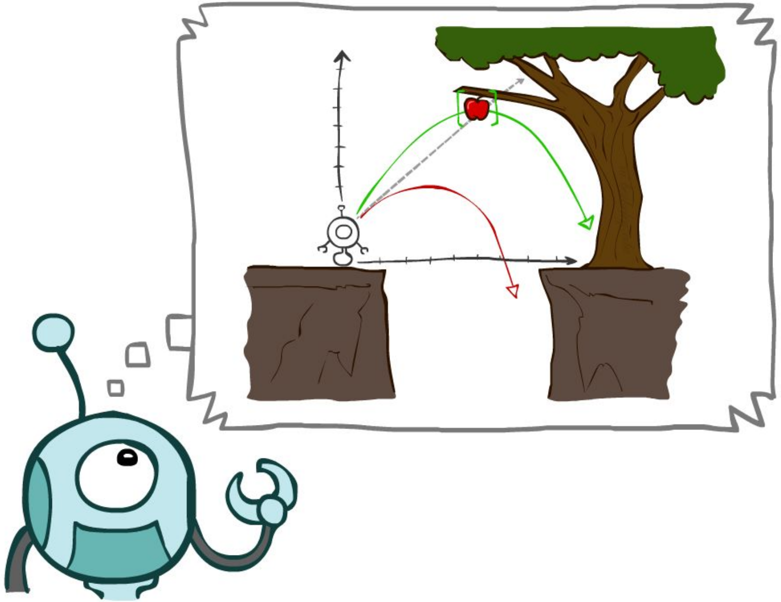
Search Problems
Theory and Applications

Prof. Dr. Eduardo Noronha
Instituto Federal de Goiás (IFG)

[Some slides were adapted from Dan Klein and Pieter Abbeel at UC Berkeley]

Today

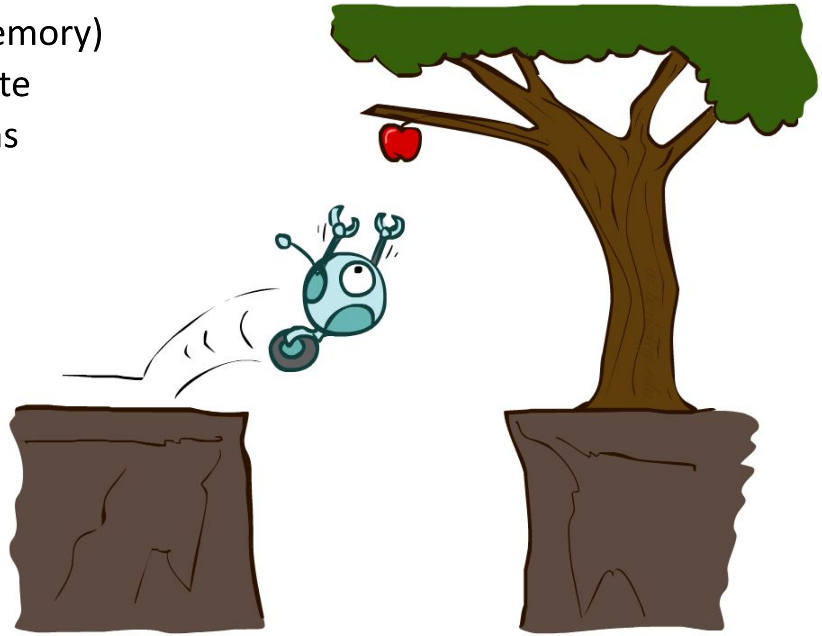
- Agents that Plan Ahead
- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search
- Implementation
- Applications



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?



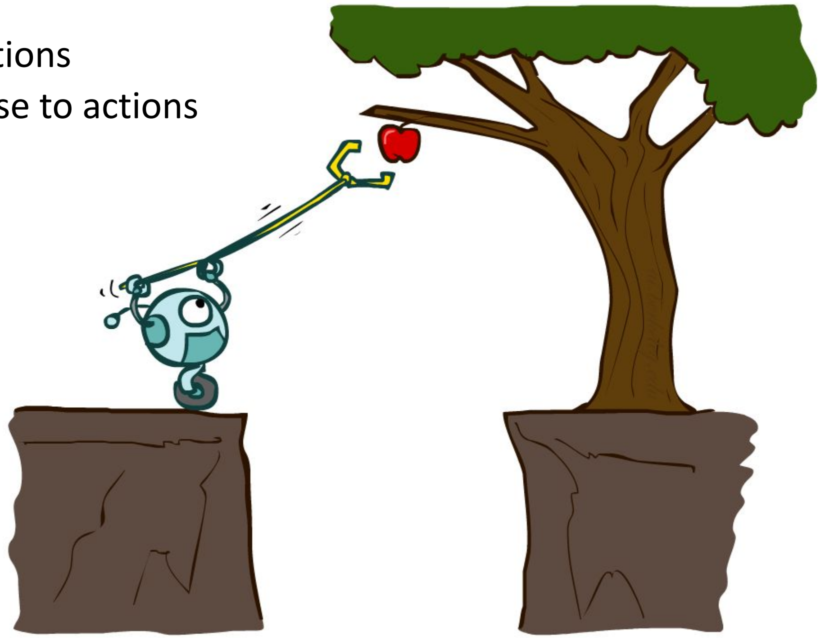
[Demo: reflex optimal (L2D1)]
[Demo: reflex optimal (L2D2)]

Reflex Agent - Video

Reflex Odd - Video

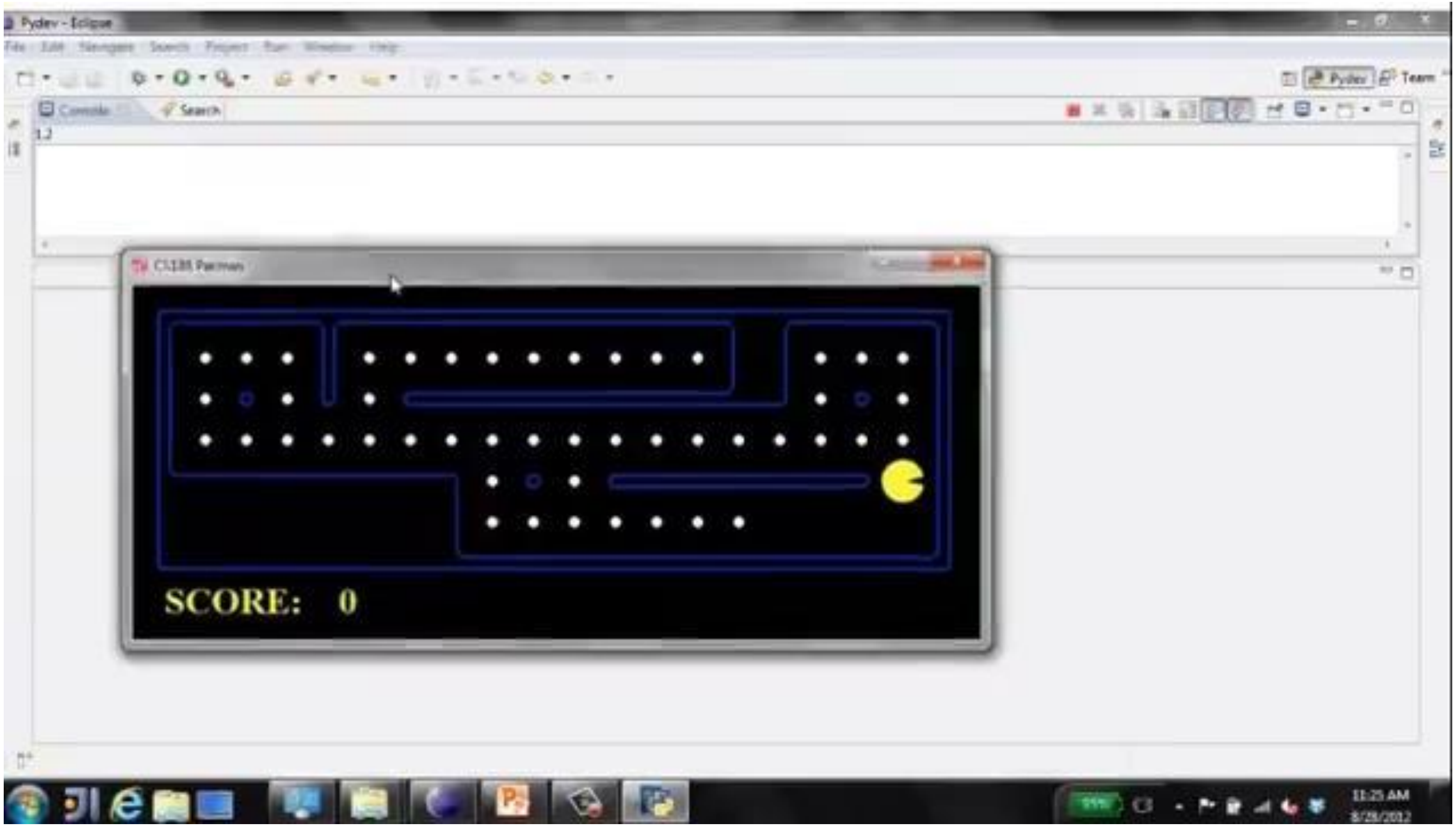
Planning Agents

- 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

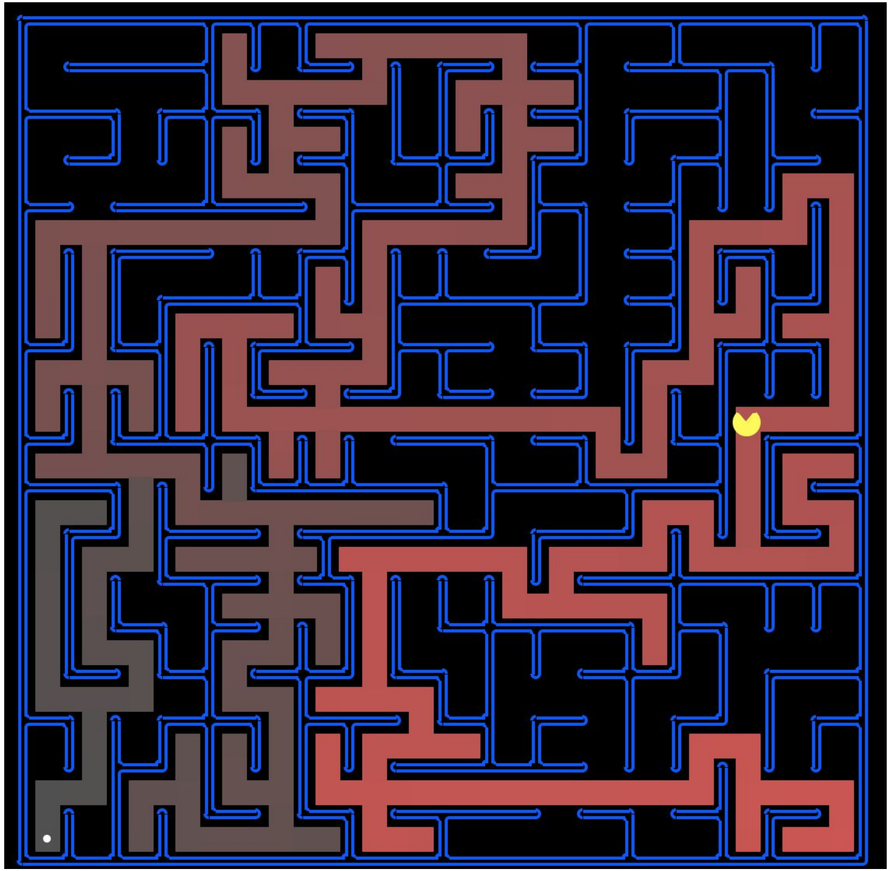


Replanning Video

Mastermind Video



Our Goal: Help Pac-man finds its way!



<u>Search</u>: breadth- first, depth-first, uniform cost search.

<u>Heuristic Search</u>: Best-first, A*

State-Space Search Problems

General problem: Find a path from a **start state** to a **goal state** given:

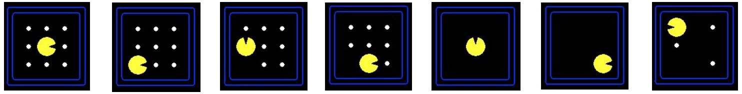
- A goal test: Tests if a given state is a goal state
- A successor function (transition model): Given a state and action, generate successor state

Variants:

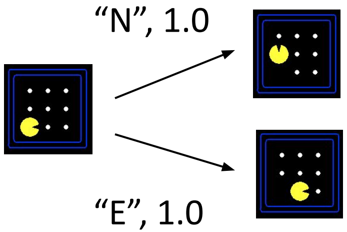
- Find any path vs. a least-cost path (if each step has a different cost i.e. a "step-cost")
- Goal is completely specified, task is to find a path or least-cost path (i.e., Route planning)
- Path doesn't matter, only finding the goal state – 8 puzzle, N queens, Rubik's cube

Search Problems

- A search problem consists of:
  - A state space



  - A successor function (with actions, costs)

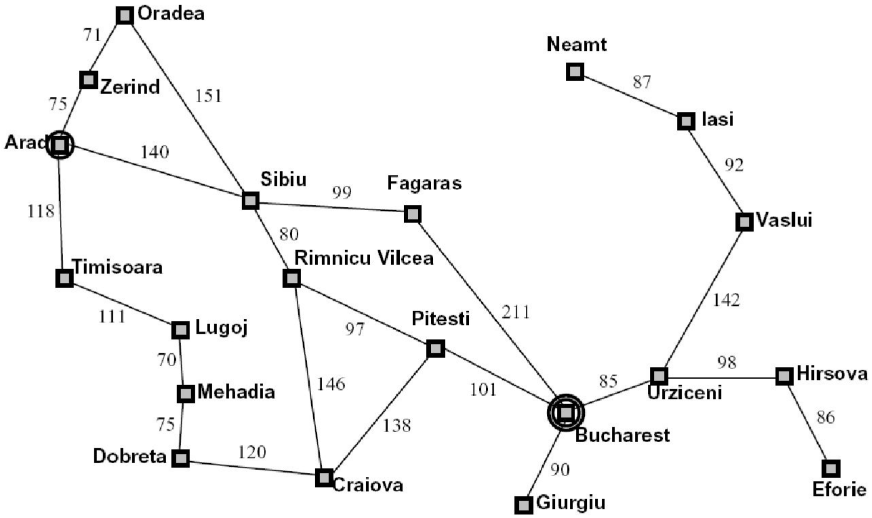


  - A start state and a goal test

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

Search Problems Are Models

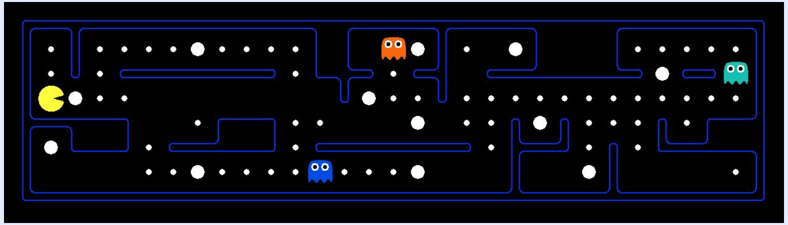
Example: Traveling in Romania



- 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)

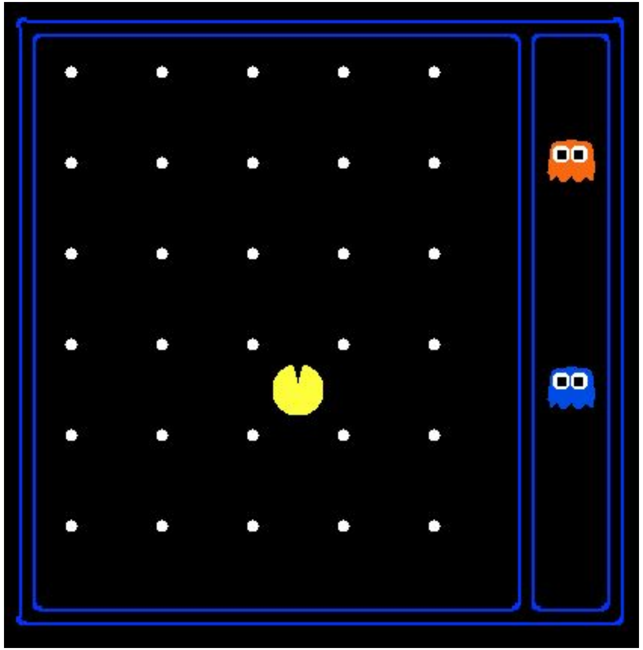
- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END

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

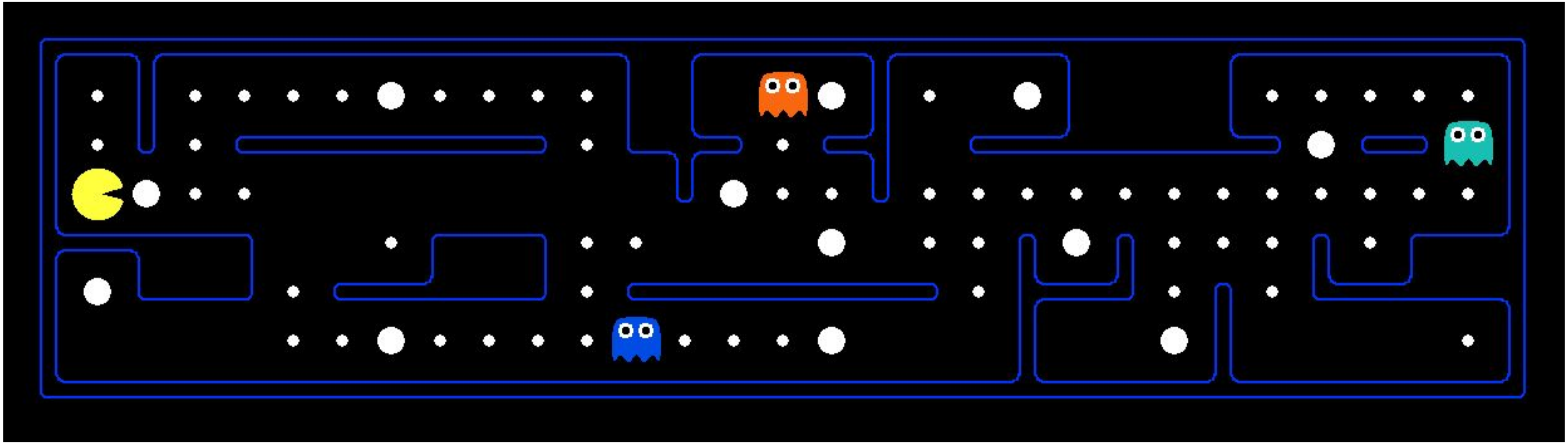
State Space Sizes?

- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

- How many
  - World states?
    $120x(2^{30})x(12^{2})x4$
  - States for pathing?
    120
  - States for eat-all-dots?
    $120x(2^{30})$

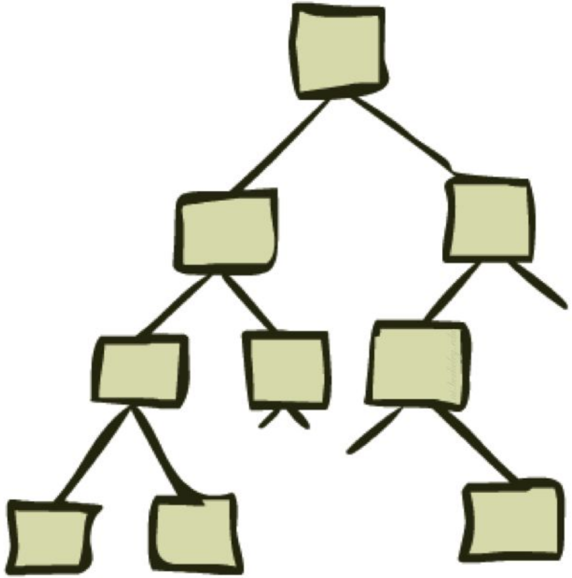


Quiz: Safe Passage



- 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

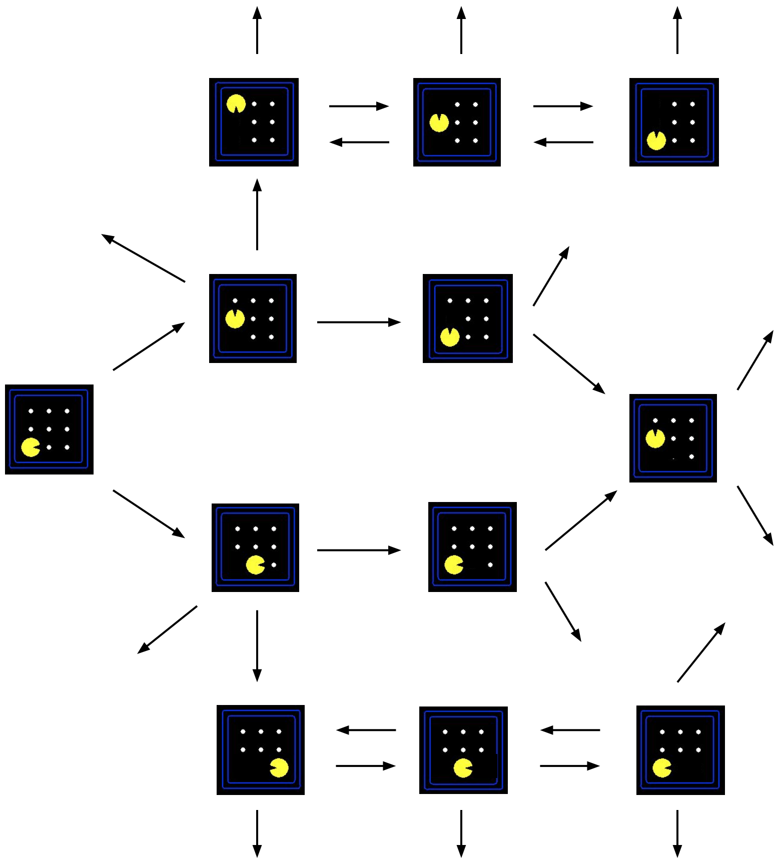


State Space Graphs

- 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)

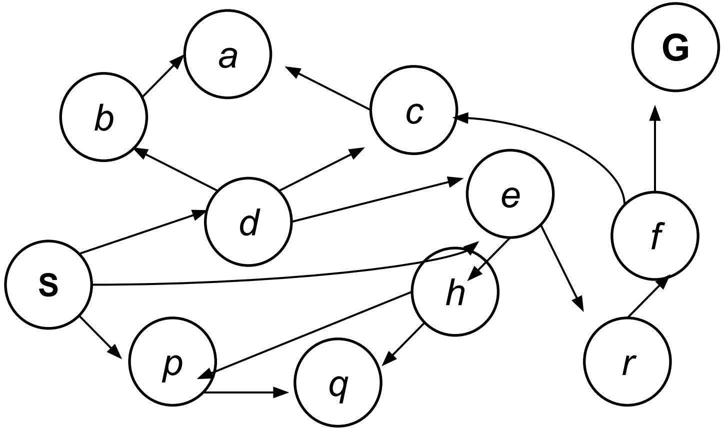
- In a state space graph, each state occurs only once!

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



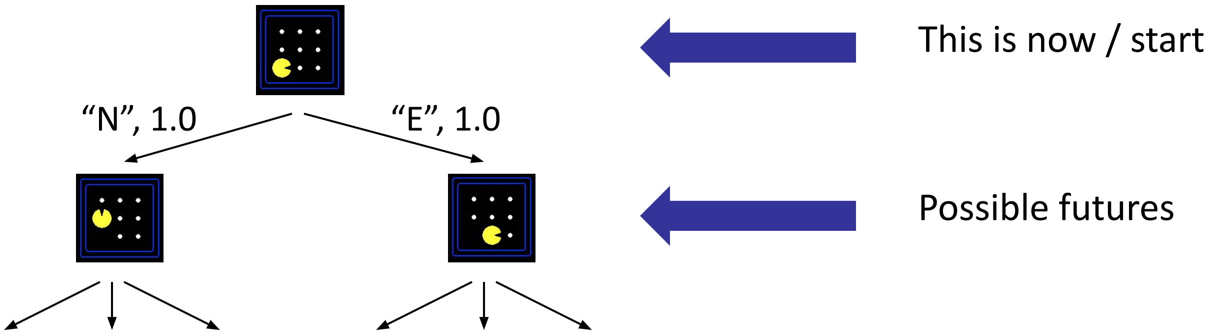
State Space Graphs

- 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



Tiny search graph for a tiny search problem

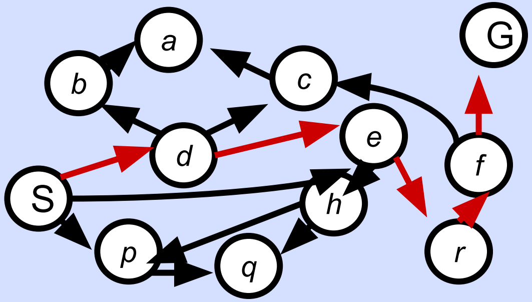
Search Trees



- 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

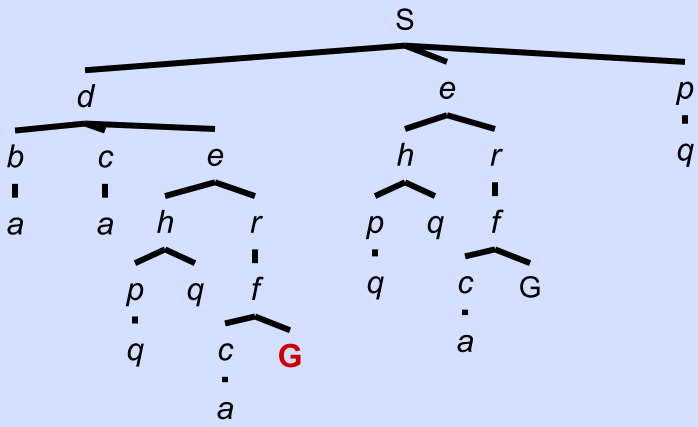
State Space Graph



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

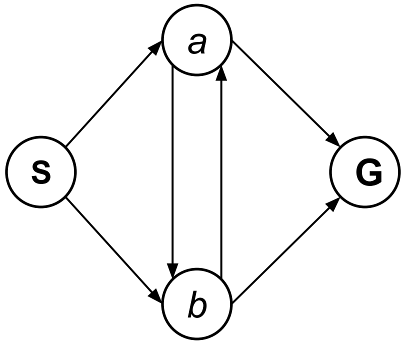
We construct both on demand – and we construct as little as possible.

Search Tree



Quiz: State Space Graphs vs. Search Trees

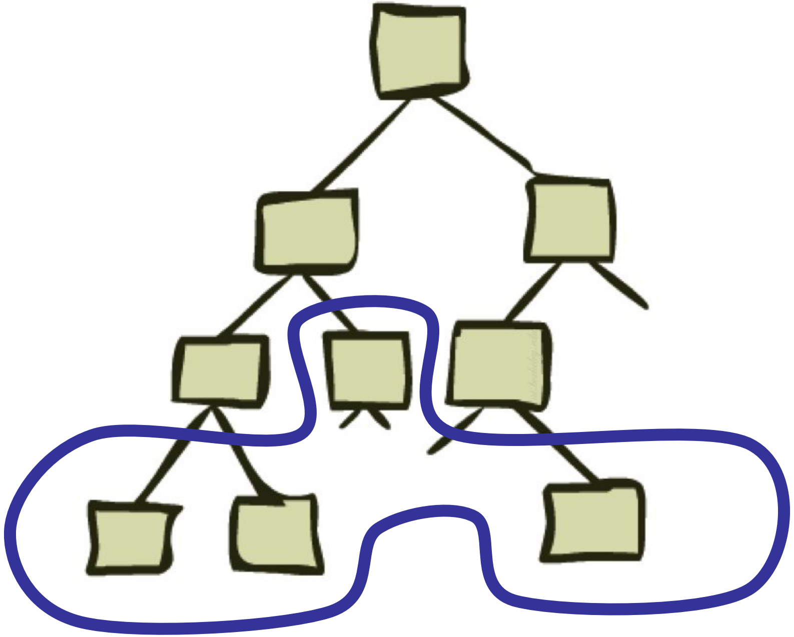
Consider this 4-state graph:



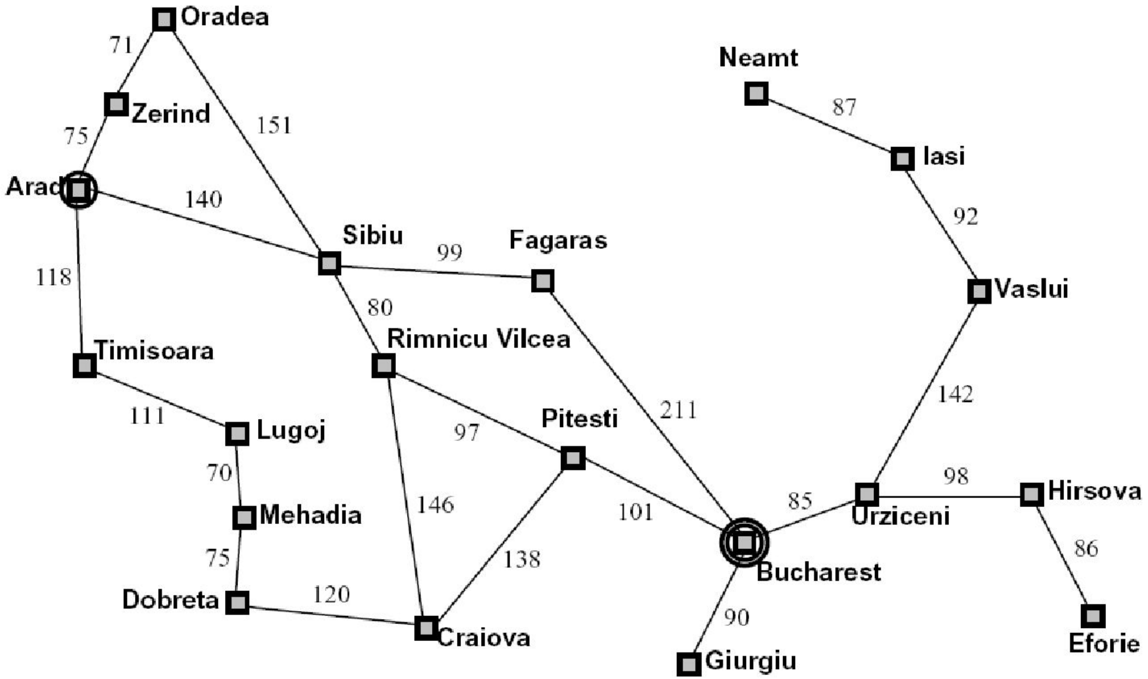
How big is its search tree (from S)?

Important: Lots of repeated structure in the search tree!

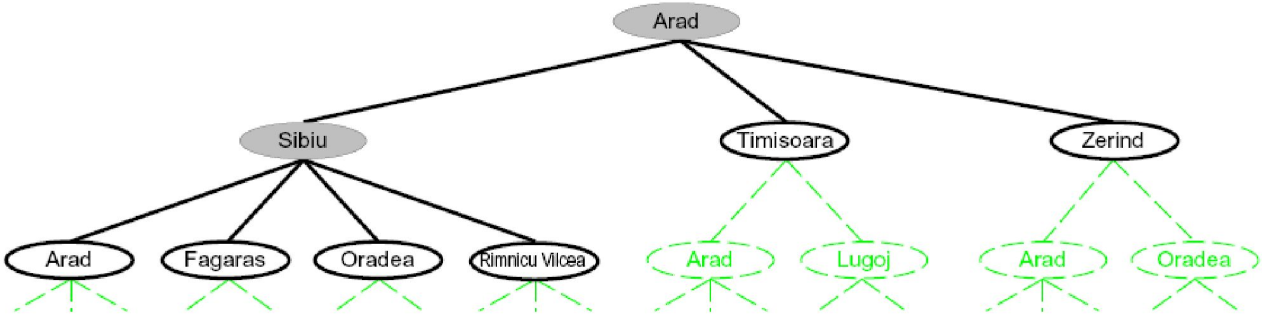
Tree Search



Search Example: Romania



Searching with a Search Tree



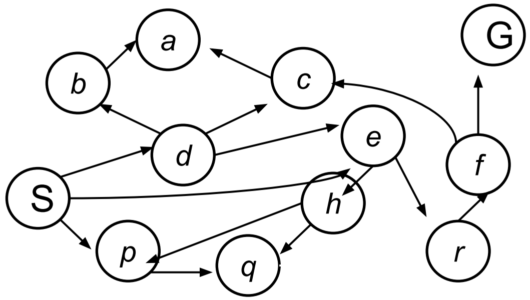
- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

General Tree Search

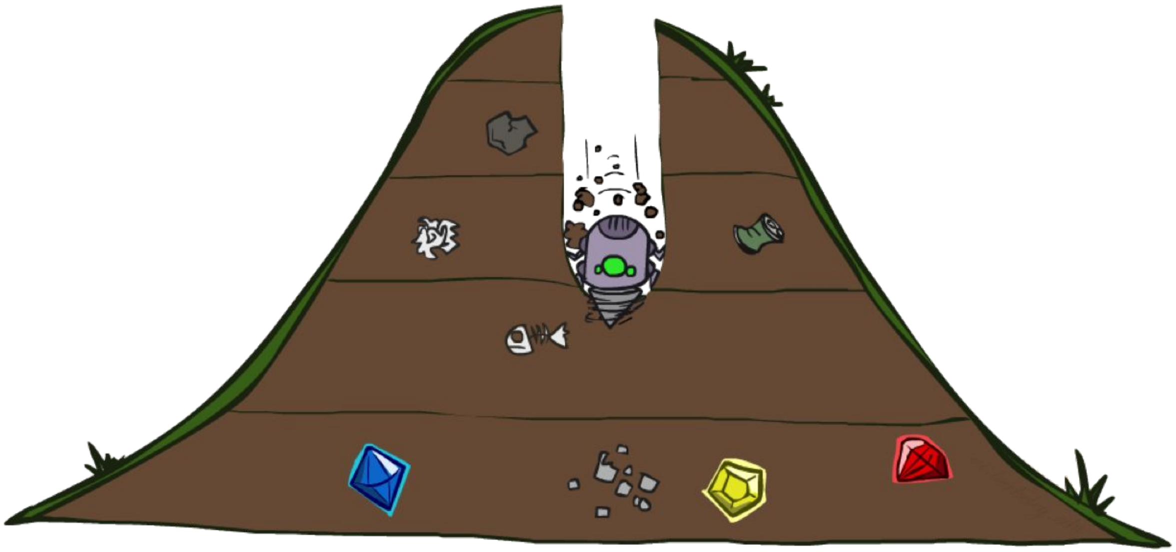
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains 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?

Example: Tree Search



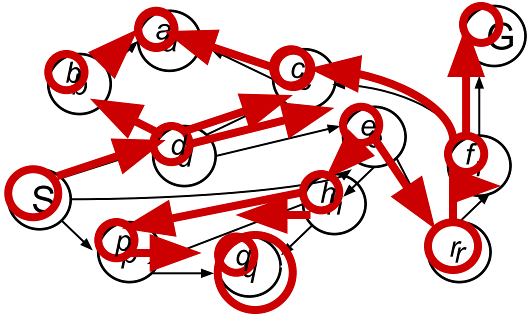
Depth-First Search

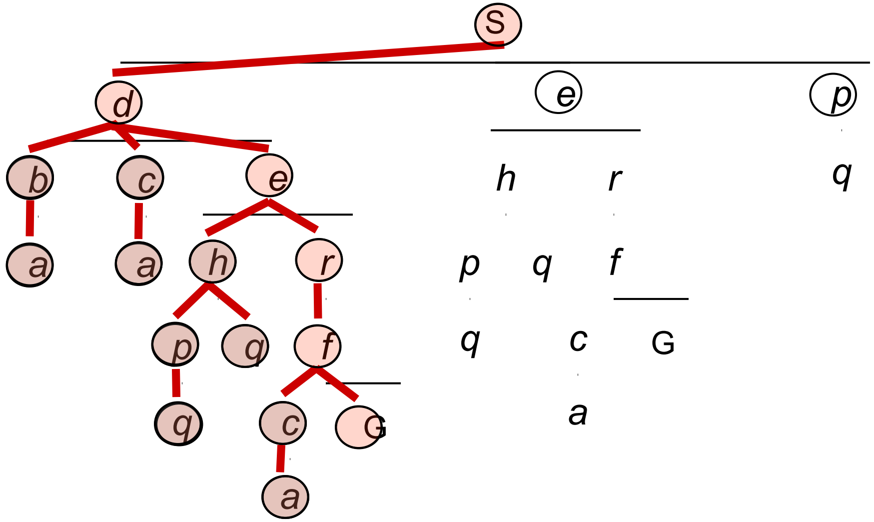


Depth-First Search

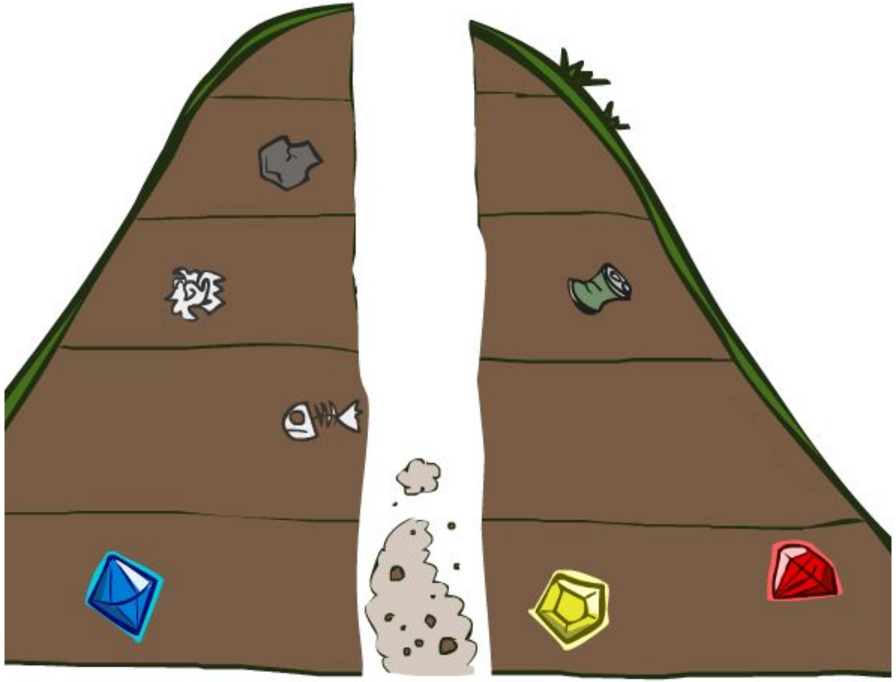
Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack





Search Algorithm Properties

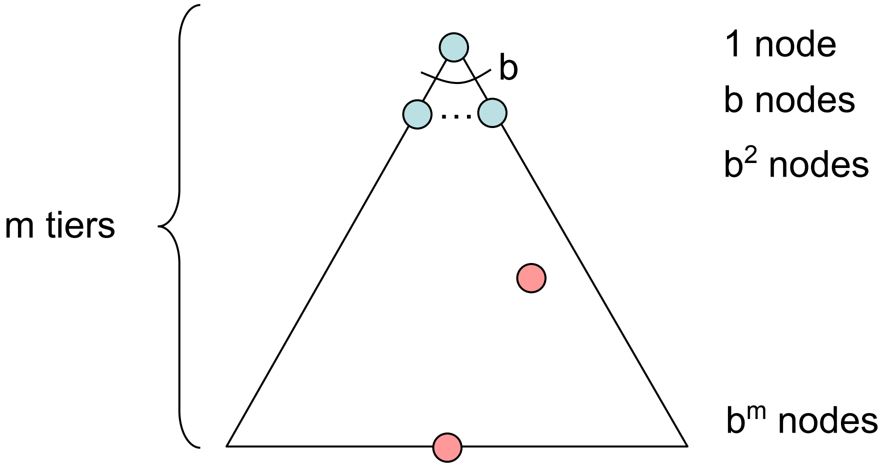


Search Algorithm Properties

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

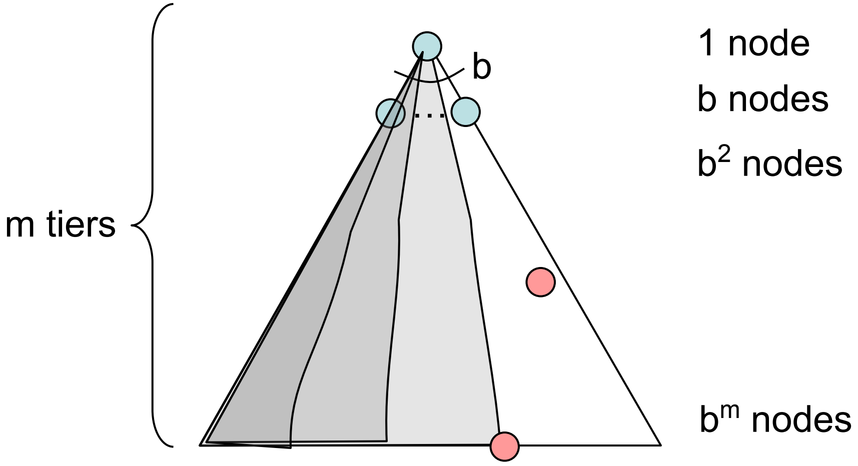
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths

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

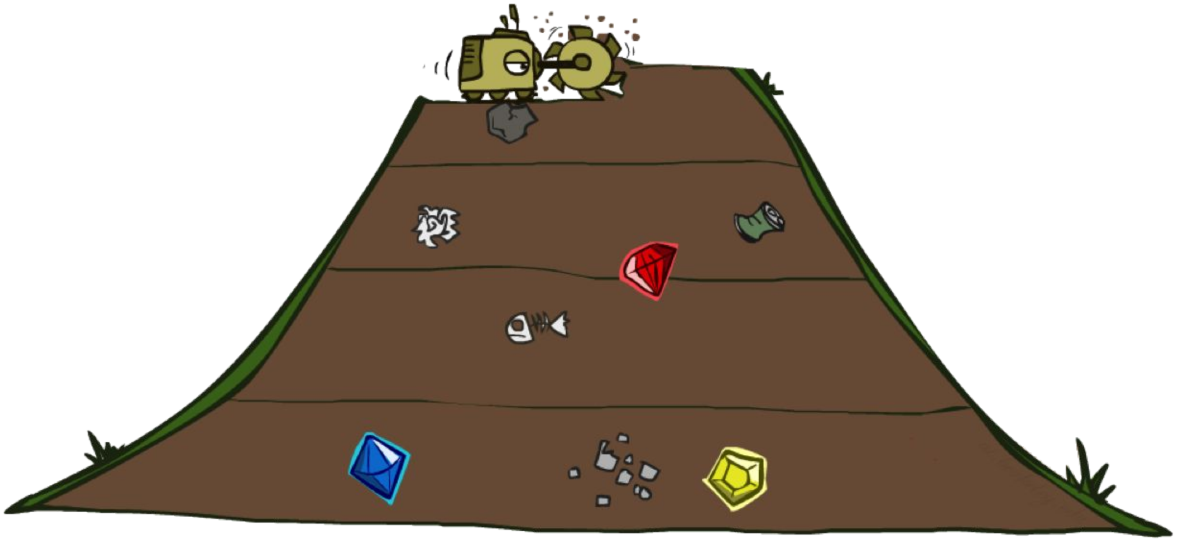


Depth-First Search (DFS) Properties

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



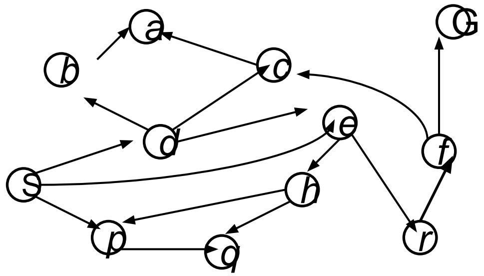
Breadth-First Search

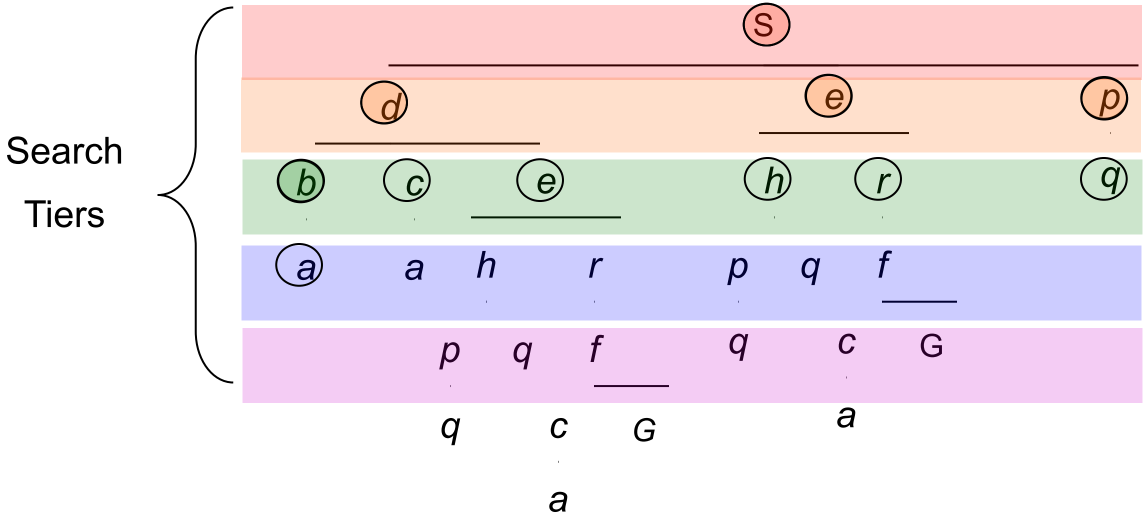


Breadth-First Search

Strategy: expand a shallowest node first

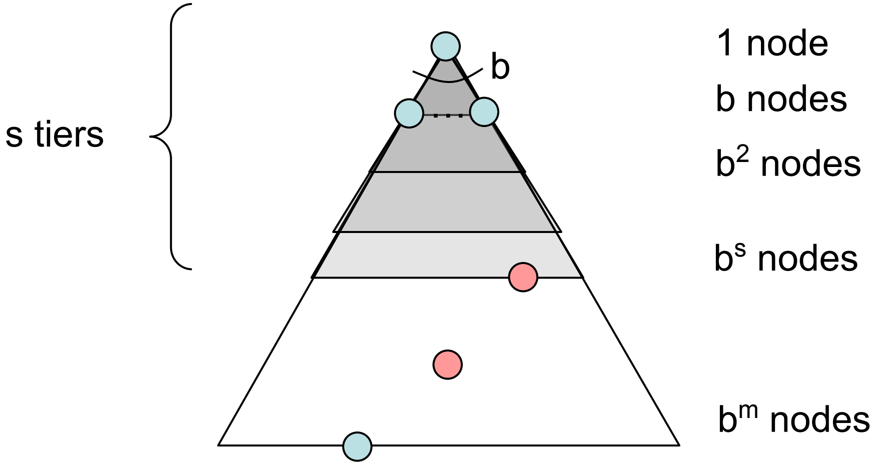
Implementation: Fringe is a FIFO queue



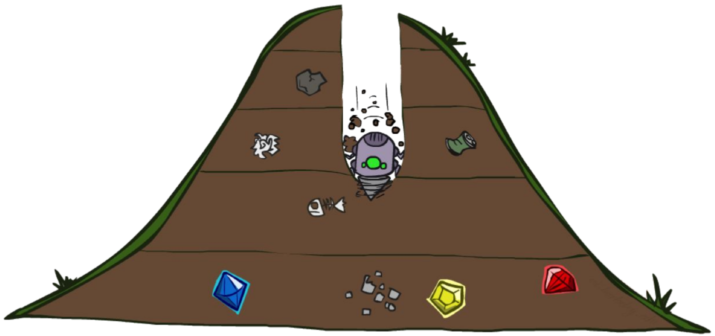


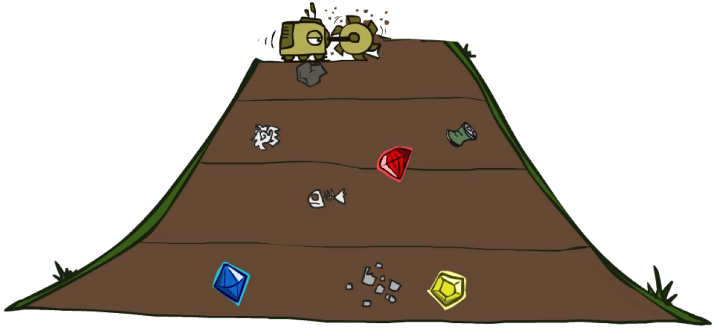
Breadth-First Search (BFS) Properties

- 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)



Quiz: DFS vs BFS





Quiz: DFS vs BFS

- When will BFS outperform DFS?

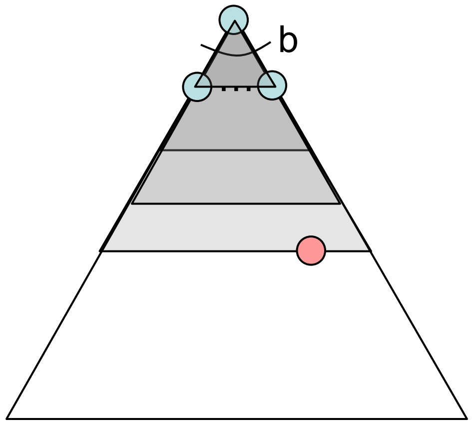
- When will DFS outperform BFS?

[Demo: dfs/bfs maze water (L2D6)]

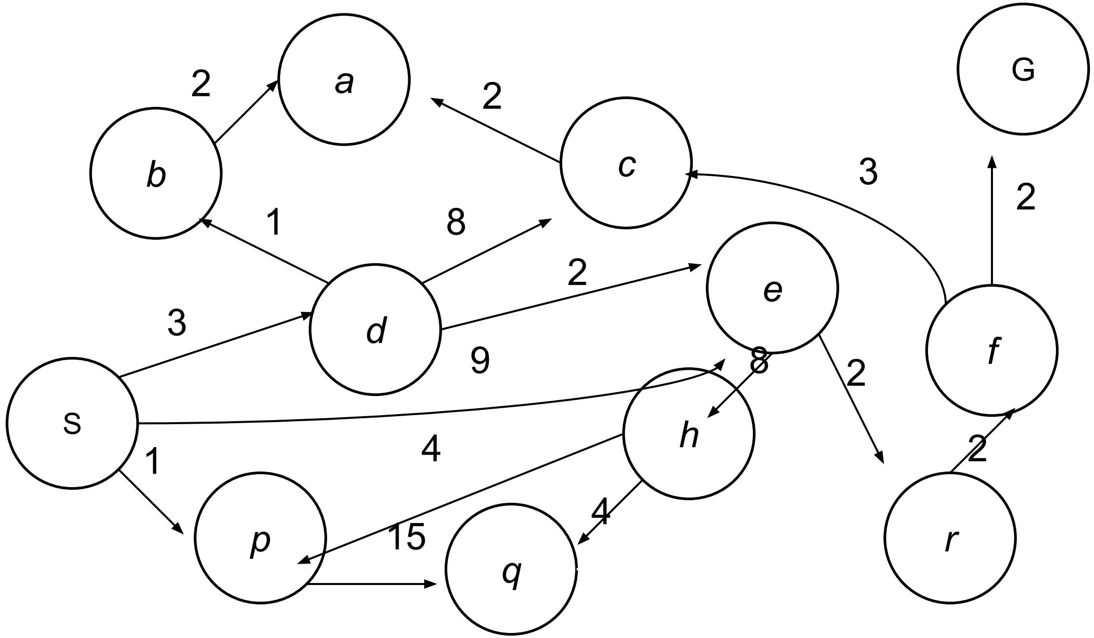
Iterative Deepening

- 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!

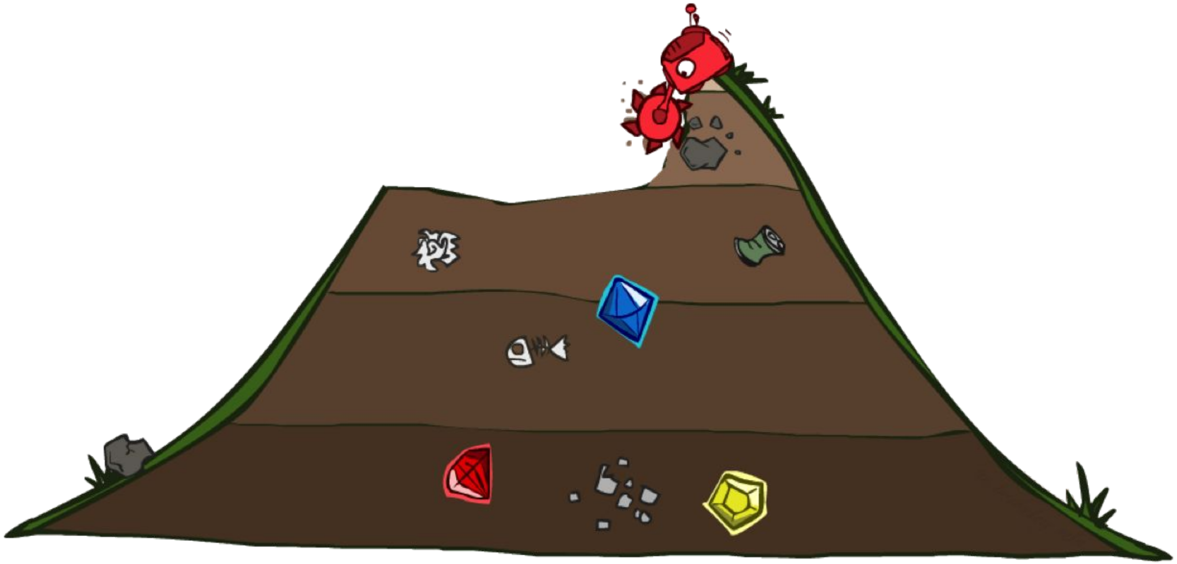


Cost-Sensitive Search



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.

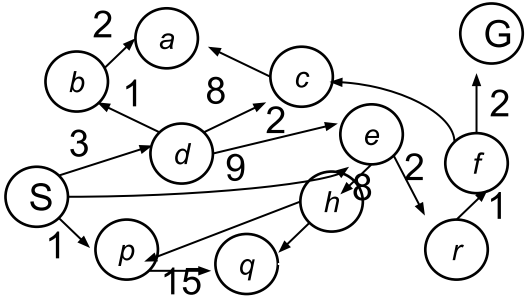
Uniform Cost Search



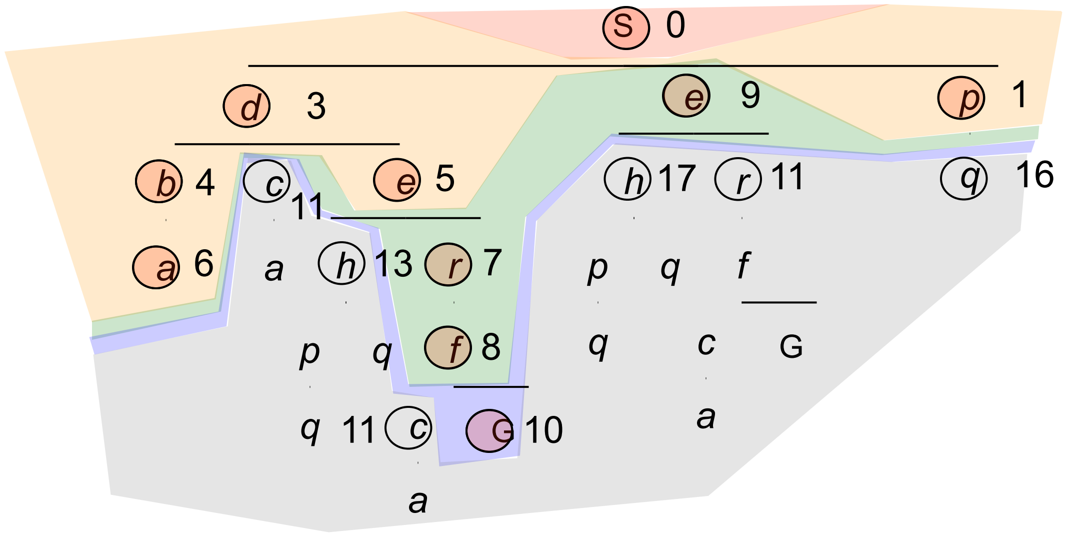
Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)

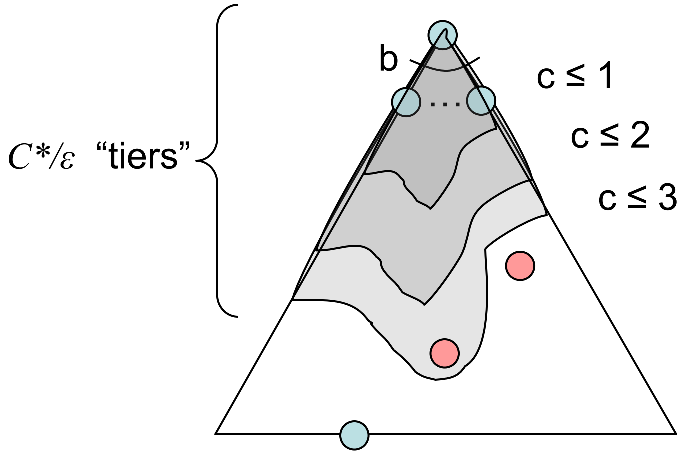


Cost contours



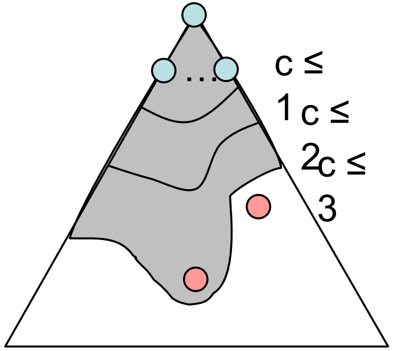
Uniform Cost Search (UCS) Properties

- 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^*/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*)



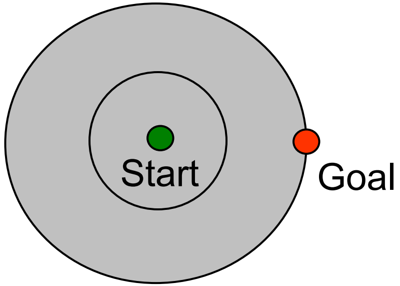
Uniform Cost Issues

- Remember: UCS explores increasing cost contours



- The good: UCS is complete and optimal!

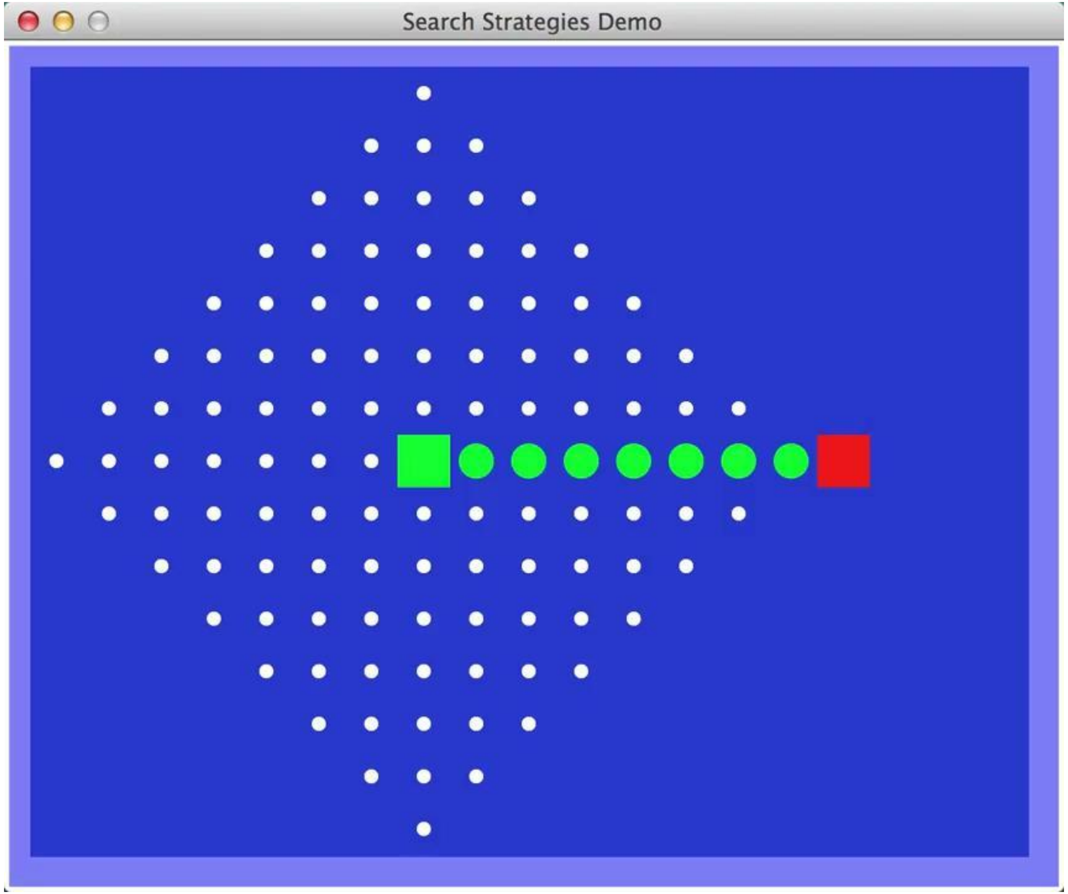
- The bad:
  - Explores options in every “direction”
  - No information about goal location



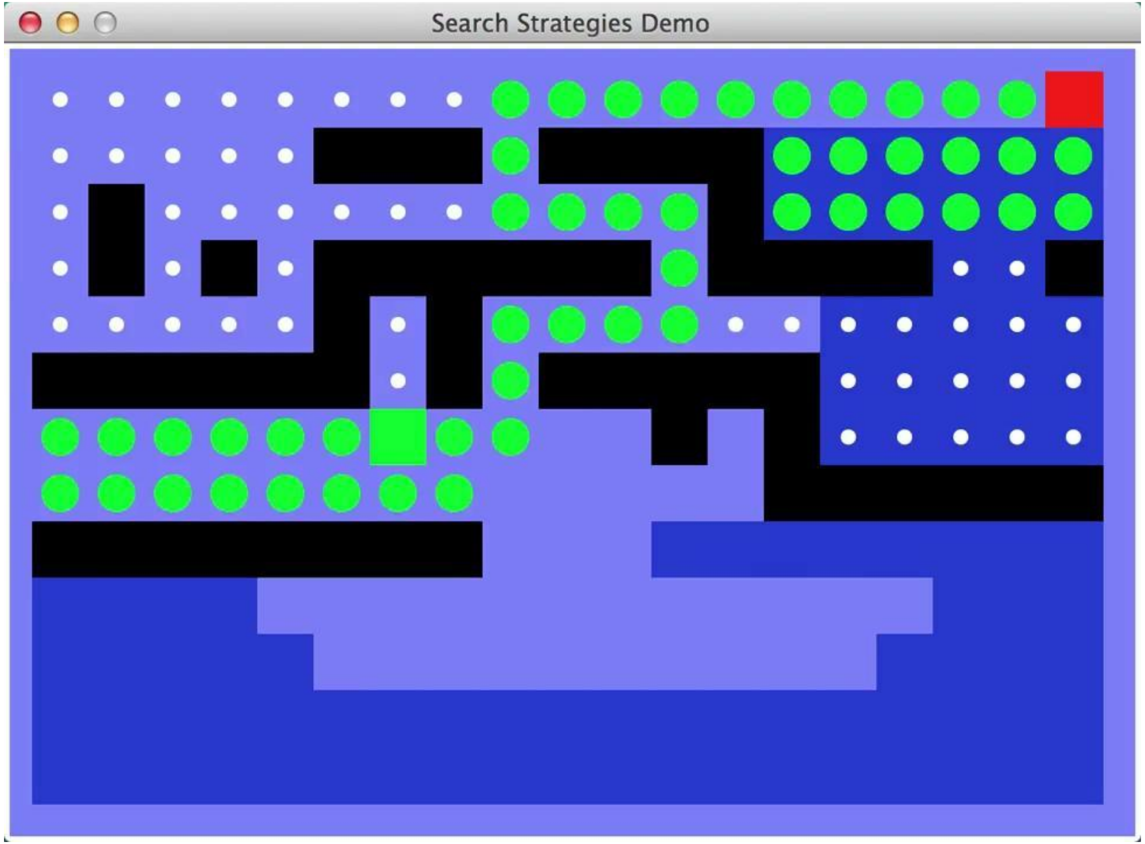
- We’ll fix that soon!

[Demo: empty grid UCS (L2D5)]
[Demo: maze with deep/shallow water DFS/BFS/UCS (L2D7)]

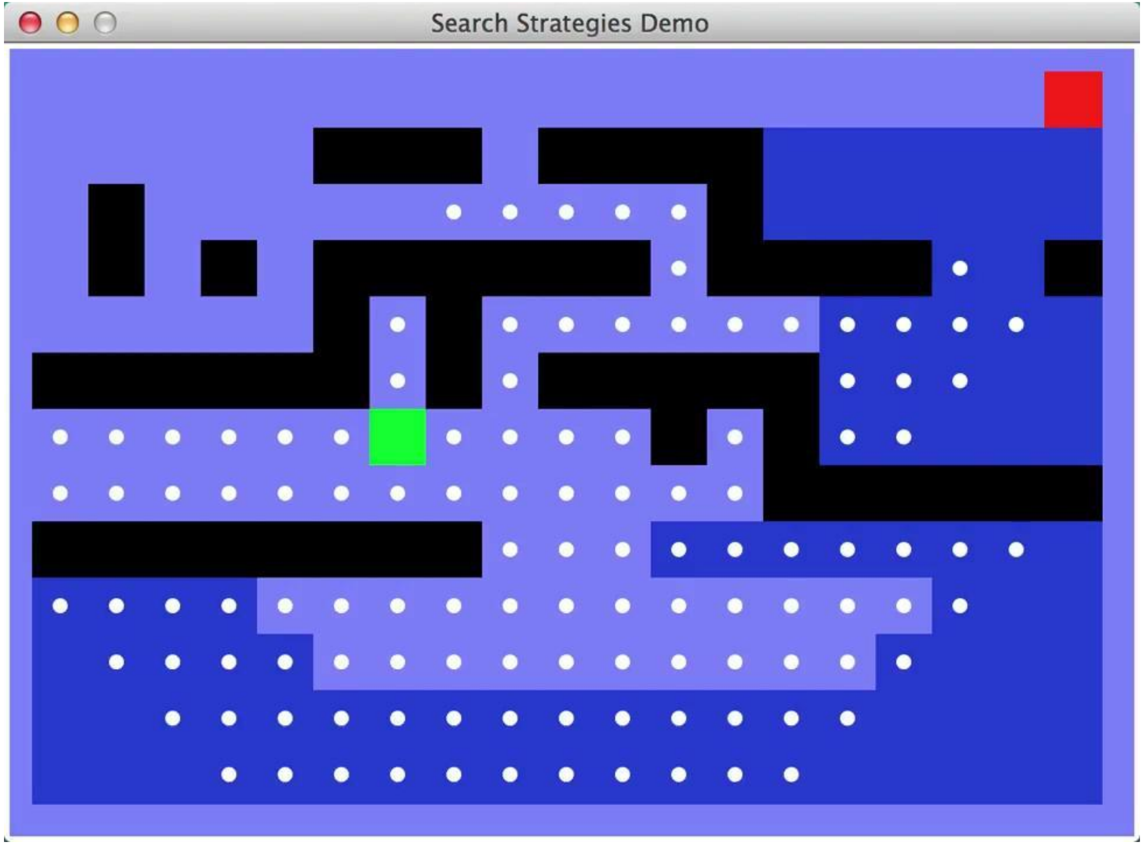
Video of Demo Empty UCS



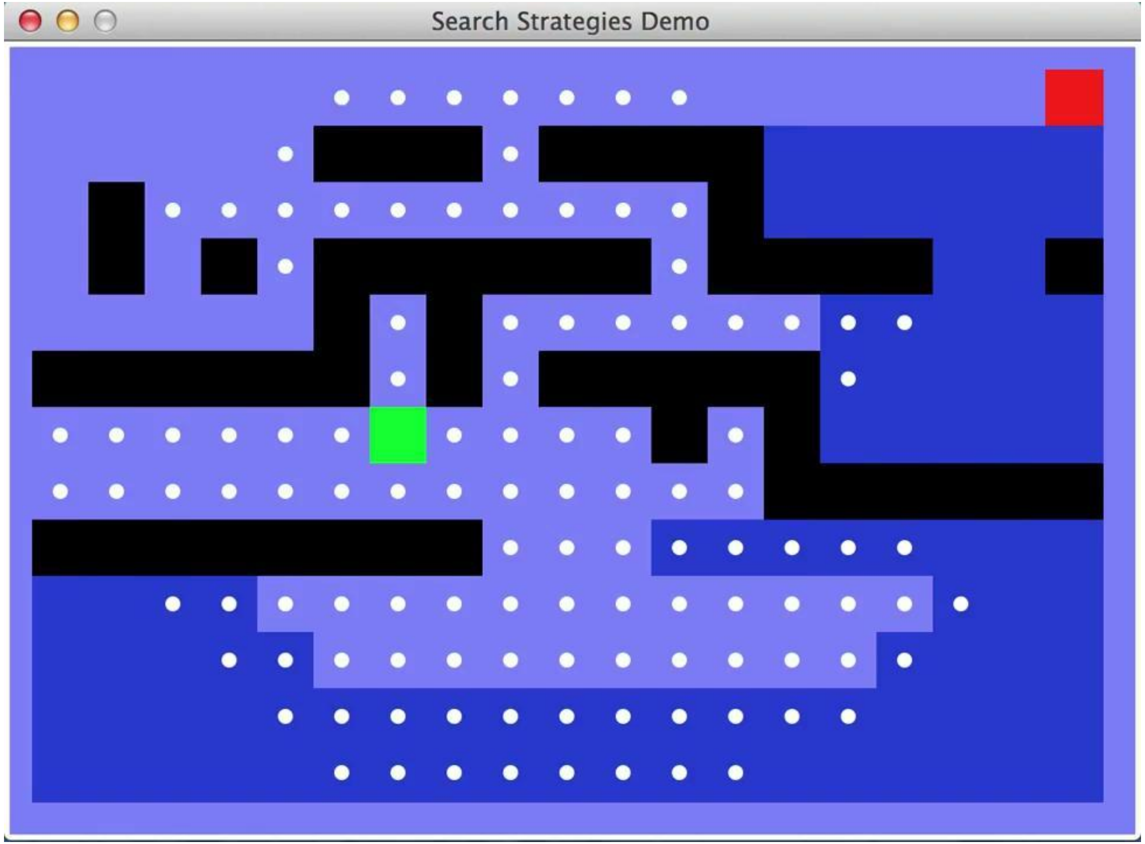
Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 1)



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

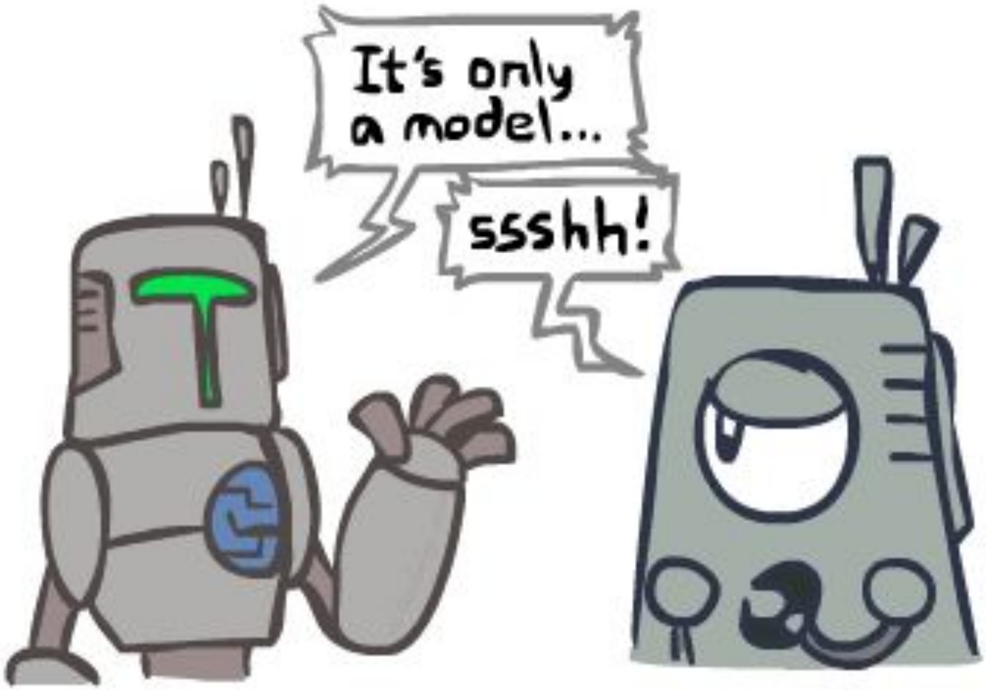


Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 3)

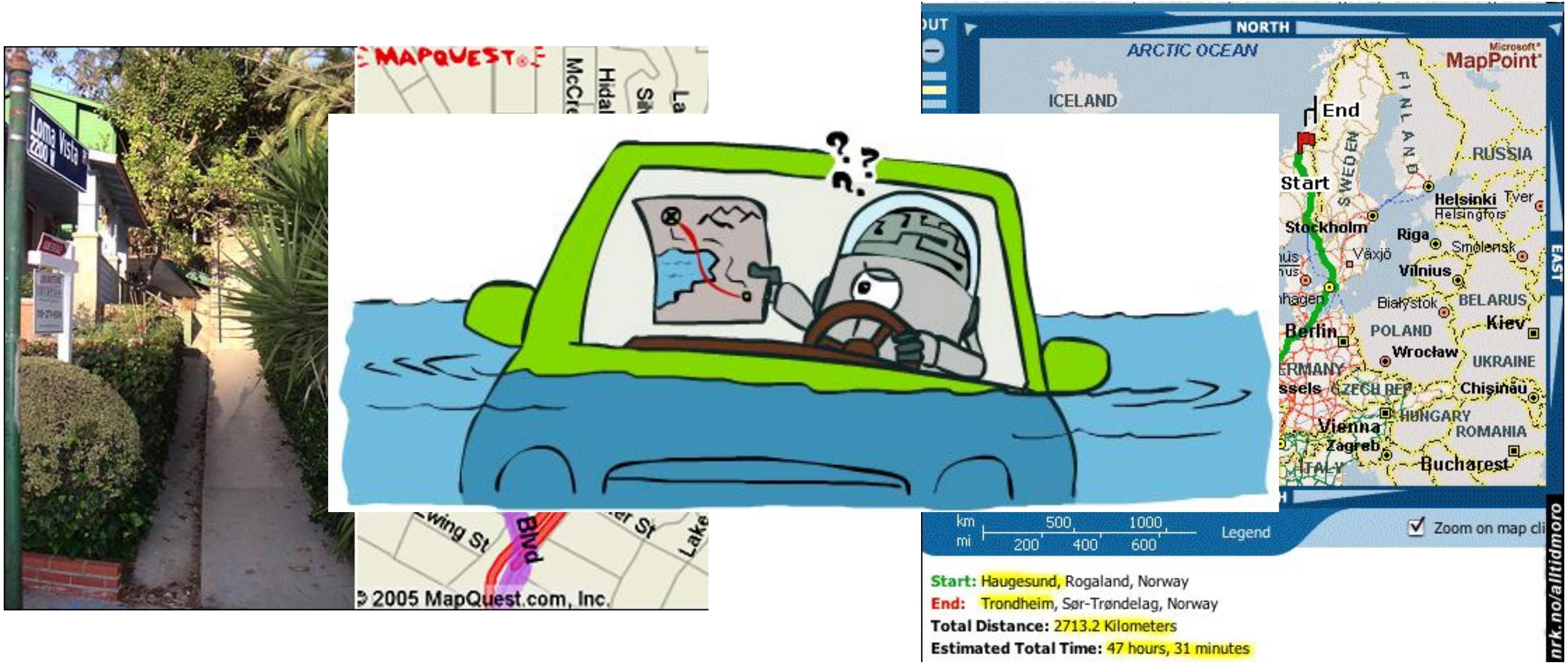


Search and Models

- 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?



Some Hints for P1

- Graph search is almost always better than tree search (when not?)
- Implement your closed list as a dict or set!
- Nodes are conceptually paths, but better to represent with a state, cost, last action, and reference to the parent node

Implementation

https://inst.eecs.berkeley.edu/~cs188/fa24/projects/proj1/

python pacman.py

python pacman.py -h

python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch

Testando SearchAgent:

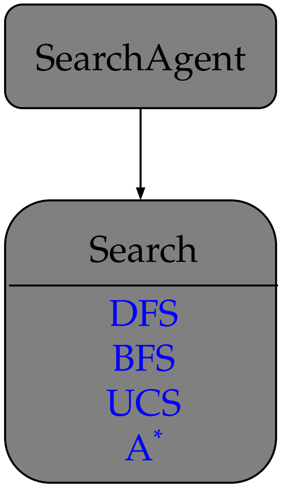
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch

- Padrão é rodar *DFS*
- Encontrar a posição (1,1) → *PositionSearchProblem*

```
def __init__(self, fn='depthFirstSearch', prob='PositionSearchProblem', heuristic='nullHeuristic'):
```

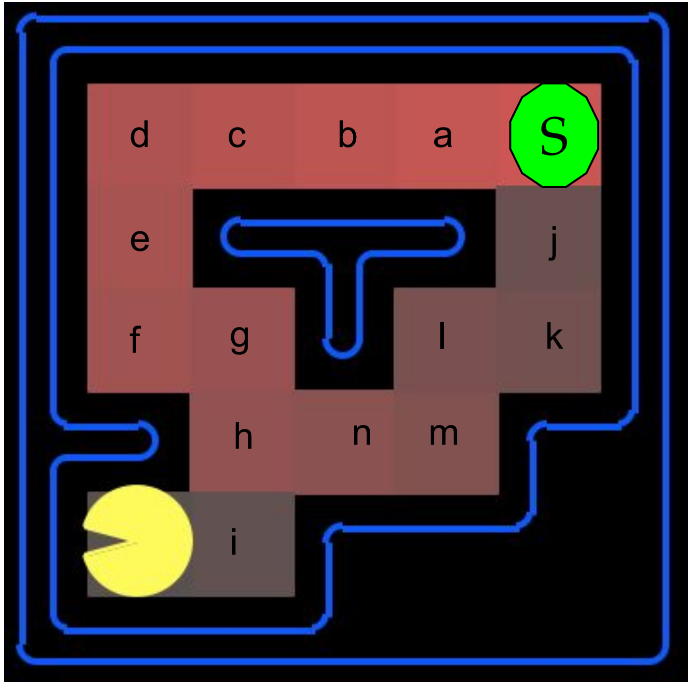
Main files

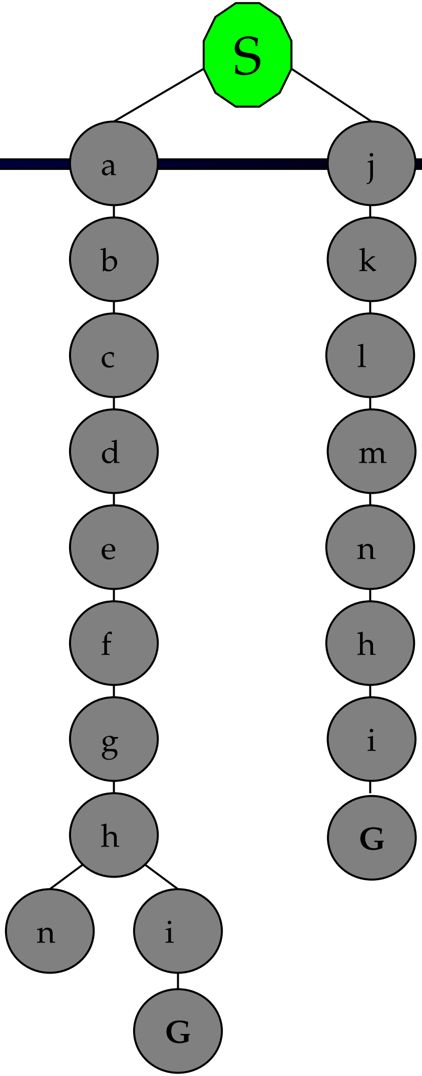
`search.py`	Where all of your search algorithms will reside.
`searchAgents.py`	Where all of your search-based agents will reside.
Files you might want to look at:	
`pacman.py`	The main file that runs Pacman games. This file describes a Pacman GameState type, which you use in this project.
`game.py`	The logic behind how the Pacman world works. This file describes several supporting types like AgentState, Agent, Direction, and Grid.
`util.py`	Useful data structures for implementing search algorithms.



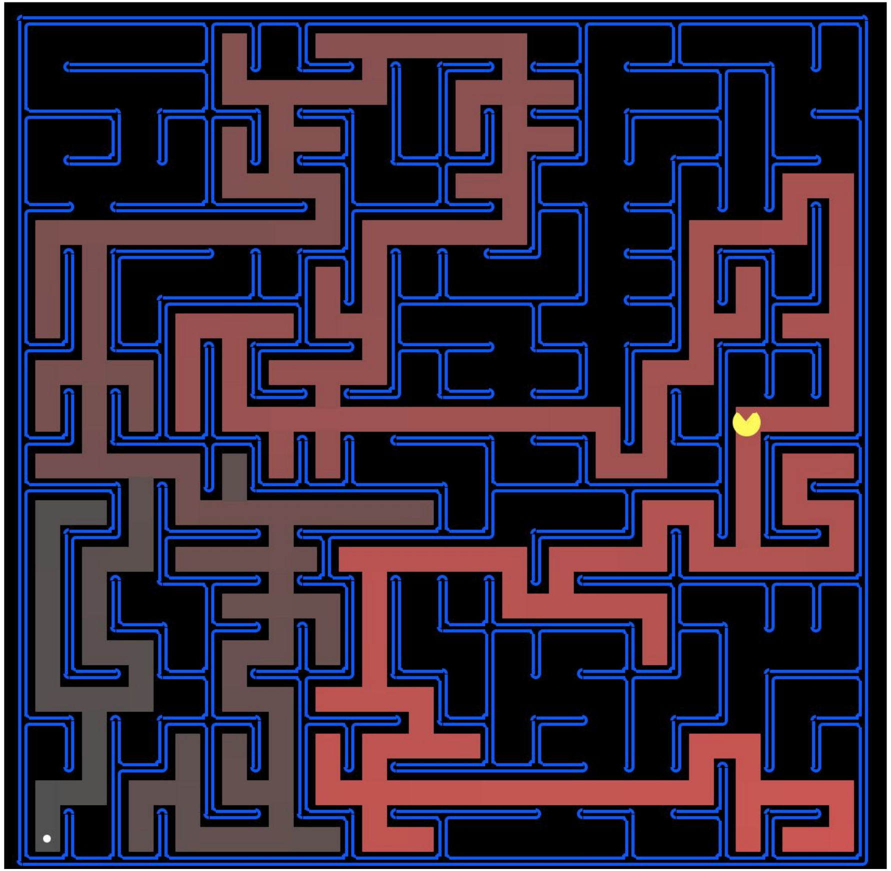
Código: https://inst.eecs.berkeley.edu/~cs188/fa24/assets/projects/search.zip

TinyMaze Layout



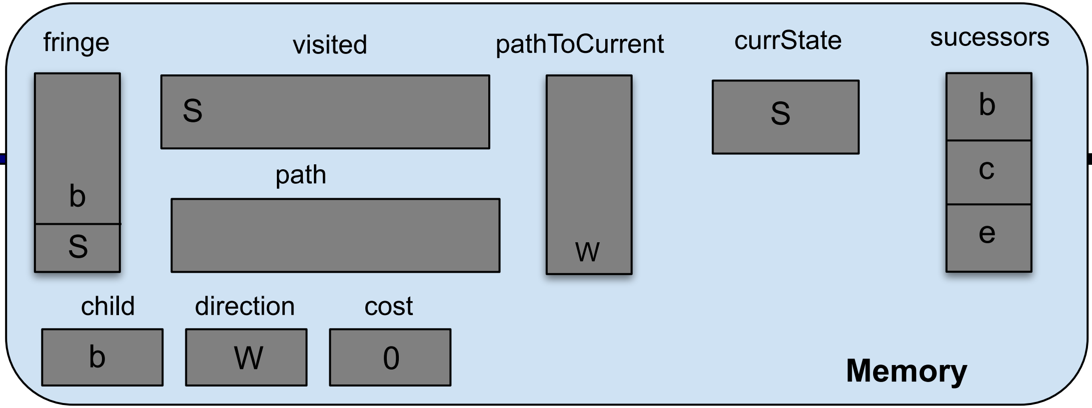


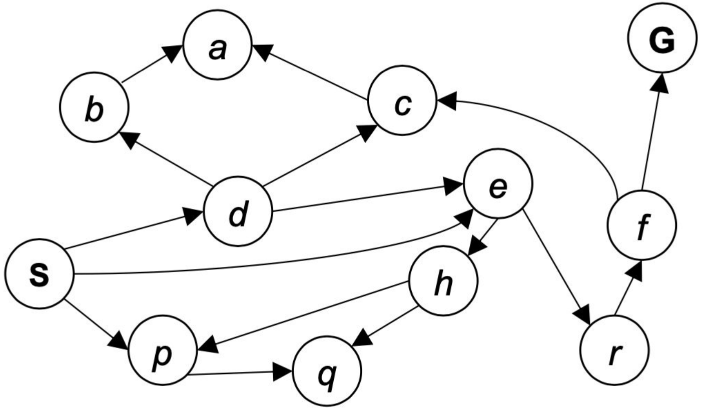
Qual o tamanho da árvore?

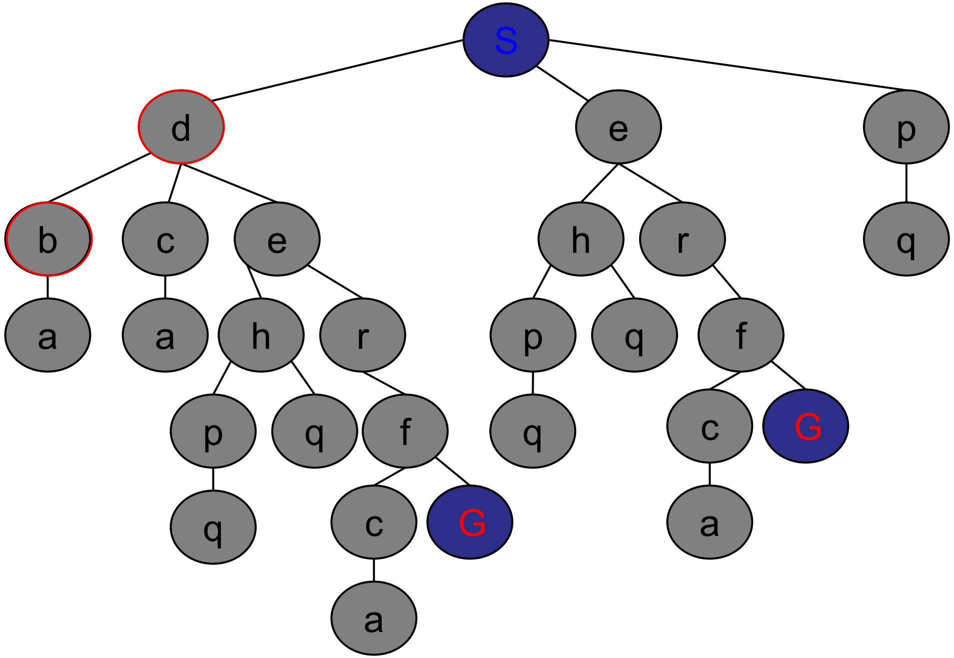


?

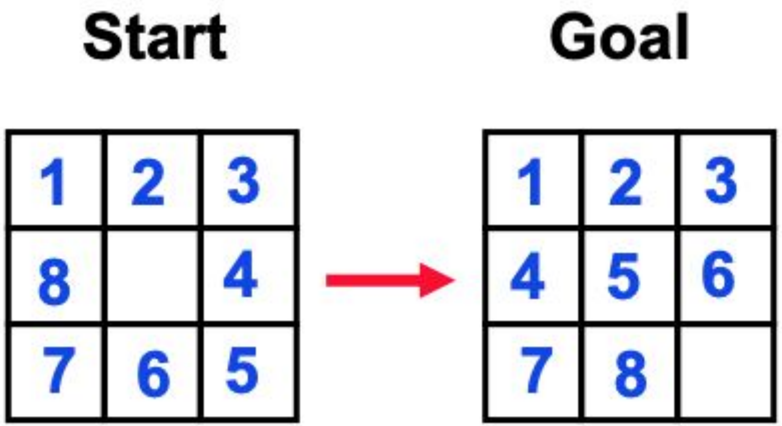
```python
def depthFirstSearch(problem):
    fringe = Stack()
    fringe.push(problem.getStartState())
    visited = []
    path=[]
    pathToCurrent=Stack()
    currState = fringe.pop()
    while not problem.isGoalState(currState):
        if currState not in visited:
            visited.append(currState)
            successors = problem.getSuccessors(currState)
            for child,direction,cost in successors:
                fringe.push(child)
                tempPath = path + [direction]
                pathToCurrent.push(tempPath)
        currState = fringe.pop()
        path = pathToCurrent.pop()
    return path
```





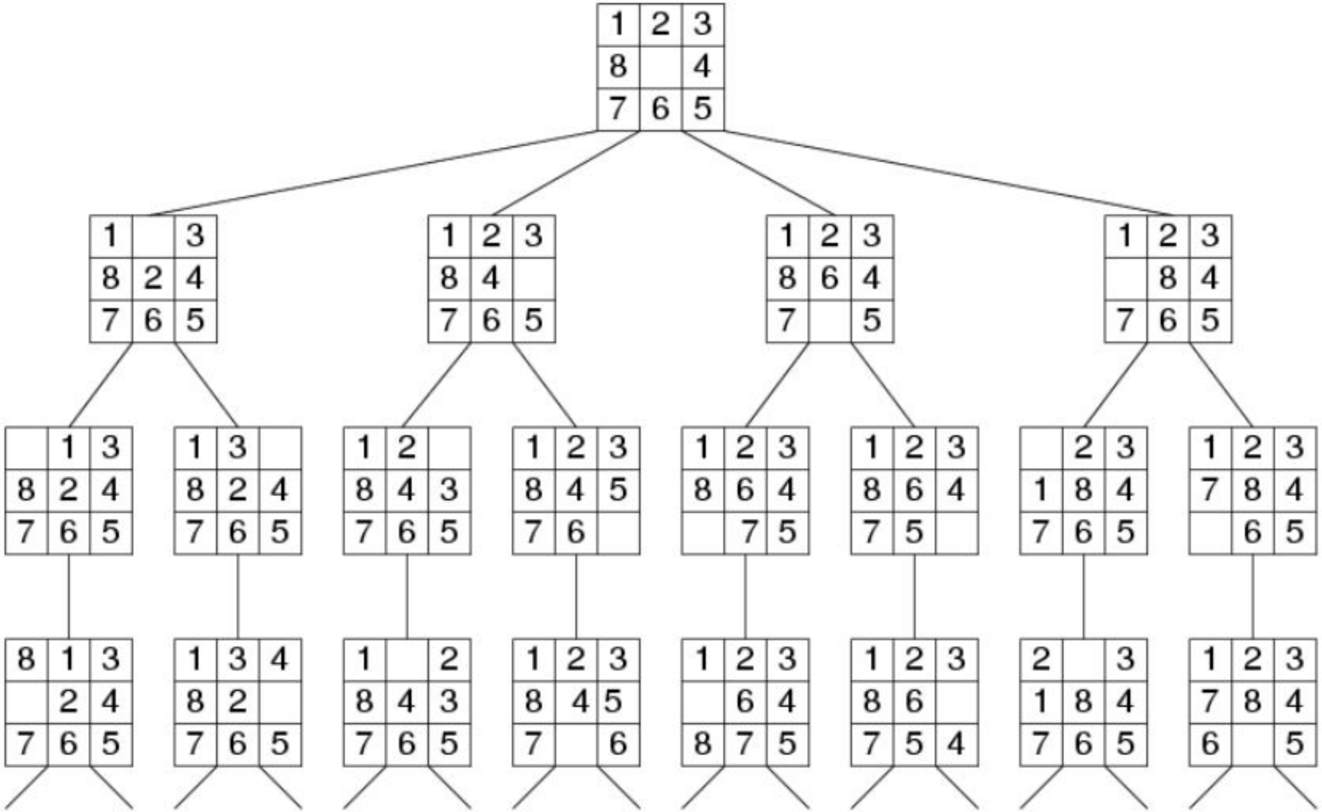


The 8-puzzle



Como modelar isto?

Tree Structure



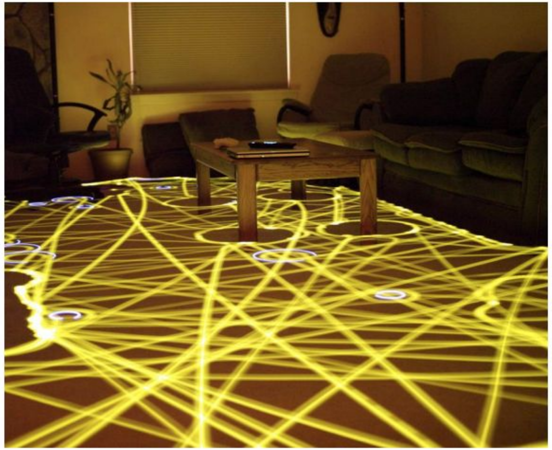
Facebook friends

Find the shortest chain of Facebook friends that goes from Person A to Person B

Robotics

Commercial

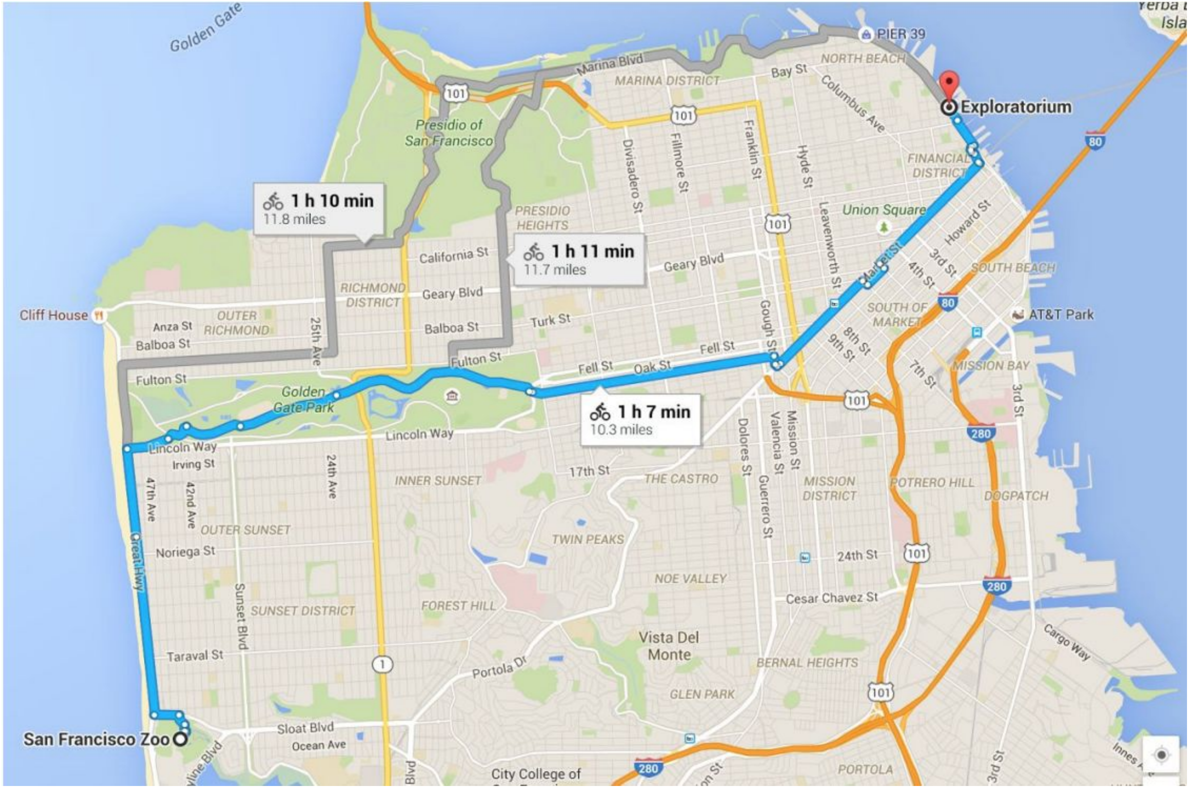
Search & Rescue



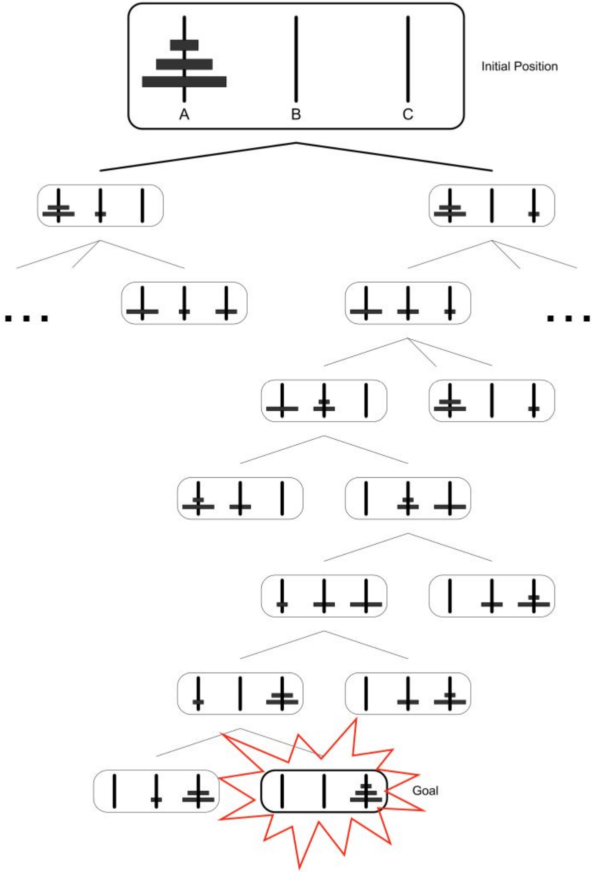
Domestic

Source: https://cs.stanford.edu/people/abisee/gs.pdf

Route Planning



Hanoi Tower

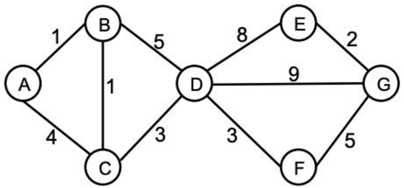


Time and Space Complexity

Time and memory usage when b = 10:

solution depth	nodes considered	time	memory
0	1	1 millisecond	100 bytes
4	11,111	11 seconds	1 megabyte
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	***1 terabyte***
12	10^{12}	35 years	111 terabytes

Exercise 1



Node	h_1	h_2
A	9.5	10
B	9	12
C	8	10
D	7	8
E	1.5	1
F	4	4.5
G	0	0

Consider the state space graph shown above. A is the start state and G is the goal state. The costs for each edge are shown on the graph. Each edge can be traversed in both directions. Note that the heuristic h_1 is consistent but the heuristic h_2 is not consistent.

(a) Possible paths returned

For each of the following graph search strategies (*do not answer for tree search*), mark which, if any, of the listed paths it could return. Note that for some search strategies the specific path returned might depend on tie-breaking behavior. In any such cases, make sure to mark *all* paths that could be returned under some tie-breaking scheme.

Search Algorithm	**A-B-D-G**	**A-C-D-G**	**A-B-C-D-F-G**
Depth first search			
Breadth first search			
Uniform cost search			
A* search with heuristic h_1			
A* search with heuristic h_2			

Exercise 2

Try running our 8-puzzle solver on the initial state shown at right!

	8	7
6	5	4
3	2	1

Hanoi Tower

Implementation using Depth First Search

Homework - 25%

1. Python DFS, BFS, UCS (Berkeley framework)
2. Exercise 1 - by hand
3. Hanoi Tower - Python

<u>Deadline</u>: May, 27rd, 2025