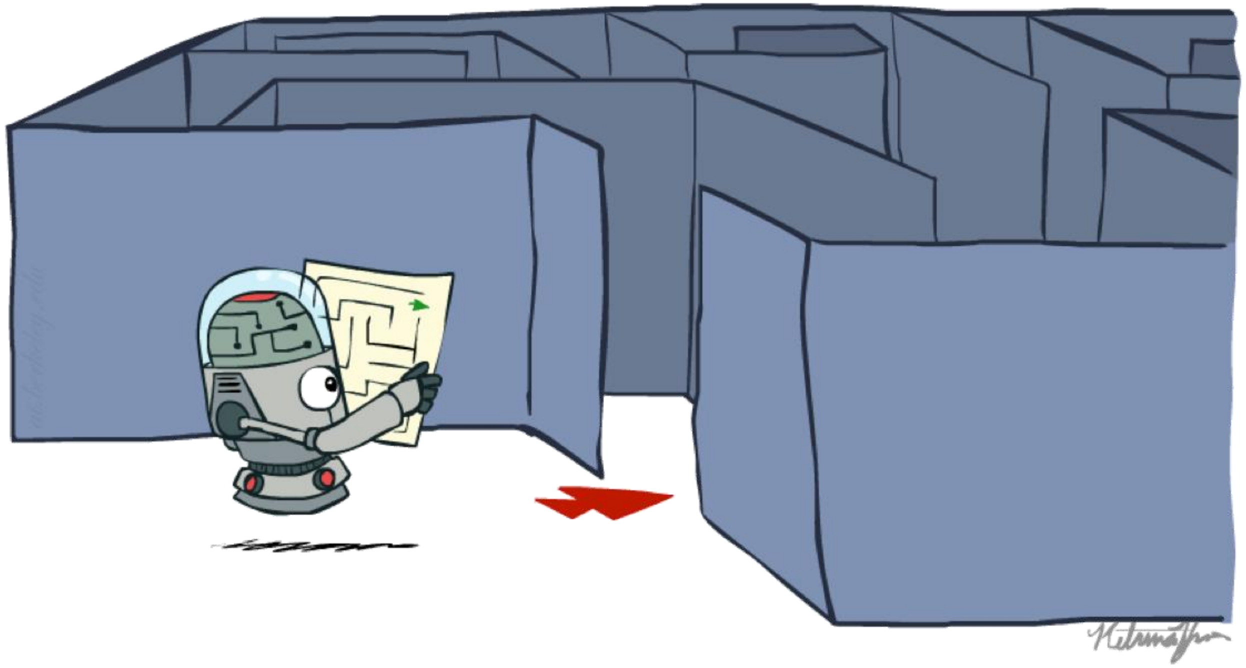
Announcement

- Programming Assignment 1A: search
  - Instructions at Blackboard -> “Programming Assignments”
  - Submission at AutoLab
  - Due: Oct. 6, 11:59pm
- Grace Days
  - 5 days for the whole semester
- AutoLab registration
  - See BB announcement/email

Search

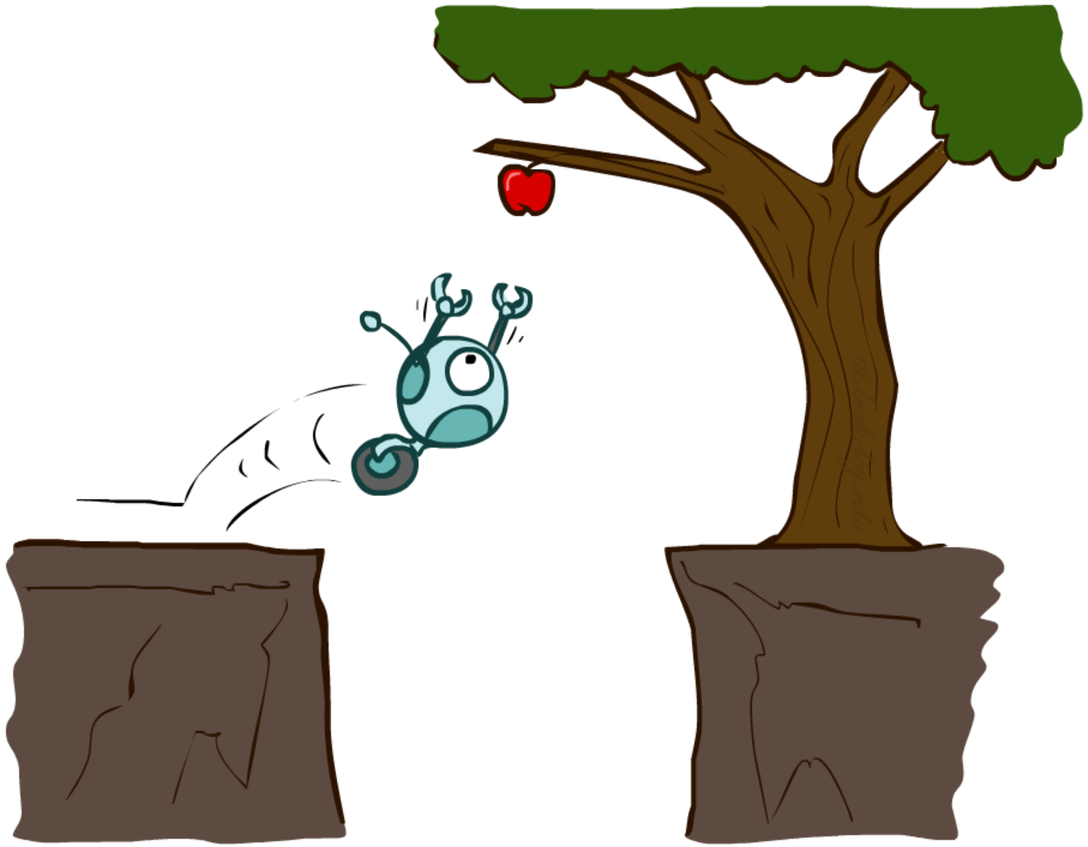


AIMA Chapter 3

[Adapted from slides by Dan Klein and Pieter Abbeel at UC Berkeley (ai.berkeley.edu).]

Reflex Agents

- Reflex agents:
  - Choose action based on current percept (and maybe memory)
    - Require a mapping from percepts to actions
  - Do not consider the future consequences of their actions
  - Consider how the world IS

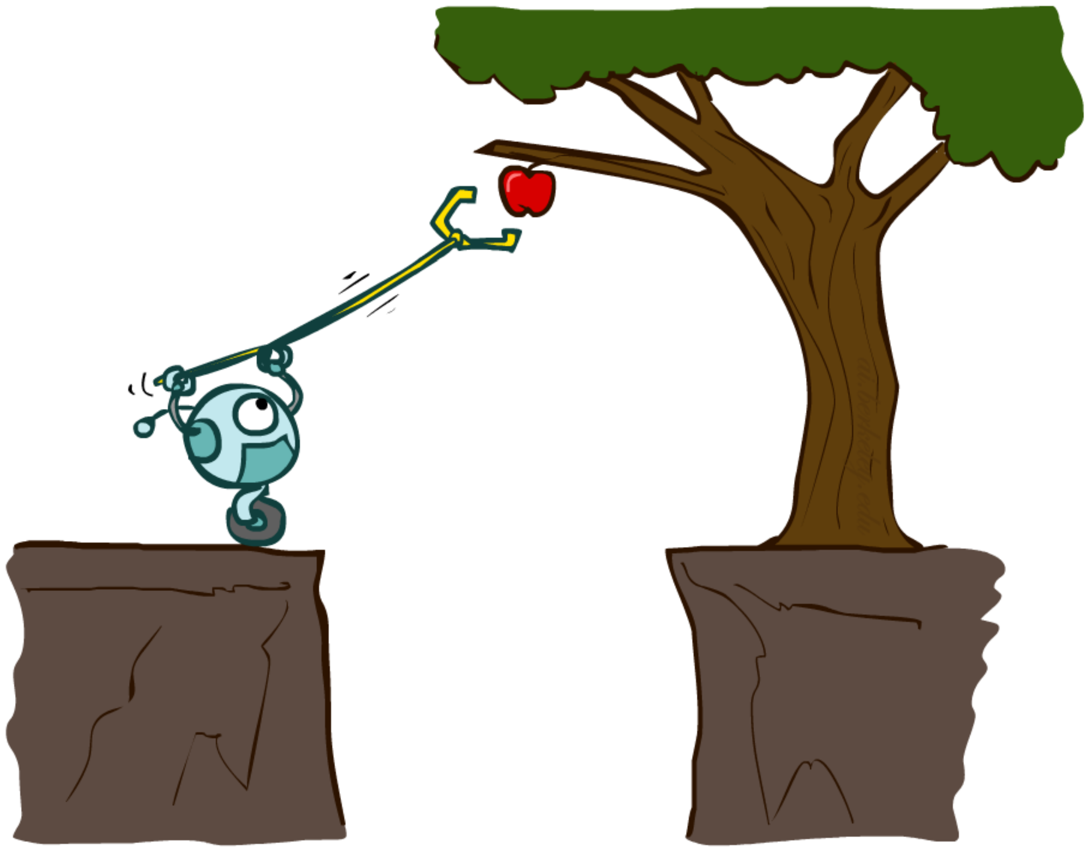


Reflex Agents

Roomba from iRobot

Planning Agents

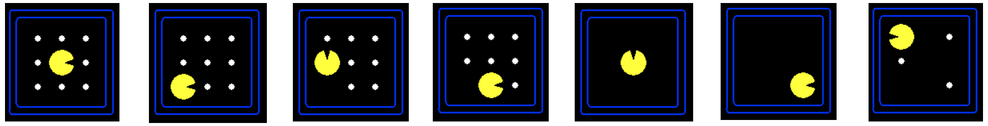
- Planning agents:
  - Ask “what if”
  - Decisions based on (hypothesized) consequences of actions
    - Must have a model of how the world evolves in response to actions
    - Must formulate a goal
  - Consider how the world WOULD BE



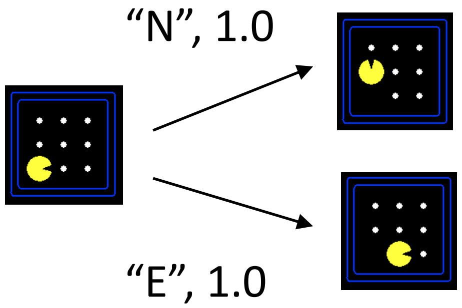
Search Problems

Search Problems

- A search problem consists of:
  - A state space



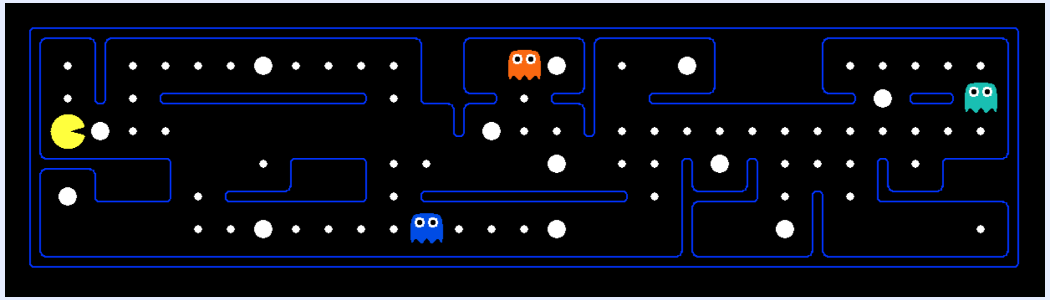
  - A successor function (with actions, costs)



  - A start state and a goal test
- A solution is a sequence of actions (a plan) which transforms the start state to a goal state

What’s in a State Space?

The world state includes every last detail of the environment



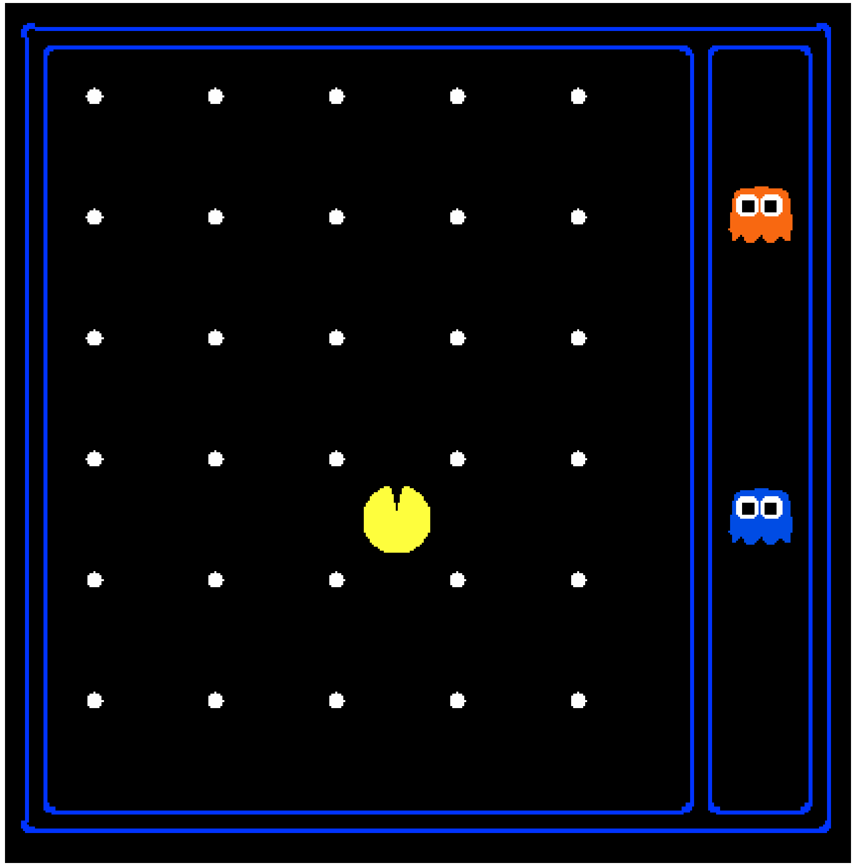
A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END

- Problem: Eat-All-Dots
  - States: {(x,y), dot booleans}
  - Actions: NSEW
  - Successor: update location and possibly a dot boolean
  - Goal test: dots all false

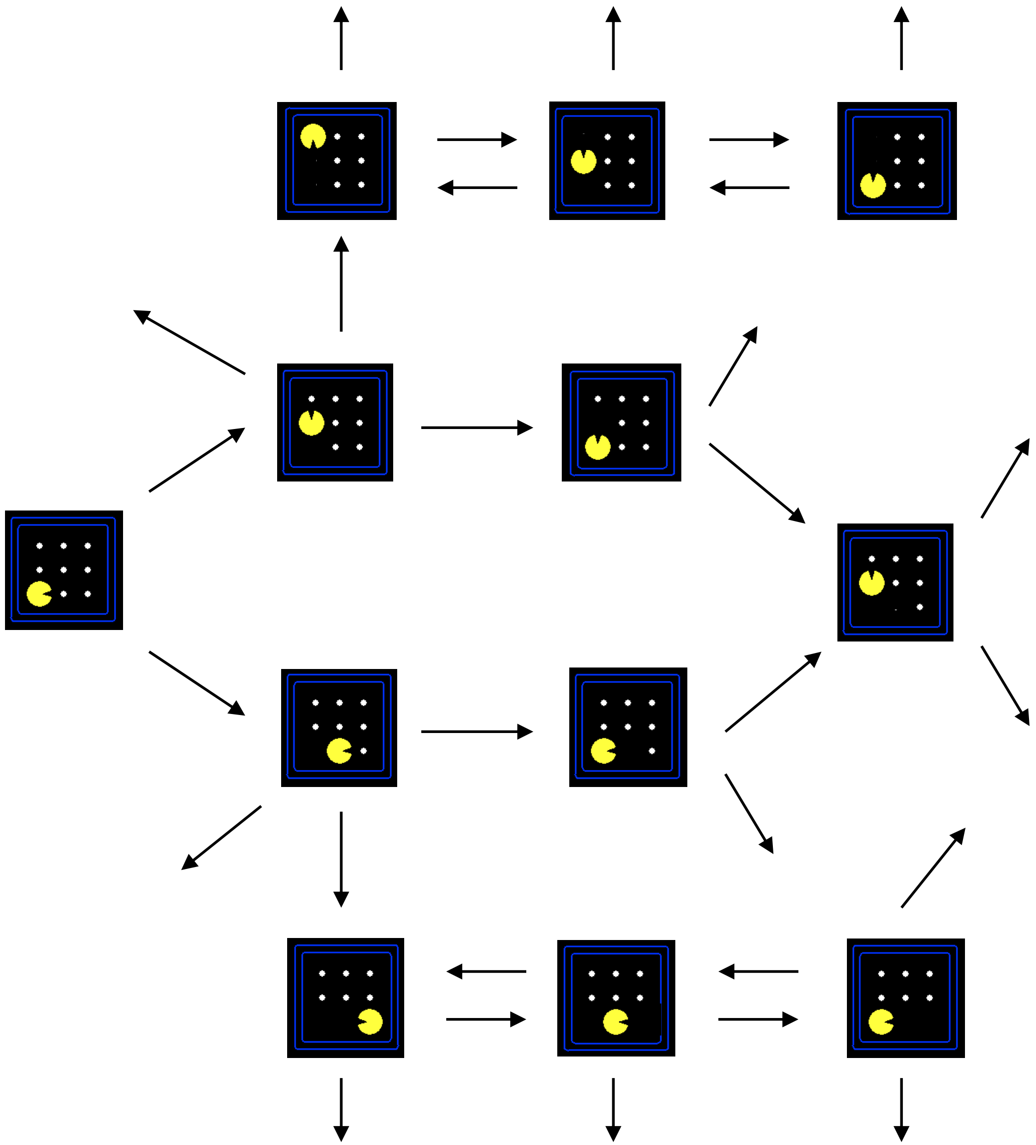
State Space Sizes?

- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW
- How many
  - World states?
    $120 \times (2^{30}) \times (12^2) \times 4$
  - States for pathing?
    120
  - States for eat-all-dots?
    $120 \times (2^{30})$

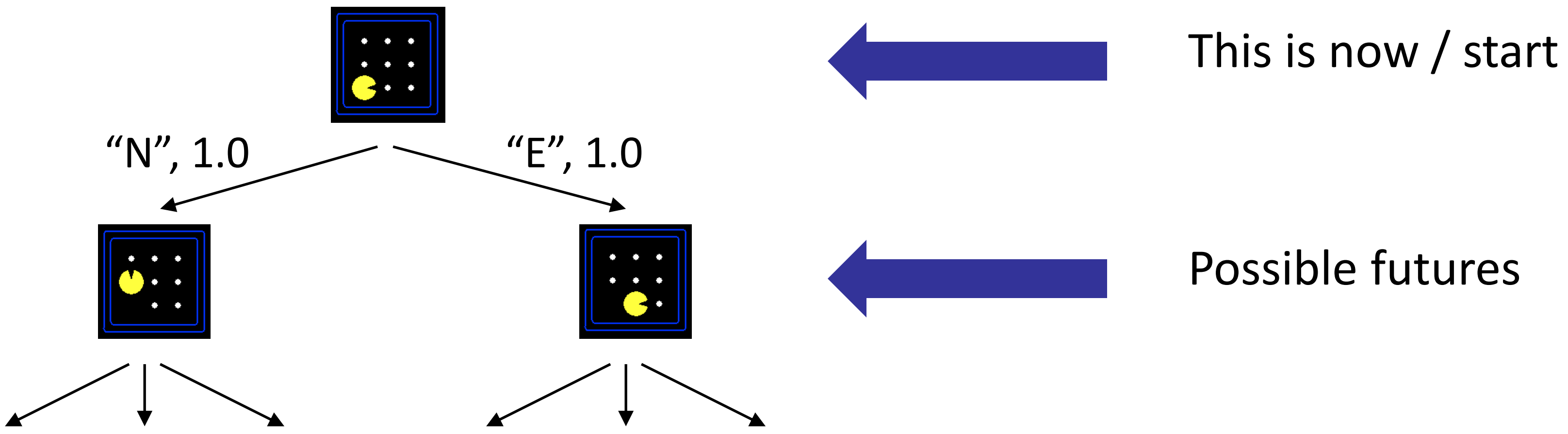


State Space Graphs

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!
- We can rarely build this full graph in memory (it’s too big), but it’s a useful idea



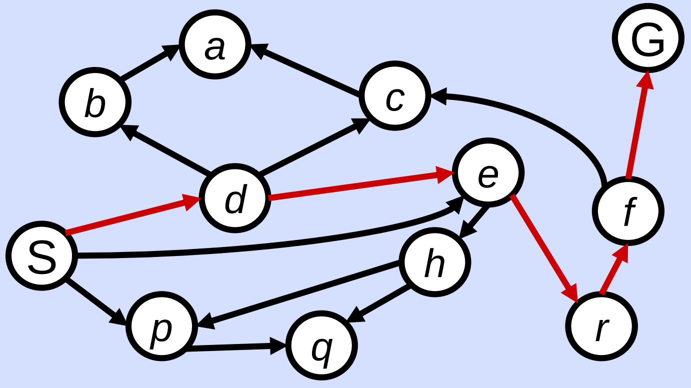
Search Trees



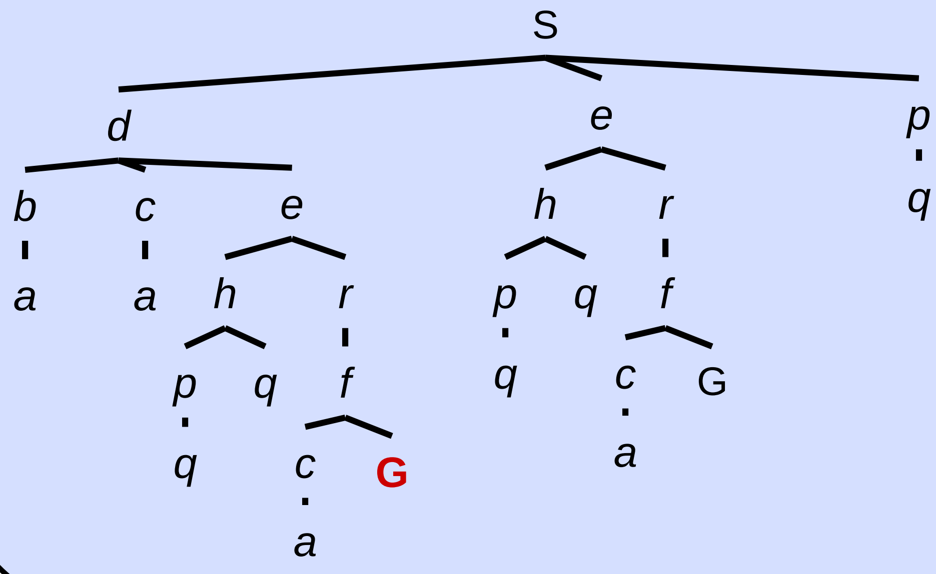
- A search tree:
  - A “what if” tree of plans and their outcomes
  - The start state is the root node
  - Children correspond to successors
  - For most problems, we can never actually build the whole tree

State Space Graphs vs. Search Trees

State Space Graph

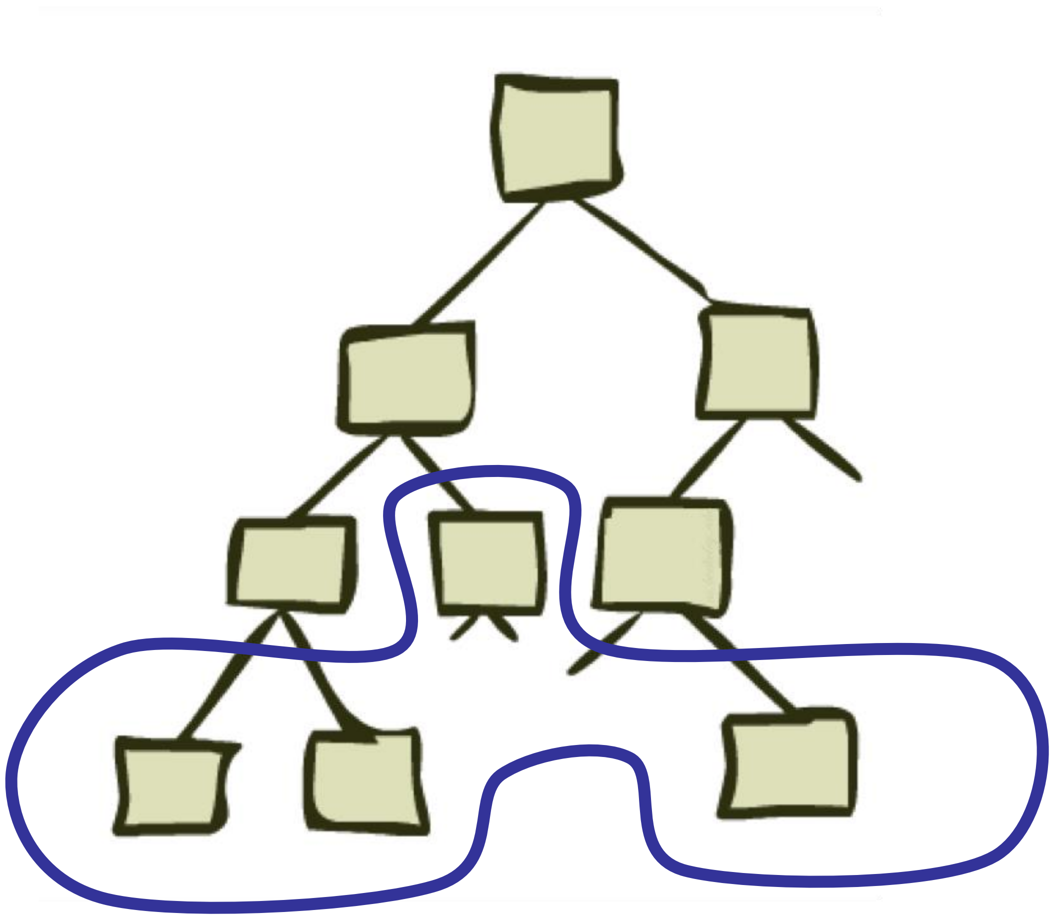


Search Tree



Each NODE in in the search tree is an entire PATH in the state space graph, corresponding to a PLAN that achieves the state.

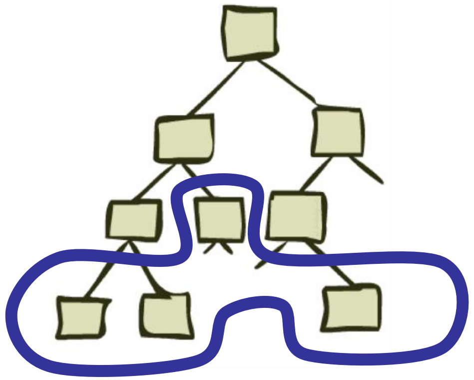
Tree Search



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
```

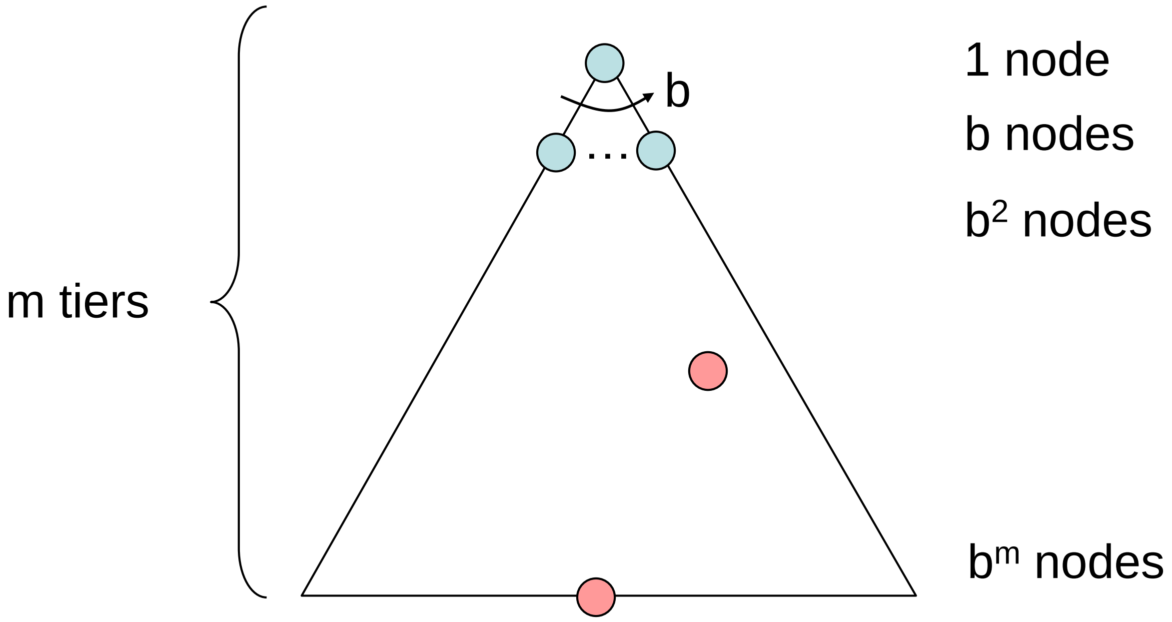
- Important concepts:
  - Fringe (frontier)
  - Expansion
  - Exploration strategy ← determines the search algorithm



Search Algorithm Properties

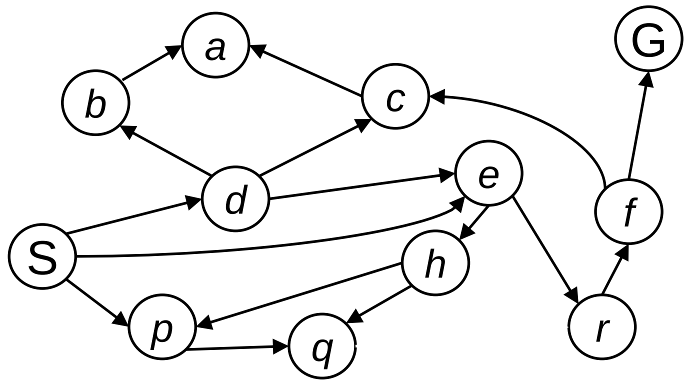
- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths

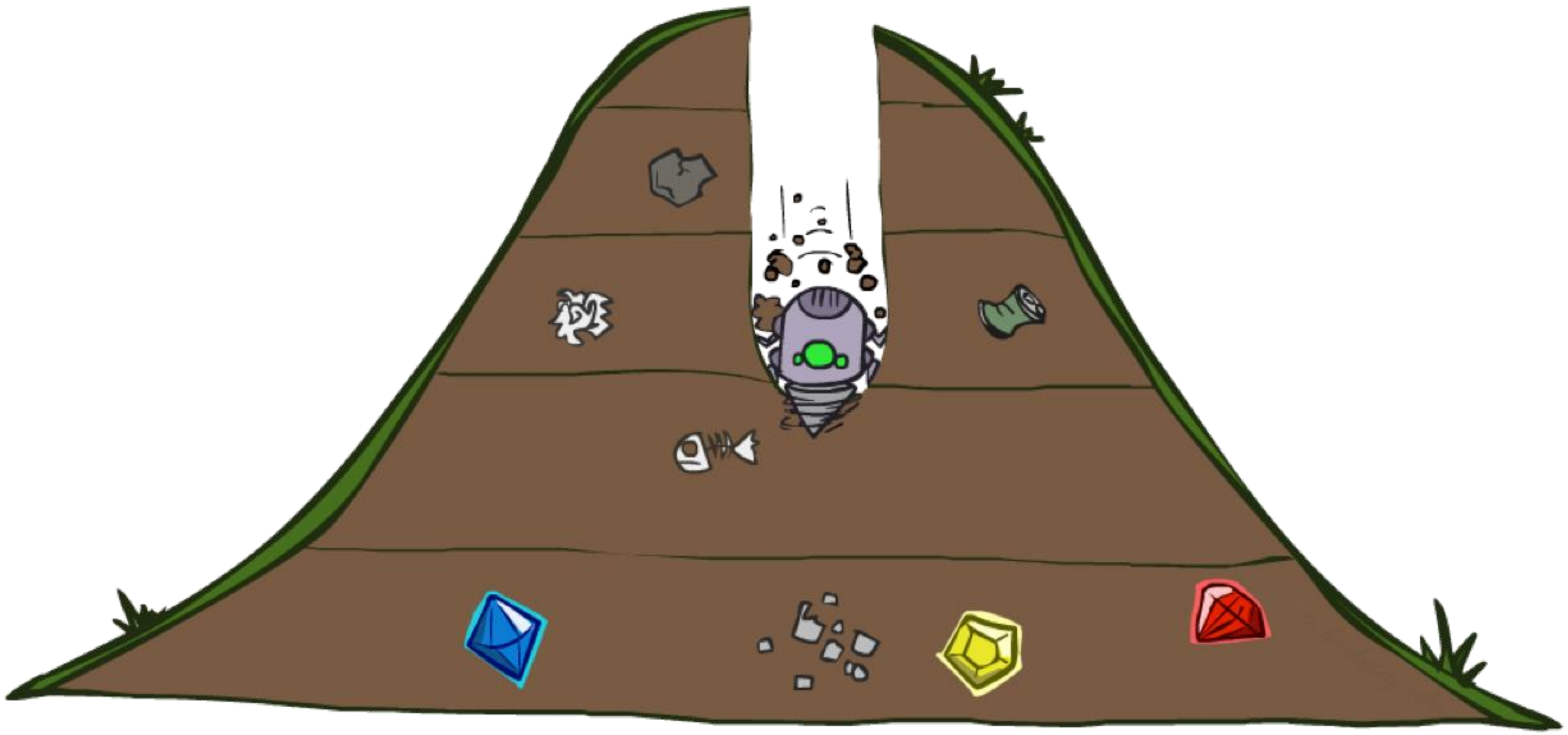


- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots.\ b^m = O(b^m)$

Example: Tree Search



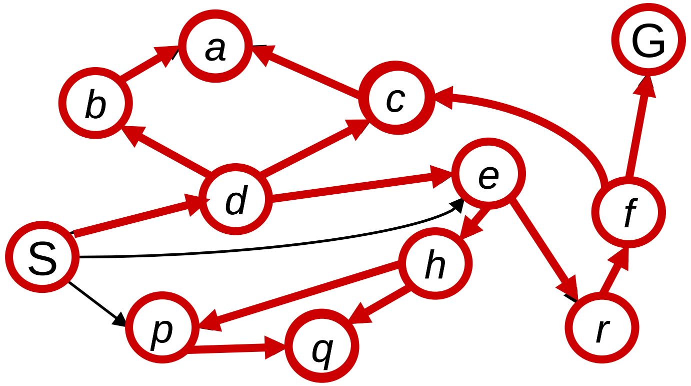
Depth-First Search

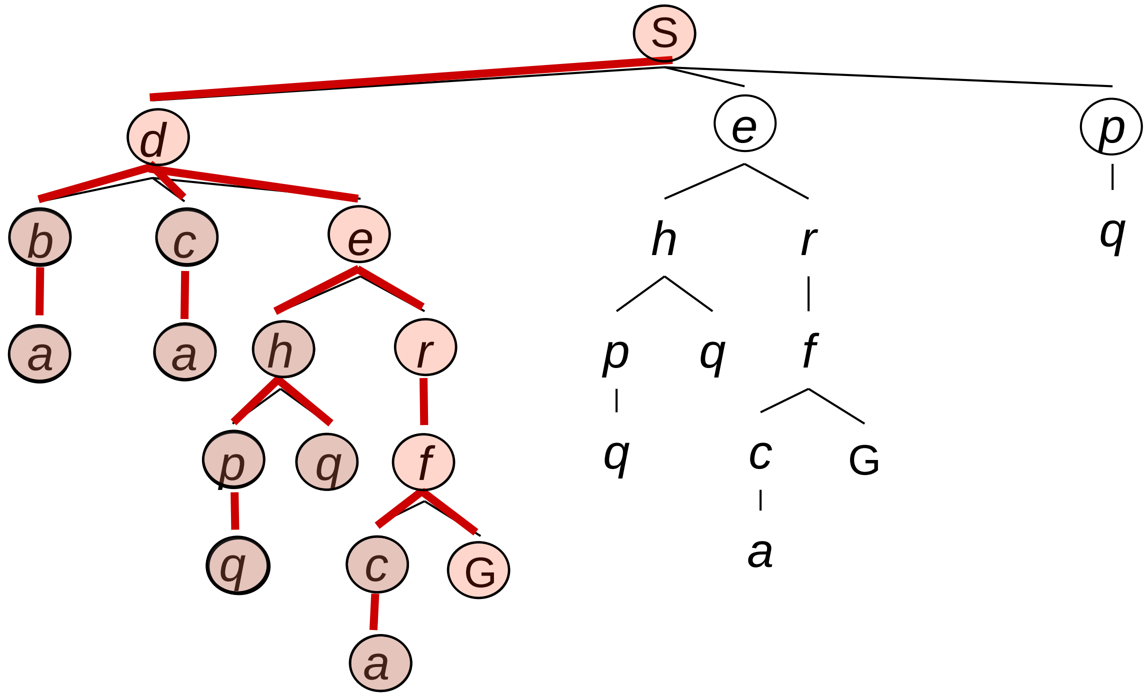


Depth-First Search

Strategy: expand a deepest node first

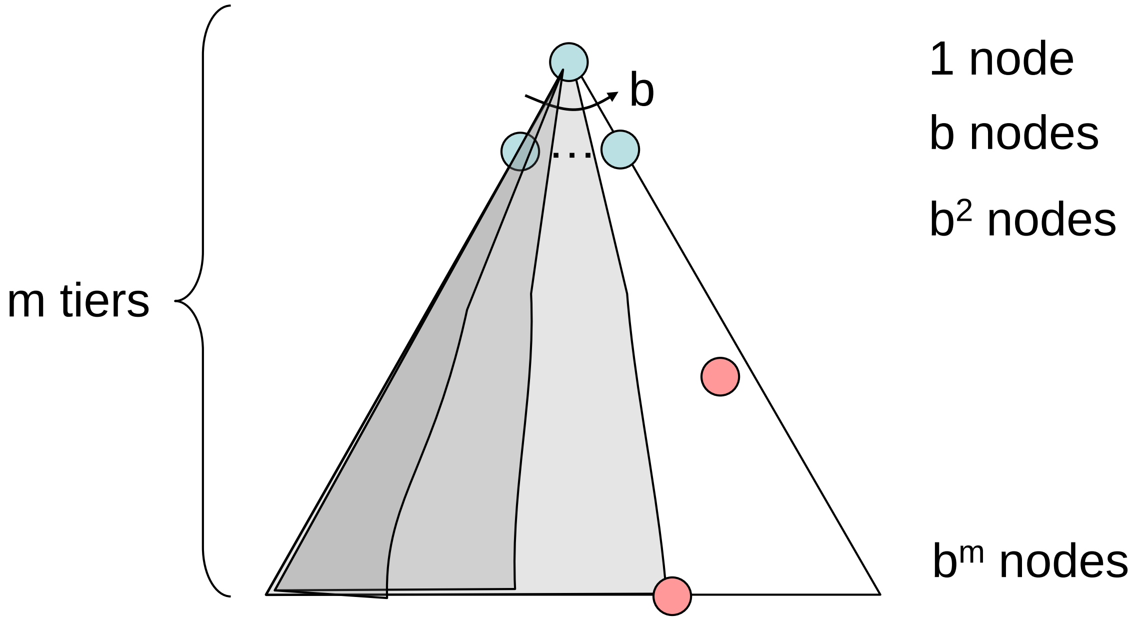
Implementation: Fringe is a LIFO stack



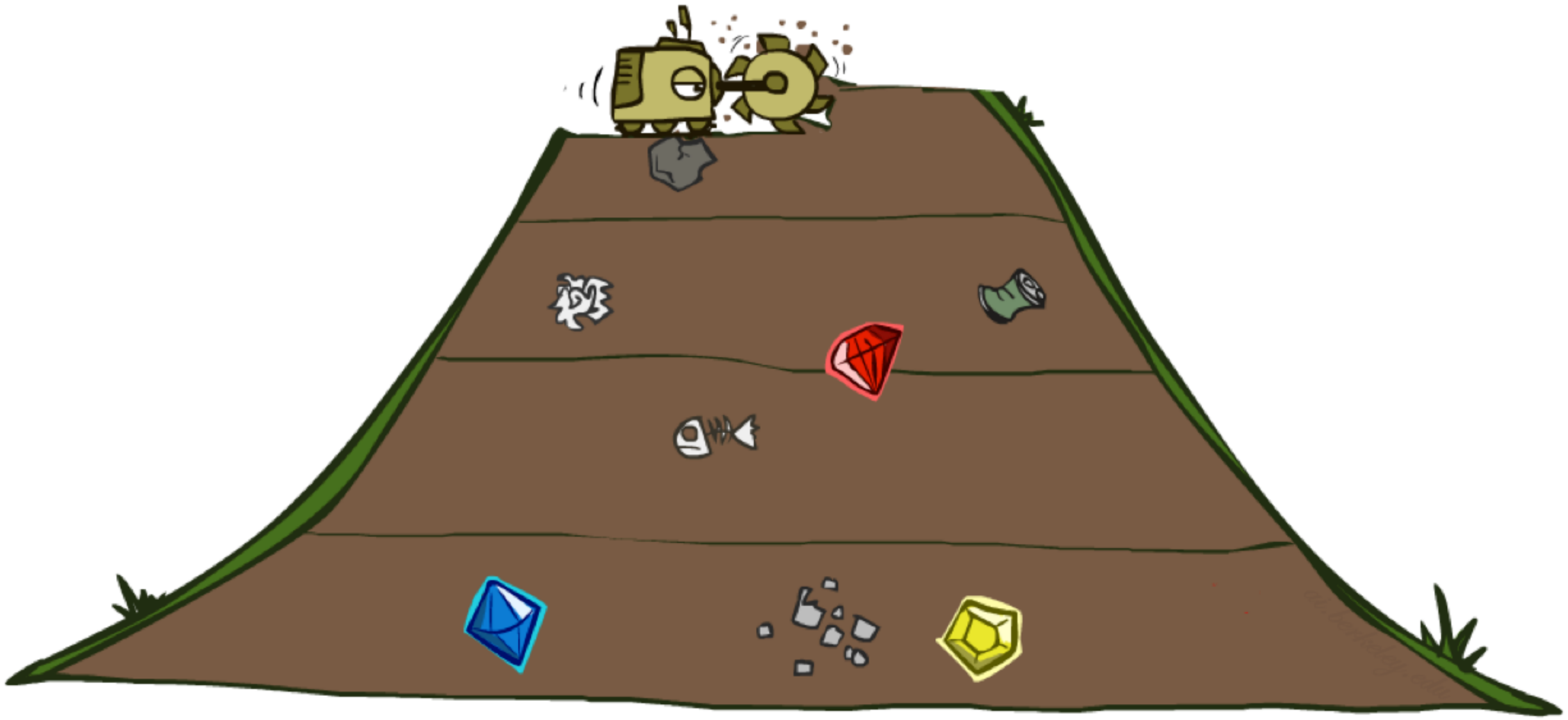


Depth-First Search (DFS) Properties

- What nodes DFS expand?
  - Left to right
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the fringe take?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?
  - m could be infinite, so only if we prevent cycles (more later)
- Is it optimal?
  - No, it finds the “leftmost” solution, regardless of depth or cost



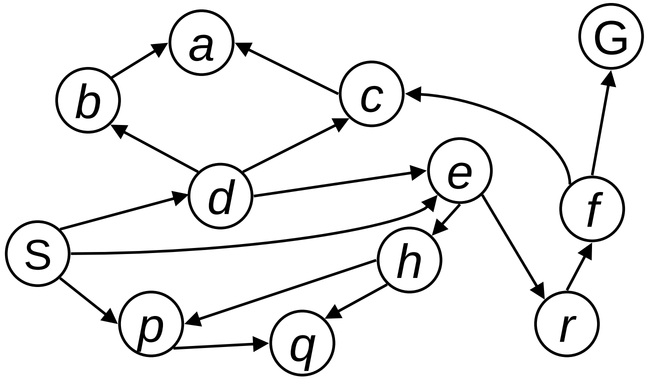
Breadth-First Search

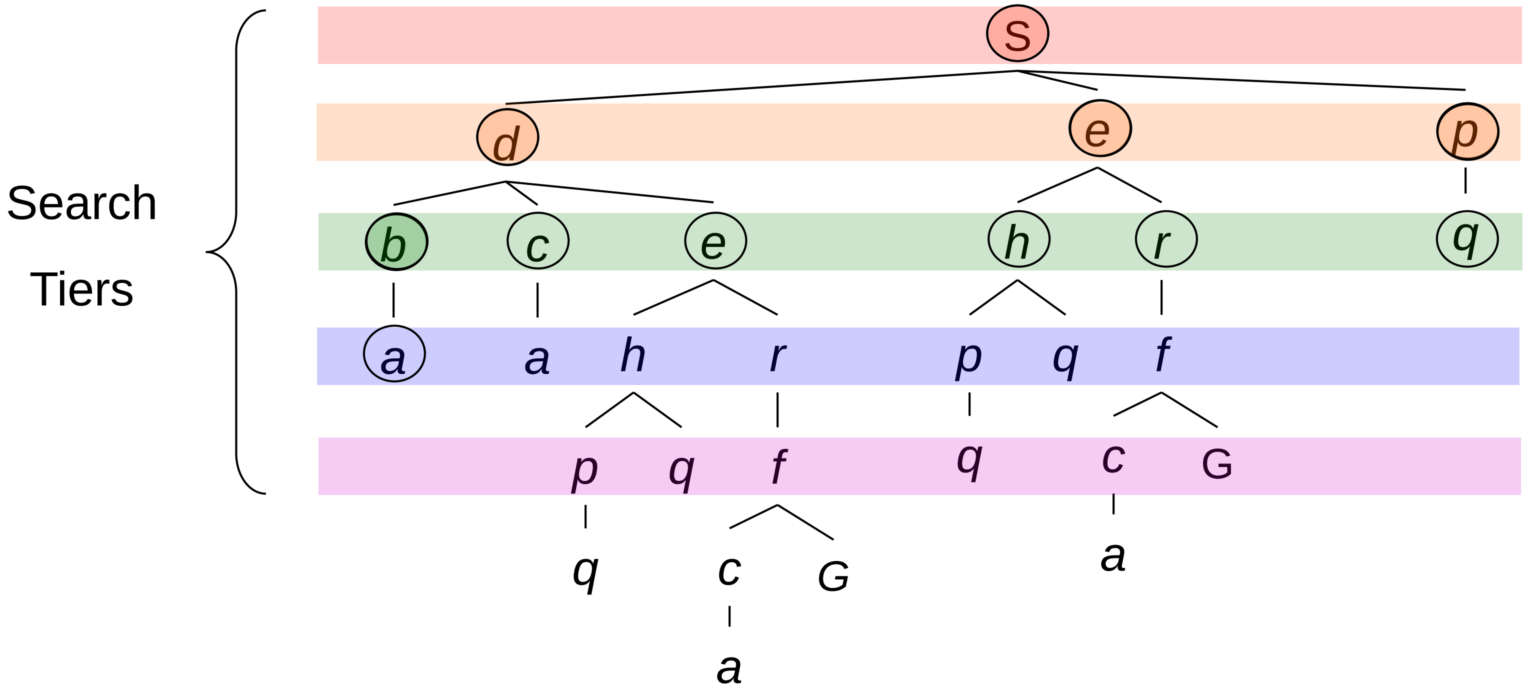


Breadth-First Search

Strategy: expand a shallowest node first

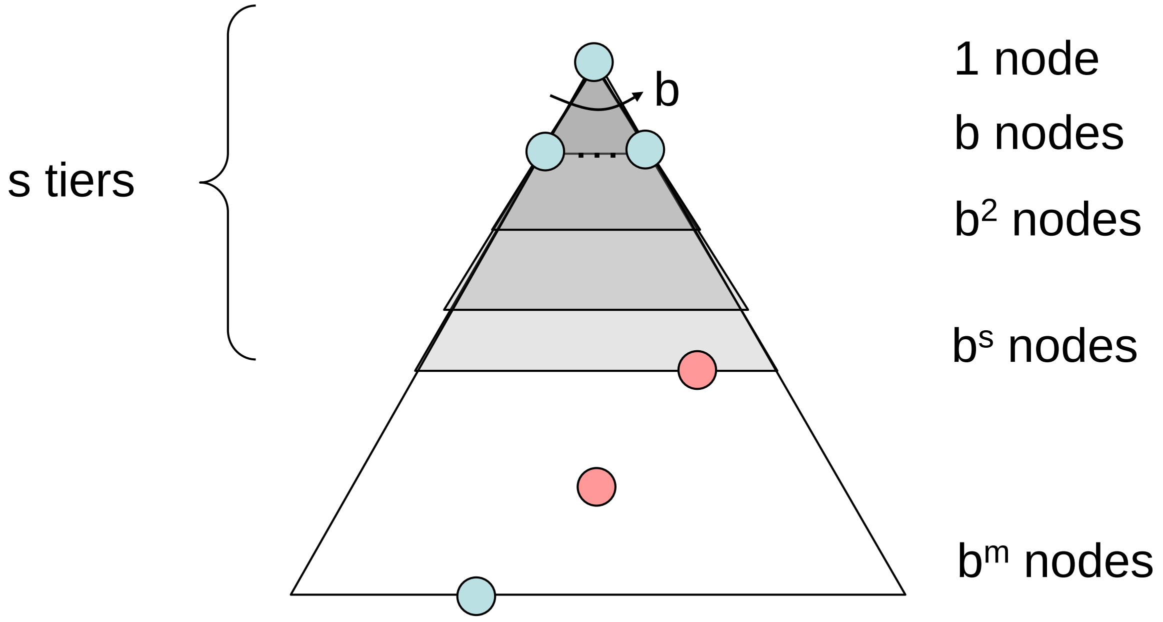
Implementation: Fringe is a FIFO queue





Breadth-First Search (BFS) Properties

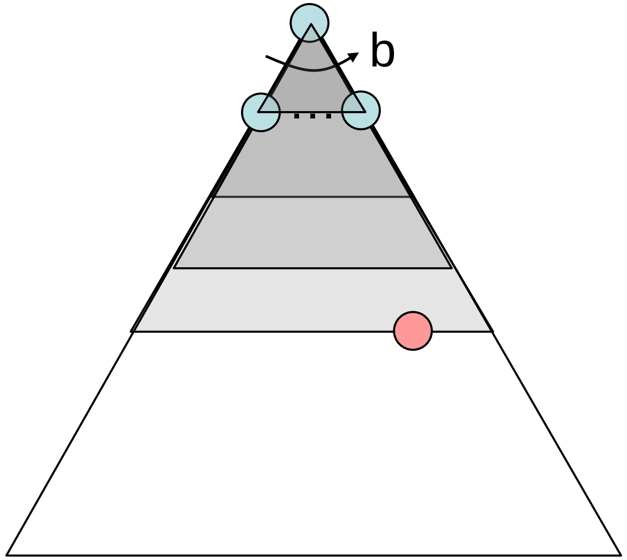
- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$
- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?
  - s must be finite if a solution exists, so yes!
- Is it optimal?
  - Only if costs are all 1 (more on costs later)



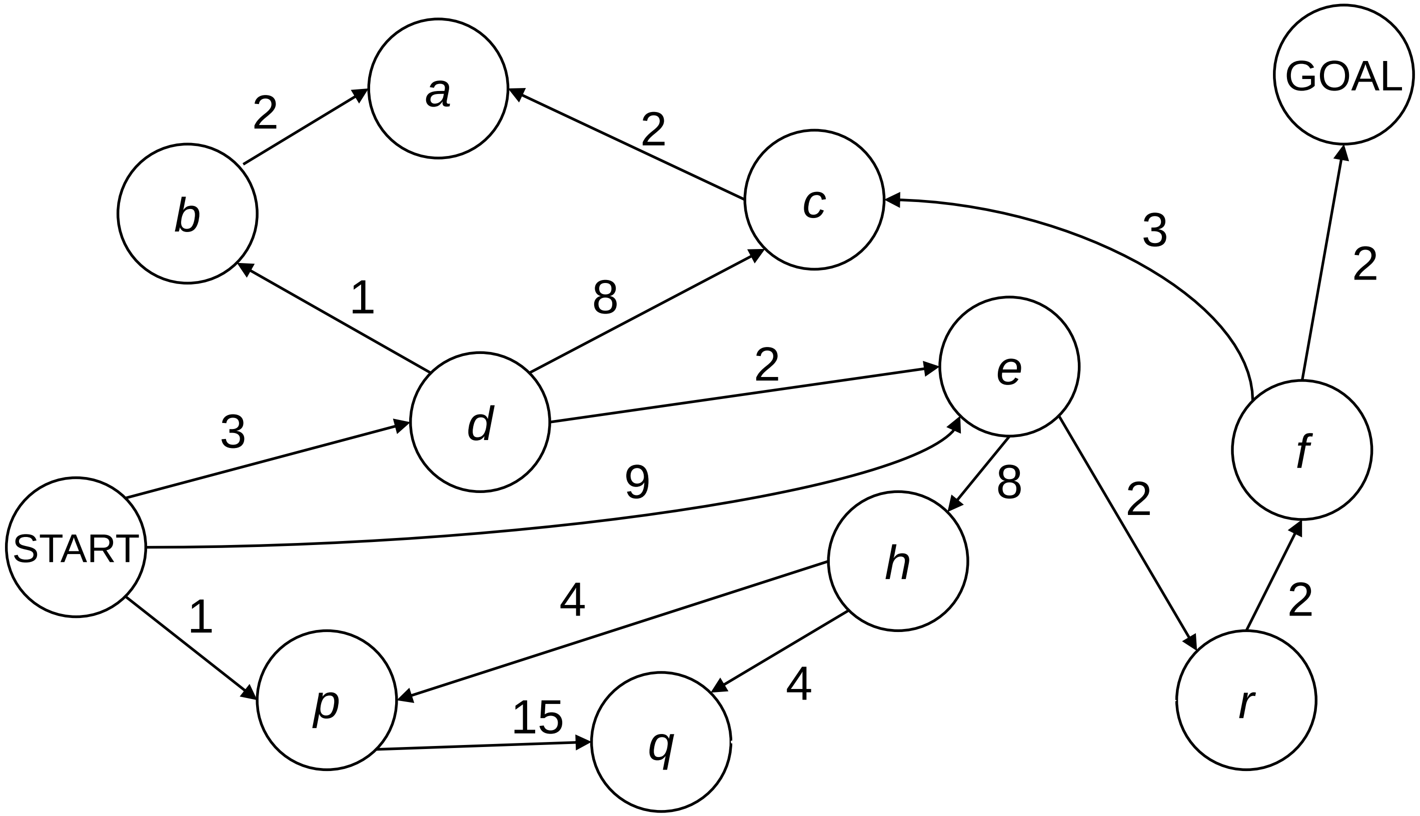
Iterative Deepening

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..

- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!

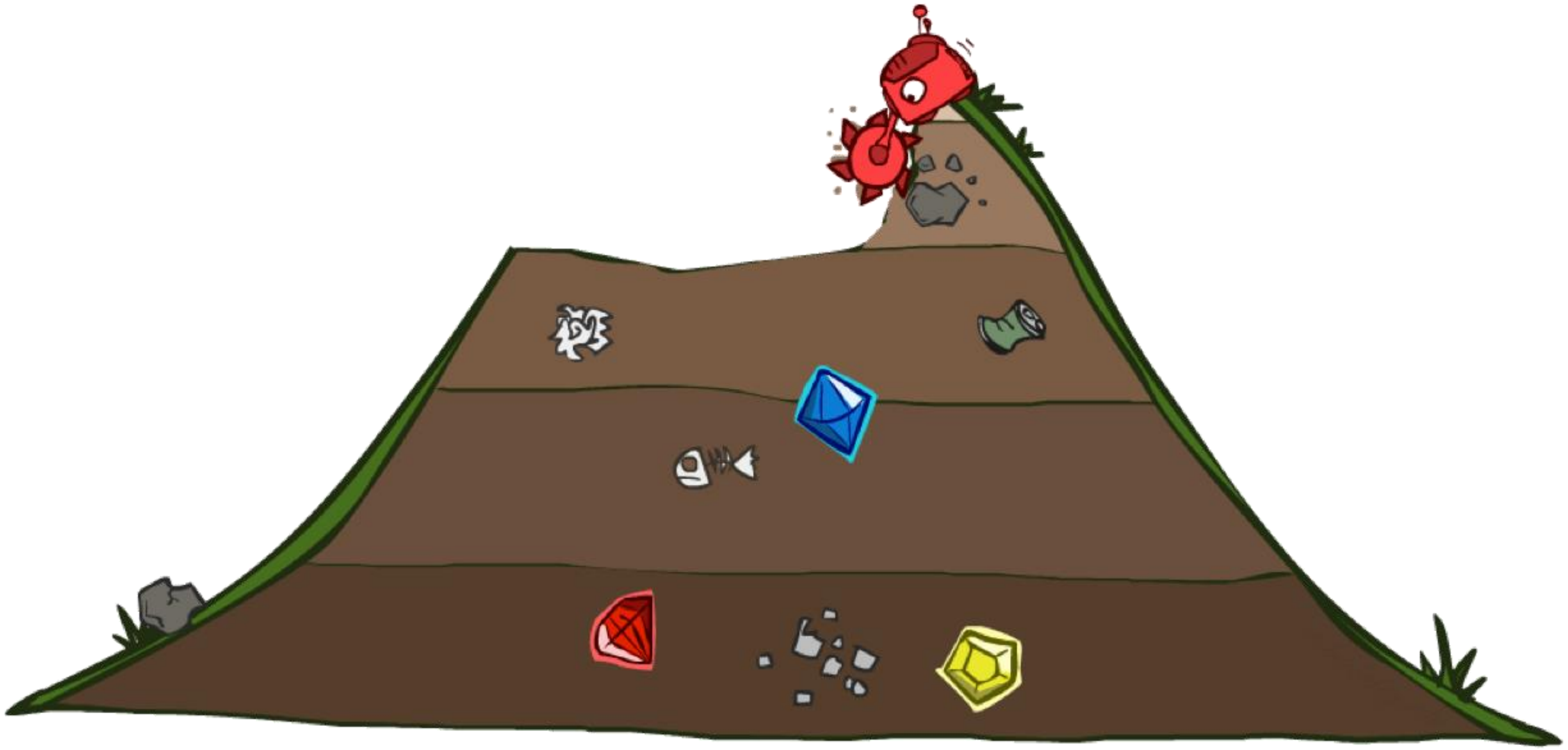


Cost-Sensitive Search



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.

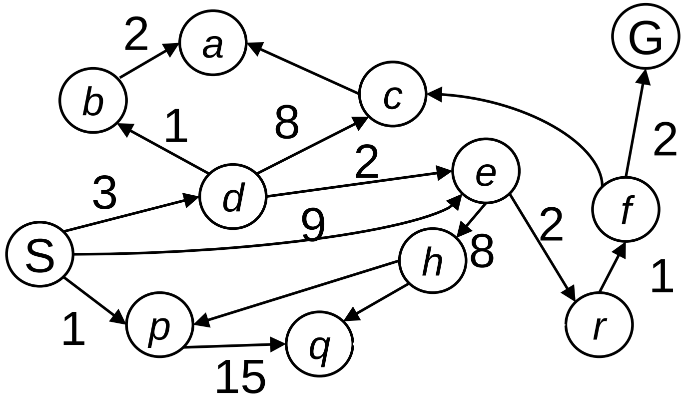
Uniform Cost Search



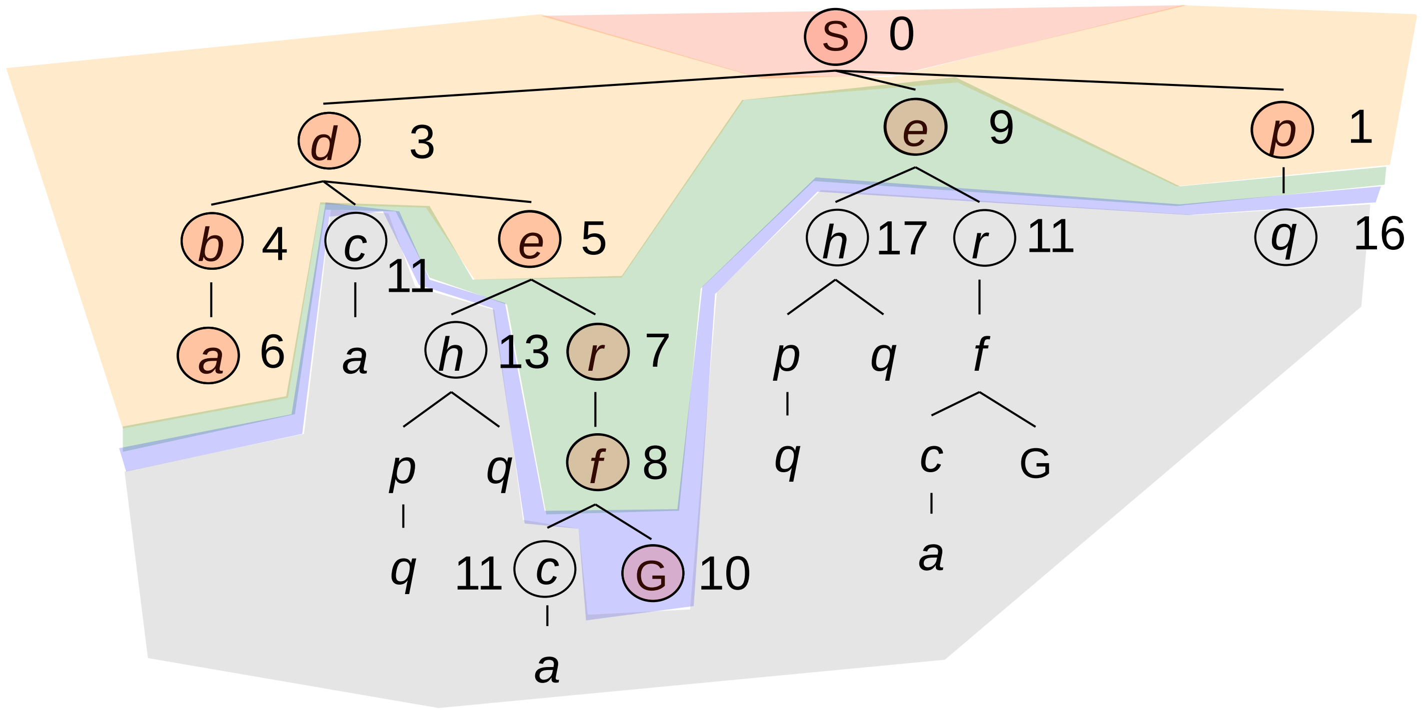
Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)

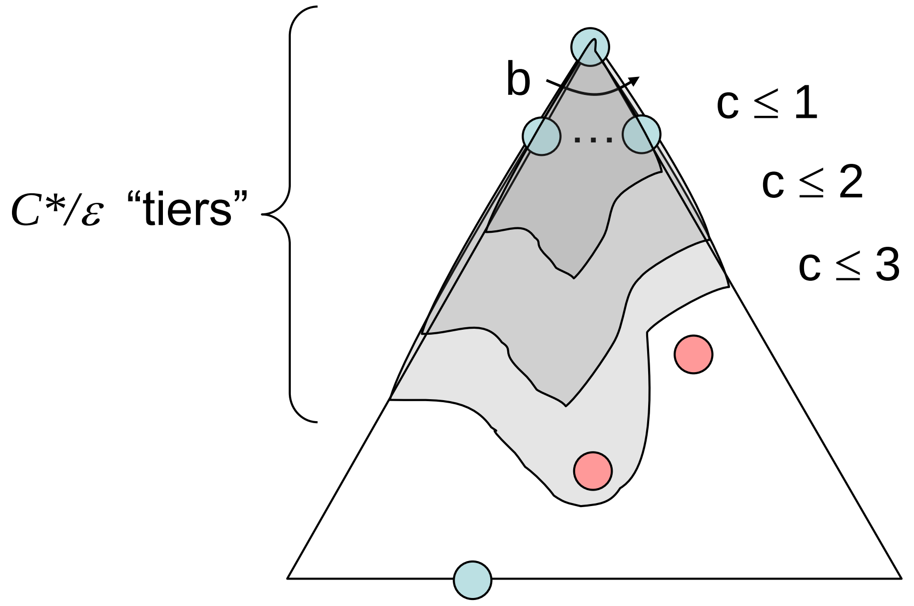


Cost contours



Uniform Cost Search (UCS) Properties

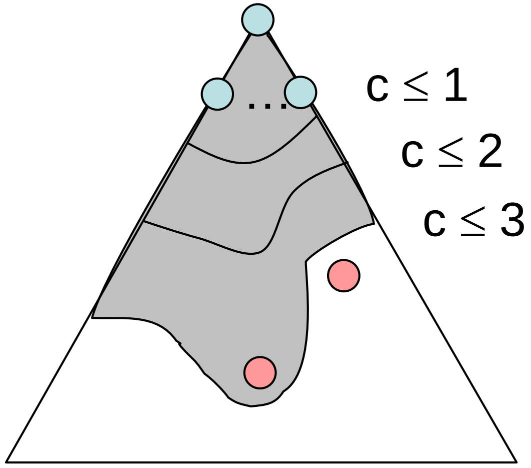
- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs $C*$ and arcs cost at least ε, then the "effective depth" is roughly $C*/\varepsilon$
  - Takes time O(b$^{C*/\varepsilon}$) (exponential in effective depth)

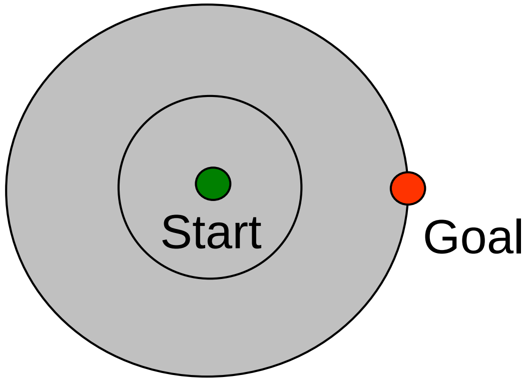


- How much space does the fringe take?
  - Has roughly the last tier, so O(b$^{C*/\varepsilon}$)
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!
- Is it optimal?
  - Yes!

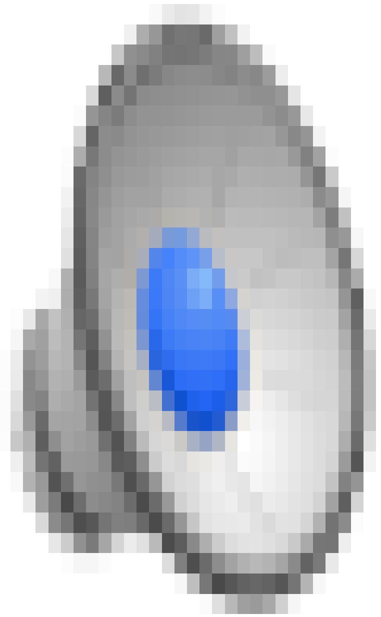
Uniform Cost Issues

- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location
- We'll fix that soon!

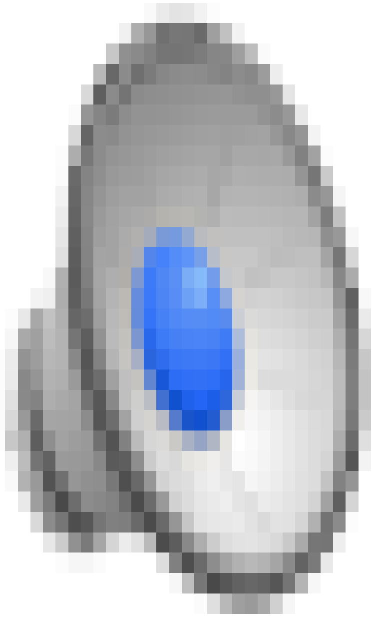




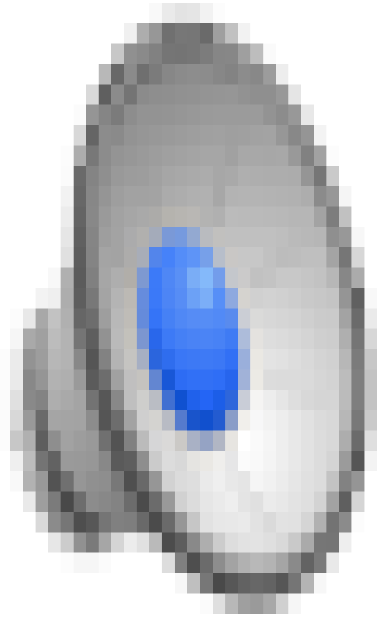
Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 1)



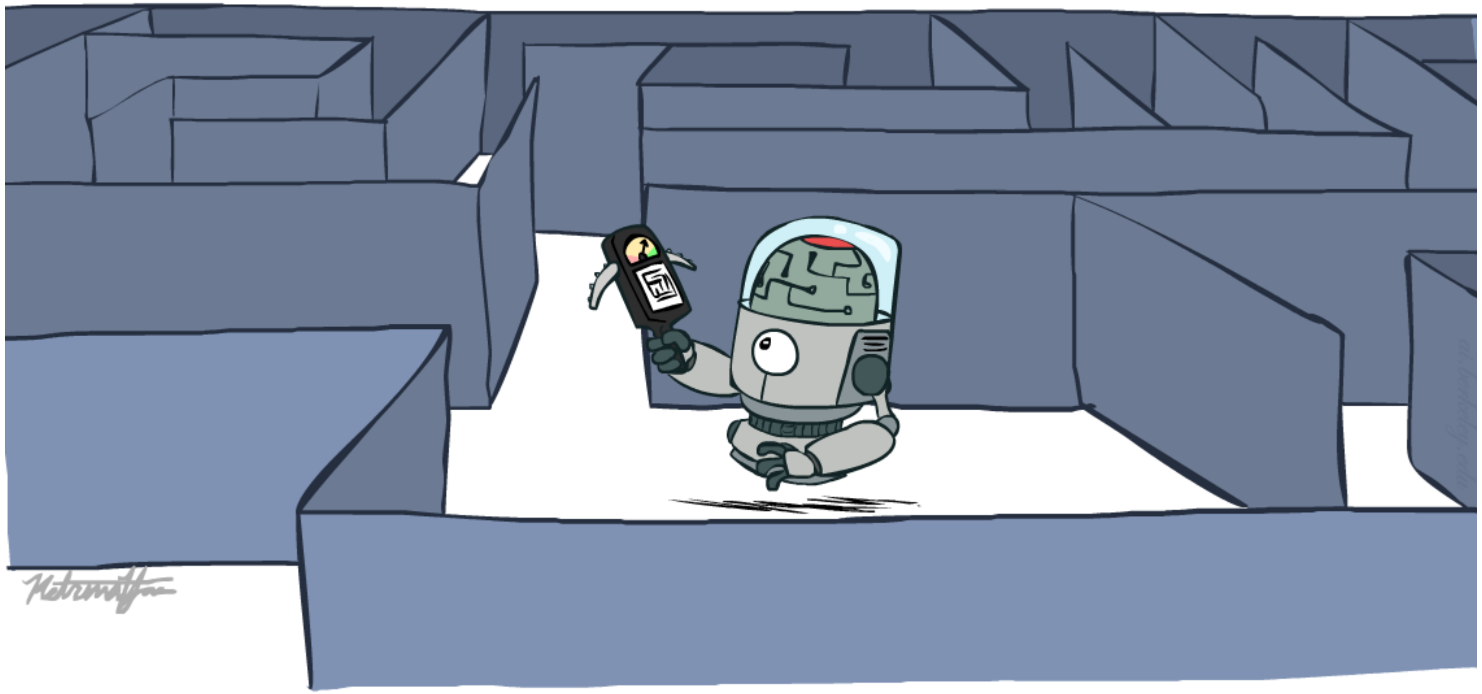
Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 2)



Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 3)

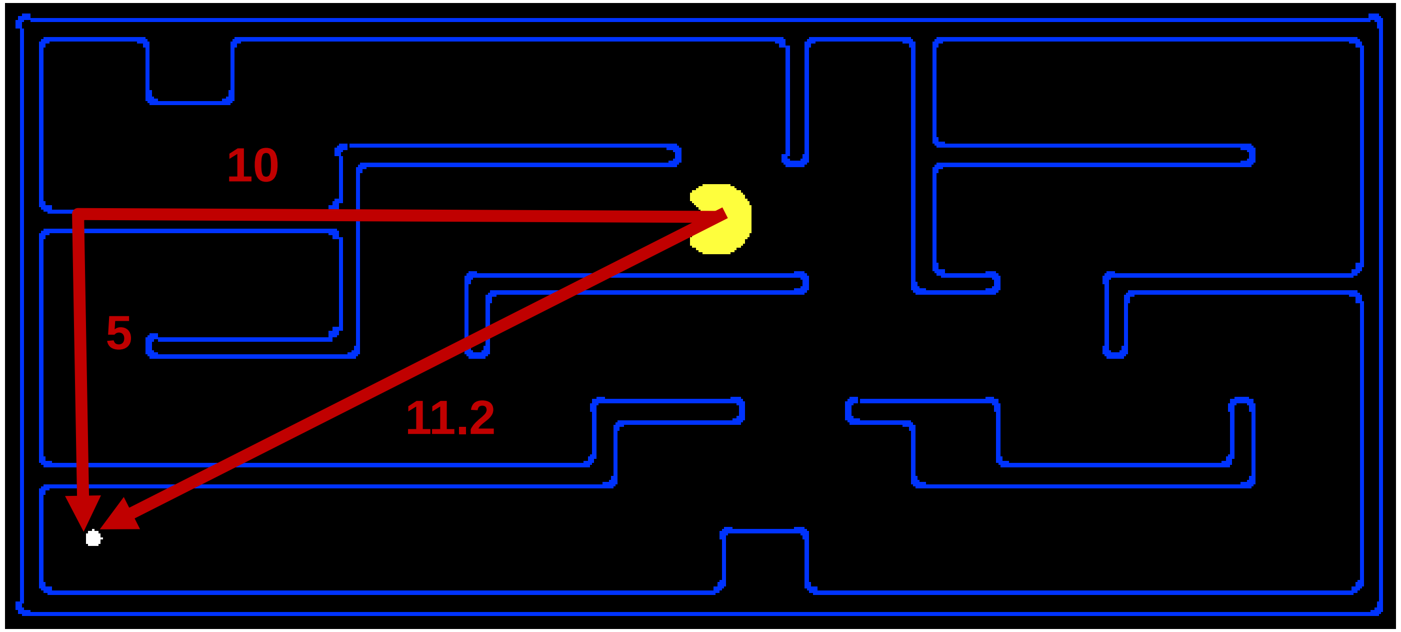


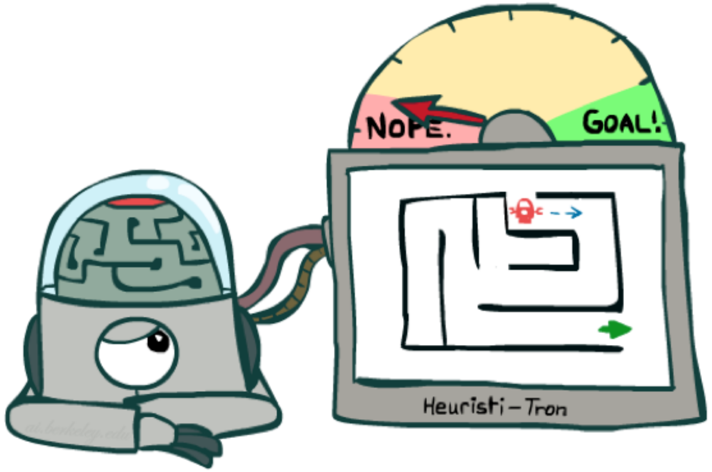
Informed Search

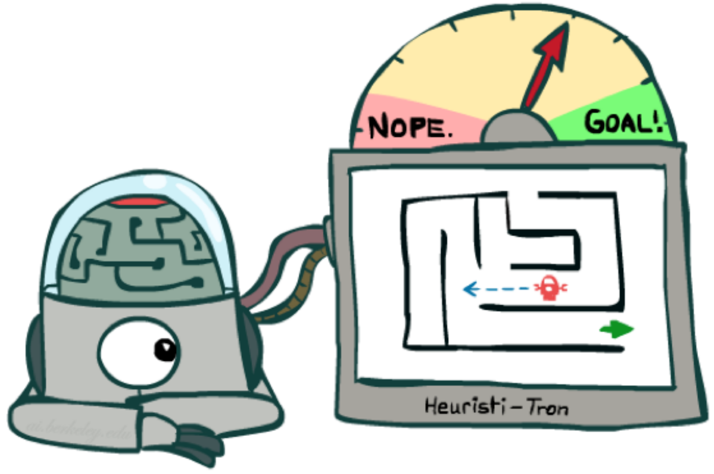


Search Heuristics

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



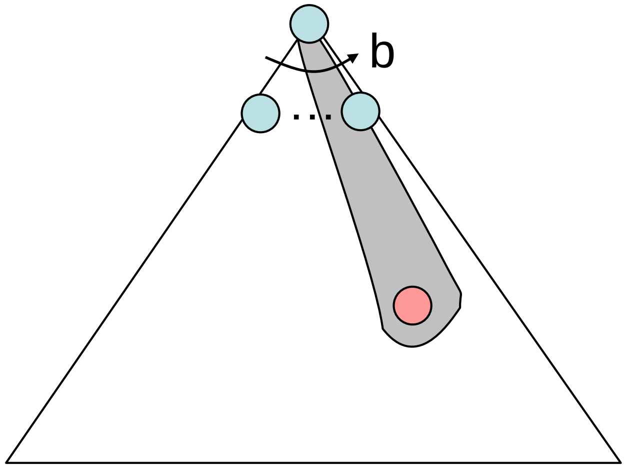


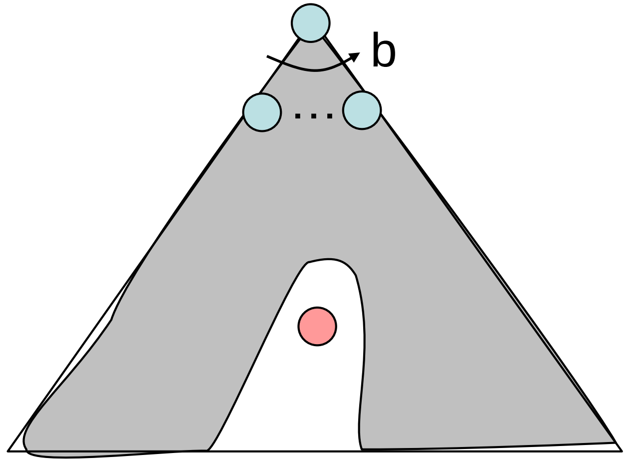


Greedy Search

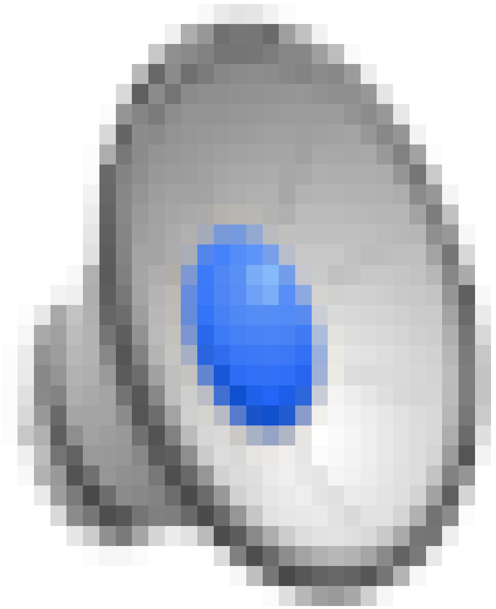
Greedy Search

- Strategy: expand a node that you think is closest to a goal state
- The ideal scenario:
  - Best-first takes you straight to the goal
- Worst-case: like a badly-guided DFS





Video of Demo Contours Greedy (Pacman Small Maze)



A* Search

A* Search

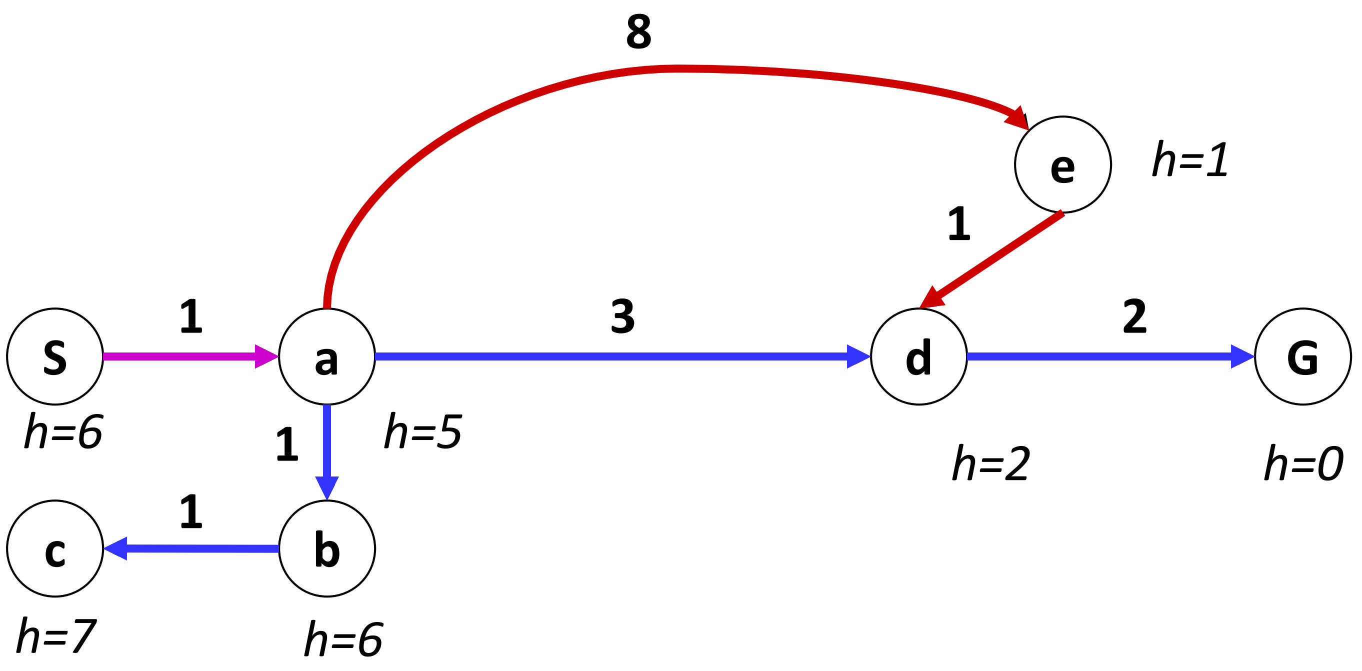
UCS

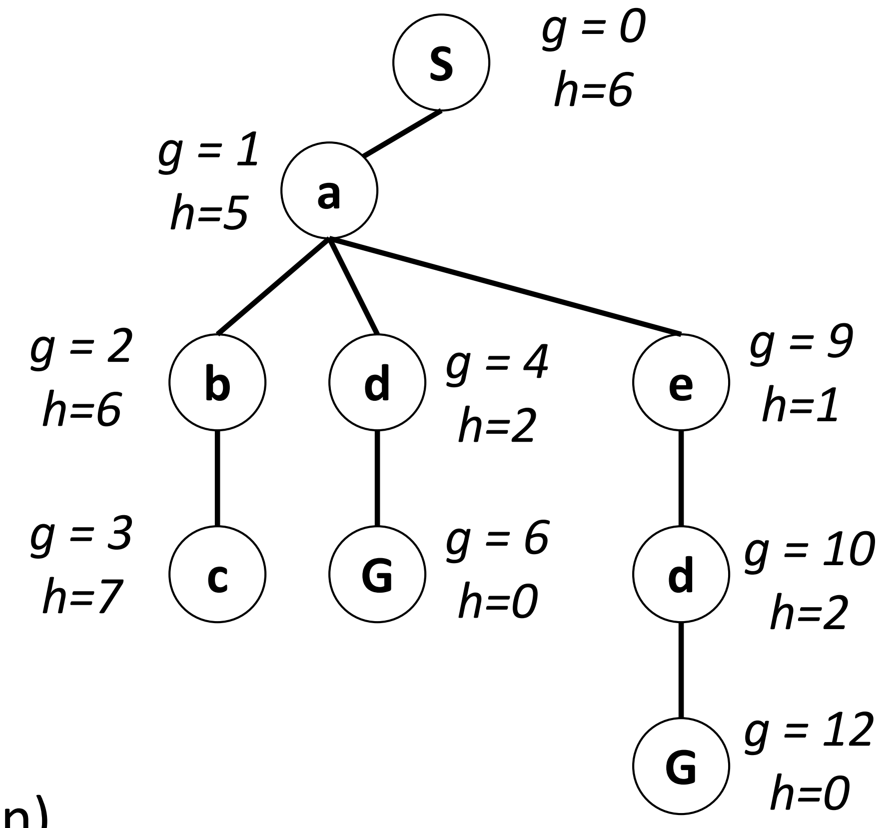
Greedy

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)

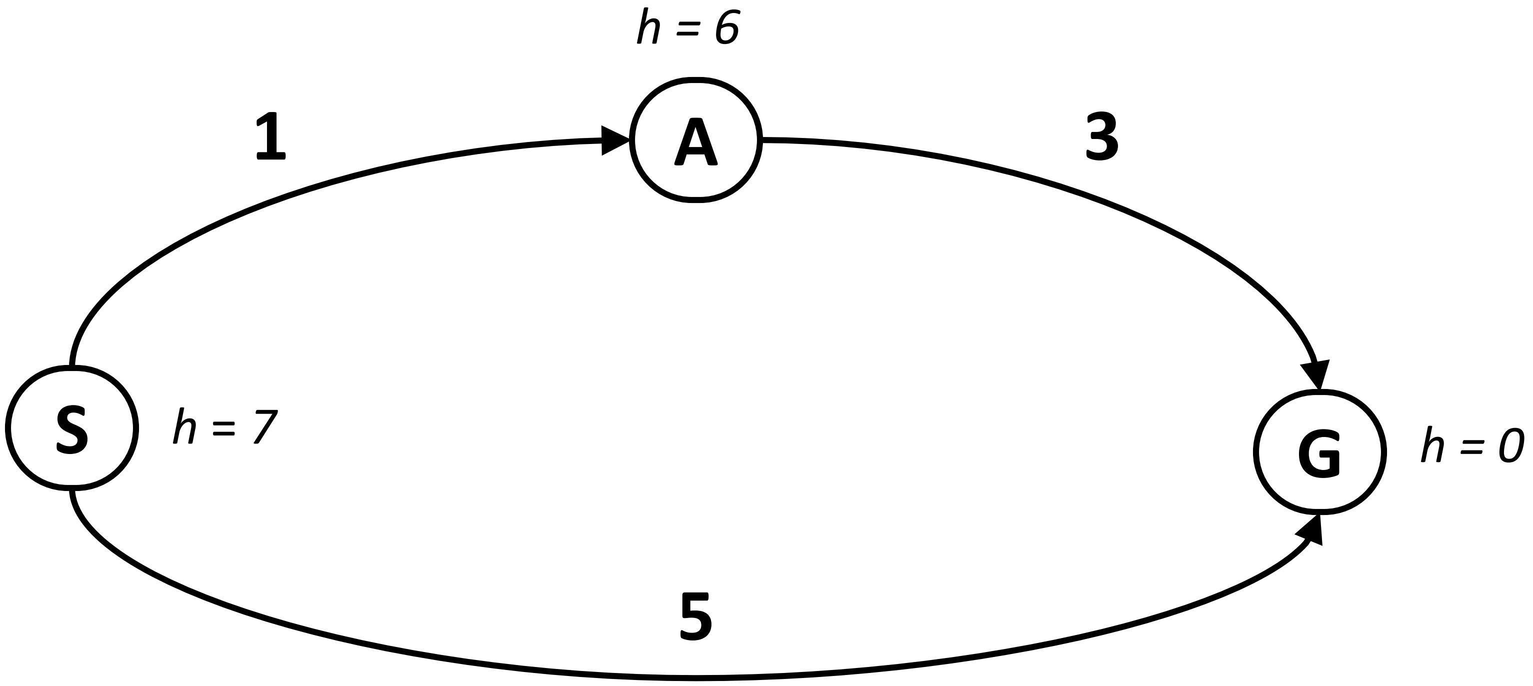




- A* Search orders by the sum: f(n) = g(n) + h(n)

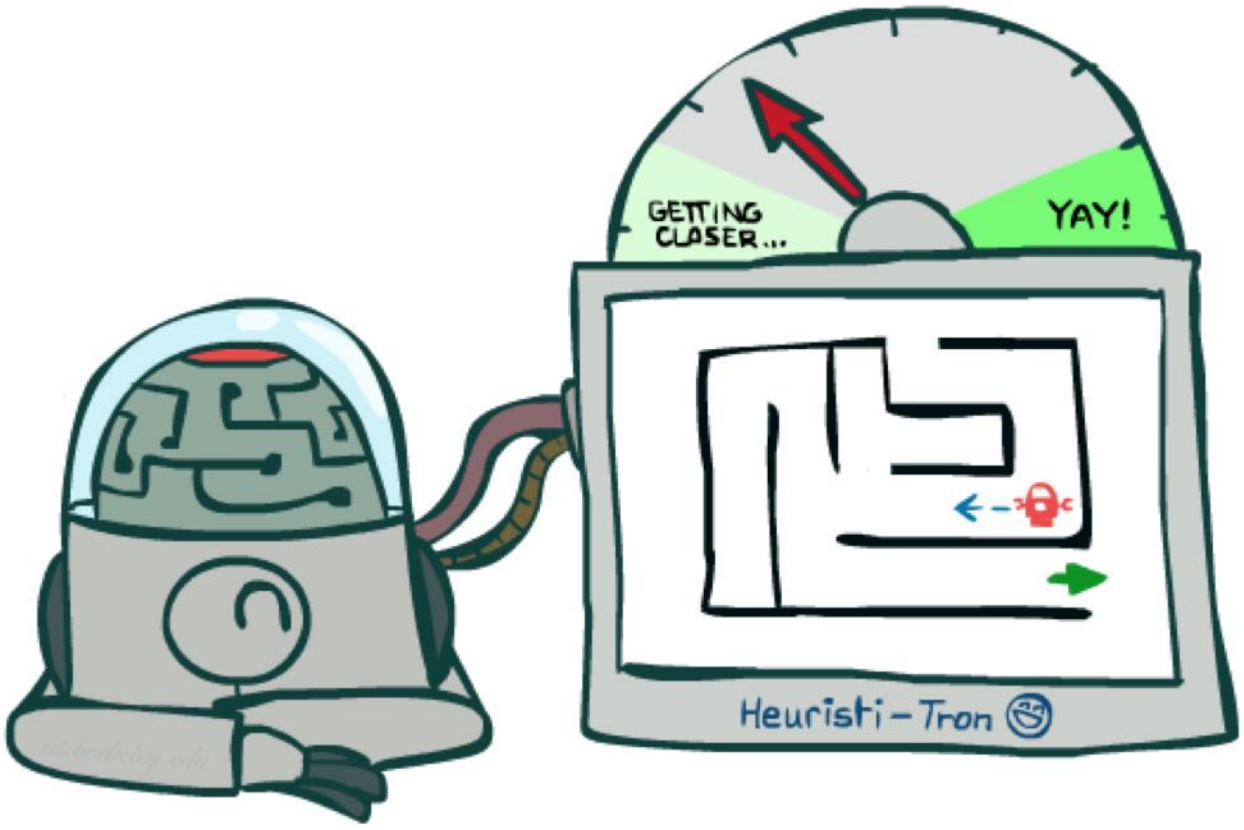
Example: Teg Grenager

Is A* Optimal?



- What went wrong?
- Over-estimated goal cost

Admissible Heuristics



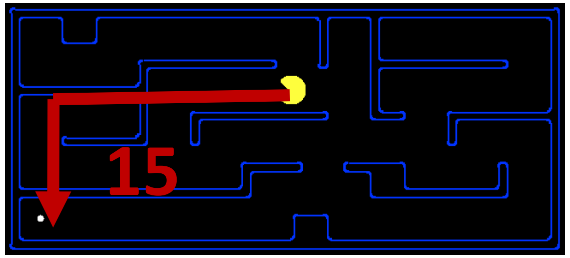
Admissible Heuristics

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

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

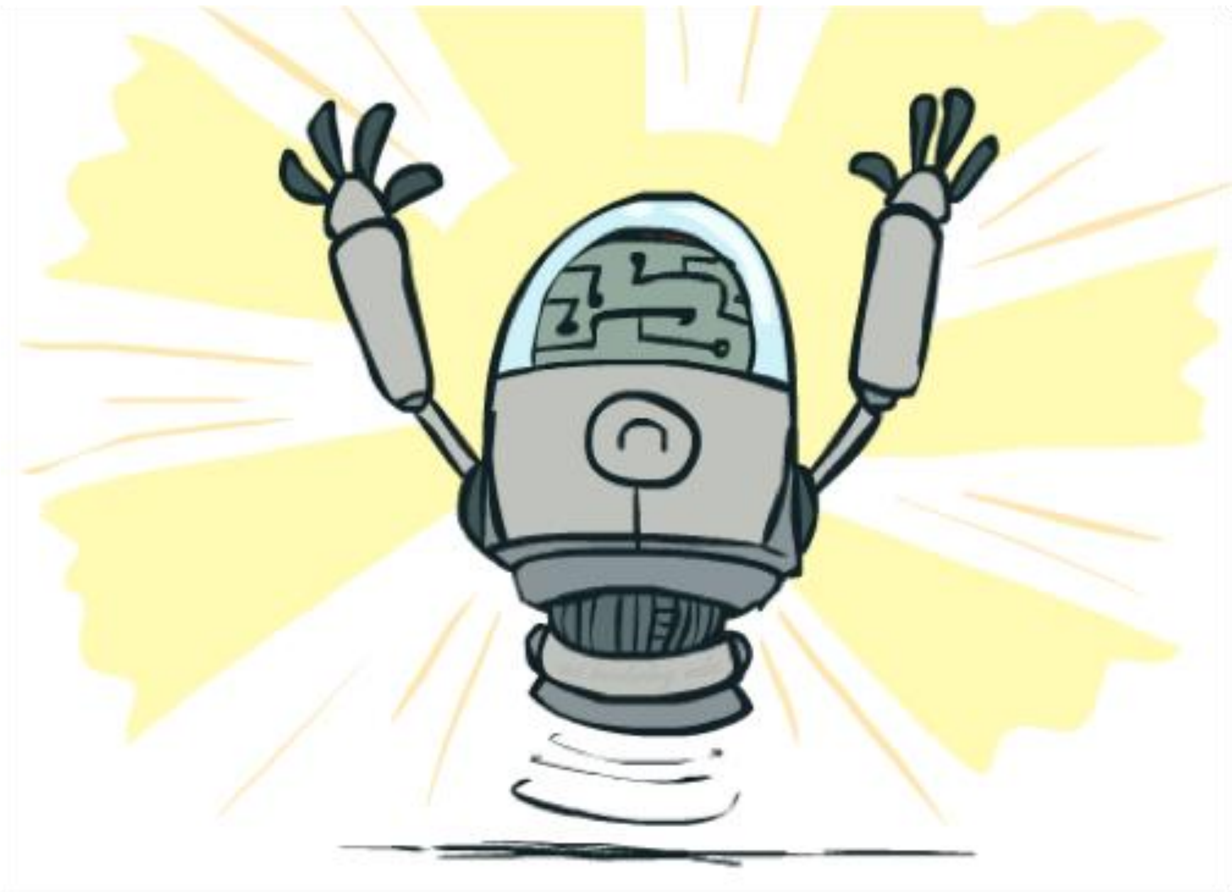
  where $h^*(n)$ is the true cost to a nearest goal

- Examples:



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

Optimality of A* Tree Search



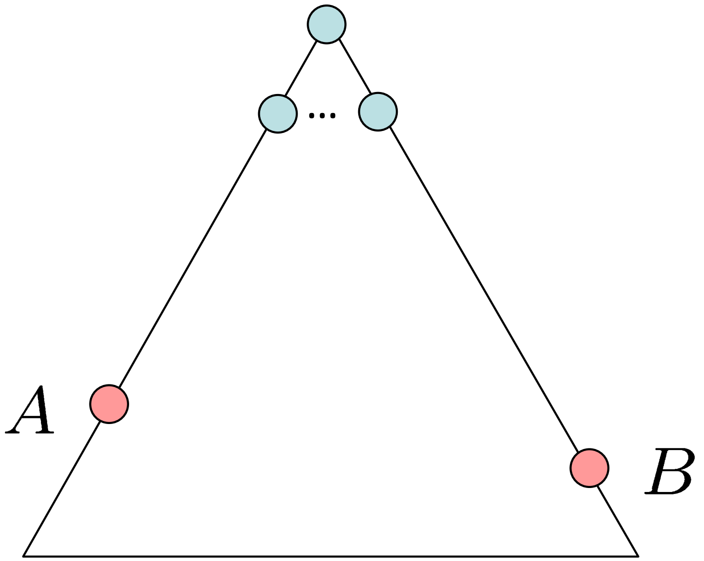
Optimality of A* Tree Search

Assume:

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

Claim:

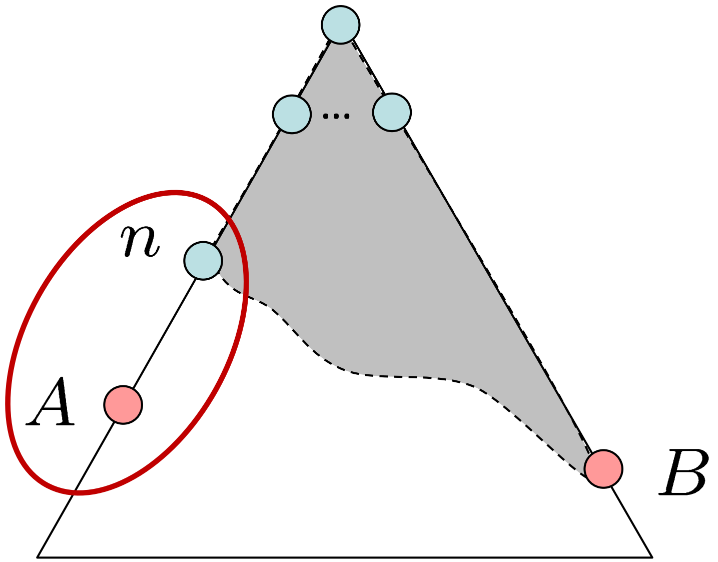
- A will exit the fringe before B



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)

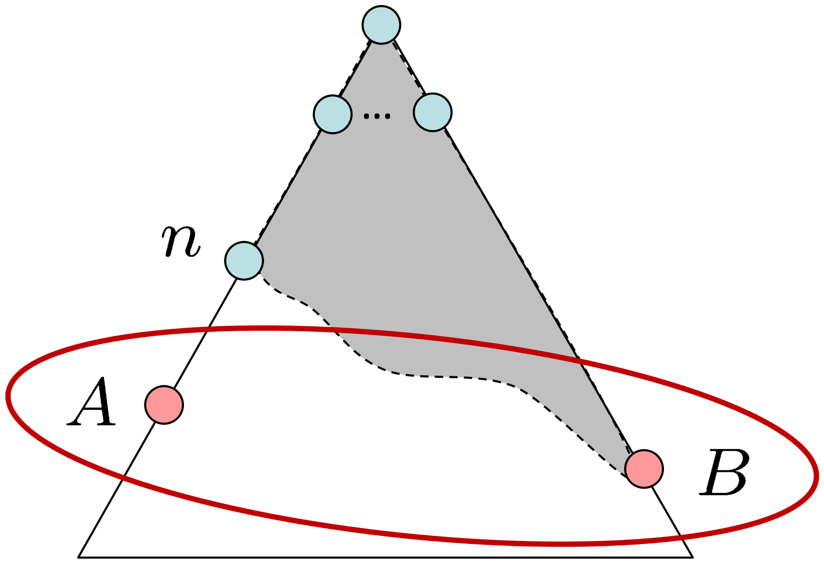


$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	h = 0 at a goal

Optimality of A* Tree Search: 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)



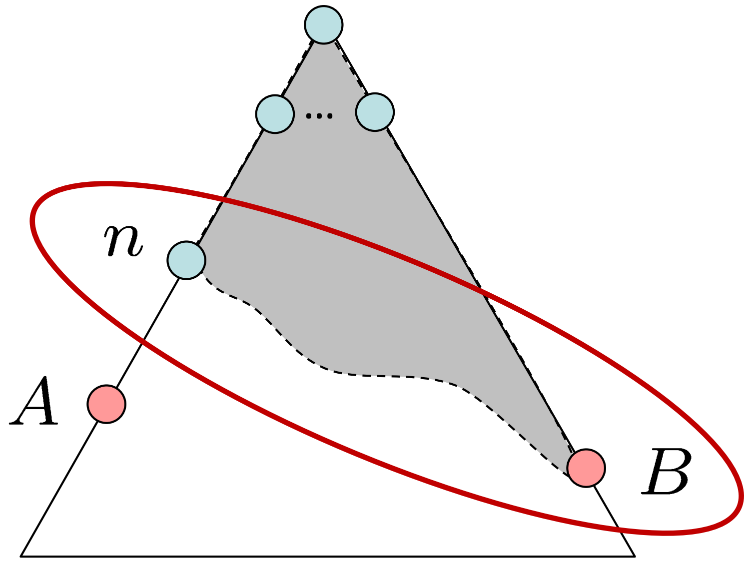
$g(A) < g(B)$ B is suboptimal

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

Optimality of A* Tree Search: Blocking

Proof:

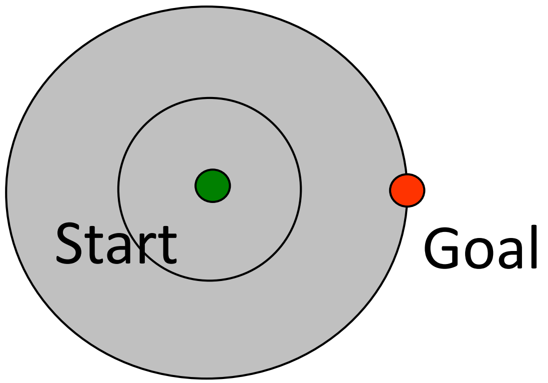
- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)
  3. *n* expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal



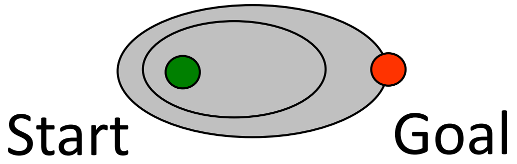
$$f(n) \leq f(A) < f(B)$$

UCS vs A* Contours

- Uniform-cost expands equally in all "directions"

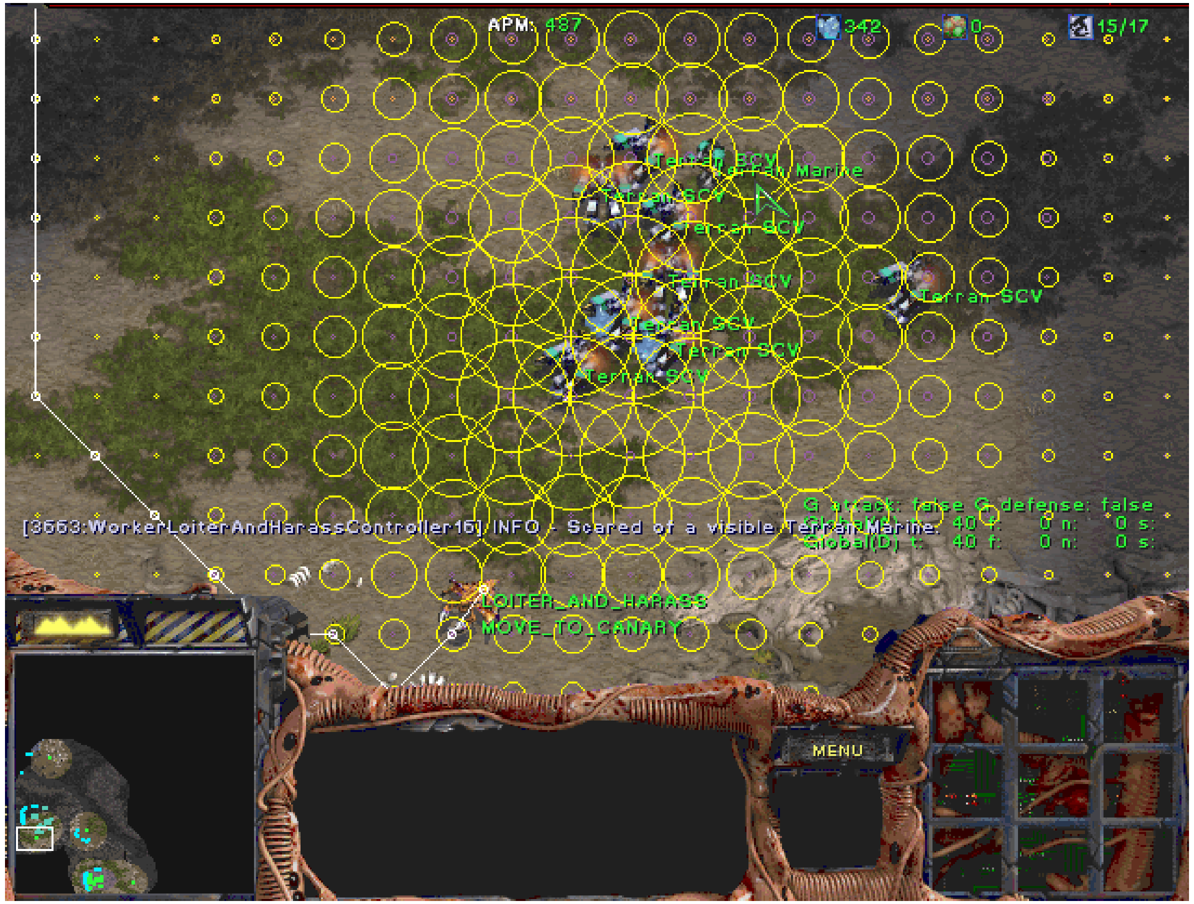


- A* expands mainly toward the goal, but does hedge its bets to ensure optimality

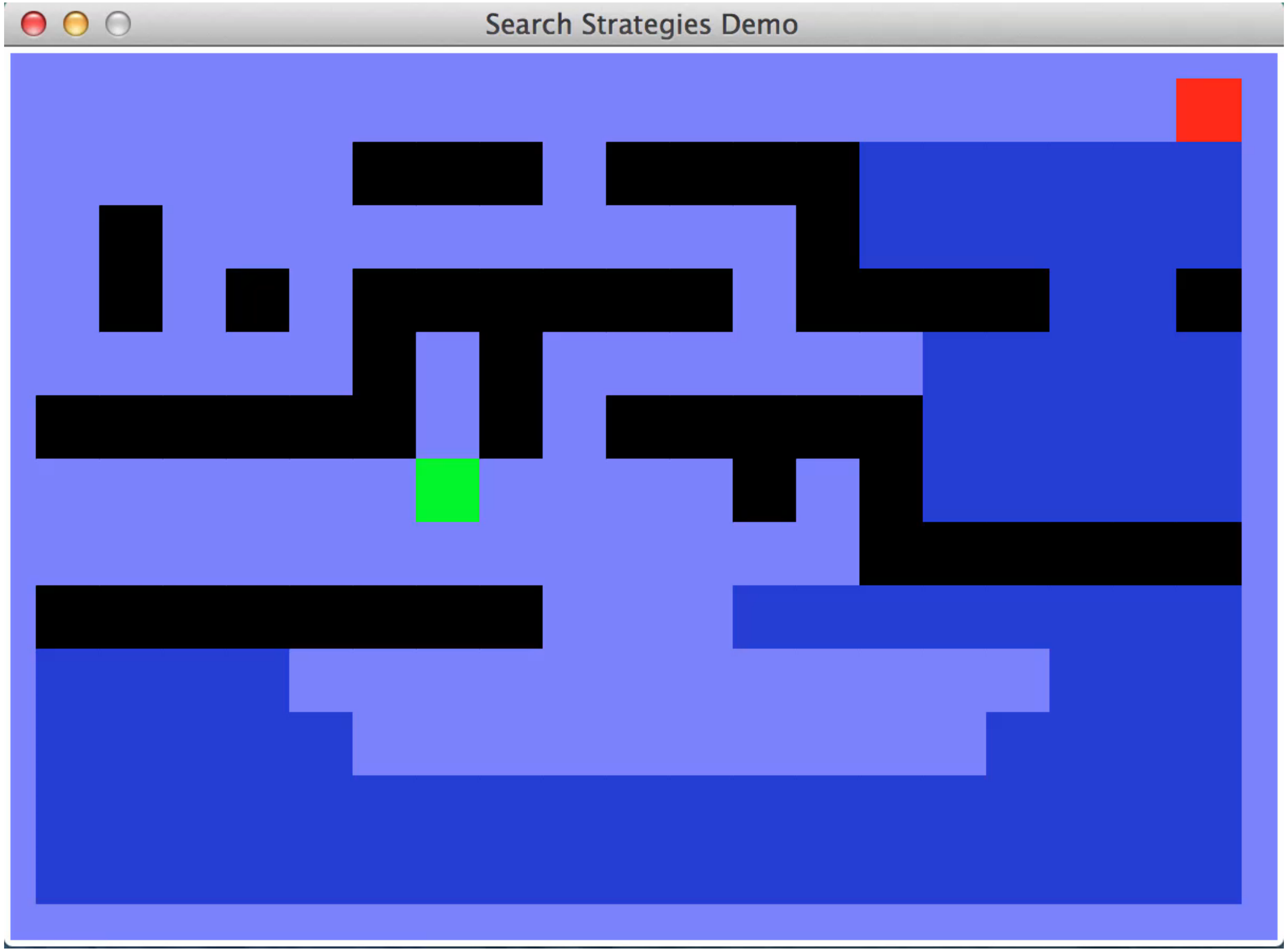


A* Applications

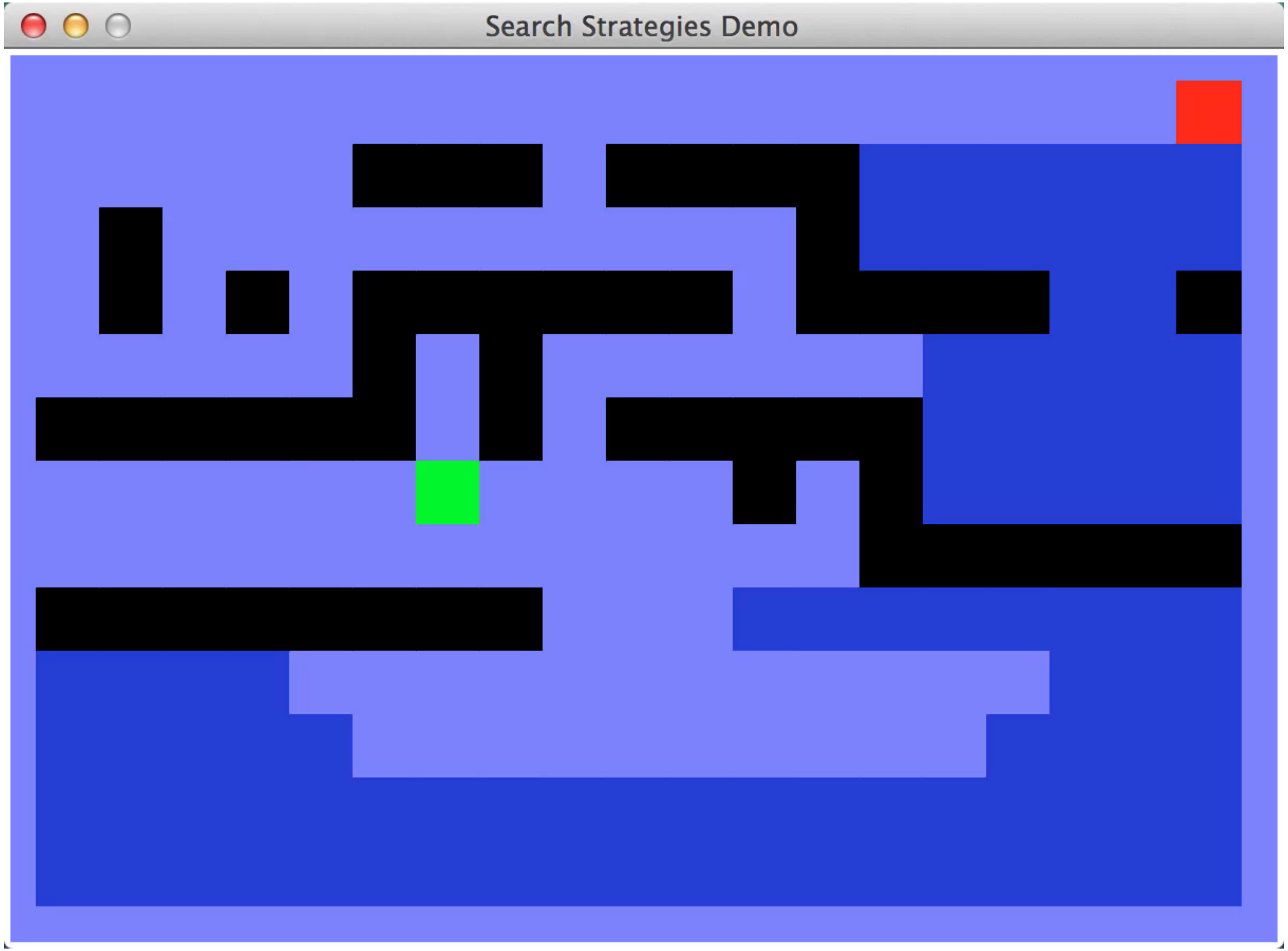
- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …



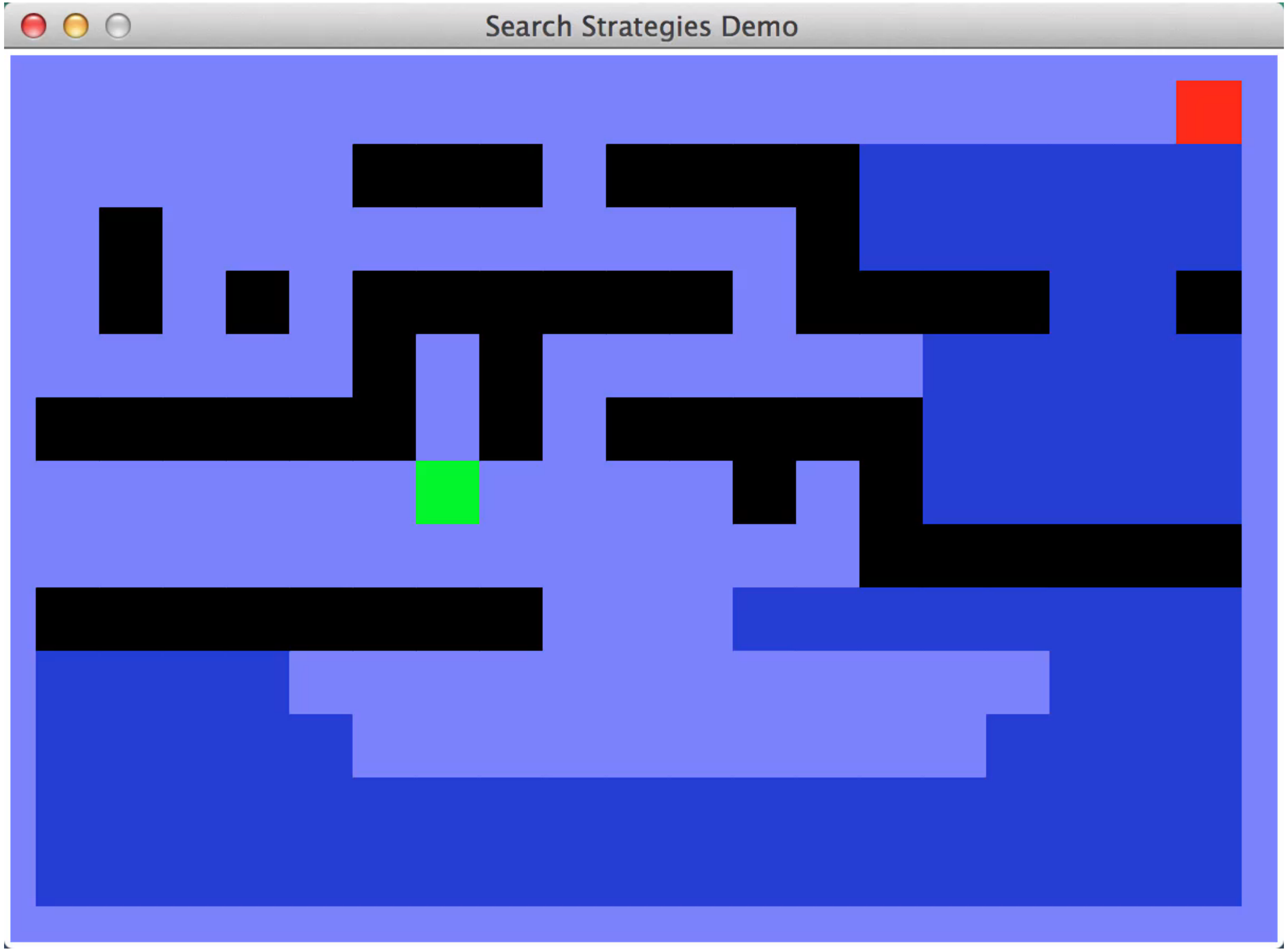
Video of Demo Maze with Deep/Shallow Water --- UCS, Greedy, A*



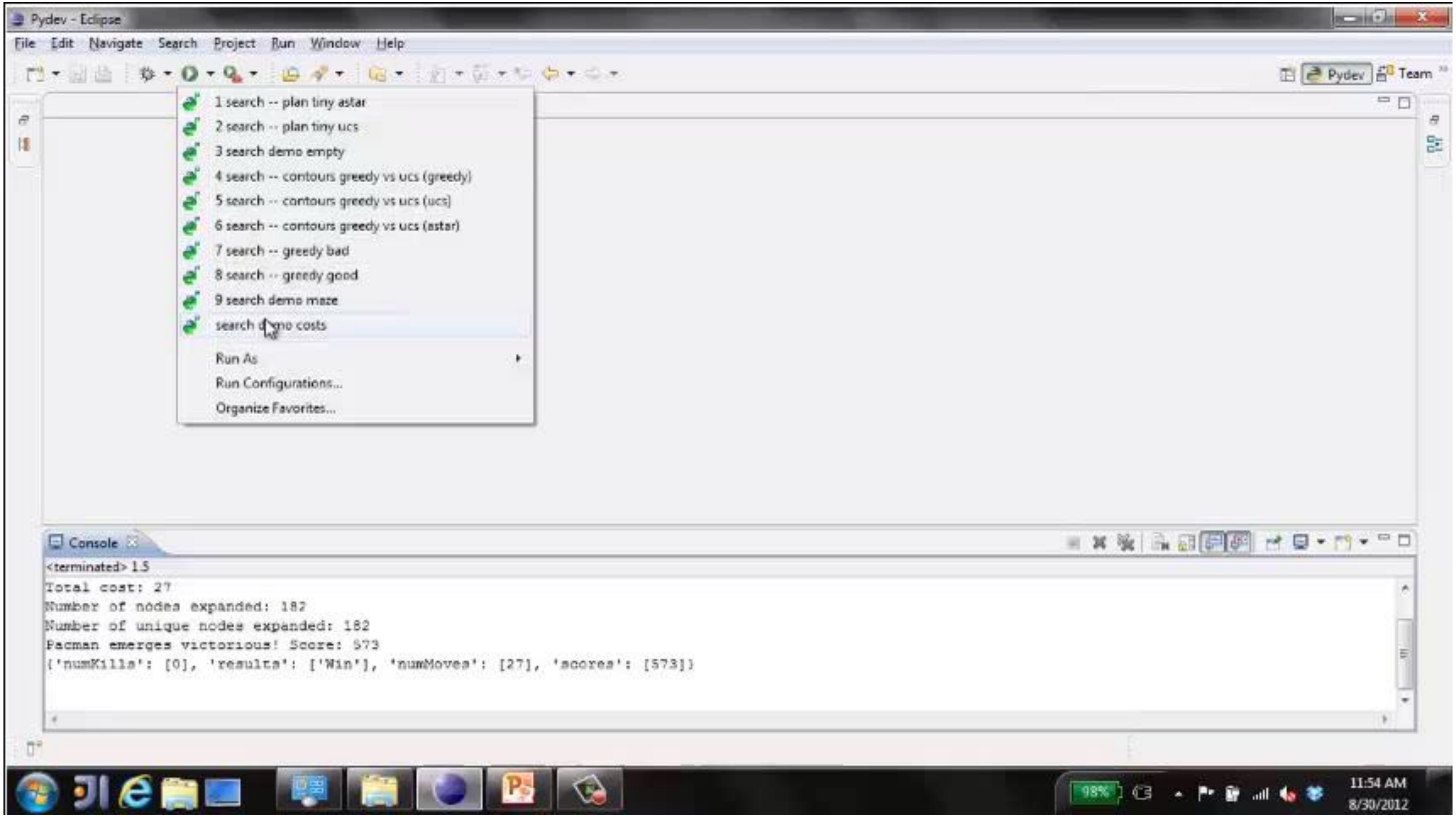
Video of Demo Maze with Deep/Shallow Water --- UCS, Greedy, A*



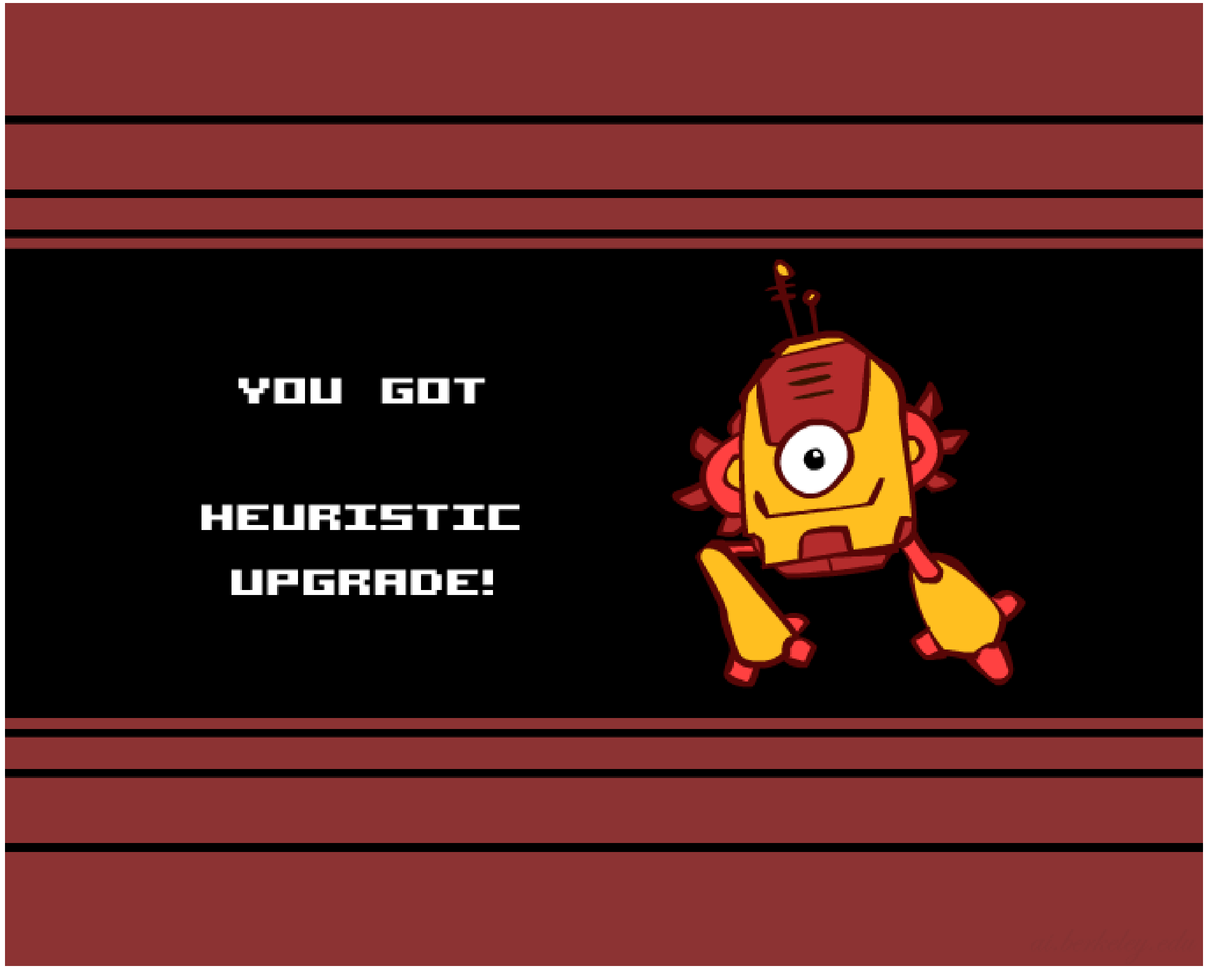
Video of Demo Maze with Deep/Shallow Water --- UCS, Greedy, A*



Video of Demo Empty Water Shallow/Deep – Guess Algorithm



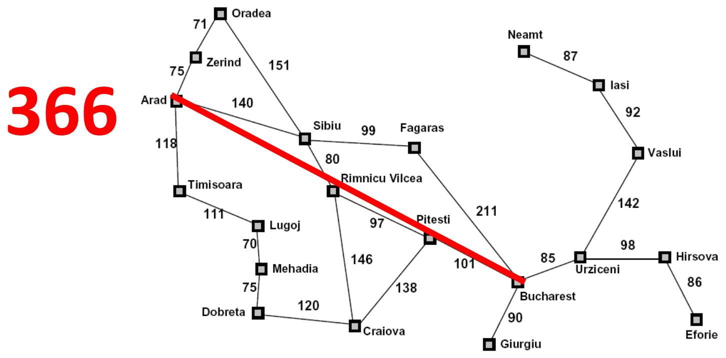
Creating Heuristics

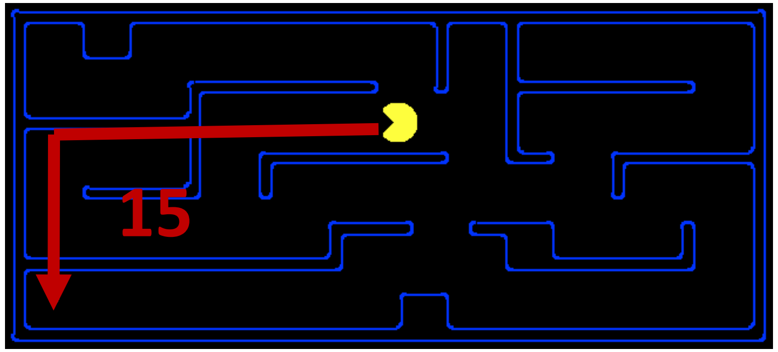


Creating Admissible Heuristics

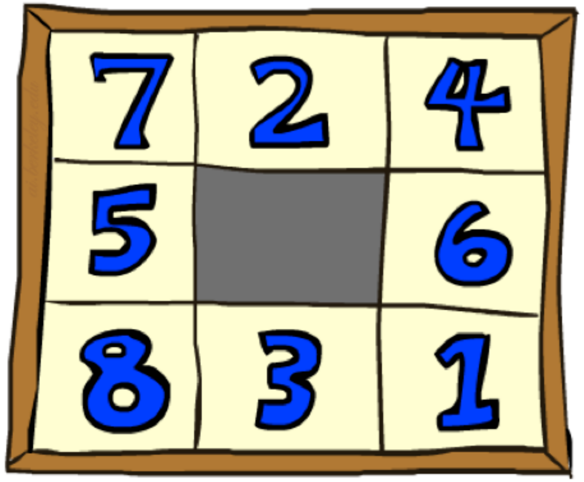
- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
  - Inadmissible heuristics are often useful too

- Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available

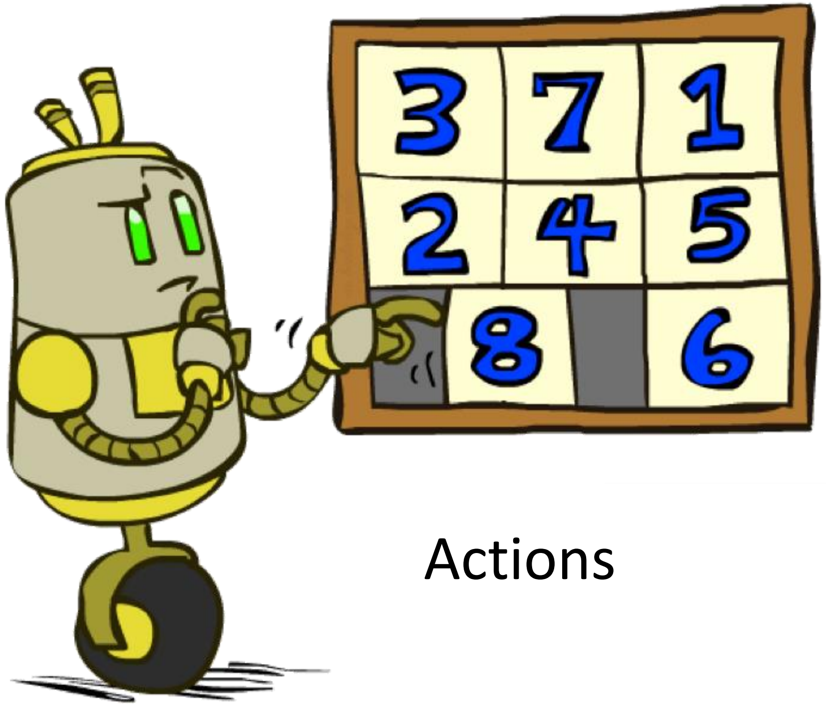




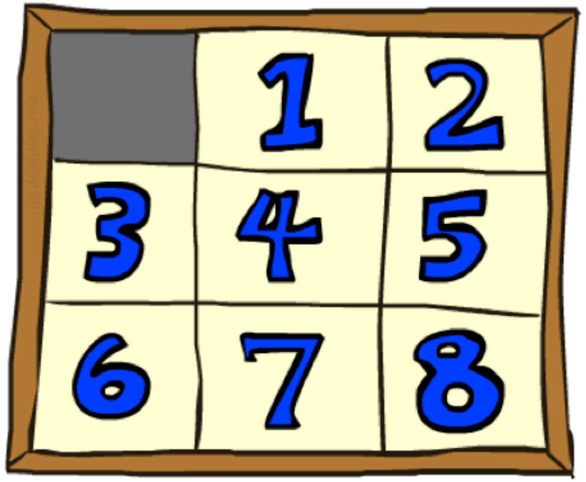
Example: 8 Puzzle



Start State



Actions

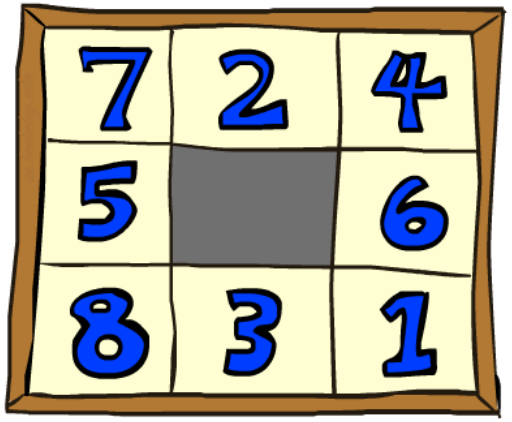


Goal State

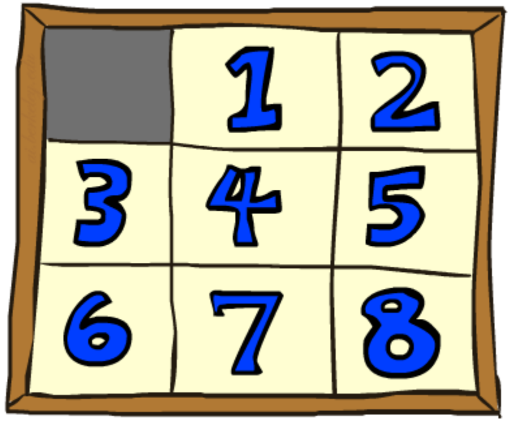
- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

8 Puzzle I

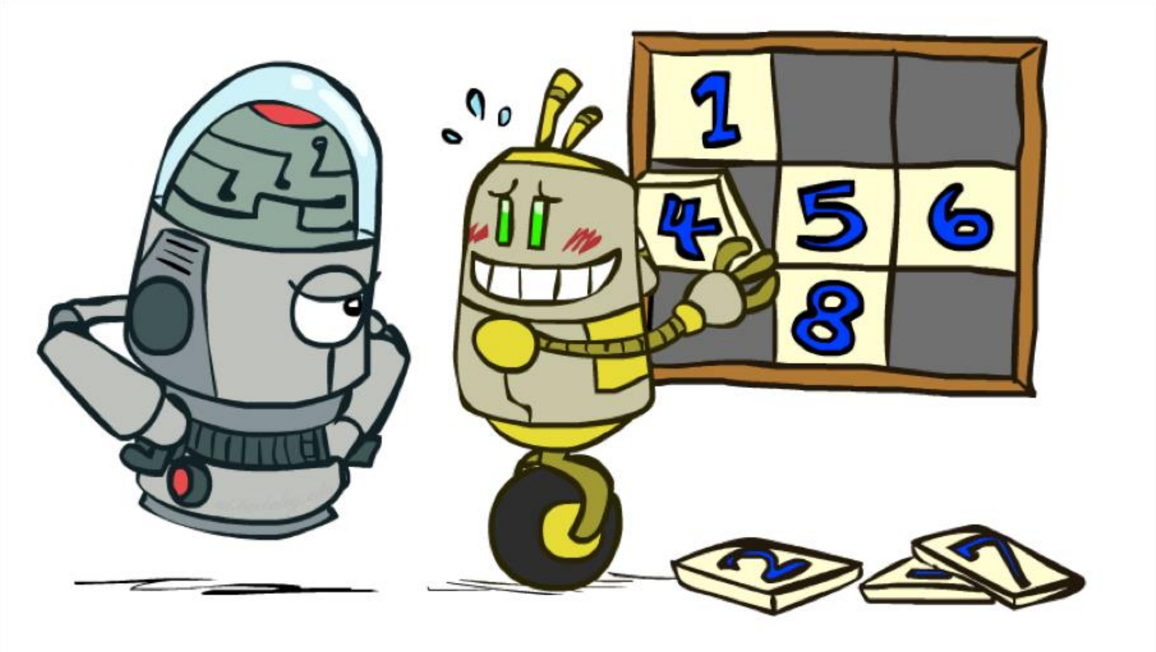
- Heuristic: Number of tiles misplaced
- h(start) = 8
- Is it admissible?
- This is a *relaxed-problem* heuristic



Start State



Goal State

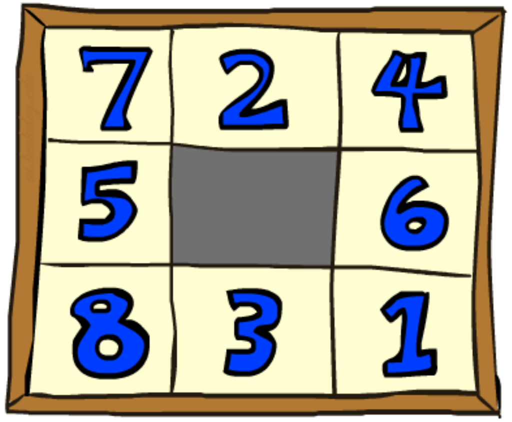


	Average nodes expanded when the optimal path has…		
	…4 steps	…8 steps	…12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

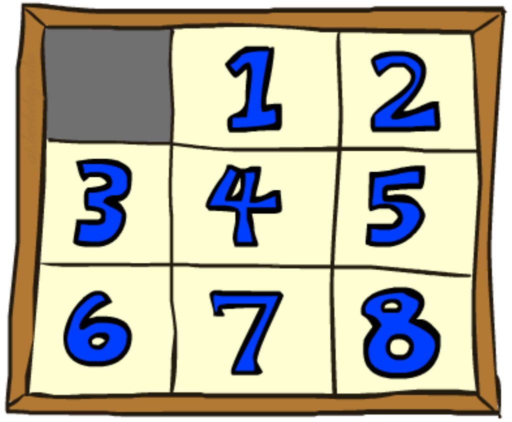
Statistics from Andrew Moore

8 Puzzle II

- Heuristc: total *Manhattan* distance
- h(start) = 3 + 1 + 2 + … = 18
- Is it admissible?
- *Relaxed-problem:* any tile could slide in any direction at any time, ignoring other tiles



Start State

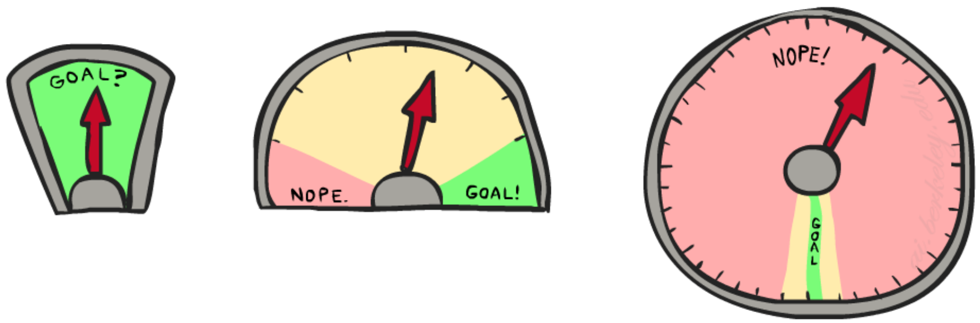


Goal State

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

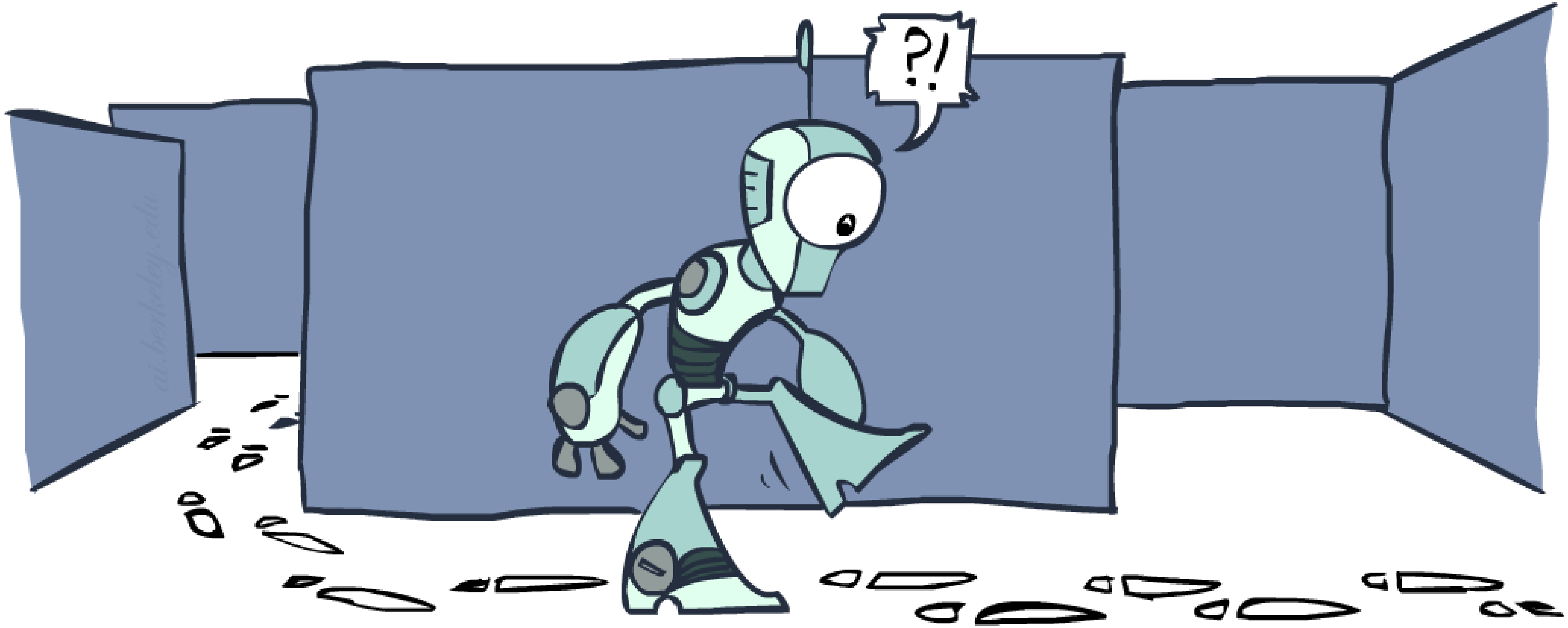
8 Puzzle III

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



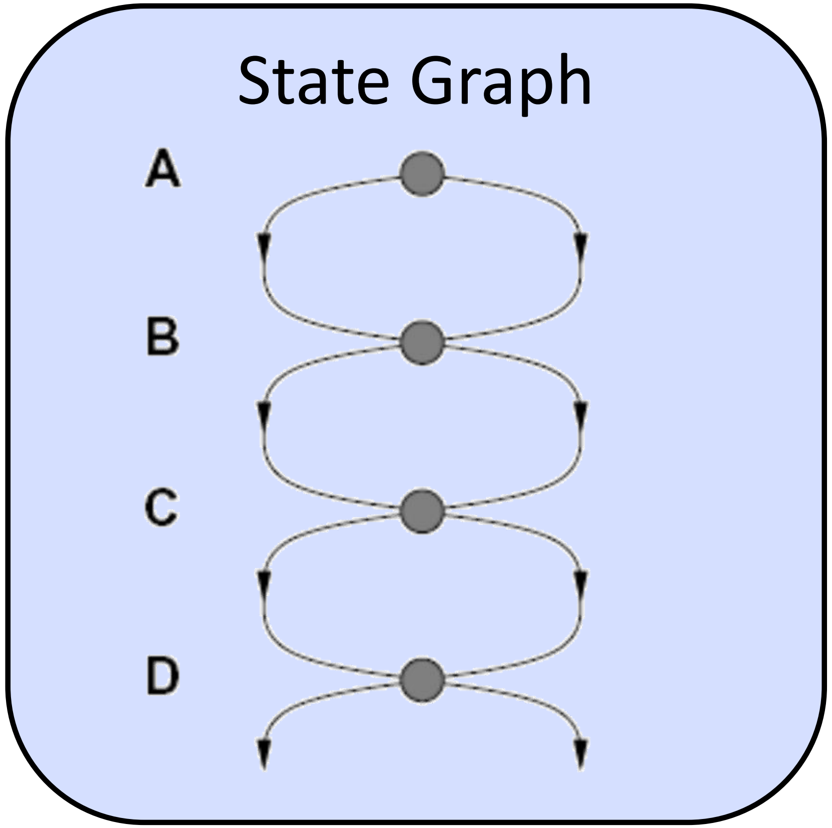
- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

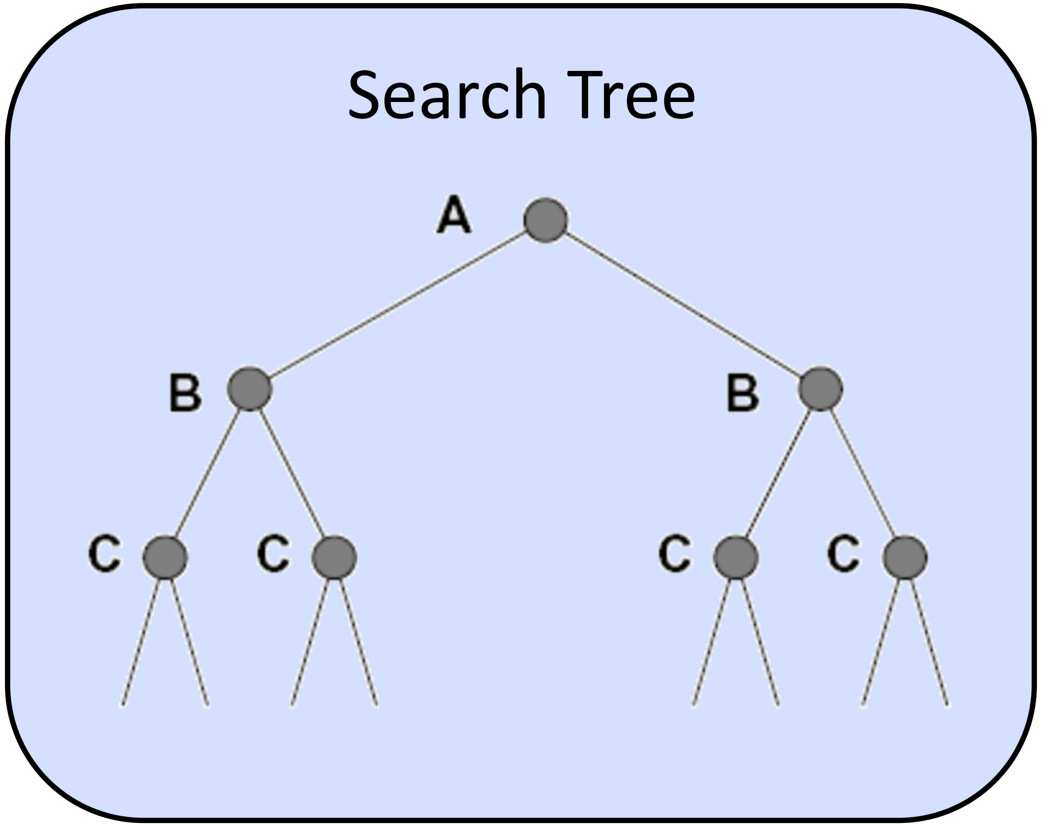
Graph Search



Tree Search: Extra Work!

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



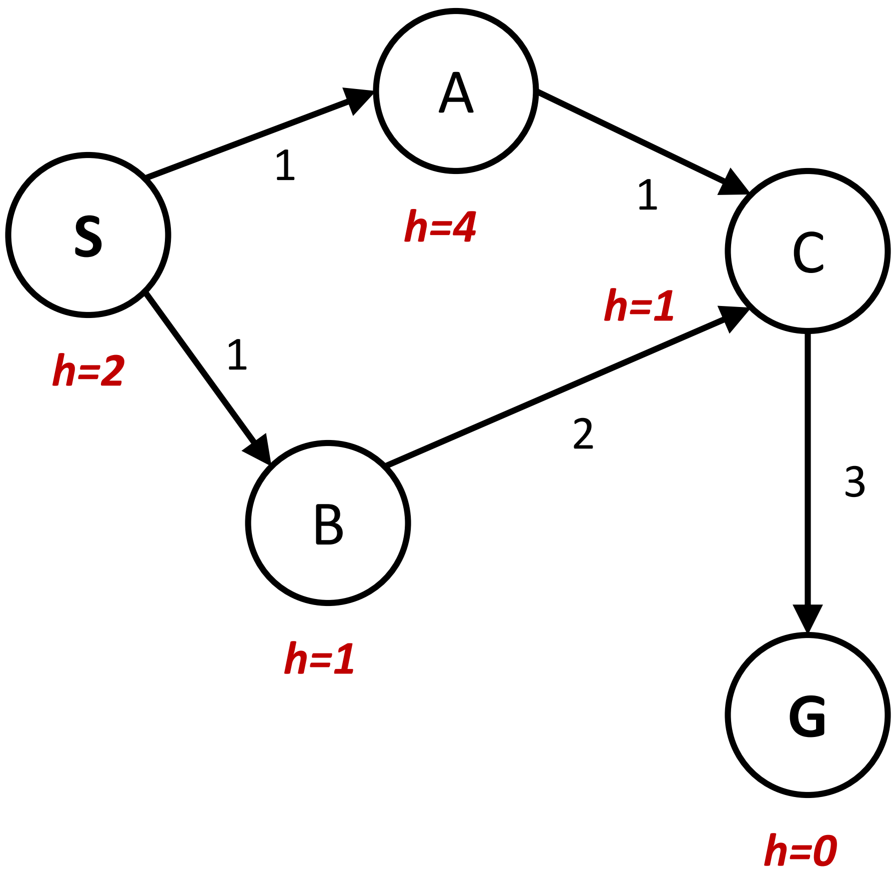


Graph Search

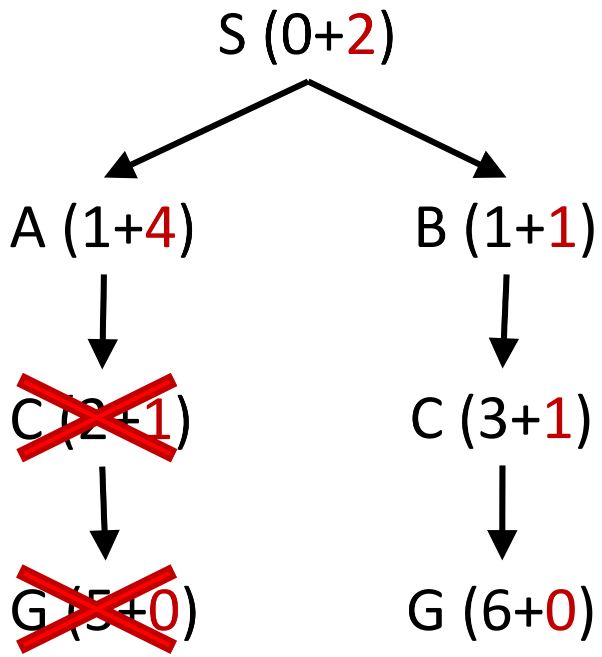
- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states (“closed set”)
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set
- Can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

A* Graph Search Gone Wrong?

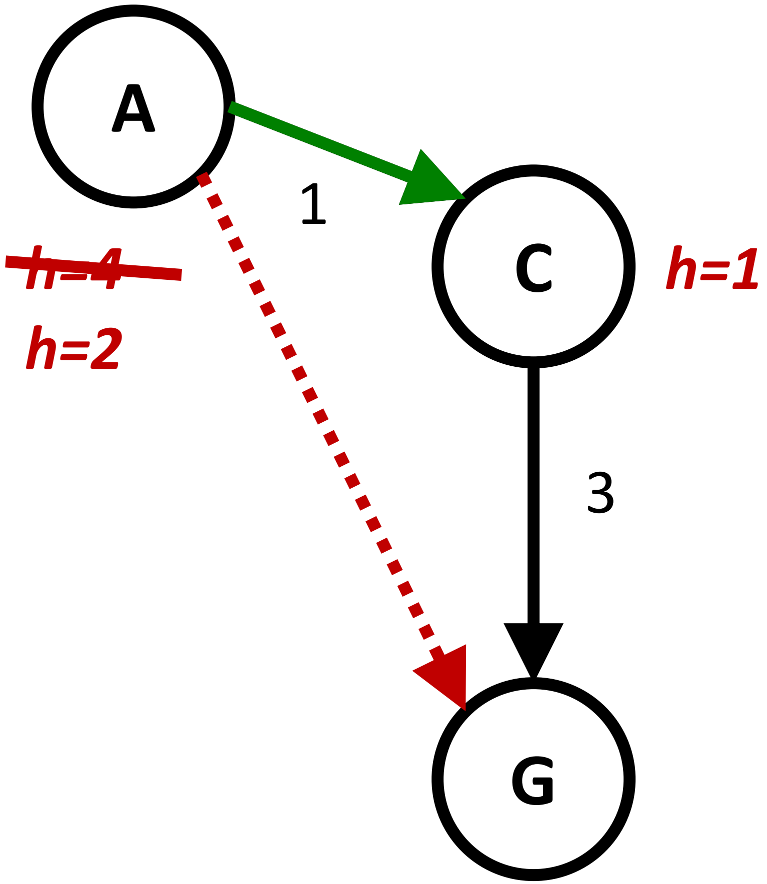
State space graph



Search tree



Consistency of Heuristics



- Admissibility: heuristic cost ≤ actual cost to goal

  h(A) ≤ actual cost from A to G

- Consistency: heuristic "arc" cost ≤ actual cost for each arc

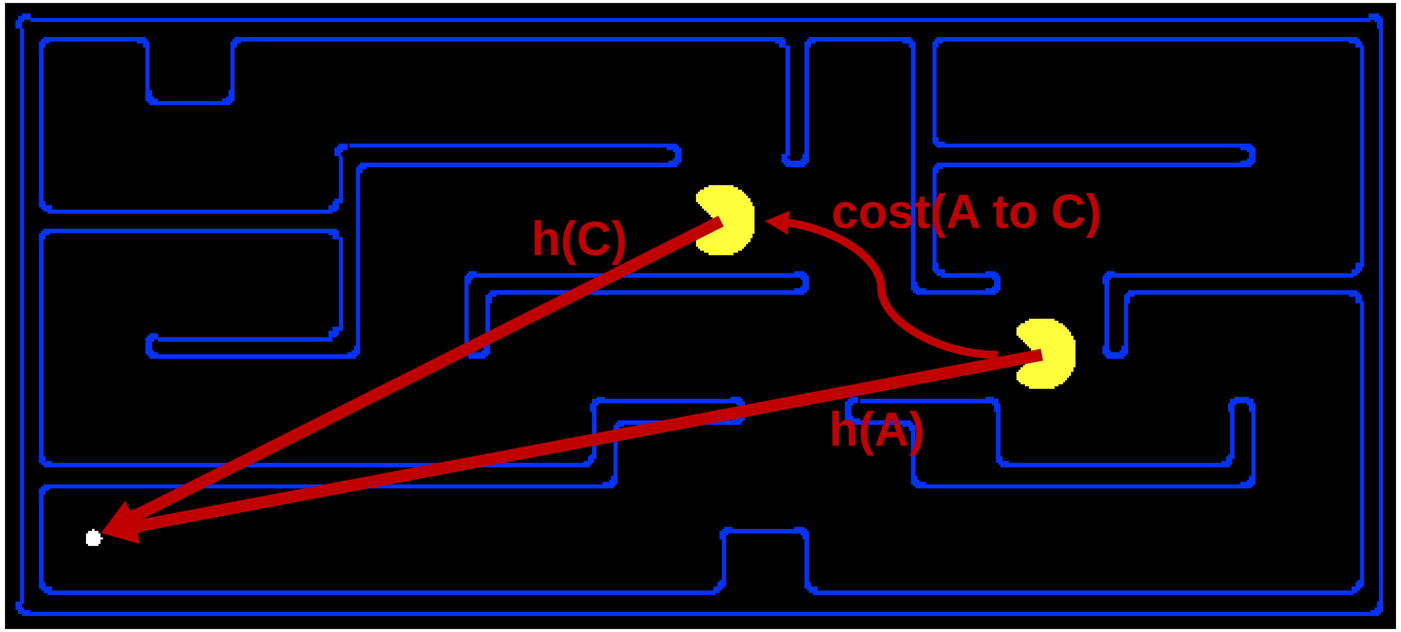
  h(A) – h(C) ≤ cost(A to C)

- Consistency implies admissibility

- A* graph search is optimal if heuristic is consistent
  - See textbook for a proof

Consistency of Heuristics

- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems
- Example: Manhattan distance or Euclidean distance for pathing



Summary

- Why search
  - Agents that Plan Ahead
- Search Problems
  - state space, successor function
  - start state and goal test
- Uninformed Search Methods
  - DFS, BFS, UCS
- Informed Search
  - Greedy, A*
- Graph Search