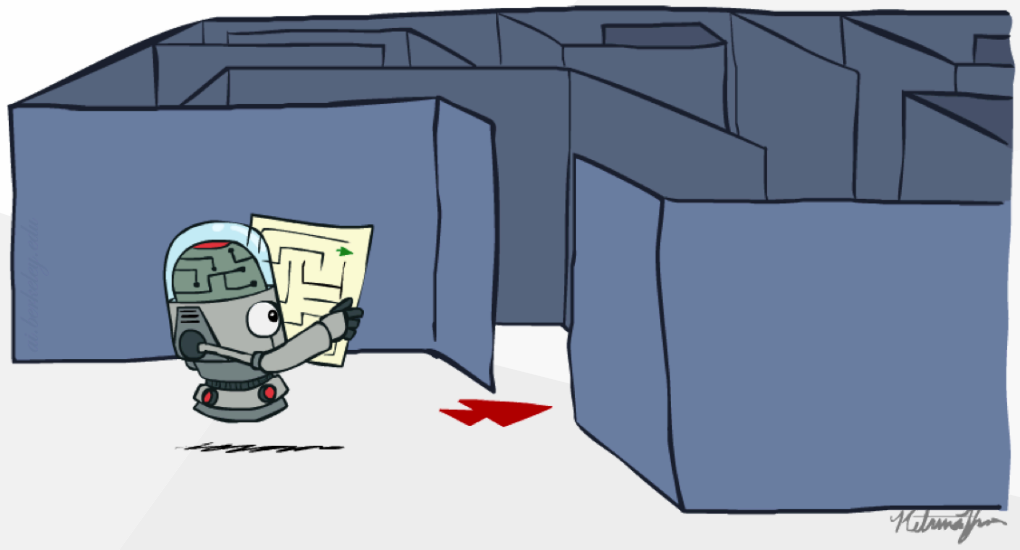
CS 6460: Artificial Intelligence

Search



Instructor: George Rudolph
Utah Valley University Spring 2026

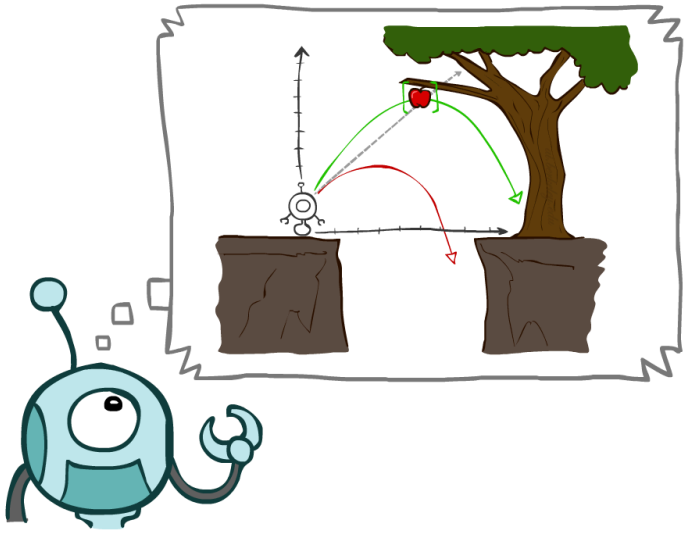
slides adapted from Dan Klein and Pieter Abbeel of UC Berkley

Learning Outcomes

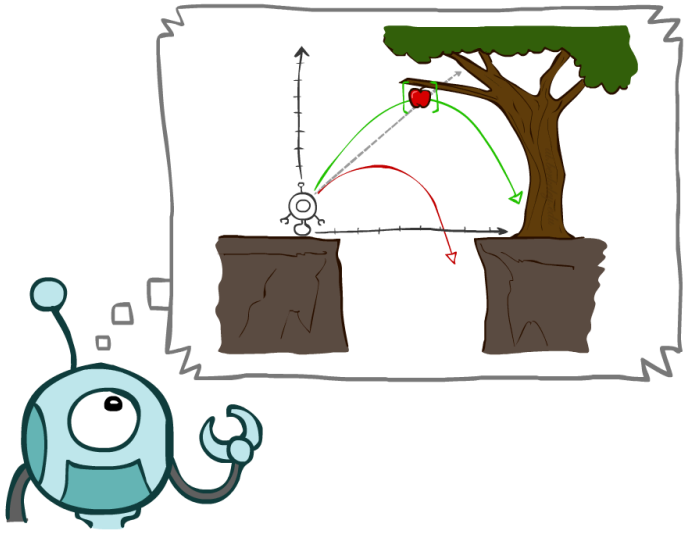
1. Define Agents that Plan Ahead
2. Frame Problems as Search Problems
3. Implement Uninformed Search Algorithms

- Depth-First Search
- Breadth-First Search
- Uniform-Cost Search

1. Analyze Uninformed Search Algorithms using Big-O complexity

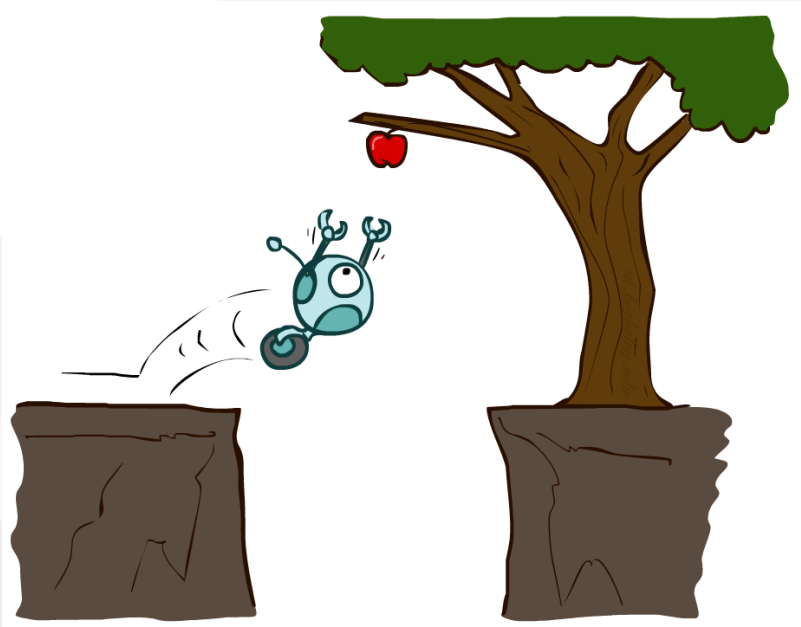


Agents that Plan



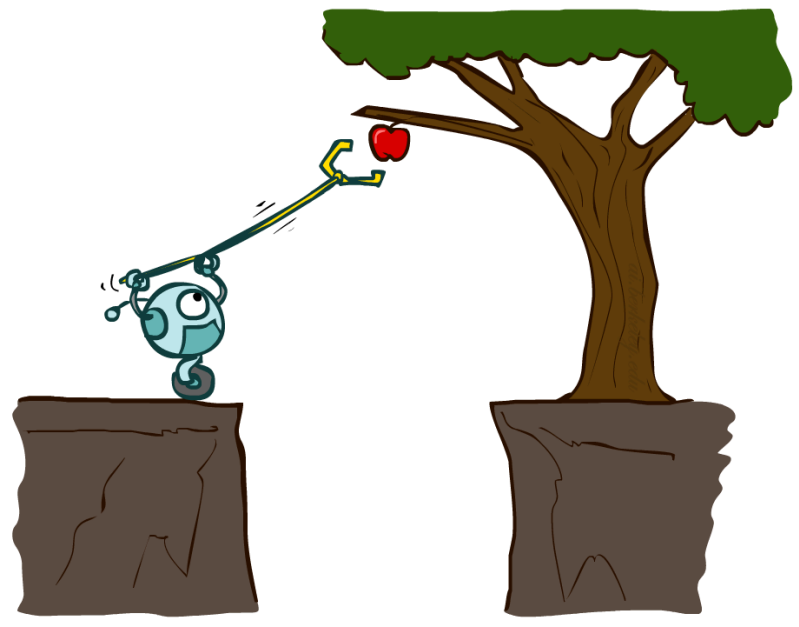
Reflex Agents

- 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?

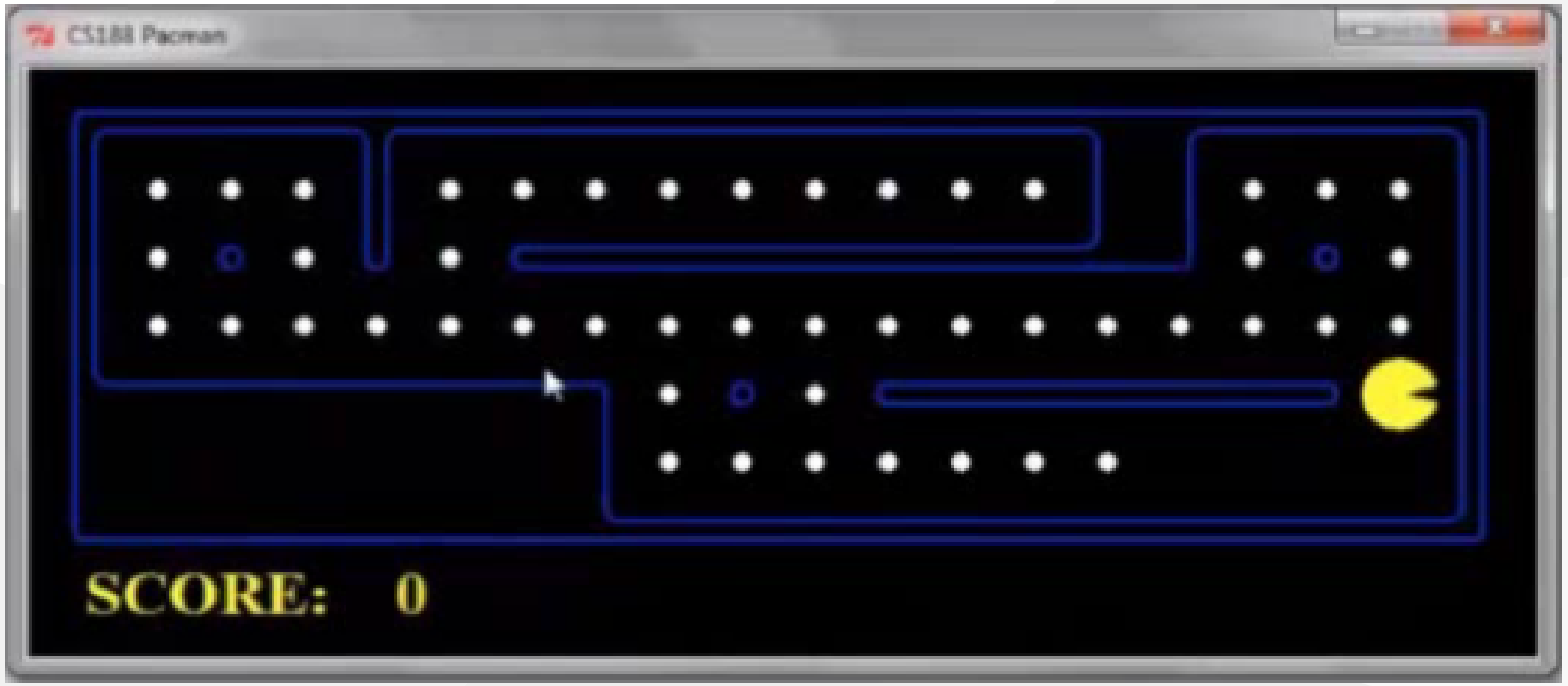


Planning Agents

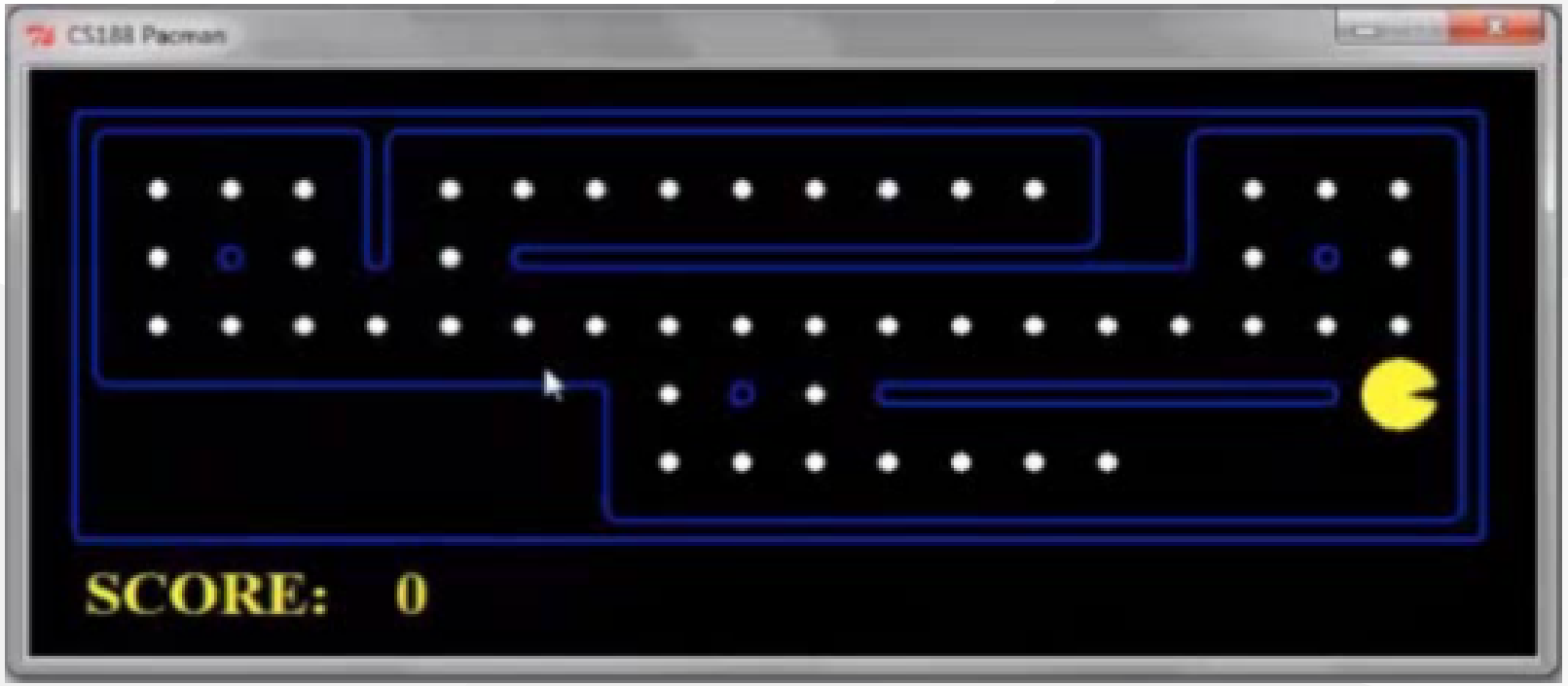
- Planning agents:
  - Ask “what if”
  - Decisions based on (hypothesized) consequences of actions
  - Must have a model of how the world evolves in response to actions
  - Must formulate a goal (test)
  - Consider how the world WOULD BE
- Optimal vs. complete planning
- Planning vs. replanning



Move to nearest dot and eat it



Precompute optimal plan, execute it

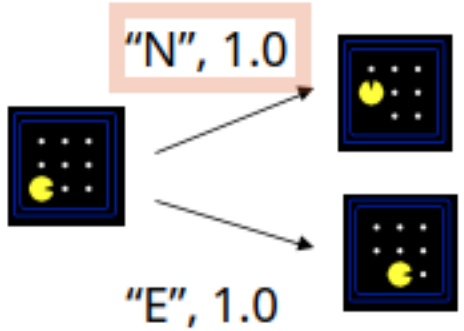


Search Problems

Framing of Search Problem

- A search problem consists of:
  - A state space

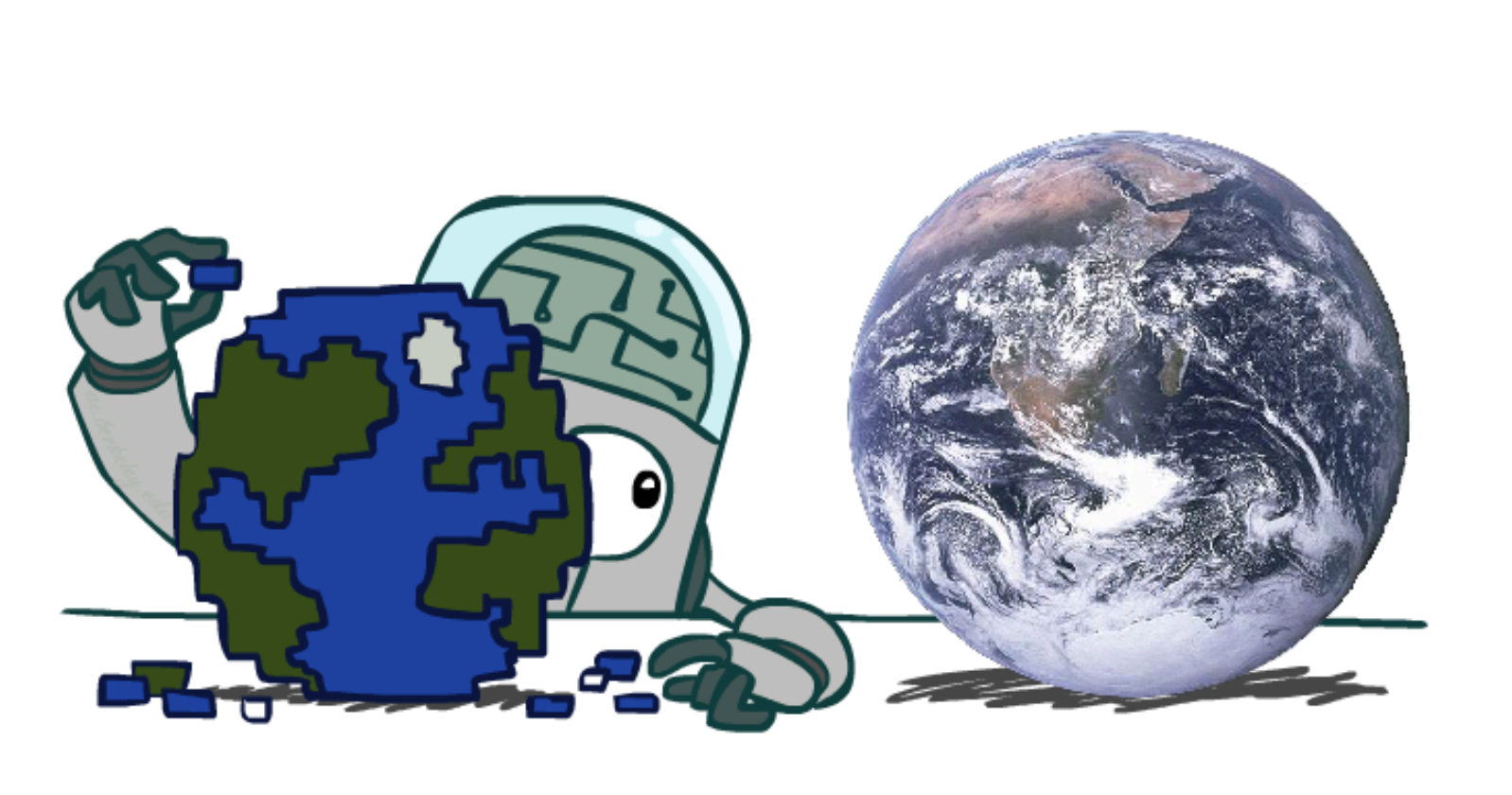
  - A successor function (with actions, costs)



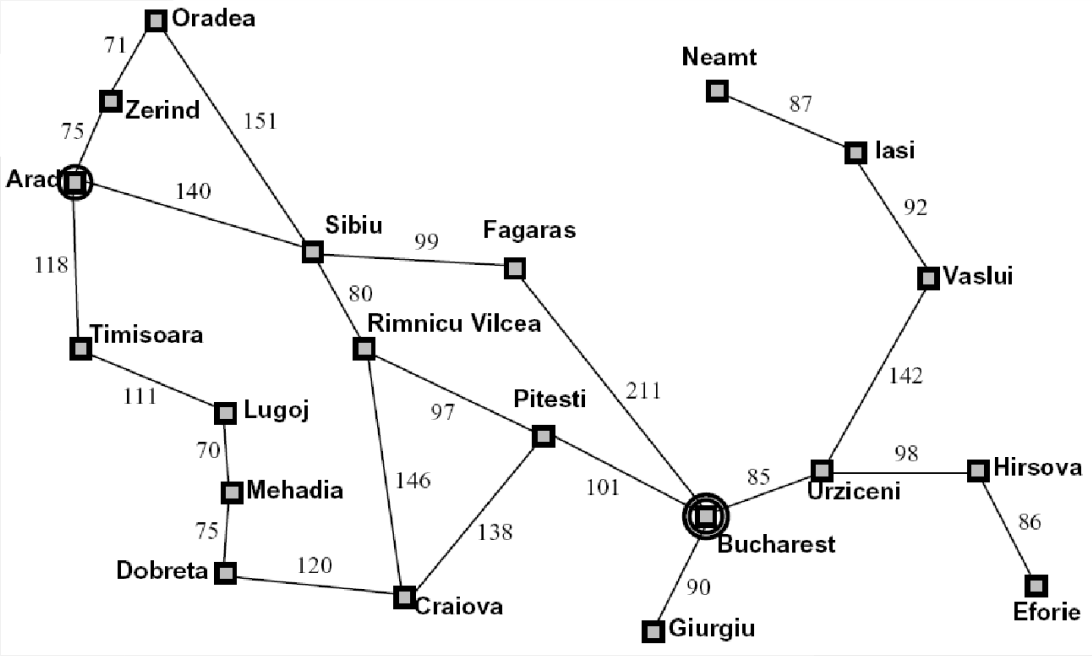
  - A start state and a goal test

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

Search Problems Are Models



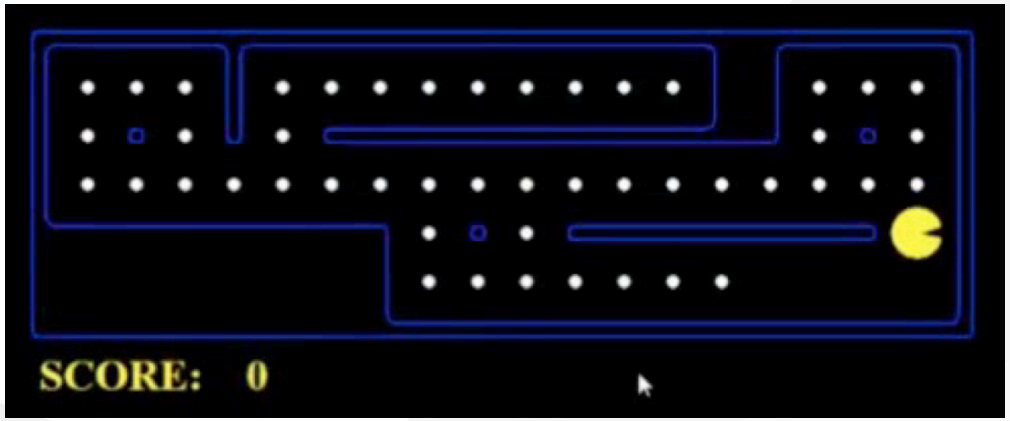
Example: Traveling in Romania



- 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 detail of the environment



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

Problem: Path Planning

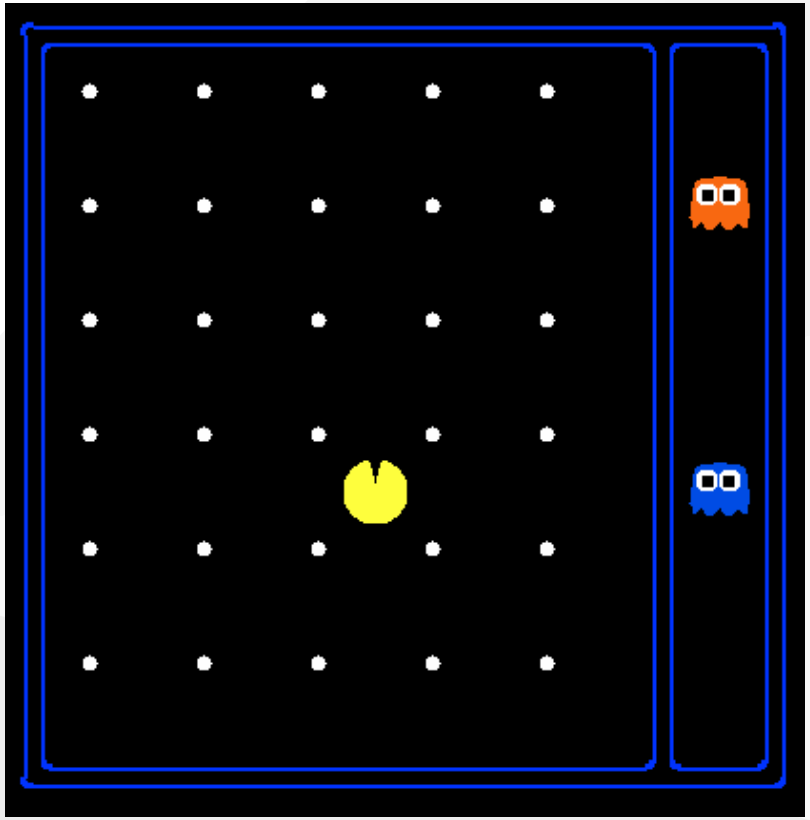
- States: (x, y) location
- Actions: NSEW
- Successor: update location only
- Goal test: is (x,y)=END

Problem: Eat-All-Dots

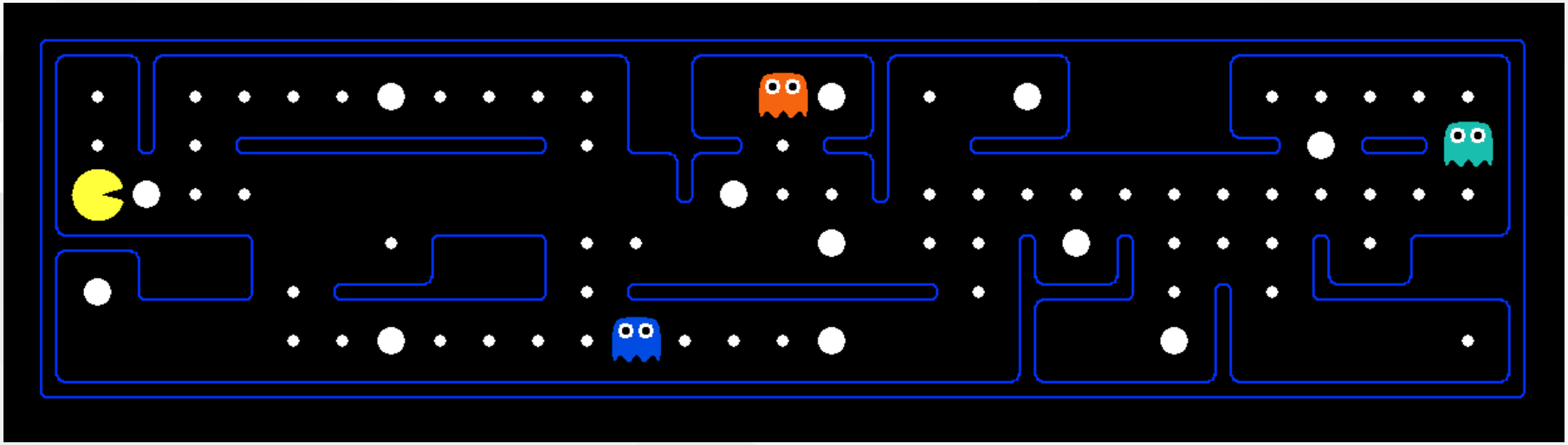
- States: {(x,y), dot booleans}
- Actions: NSEW
- Successor: update location and possibly a dot boolean
- Goal test: dots all false

State Space Sizes?

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



Quiz: Safe Passage



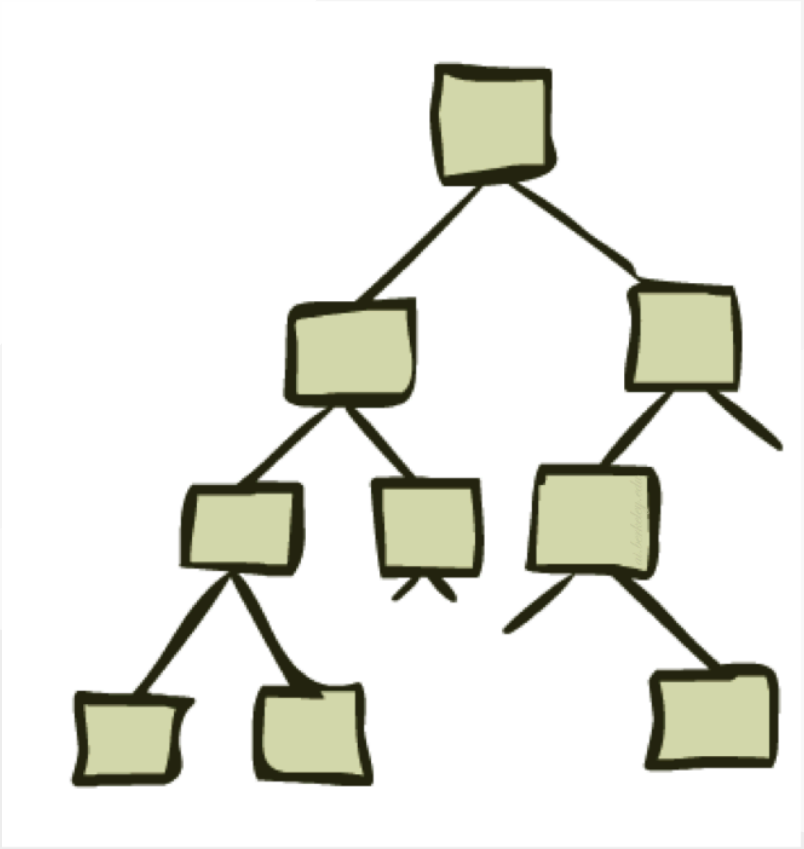
Problem: eat all dots while keeping the ghosts perma-scared

What does the state space have to specify?

- agent position, dot booleans, power pellet booleans, remaining scared time

State Space Graphs and Search Trees

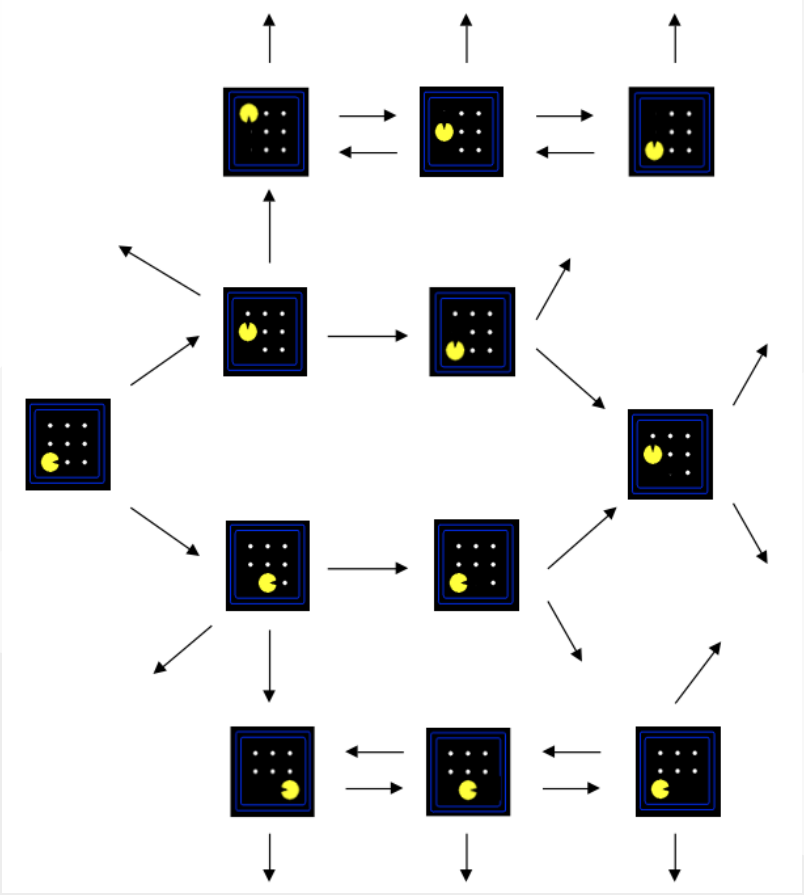
- Each branch is a possible choice
- Each node is a possible state
- Same state may occur in several branches



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!

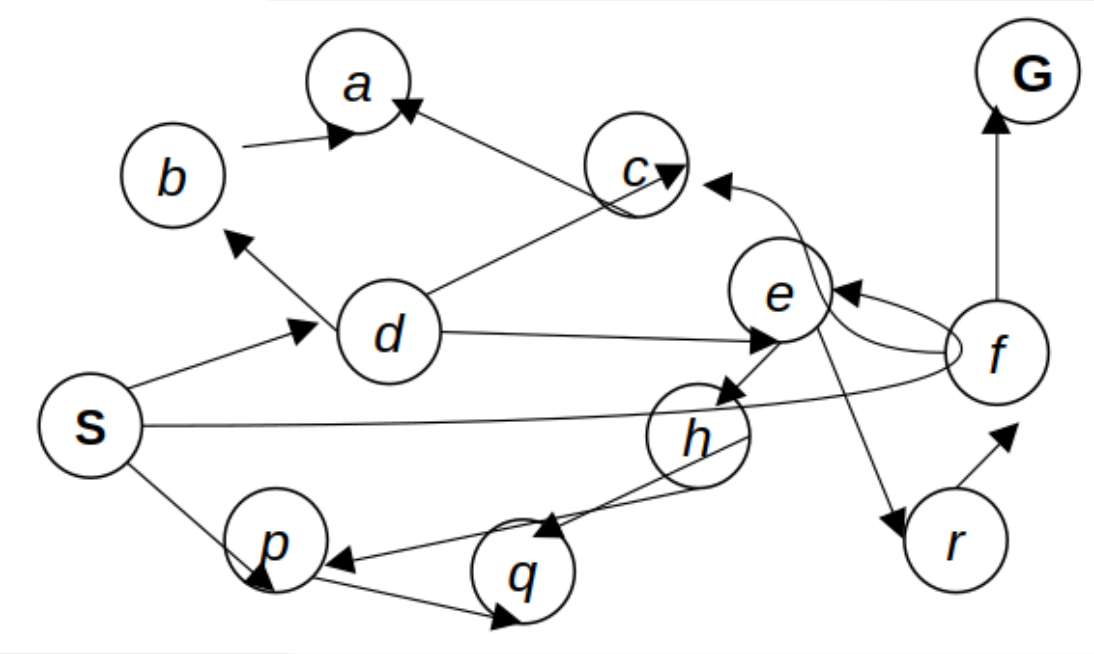
We can rarely build this full graph in memory (it's too big), but it's a useful idea



An Abstract Example

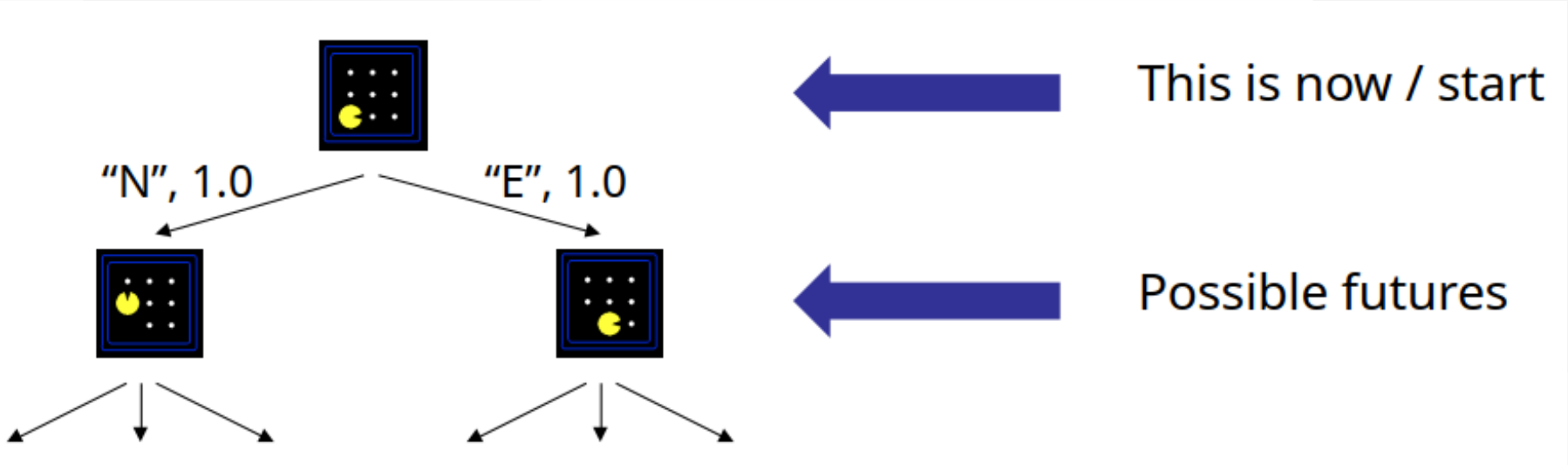
- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!

We can rarely build this full graph in memory (it's too big), but it's a useful idea



Tiny state space graph for a tiny search problem

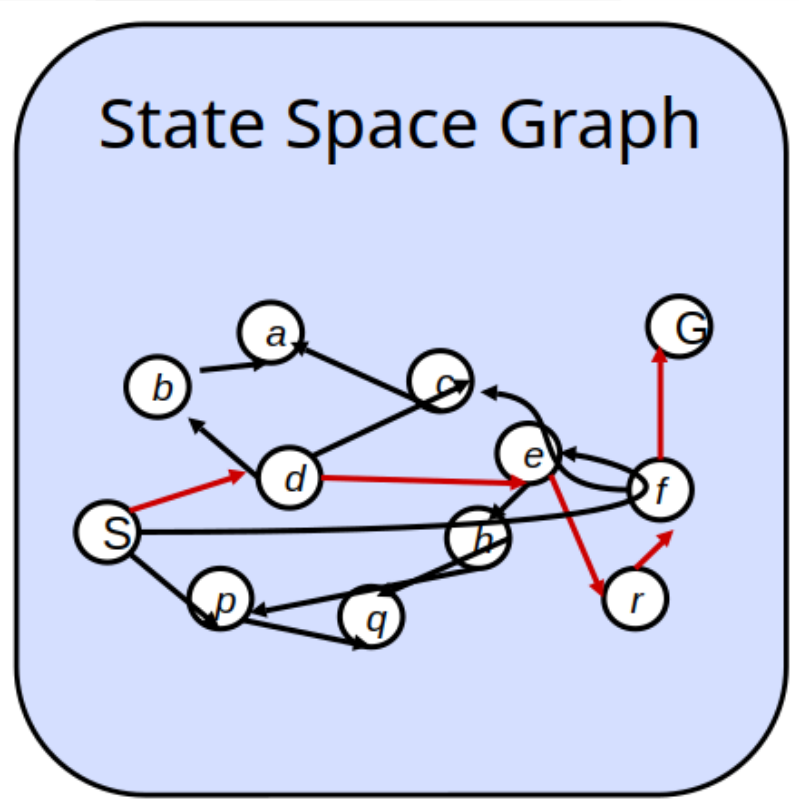
Search Trees

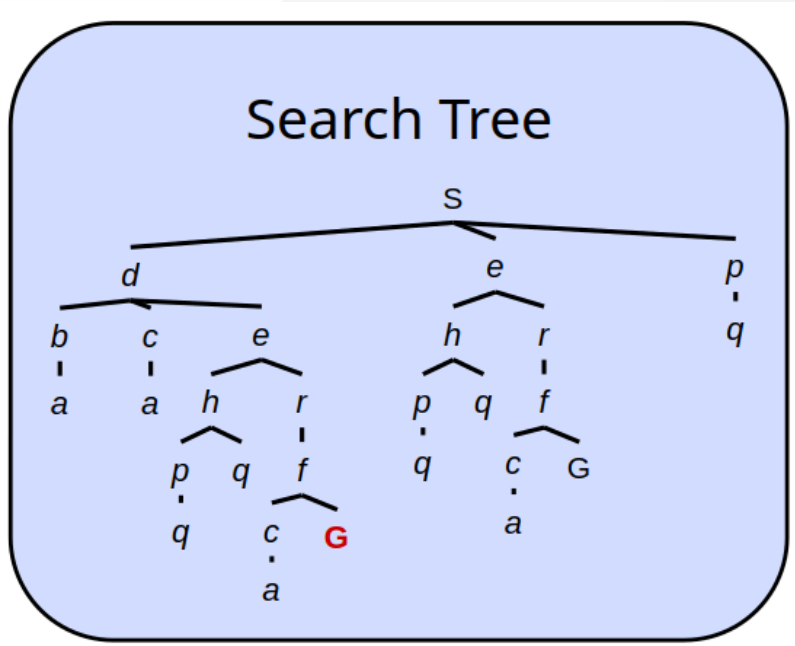


- A search tree:
  - A "what if" tree of plans and their outcomes
  - The start state is the root node
  - Children correspond to successors
  - Nodes show states, but correspond to PLANS that achieve those states

For most problems, we can never actually build the whole tree

State Space Graphs vs. Search Trees

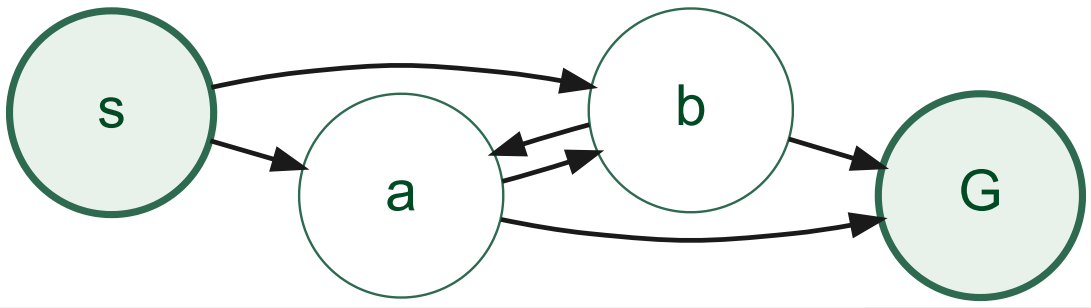




- Each NODE in in the search tree is an entire PATH in the state space graph.
- We construct graph and search on demand, and we construct as little as possible
  - lazy construction (could use generator…)

State Graph Size vs Search Space Size

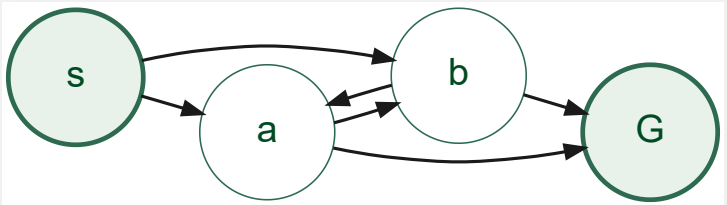
Consider this 4-state graph.



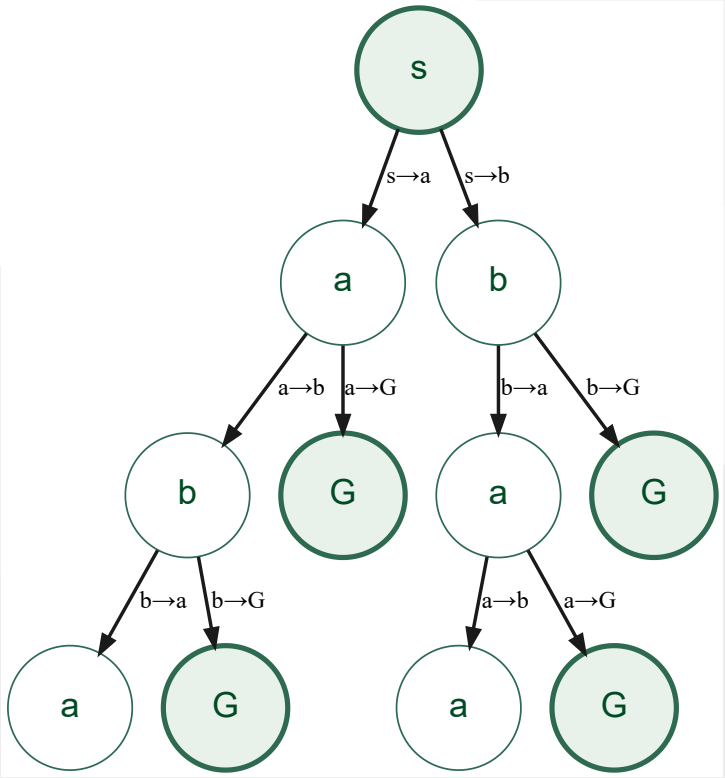
How big is its search tree (from S)?

Repeated Structure in Search Tree

Consider this 4-state graph.

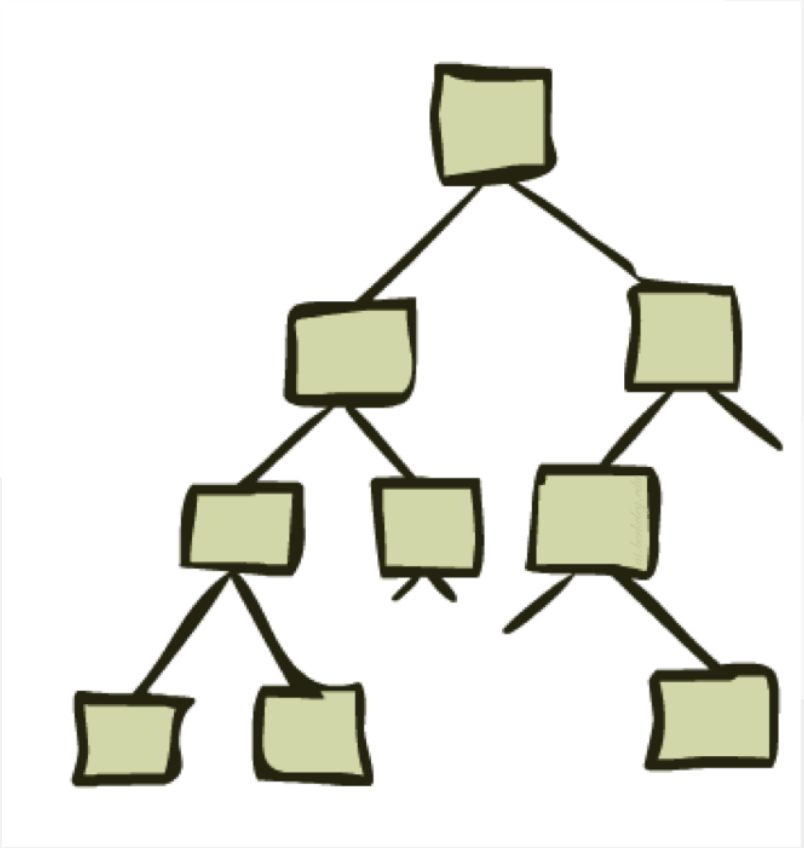


First 4 levels of search tree

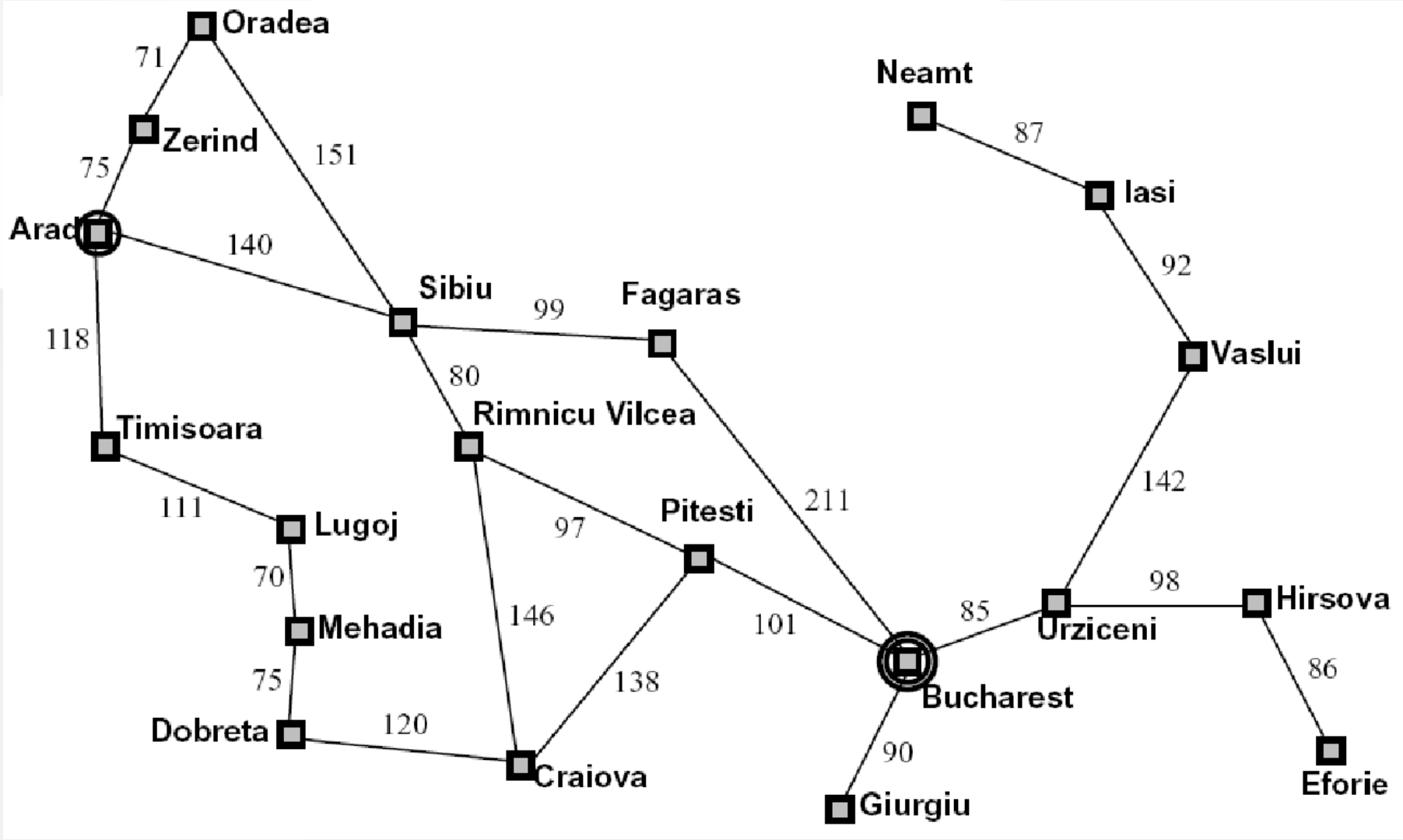


How big is its search tree (from S)?

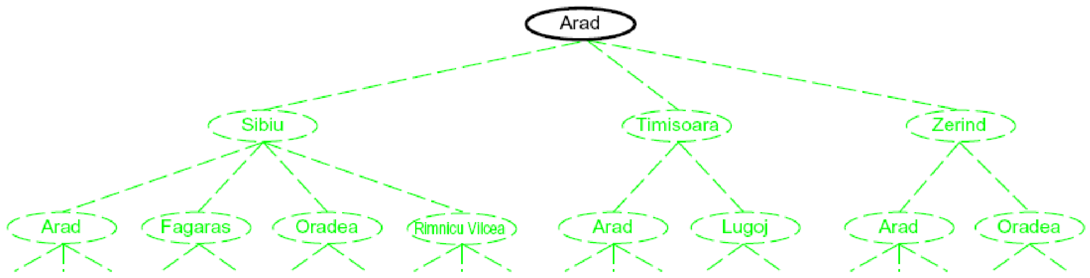
Tree Search

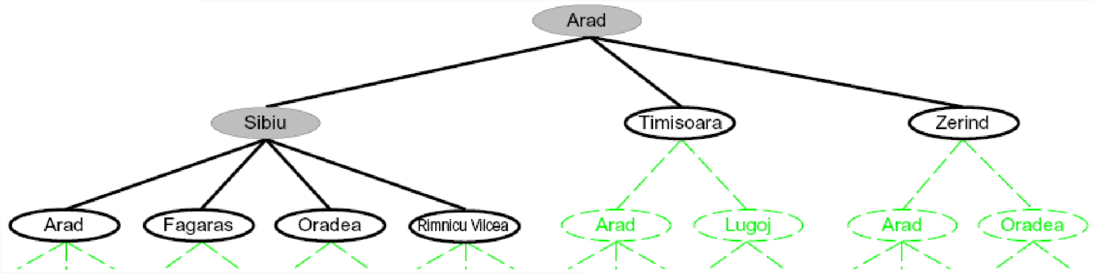


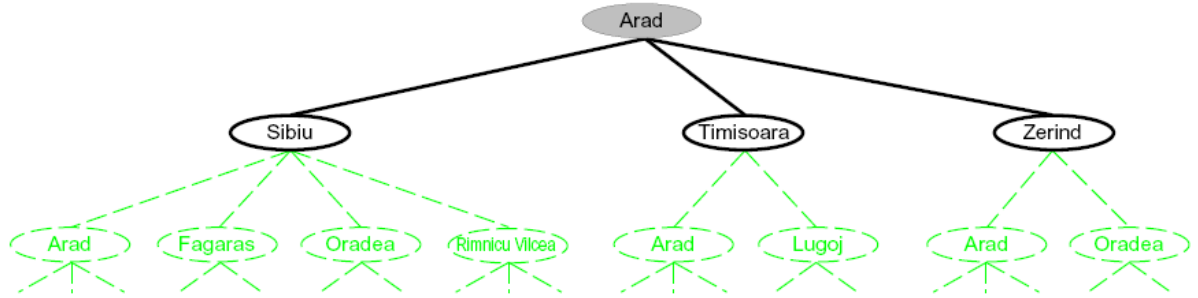
Search Example: Navigate Romania



Searching with a Search Tree







Search Algorithm

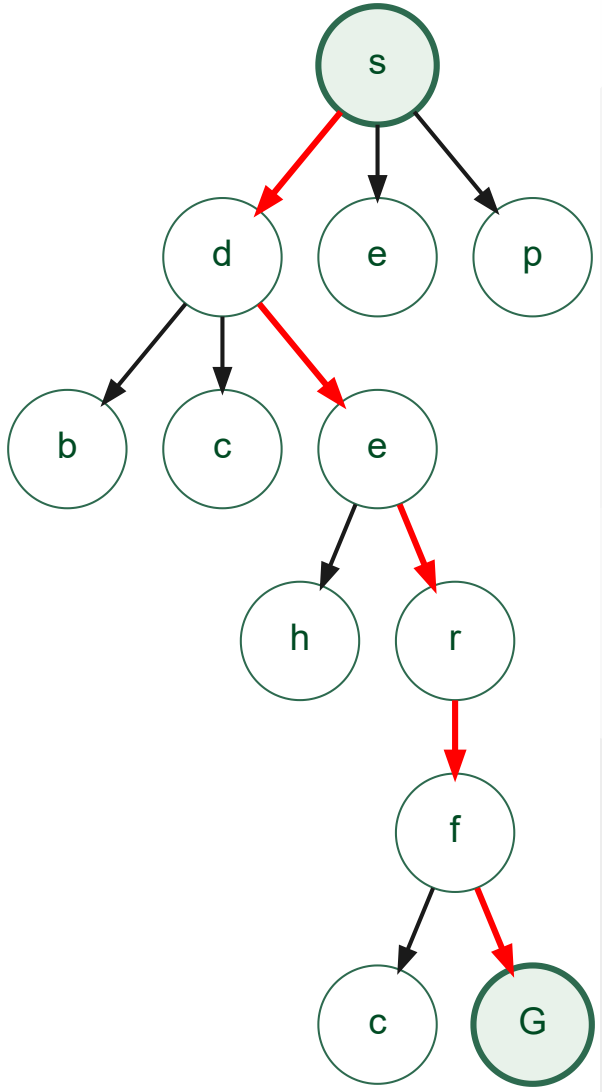
1. Expand out potential plans (tree nodes)
2. Maintain a fringe of partial plans under consideration
3. Try to expand as few tree nodes as possible

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?

Example: Tree Search



Depth-First Search

DFS

Idea: Full speed ahead in one direction until you can't go further

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack

Try It: DFS Water Maze Simulation

Run the DFS simulation:

Linux/Mac:

```
cd code
./run_dfs_simulation.sh
```

Windows:

```
cd code
run_dfs_simulation.bat
```

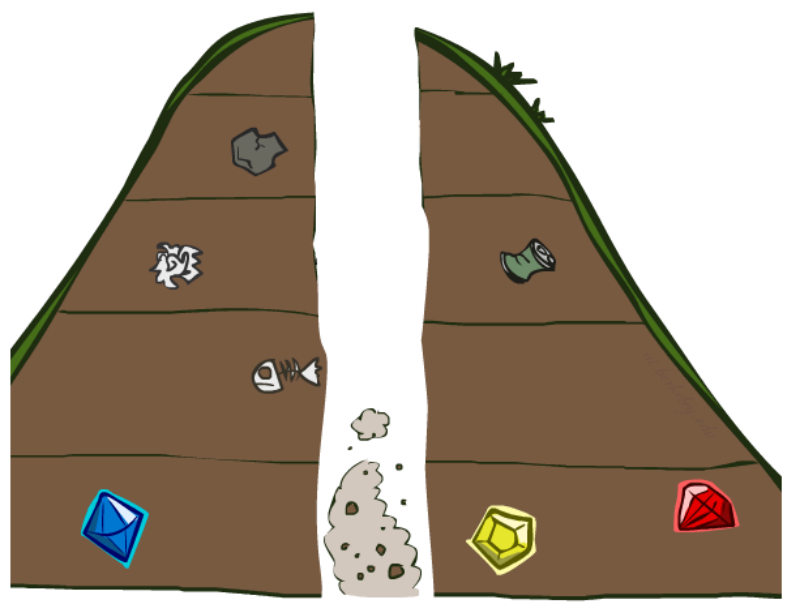
Or manually:

```
cd code/dfs_water
python3 dfs_water_maze.py
```

Requirements: Python 3.13+ with numpy and matplotlib

Watch water spread through the maze using DFS - it goes deep first!

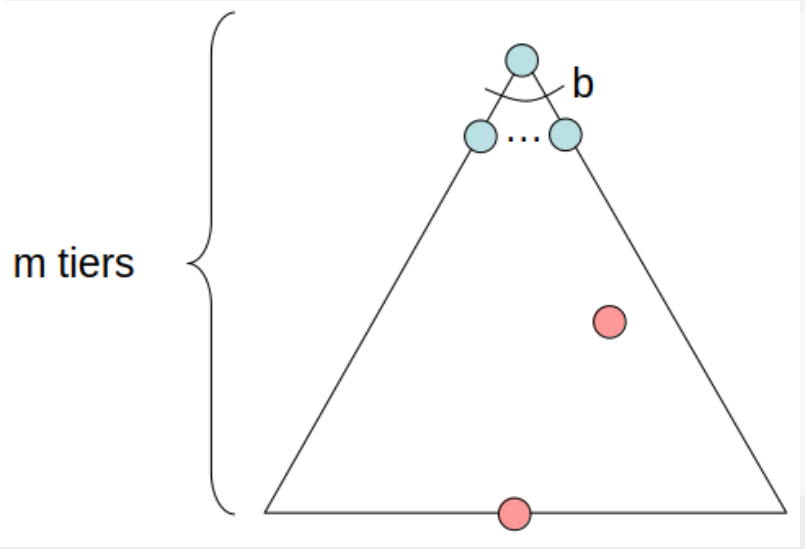
Search Algorithm Properties



Properties

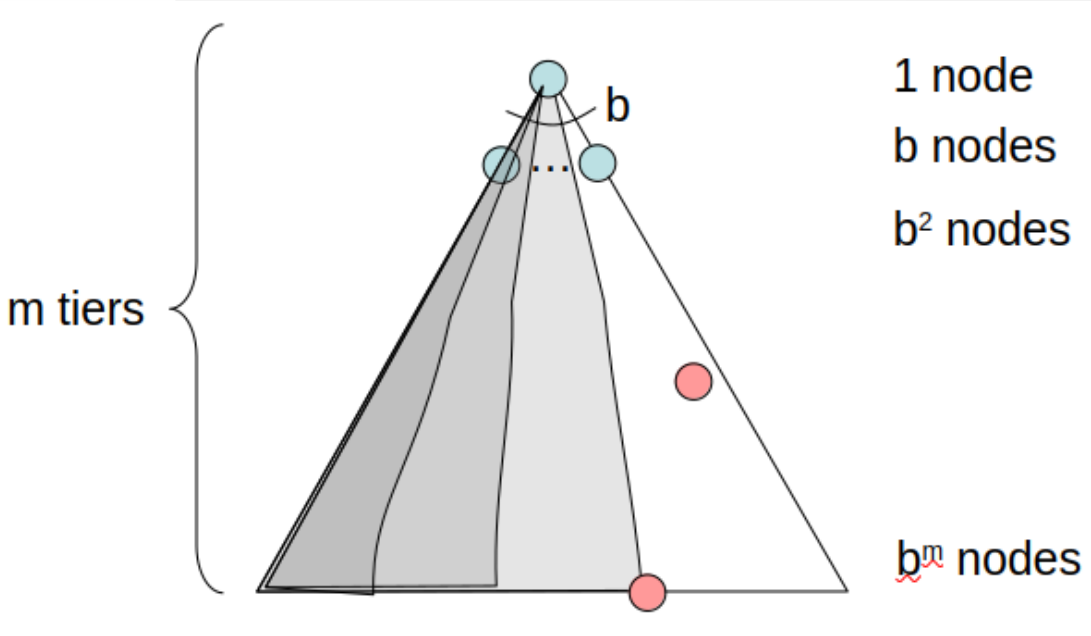
- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths s
- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots + b^m = O(b^m)$

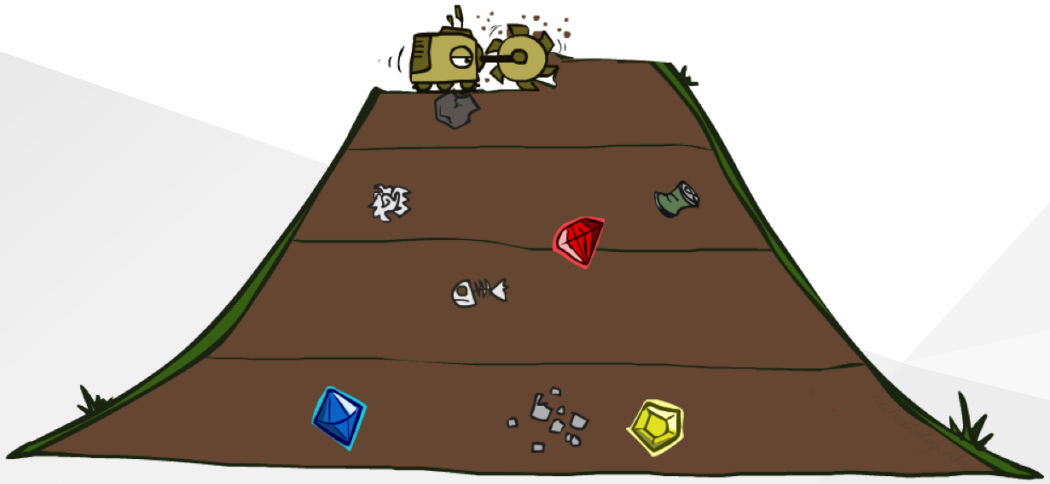


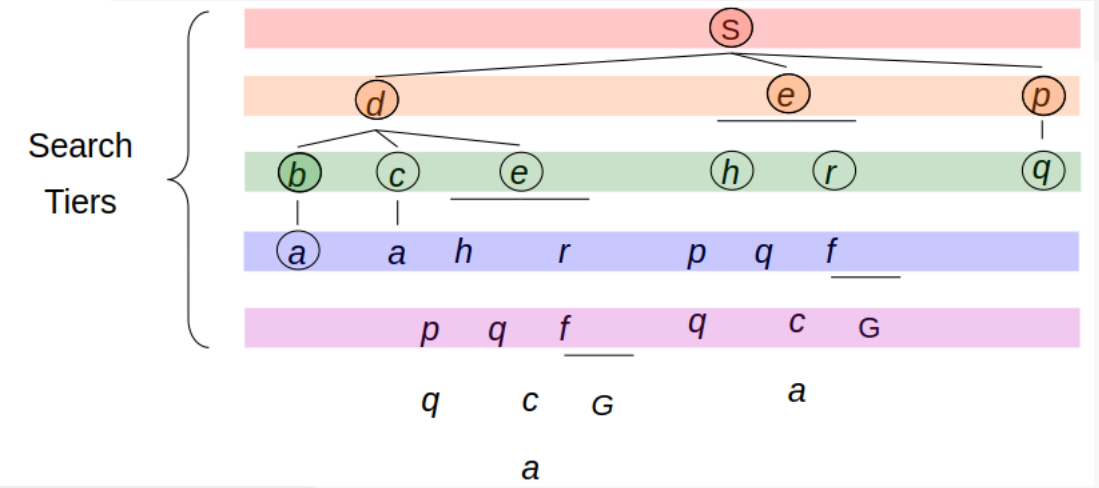
Even More DFS Properties

- What nodes does DFS expand?
  - Some left prefix of the tree
  - Could process whole tree
  - If m is finite, takes $O(b^m)$ time
- How much space does fringe take?
  - only has siblings on path to root
  - $O(bm)$
- Is it complete?
  - m could be infinite...
  - only if we prevent cycles
- Is it optimal?
  - No
  - Finds leftmost solution
  - Does not consider cost or depth



Breadth-First Search (BFS)





Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue

Try It: BFS Water Maze Simulation

Run the BFS simulation:

Linux/Mac:

```
cd code
./run_bfs_simulation.sh
```

Windows:

```
cd code
run_bfs_simulation.bat
```

Or manually:

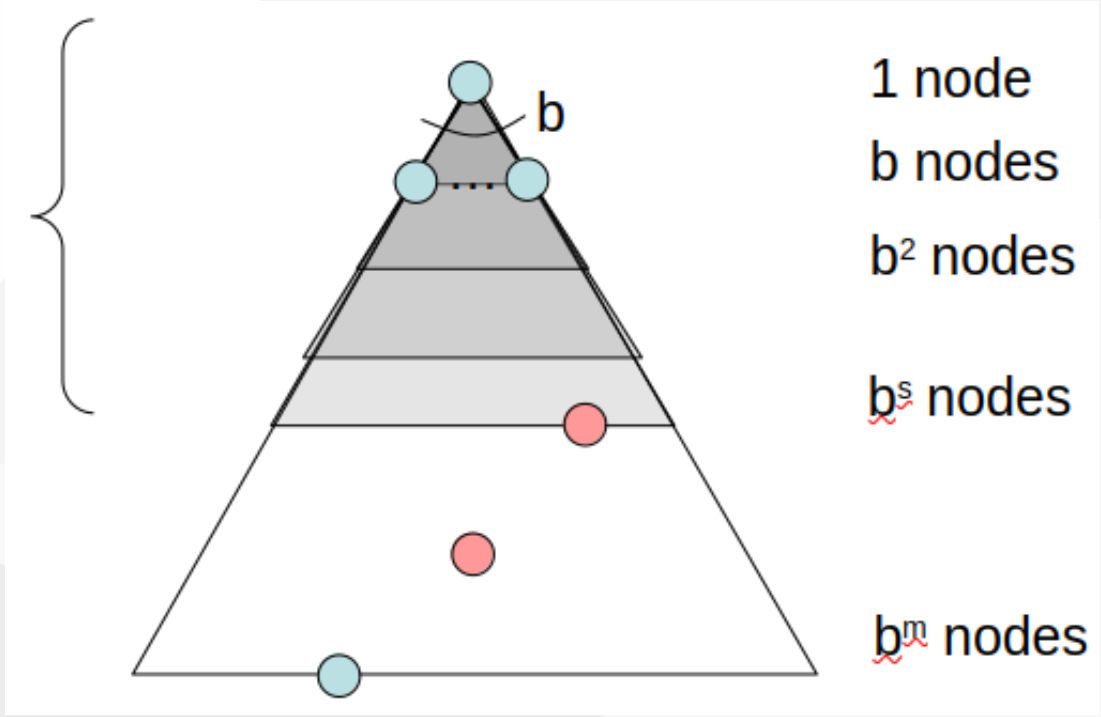
```
cd code/bfs_water
python3 bfs_water_maze.py
```

Requirements: Python 3.13+ with numpy and matplotlib

Watch water spread level by level using BFS - notice the difference from DFS!

BFS Properties

- What nodes does BFS expand?
  - processes all nodes above the shallowest solution s
  - search time is $O(b^s)$
- How much space does fringe take?
  - $O(b^s)$
- Is it complete?
  - If solution exists, s must be finite
  - So yes
- Is it optimal?
  - only if all costs are 1 (or equal)



DFS vs BFS

Sr. No.	Key	BFS	DFS
1	Definition	BFS, stands for Breadth First Search.	DFS, stands for Depth First Search.
2	Data structure	BFS uses Queue to find the shortest path.	DFS uses Stack to find the shortest path.
3	Source	BFS is better when target is closer to Source.	DFS is better when target is far from source.
4	Suitablity for decision tree	As BFS considers all neighbour so it is not suitable for decision tree used in puzzle games.	DFS is more suitable for decision tree. As with one decision, we need to traverse further to augment the decision. If we reach the conclusion, we won.
5	Speed	BFS is slower than DFS.	DFS is faster than BFS.
6	Time Complexity	Time Complexity of BFS = O(V+E) where V is vertices and E is edges.	Time Complexity of DFS is also O(V+E) where V is vertices and E is edges.

Quiz: DFS vs BFS

1. When will BFS outperform DFS?
2. When will DFS outperform BFS?

Example: Maze Water with BFS

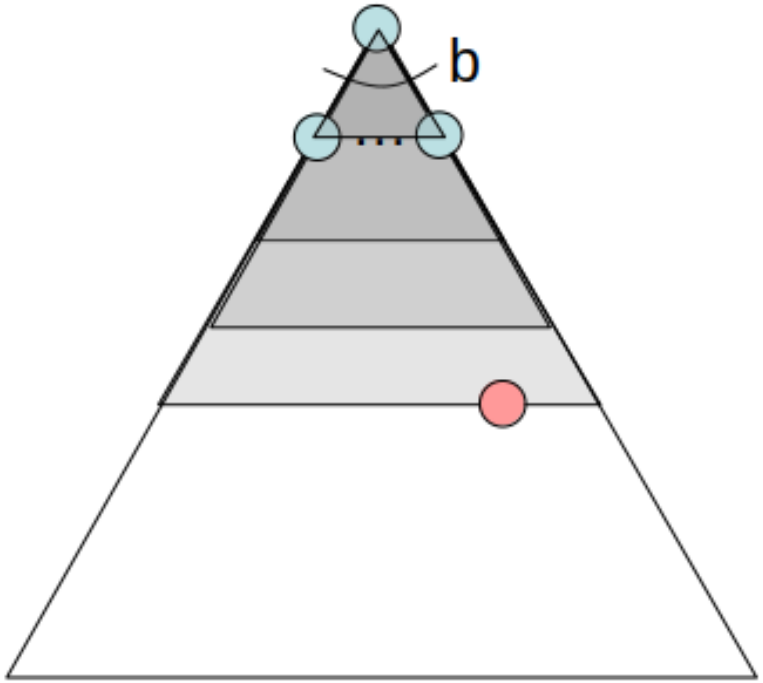
- BFS spreads out in concentric circles from root.
- Explores a lot of the state space

Example: Maze Water with DFS

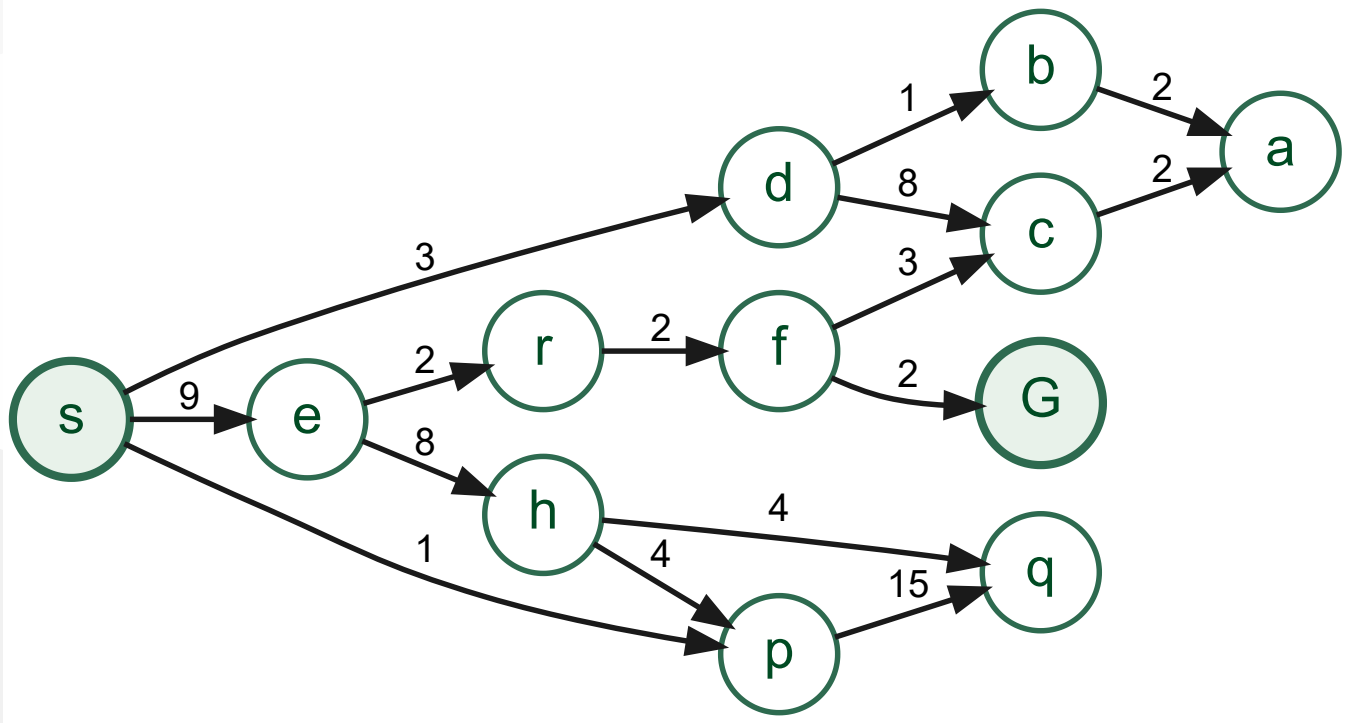
- Explores less but finds a terrible solution.
- It is checking for repeated states
  - Otherwise it would never complete

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

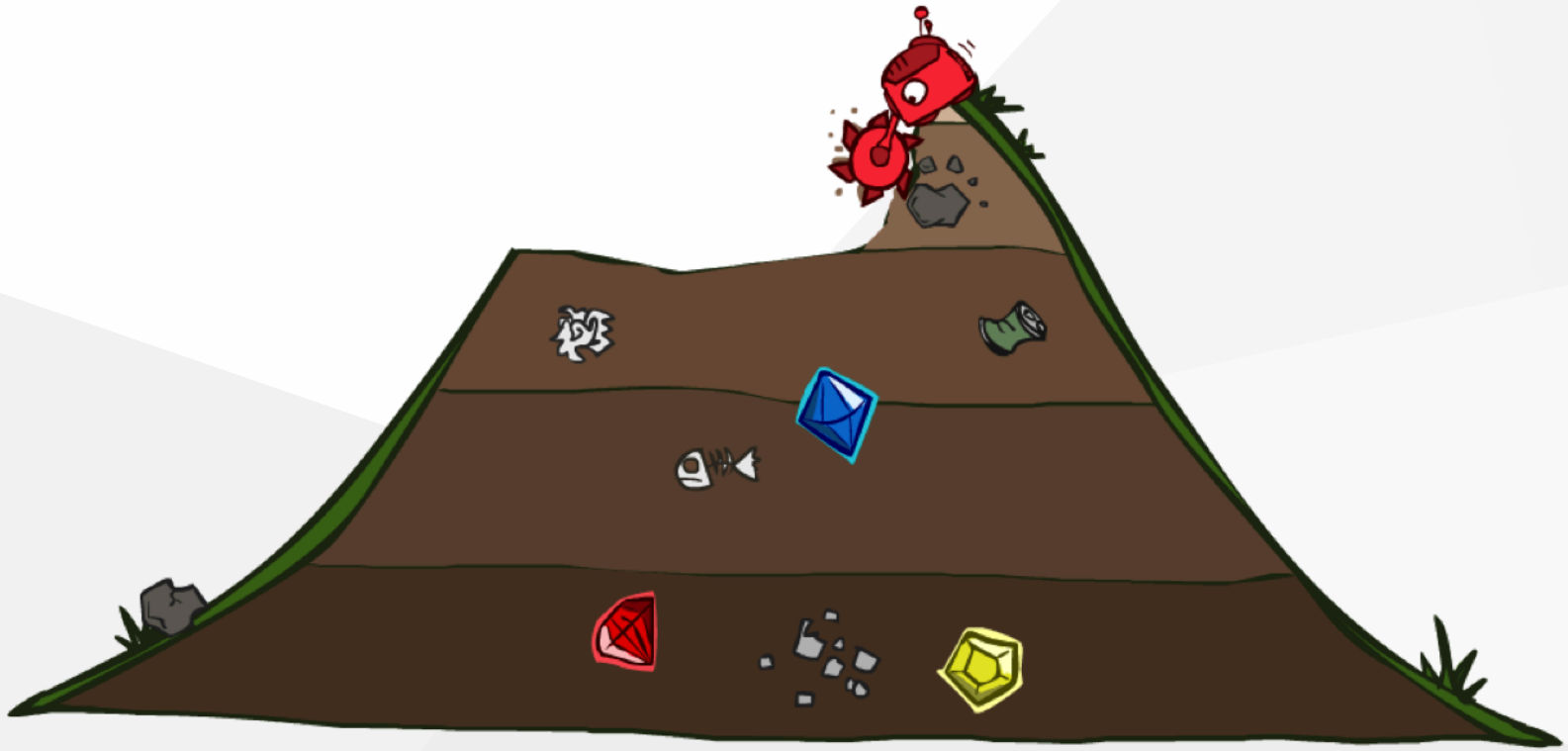


BFS finds the shortest path in terms of number of actions.

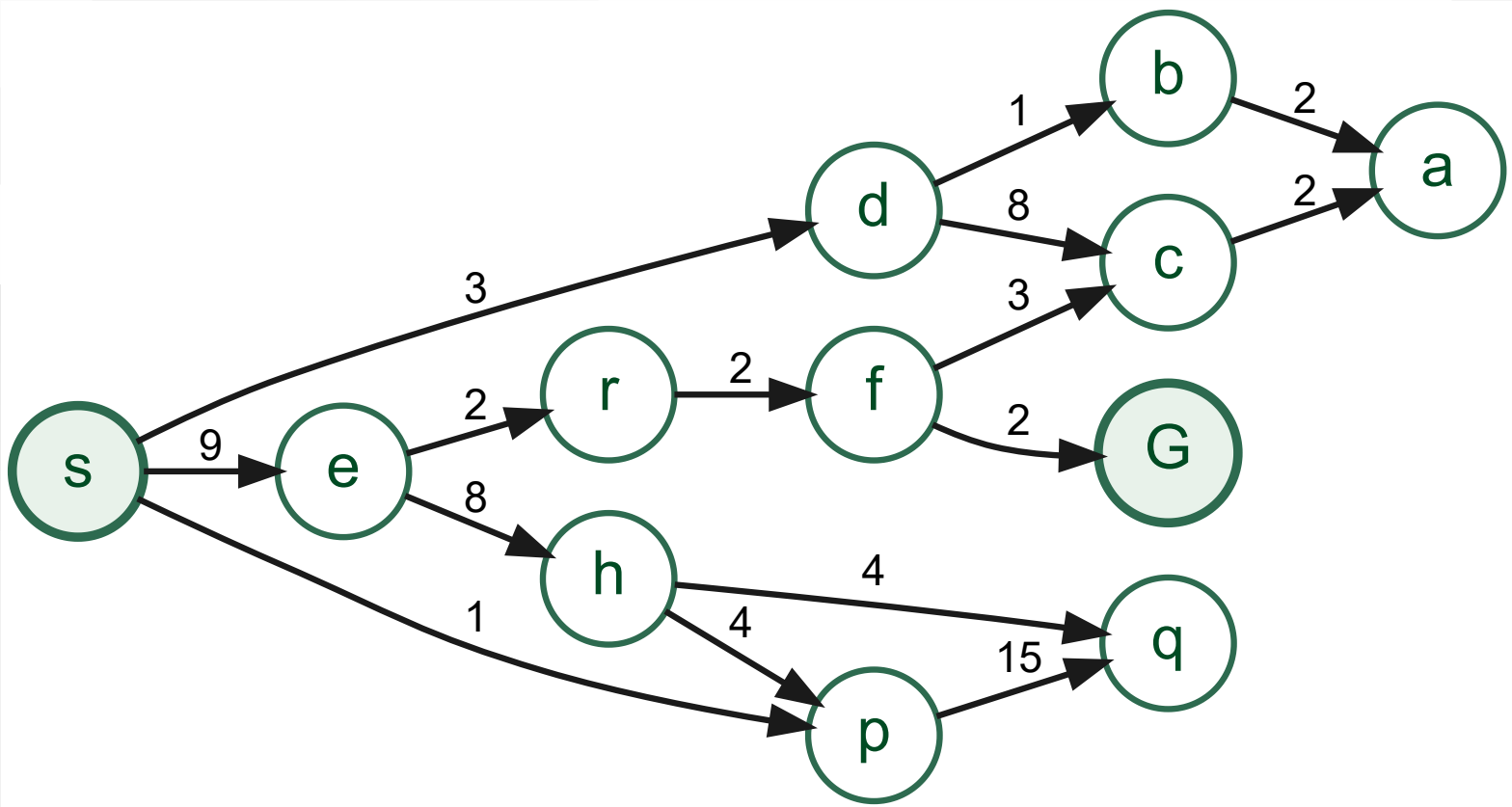
It does not find the least-cost path. We will now cover

a similar algorithm which does find the least-cost path.

Uniform Cost Search (UCS)



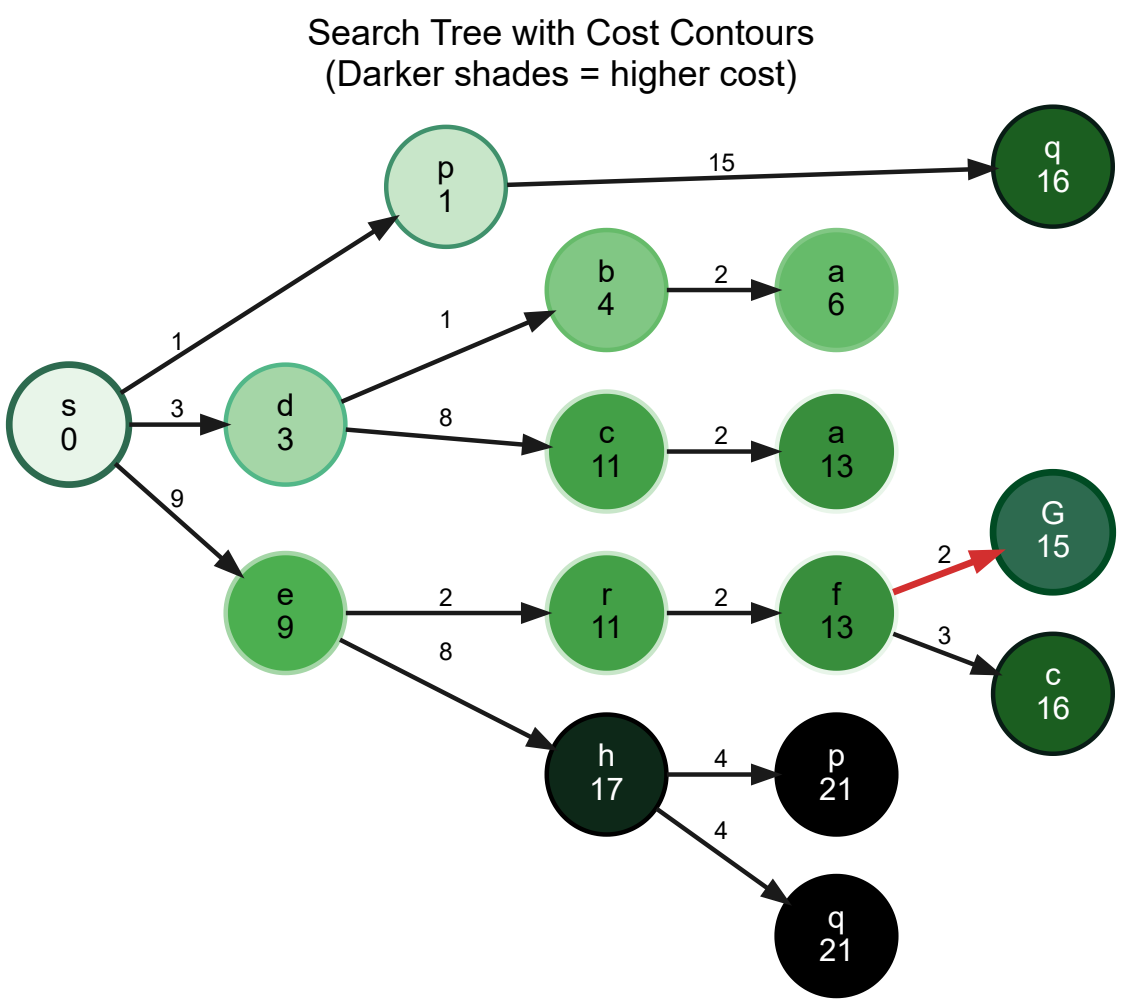
UCS Profile



Strategy: expand a cheapest node first

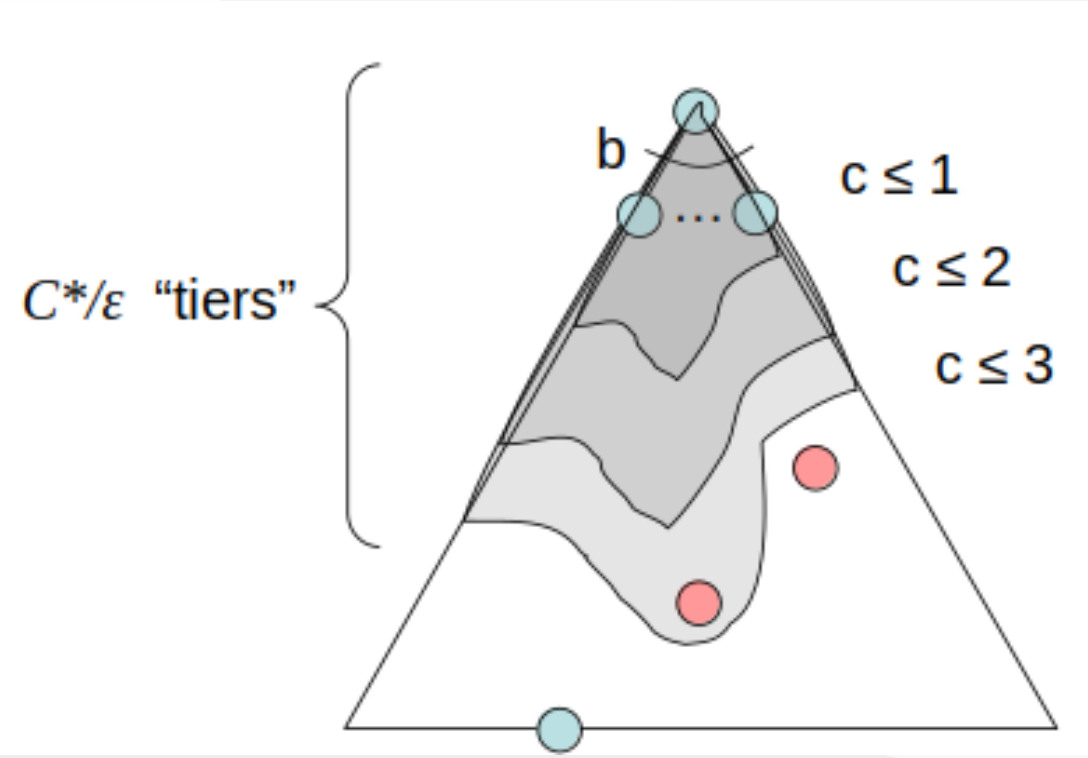
Fringe is a priority queue (priority: cumulative cost)

Example Cost profile



Uniform Cost Search (UCS) Properties

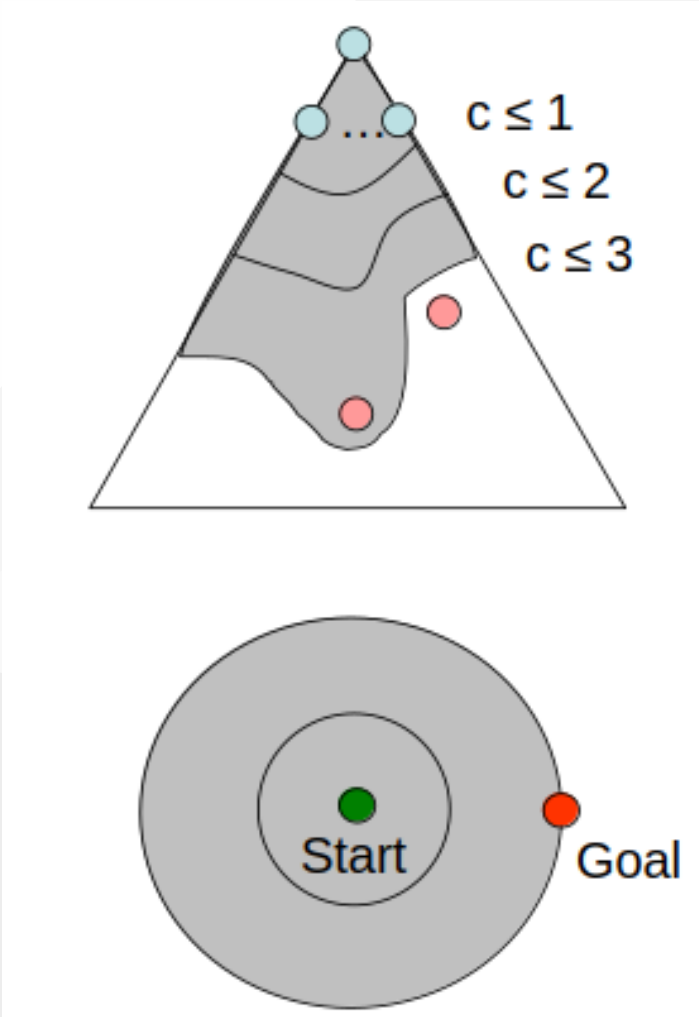
- What nodes does UCS expand?
  - All nodes with cost less than cheapest solution
  - Let solution cost = C^*
  - Let arcs cost $\geq \varepsilon$
  - "effective depth" $\approx C^*/\varepsilon$
  - Takes time $O(b^{C^*/\varepsilon})$
- How much space does the fringe take?
  - $O(b^{C^*/\varepsilon})$
- Is it complete?
  *Yes if: best solution has a finite cost and minimum arc cost is positive
- Is it optimal?
  - Yes! Proof via A^*



Uniform Cost Issues

- 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 the bad soon!



Guess Which One --- BFS or UCS?

Run the simulation:

Linux/Mac:

```
cd code
./run_guess_simulation.sh
```

Windows:

```
cd code
run_guess_simulation.bat
```

Or manually:

```
cd code/guess_bfs_ucs
python3 guess_bfs_water_maze.py
```

Requirements: Python 3.13+ with numpy and matplotlib

The maze has:

- Shallow water (cost 1) - light blue background
- Deep water (cost 2) - dark blue background

Guess Which of the Three --- DFS, BFS, UCS

Run the simulation:

Linux/Mac:

```
cd code
./run_guess_ucs_simulation.sh
```

Windows:

```
cd code
run_guess_ucs_simulation.bat
```

Or manually:

```
cd code/guess_bfs_ucs
python3 guess_ucs_water_maze.py
```

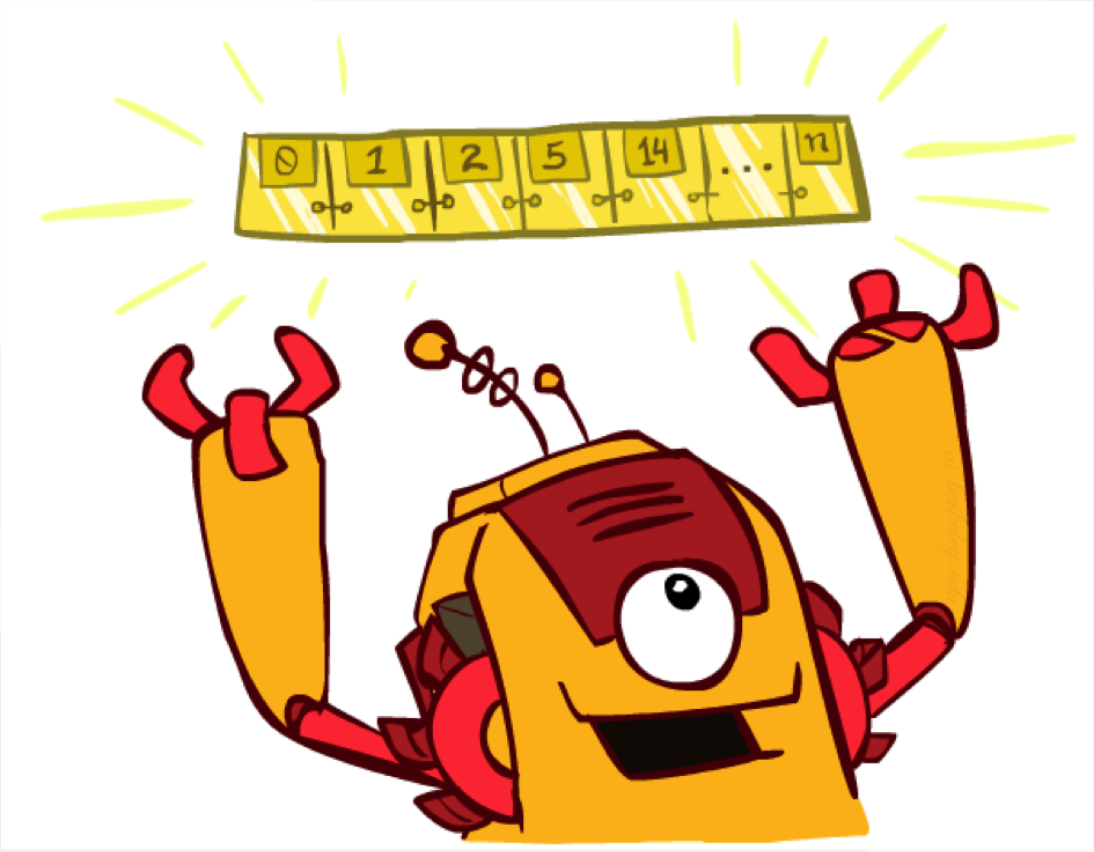
Requirements: Python 3.13+ with numpy and matplotlib

The maze has:

- Shallow water (cost 1) - light blue background
- Deep water (cost 2) - dark blue background

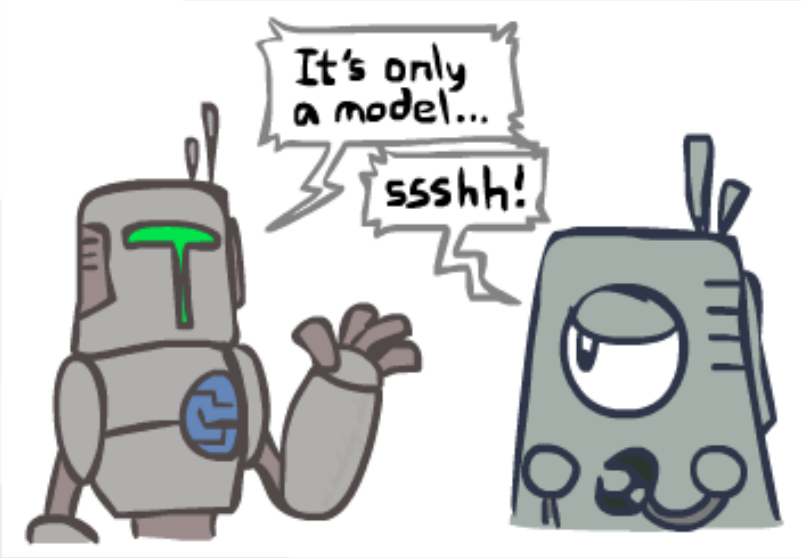
The One Queue

- All search algorithms same except different fringe strategies
- All fringes are priority queues: collections of nodes with attached priorities
- For DFS and BFS, you can avoid the $log(n)$ overhead from an actual priority queue by using stack or queue
- Can create one implementation that takes a variable queuing object (Stack is a LIFO Queue...)
- but don't do that for this class



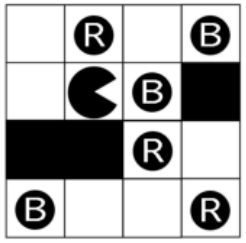
Search and Models

- 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...



State Space Example

There are two kinds of food pellets, each with a different color (red and blue). Pacman is only interested in tasting the two different kinds of food: the game ends when he has eaten 1 red pellet and 1 blue pellet (though Pacman may eat more than one of each pellet). Pacman has four actions: moving up, down, left, or right, and does not have a "stay" action. There are K red pellets and K blue pellets, and the dimensions of the board are N by M.



$K = 3,\ N = 4,\ M = 4$

(a) Give an efficient state space formulation of this problem. Specify the domain of each variable in your state space.

(b) Give a tight upper bound on the size of the state space.

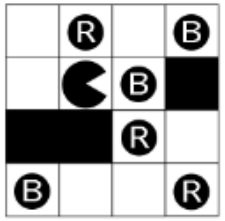
(c) Give a tight upper bound on the branching factor of the search problem.

(d) Assuming Pacman starts the game in position (x,y), what is the initial state?

(e) Define a goal test for the problem.

State Space Example (Solution)

There are two kinds of food pellets, each with a different color (red and blue). Pacman is only interested in tasting the two different kinds of food: the game ends when he has eaten 1 red pellet and 1 blue pellet (though Pacman may eat more than one of each pellet). Pacman has four actions: moving up, down, left, or right, and does not have a "stay" action. There are K red pellets and K blue pellets, and the dimensions of the board are N by M.



$K = 3, N = 4, M = 4$

(a) Give an efficient state space formulation of this problem. Specify the domain of each variable in your state space.

We need two variables to describe the location of pacman, one boolen variable showing whether pacmac already ate a red pellet, and another boolean variable for the blue pellets. Formally:

$(x \in [1:N], y \in [1:M], eaten_R \in \{T, F\}, eaten_B \in \{T, F\})$

(b) Give a tight upper bound on the size of the state space.

There are at most $N \times M$ possible locations for pacman and 4 possible assignments to the boolean variables so the size of the state space is upper bounded by $4 \times N \times M$

(c) Give a tight upper bound on the branching factor of the search problem.

Each state has at most four distinct successors corresponding to the four possible actions. The branching factor is at most 4.

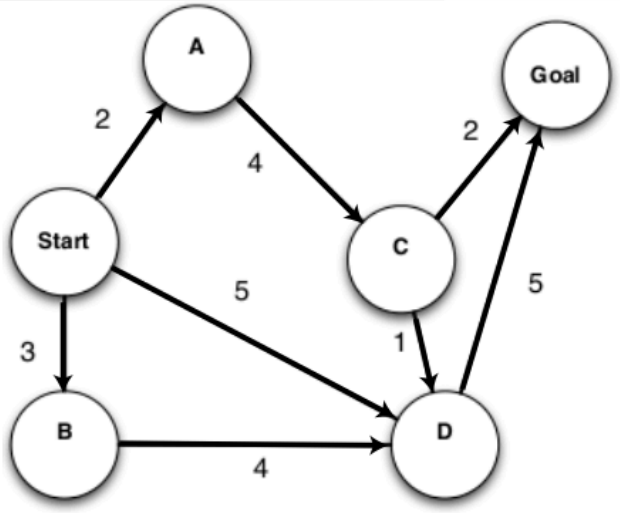
(d) Assuming Pacman starts the game in position (x,y), what is the initial state?

(x, y, F, F). The two boolean state variables are both $false$.

(e) Define a goal test for the problem.

$(eaten_R == T)\&\&(eaten_B == T)$

Search Problem Example



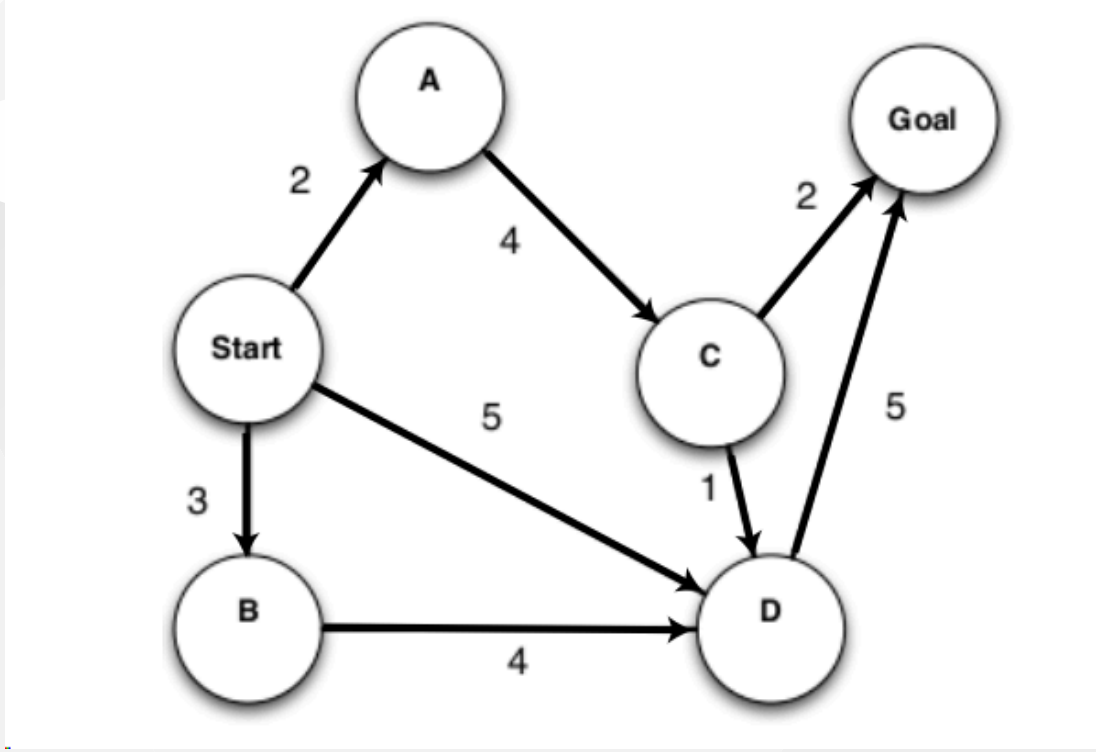
For each of the following graph search strategies, work out the order in which states are expanded, as well as the path returned by graph search. In all cases, assume ties resolve in such a way that states with earlier alphabetical order are expanded first. Remember that in graph search, a state is expanded only once.

a) Depth-first search.

b) Breadth-first search.

c) Uniform cost search.

Search Problem Example (solution)



a) Depth-first search.
States Expanded: Start, A, C, D, B, Goal
Path Returned: Start-A-C-D-Goal

b) Breadth-first search.
States Expanded: Start, A, B, D, C, Goal
Path Returned: Start-D-Goal

c) Uniform cost search.
States Expanded: Start, A, B, D, C, Goal
Path Returned: Start-A-C-Goal