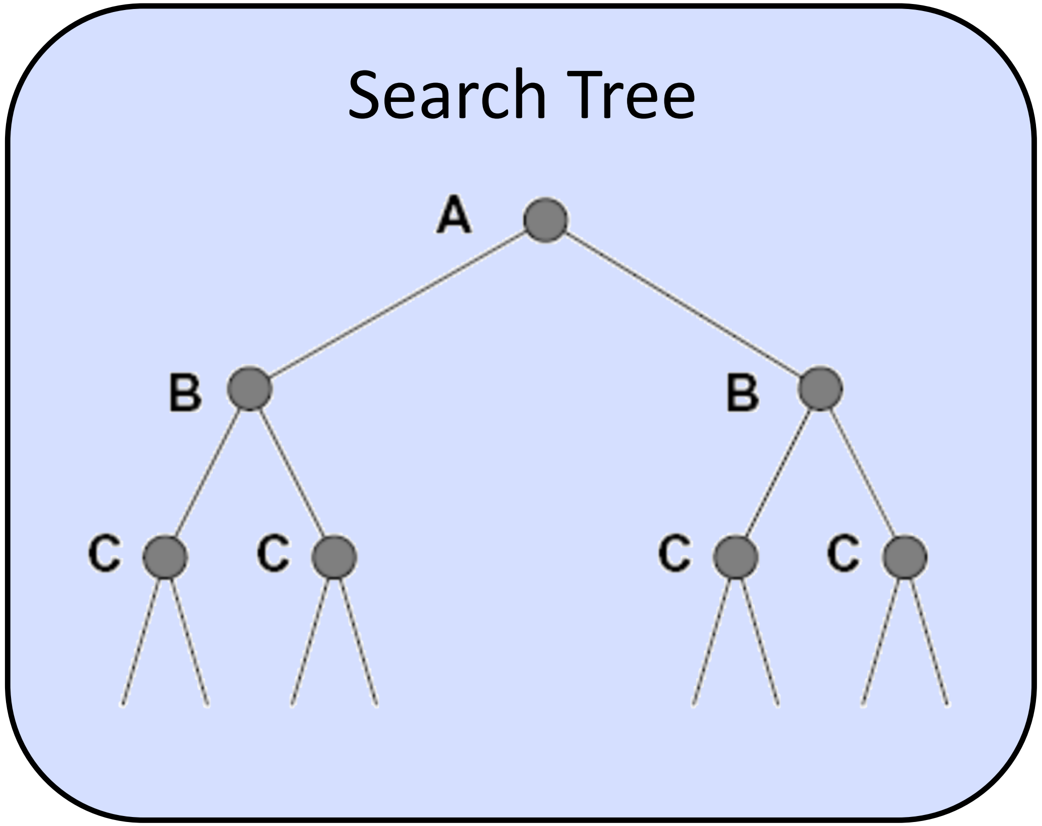
function TREE-SEARCH(*problem, strategy*) **returns** a solution, or failure
  initialize the search tree using the initial state of *problem*
  loop do
    if there are no candidates for expansion **then return** failure
    choose a leaf node for expansion according to *strategy*
    if the node contains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



Redundant States

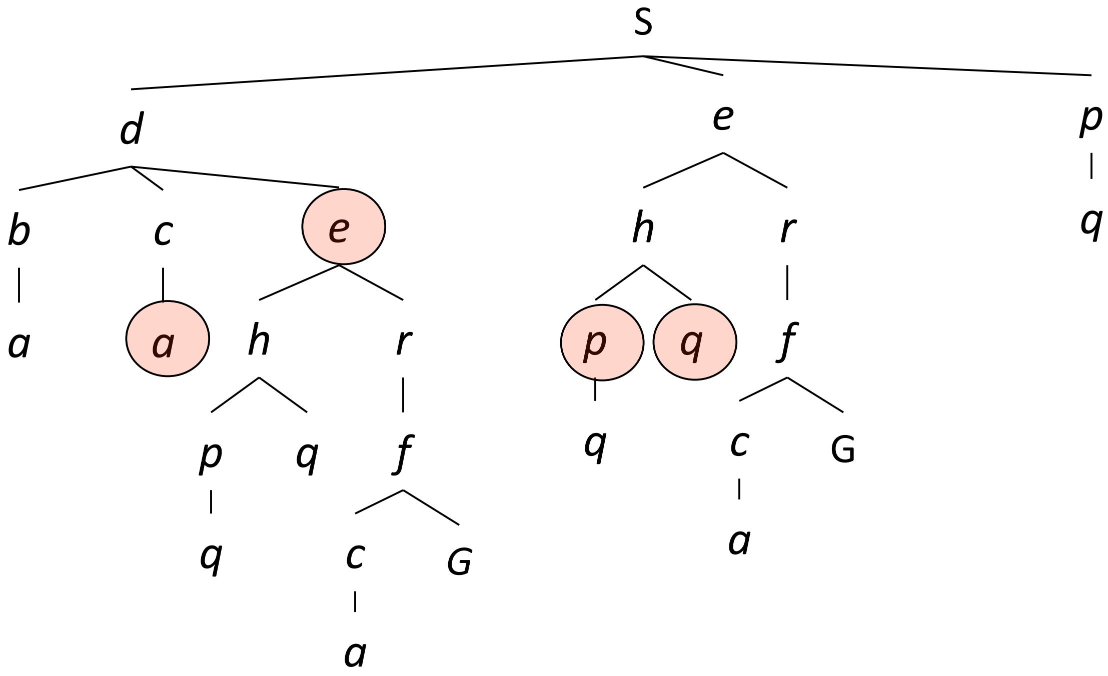
- Failure to detect repeated states can cause exponentially more work





Redundant States

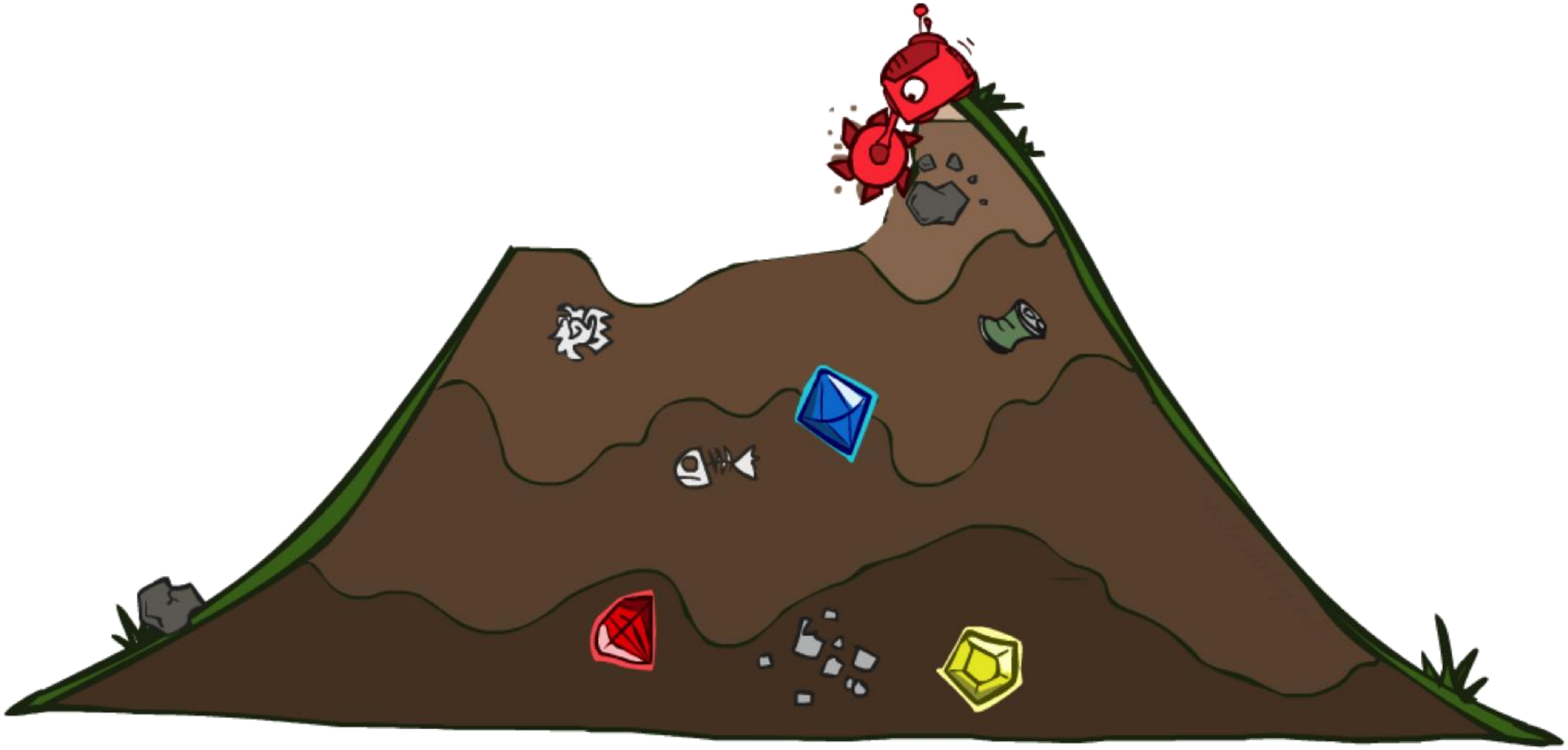
- In BFS, for example, we shouldn't bother expanding the circled nodes (why?)



Graph Search

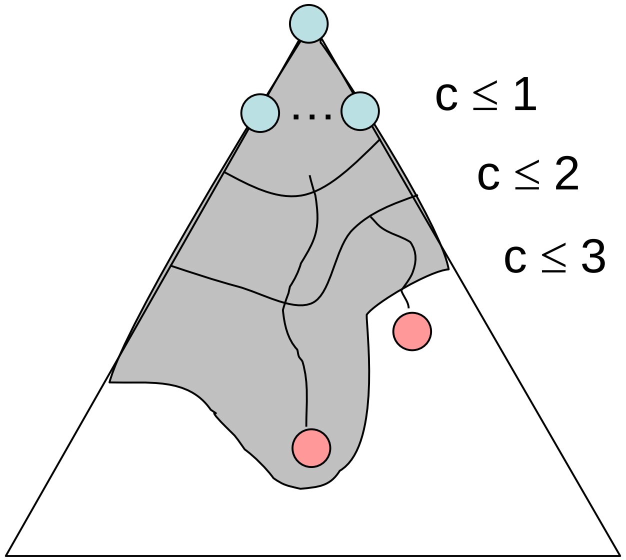
```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then 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
```

Uninformed Search



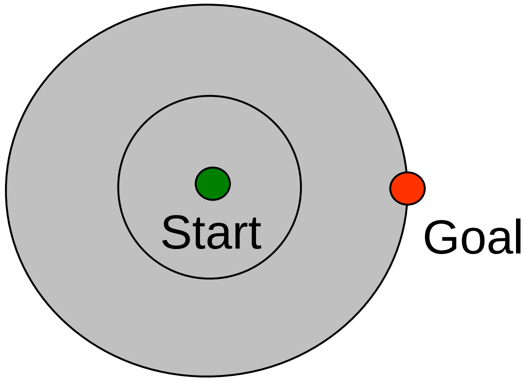
Uniform Cost Search

- Strategy: expand lowest path cost

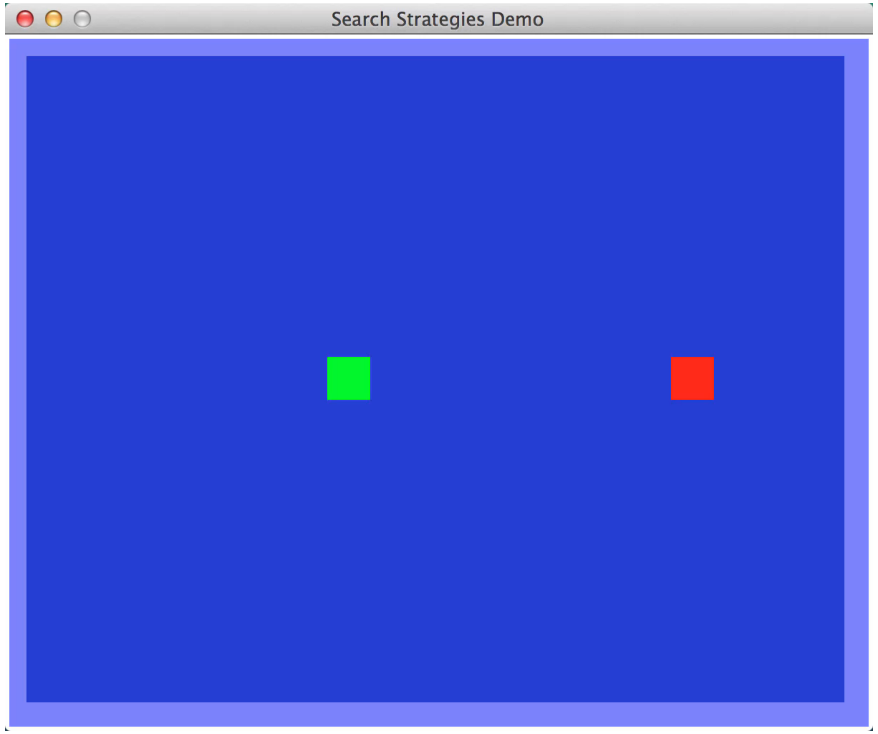


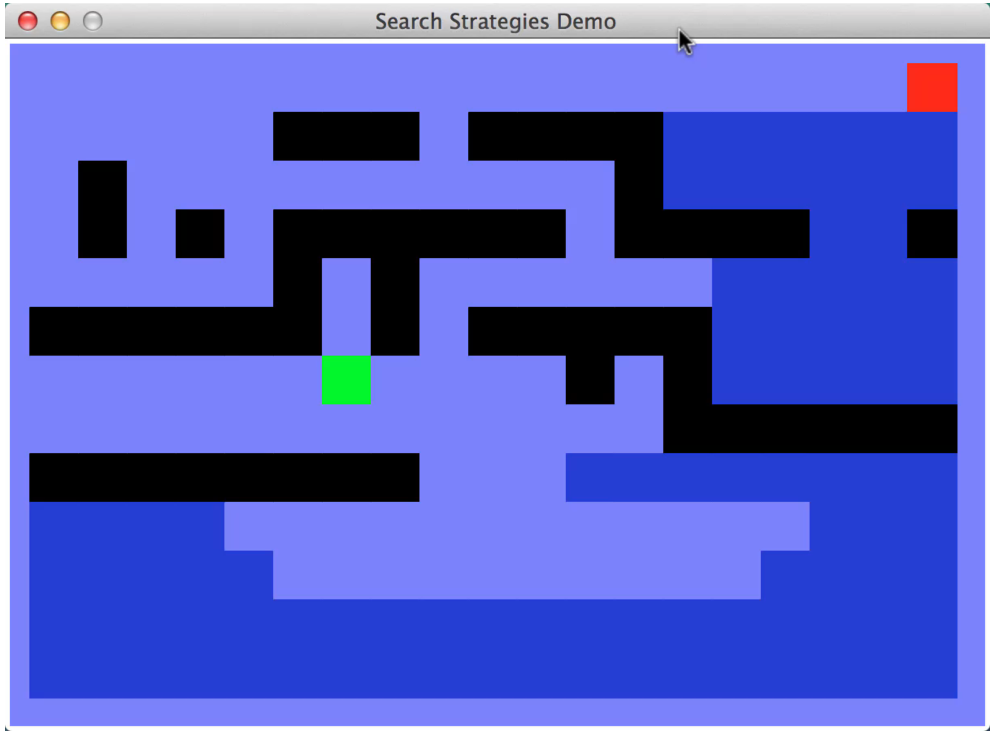
- The good: UCS is complete and optimal!

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



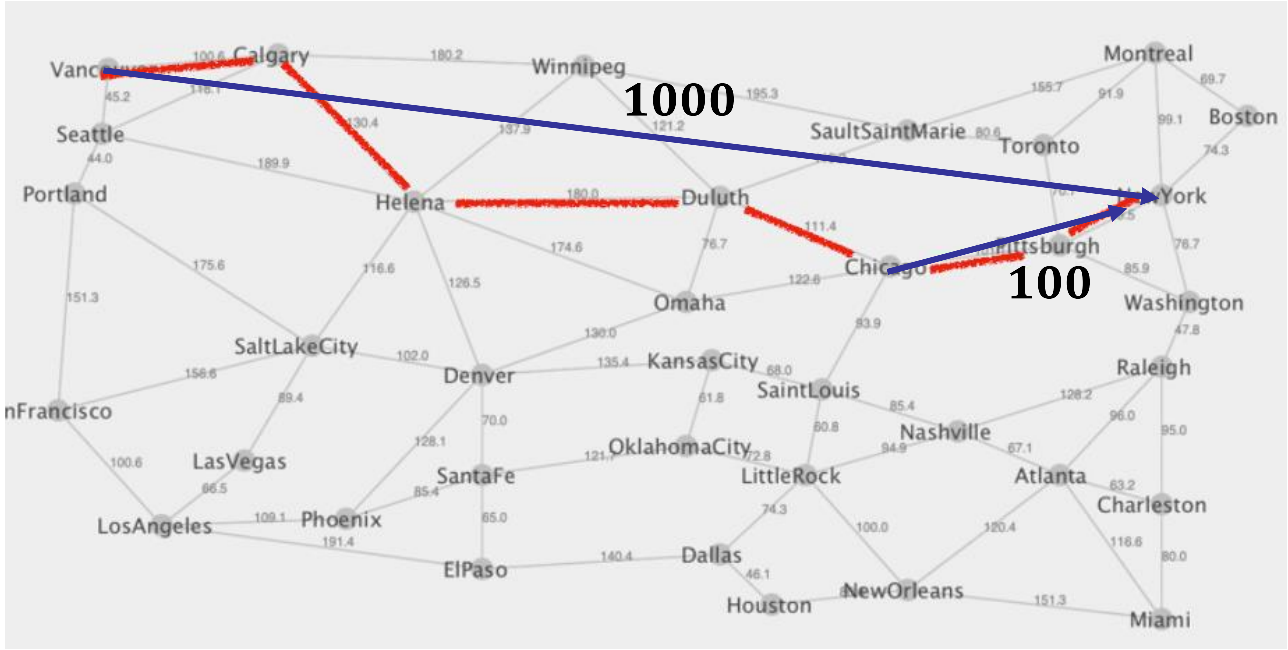
UCS Examples

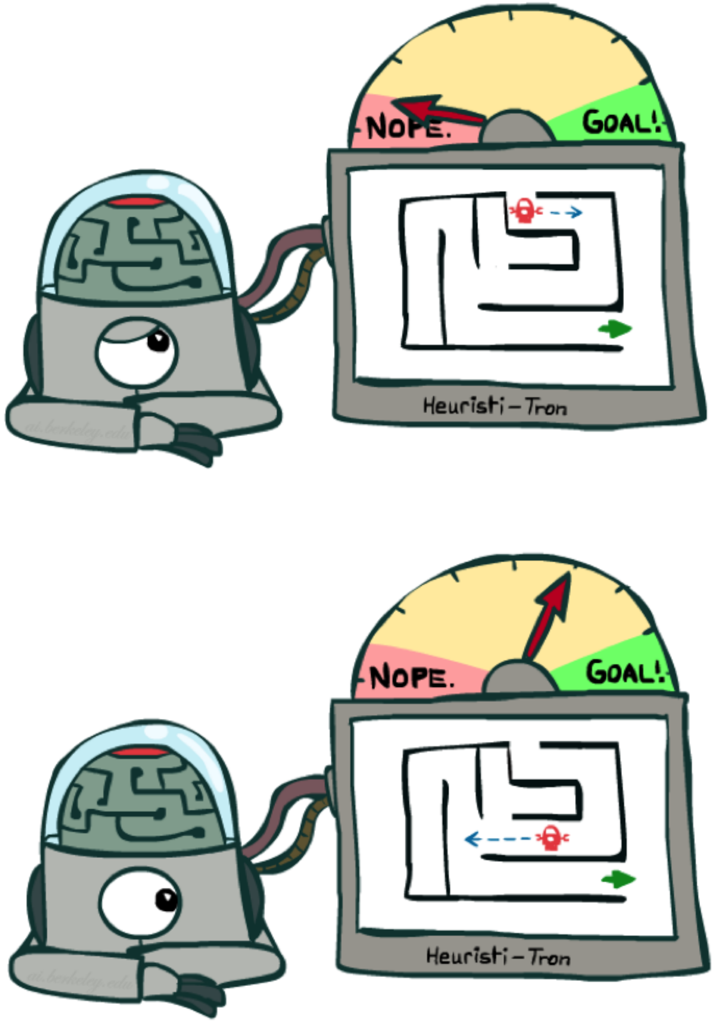




Search Heuristics

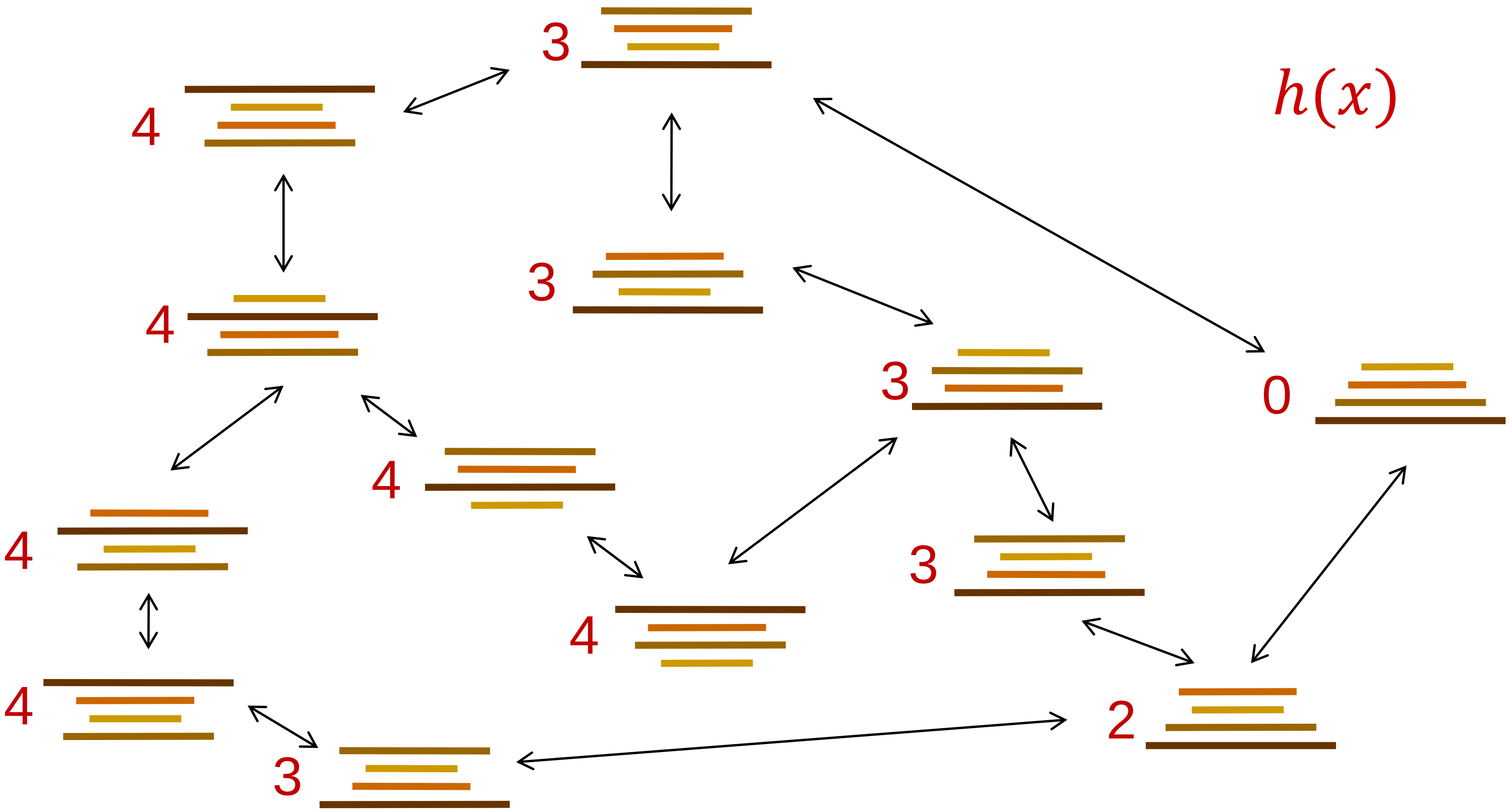
- Heuristic $h(s)$
  - A function that *estimates* how close a state is to a goal
  - Designed for a specific goal
  - Examples: Manhattan distance, Euclidean distance (for pathing)





Example: Heuristic Function

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

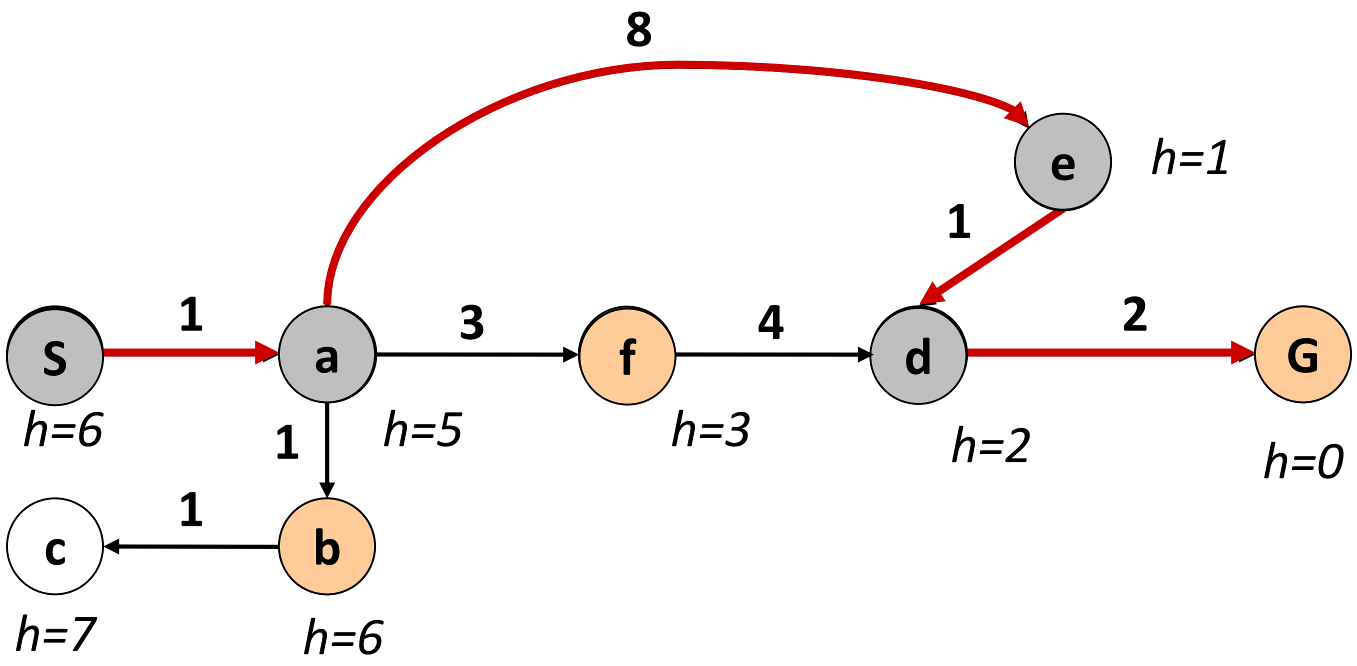


Greedy Search

Greedy Search

Strategy: expand node that appears to be closest to the goal

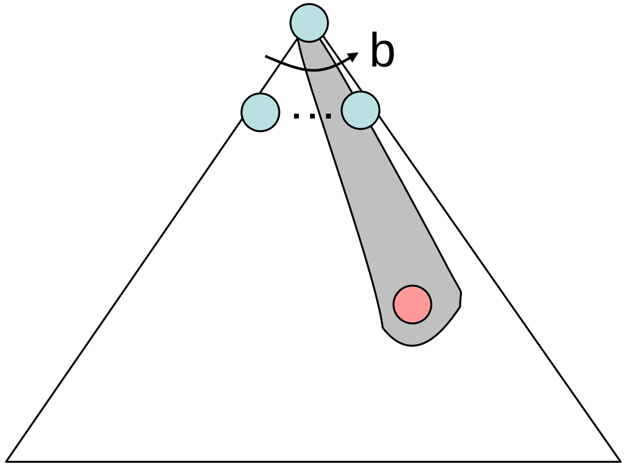
Fringe is a priority queue (priority: heuristic)



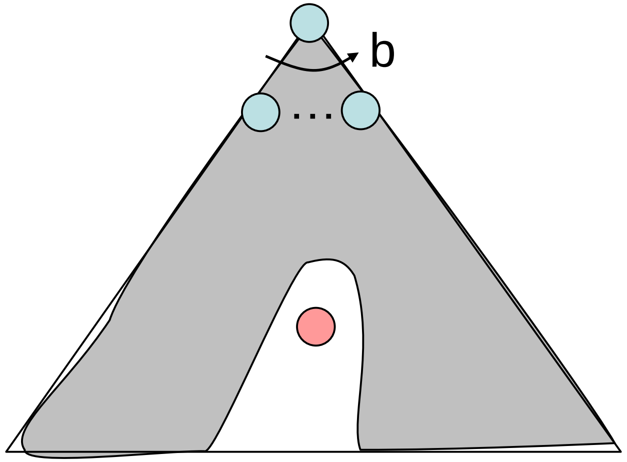
- Expand the node that seems closest…
- What can go wrong?

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

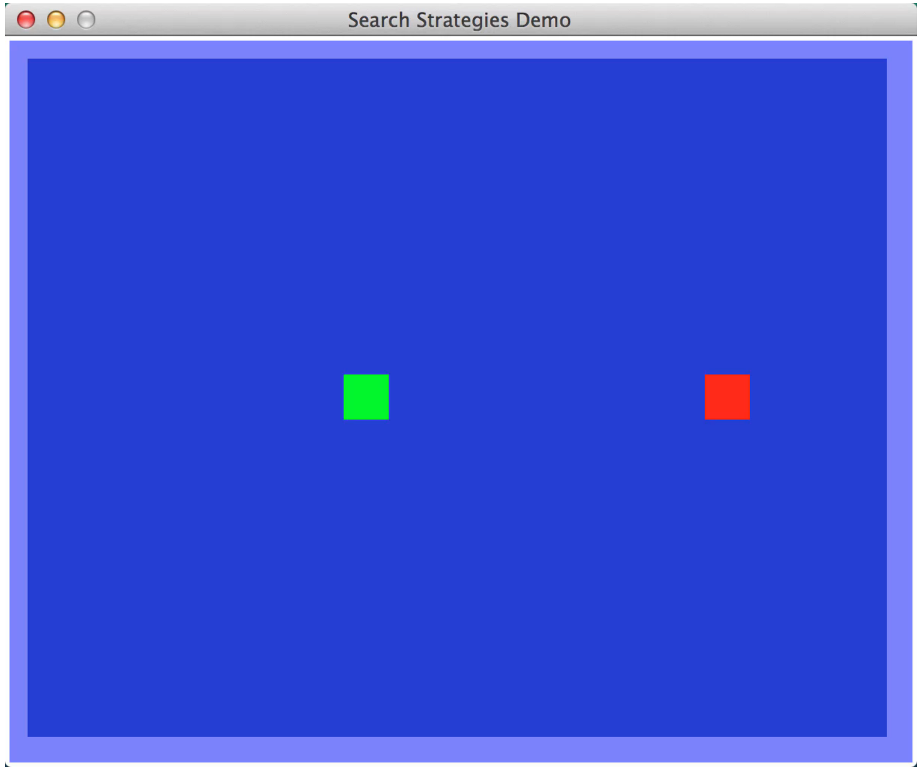


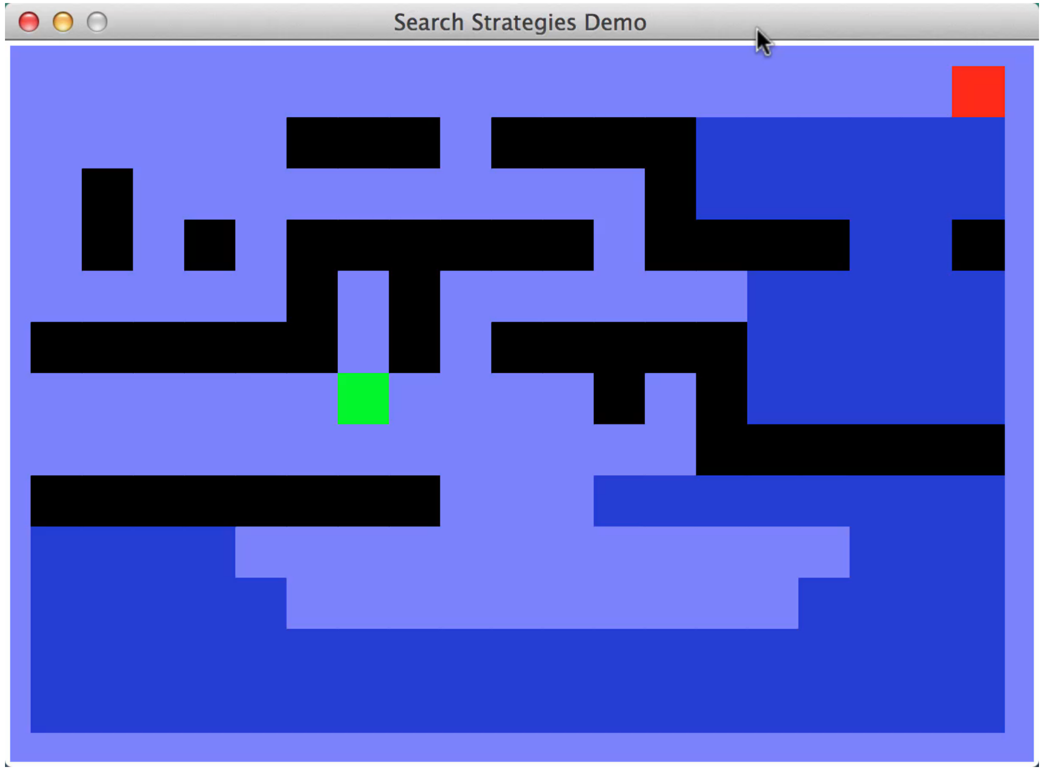
- A common case:
  - Greedy search takes you straight to a goal, regardless of its true cost



- Worst-case: like a badly-guided DFS
  - No guarantee of completeness or optimality!

Greedy Search Examples

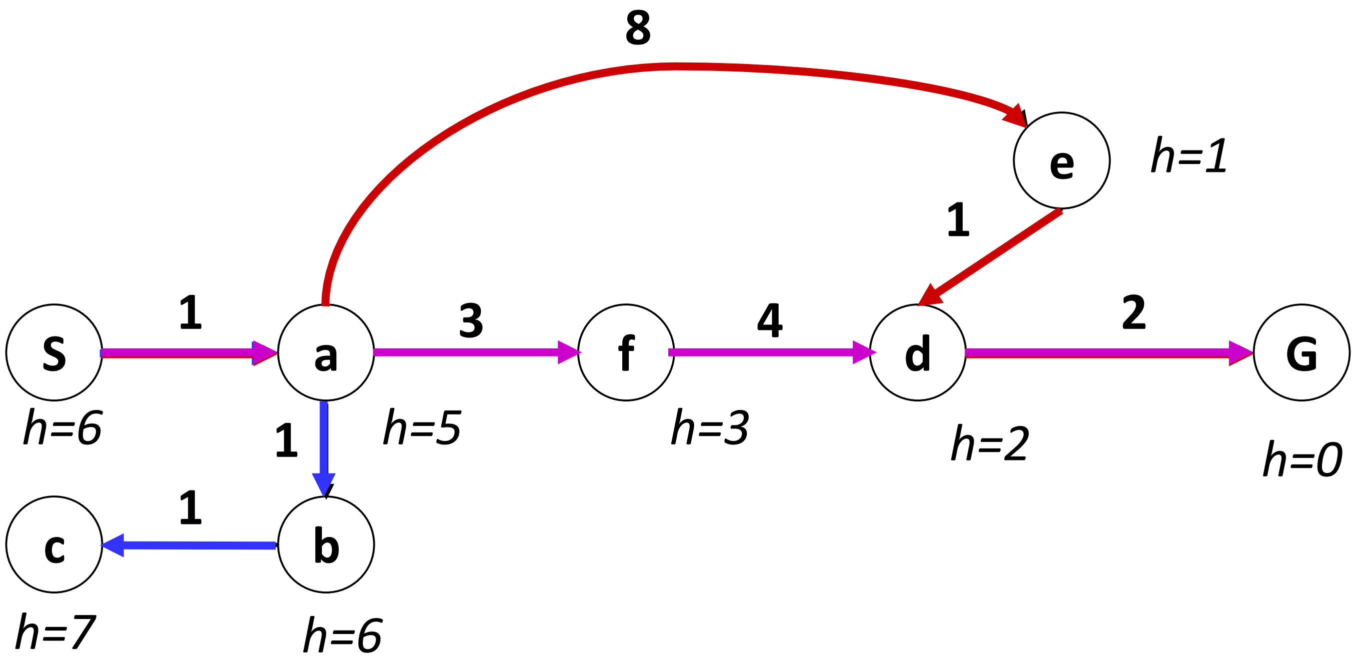




A* Search

Combining UCS and Greedy

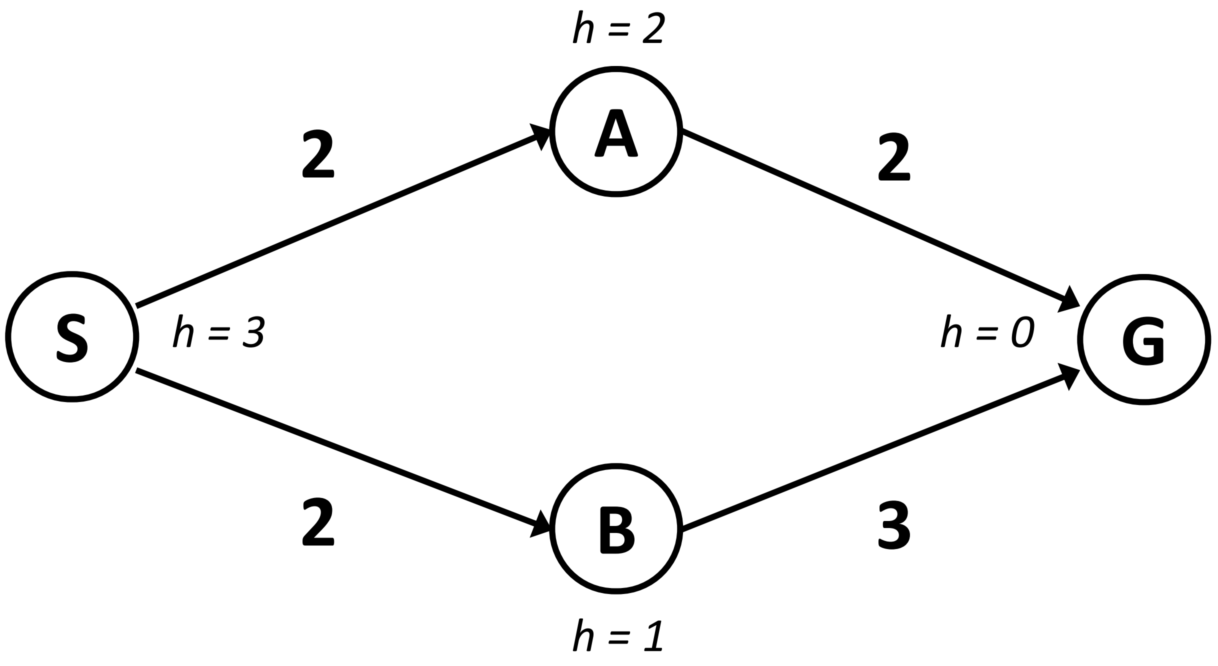
- Uniform-cost orders by path cost, or *backward cost* $g(n)$
- Greedy orders by goal proximity, or *forward cost* $h(n)$



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

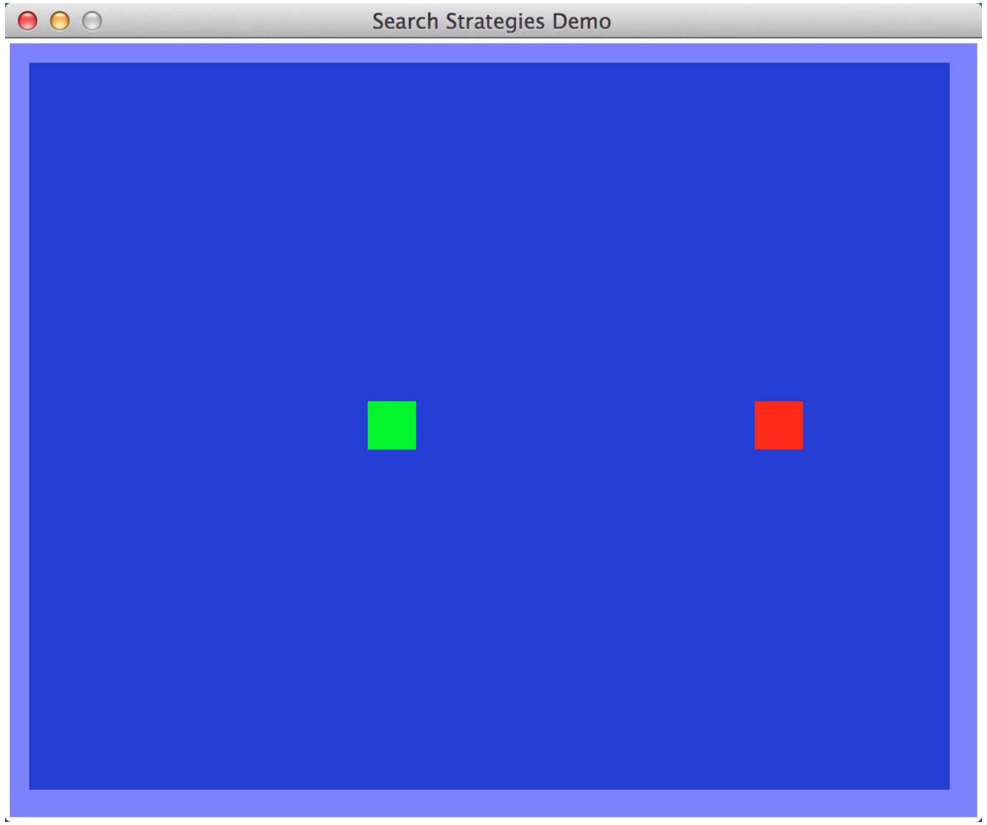
When should A* terminate?

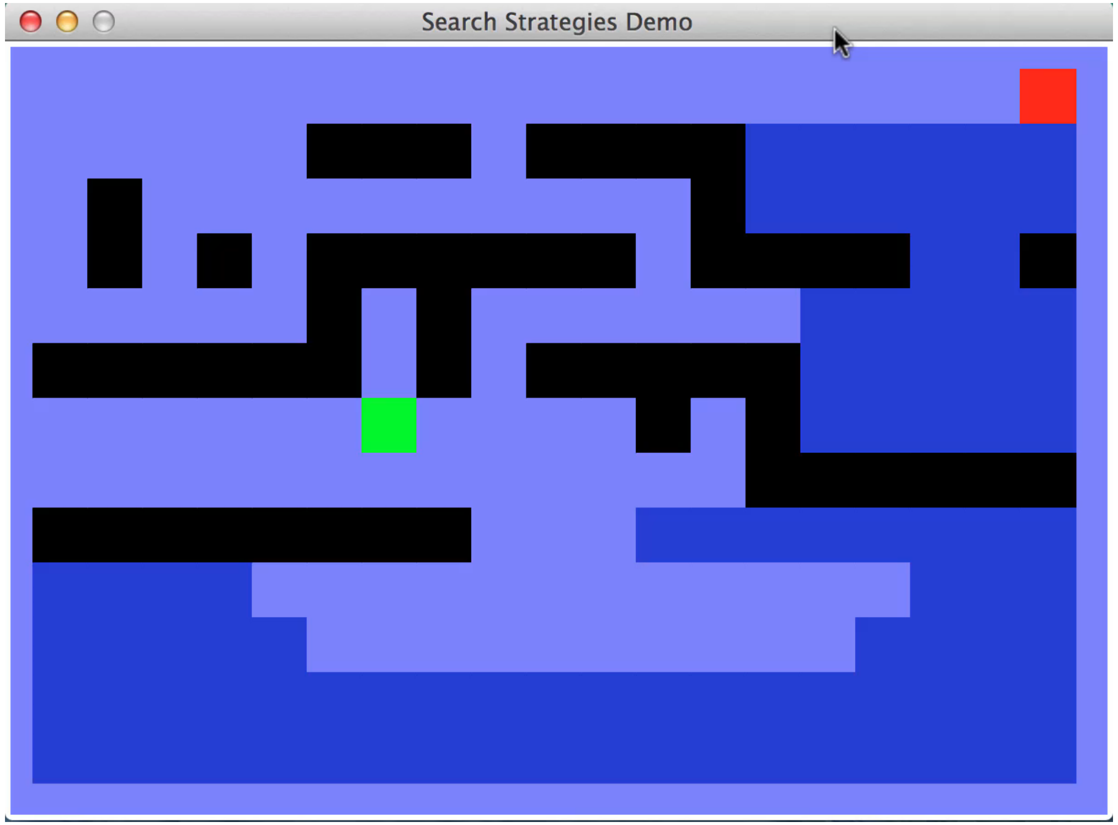
- Should we stop when we enqueue a goal?



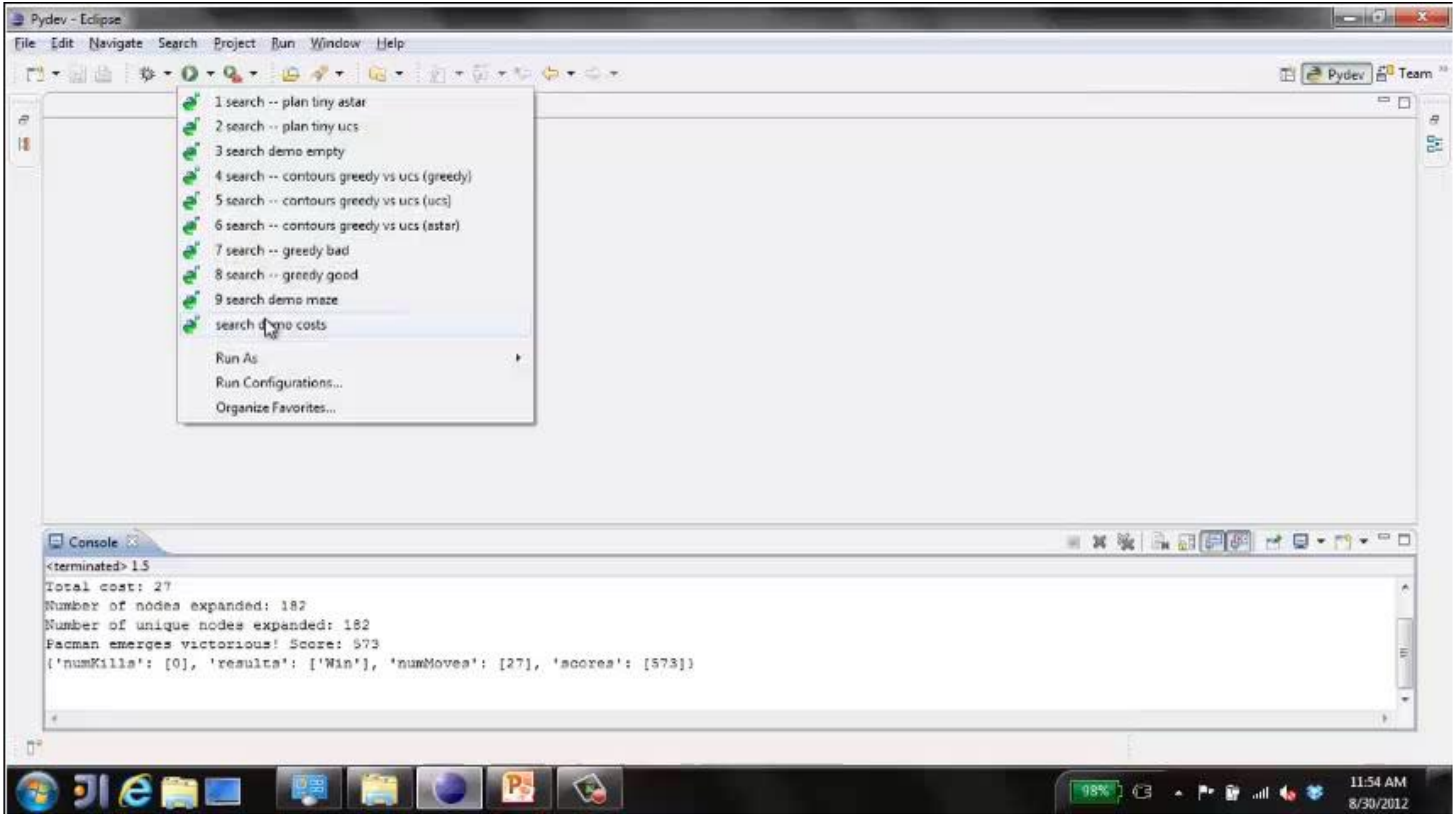
- No: only stop when we dequeue a goal

A* Examples





Guess the Search Algorithm

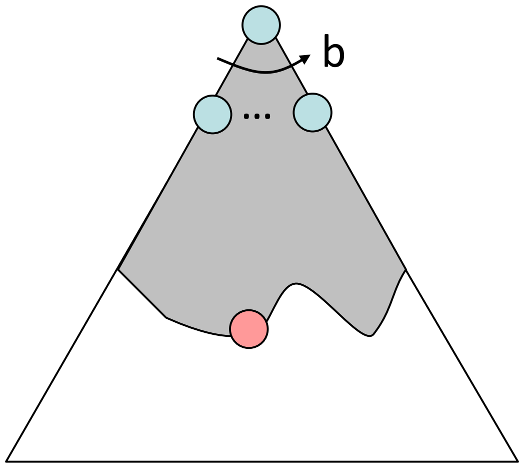


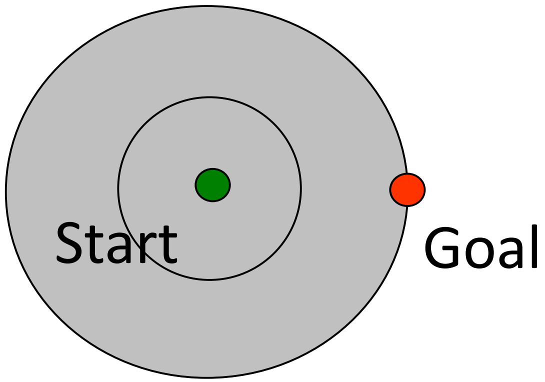
A* Applications

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

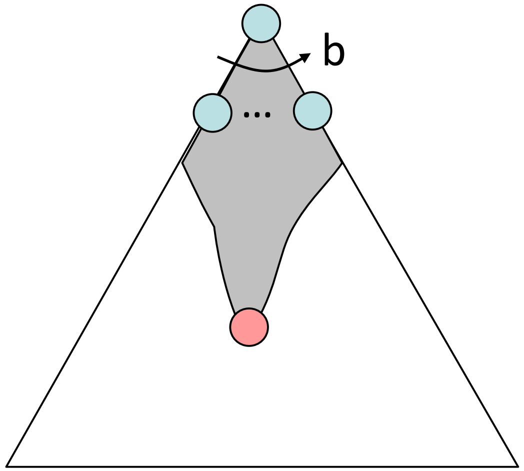
UCS vs A* Contours

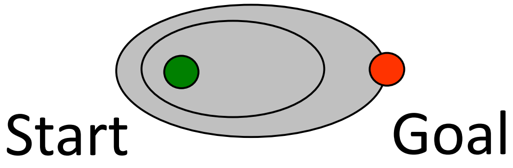
- Uniform-cost expands equally in all “directions”



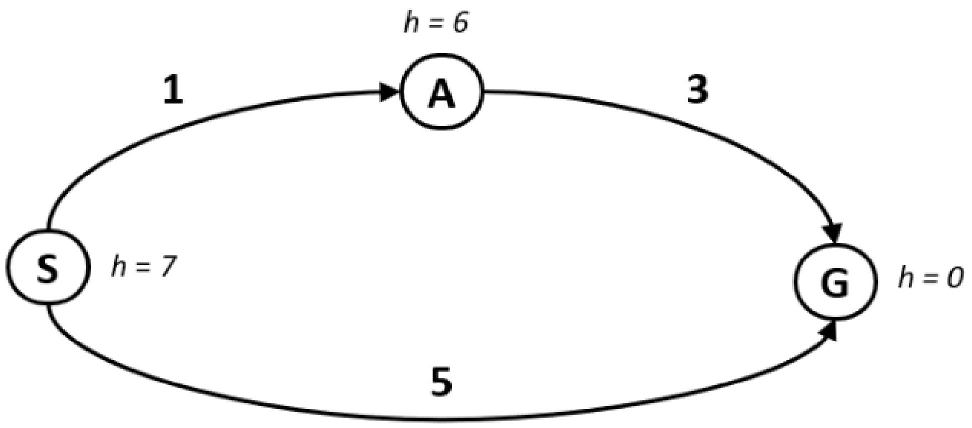


- A* expands mainly toward the goal, but does hedge its bets to ensure optimality





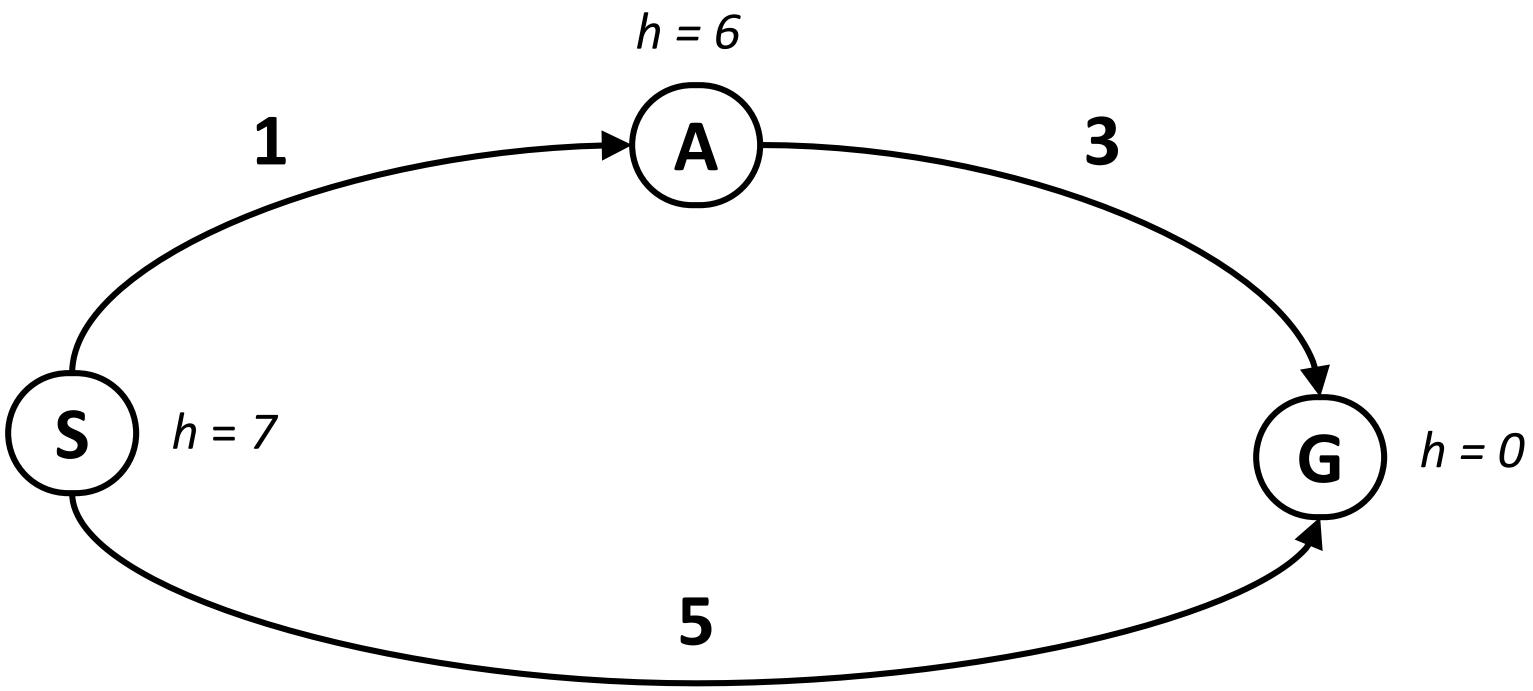
Which solution does A* return?



S ->
A ->
G

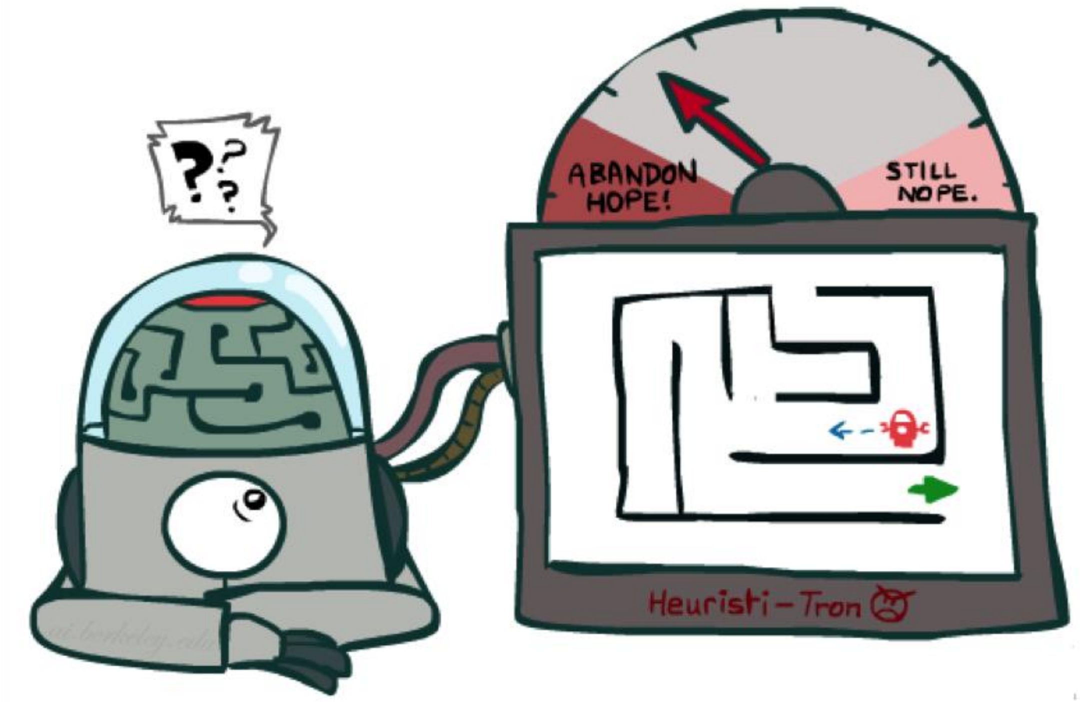
S ->
G

Is A* Optimal?

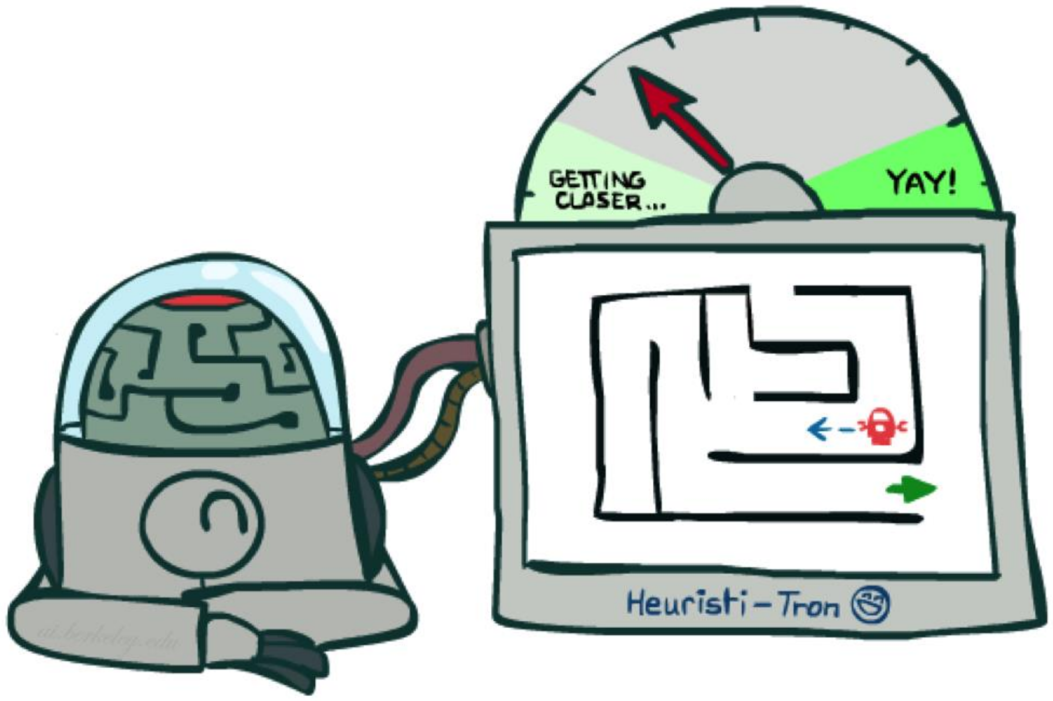


- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

Admissible Heuristics



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



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

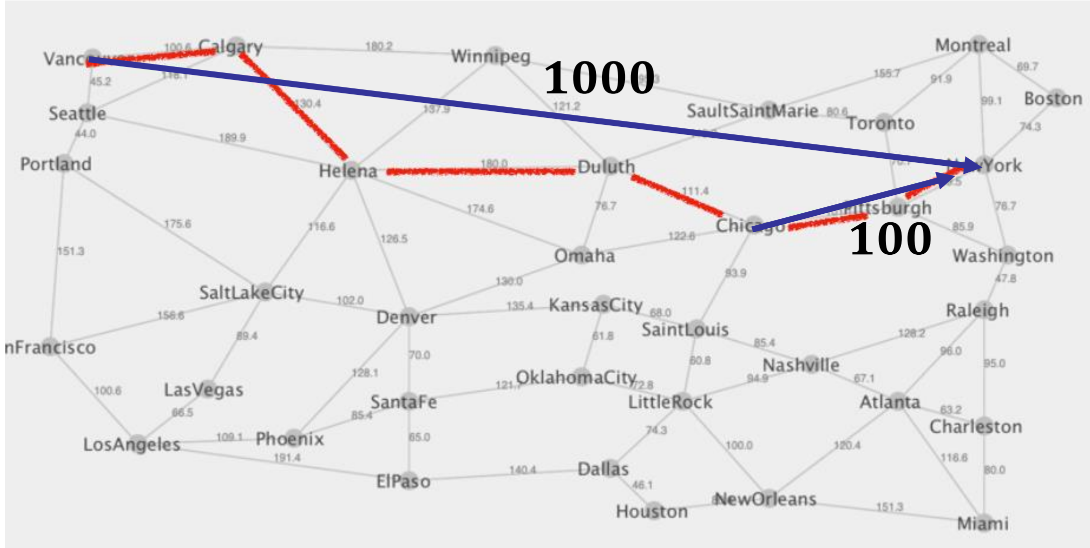
Admissible Heuristics

- 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

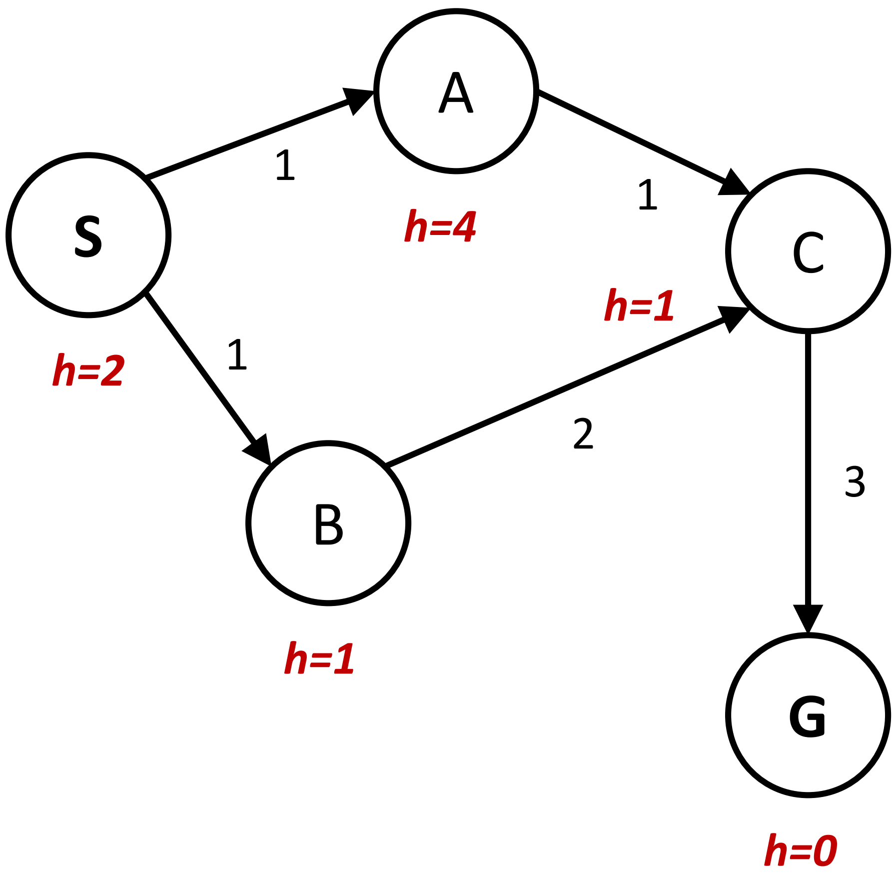
- Examples:



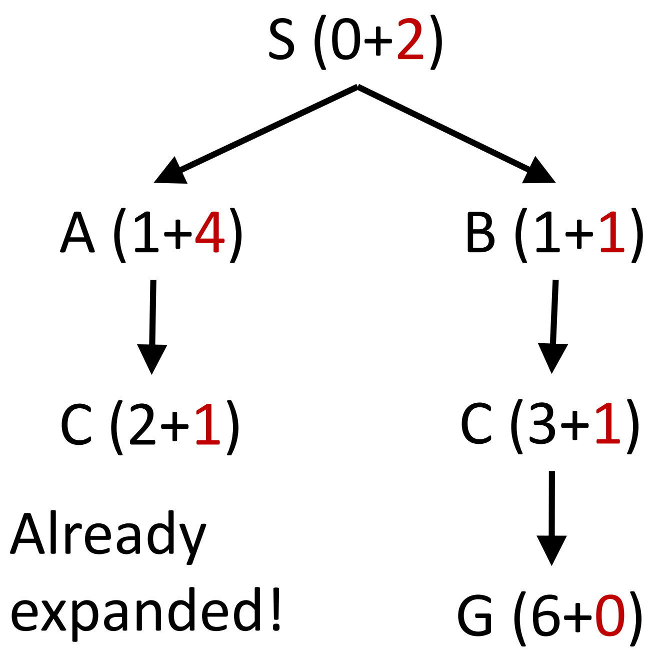
4

What About the Closed Set?

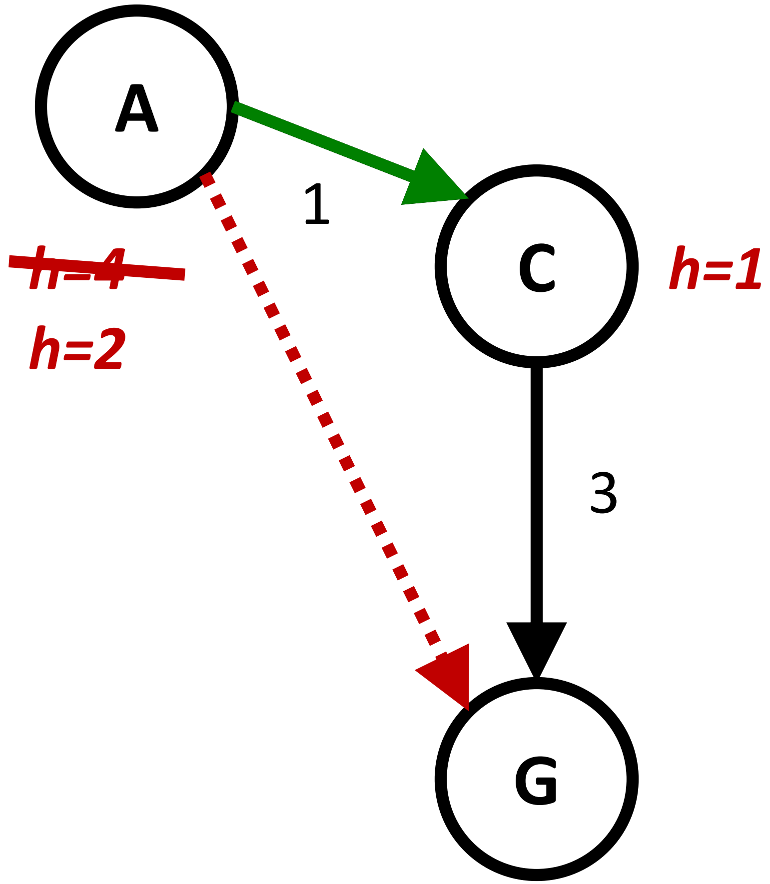
State space graph



Search tree



Consistent Heuristics



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

    h(A) ≤ actual cost from A to G

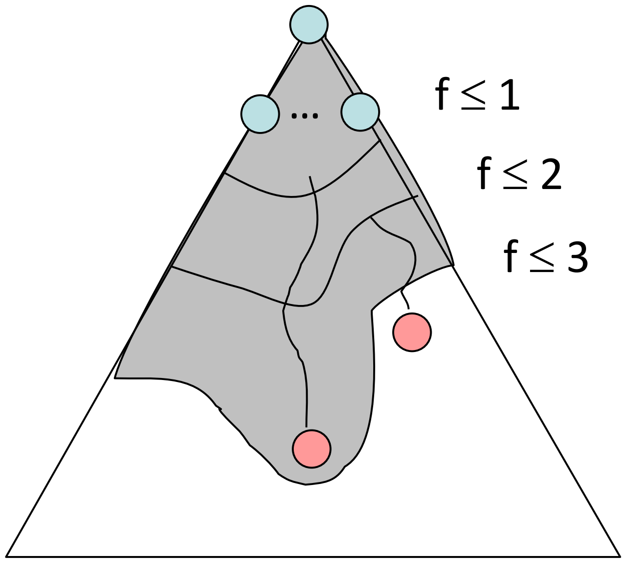
  - Consistency: “heuristic arc” cost ≤ actual cost for each arc

    h(A) – h(C) ≤ cost(A to C)

- What this means:
  - The f value along a path never decreases
  - True costs g increase moving toward the goal
  - Estimated costs h decrease no faster than g increasing

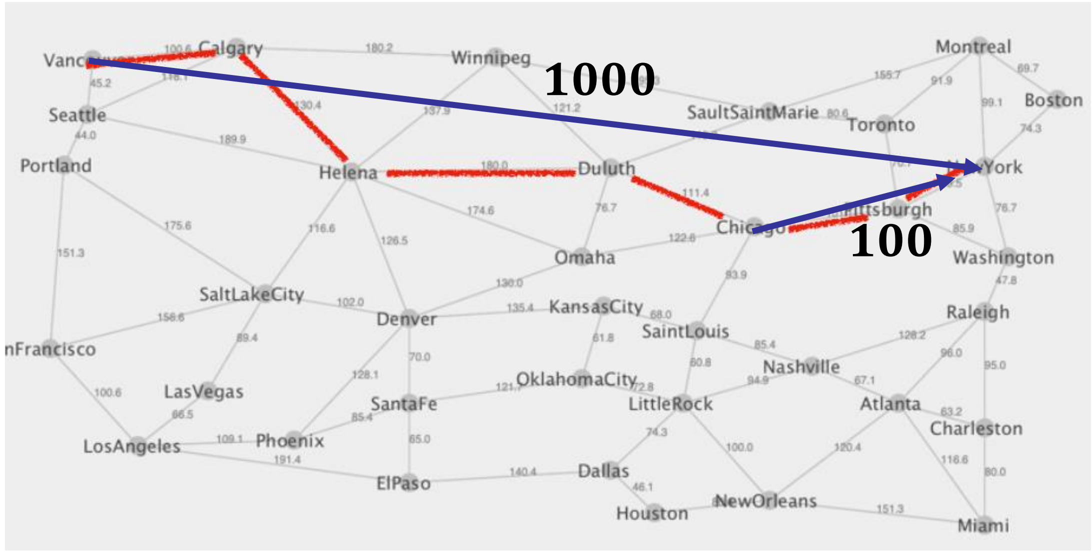
Optimality of A* Search

- Sketch: consider what A* does with a consistent heuristic:
  - Fact 1: A* expands nodes in increasing total f value (f-contours)
  - Fact 2: Consistency ensures that the first time A* reaches s, it is along an optimal path
  - Result: A* graph search is optimal!

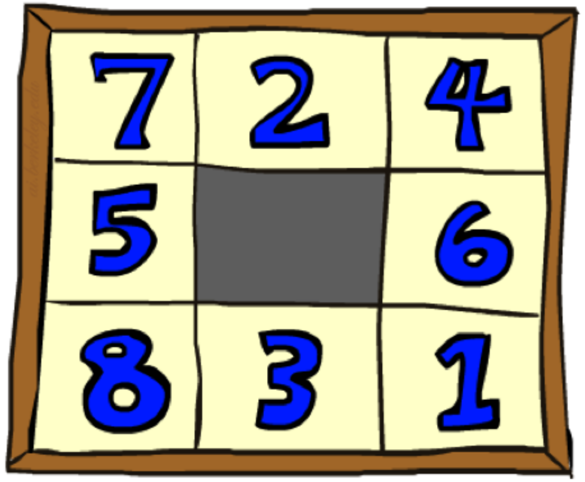


Finding Good Heuristics

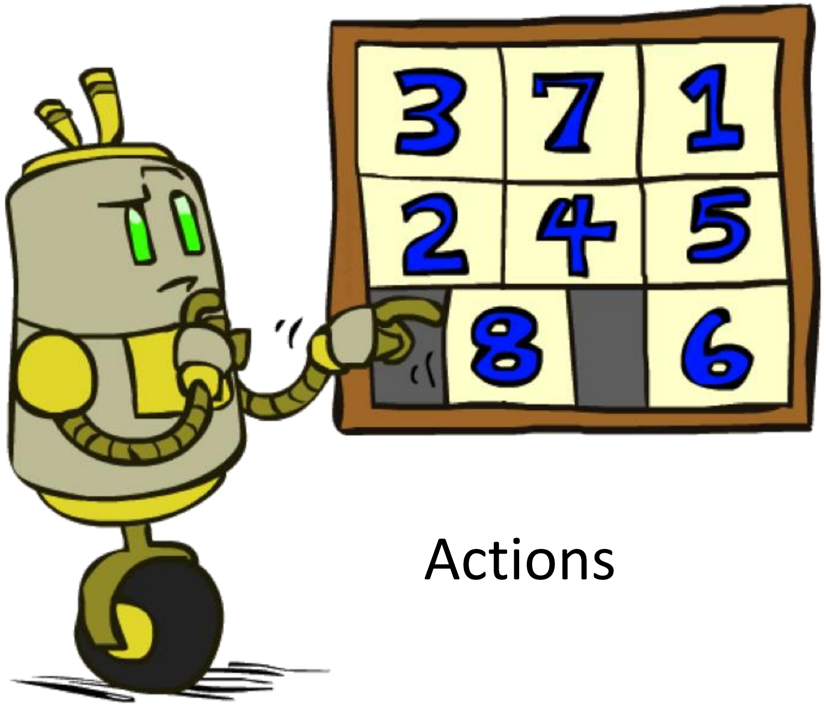
- Consistency is a stronger condition—consistency implies admissibility.
- In general, most natural admissible heuristics are also consistent.
- Often, admissible heuristics are solutions to *relaxed problems* with additional actions available.



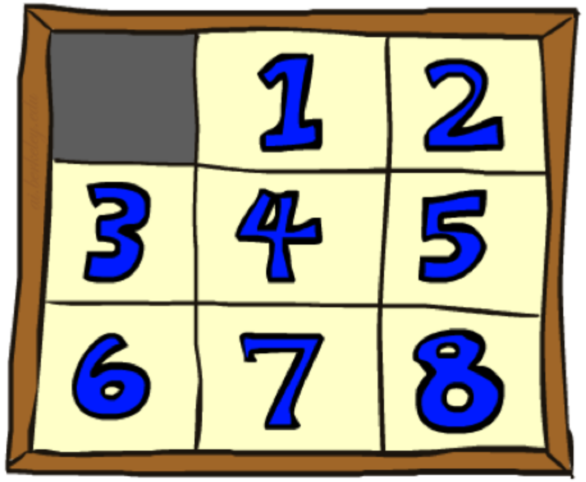
Example: 8 Puzzle



Start State



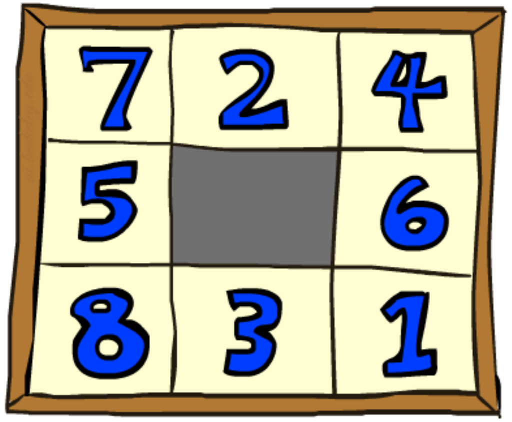
Actions



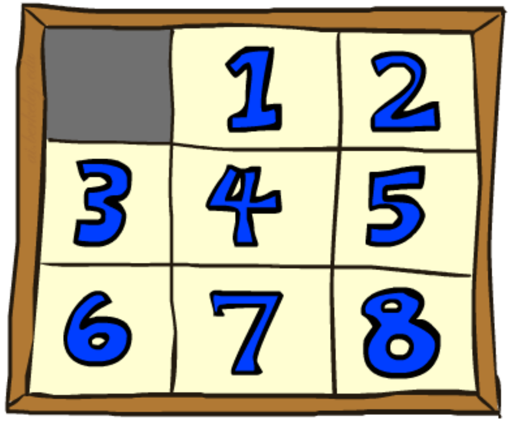
Goal State

Misplaced Tiles Heuristic

- Heuristic: Number of tiles misplaced
- Why is it admissible?
- This is a *relaxed-problem* heuristic



Start State



Goal State

h(

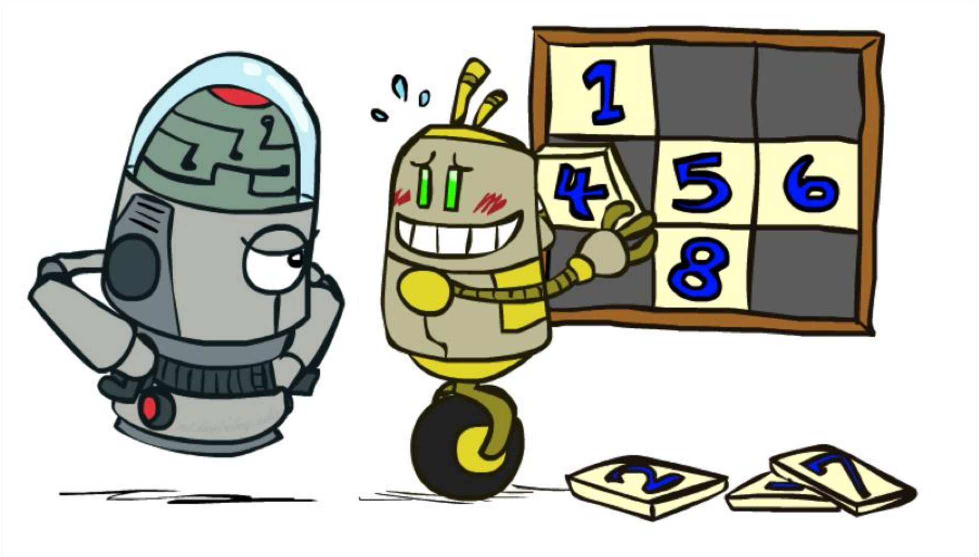
	1	2
3	4	5
6	7	8

) = 0

h(

1	4	2
	5	8
3	6	7

) = 7



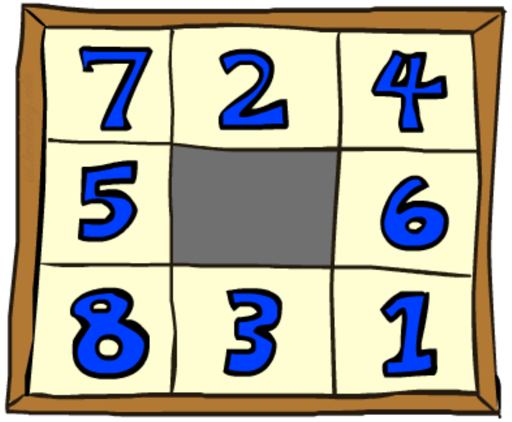
Manhattan Distance Heuristic

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?

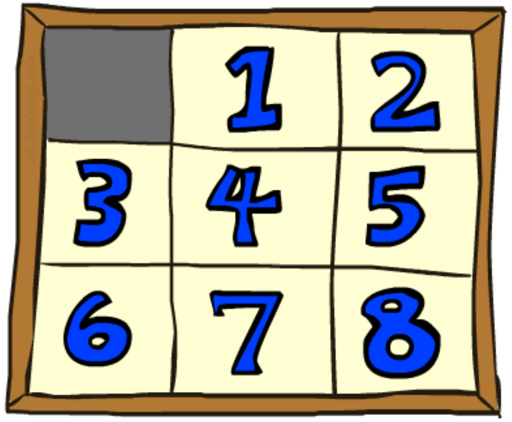
- Total *Manhattan* distance

- Why is it admissible?

- h(start) = 3 + 1 + 2 + … = 18



Start State



Goal State

Actual Cost?

- 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 better approximate the true cost, fewer nodes are expanded but more work is needed to compute the heuristic itself

Summary

- Heuristics can guide us toward the goal (à la greedy search)
- A* uses both backward costs and estimates of forward costs
- A* is optimal with admissible / consistent heuristics
- Heuristic design often relies on relaxed problems