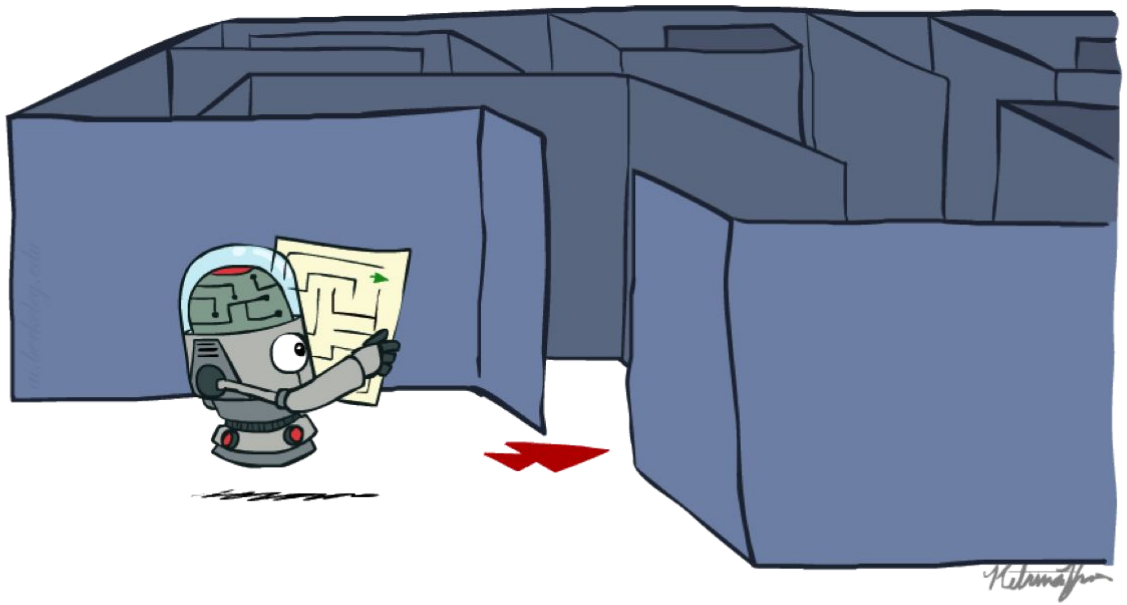
Search



These slides are primarily based on the slides created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley.
The artwork is by Ketrina Yim.

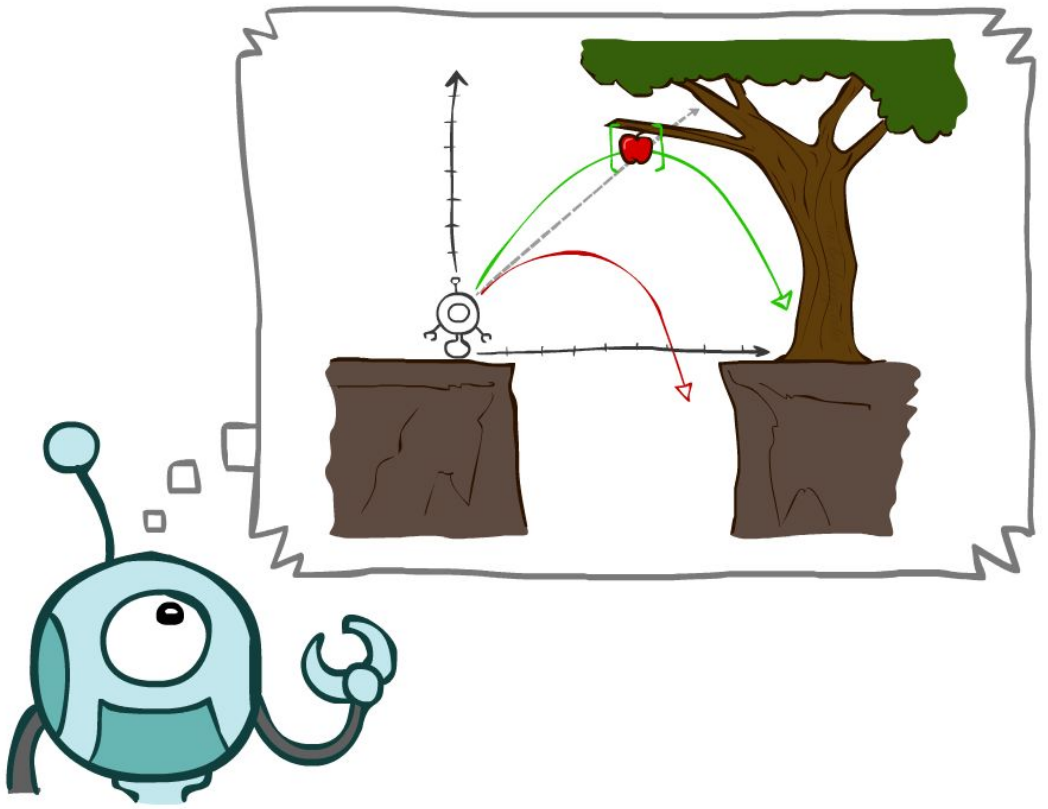
Problem Solving Agent Design

1. Formulate the problem ✔
2. Search for a solution
3. Execute the solution

Today

Uninformed Search Methods

- Depth-First Search
- Breadth-First Search
- Iterative Deepening
- Uniform-Cost Search

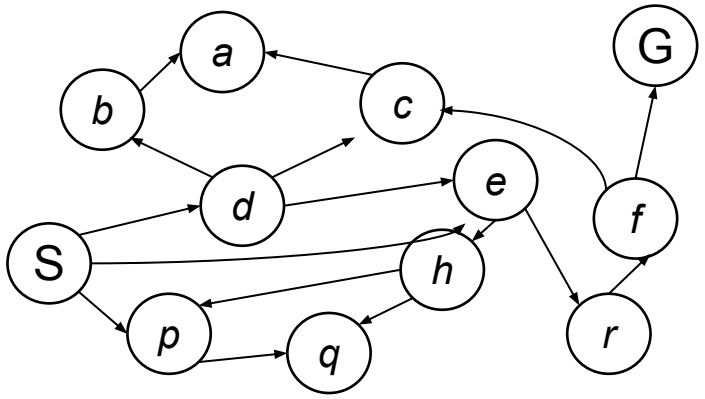


General Tree Search

```
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
```

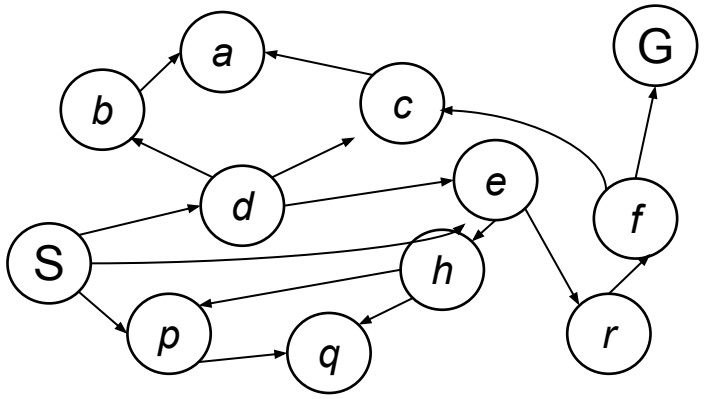
- Important ideas:
  - Fringe/Frontier
  - Expansion
  - Exploration strategy
- Main question: which fringe nodes to explore?

Example: 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
```

Example: 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
```

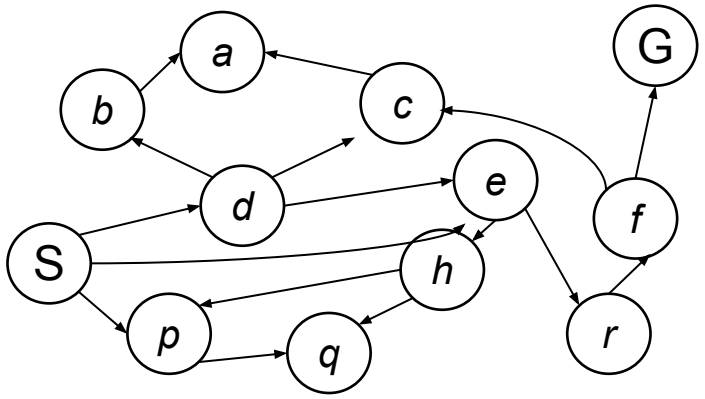
Fringe

S

Tree

S

Example: 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
```

Fringe

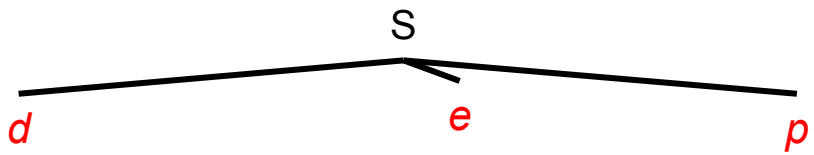
~~S~~ Does not satisfy the goal test

S -> d

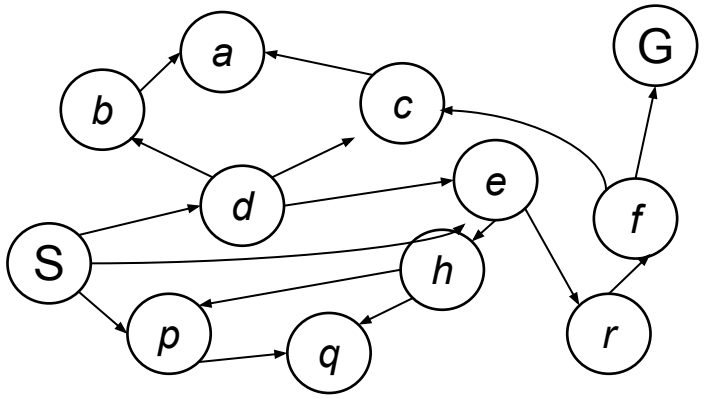
S -> e

S -> p

Tree



Example: 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
```

Fringe

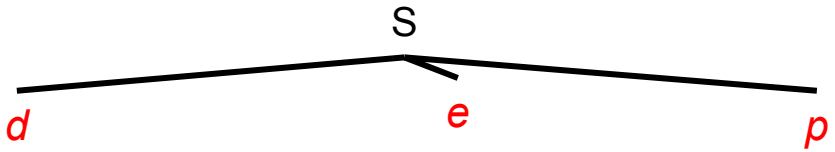
~~S~~

S -> d

S -> e

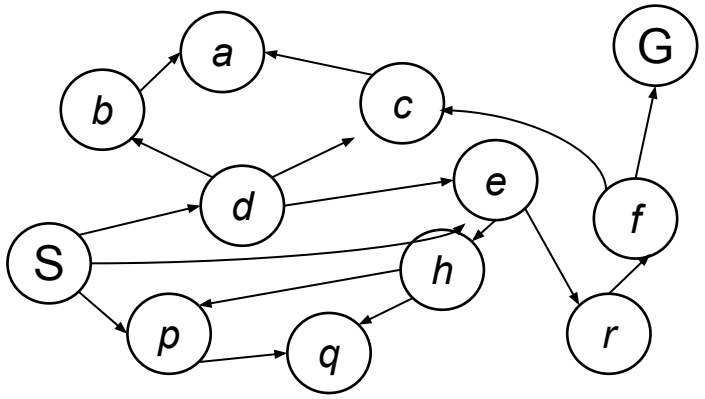
S -> p

Tree



We need some strategy to pick a node from the fringe. For now, we'll pick 'randomly'.

Example: 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
```

Fringe

~~S~~

S -> d

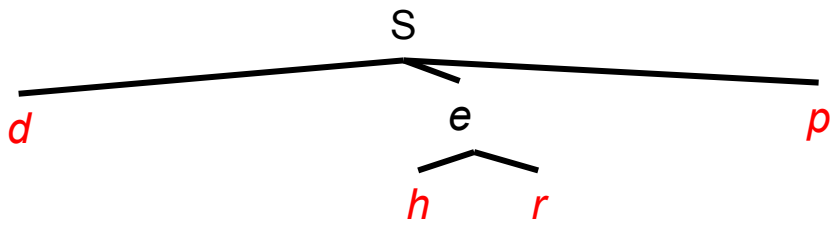
~~S -> e~~ Does not satisfy the goal test

S -> p

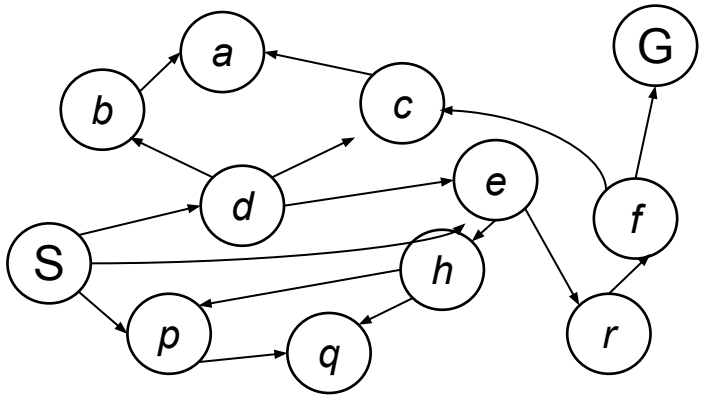
S -> e -> h

S -> e -> r

Tree



Example: 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
```

Fringe

~~S~~

S -> d

~~S -> e~~

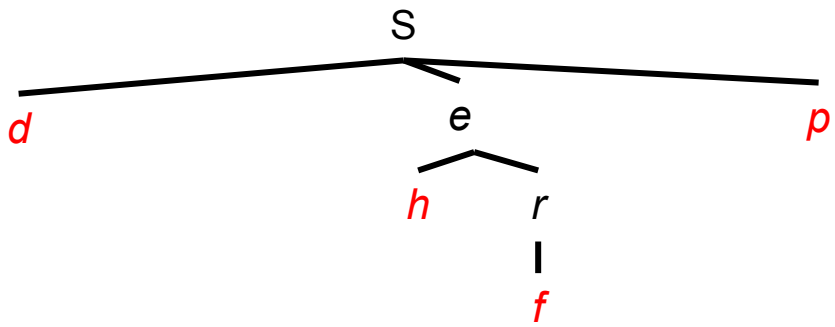
S -> p

S -> e -> h

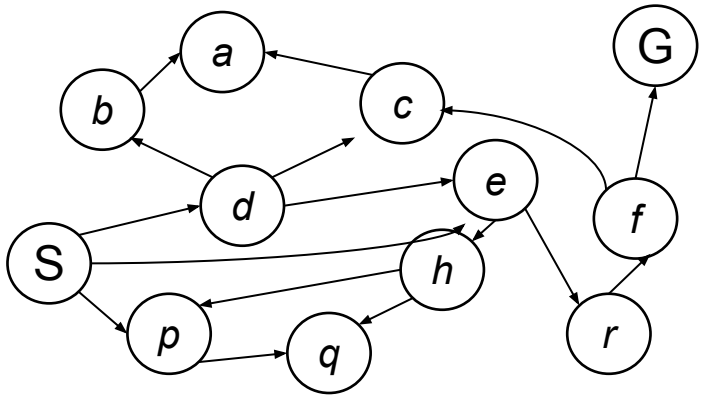
~~S -> e -> r~~ Does not satisfy the goal test

S -> e -> r -> f

Tree



Example: 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
```

Fringe

~~S~~

S -> d

S -> p

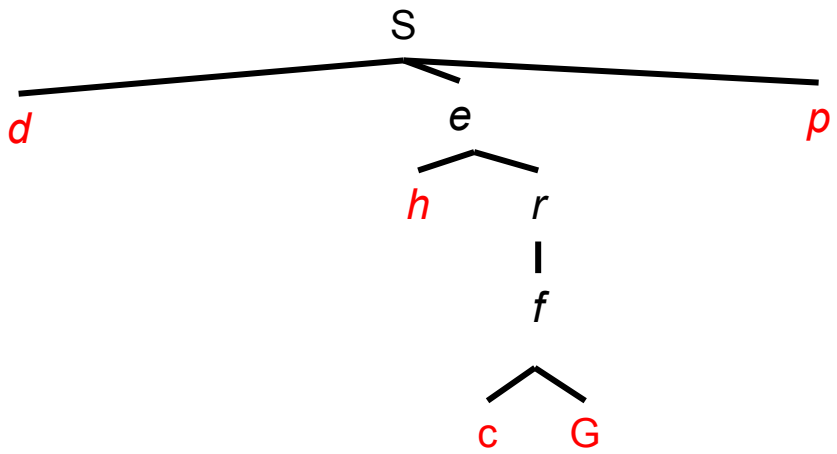
S -> e -> h

~~S -> e -> r -> f~~ Does not satisfy the goal test

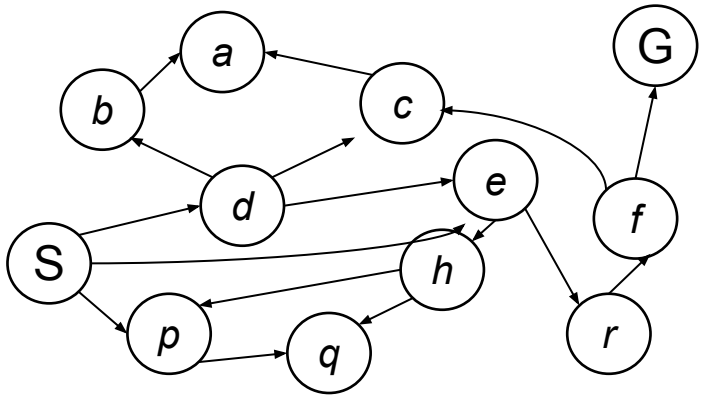
S -> e -> r -> f -> c

S -> e -> r -> f -> G

Tree



Example: 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
```

Fringe

~~S~~

S -> d

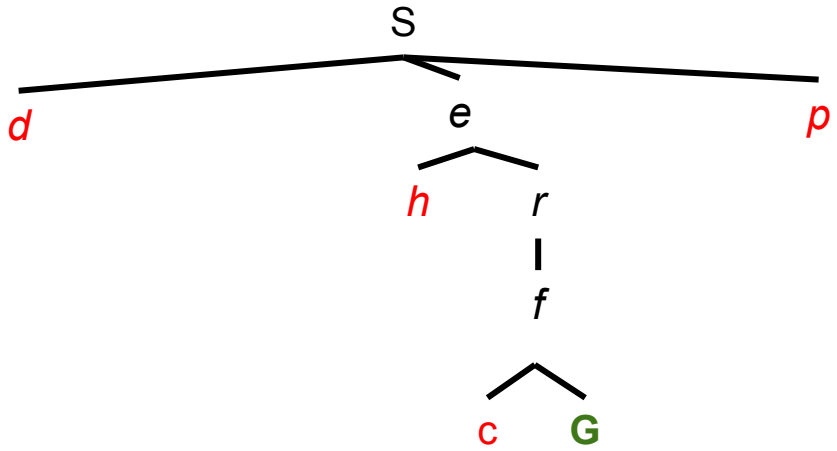
S -> p

S -> e -> h

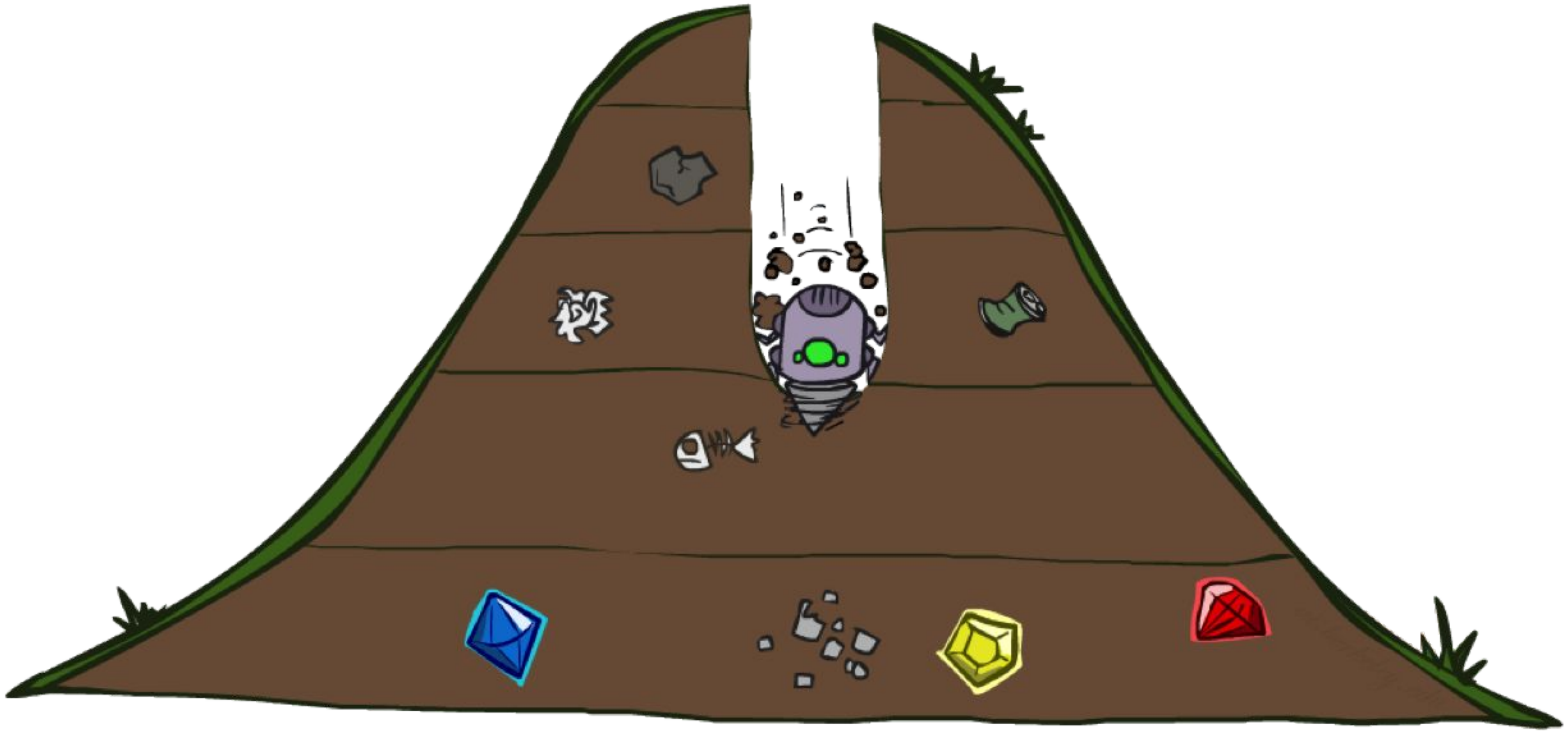
S -> e -> r -> f -> c

S -> e -> r -> f -> G Satisfies the goal test!

Tree



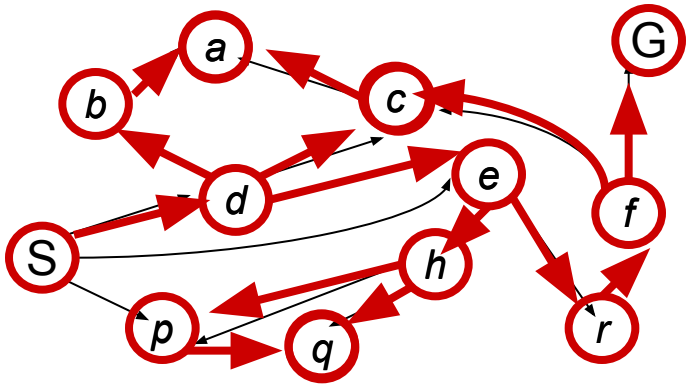
Depth-First Search

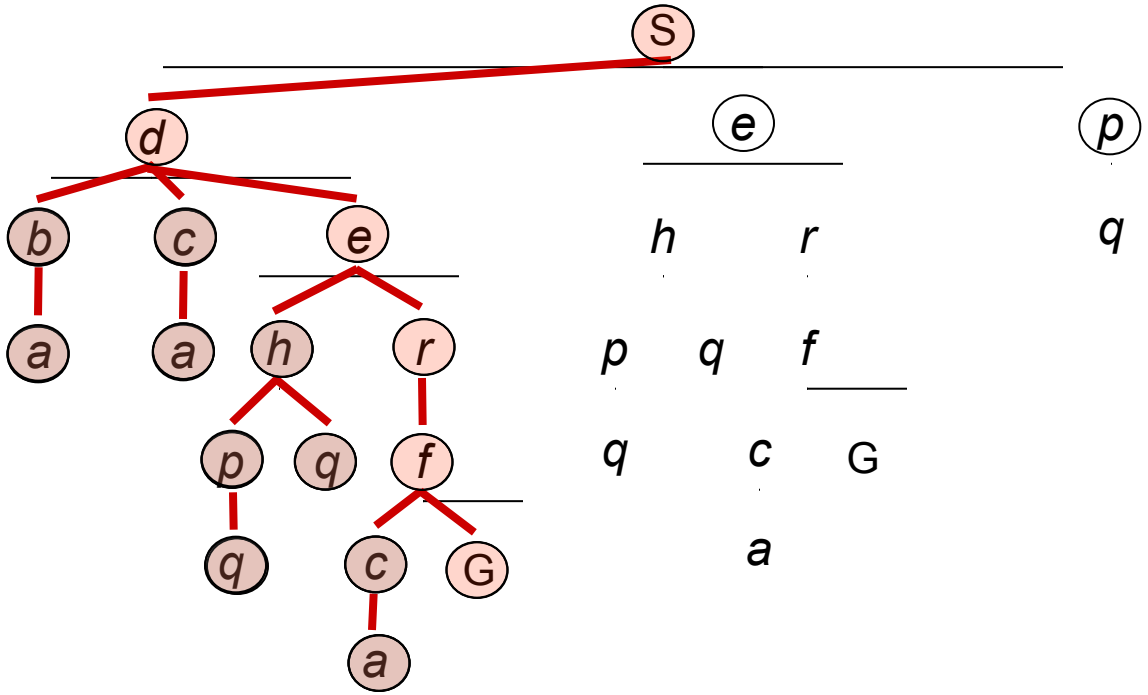


Depth-First Search

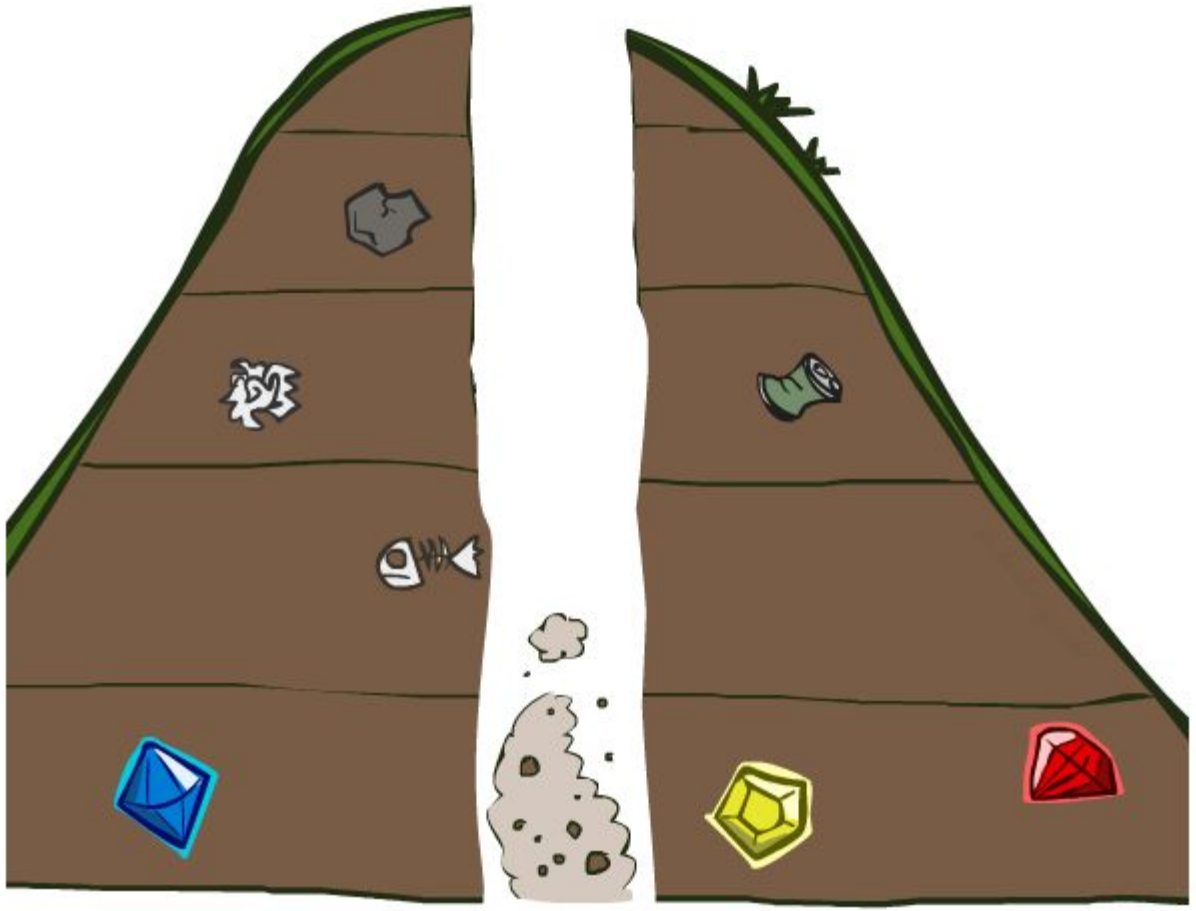
Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack





Search Algorithm Properties

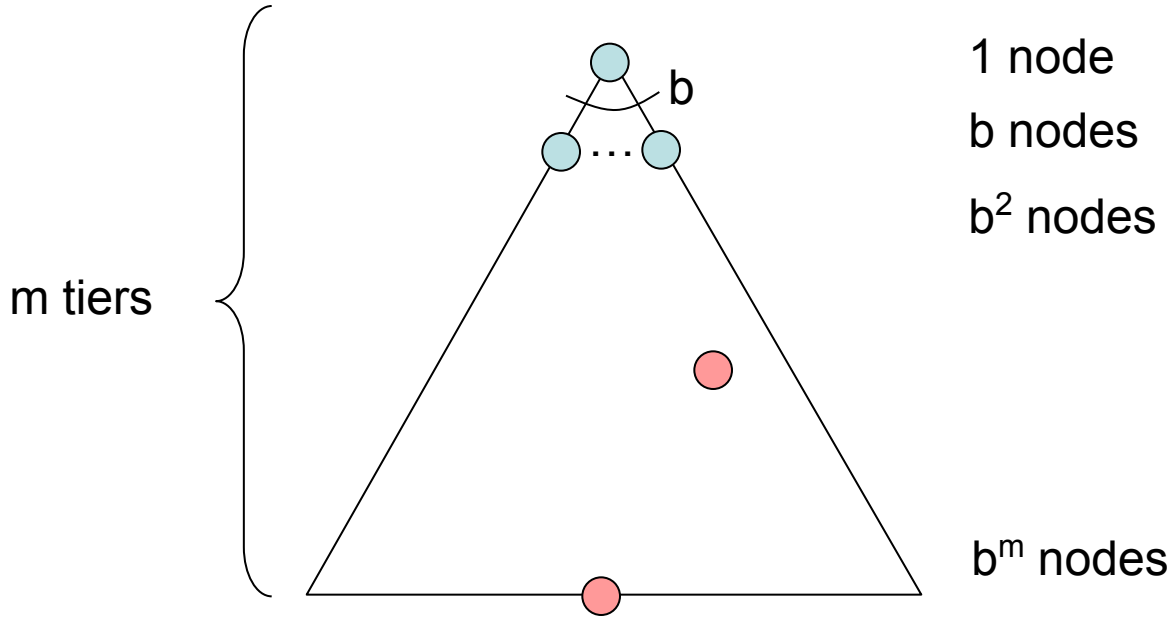


Evaluating Search Strategies

- 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?
  - $1 + b + b^2 + \ldots. b^m = O(b^m)$



Depth-First Search (DFS) Properties

Time complexity: what nodes does DFS expand?

- Some left prefix of the tree.
- Could process the whole tree!
- If m is finite, takes time $O(b^m)$

Space complexity: what is the fringe size?

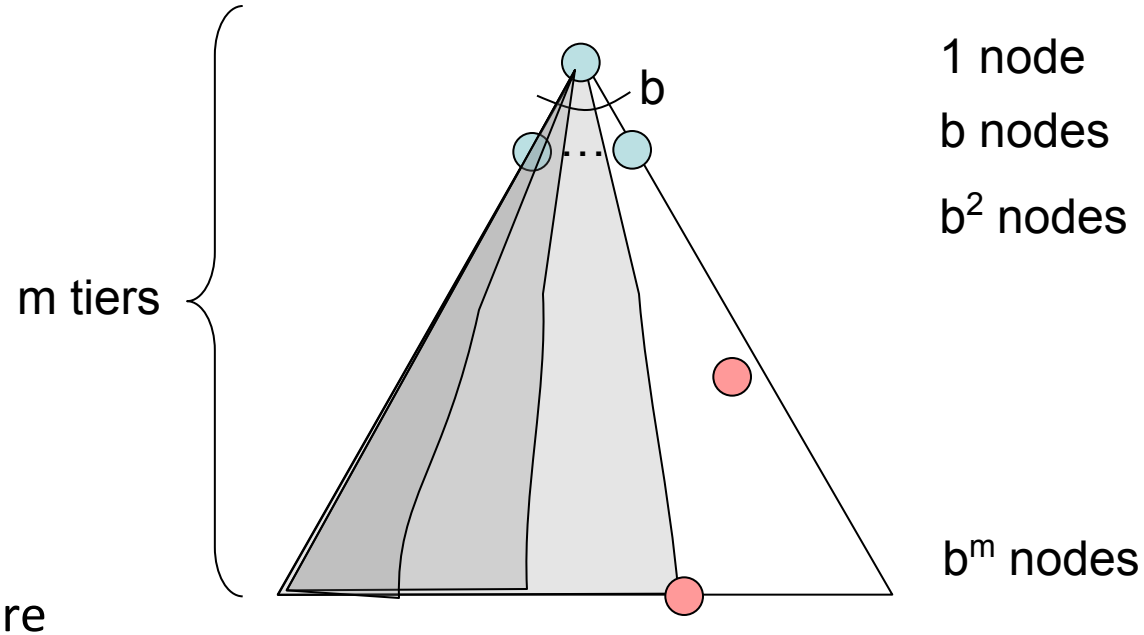
- Only has siblings on path to root, so O(bm)

Is it complete?

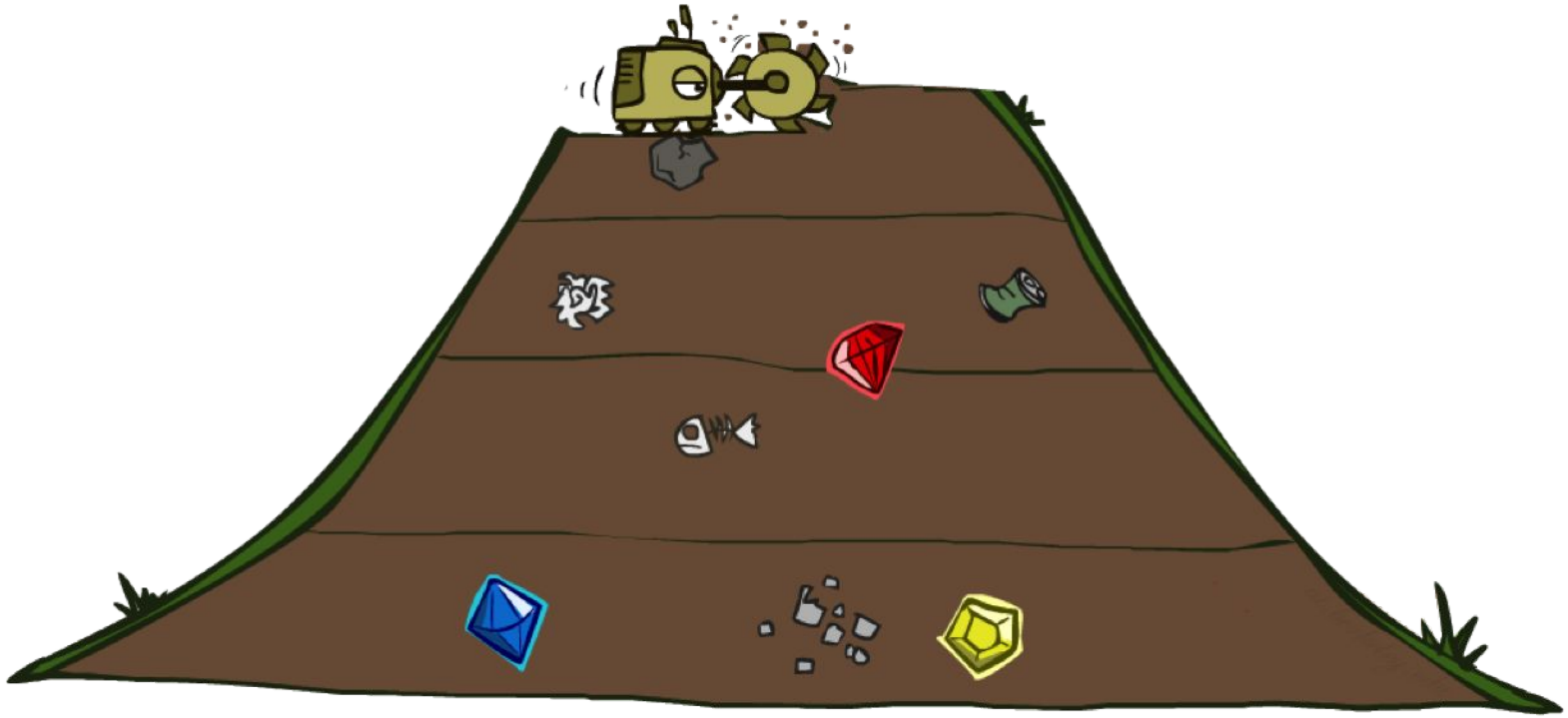
- m could be infinite, so only if we prevent cycles (more later)

Is it optimal?

- No, it finds the "leftmost" solution, regardless of depth or cost



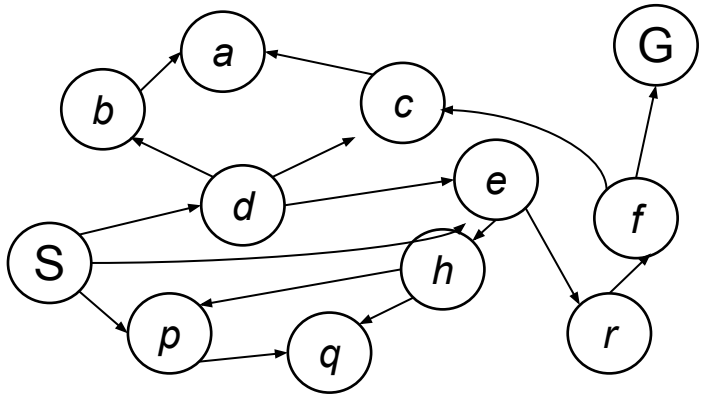
Breadth-First Search

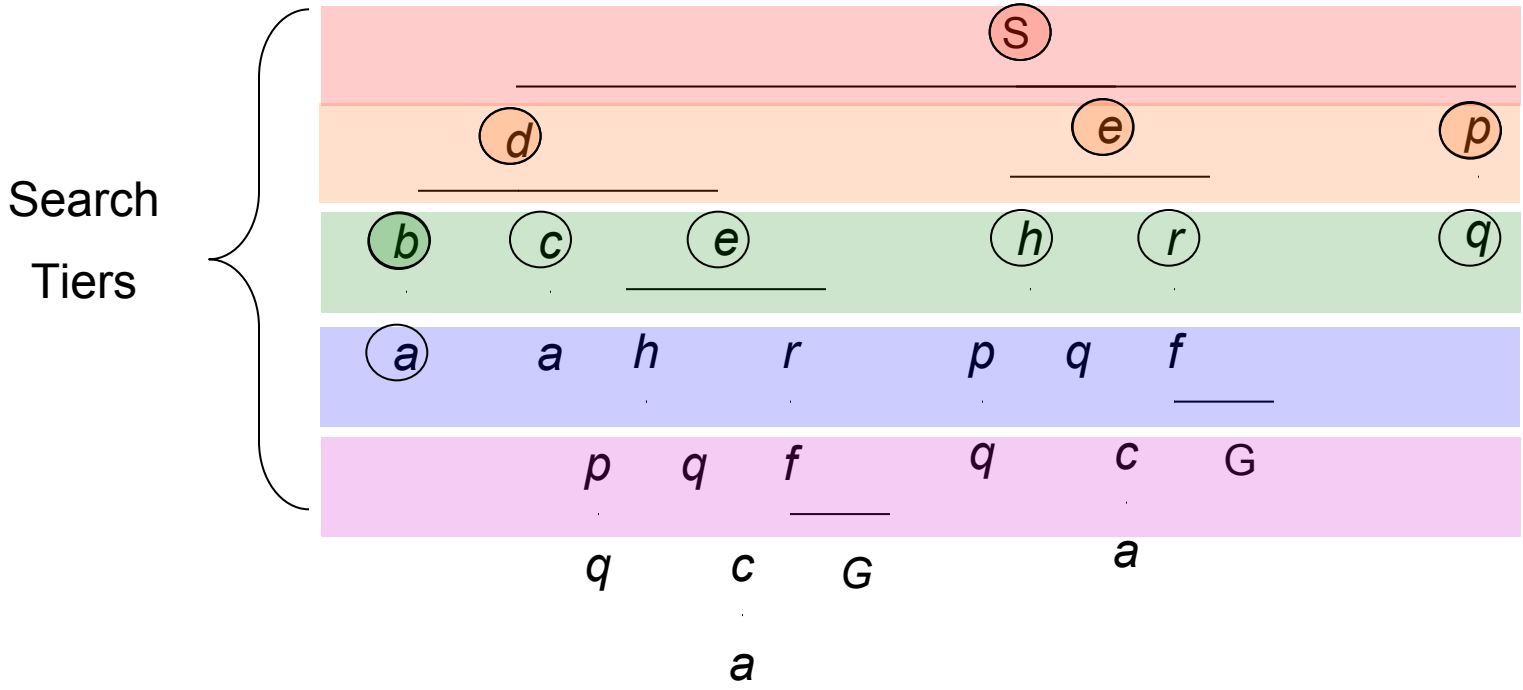


Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue

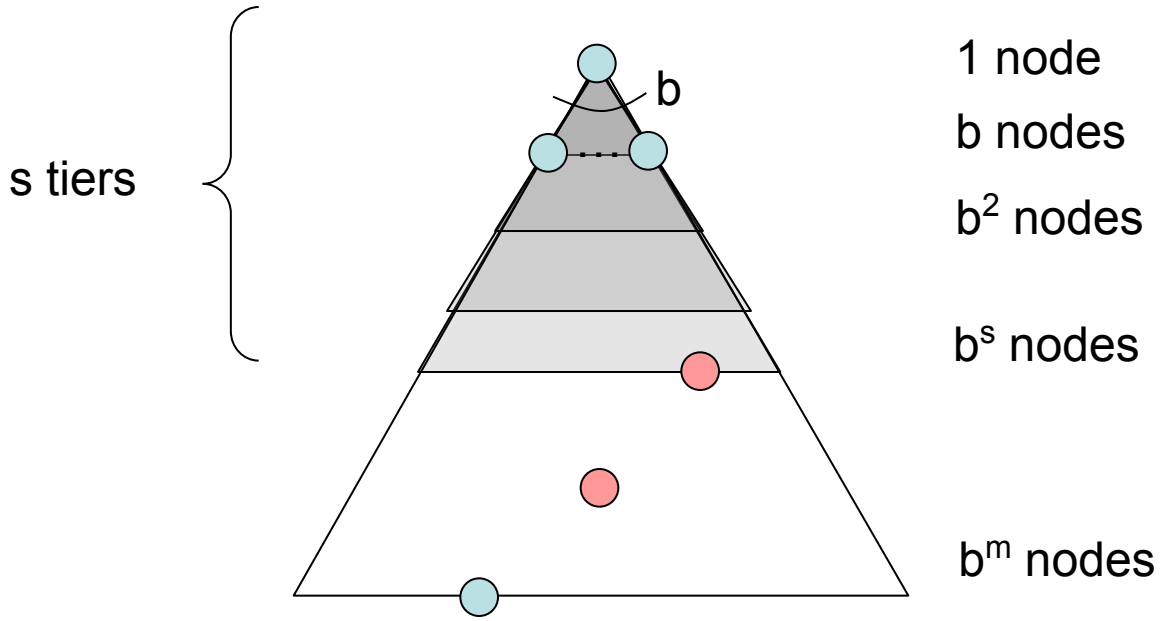




Breadth-First Search (BFS) Properties

Time complexity: what nodes does BFS expand?

- Processes all nodes above shallowest solution
- Let depth of shallowest solution be s
- Search takes time $O(b^s)$



Space complexity: fringe space

- Has roughly the last tier, so $O(b^s)$

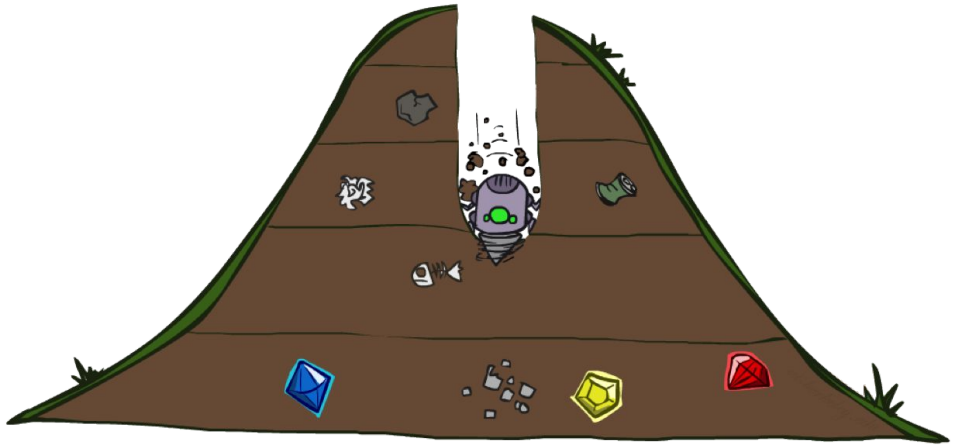
Is it complete?

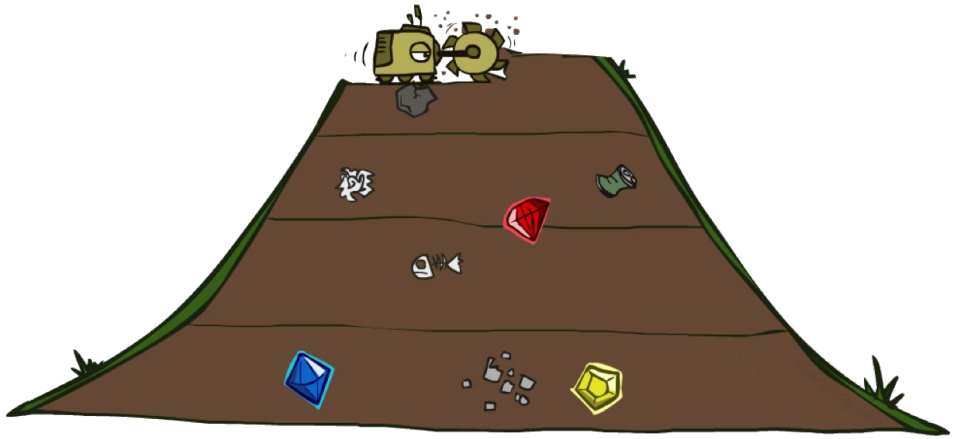
- s must be finite if a solution exists, so yes!

Is it optimal?

- Only if costs are all 1 (more on costs later)

DFS vs BFS





When will BFS outperform DFS?

When will DFS outperform BFS?

Demo DFS/BFS

empty dfs / bfs videos

Demo DFS/BFS

Pacman medium maze

DFS:

Path found with total cost of 130 in 0.0 seconds

Search nodes expanded: 146

Pacman emerges victorious! Score: 380

BFS:

Path found with total cost of 68 in 0.0 seconds

Search nodes expanded: 269

Pacman emerges victorious! Score: 442

Demo DFS/BFS

Pacman big maze

DFS:

Path found with total cost of 210 in 0 sec

Search nodes expanded: 390

Pacman emerges victorious! Score: 300

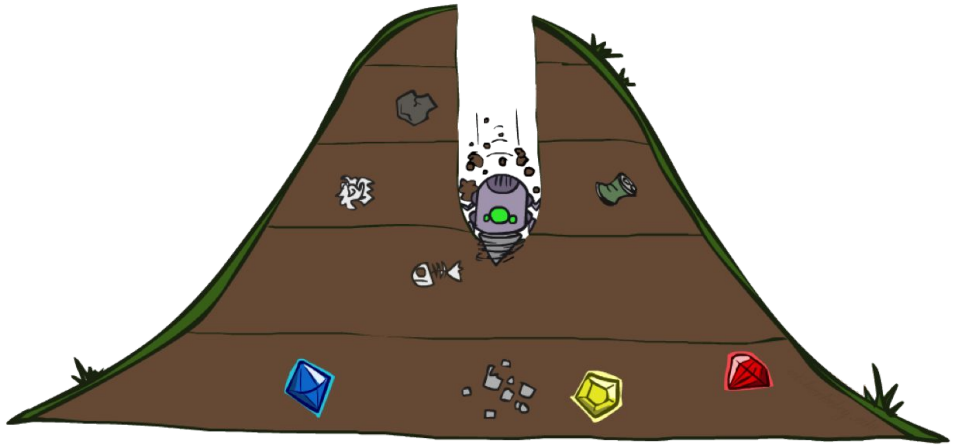
BFS:

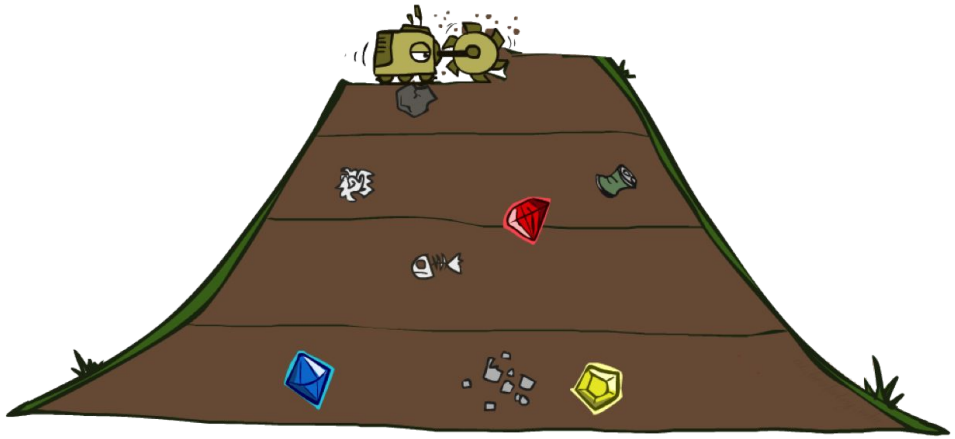
Path found with total cost of 210 in 0.1 seconds

Search nodes expanded: 620

Pacman emerges victorious! Score: 300

DFS vs BFS





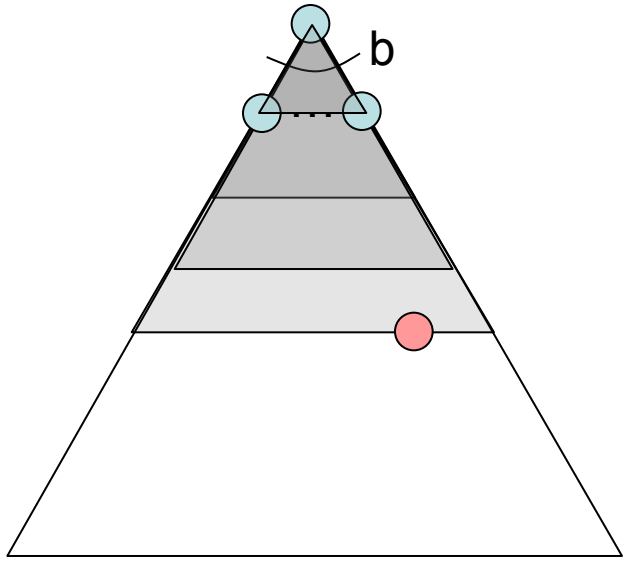
When will BFS outperform DFS?

When will DFS outperform BFS?

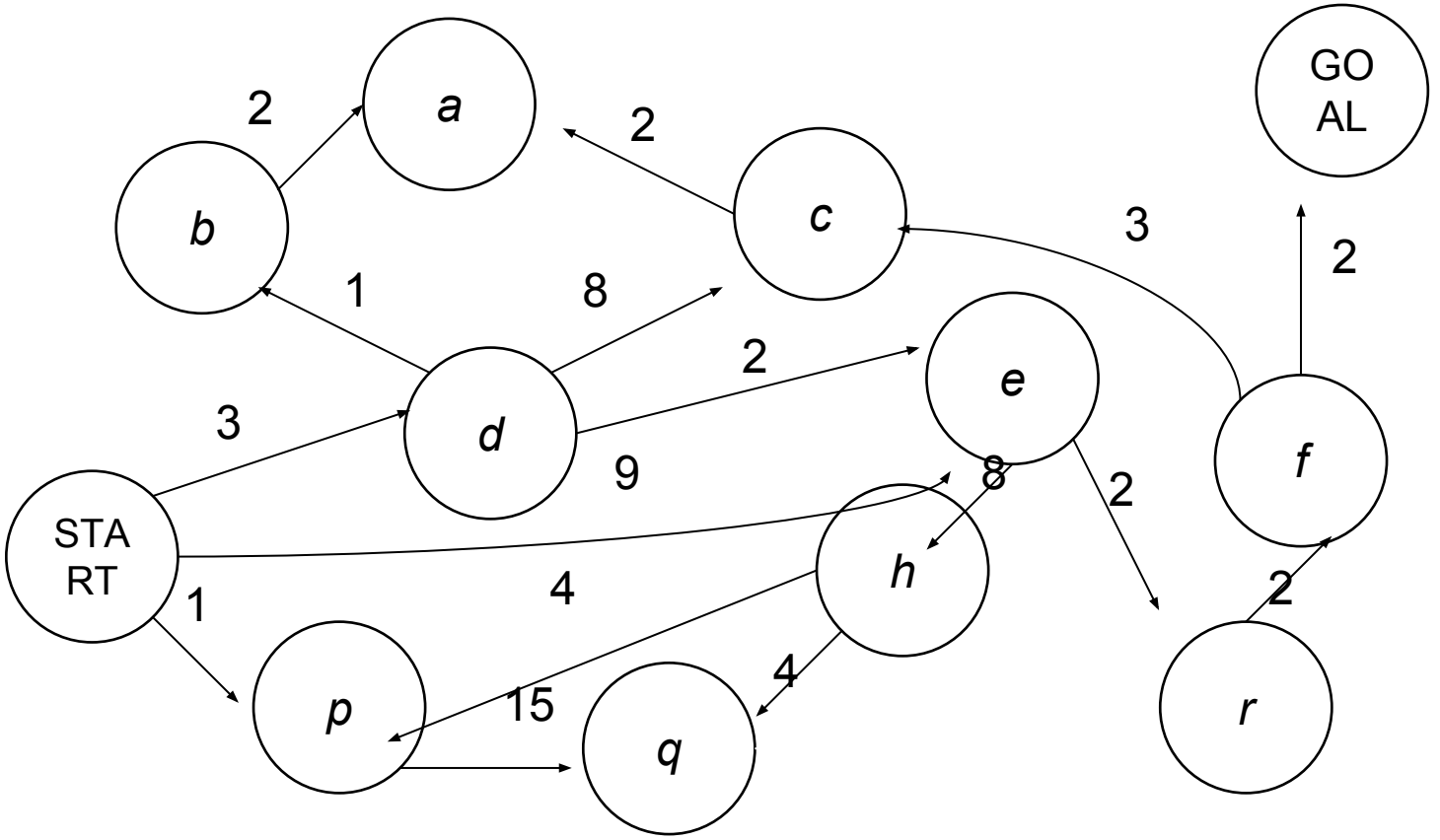
Considerations: depth of solution, space limitations, branching factor

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!

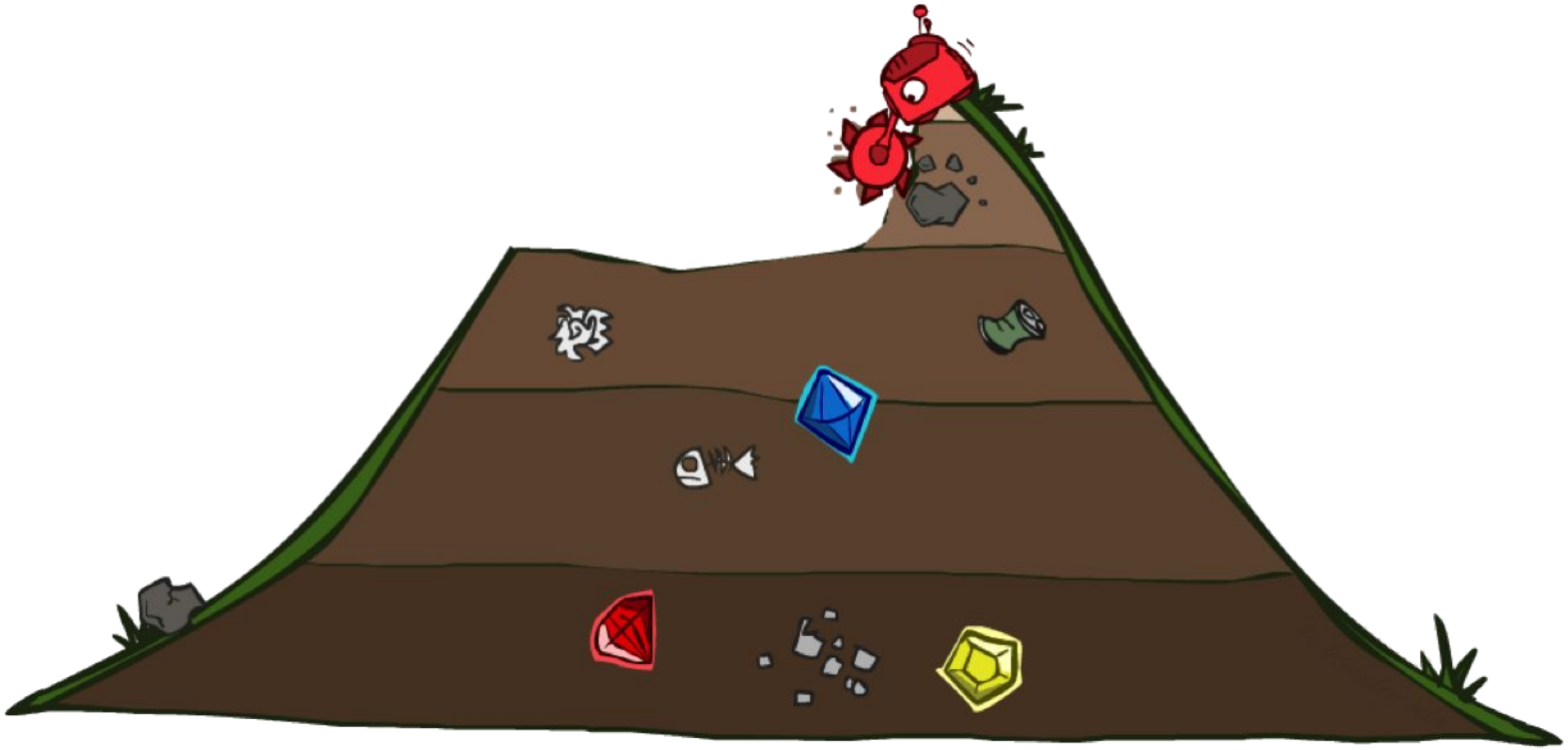


Cost-Sensitive Search



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.

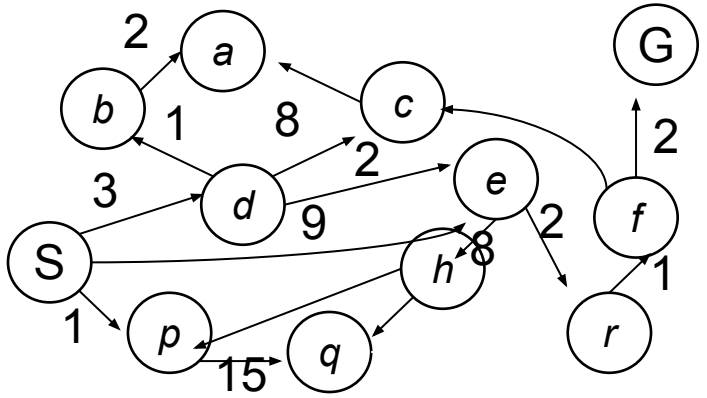
Uniform Cost Search

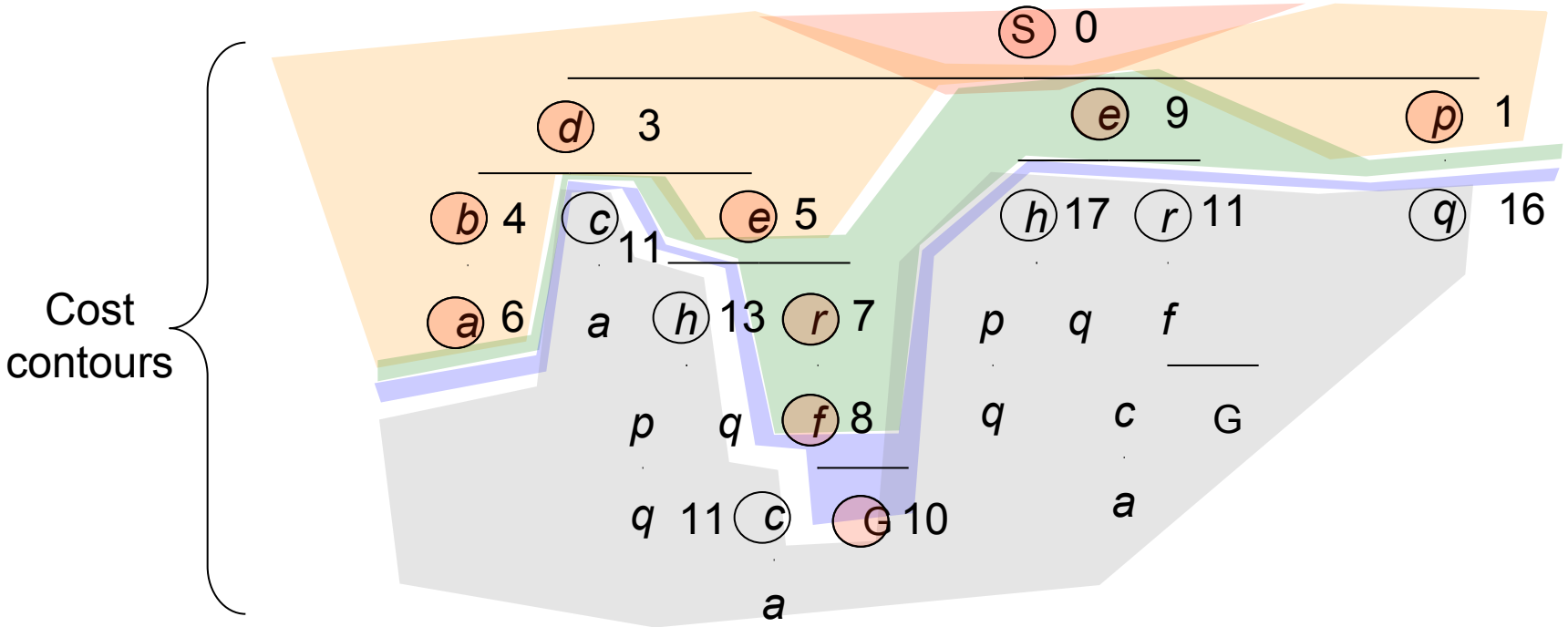


Uniform Cost Search

Strategy: expand a cheapest node first:

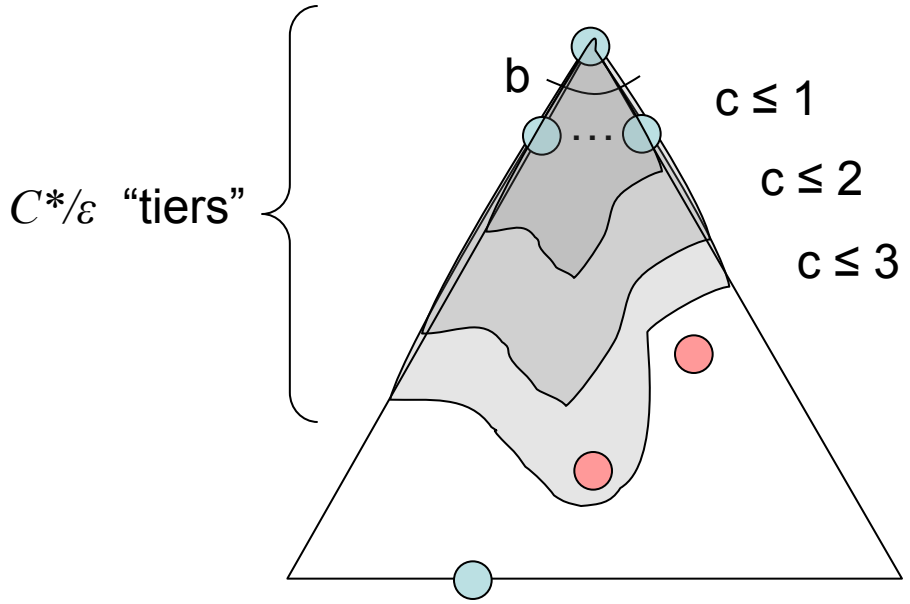
Fringe is a priority queue (priority: cumulative cost)





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)



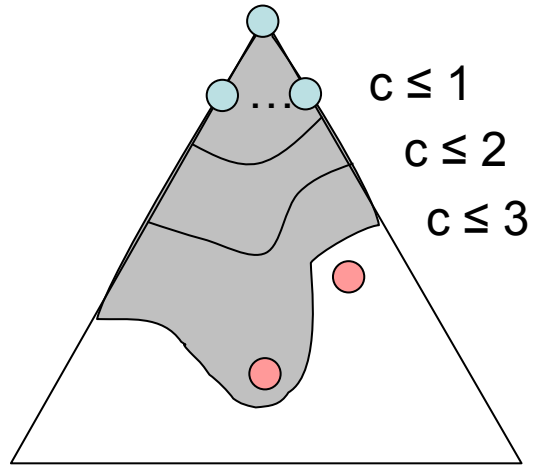
- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$

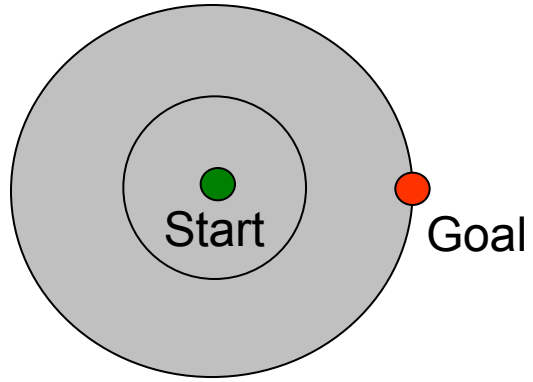
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!

- Is it optimal?
  - Yes! (Proof next lecture via A*)

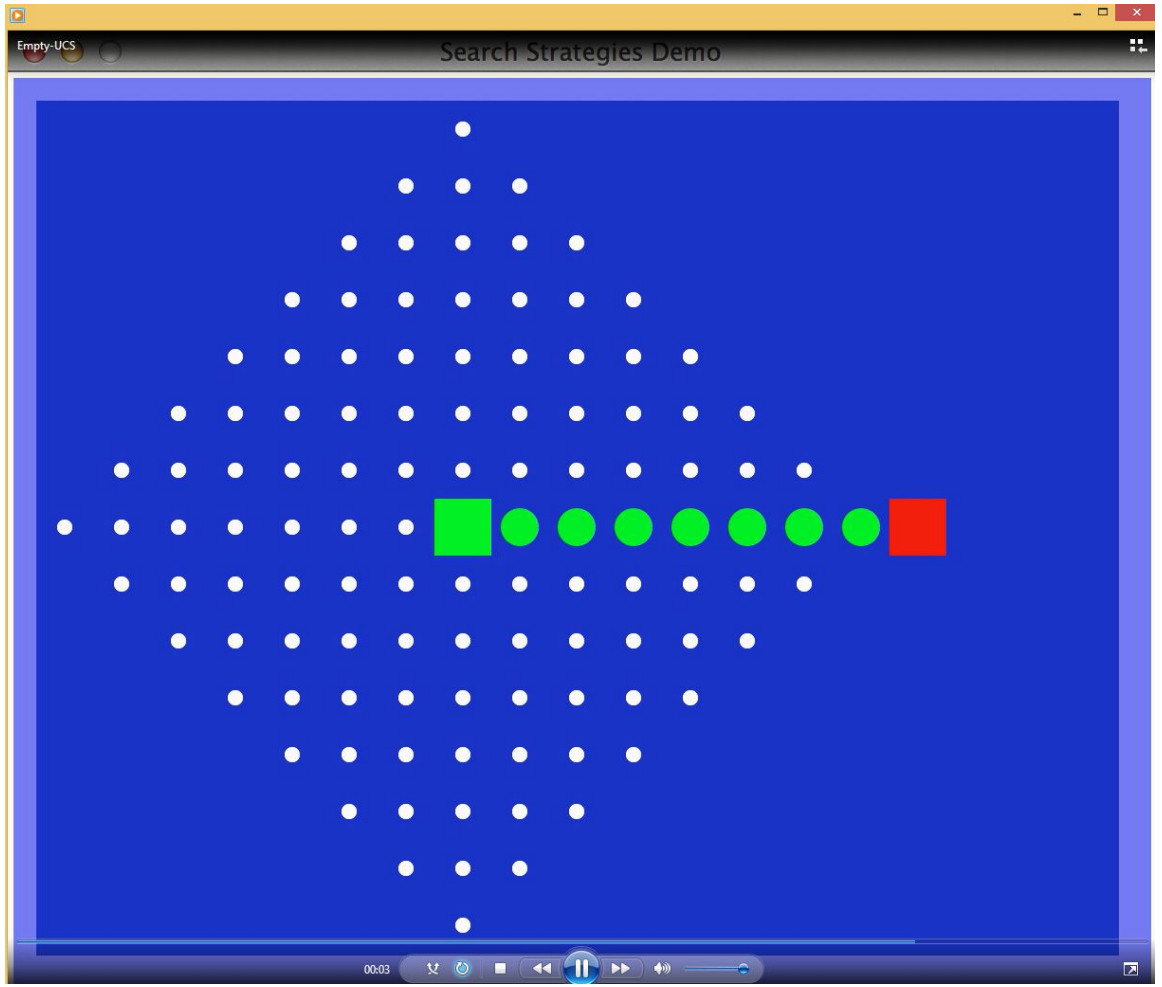
Uniform Cost Issues

- 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 next week!





Video of Demo Empty UCS

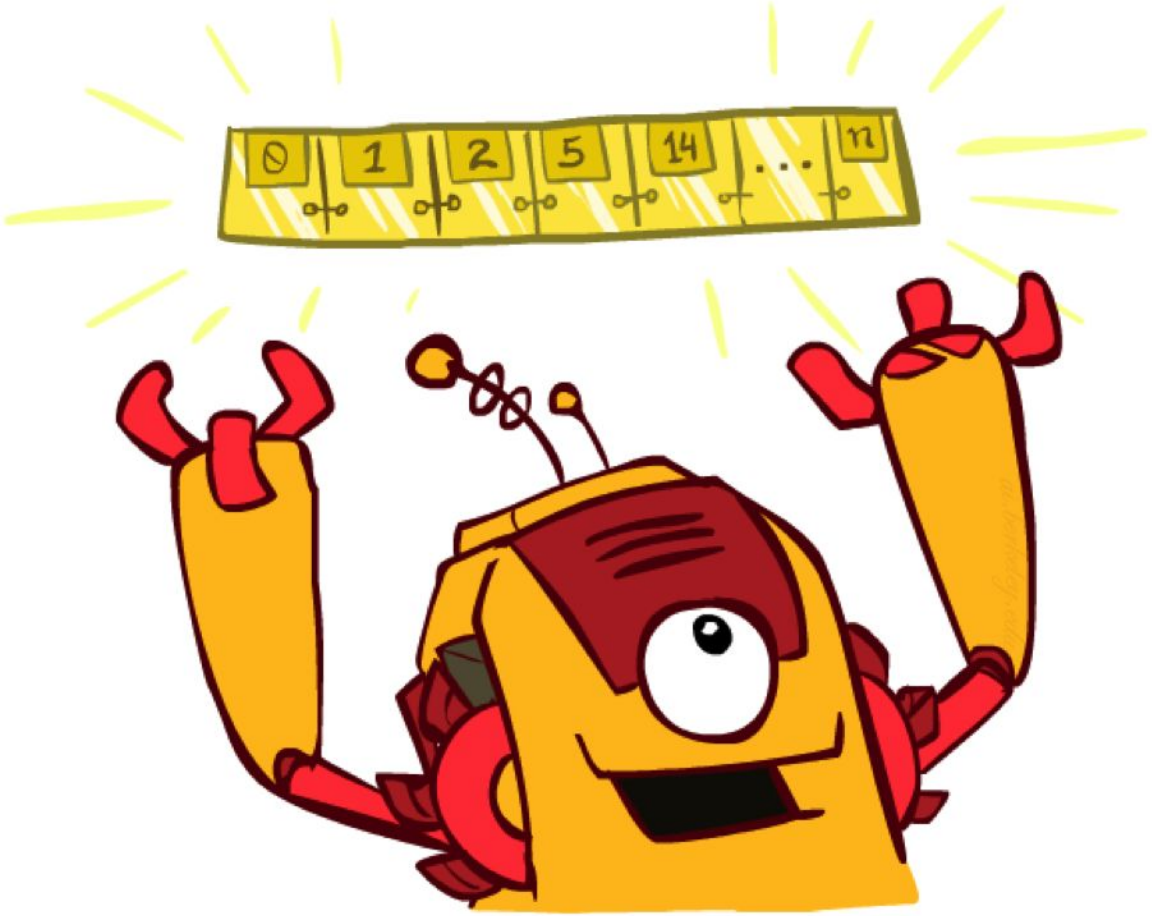


Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS?

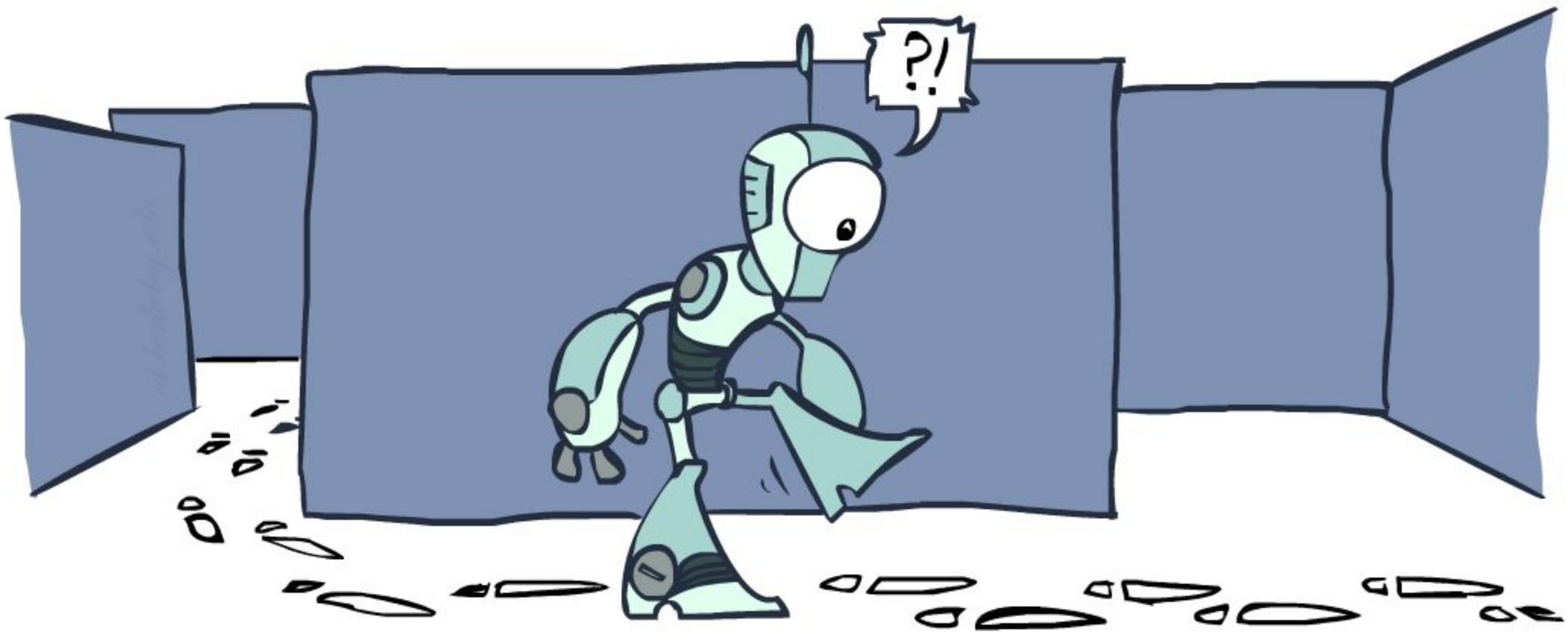
A. DFS
B. BFS
C. UCS

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.



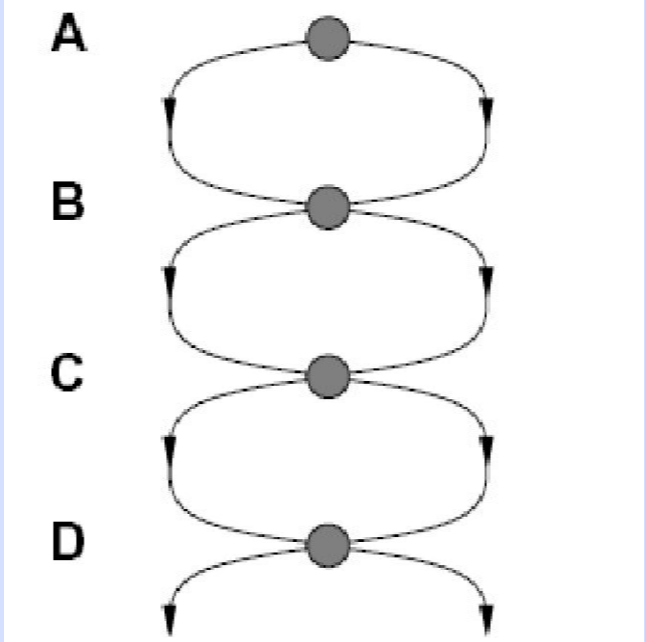
Graph Search



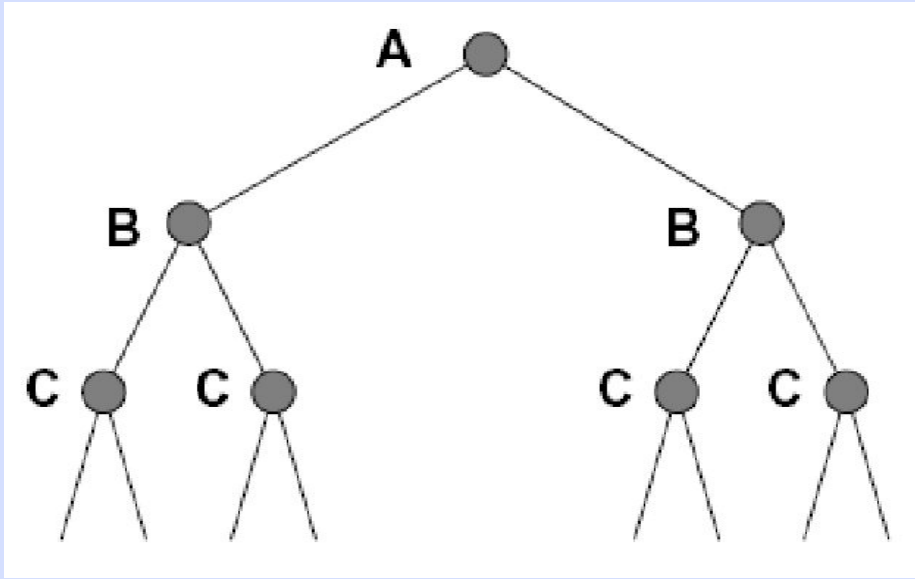
Tree Search: Extra Work!

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

State Graph

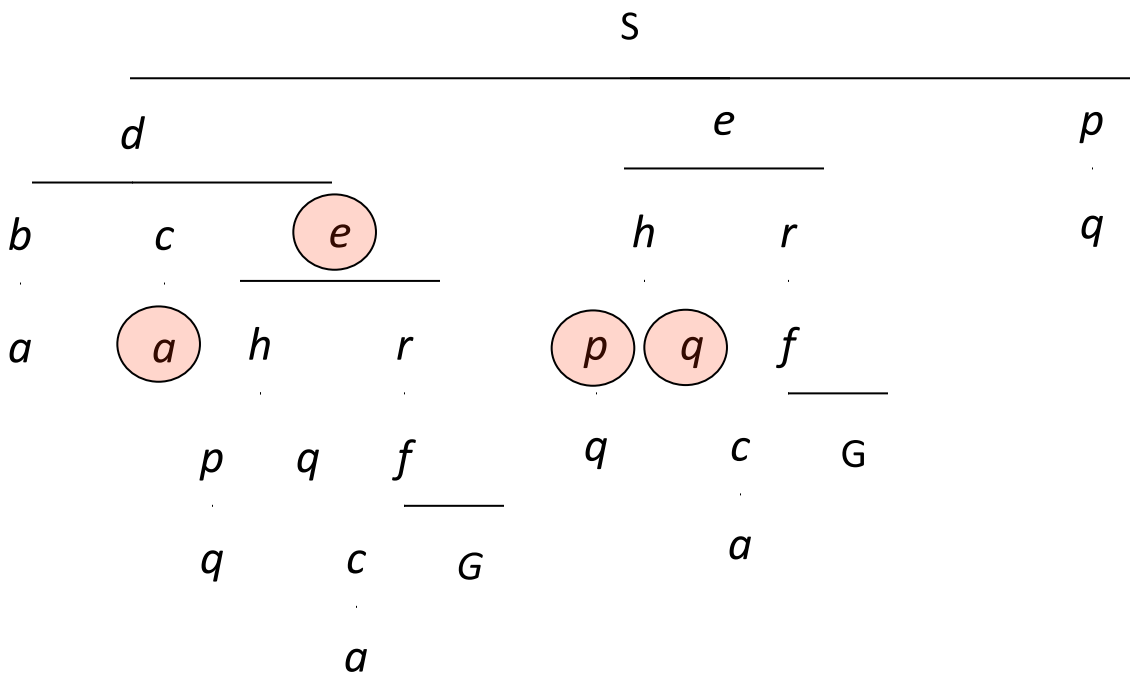


Search Tree



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/explored set
- Important: store the closed set as a set, not a list

Tree Search Pseudo-Code

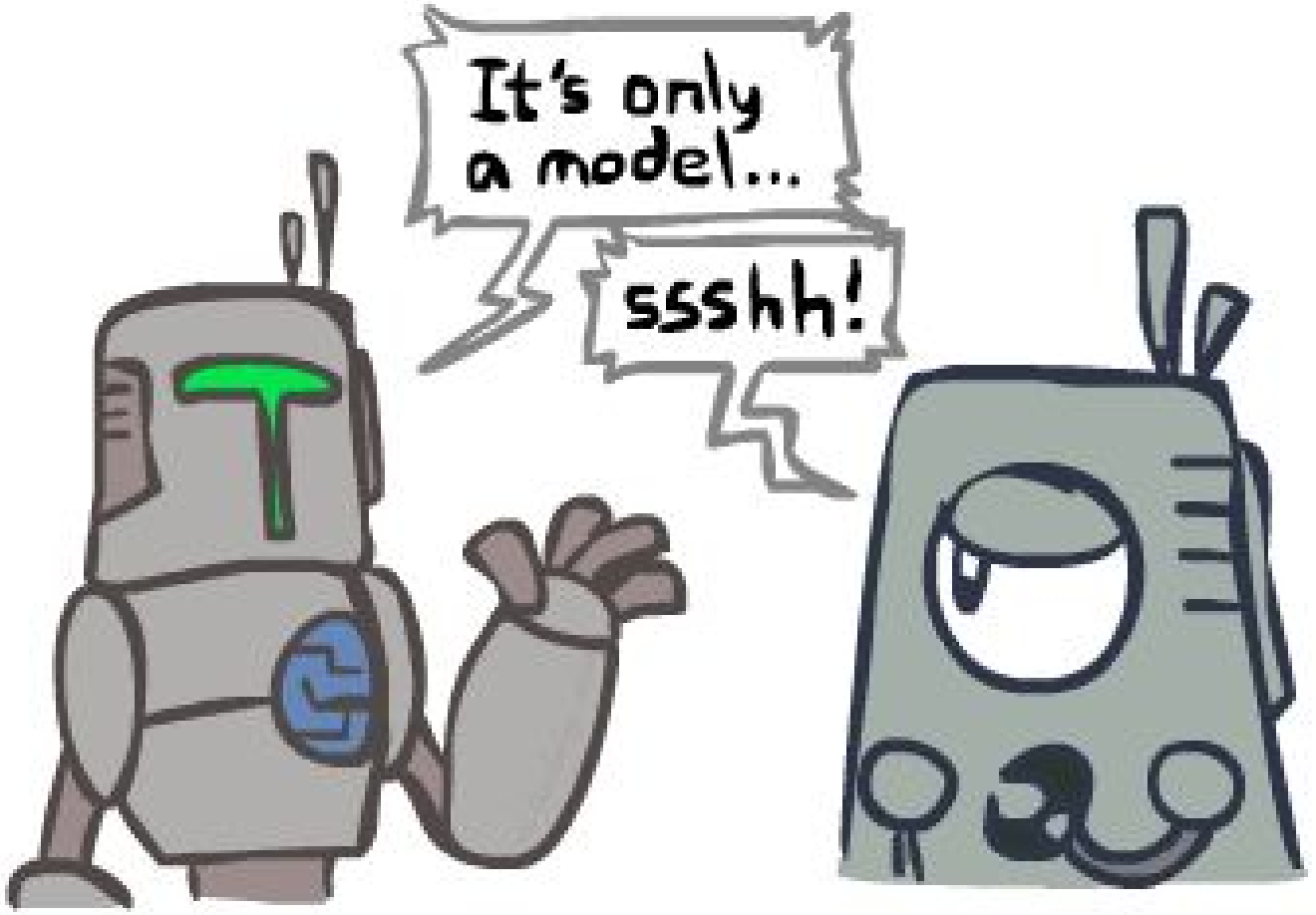
```
function TREE-SEARCH(problem, fringe) return a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        for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

Search and Models

- Search operates over models of the world
  - The agent doesn't actually try all the plans out in the real world!
  - It is all "in simulation"
  - Your search is only as good as your models…



Hints for Homework 2

- Graph search is better than tree search.
- Implement your explored/closed set as a set!
- Nodes are conceptually paths - Look into the Node class in util.
- Use the Queue, Stack and PriorityQueue classes in util.