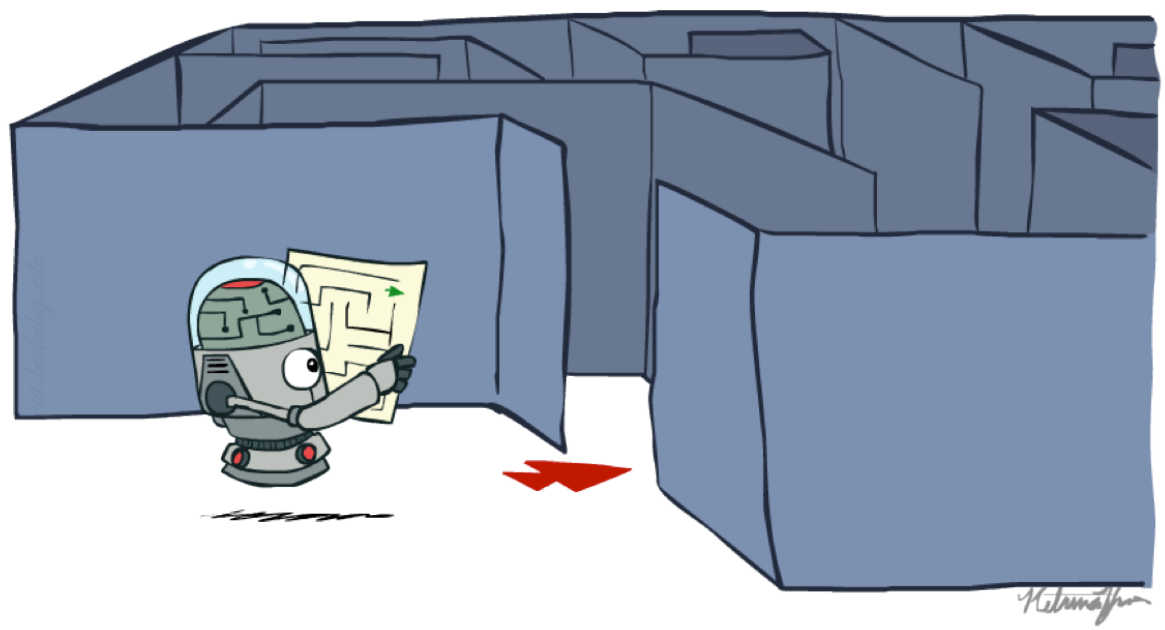
Uninformed Search

Lecture 4

What are common search strategies that operate only given the search problem formalism? How do they compare?

Agenda

- A quick refresher
- DFS, BFS, ID-DFS, UCS
- Unification!



Search Problem Formalism

Defined via the following components:

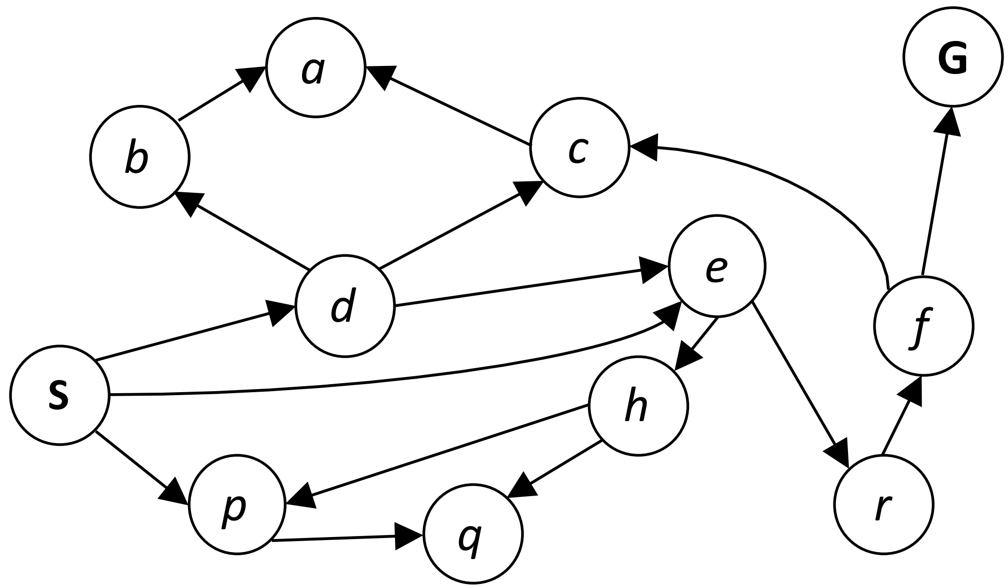
- The **initial state** the agent starts in
- A **successor/transition function**
  - $S(x)$ = {action+cost->state}
- A **goal test**, which determines whether a given state is a goal state
- A **path cost** that assigns a numeric cost to each path

A **solution** is a sequence of actions leading from initial state to a goal state. (**Optimal** = lowest path cost.)

Together the initial state and successor function implicitly define the **state space**, the set of all reachable states

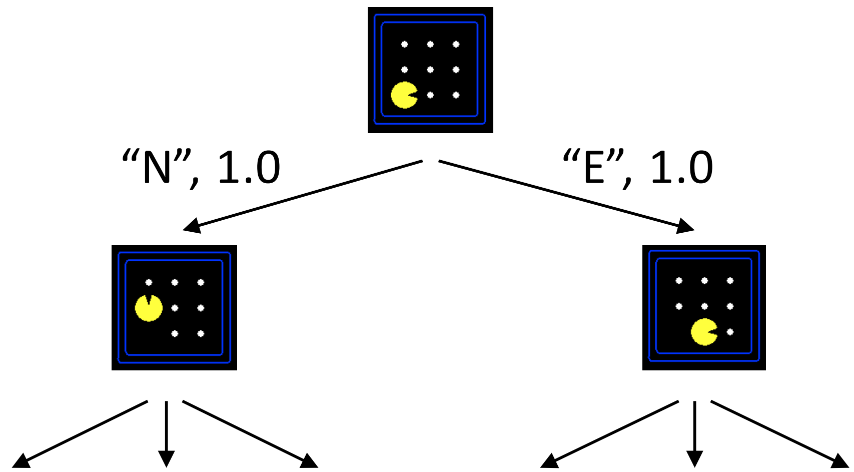
State Space Graph

- 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 node(s)
- In a search 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 Tree

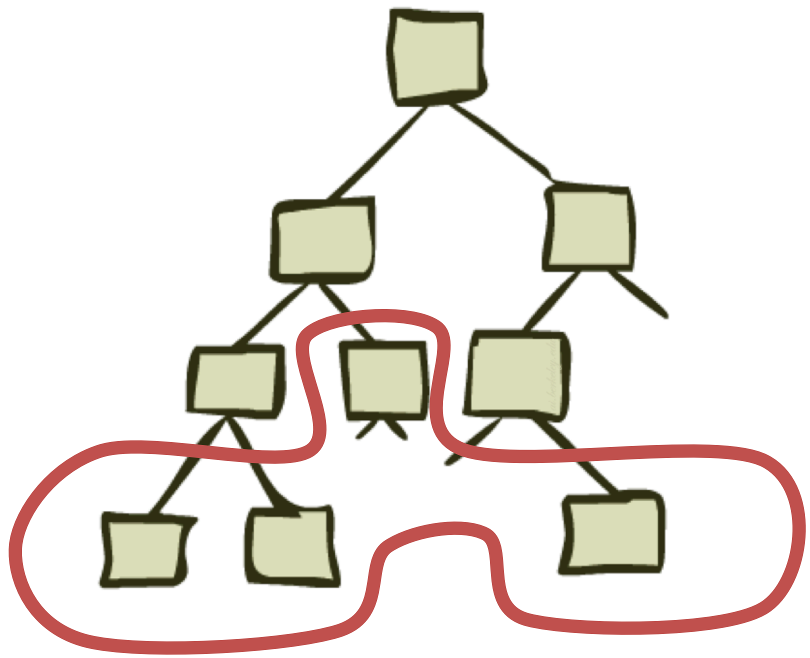
- A “what if” tree of plans and their outcomes
- The start state is the root node
- Children correspond to successors
- Nodes show states, but correspond to PLANS that achieve those states
- **For most problems, we can never actually build the whole tree**



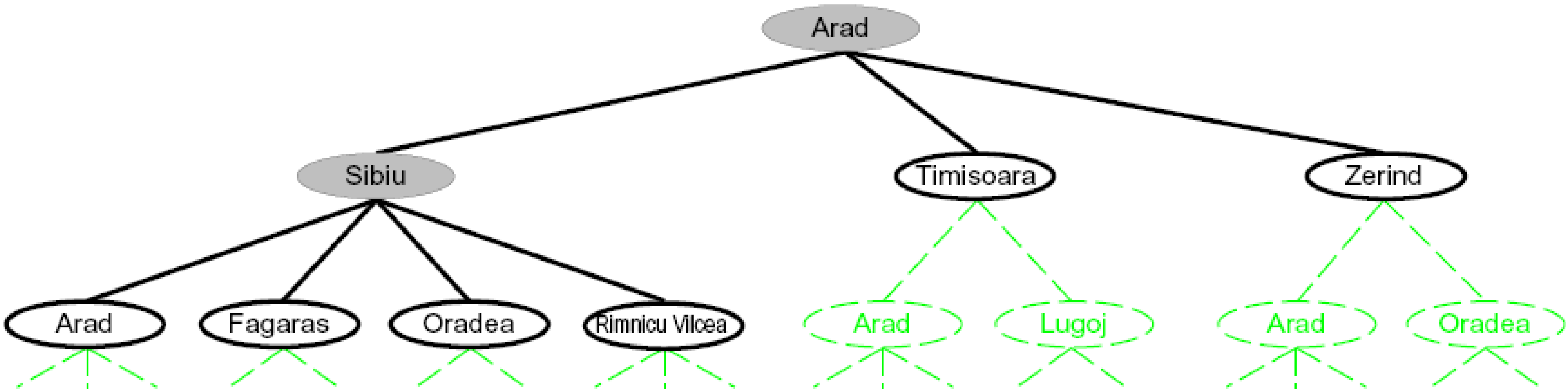
Searching for Solutions

Basic idea: incrementally build a search tree until a goal state is found

- Root = initial state
- Expand via transition function to create new nodes
- Nodes that haven't been expanded are **leaf nodes** and form the **frontier** (**open list**)
- Different **search strategies** (next lecture) choose next node to expand (as few as possible!)
- Use a **closed list** to prevent expanding the same state more than once



General Algorithm




```
function GRAPH-SEARCH(problem, fringe) returns 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
          fringe ← INSERTALL(EXPAND(node, problem), fringe)
  end
```

Queue (FIFO)
Stack (LIFO)
Priority Queue

Evaluating a Search Strategy

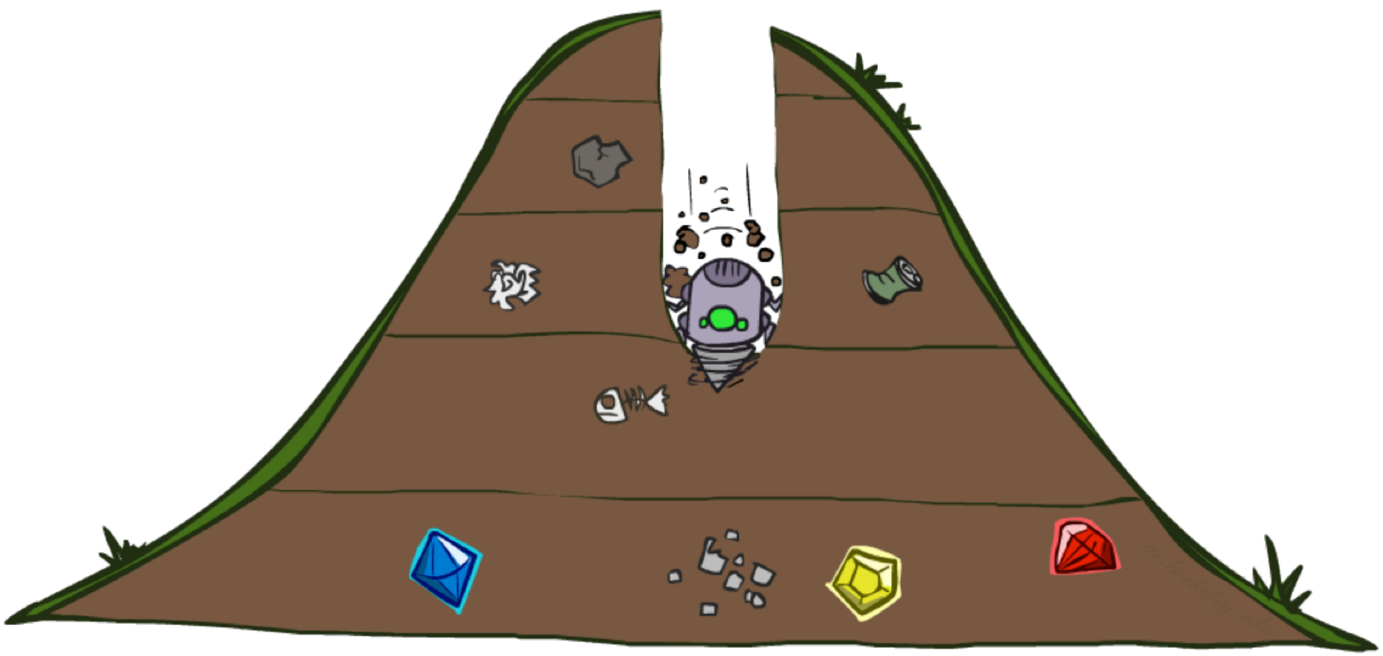
Solution

- **Completeness**: does it always find *a* solution if one exists?
- **Optimality**: does it always find a least-cost solution?

Efficiency

- **Time Complexity**: number of nodes generated/expanded
- **Space Complexity**: maximum number of nodes in memory

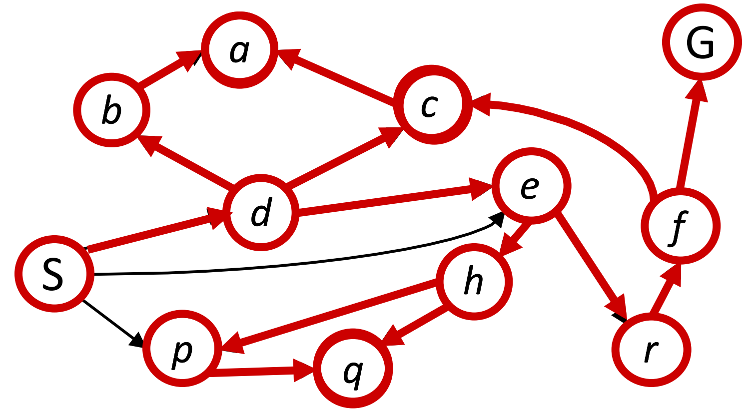
Depth-First Search (DFS)

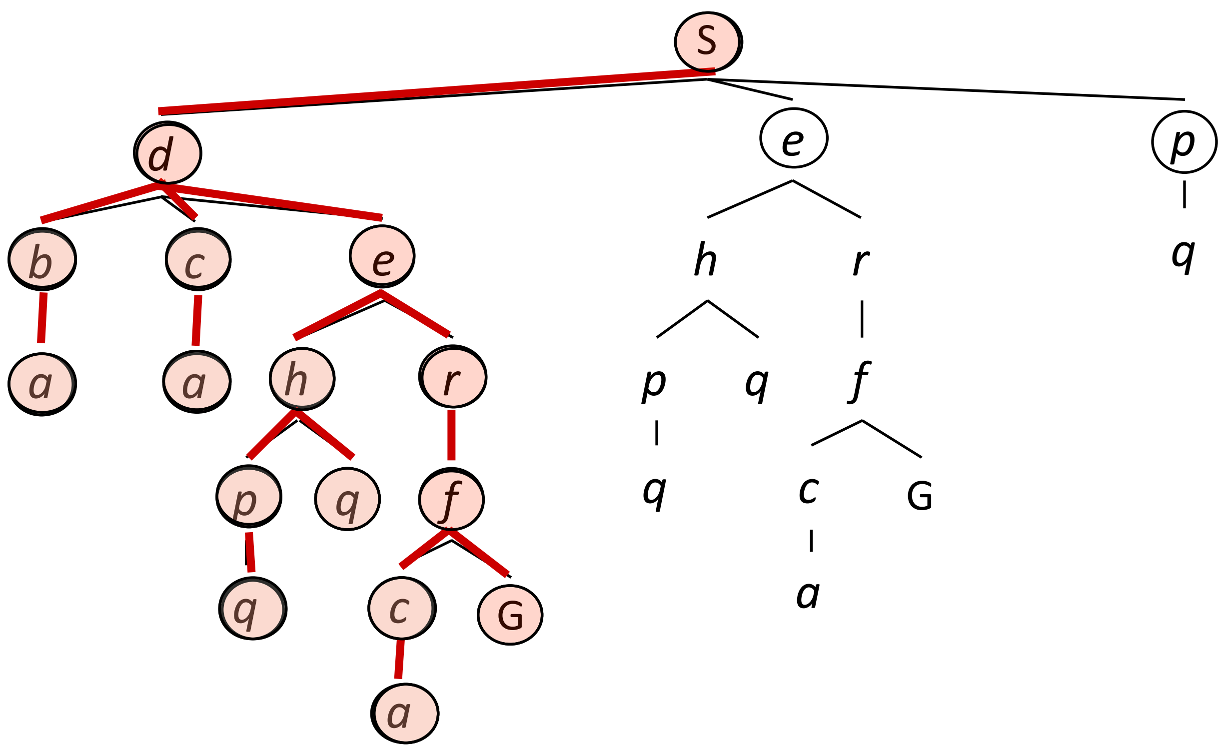


DFS Example

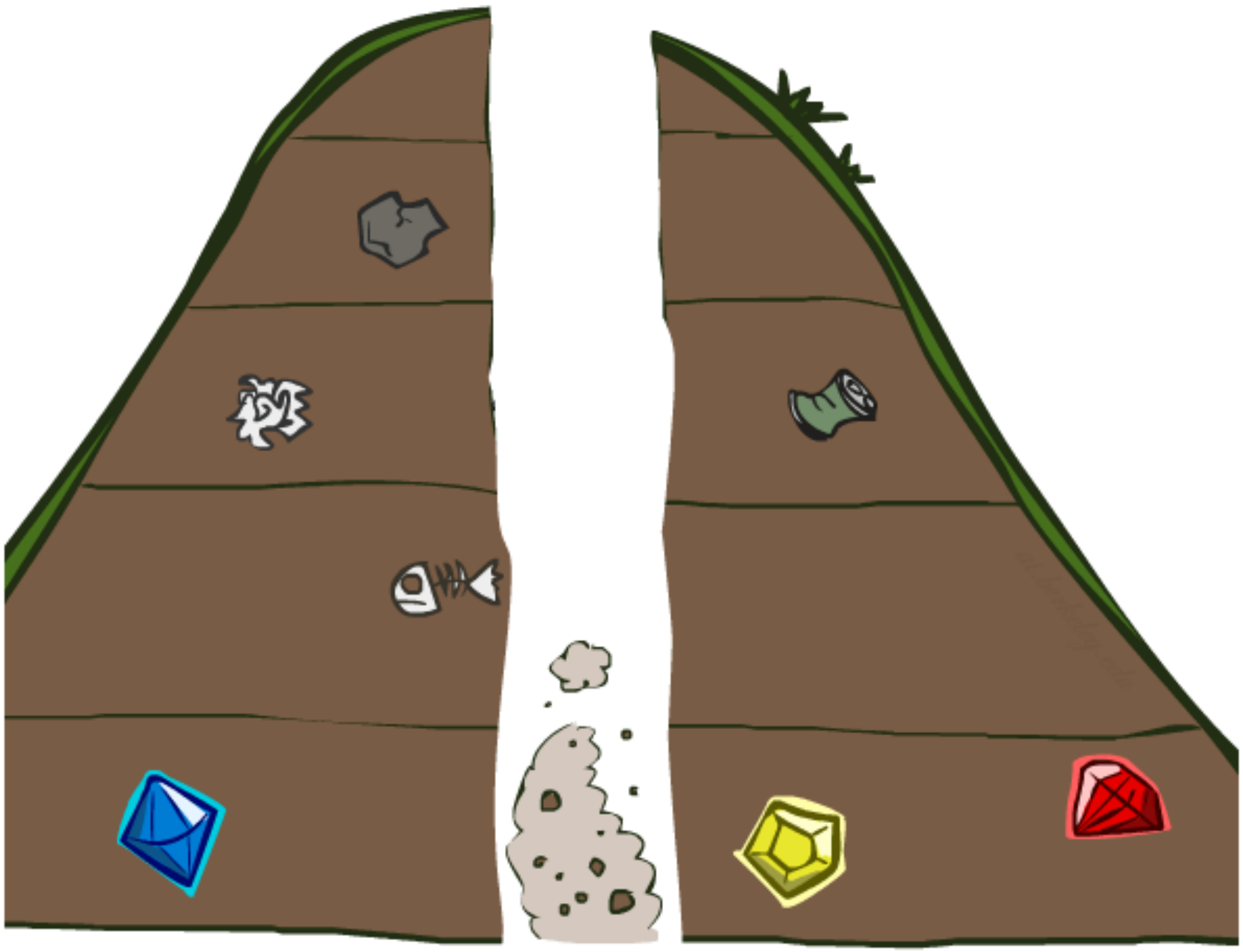
Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack





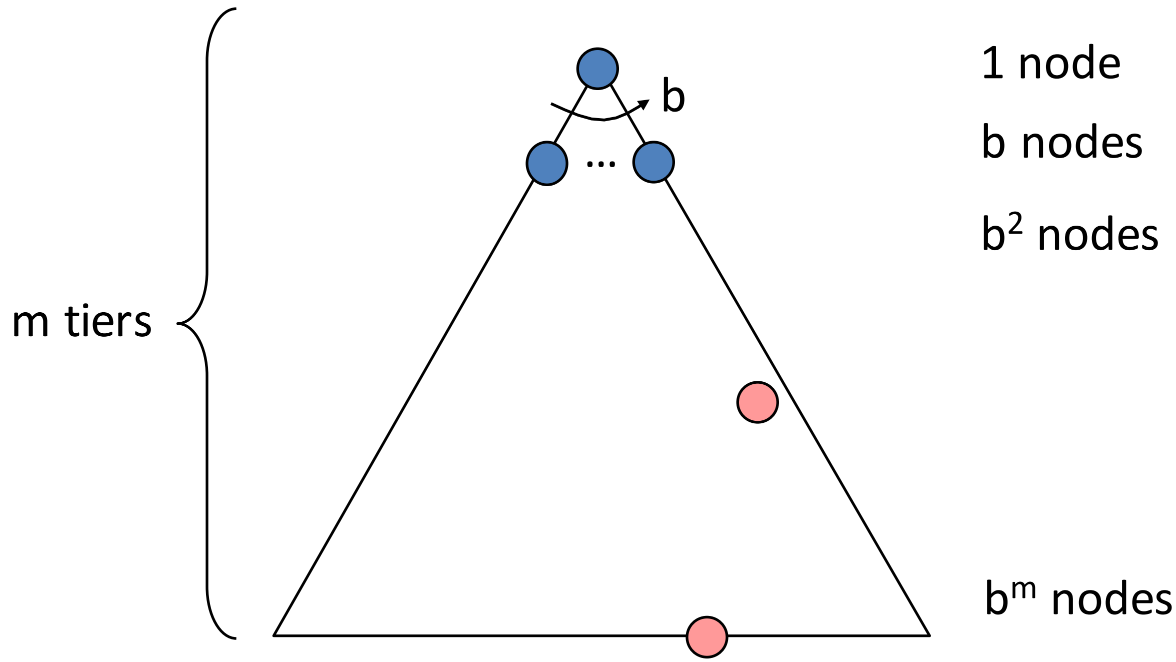
Let's Evaluate!



Search Tree

Properties

- **B**ranching factor
- **M**aximum depth
- Solutions at various depths



Number of nodes in the tree?

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

DFS Evaluation

<u>Time</u>

- Expands left
  - Could be whole tree!
- Assuming finite depth, $O(b^m)$

<u>Space</u>

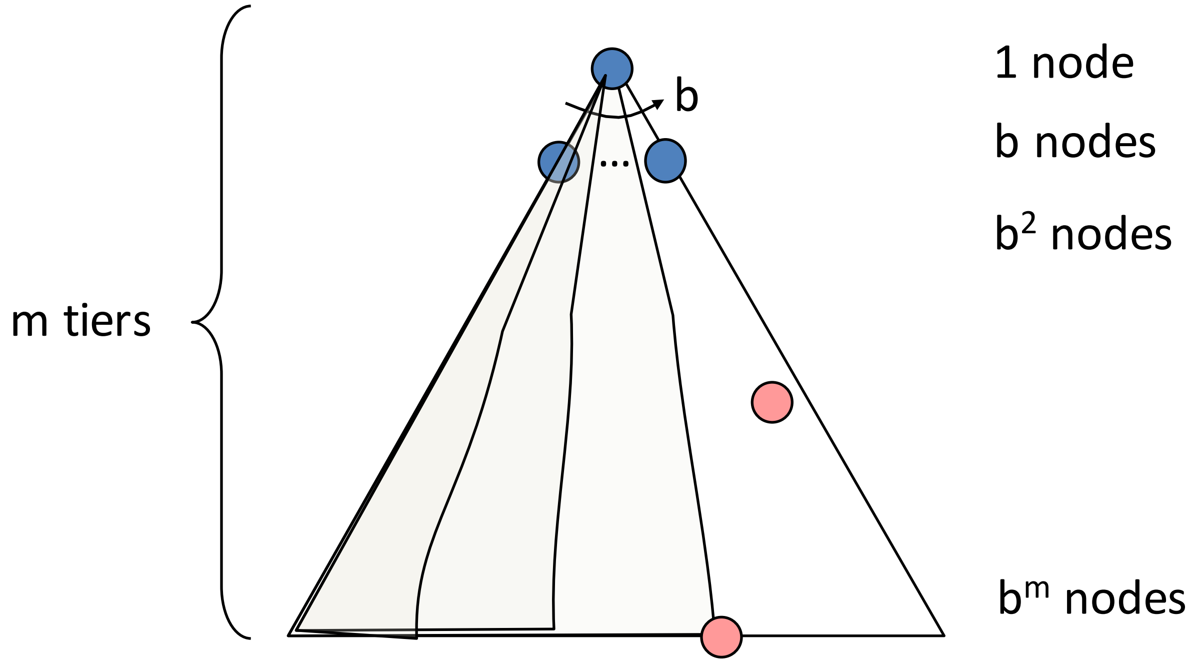
- Only siblings on path, O(bm)

<u>Complete</u>

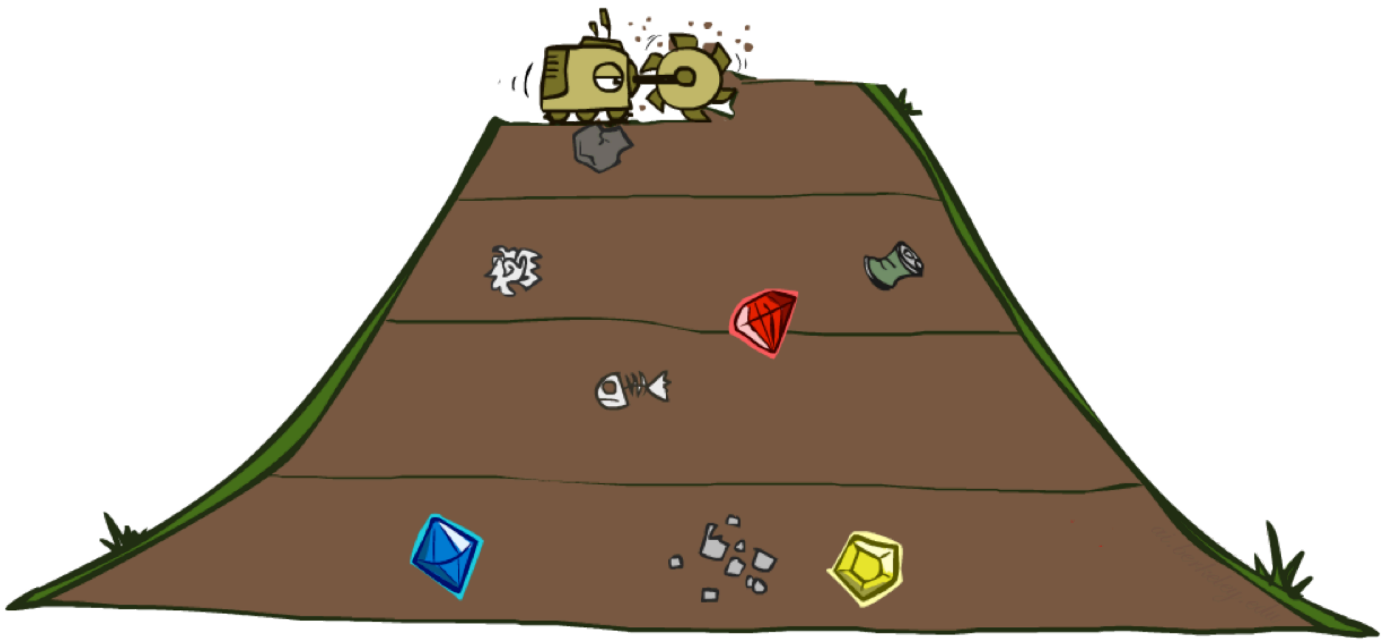
- Only if finite

<u>Optimal</u>

- No, "left-most" w/o regard to cost/depth



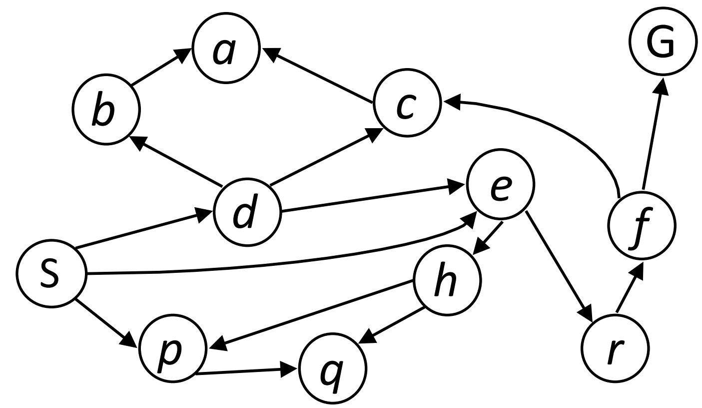
Breadth-First Search (BFS)

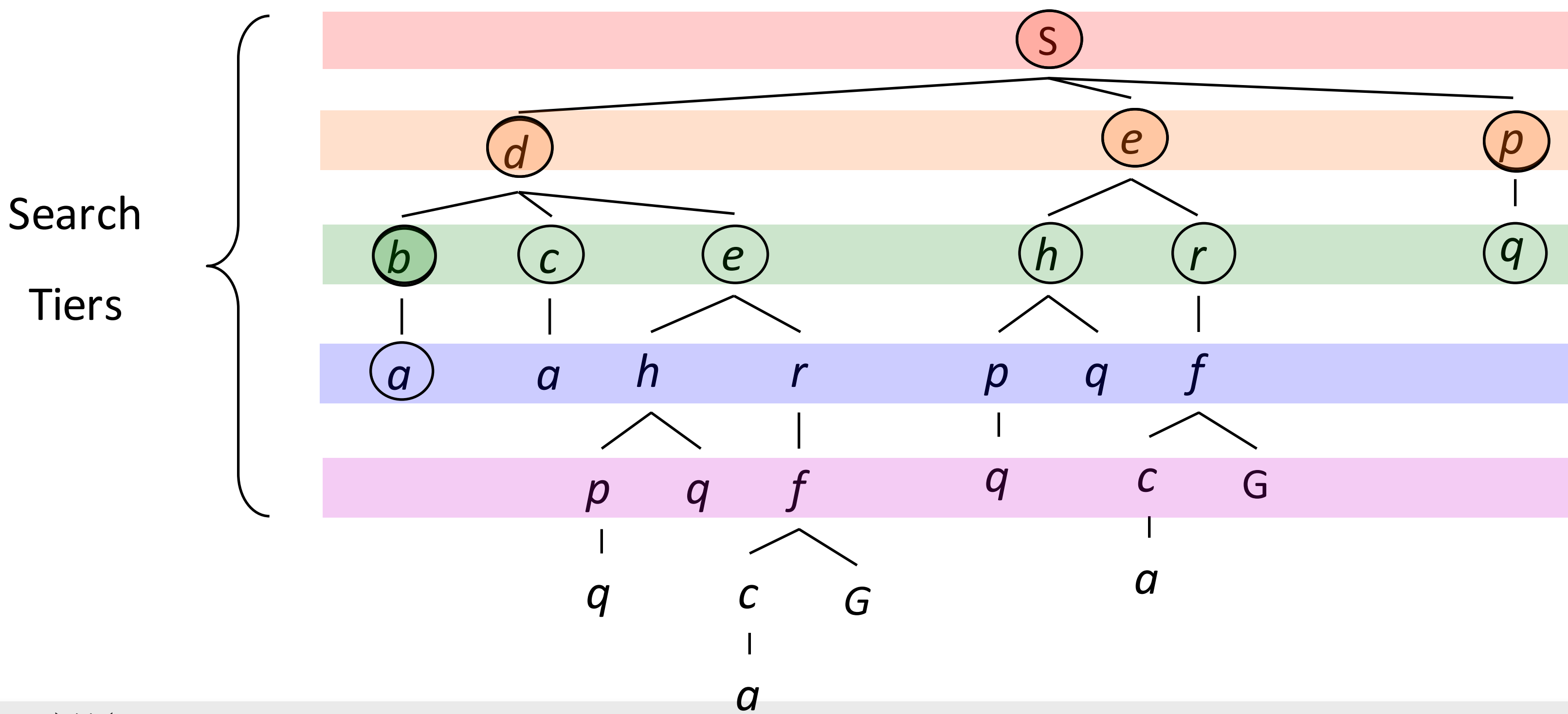


BFS Example

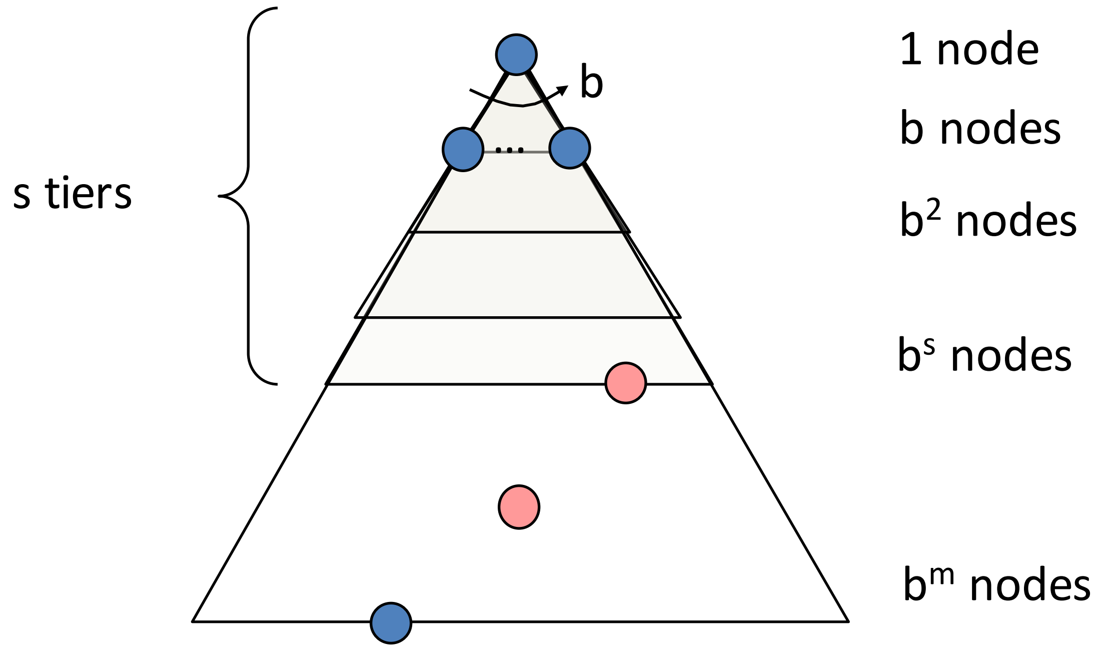
Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue





BFS Evaluation



Time

- Processes all nodes above shallowest solution, $O(b^s)$

Space

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

Complete

- Yes!

Optimal

- Only if all costs are 1 (more later)

DFS vs. BFS

- Empty-DFS
- Empty-BFS
- Maze-DFS
- Maze-BFS

Grounding the Branching Factor

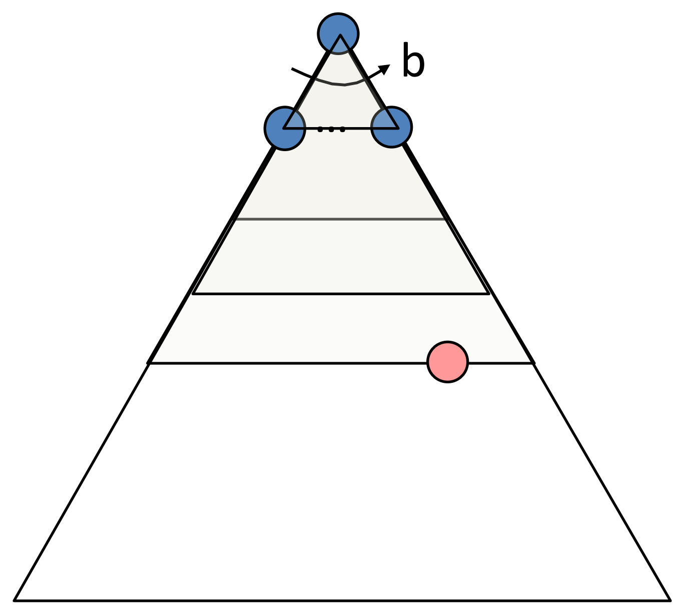
Depth	Nodes	Time	Memory
2	110	0.11 msecs	107 KB
4	11,110	11 msecs	10.6 MB
6	10^6	1.1 secs	1 GB
8	10^8	2 mins	103 GB
10	10^{10}	3 hours	10 TB
12	10^{12}	13 days	1 PB
14	10^{14}	3.5 years	99 PB
16	10^{16}	350 years	10 EB

Assumptions
- b = 10
- 1 million nodes/second
- 1000 bytes/node

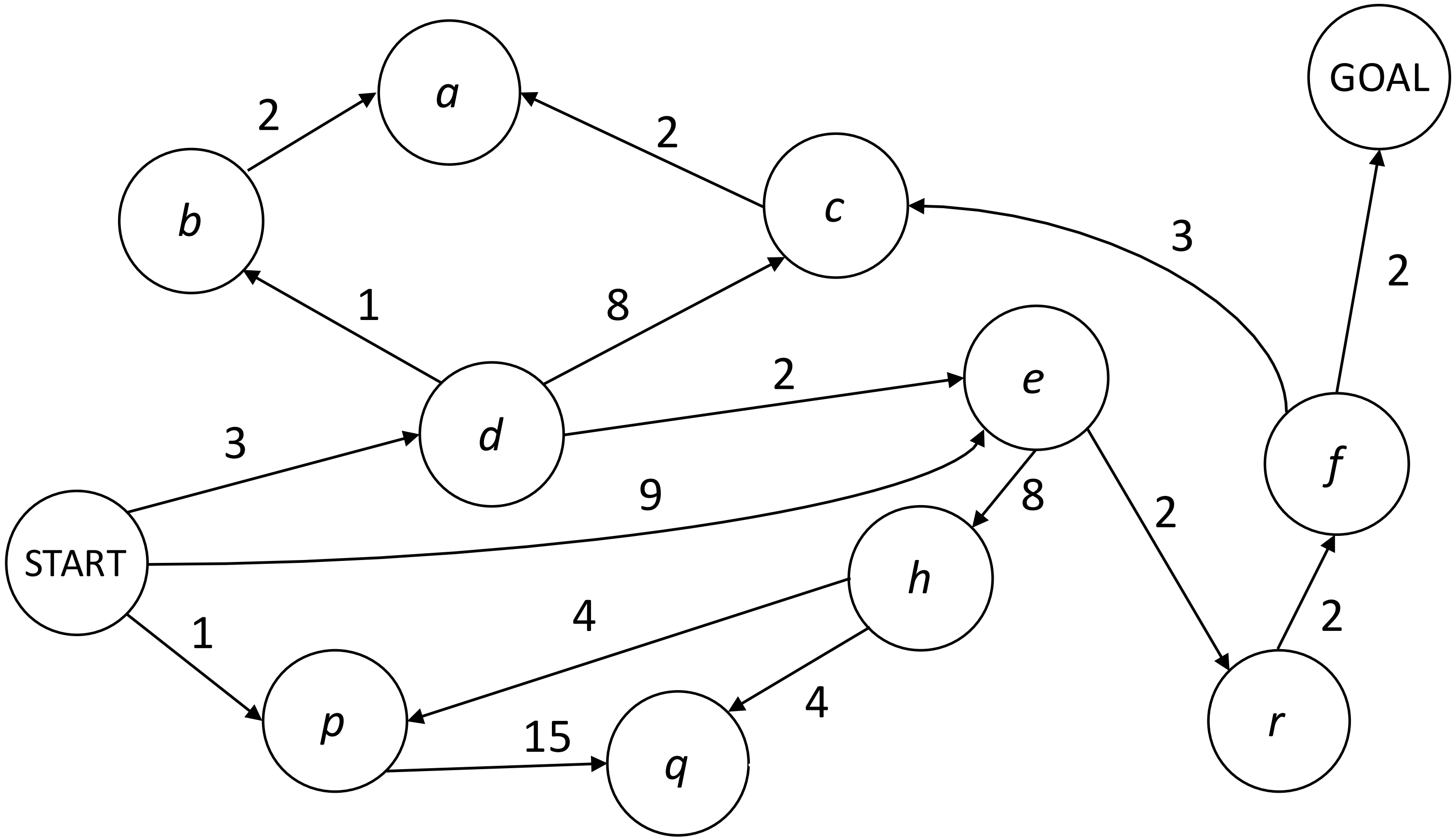
Memory often becomes the limiting factor

Iterative Deepening DFS

- Basic idea: DFS memory with BFS time/shallow solution
  - DFS up to 1
  - DFS up to 2
  - ….
- Generally most work happens in the lowest level searched, so not too wasteful

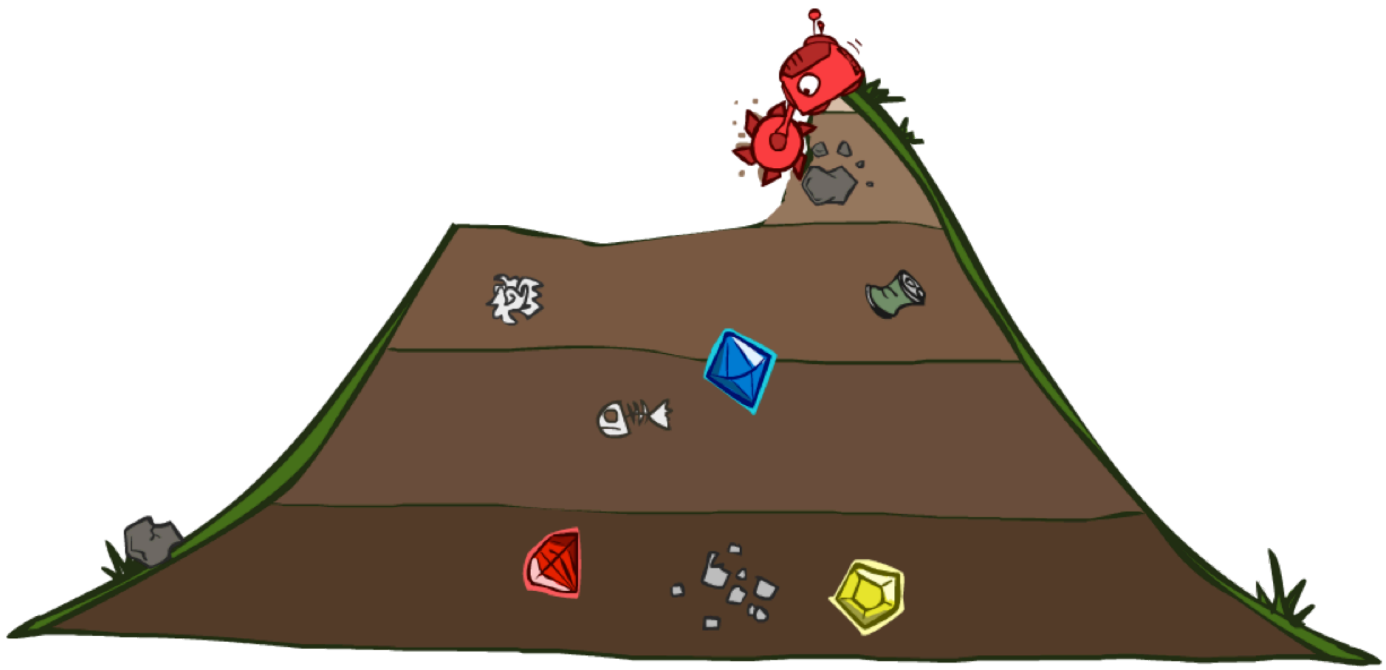


Cost-Sensitive Search



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

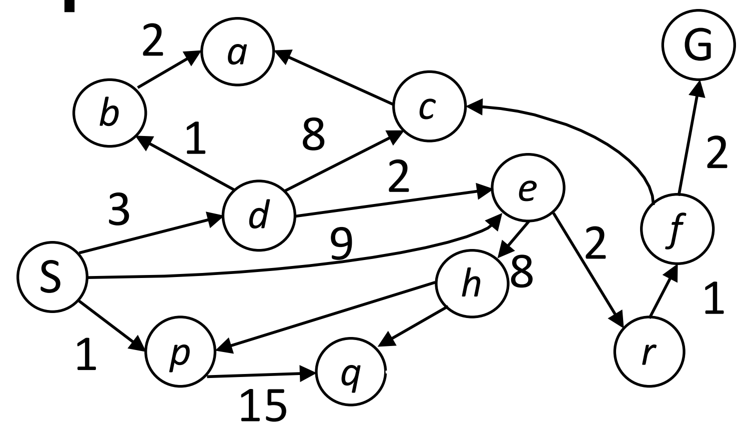
Uniform-Cost Search (UCS)



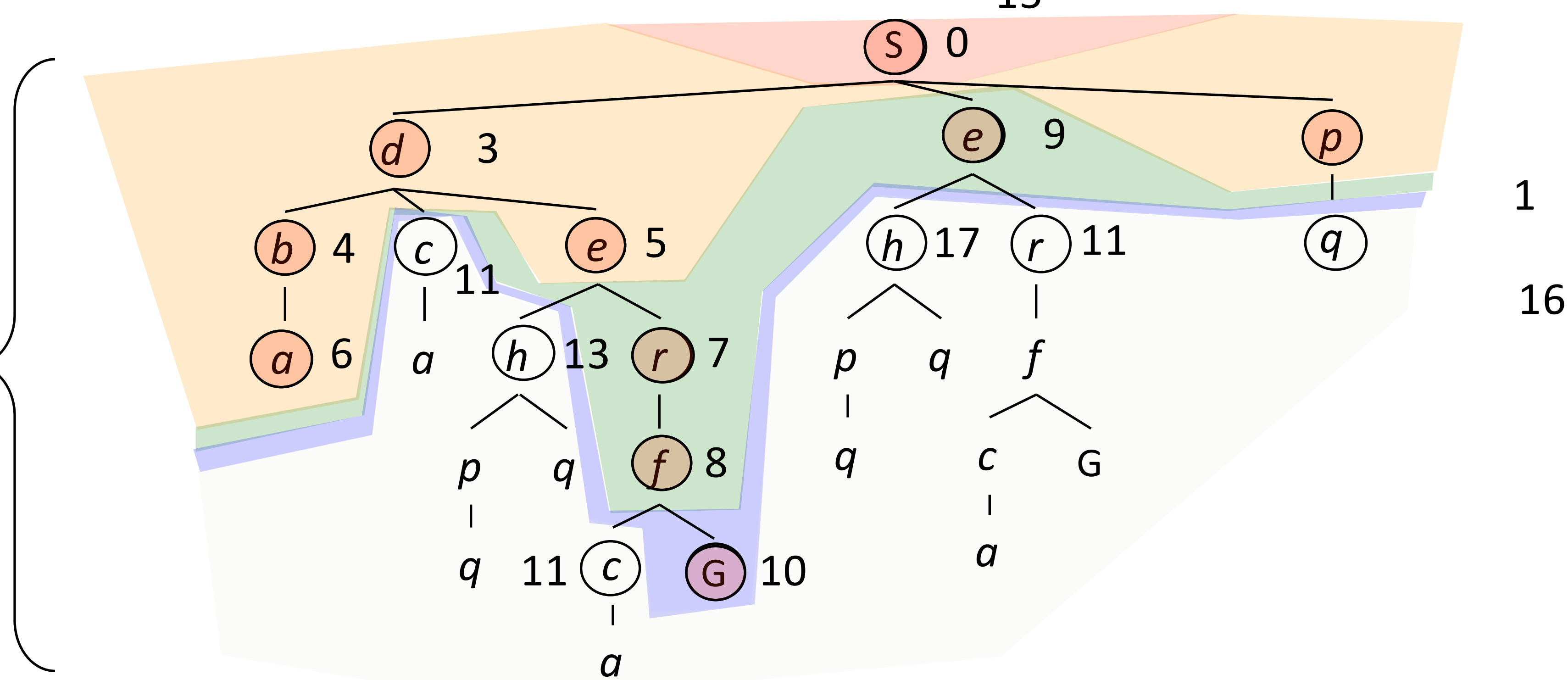
UCS Example

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)



Cost contours



UCS Evaluation

Time

- $O(b^{C^*/\varepsilon})$

Space

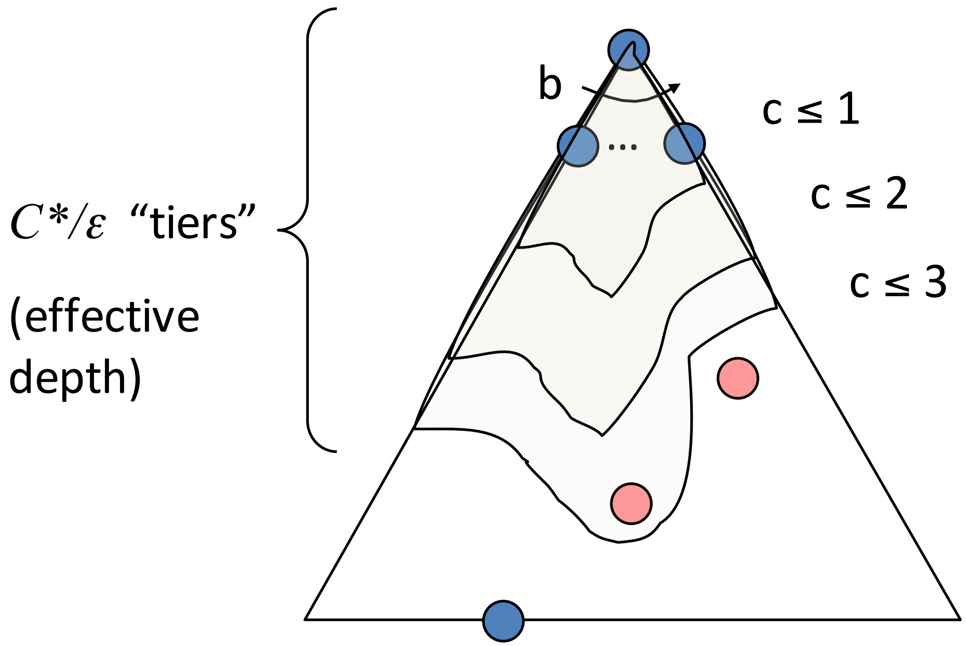
- $O(b^{C^*/\varepsilon})$

Complete

- Yes!

Optimal

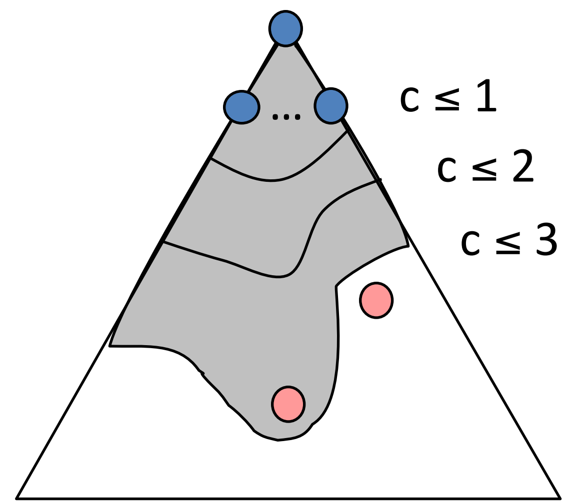
- Yes!

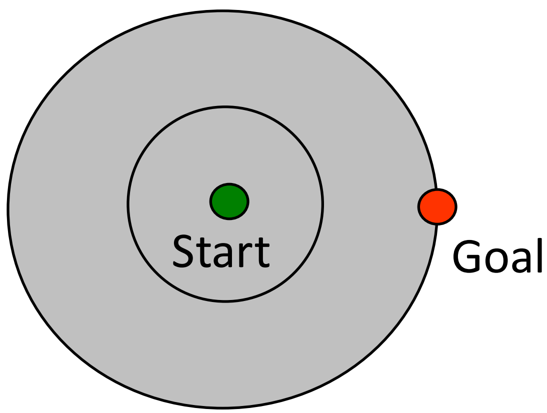


Assume solution costs C* and arcs cost at least ε

UCS vs. DFS vs. BFS

- UCS is good and optimal
- However, it still moves in every direction – it's not **informed** about goal direction…

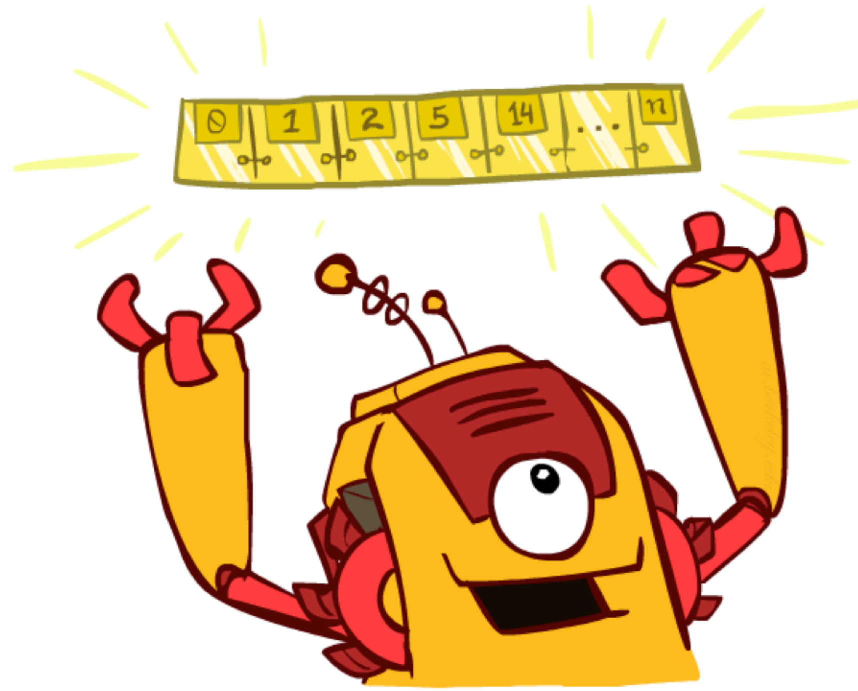




- Empty-UCS
- Maze-UCS
- MazeCost-DFS
- MazeCost-BFS
- MazeCost-UCS

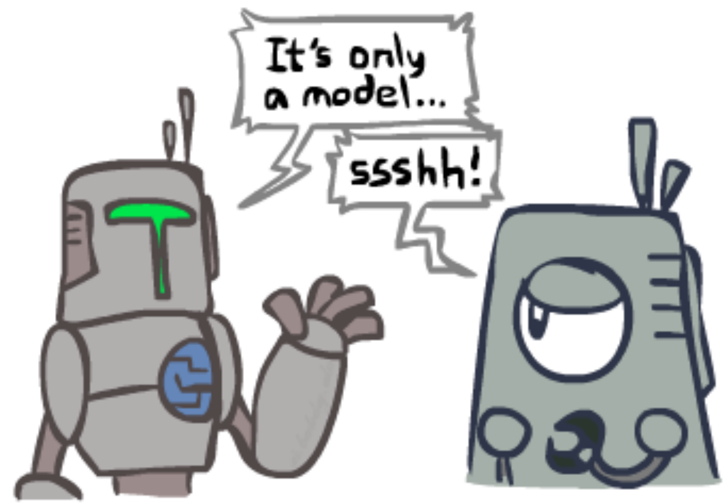
Unification

- 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

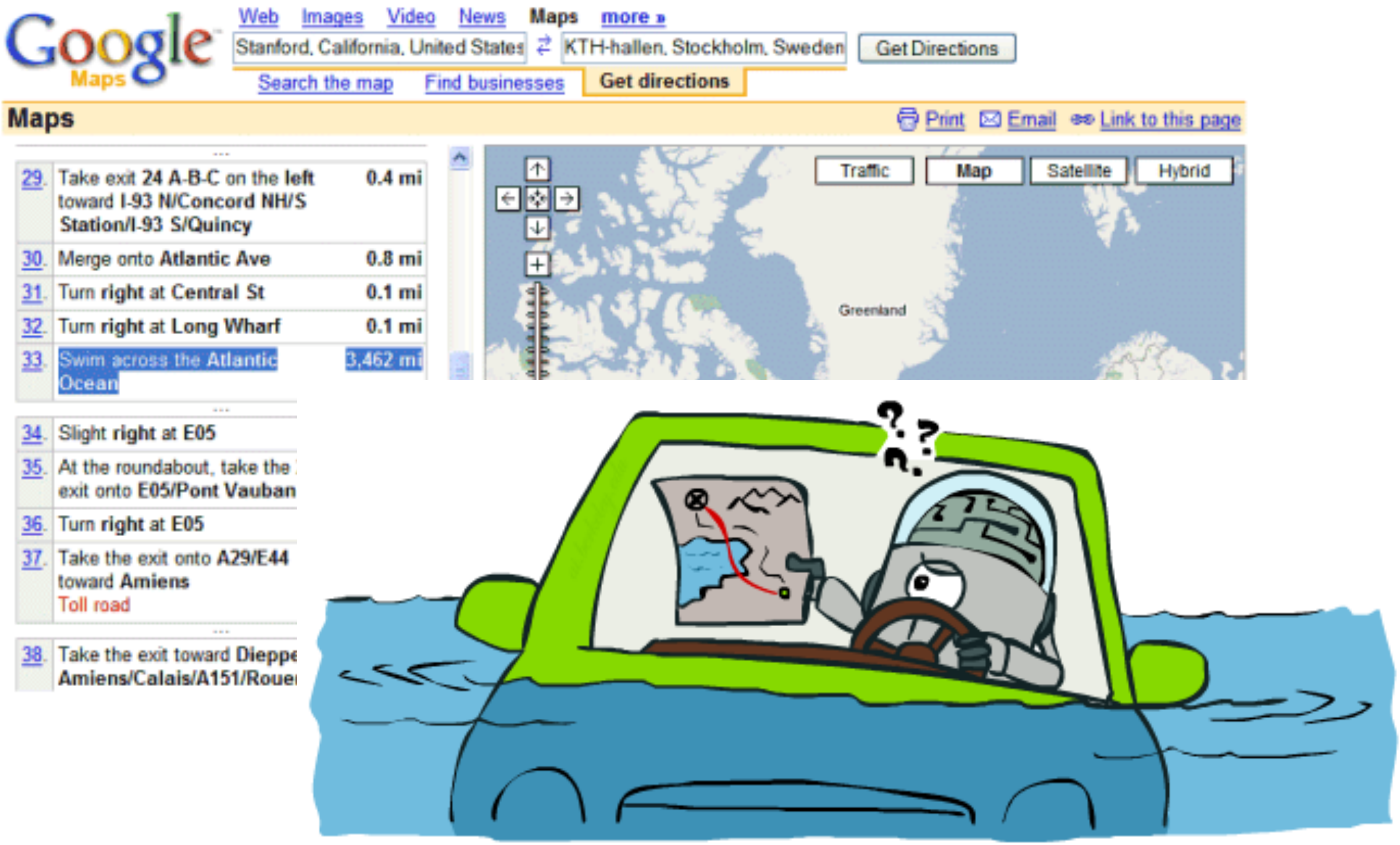


A Reminder

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



Search Gone Wrong



Summary

- We evaluated several uninformed strategies to solve a search problem
  - **DFS**, **ID-DFS**, **BFS**, **UCS**
- DFS, BFS, and UCS can all be implemented via a generic graph-search algorithm over a search tree by simply changing how the fringe is organized