COSC 411: Artificial Intelligence

Uninformed Search

Dr. Shuangquan (Peter) Wang
(spwang@salisbury.edu)

Department of Computer Science

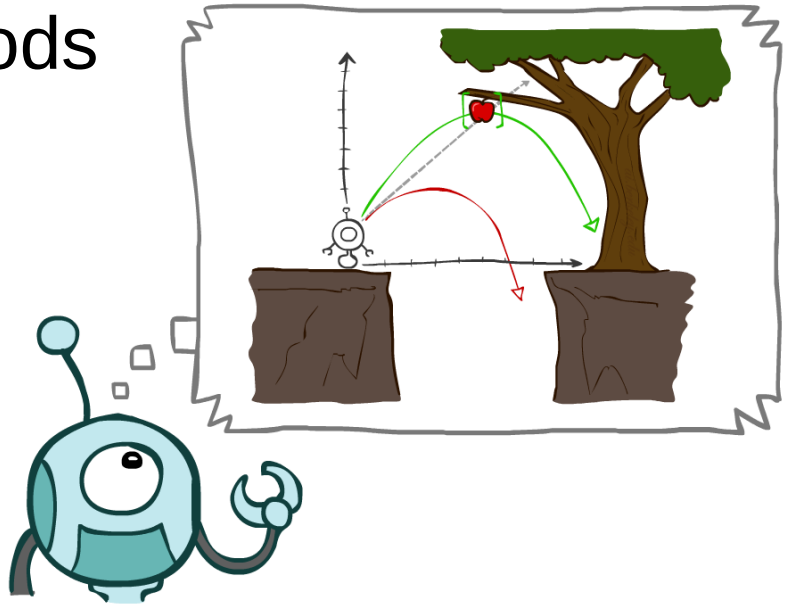
Salisbury University

About this note

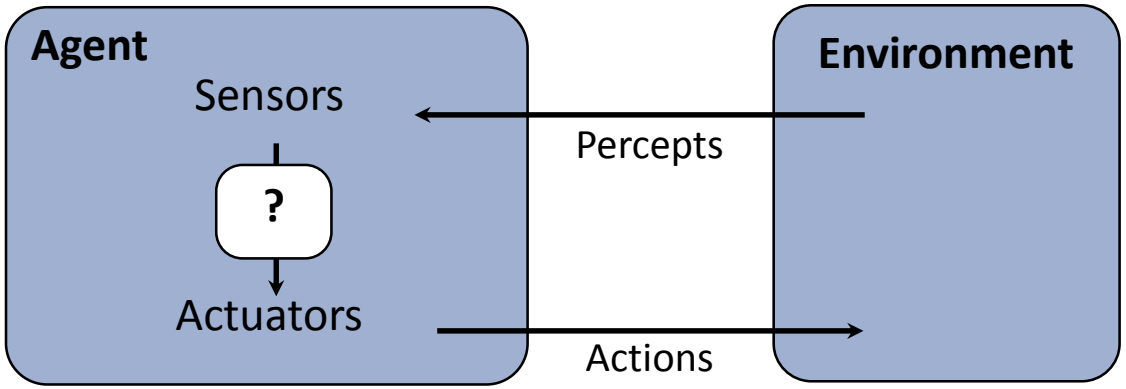
- Most slides of this note are from:
  - "CS 188 Introduction to Artificial Intelligence" teaching material, UC Berkeley, Summer 2020 (instructor: Nikita Kitaev)
    - https://inst.eecs.berkeley.edu/~cs188/su20/
  - "CS 188 Introduction to Artificial Intelligence" teaching material, UC Berkeley, Spring 2019 (instructor: Sergey Levine and Stuart Russell)
    - https://inst.eecs.berkeley.edu/~cs188/sp19/

Contents

- Agents that Plan Ahead
- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search



Agents and environments



- An agent ***perceives*** its environment through ***sensors*** and ***acts*** upon it through ***actuators***

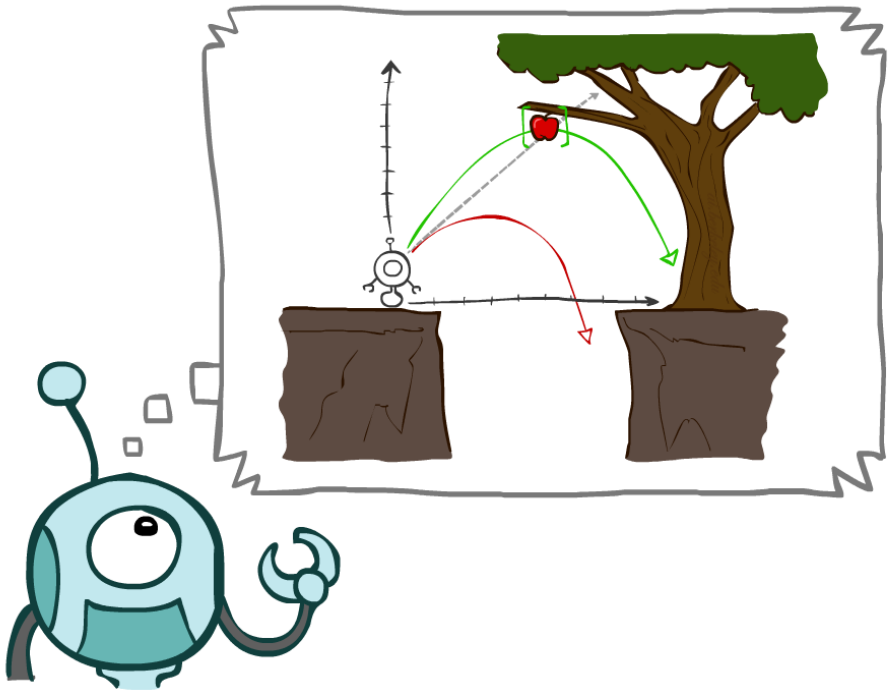
Rationality

- A ***rational agent*** chooses actions maximize the ***expected*** utility
  - agents that have a goal, and a cost
    - E.g., reach goal with lowest cost
  - agents that have numerical utilities, rewards, etc.
    - E.g., take actions that maximize total reward over time (e.g., largest profit in $)

Agent design

- **The environment type largely determines the agent design**
  - ***Fully/partially observable*** => agent requires ***memory*** (internal state)
  - ***Discrete/continuous*** => agent may not be able to enumerate ***all states***
  - ***Stochastic/deterministic*** => agent may have to prepare for ***contingencies***
  - ***Single-agent/multi-agent*** => agent may need to behave ***randomly***

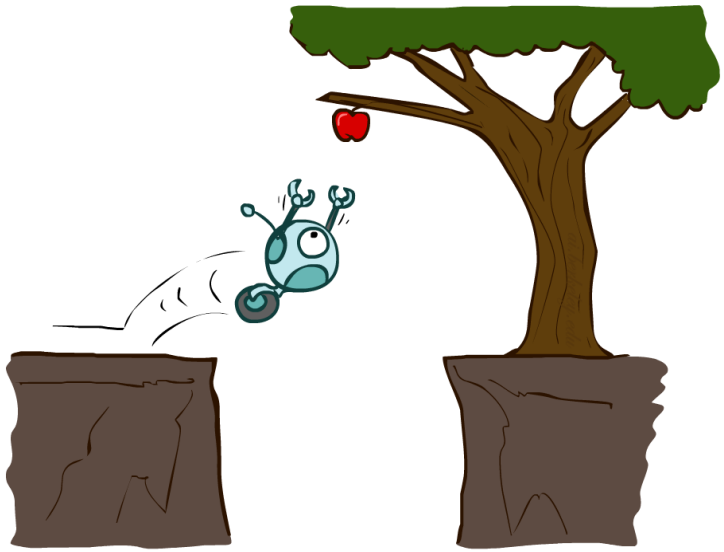
Agents that Plan



Reflex Agents

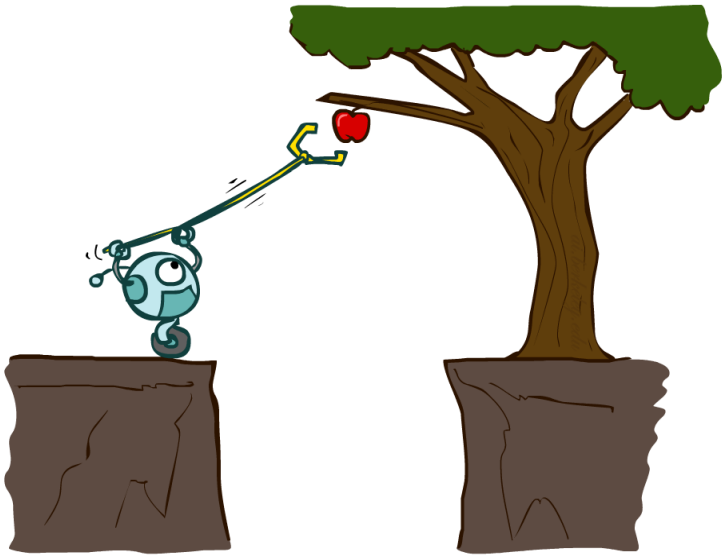
- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - May have memory or a model of the world's current state
  - Do not consider the future consequences of their actions
  - Consider how the world **IS**

- Can a reflex agent be rational?

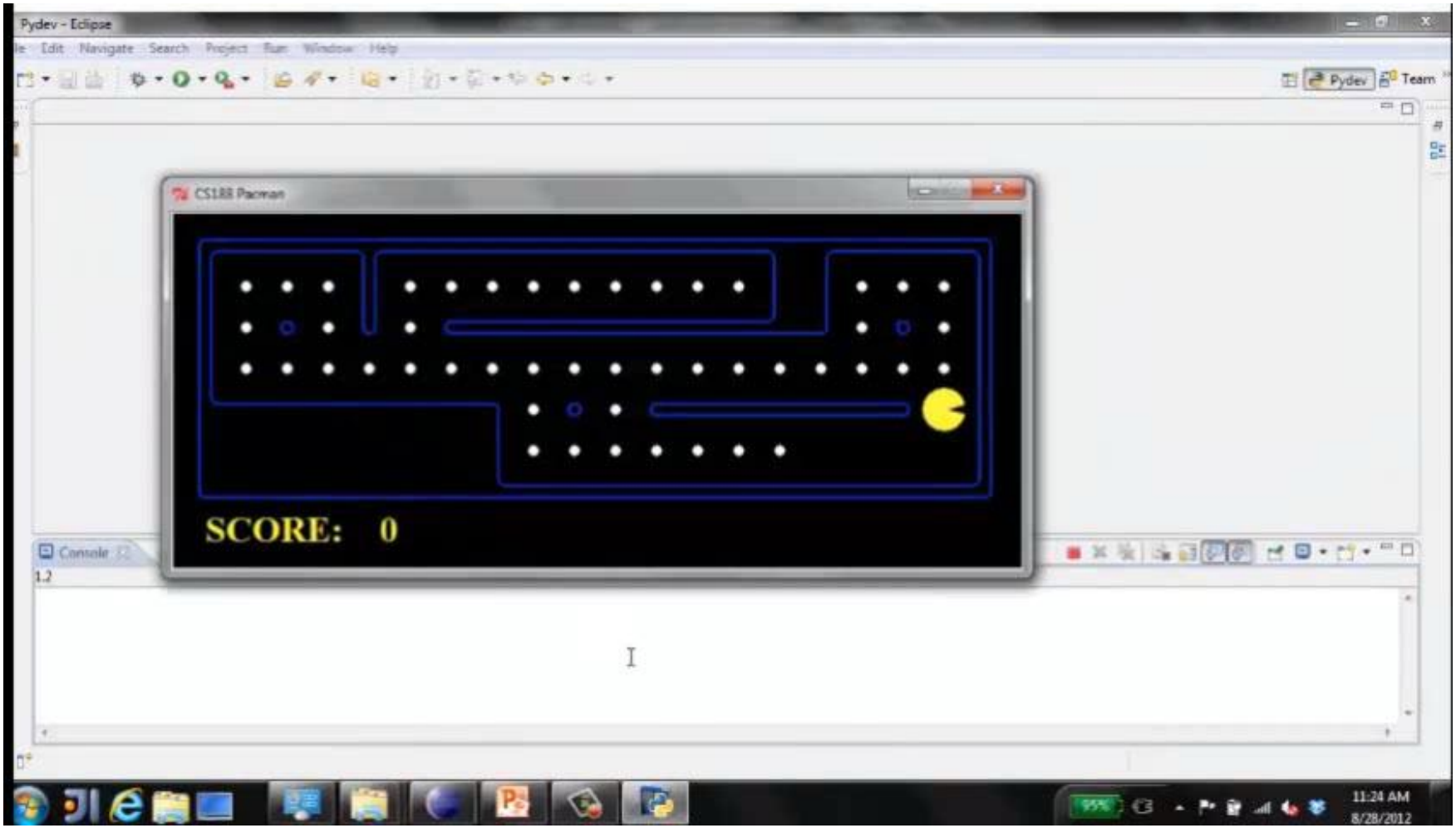


Planning Agents

- 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 (test)
  - Consider how the world **WOULD BE**

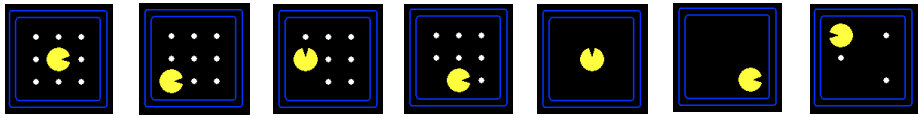


Search Problems (Pac-Man)

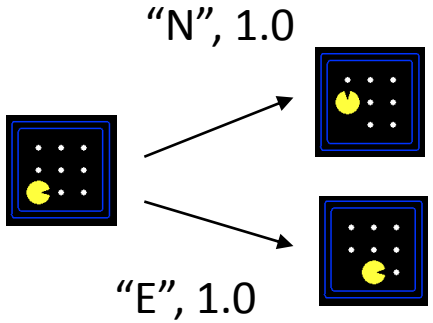


Search Problems

- A search problem consists of:
  - A state space



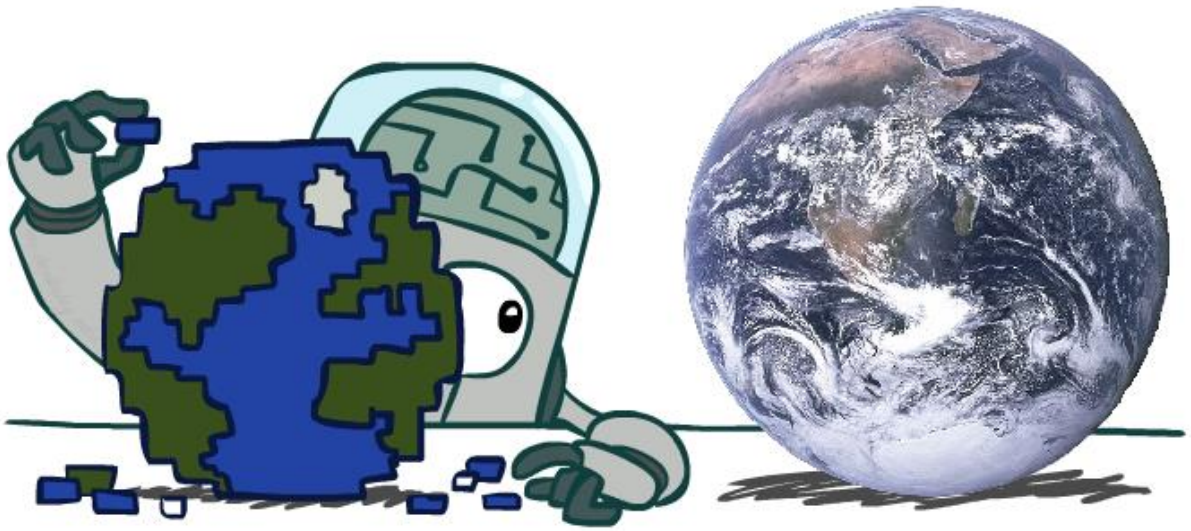
  - A successor function (with actions, costs)



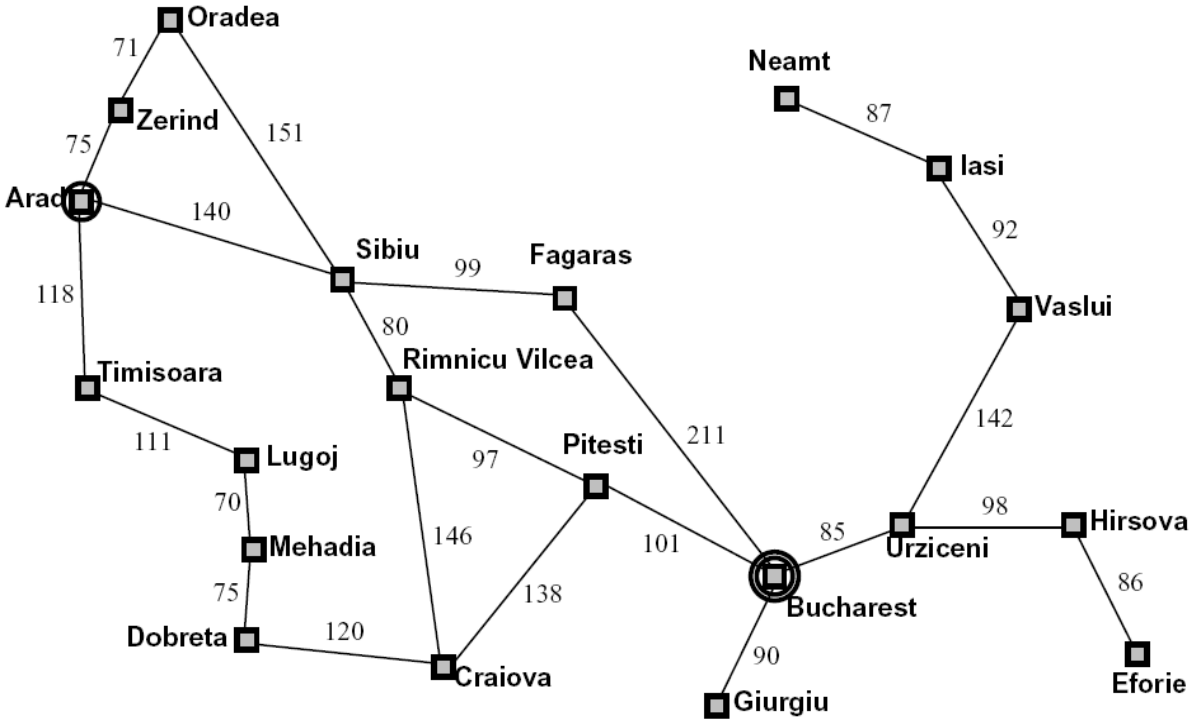
  - A start state and a goal test

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

Search Problems Are Models



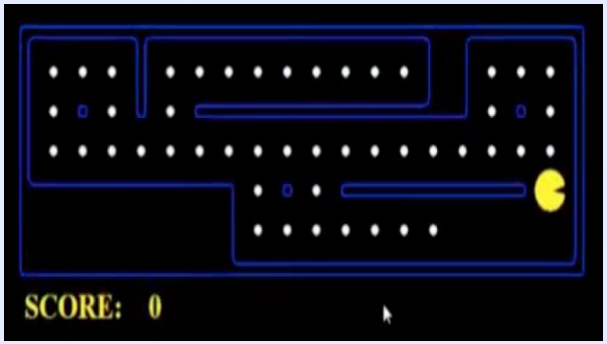
Example: Traveling in Romania



- State space:
  - Cities
- Successor function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?

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: N/S/E/W
  - Successor: update location only
  - Goal test: is (x,y)=END

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

State Space Sizes?

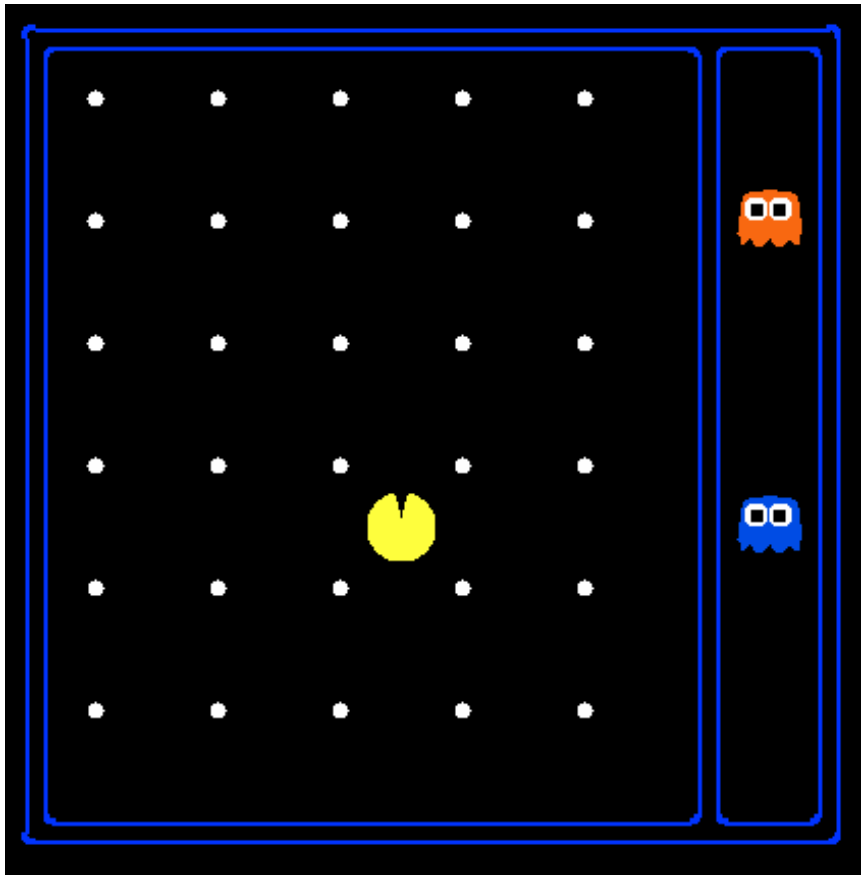
- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: N/S/E/W

- How many
  - World states?

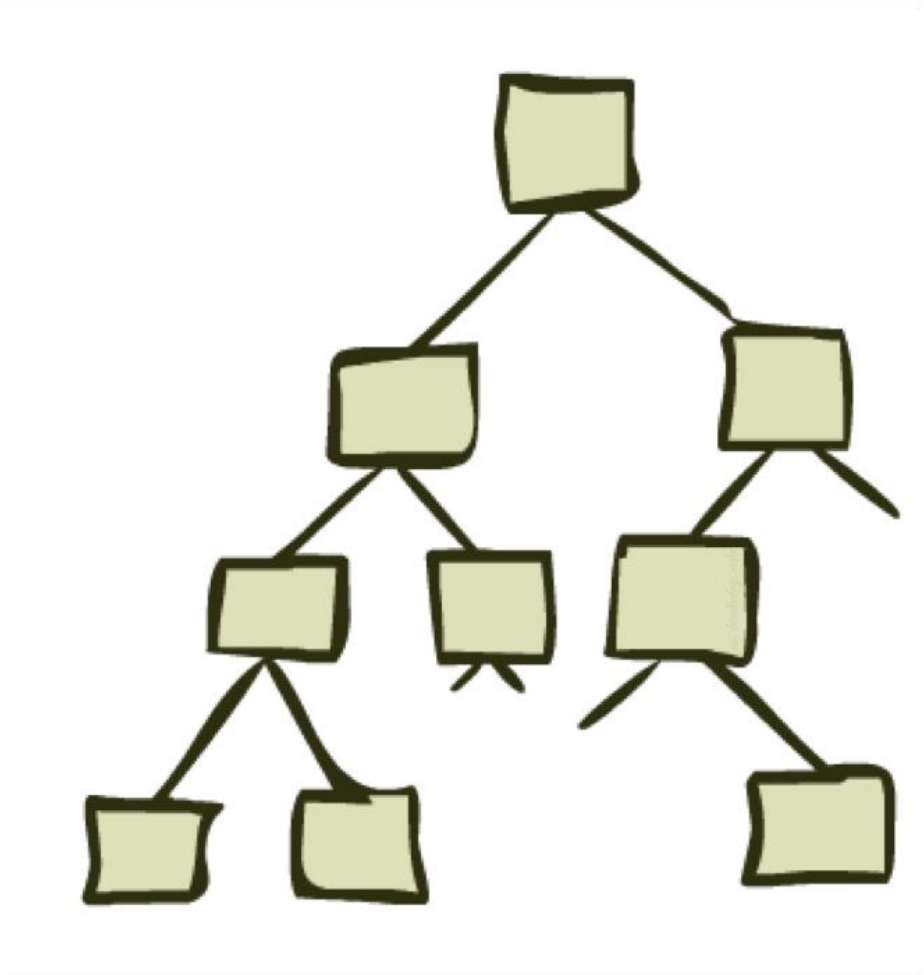
    $120 \times (2^{30}) \times (12^2) \times 4$
  - States for pathing?

    120
  - States for eat-all-dots?

    $120 \times (2^{30})$

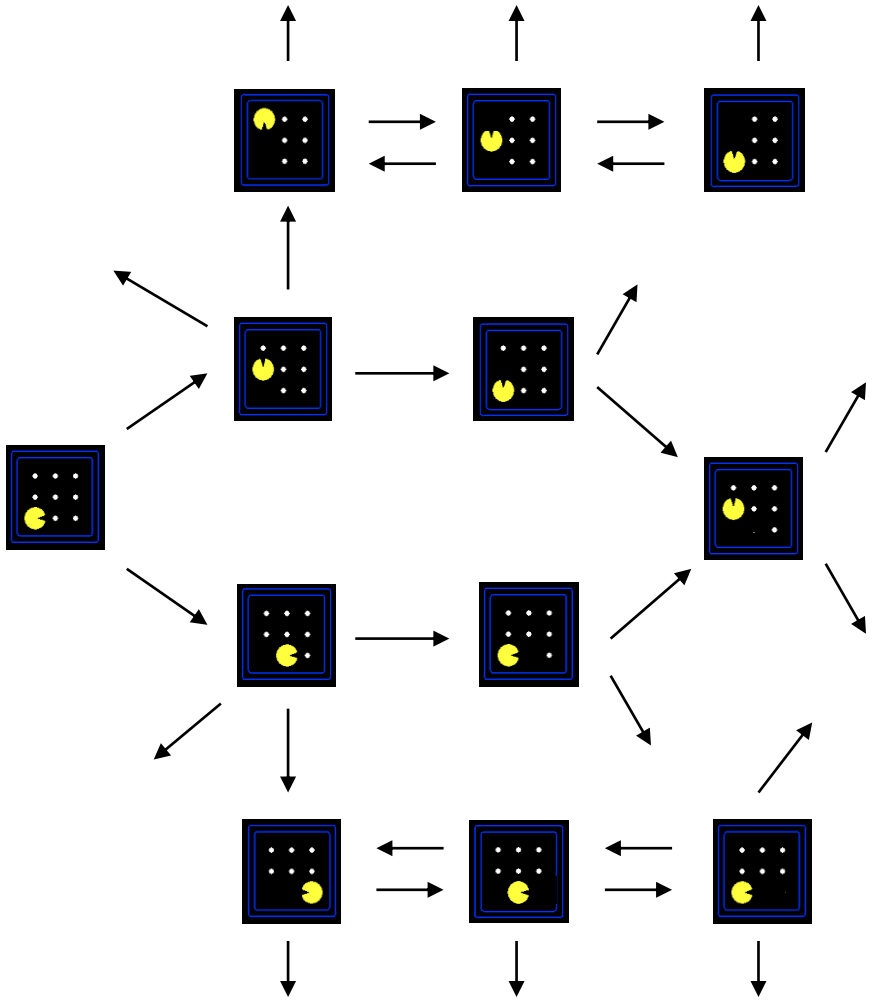


State Space Graphs and Search Trees

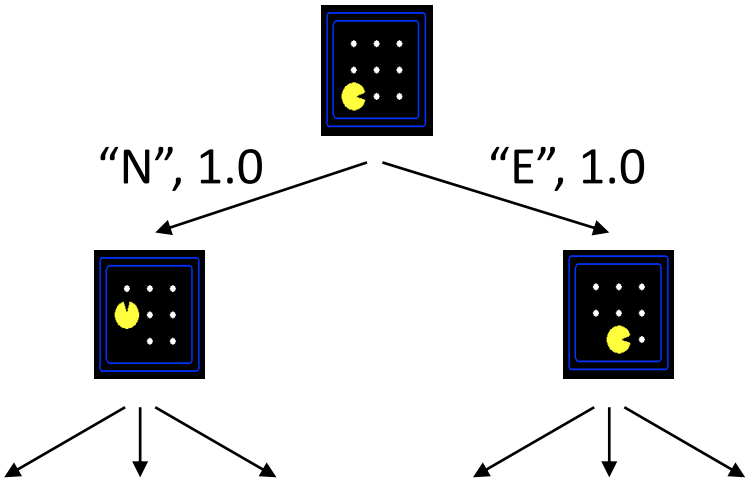


State Space Graphs

- 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

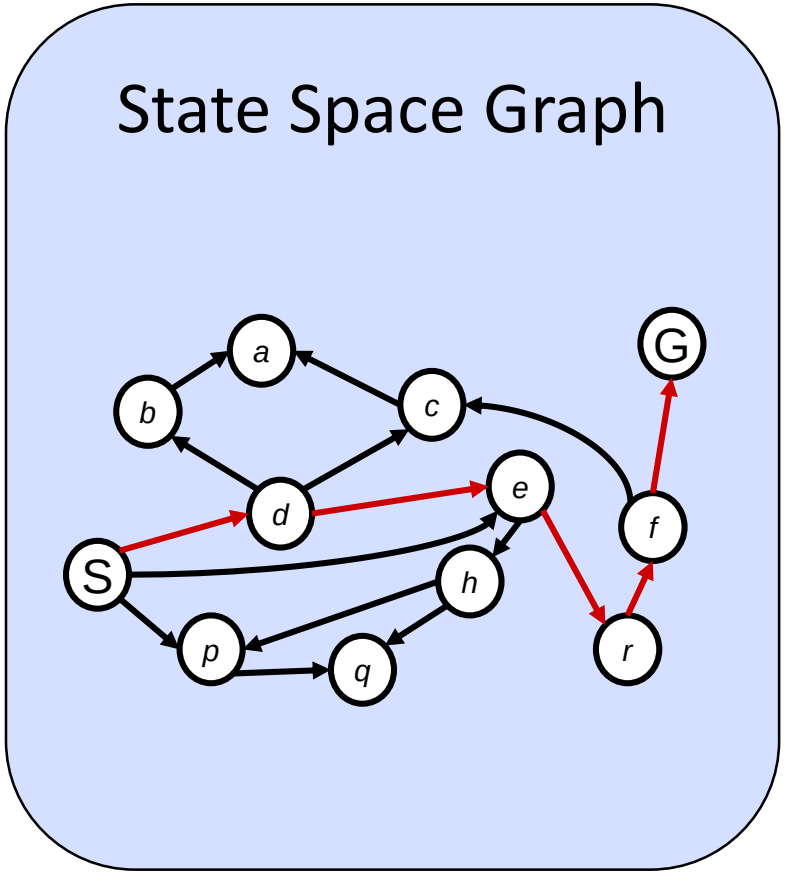


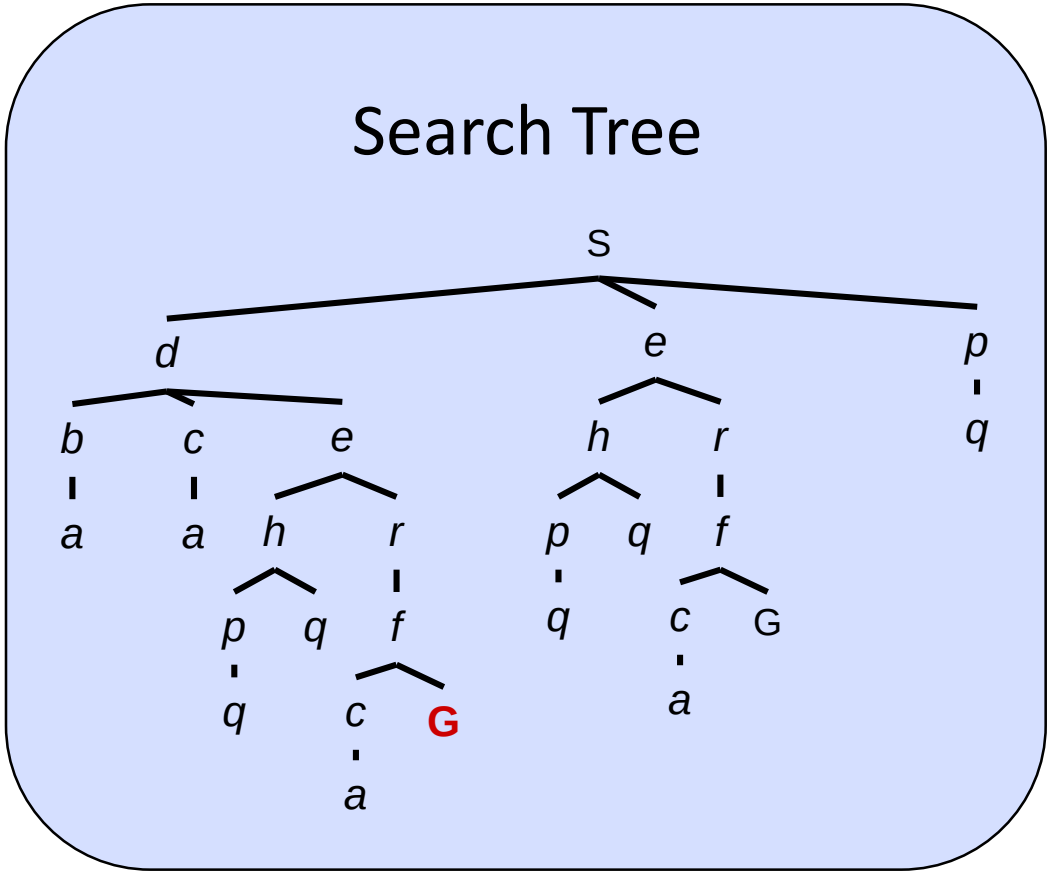
This is now / start

Possible futures

- A 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

State Space Graphs vs. Search Trees





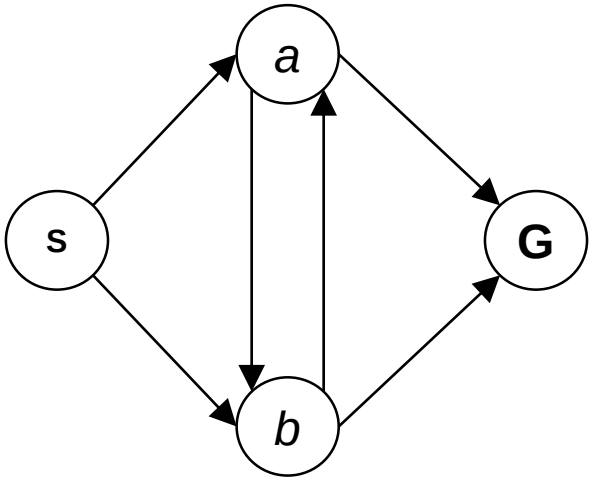
Each NODE in the search tree is an entire PATH in the state space graph.

We construct both on demand – and we construct as little as possible.

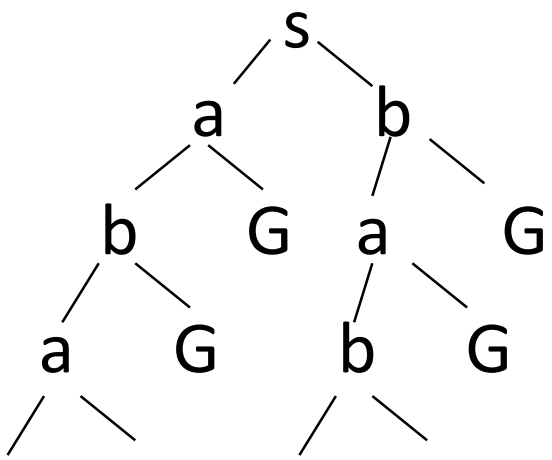
Salisbury UNIVERSITY

State Space Graphs vs. Search Trees

Consider this 4-state graph:

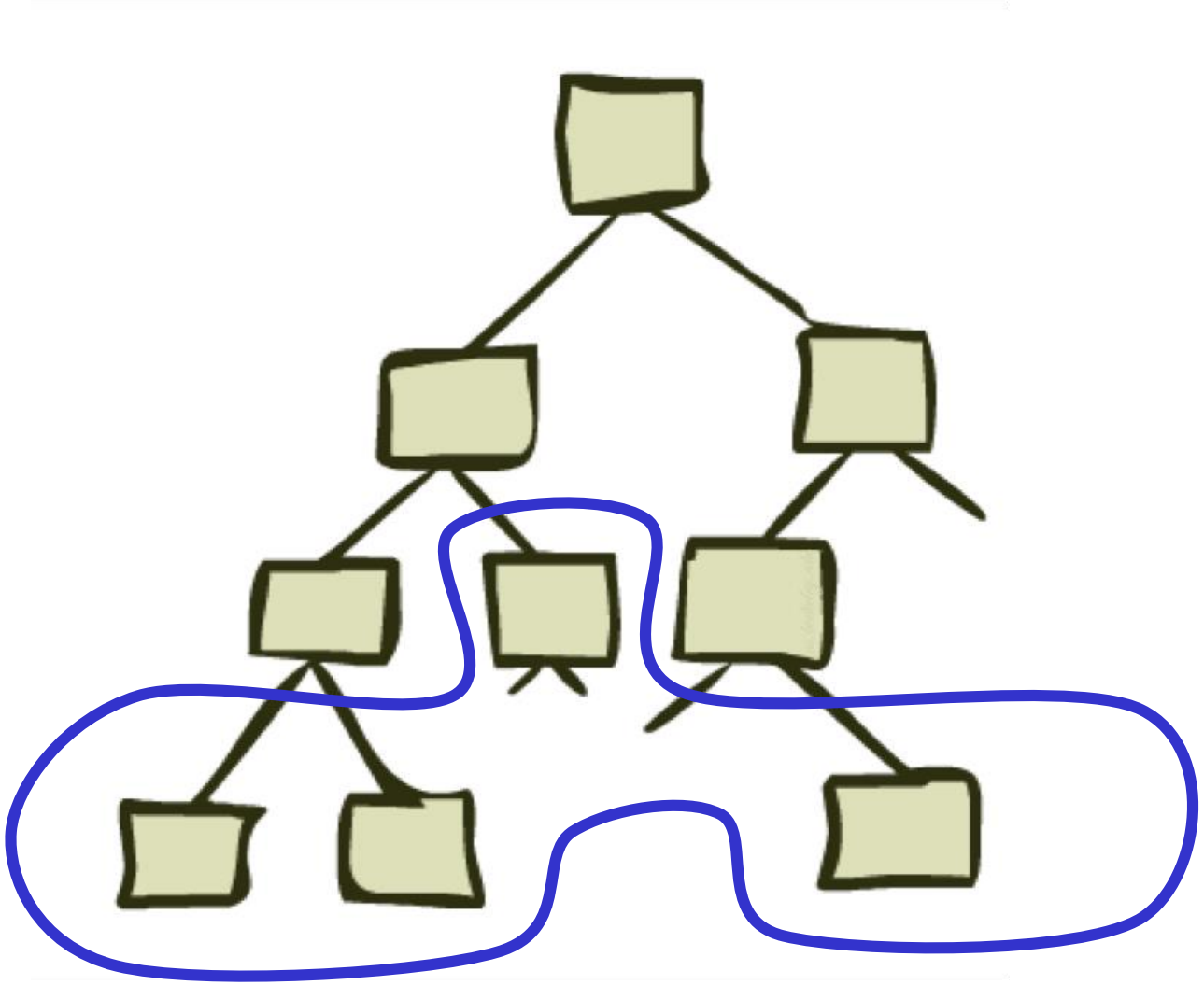


How big is its search tree (from S)?

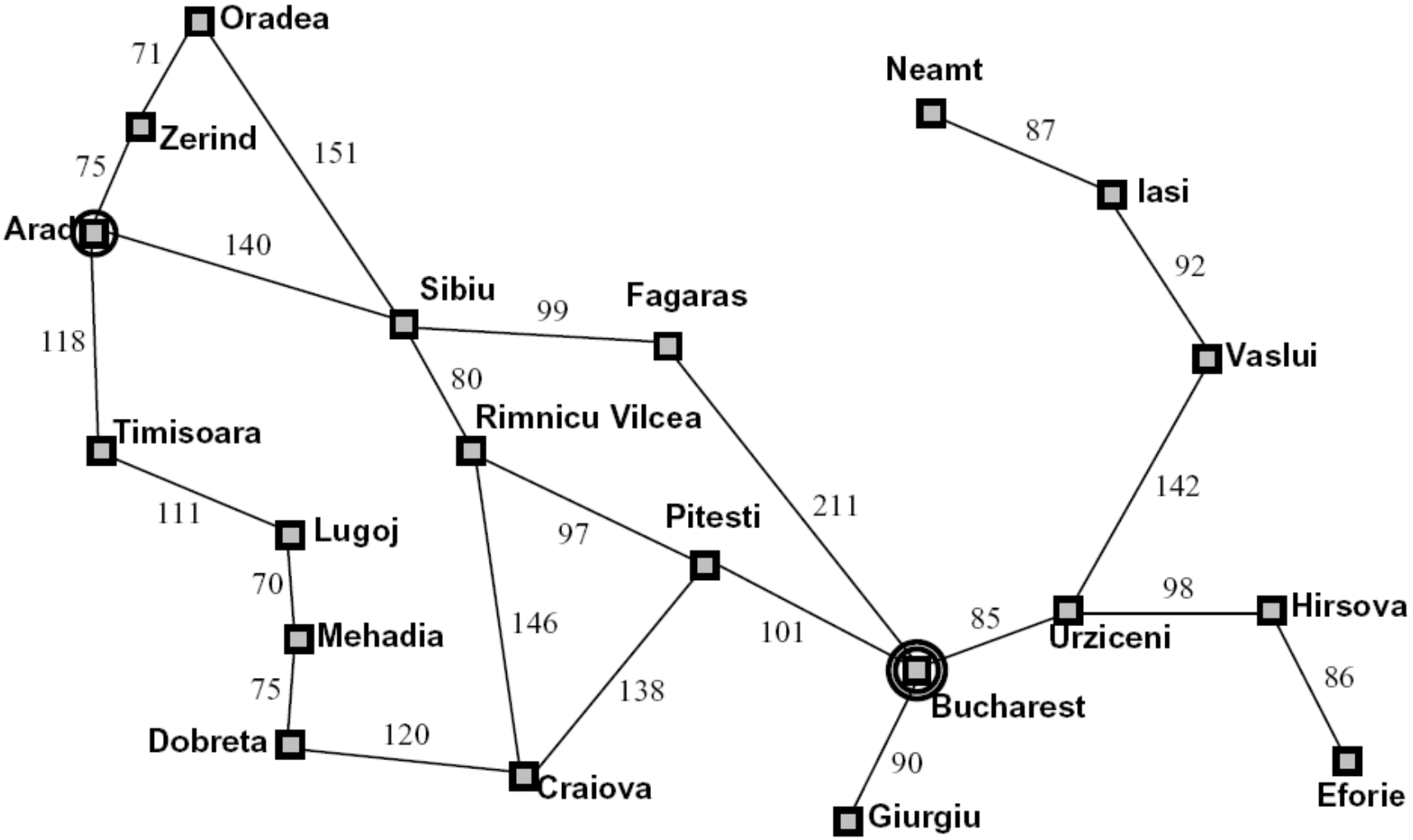


Important: Lots of repeated structure in the search tree!

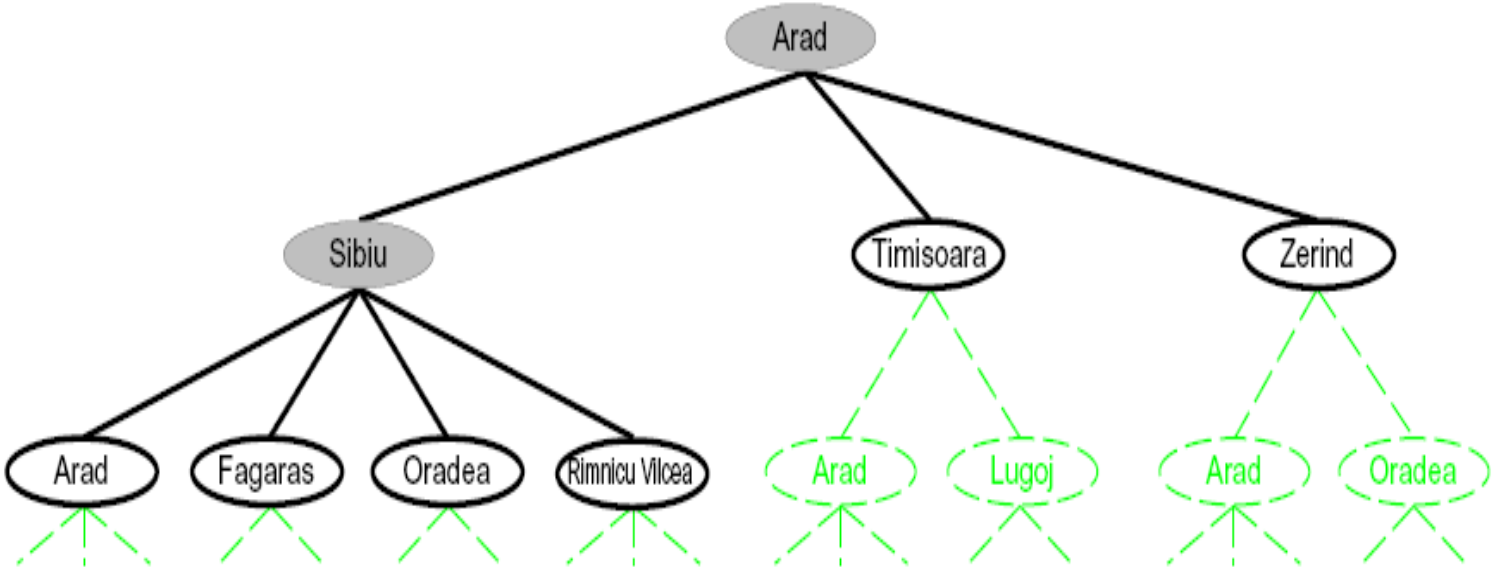
Tree Search



Search Example: Romania



Searching with a Search Tree



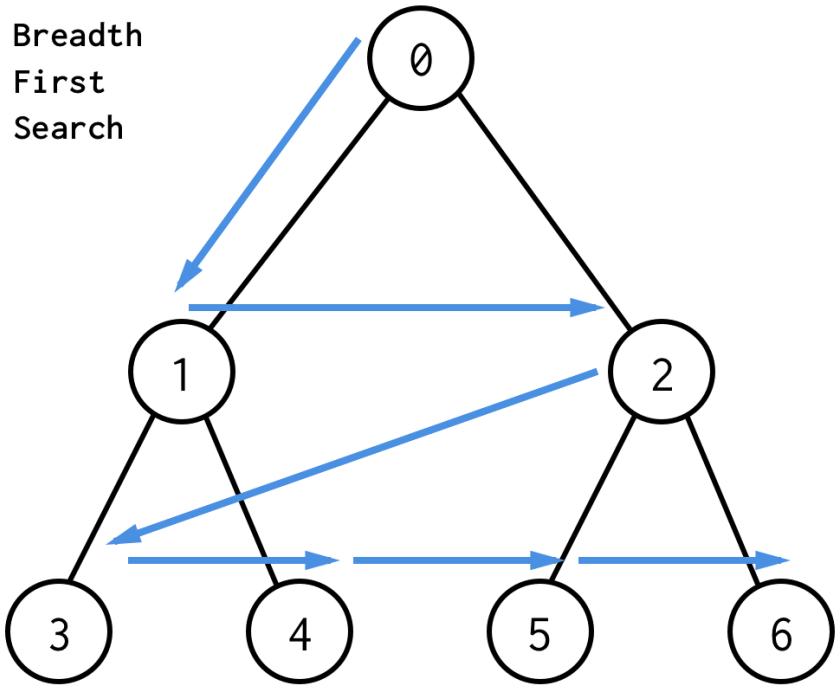
- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

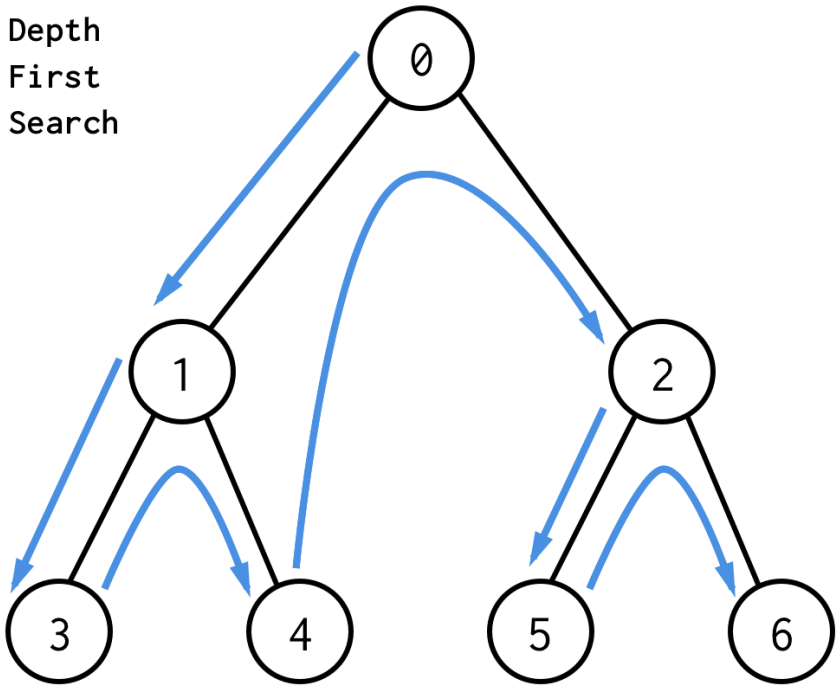
General Tree Search

```
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
  - Expansion
  - Exploration strategy
- Main question: which fringe nodes to explore?

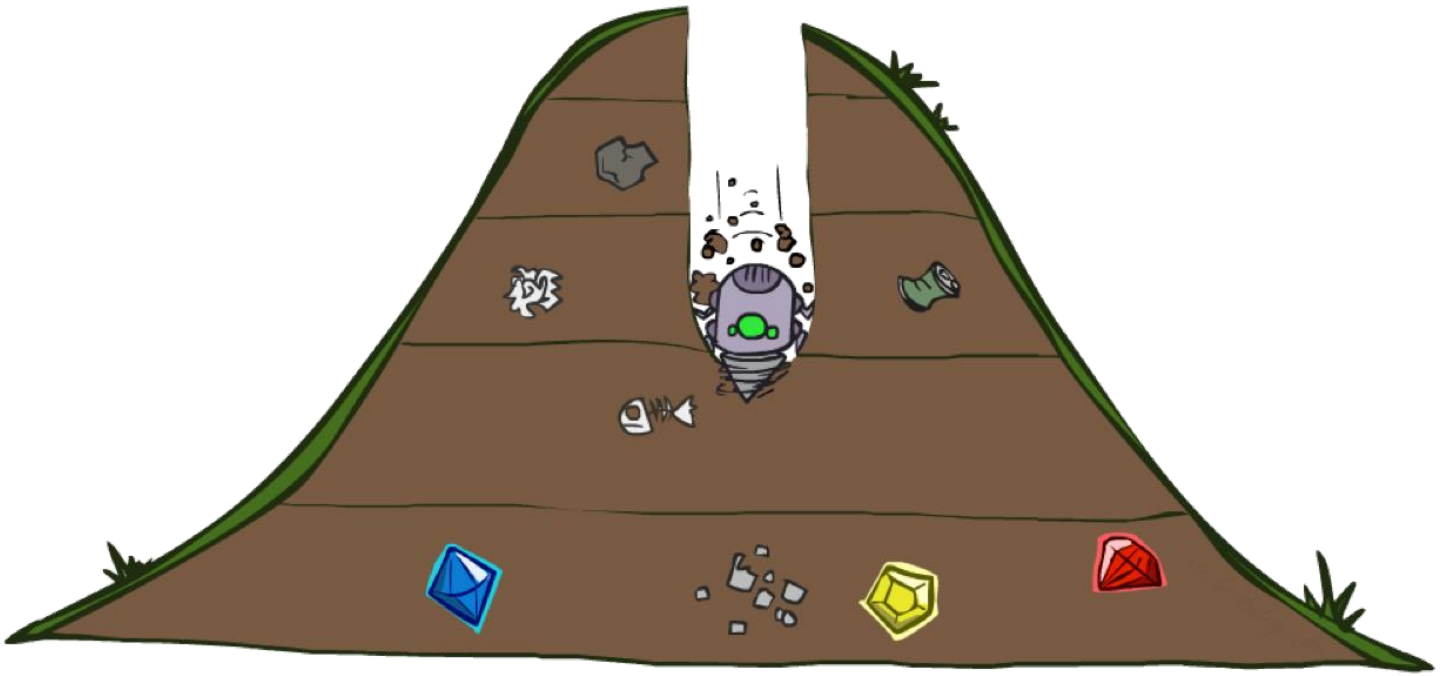
Tree Search





https://mishadoff.com/blog/dfs-on-binary-tree-array/

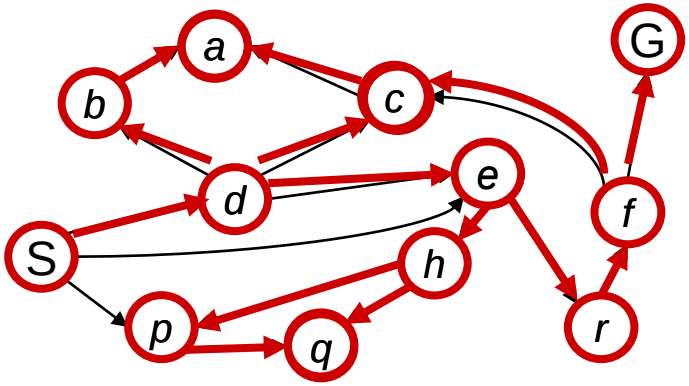
Depth-First Search

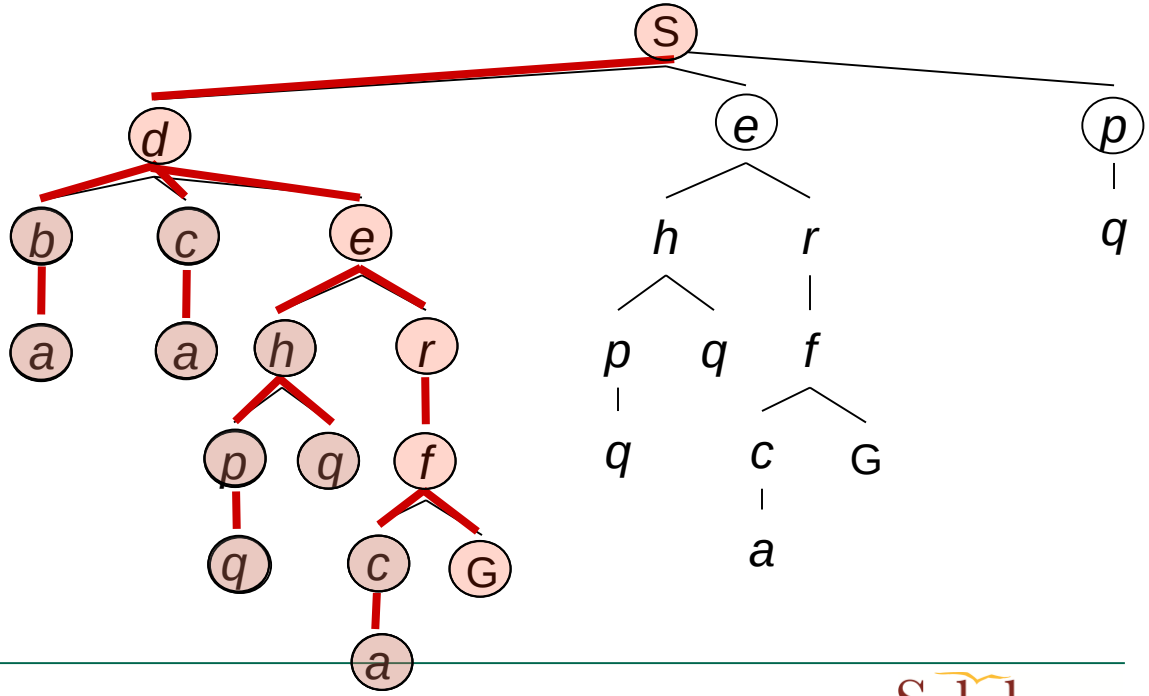


Depth-First Search

Strategy: expand a deepest node first

Implementation: Fringe is a ***LIFO stack***

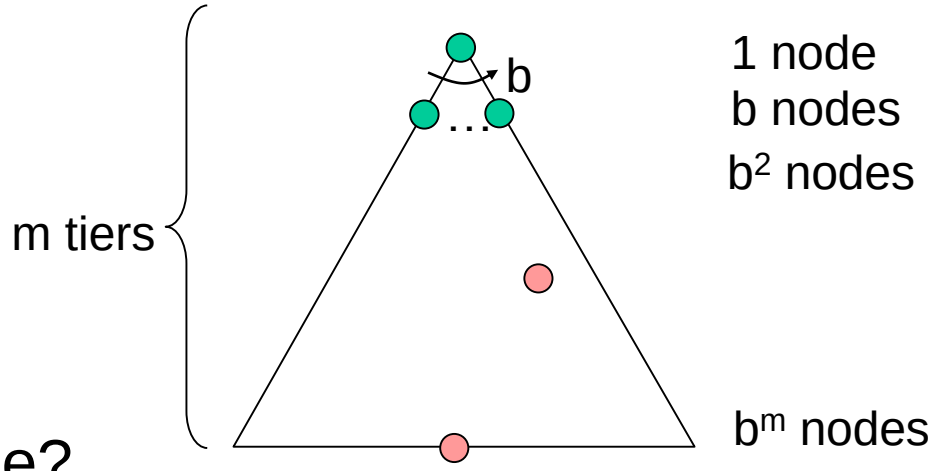




Search Algorithm Properties

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

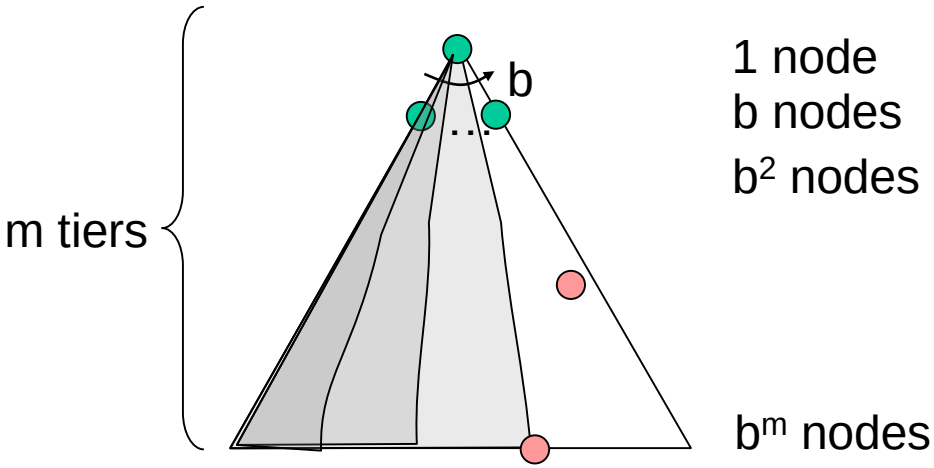
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths



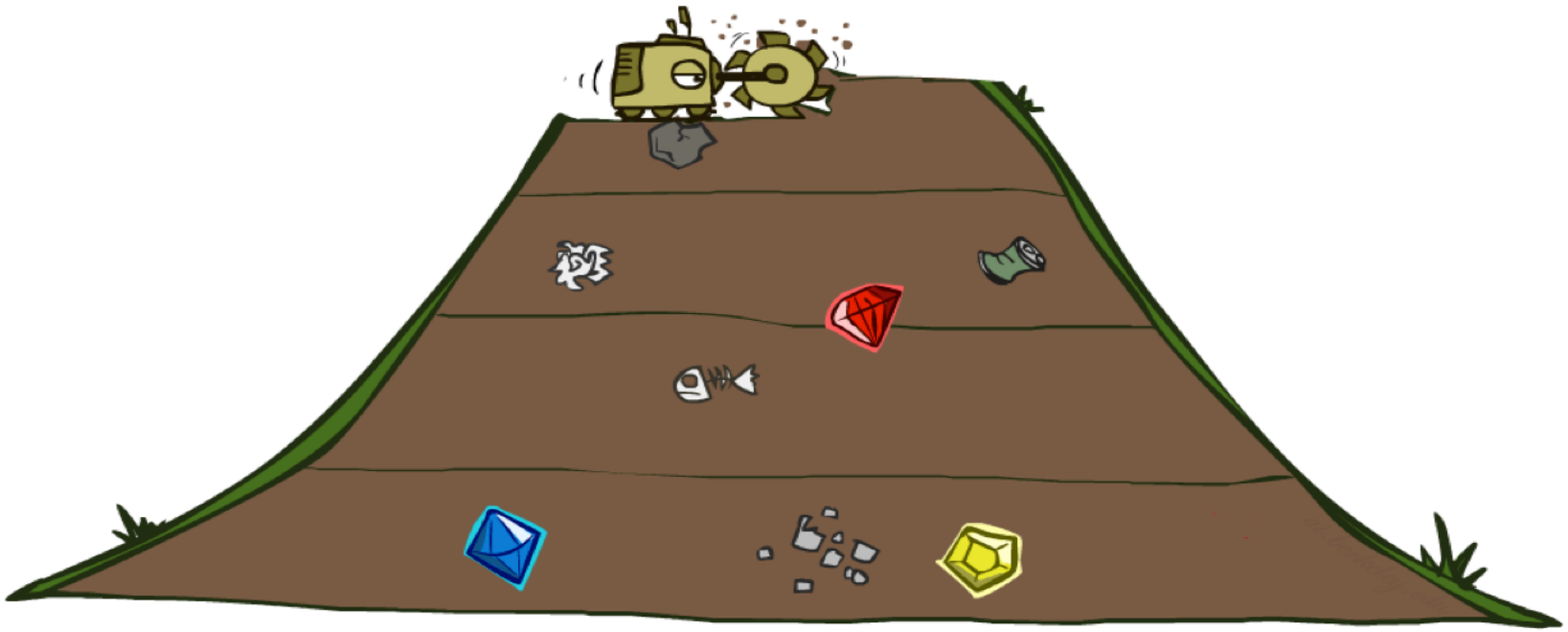
- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots. b^m = O(b^m)$

Depth-First Search (DFS) Properties

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



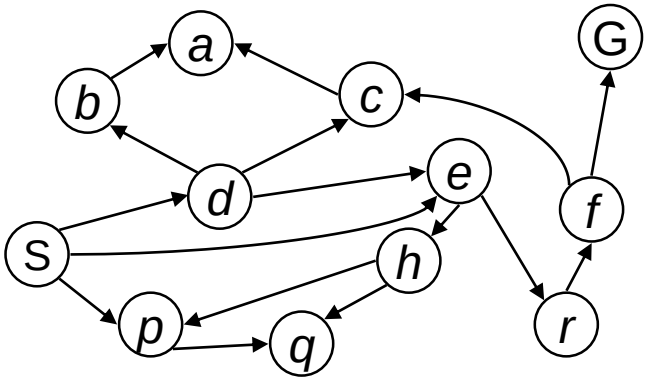
Breadth-First Search

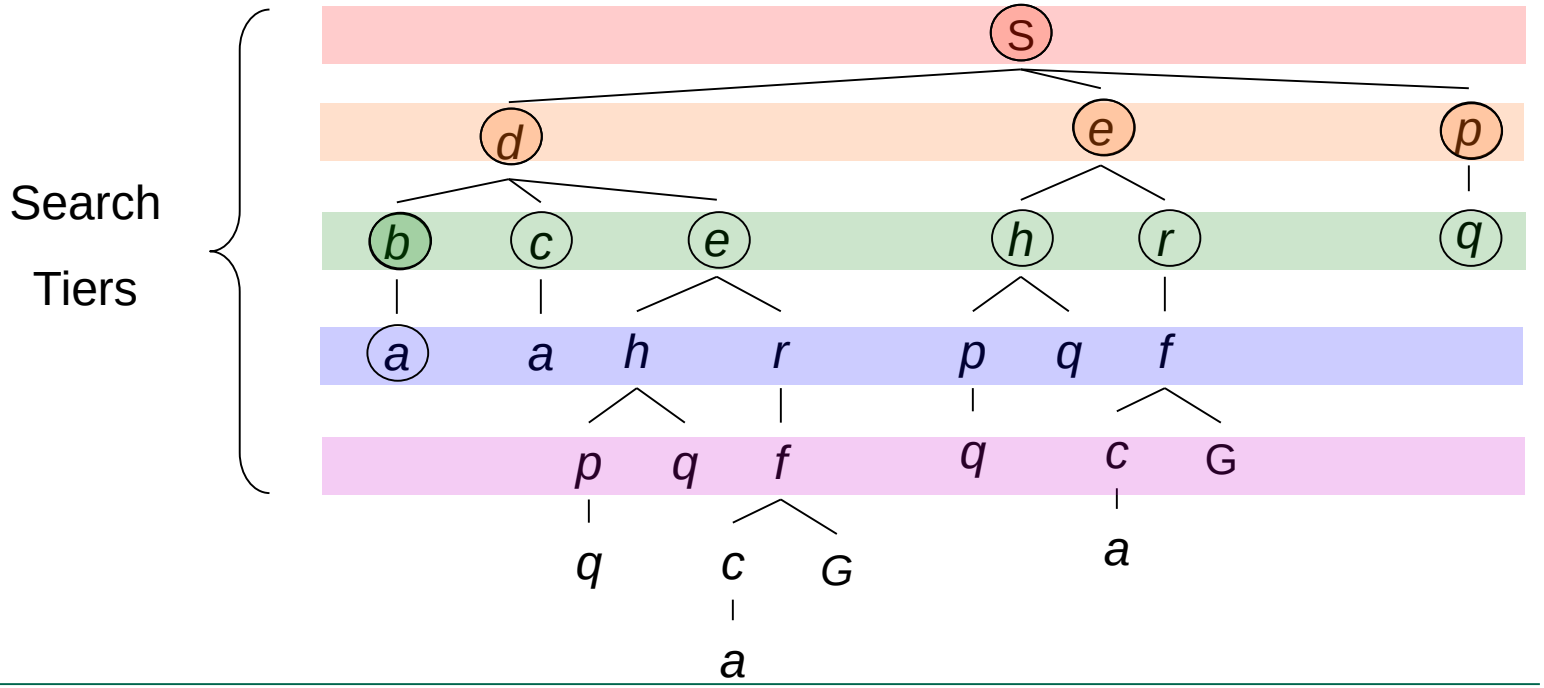


Breadth-First Search

Strategy: expand a shallowest node first

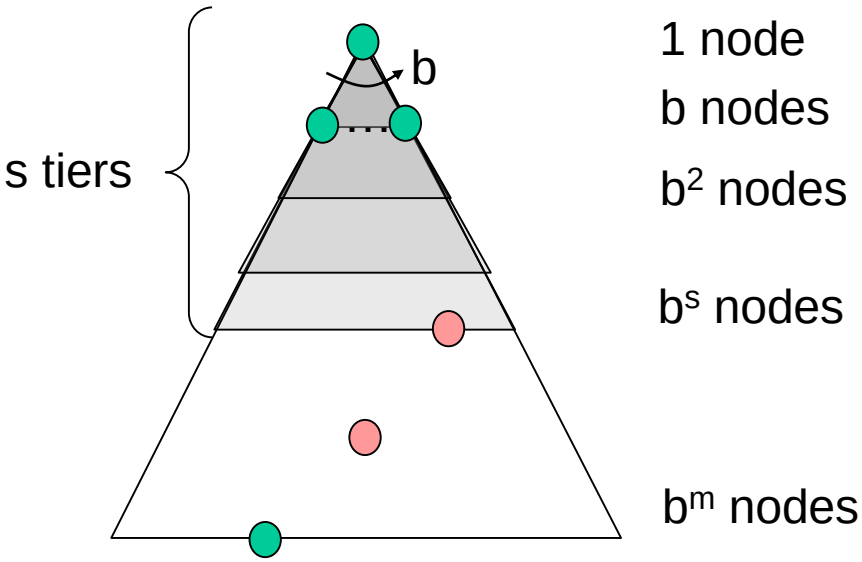
*Implementation: Fringe is a **FIFO queue***



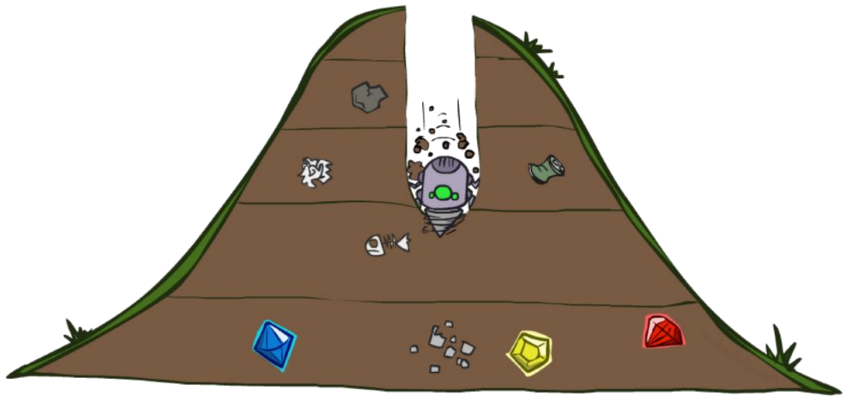


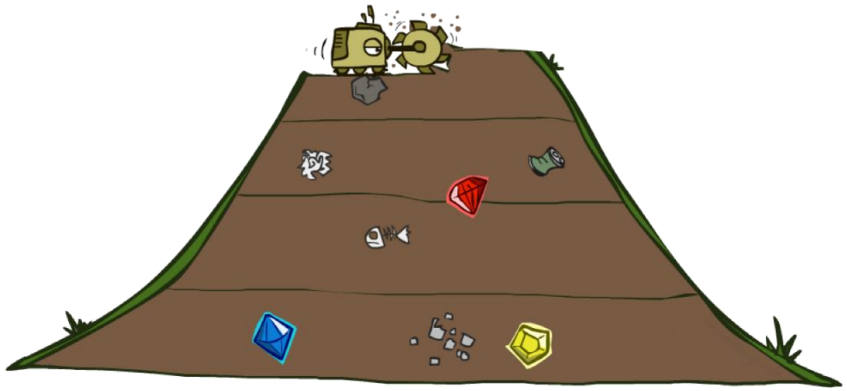
Breadth-First Search (BFS) Properties

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



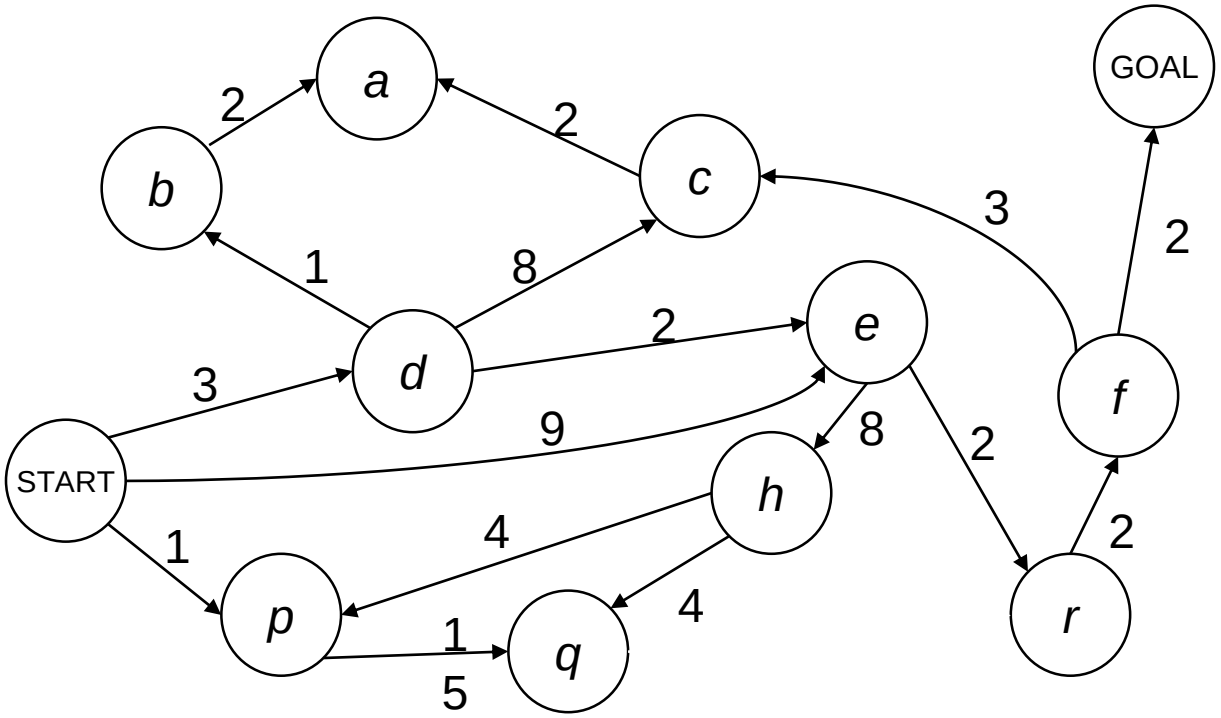
Quiz: DFS vs BFS





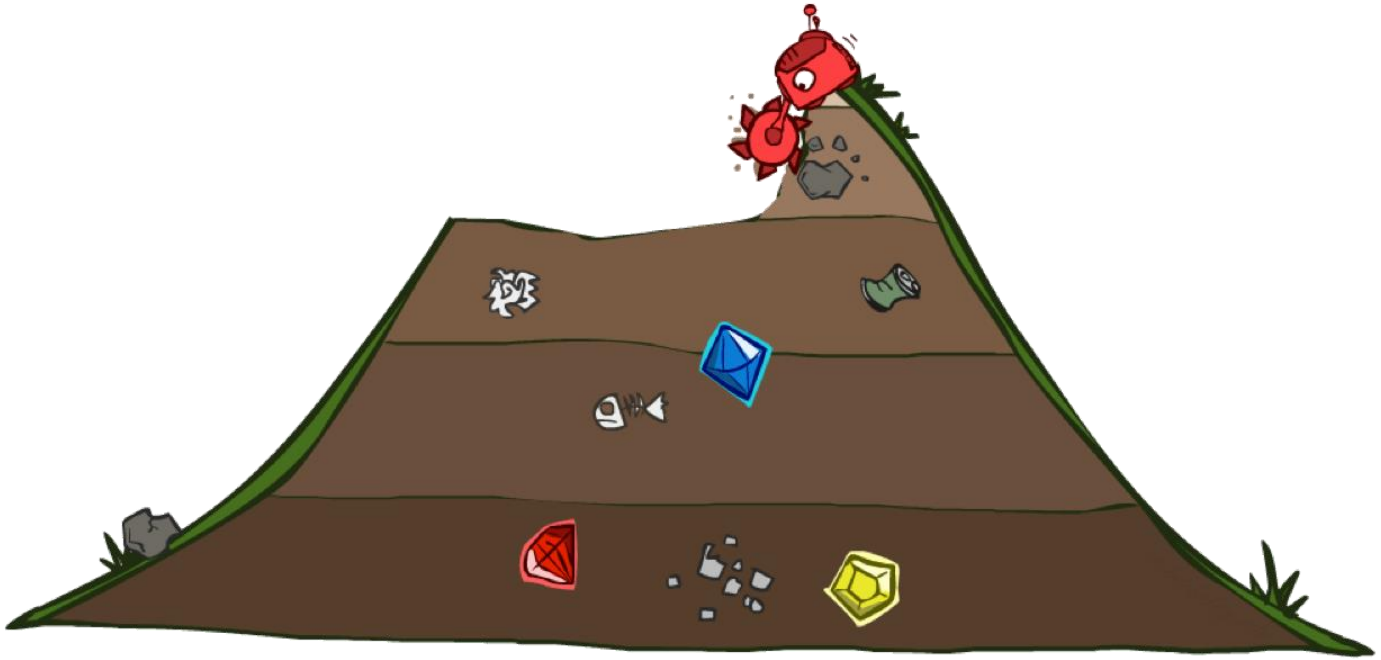
- When will BFS outperform DFS?
- When will DFS outperform BFS?

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.

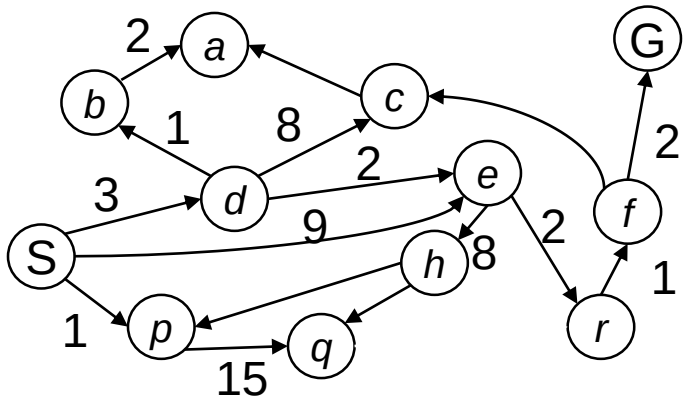
Uniform Cost Search

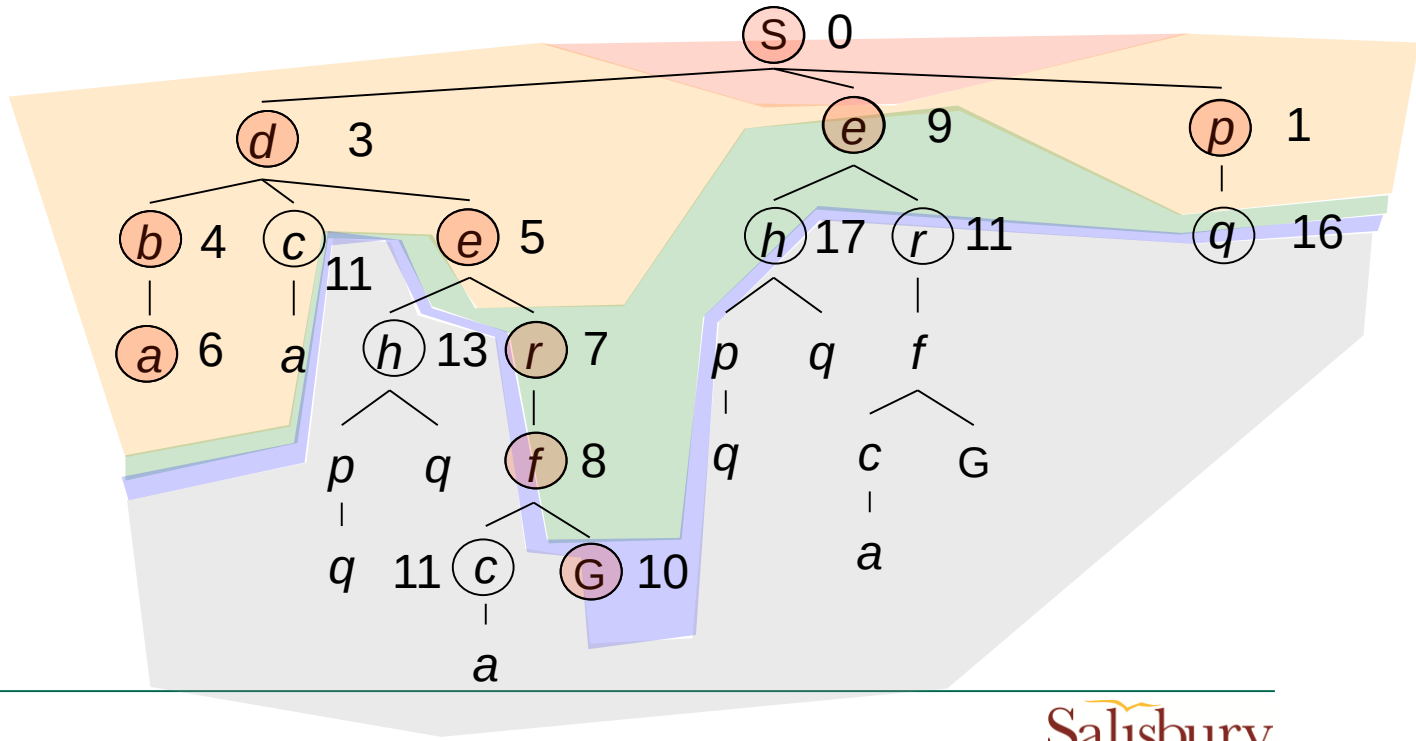


Uniform Cost Search

Strategy: expand a cheapest node first:

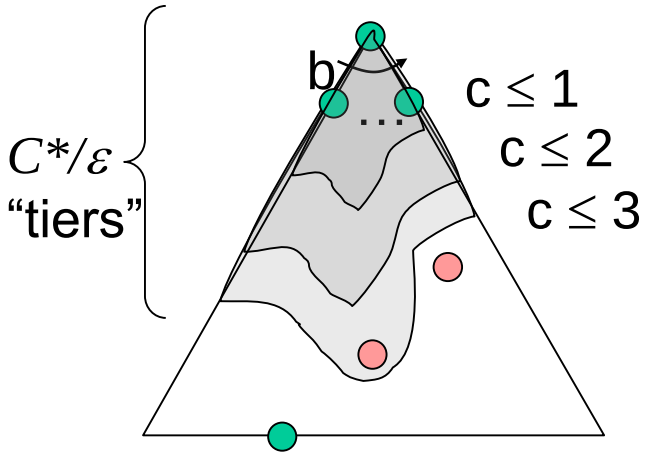
Fringe is a priority queue (priority: cumulative cost)





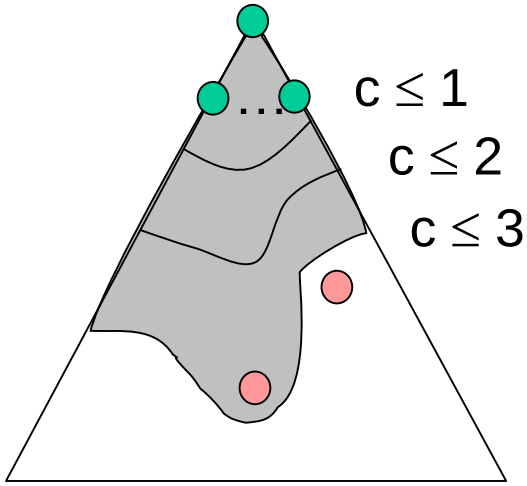
Uniform Cost Search (UCS) Properties

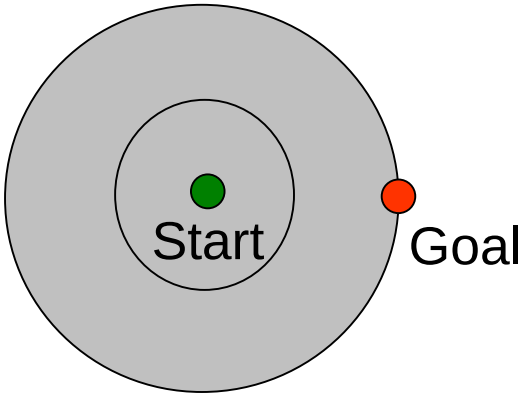
- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!
- Is it optimal?
  - Yes!



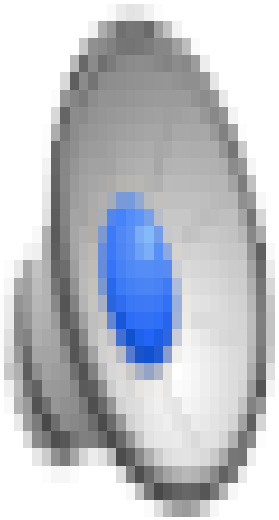
Uniform Cost Issues

- 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

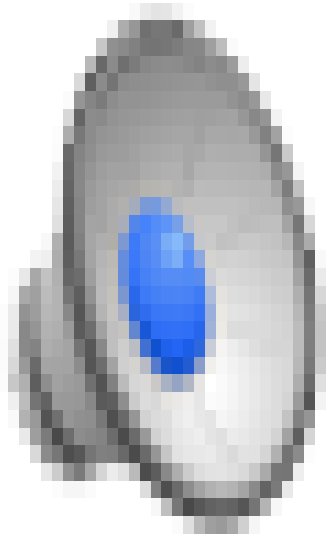




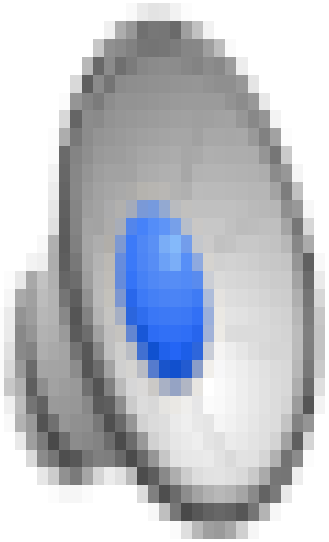
Video of Demo Empty UCS



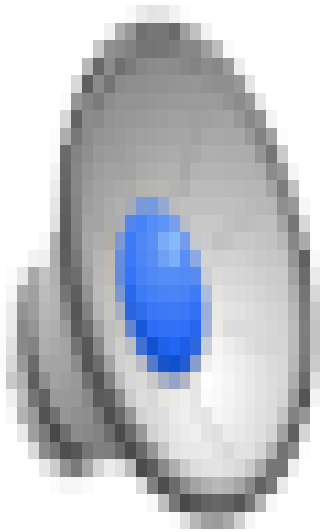
Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 1)



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

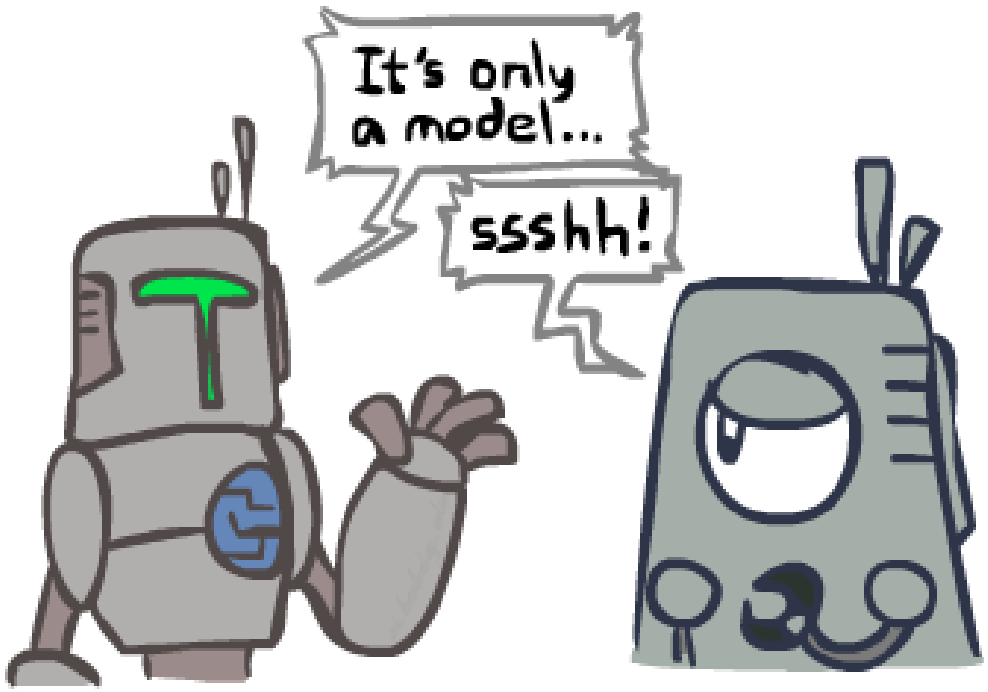


Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 3)

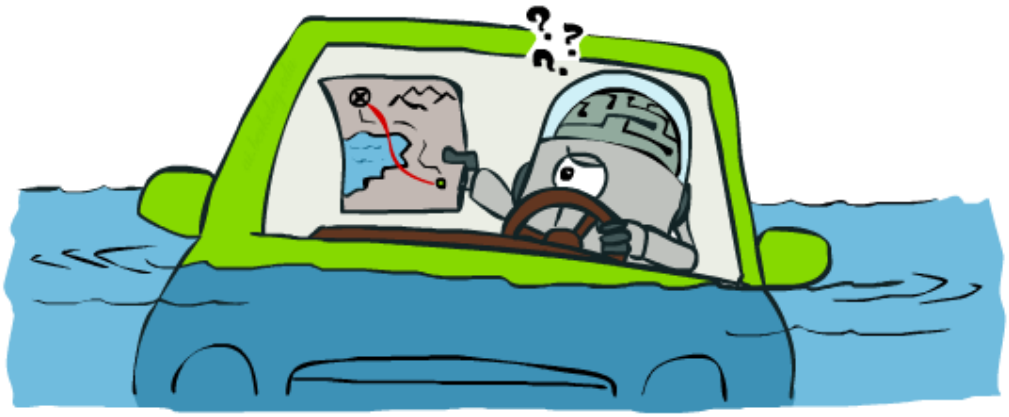


Search and Models

- 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?



Thanks