Lecture 2

- Music:
  - **9 to 5** - Dolly Parton
  - **Por una Cabeza** (instrumental) - written by Carlos Gardel, performed by Horacio Rivera
  - **Bear Necessities** - from The Jungle Book, performed by Anthony the Banjo Man

AI in the News

Introducing TensorNetwork, an Open Source Library for Efficient Tensor Calculations

Tuesday, June 4, 2019

Posted by Chase Roberts, Research Engineer, Google AI and Stefan Leichenauer, Research Scientist, X

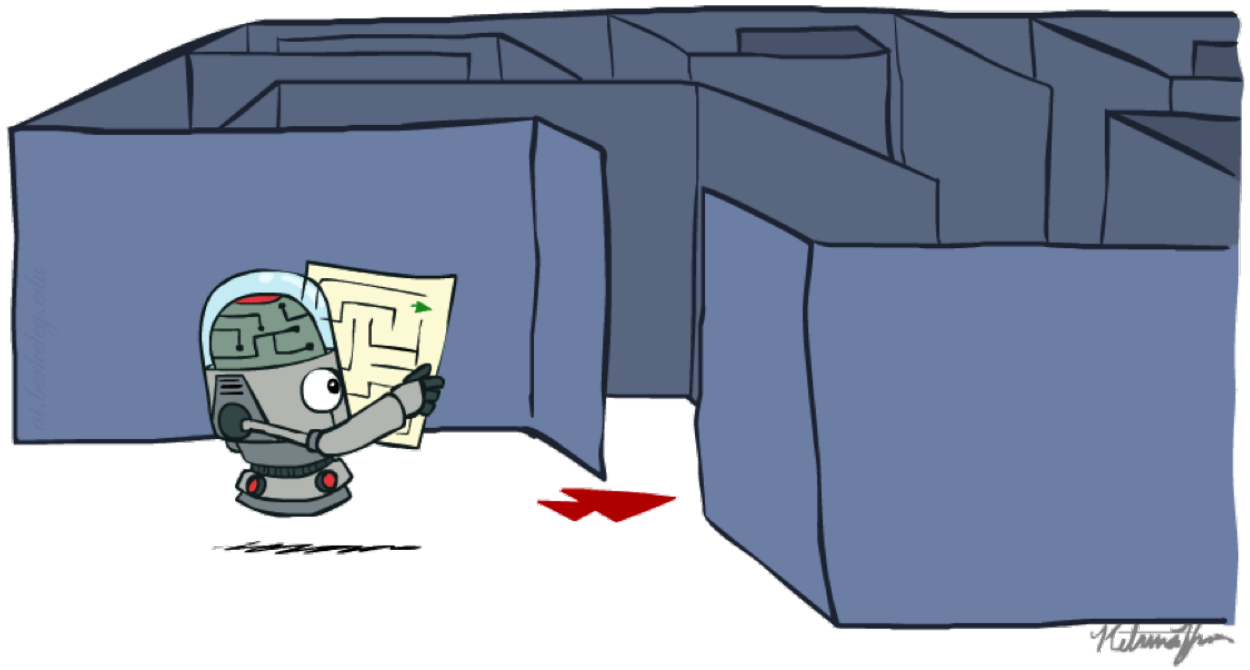
- https://ai.googleblog.com/2019/06/introducing-tensornetwork-open-source.html
- Use cases in physics!
  - Approximating quantum states is a typical use-case for tensor networks in physics.
  - "... we describe a tree tensor network (TTN) algorithm for approximating the ground state of either a periodic quantum spin chain (1D) or a lattice model on a thin torus (2D)"

Announcements

- Lecture moved to North Gate Hall, room 105, starting Wednesday (tomorrow)
- Project 0: Python Tutorial
  - Optional, but please do it. It walks you through the project submission process.
- Homework 0: Math self-diagnostic
  - Optional, but important to check your preparedness for second half of the class.
- Project 1: Search is out!
  - Best way to test your programming preparedness
  - Use post @5 on Piazza to search for a project partner if you don't have one!
- HW 1 is out!
  - 3 components: electronic, written, and self-assessment
- Sections start this week
  - Sorry for the lecture-discussion misalignment!
- Make sure you are signed up for Piazza and Gradescope
  - Check all the pinned posts on Piazza
- You can give me feedback through this link: https://tinyurl.com/aditya-feedback-form

CS 188: Artificial Intelligence

Search



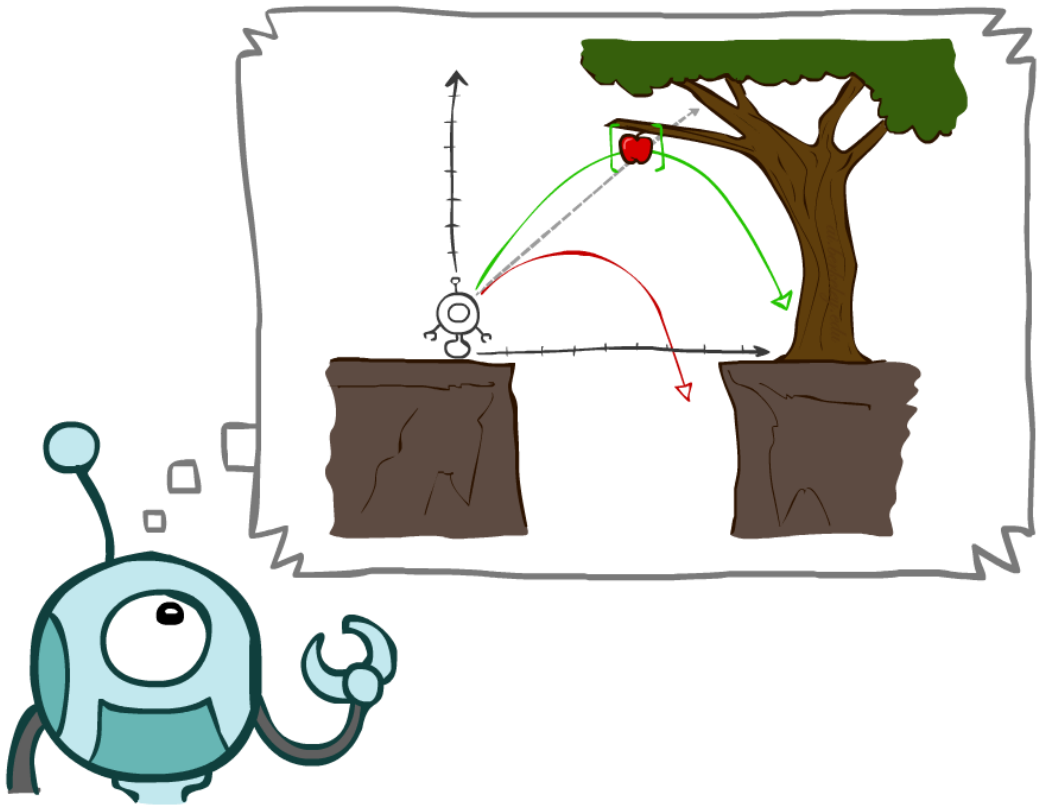
Instructors: Sergey Levine & Stuart Russell

University of California, Berkeley

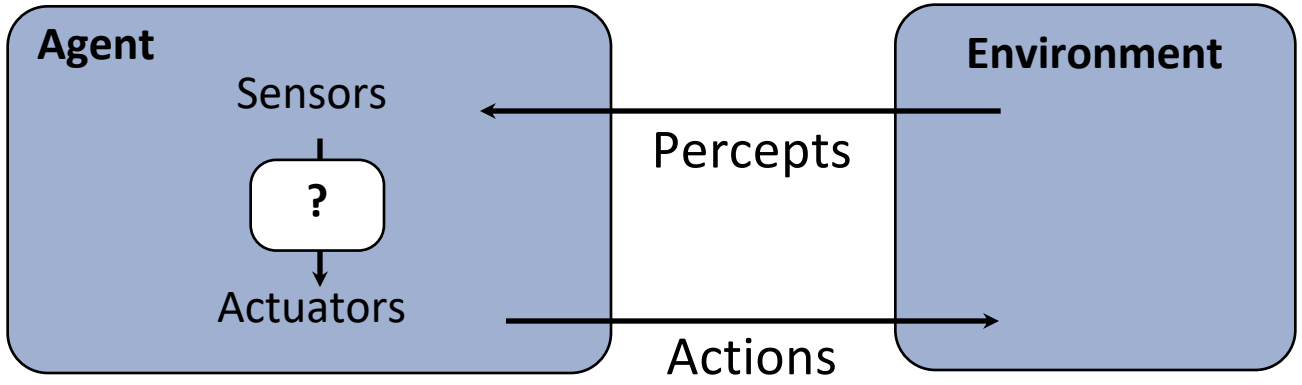
[slides adapted from Dan Klein, Pieter Abbeel]

Today

- 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***
- Q: What are some examples of this?

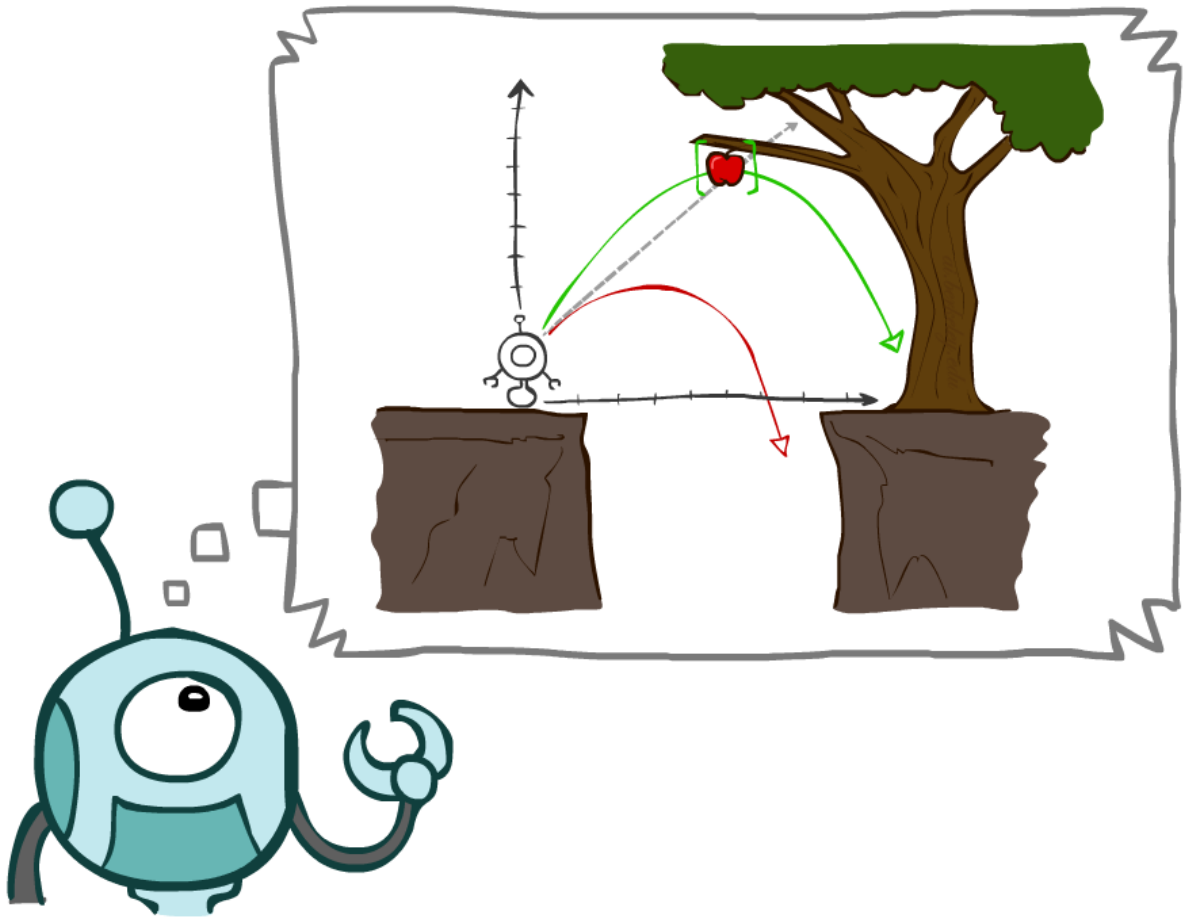
Rationality

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

Agent design

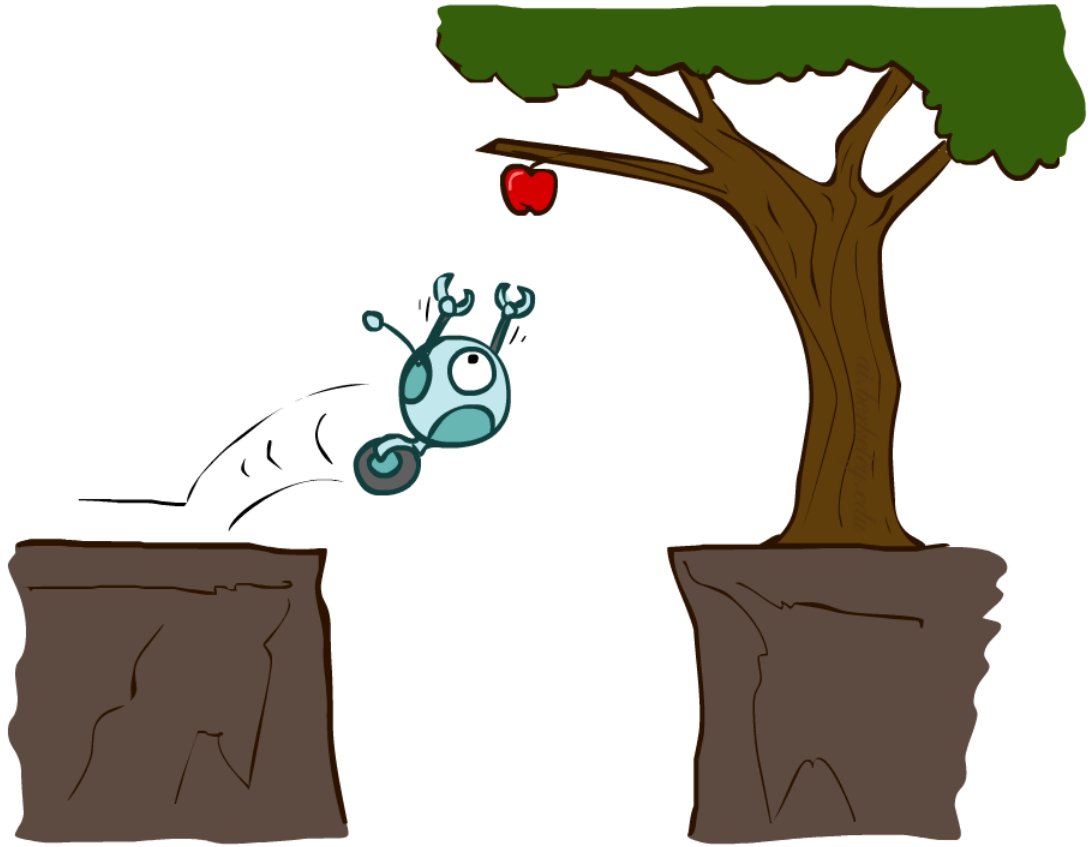
- **The environment type largely determines the agent design**
  - ***Fully/partially observable*** => agent requires ***memory*** (internal state)
  - ***Discrete/continuous*** => agent may not be able to enumerate ***all states***
  - ***Stochastic/deterministic*** => agent may have to prepare for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

Agents that Plan



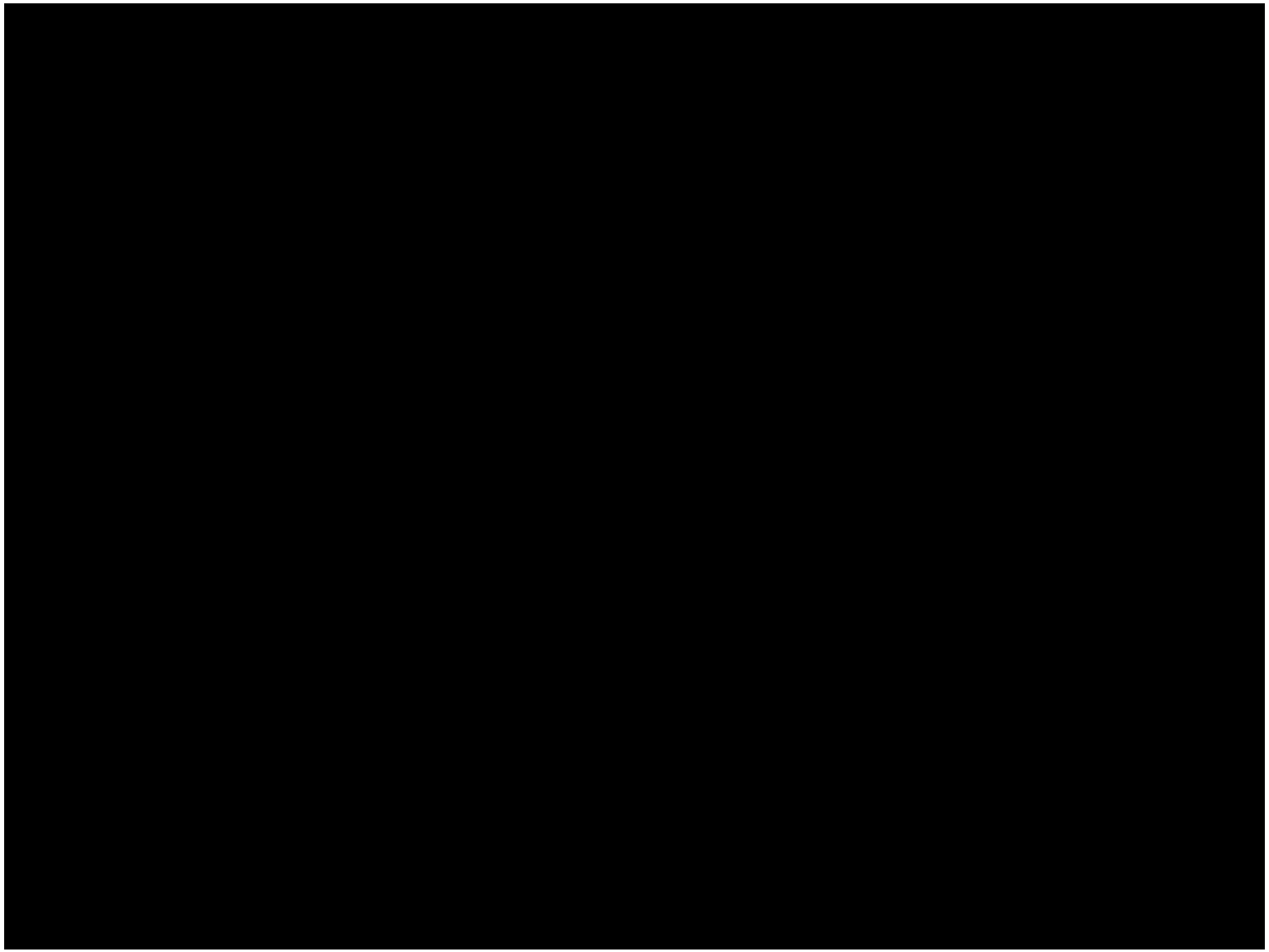
Reflex Agents

- 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?

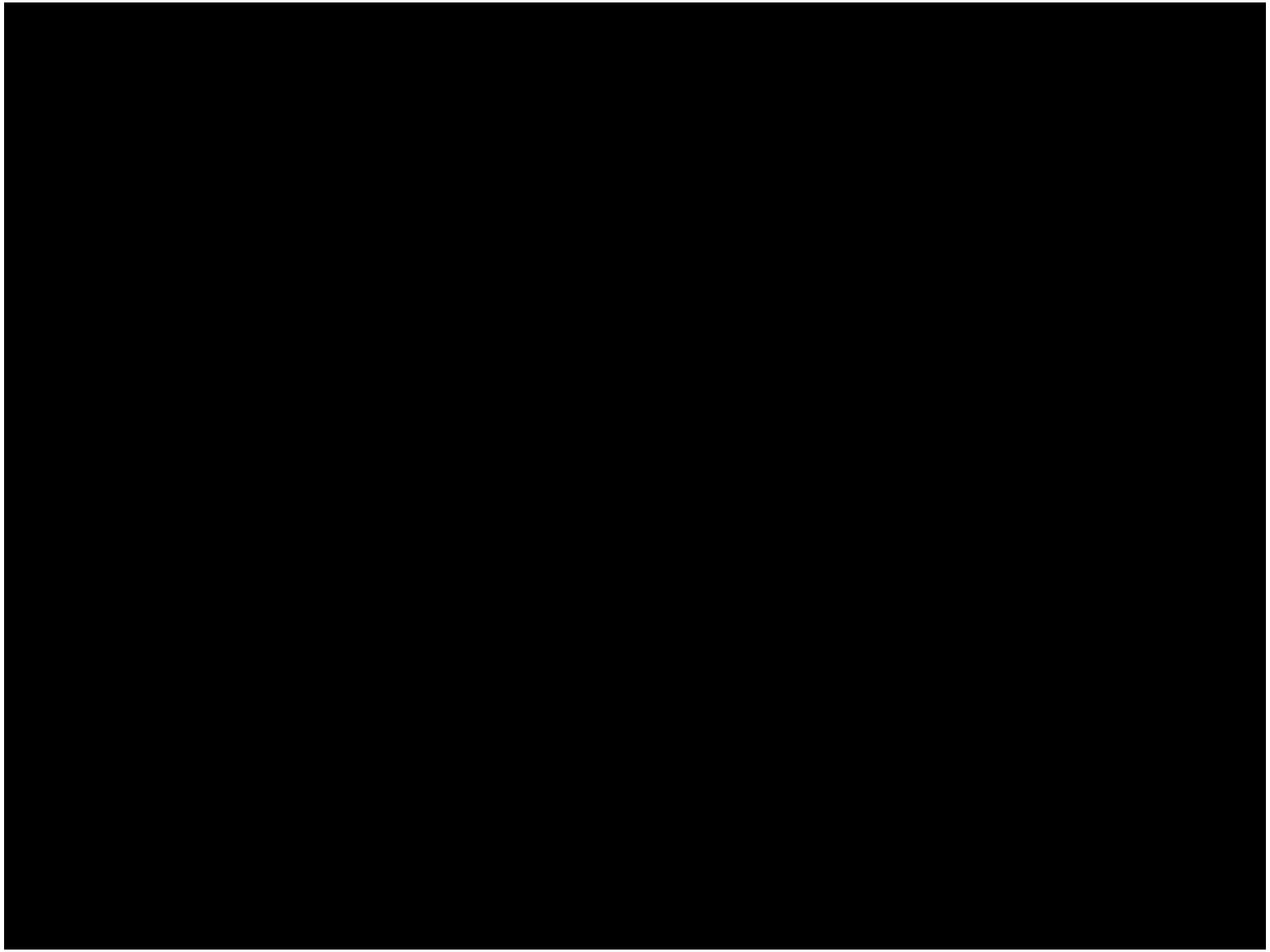


[Demo: reflex optimal (L2D1)]
[Demo: reflex optimal (L2D2)]

Video of Demo Reflex Optimal

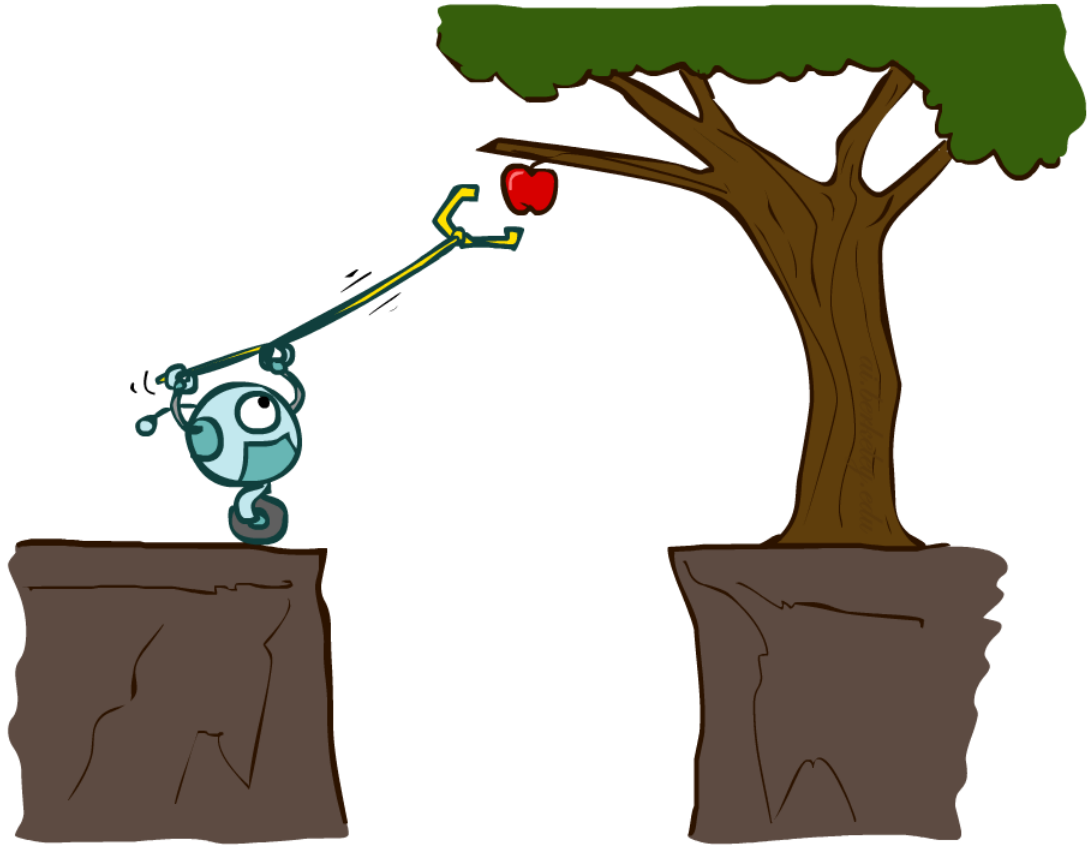


Video of Demo Reflex Odd



Planning Agents

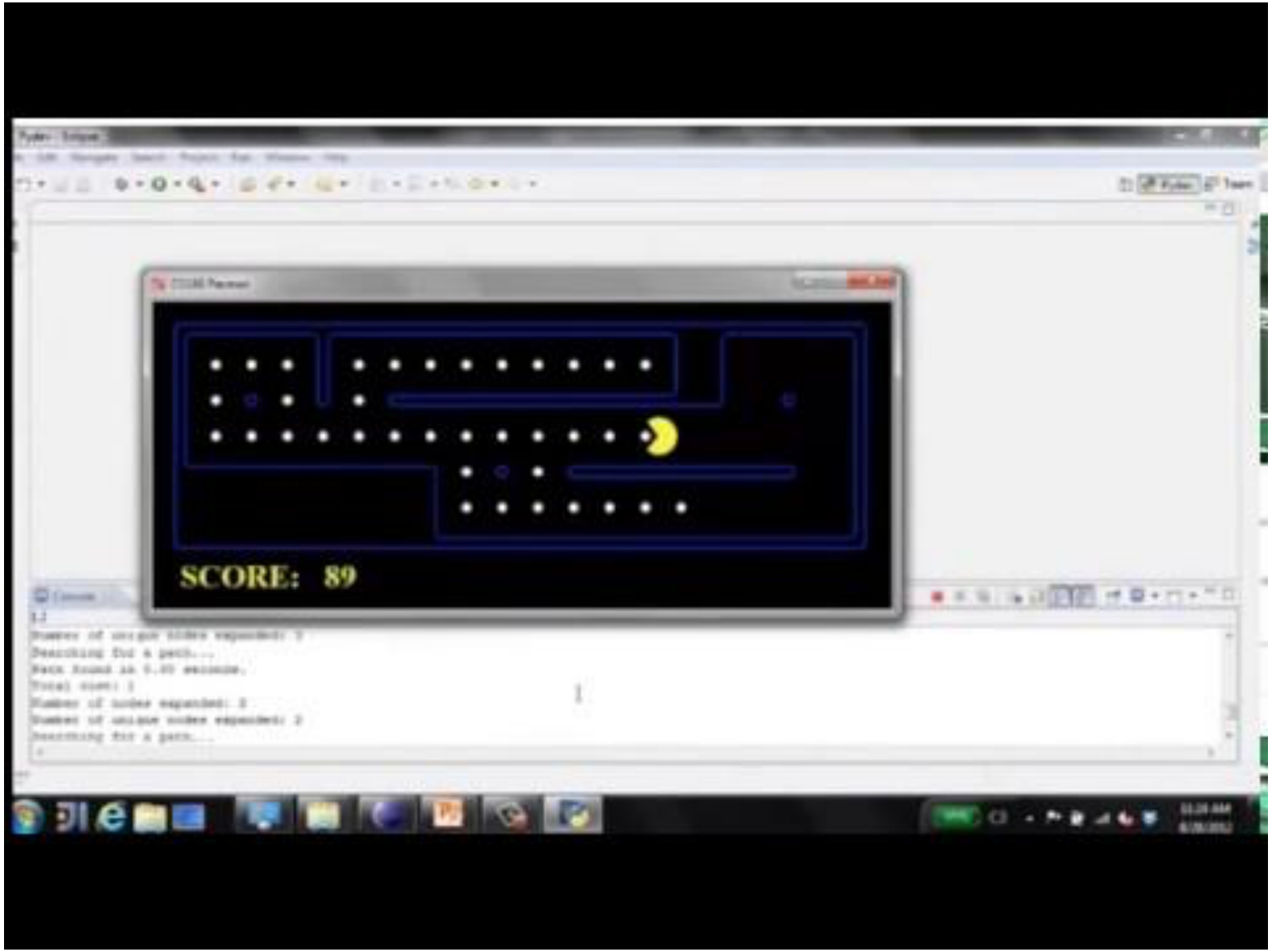
- Planning agents:
  - Ask “what if”
  - Decisions based on (hypothesized) consequences of actions
  - Must have a model of how the world evolves in response to actions
  - Must formulate a goal (test)
  - Consider how the world WOULD BE
- Planning vs. replanning



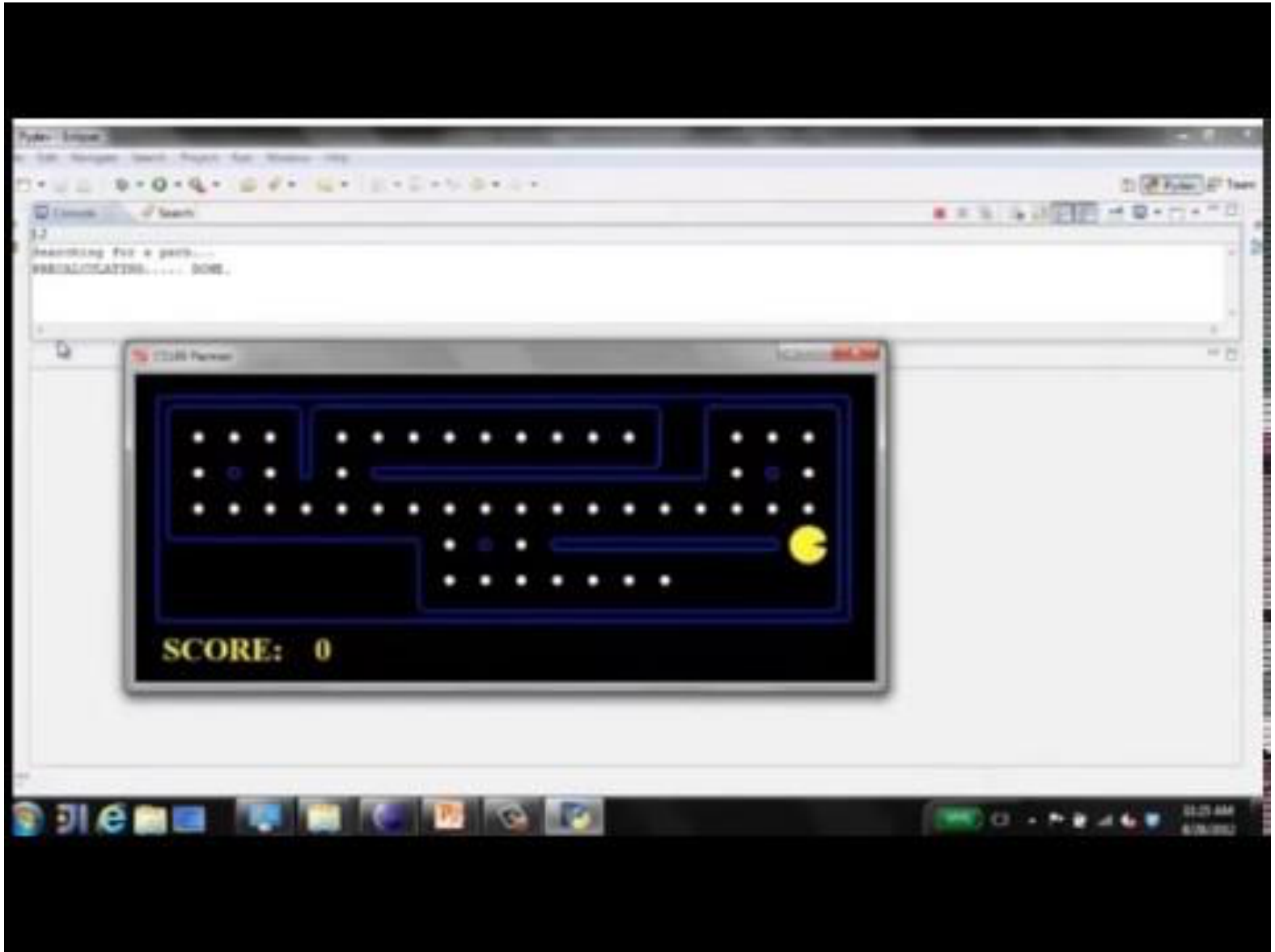
[Demo: re-planning (L2D3)]

[Demo: mastermind (L2D4)]

Video of Demo Replanning



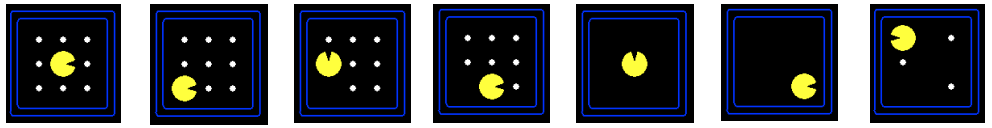
Video of Demo Mastermind



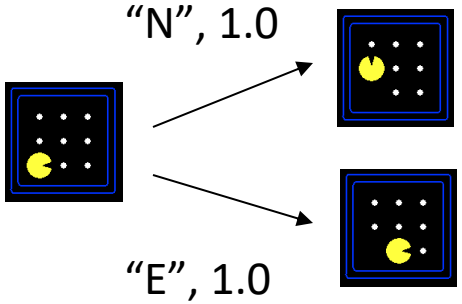
Search Problems

Search Problems

- A search problem consists of:
  - A state space



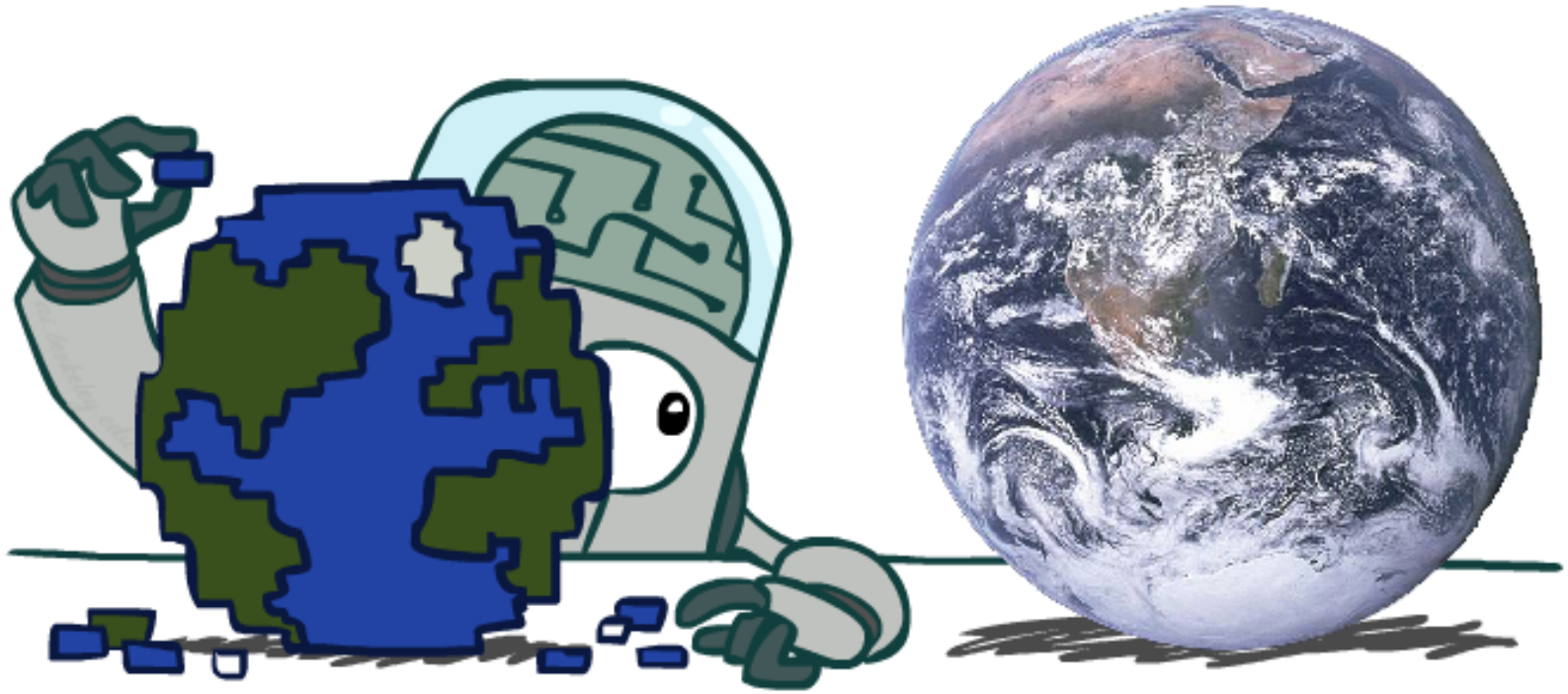
  - A successor function (with actions, costs)



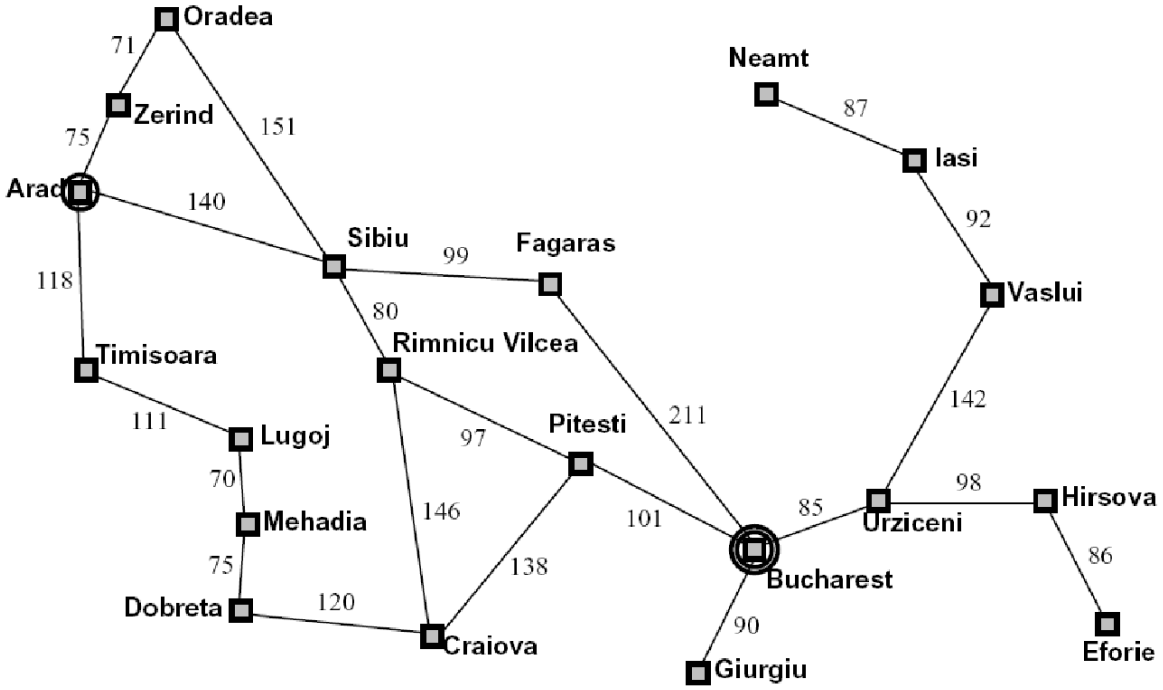
  - A start state and a goal test

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

Search Problems Are Models



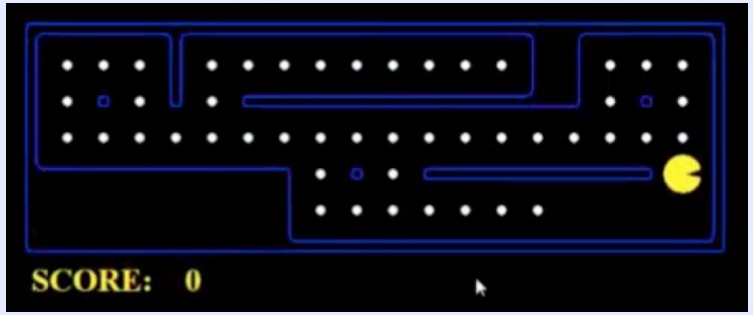
Example: Traveling in Romania



- 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?

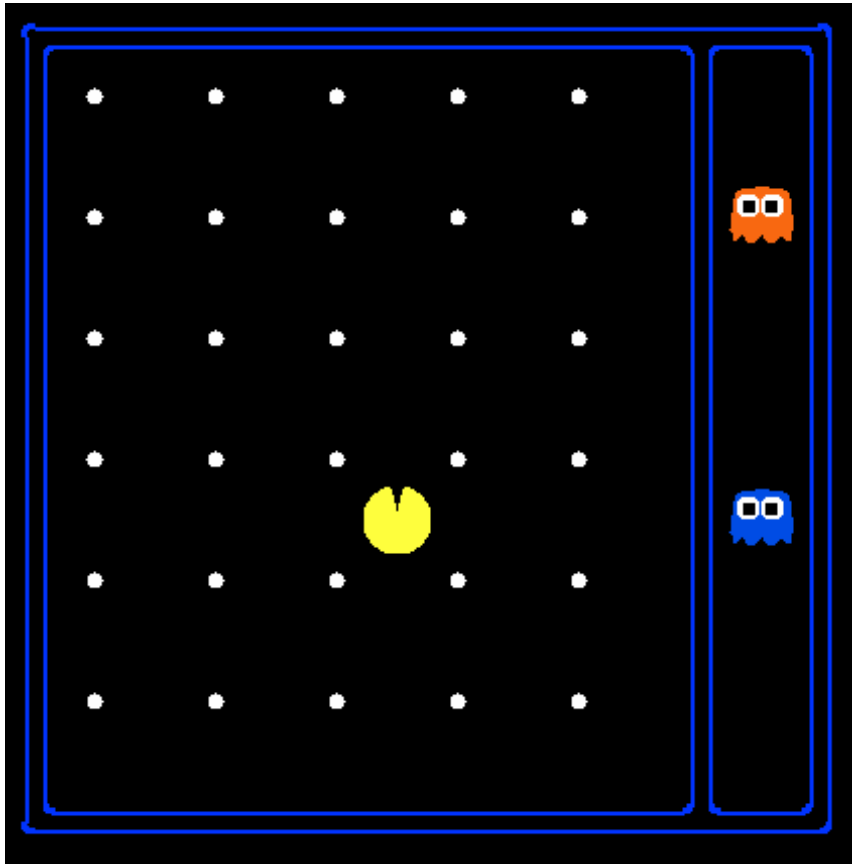
- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

- How many
  - World states?

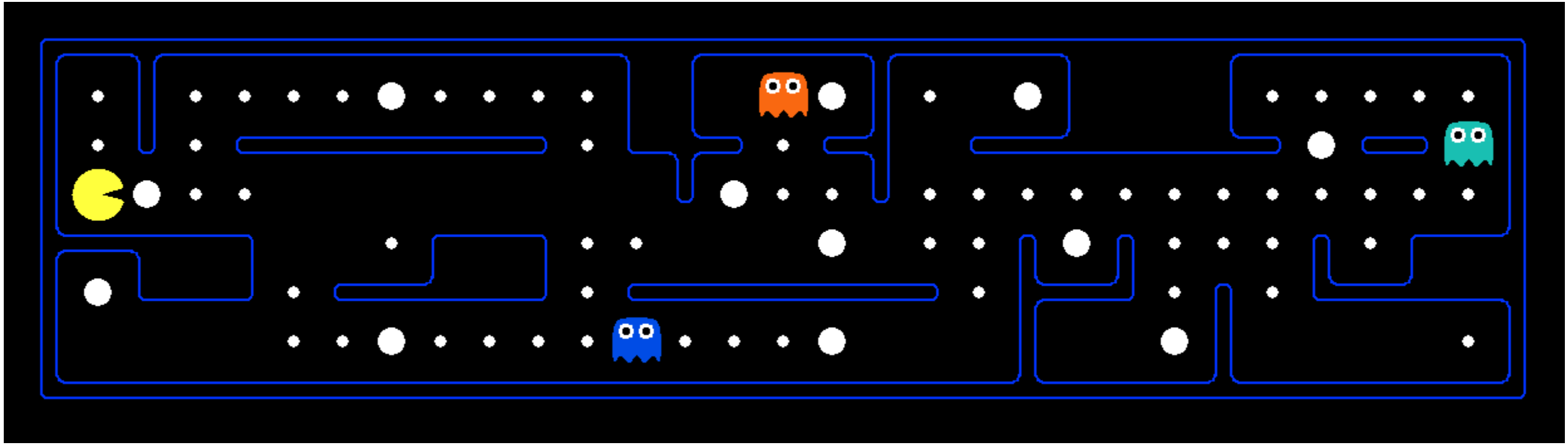
    $120x(2^{30})x(12^{2})x4$
  - States for pathing?

    120
  - States for eat-all-dots?

    $120x(2^{30})$



Quiz: Safe Passage

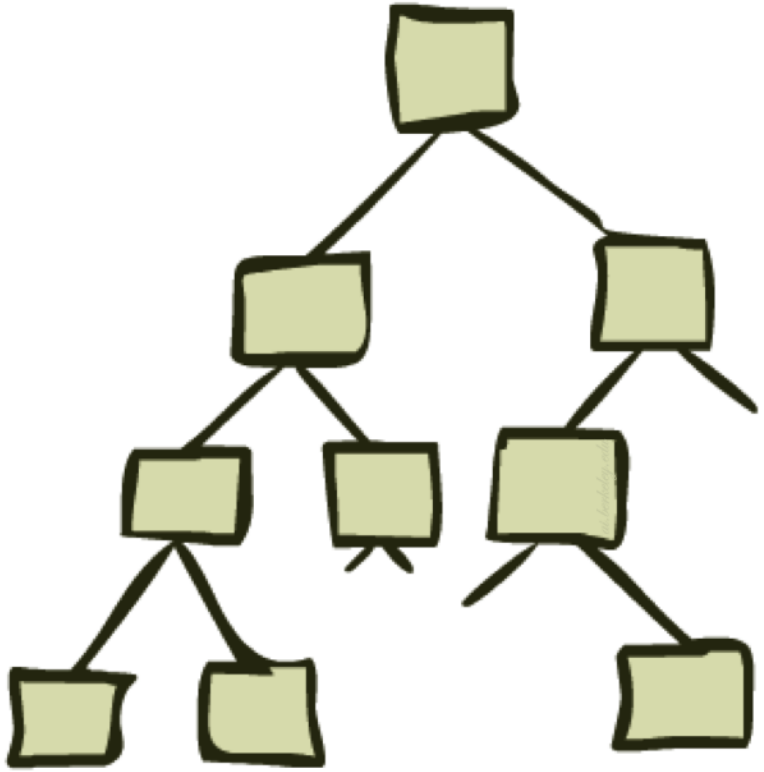


- Problem: eat all dots while keeping the ghosts perma-scared
- What does the state space have to specify?
  - (agent position, dot booleans, power pellet booleans, remaining scared time)

Agent design

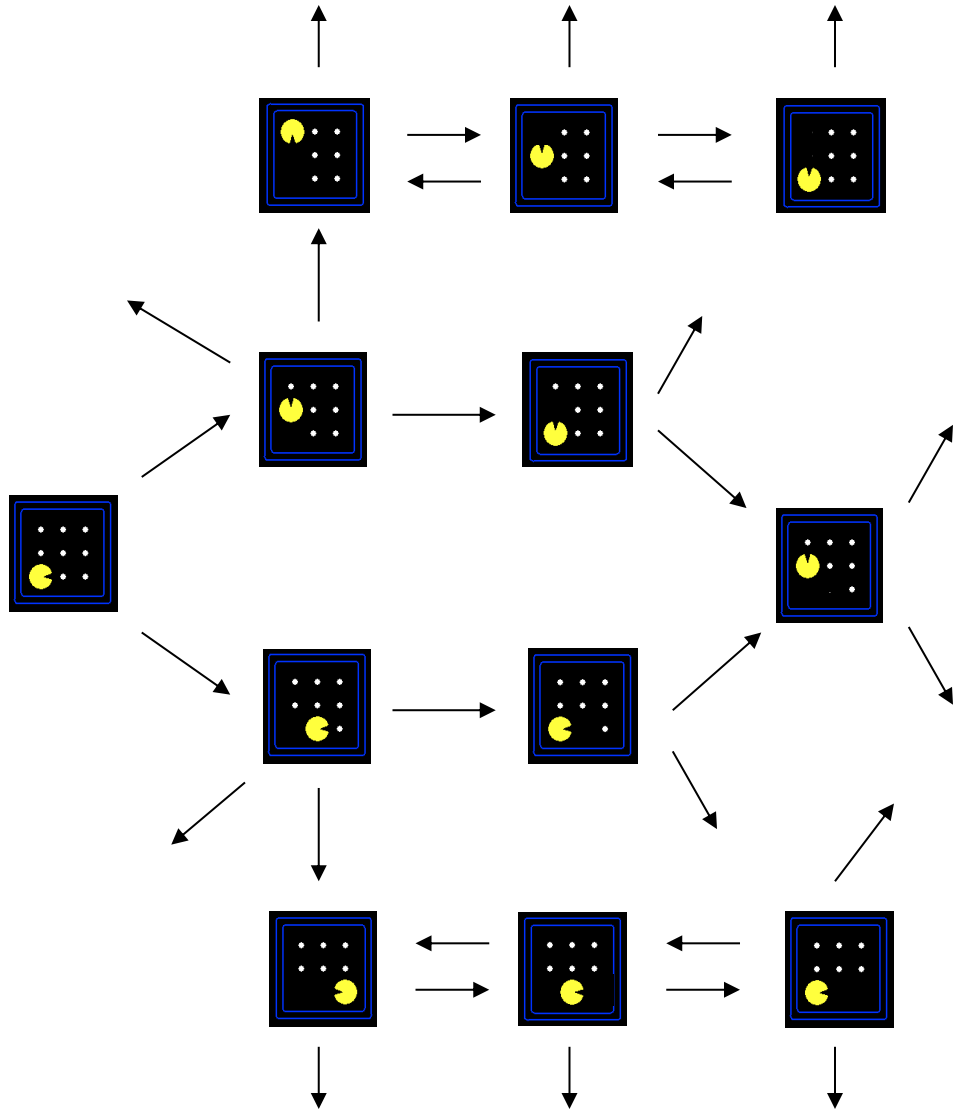
- **The environment type largely determines the agent design**
  - ***Fully/partially observable*** => agent requires ***memory*** (internal state)
  - ***Discrete/continuous*** => agent may not be able to enumerate ***all states***
  - ***Stochastic/deterministic*** => agent may have to prepare for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

State Space Graphs and Search Trees



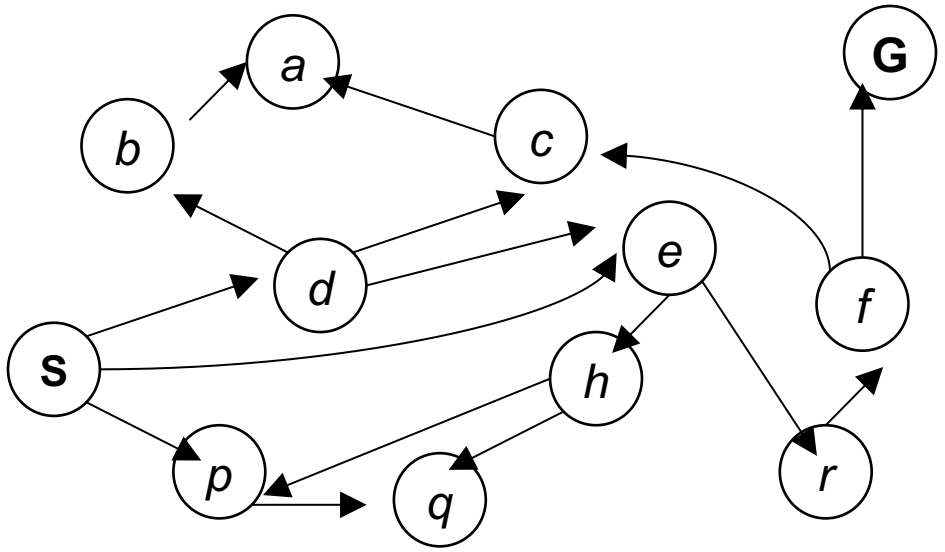
State Space Graphs

- 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



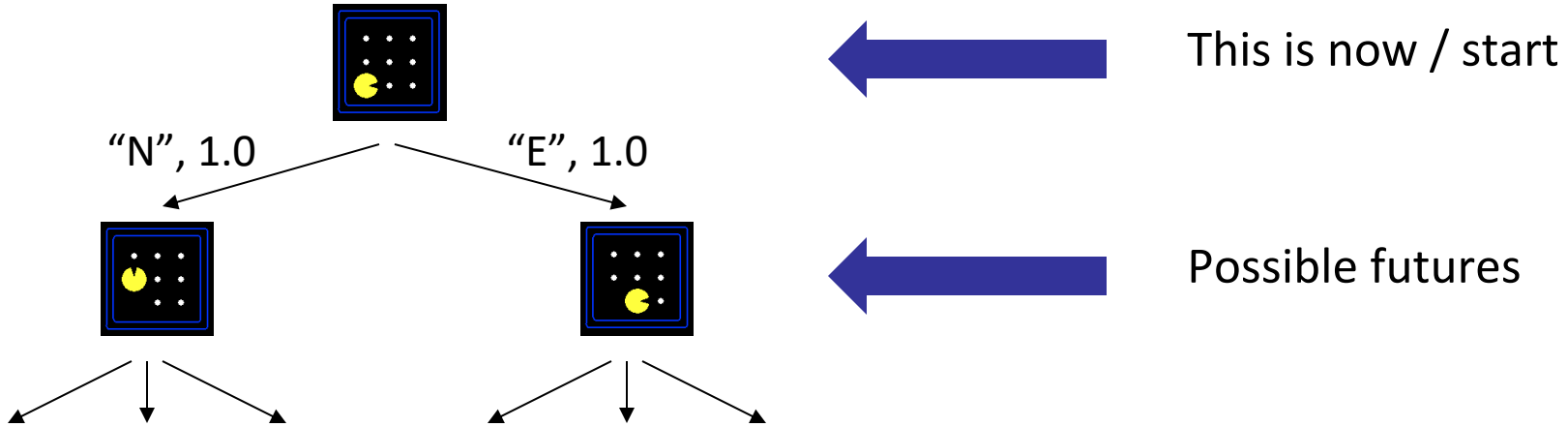
State Space Graphs

- 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

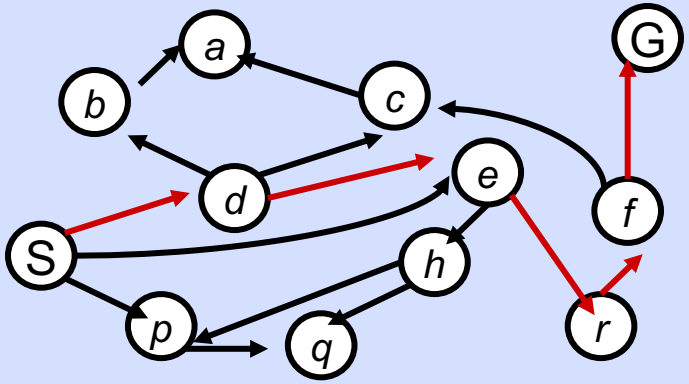
Search Trees



- 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

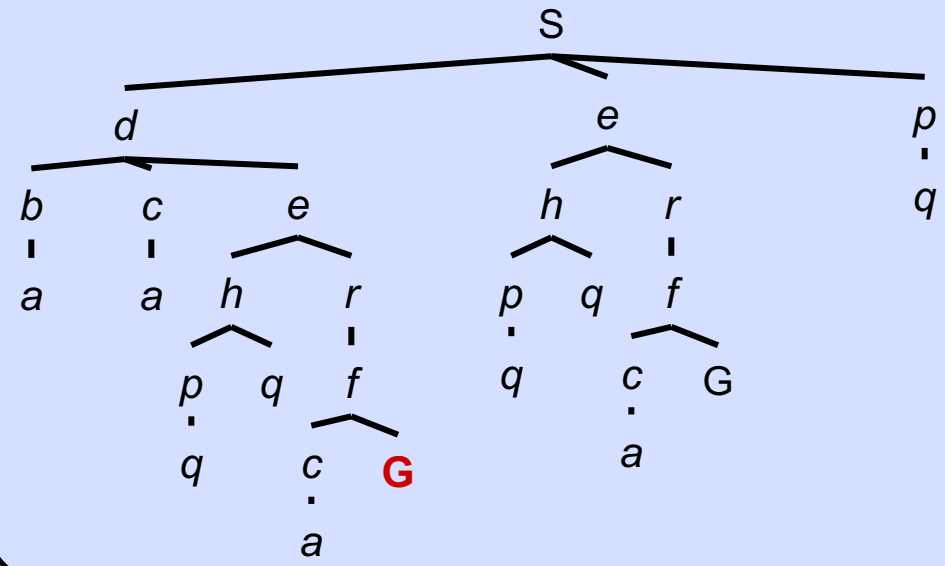
State Space Graph



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

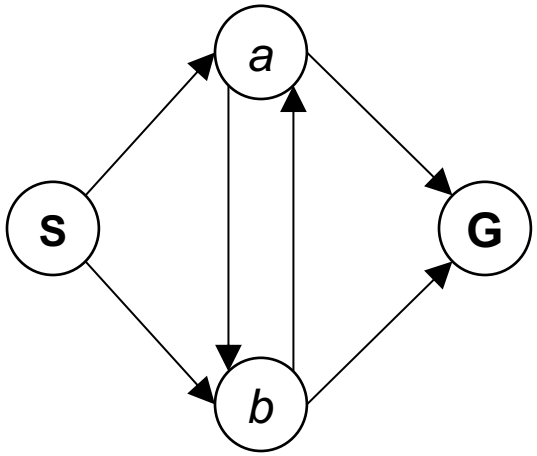
We construct both on demand – and we construct as little as possible.

Search Tree



Quiz: State Space Graphs vs. Search Trees

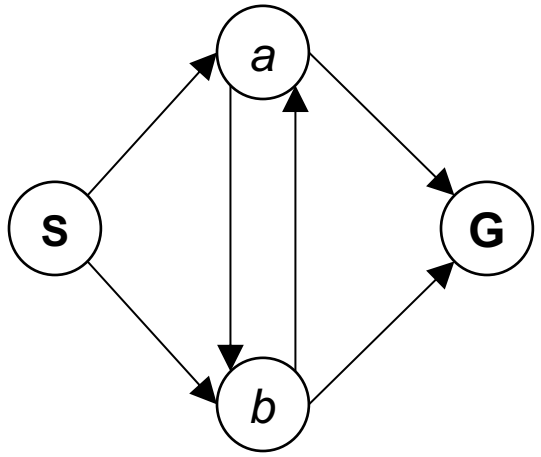
Consider this 4-state graph:



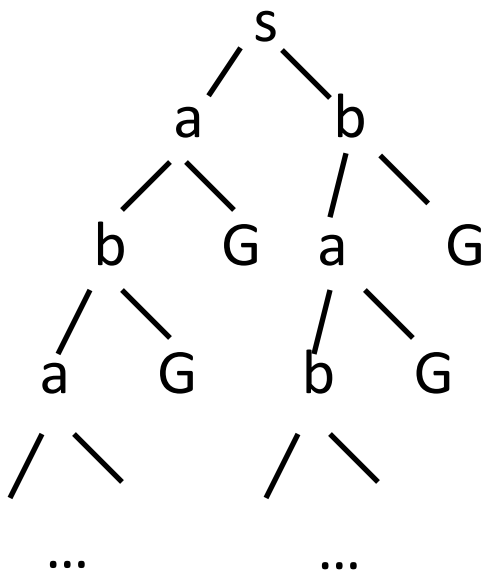
How big is its search tree (from S)?

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



How big is its search tree (from S)?

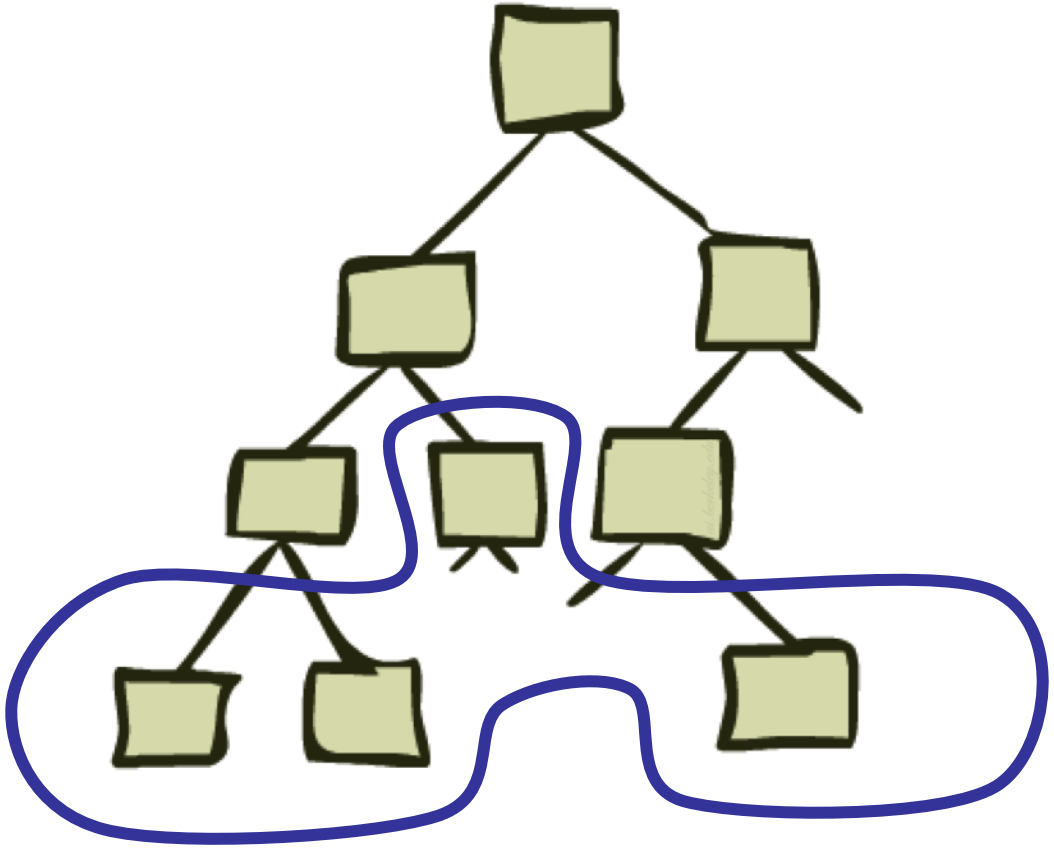


Important: Lots of repeated structure in the search tree!

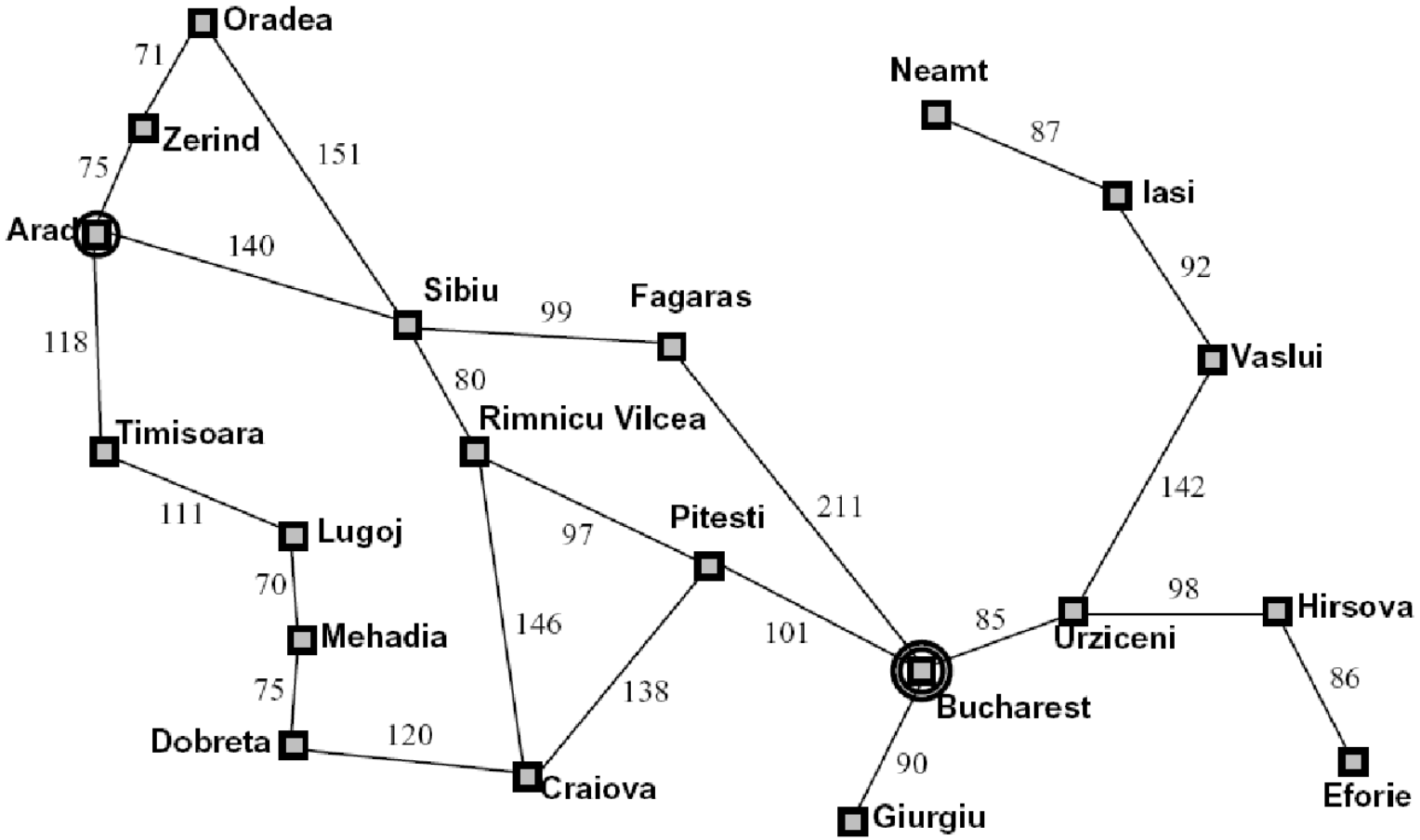
Break!

- Stand up and stretch
- Talk to your neighbors

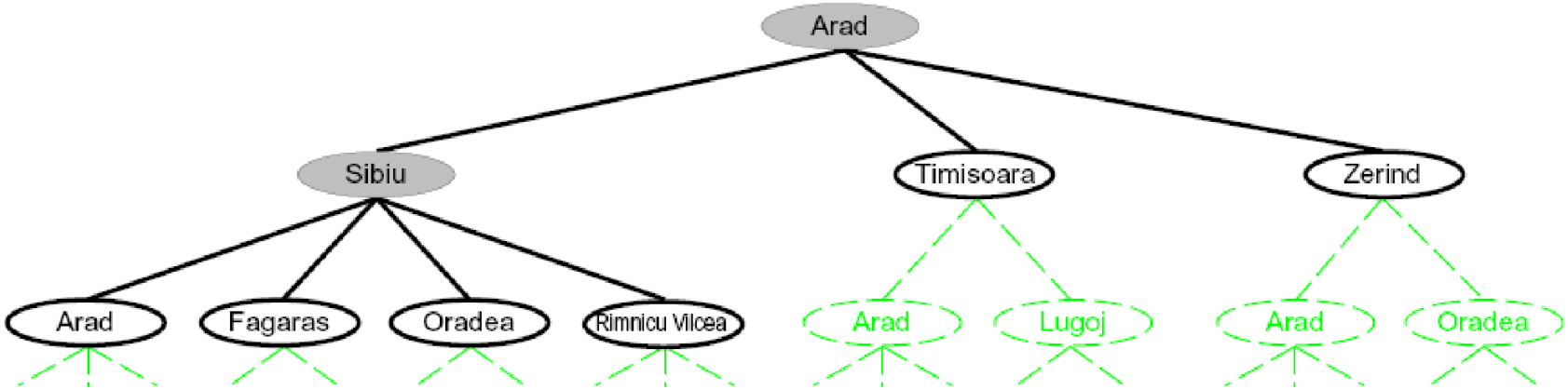
Tree Search



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

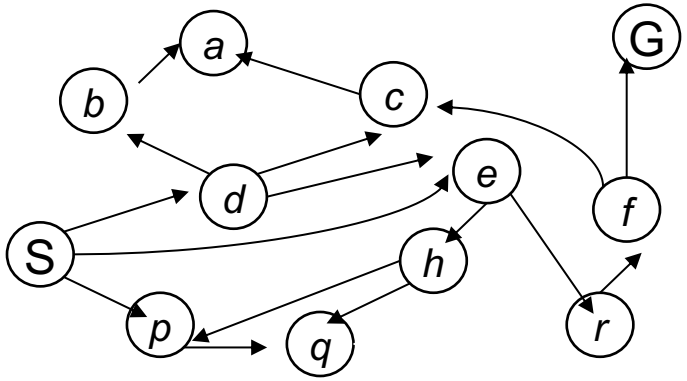
General Tree Search

```
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
```

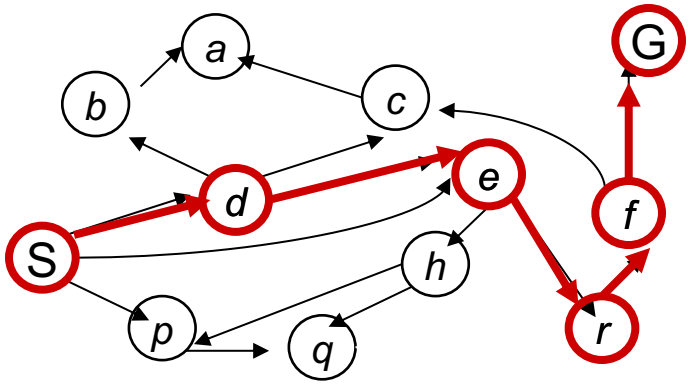
- Important ideas:
  - Fringe
    - the set of nodes that are to-be-visited
  - Expansion
    - the process of 'visiting' a node
  - Exploration strategy
    - how do we decide which fringe node to visit?

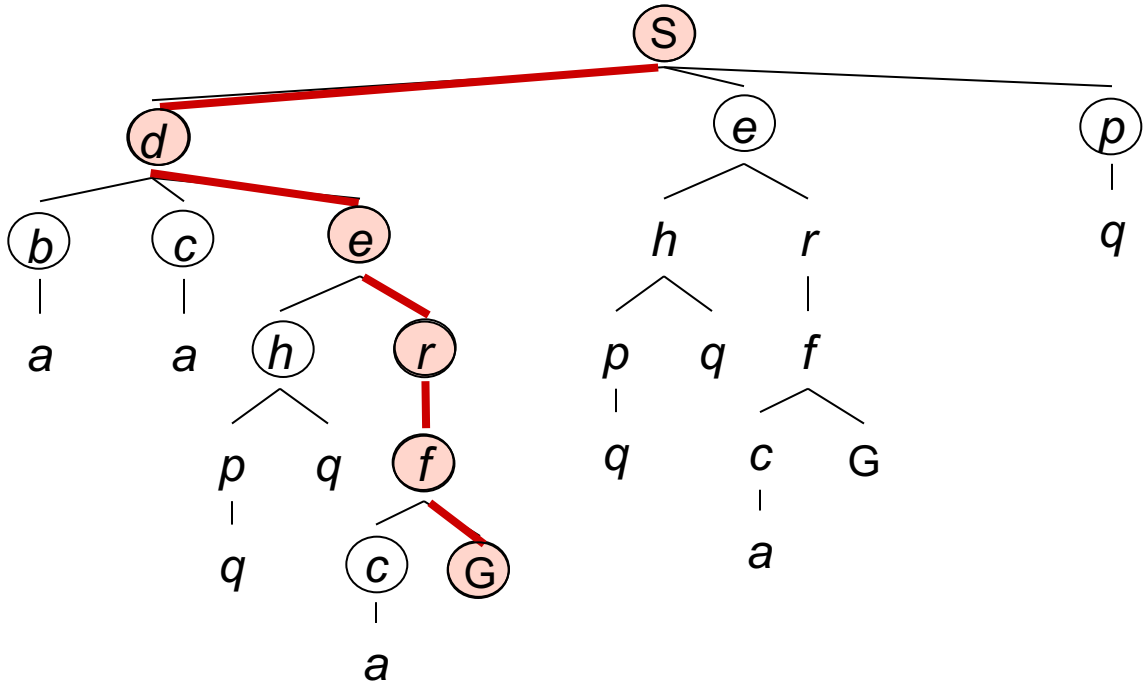
- Main question: which fringe nodes to explore?

Example: Tree Search



Example: Tree Search





~~s~~

~~s → d~~

s → e

s → p

s → d → b

s → d → c

~~s → d → e~~

s → d → e → h

~~s → d → e → r~~

~~s → d → e → r → f~~

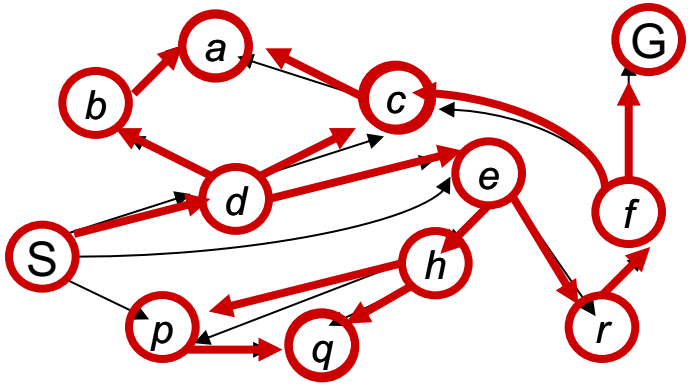
s → d → e → r → f → c

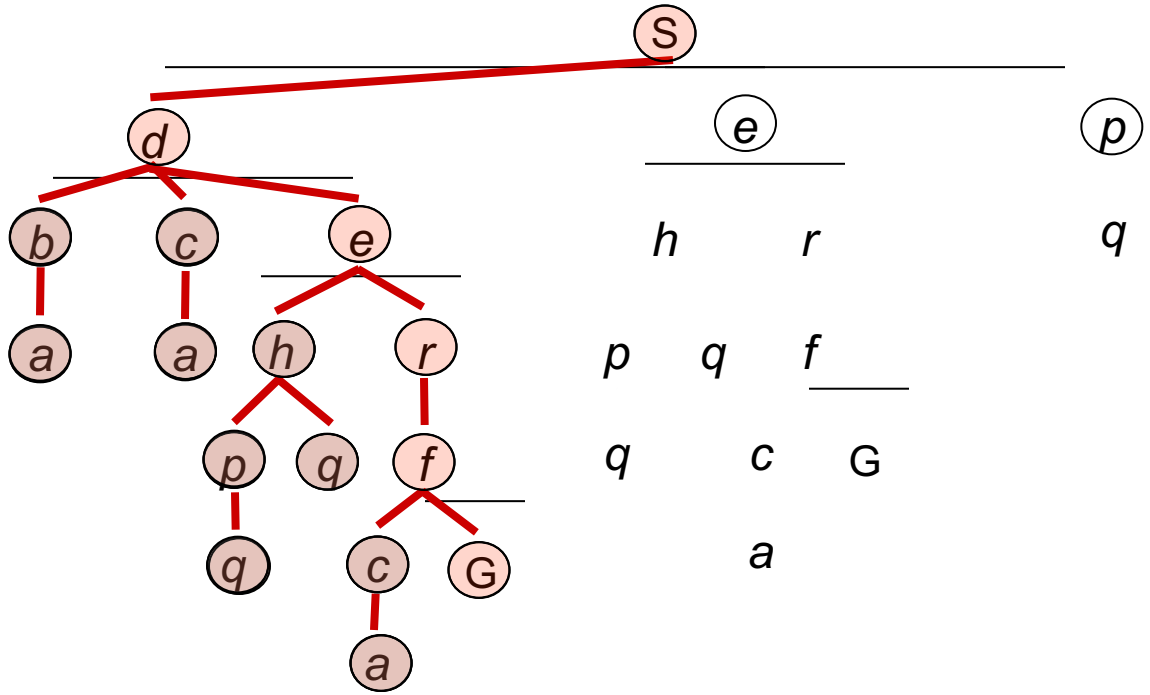
~~s → d → e → r → f → G~~

Depth-First Search

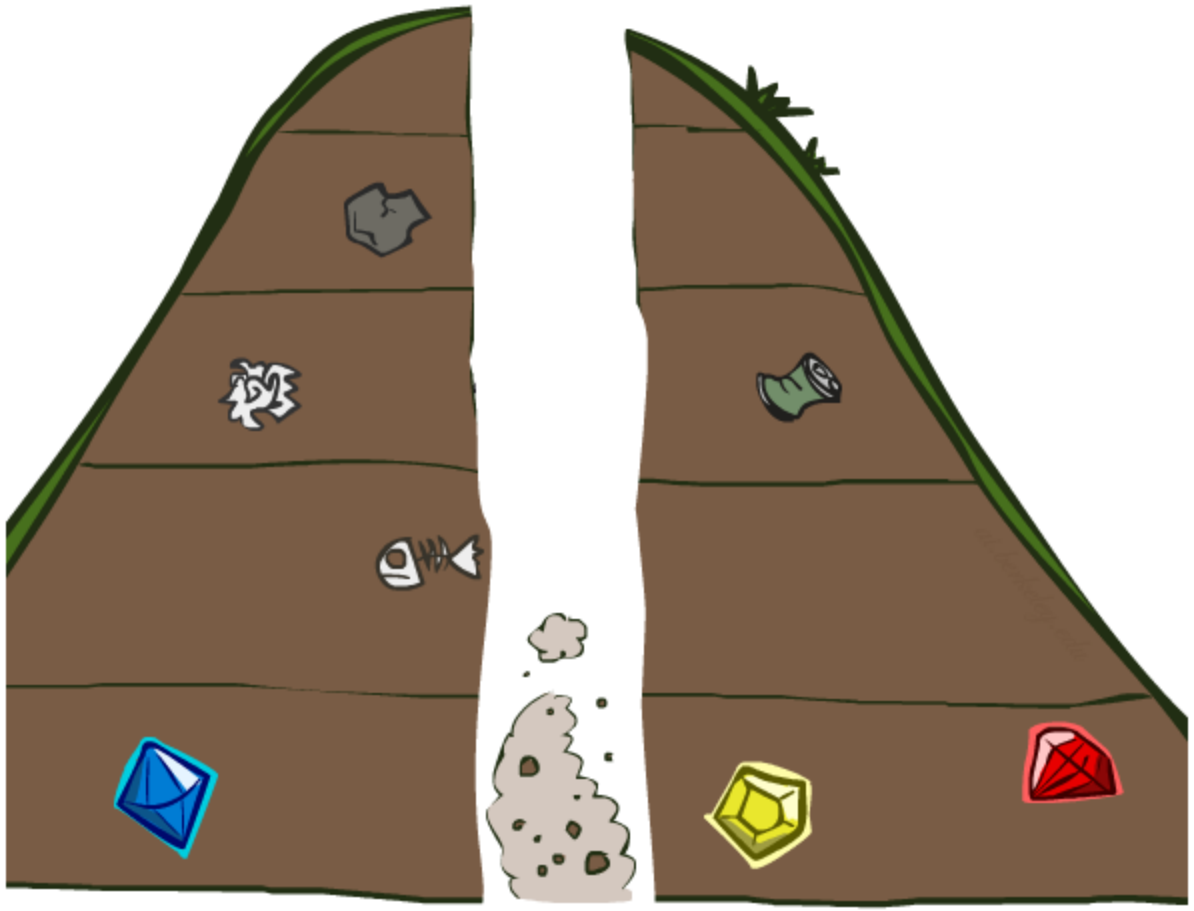
Depth-First Search

Strategy: expand a deepest node first





Search Algorithm Properties

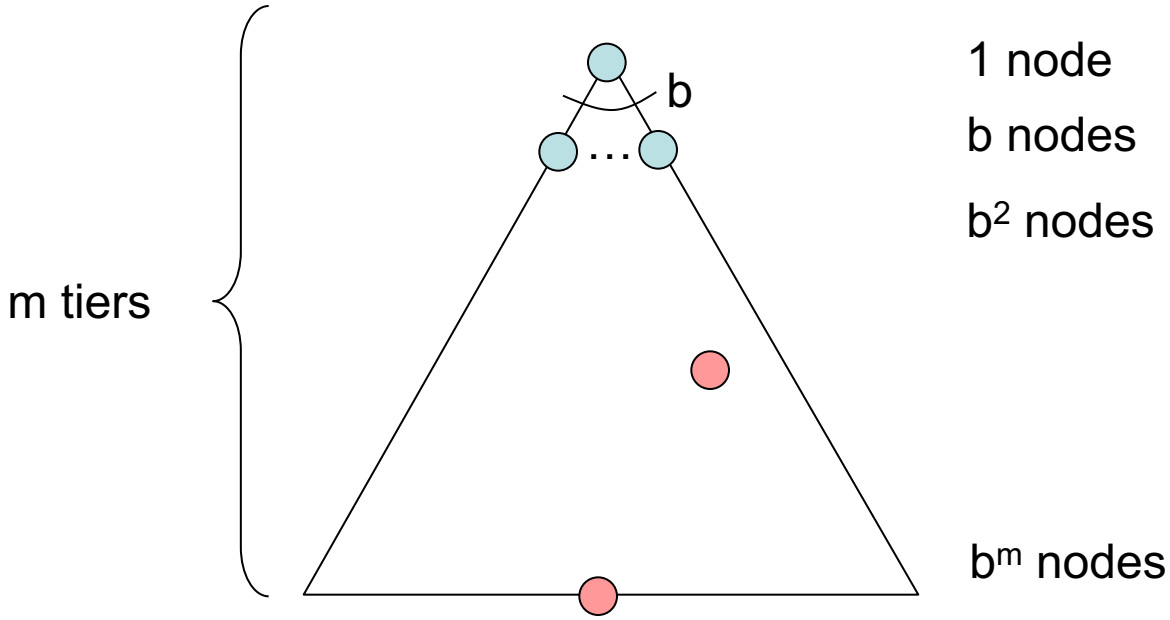


Search Algorithm Properties

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

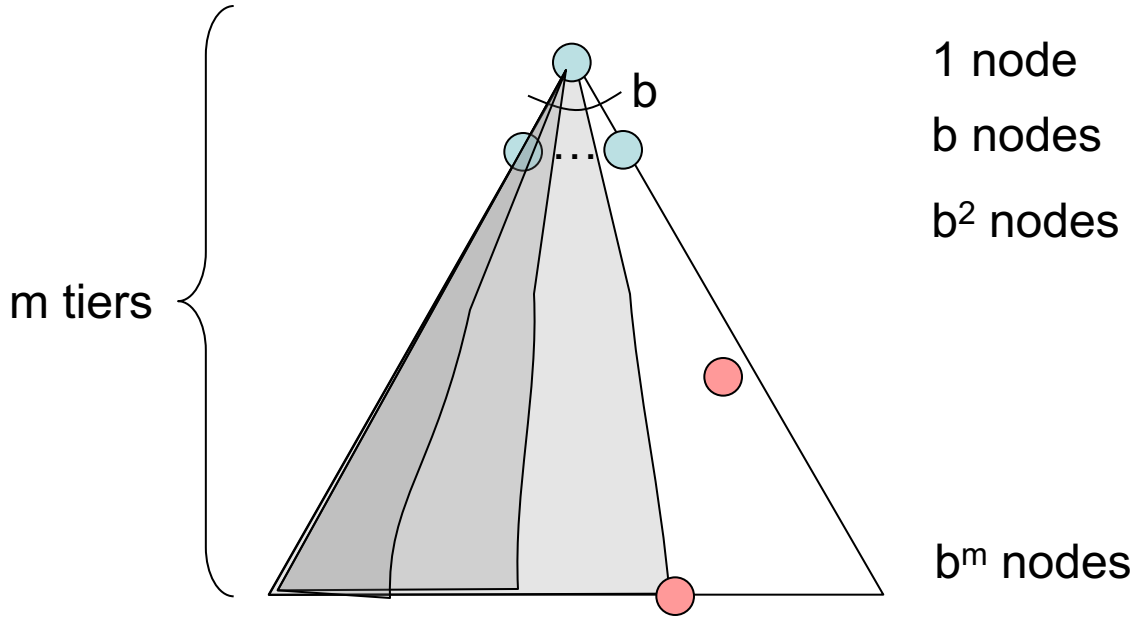
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths

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

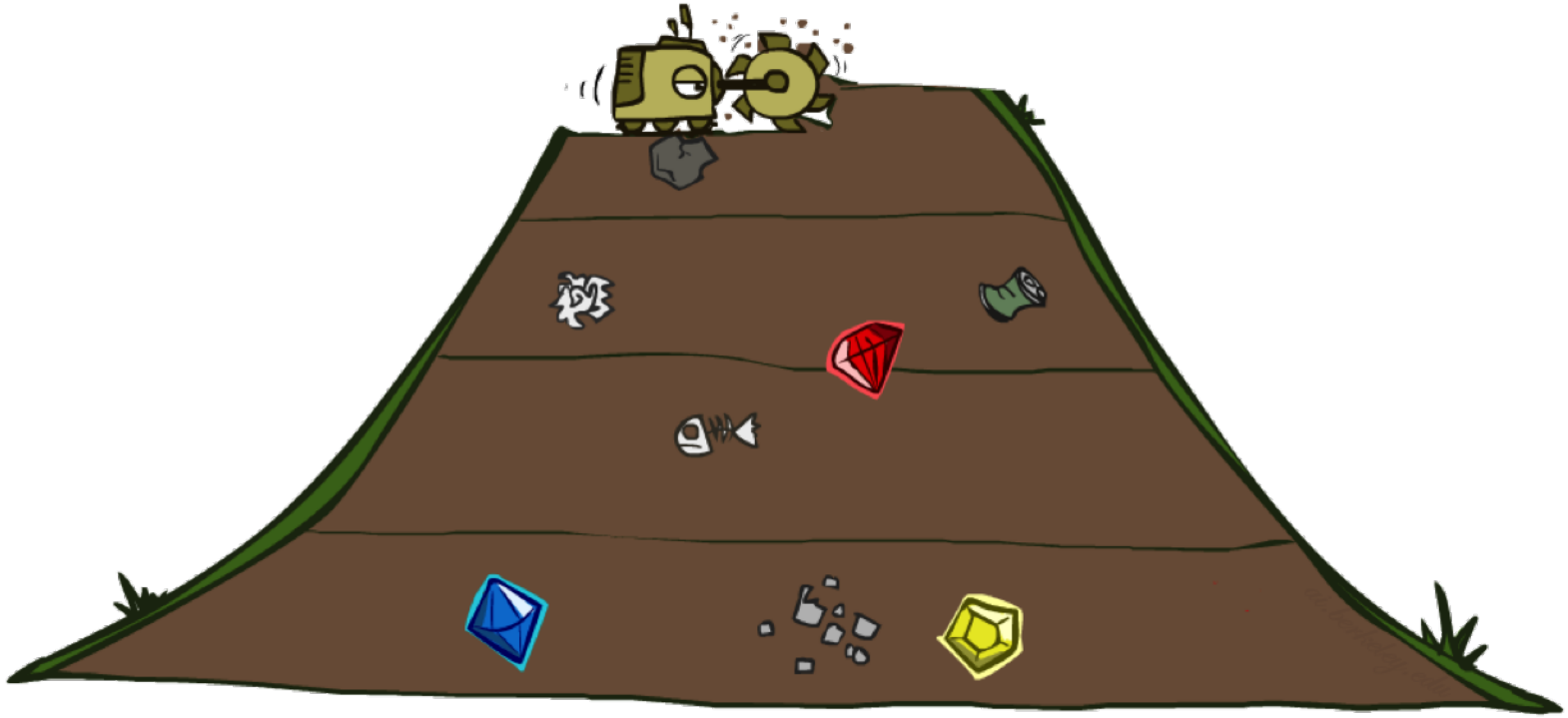


Depth-First Search (DFS) Properties

- 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

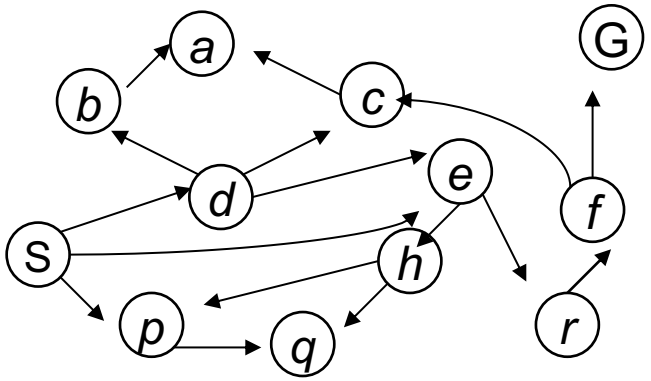


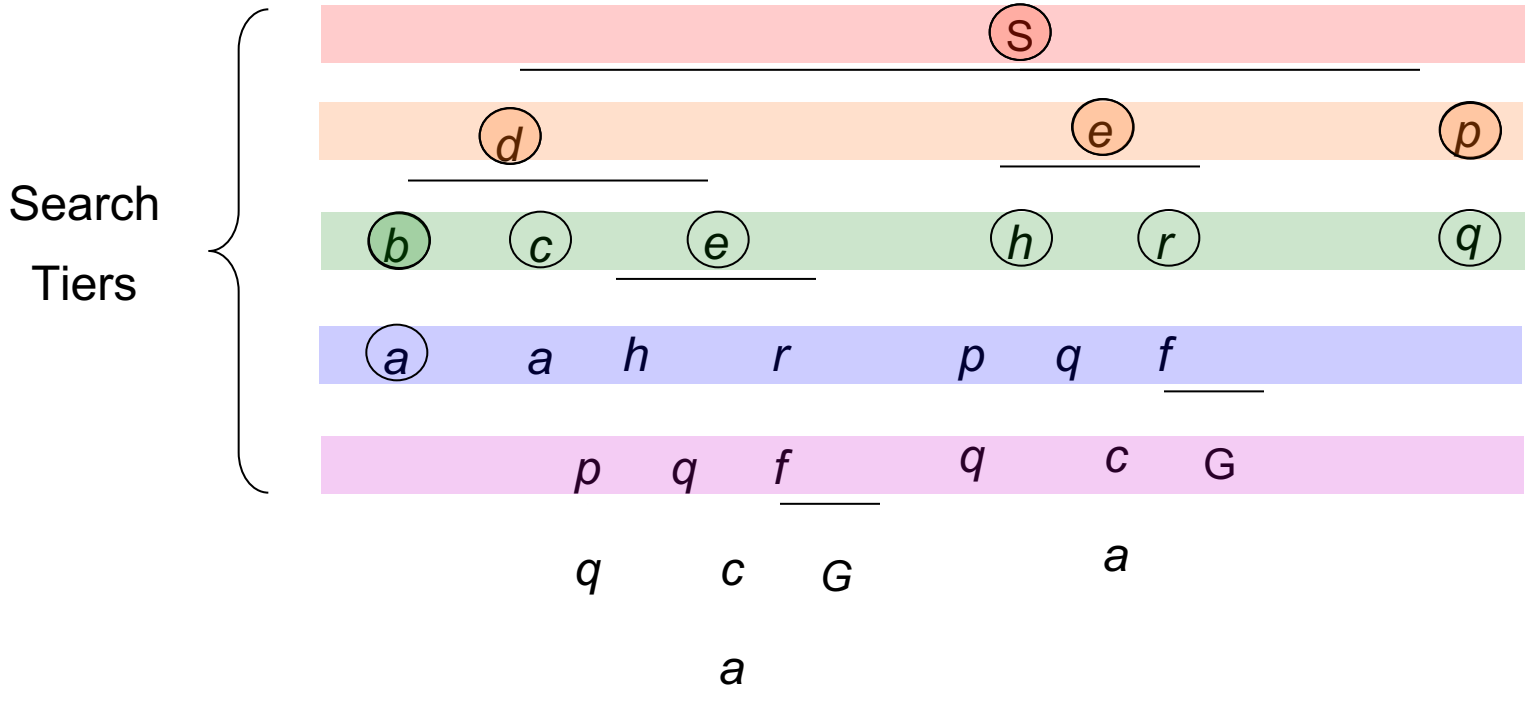
Breadth-First Search



Breadth-First Search

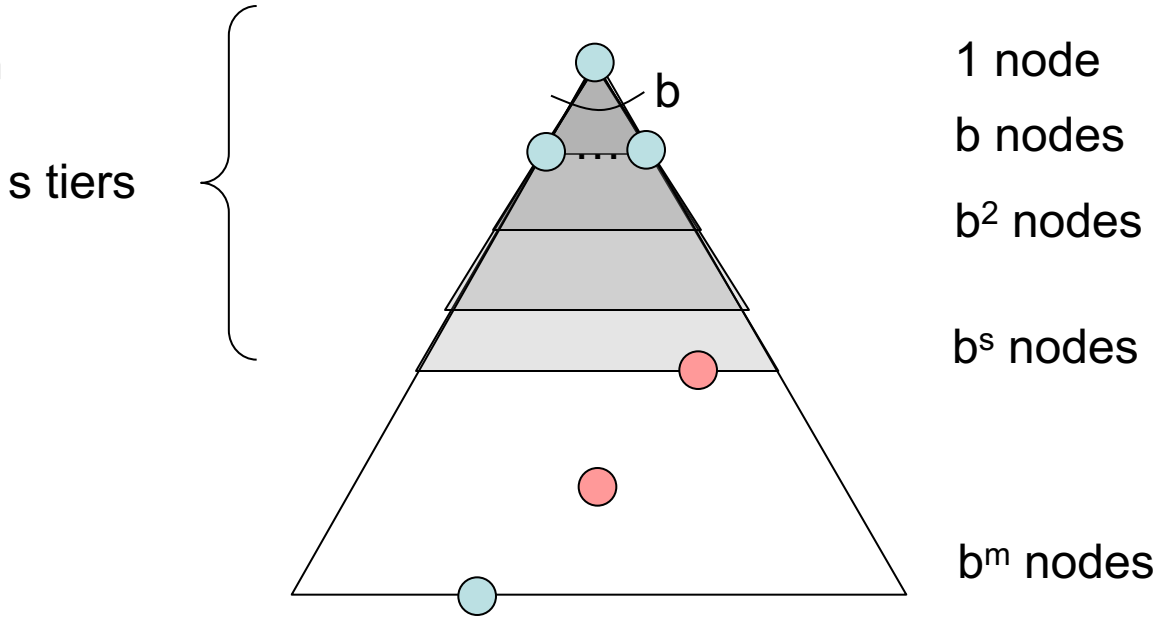
Strategy: expand a shallowest node first



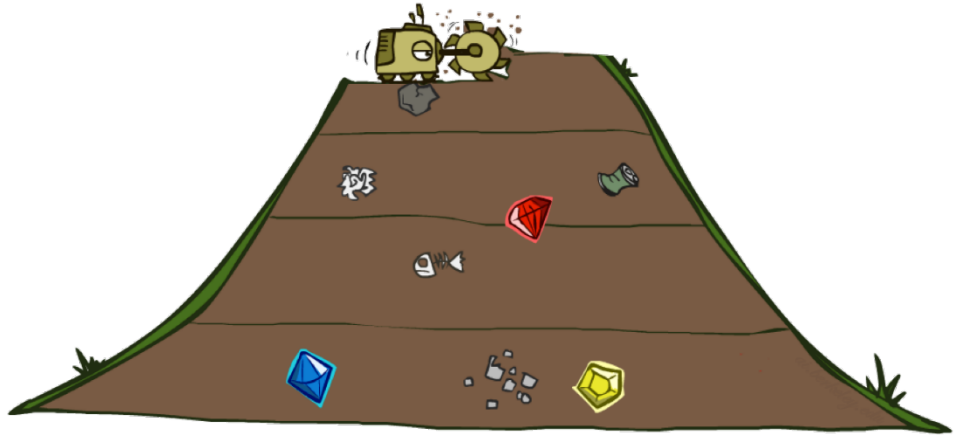


Breadth-First Search (BFS) Properties

- 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



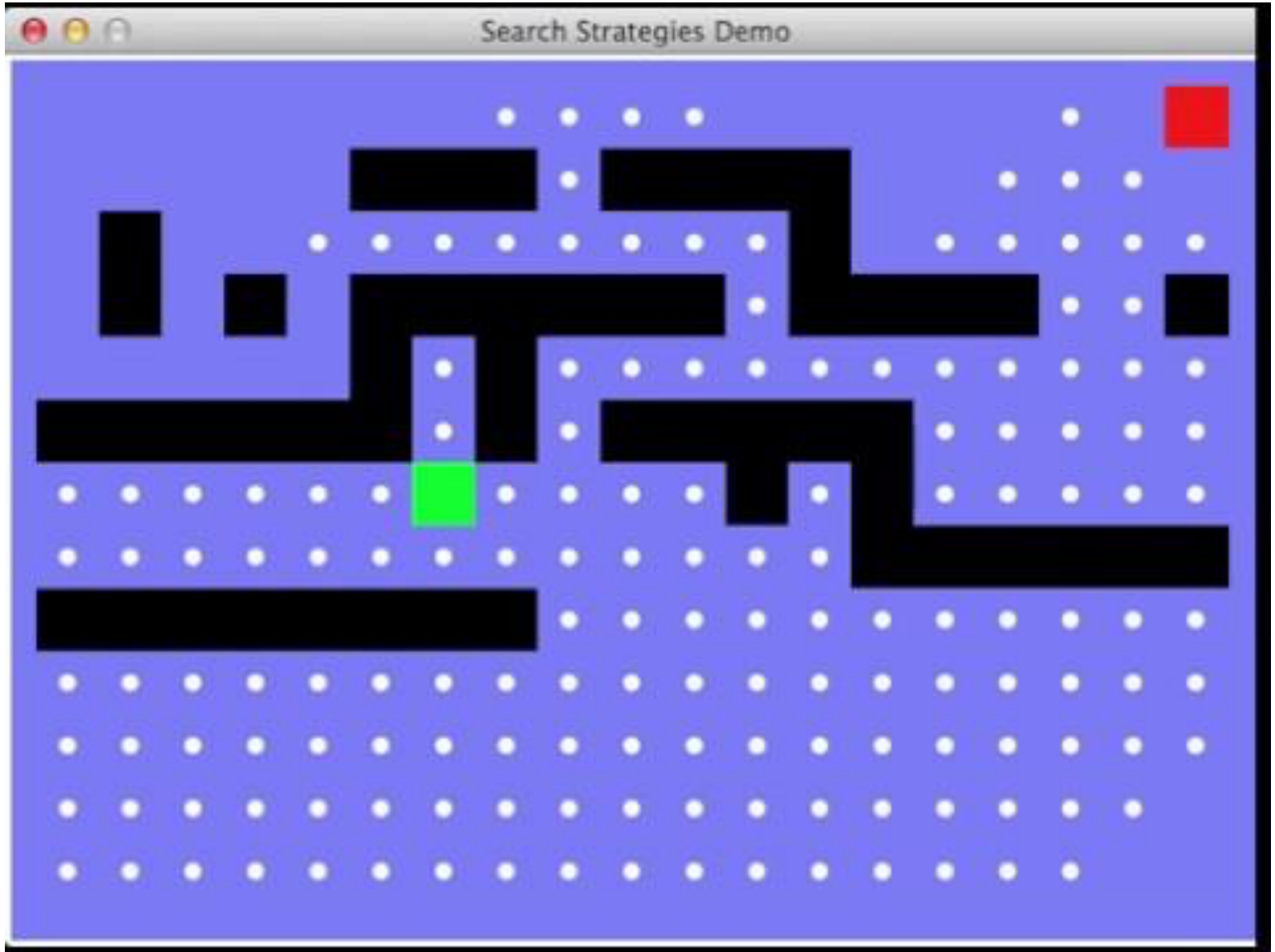
Quiz: DFS vs BFS

- When will BFS outperform DFS?

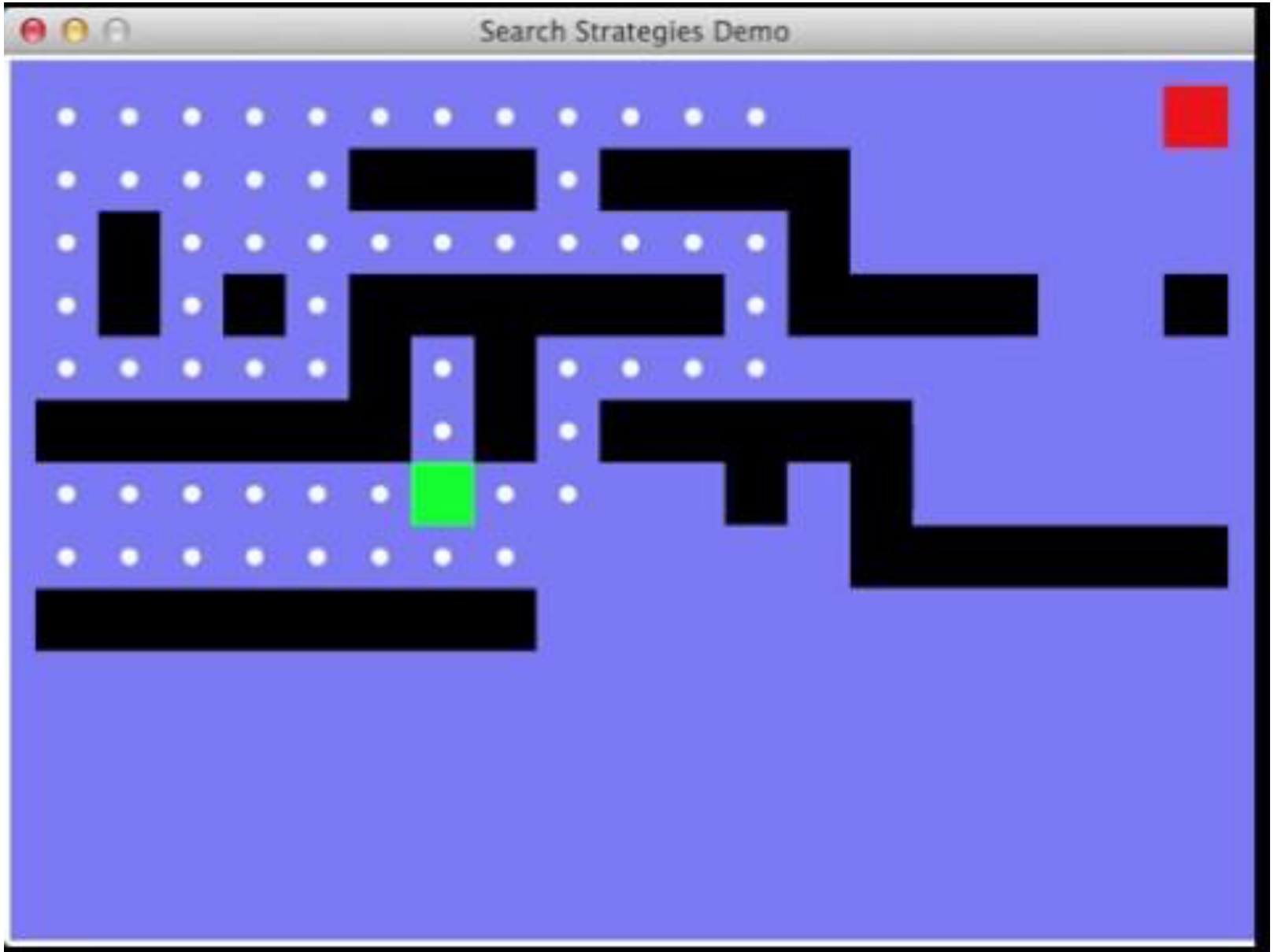
- When will DFS outperform BFS?

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

Video of Demo Maze Water DFS/BFS (part 1)

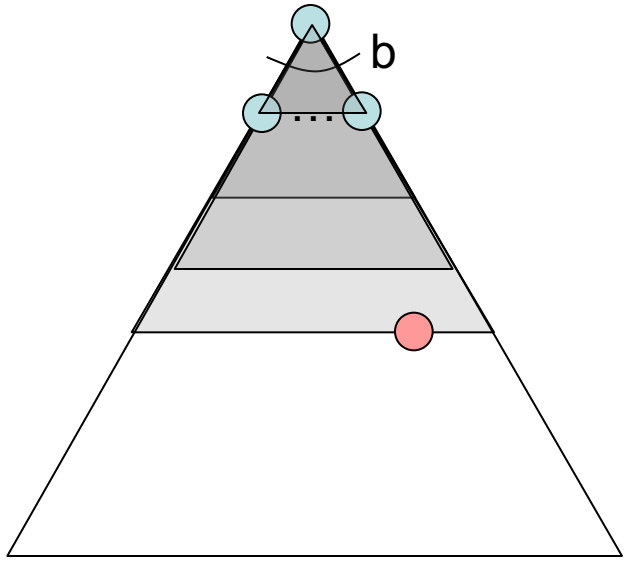


Video of Demo Maze Water DFS/BFS (part 2)

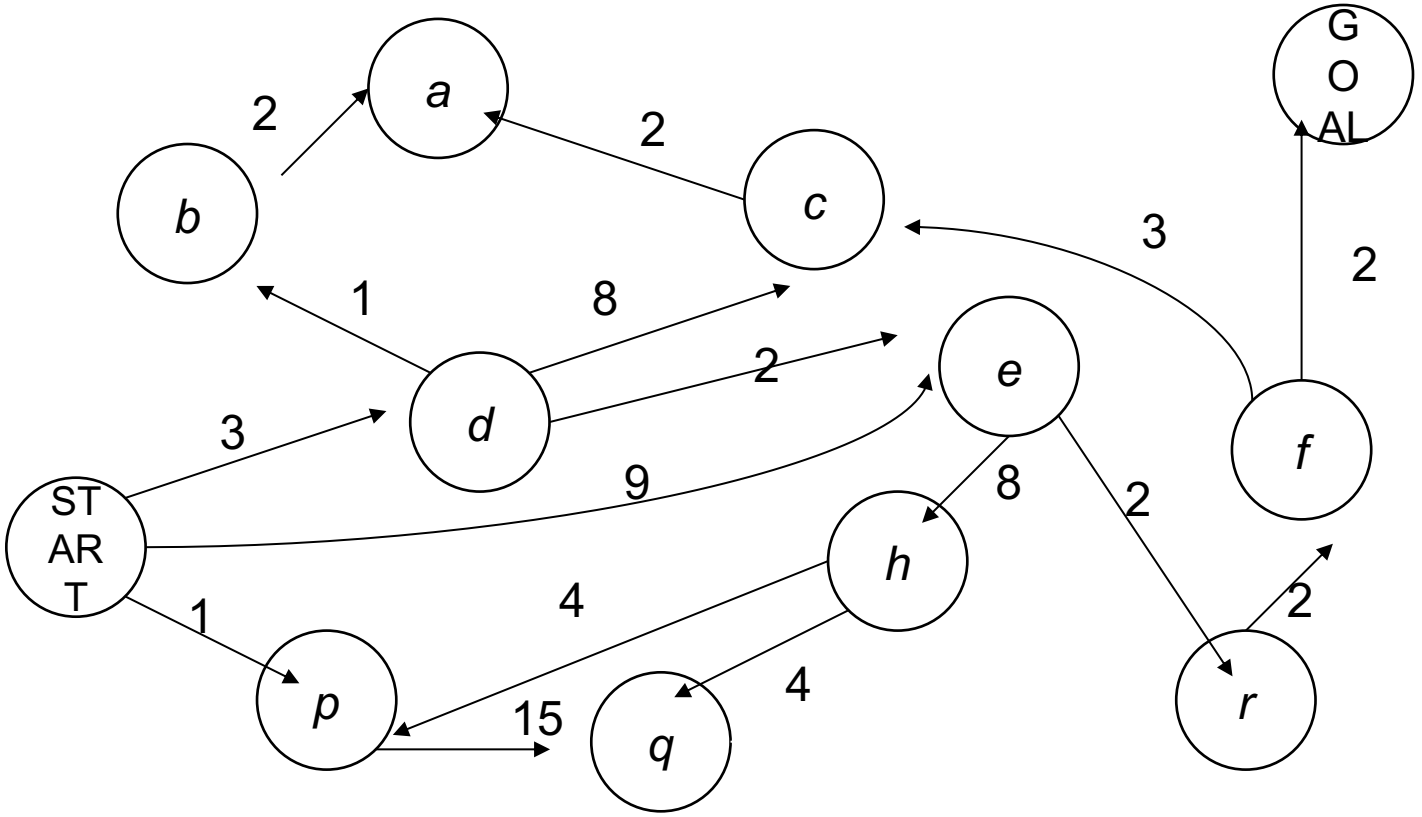


Iterative Deepening

- 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!

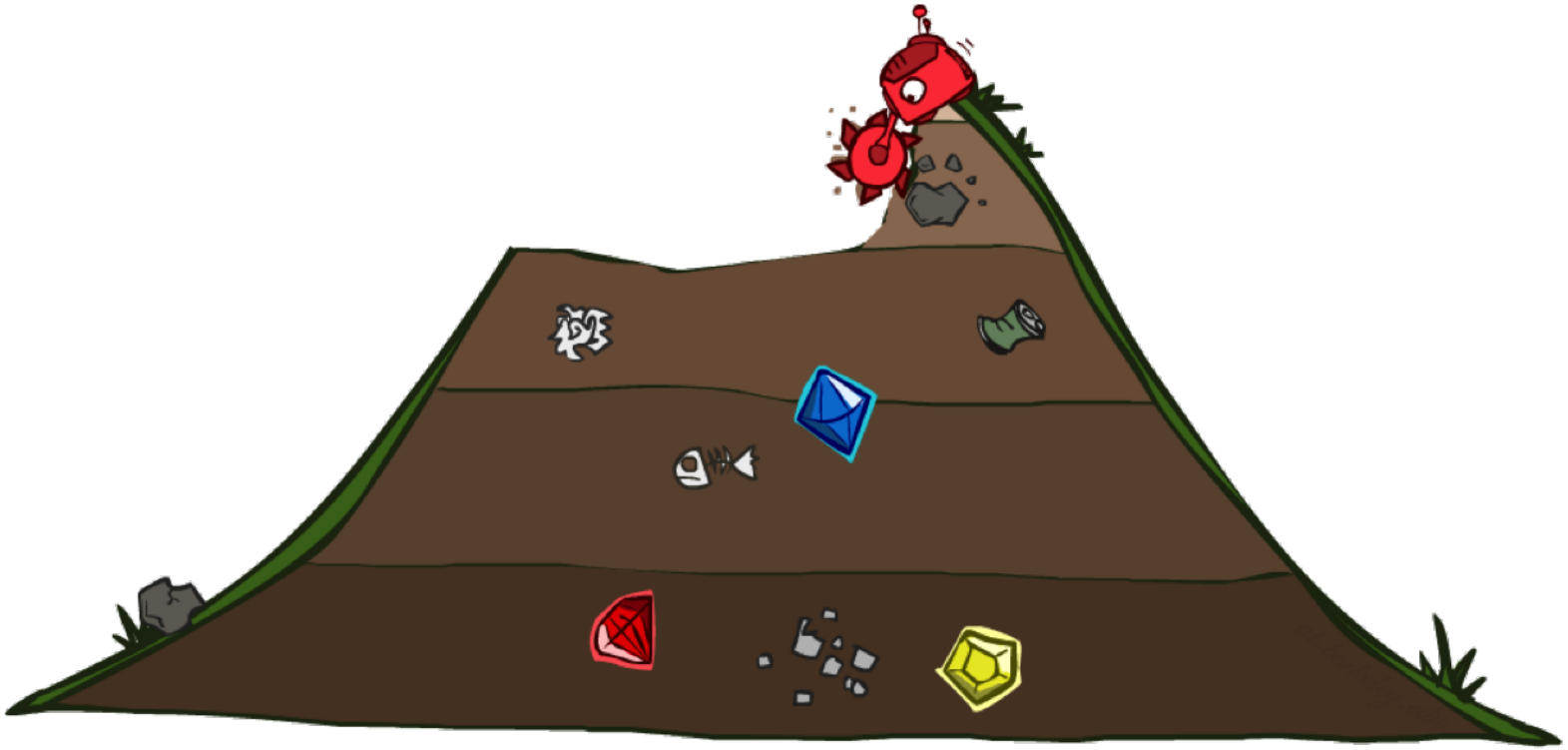


Cost-Sensitive Search



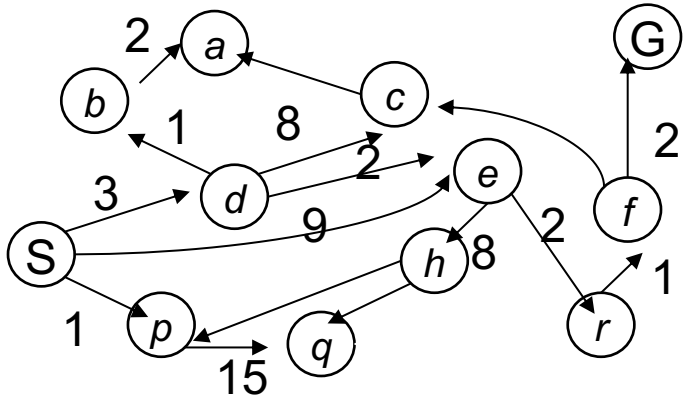
BFS finds the shortest path in terms of number of actions.
It does not find the least-cost path. We will now cover
a similar algorithm which does find the least-cost path.

Uniform Cost Search (Dijkstra's algorithm)

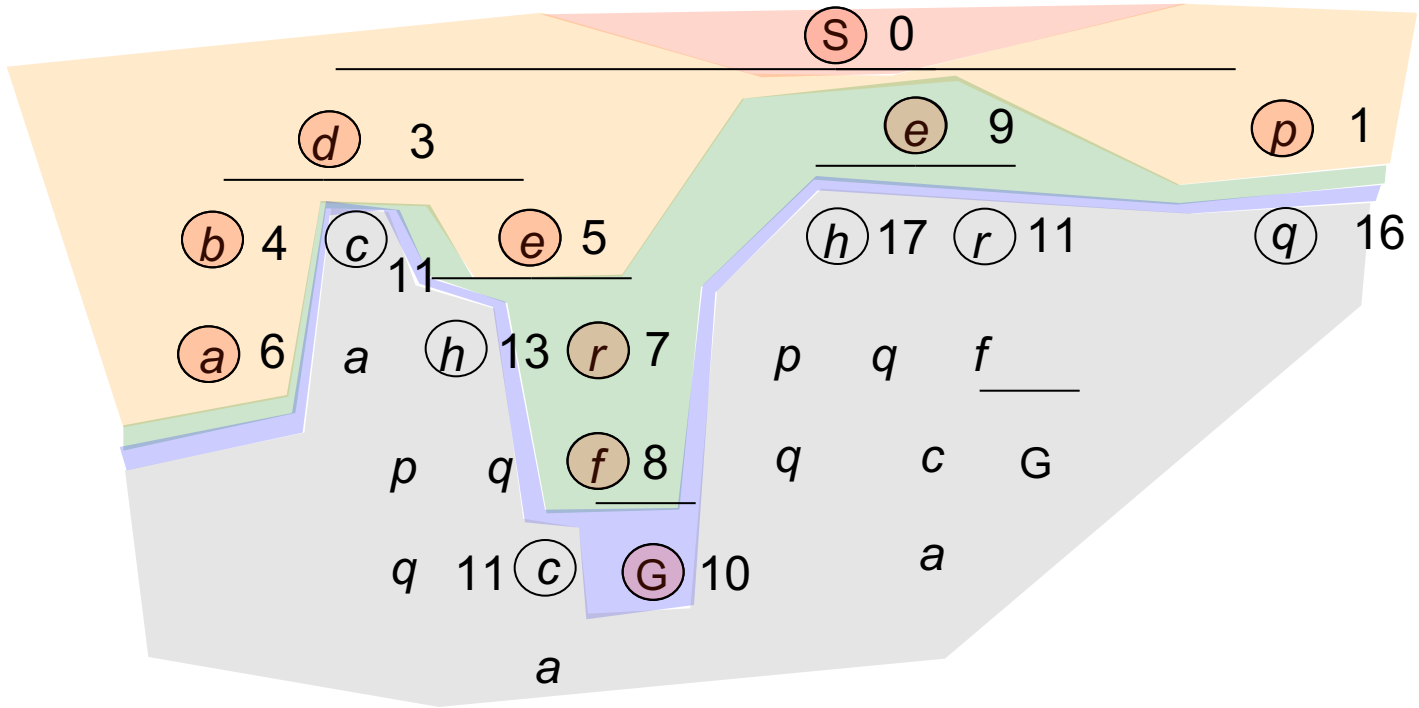


Uniform Cost Search

Strategy: expand a cheapest node first:

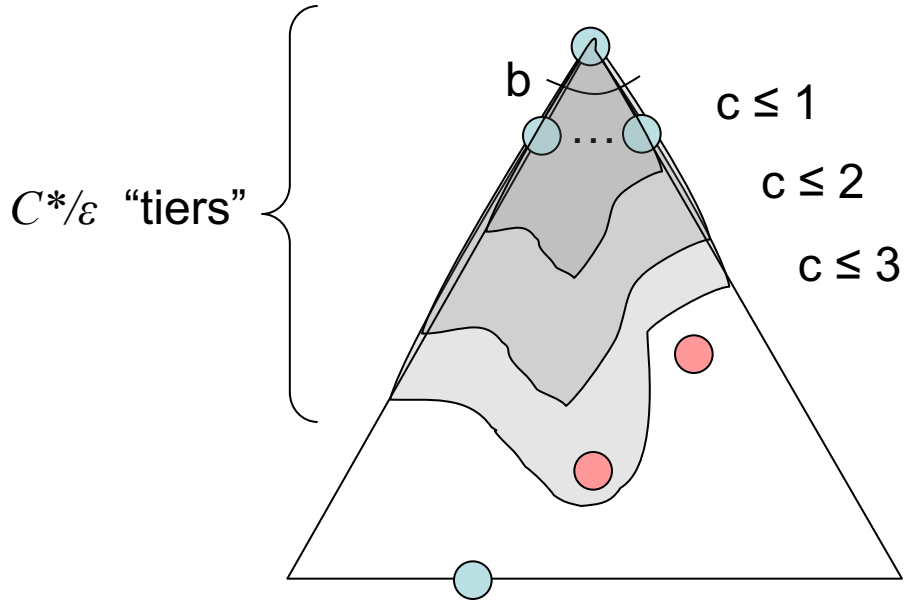


Cost contours



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*)

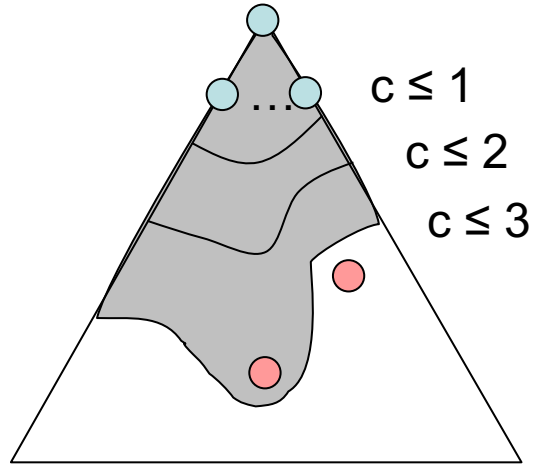
Uniform Cost Issues

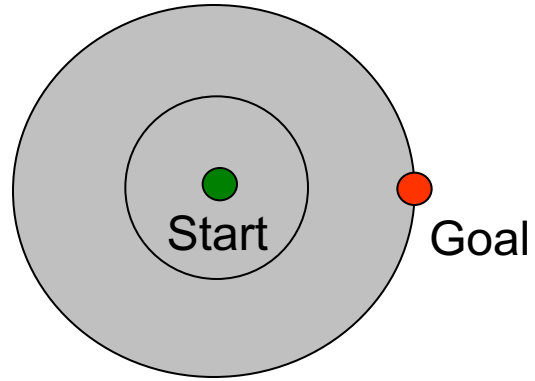
- Remember: UCS explores increasing cost contours

- The good: UCS is complete and optimal!

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

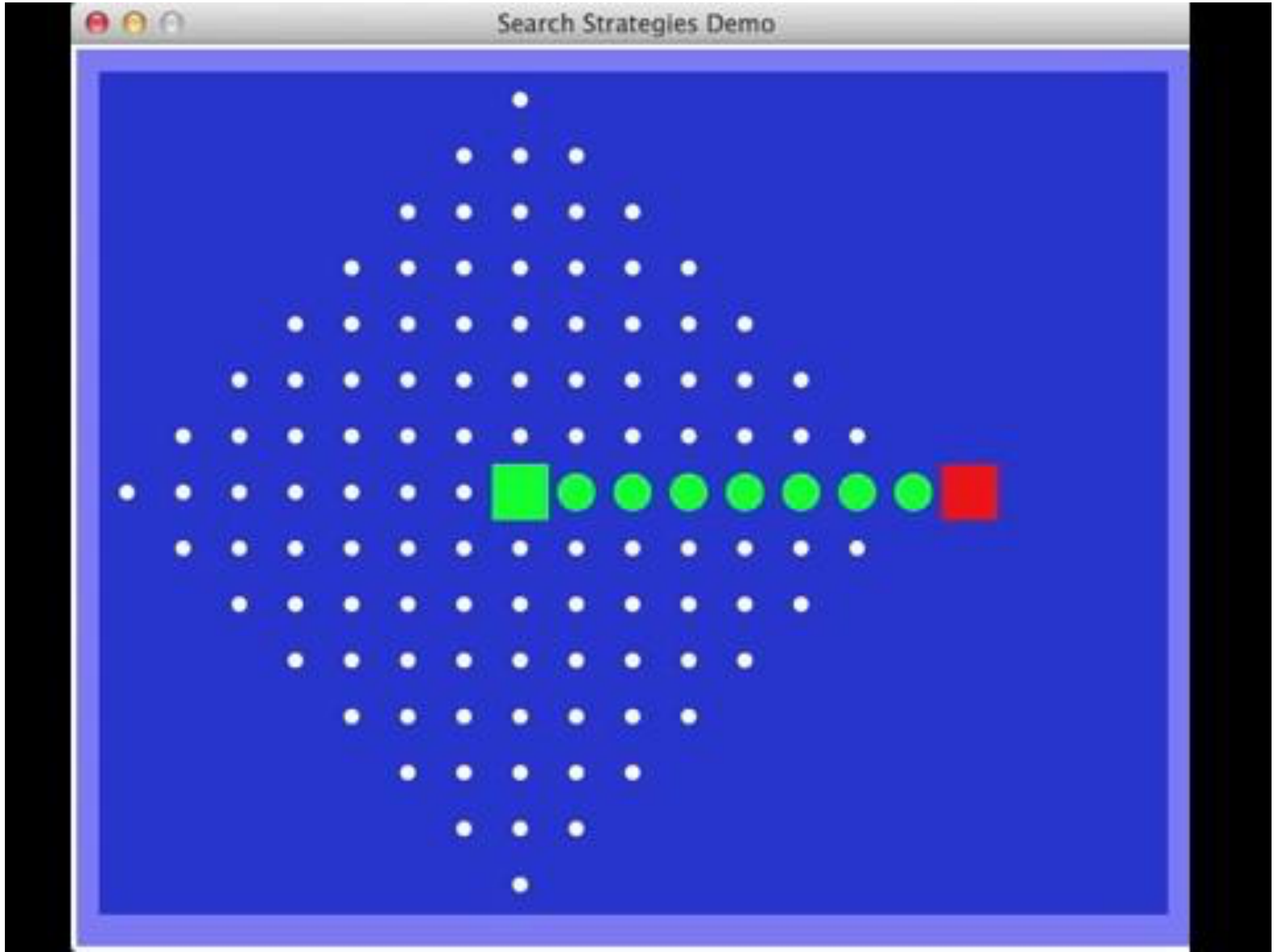
- We’ll fix that soon!



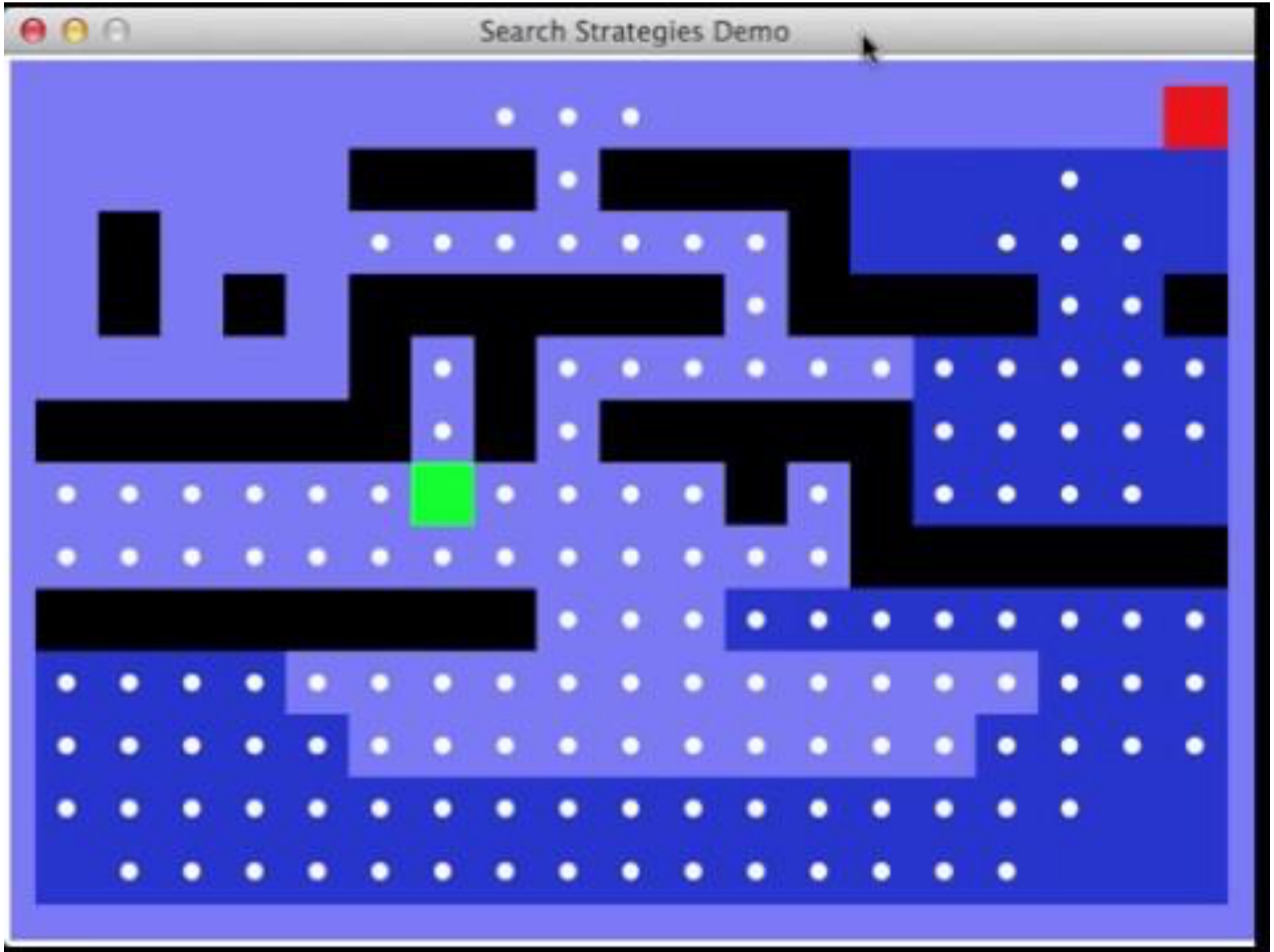


[Demo: empty grid UCS (L2D5)]
[Demo: maze with deep/shallow water DFS/BFS/UCS (L2D7)]

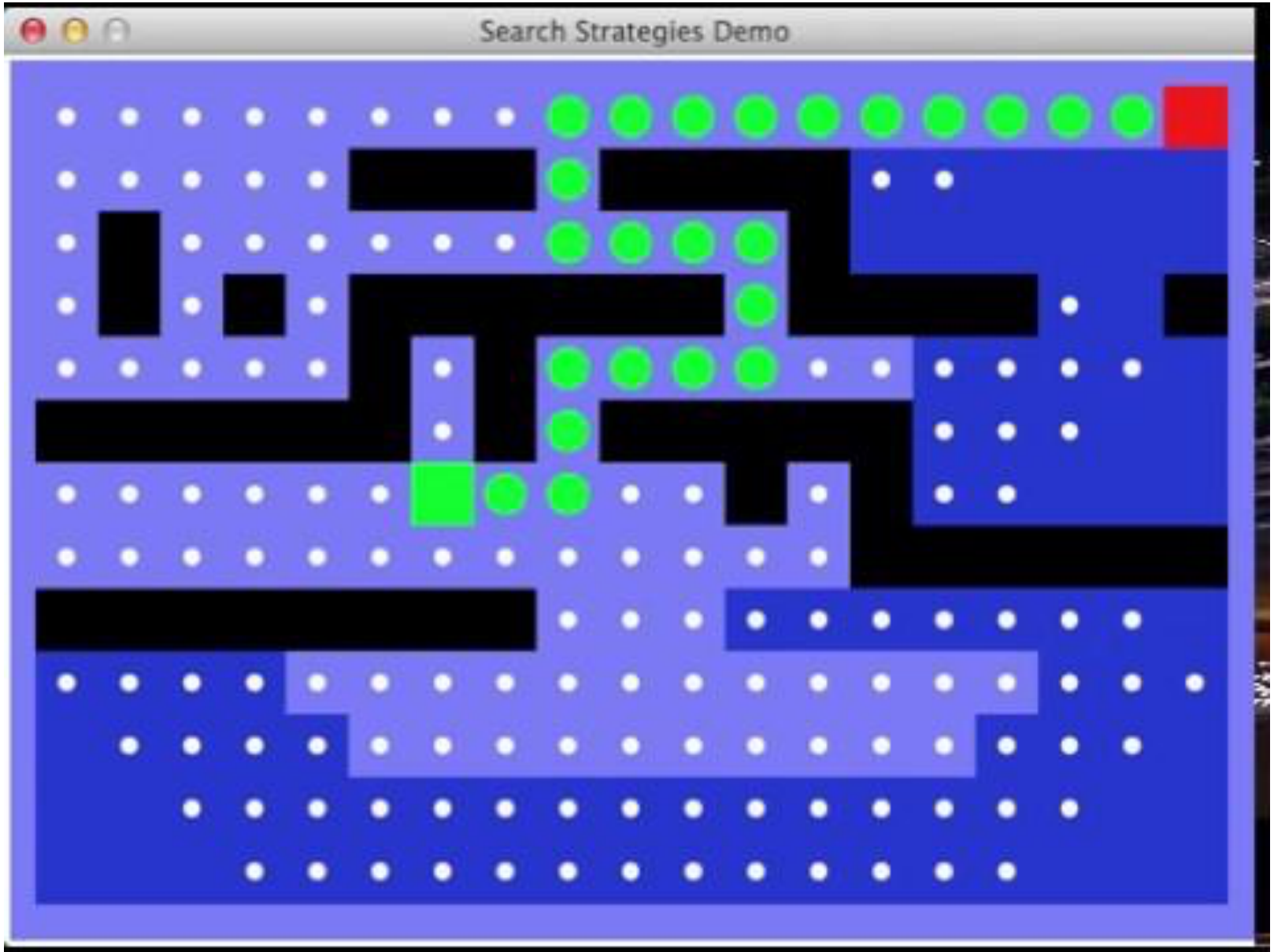
Video of Demo Empty UCS



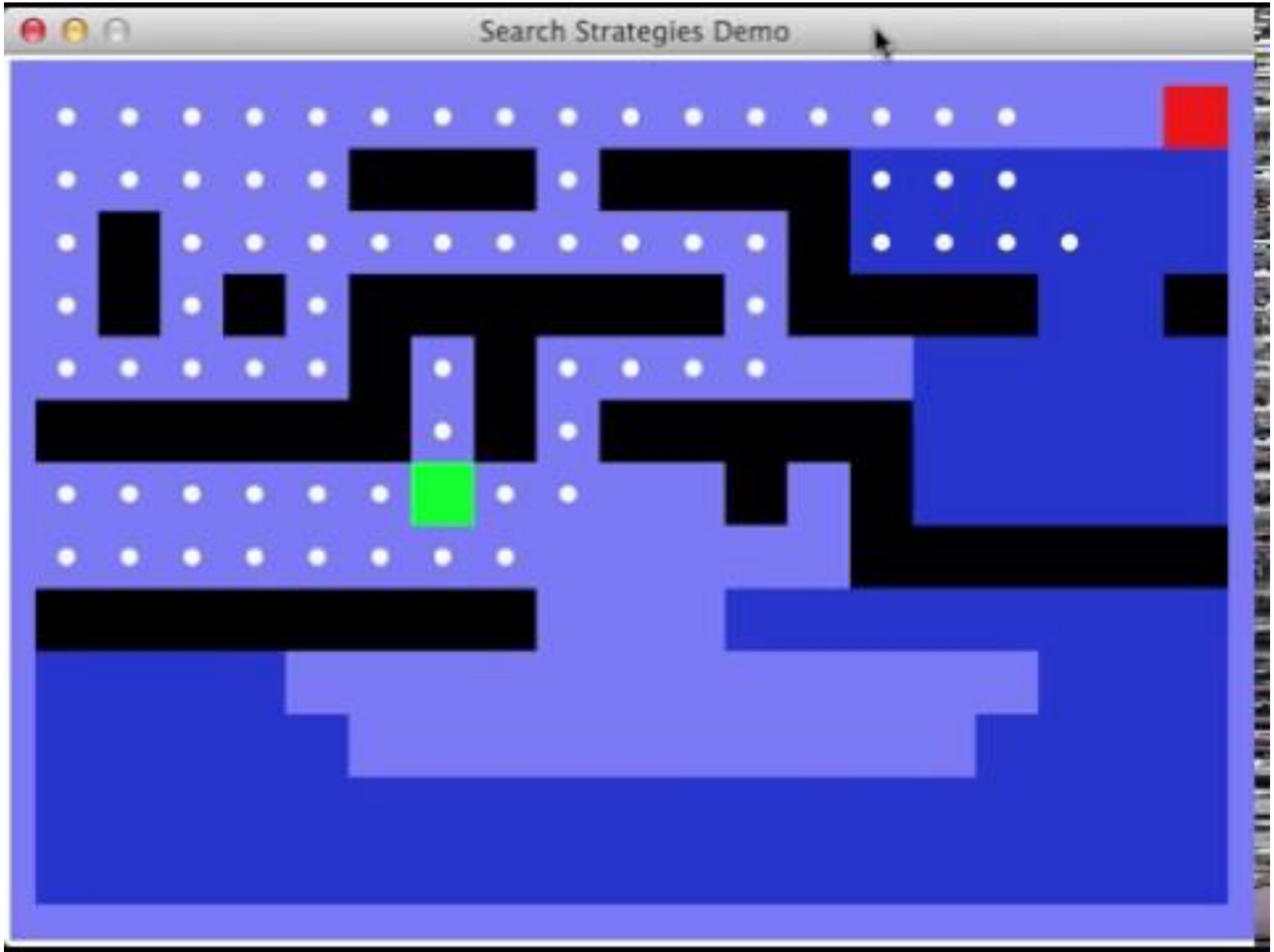
Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 1)



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

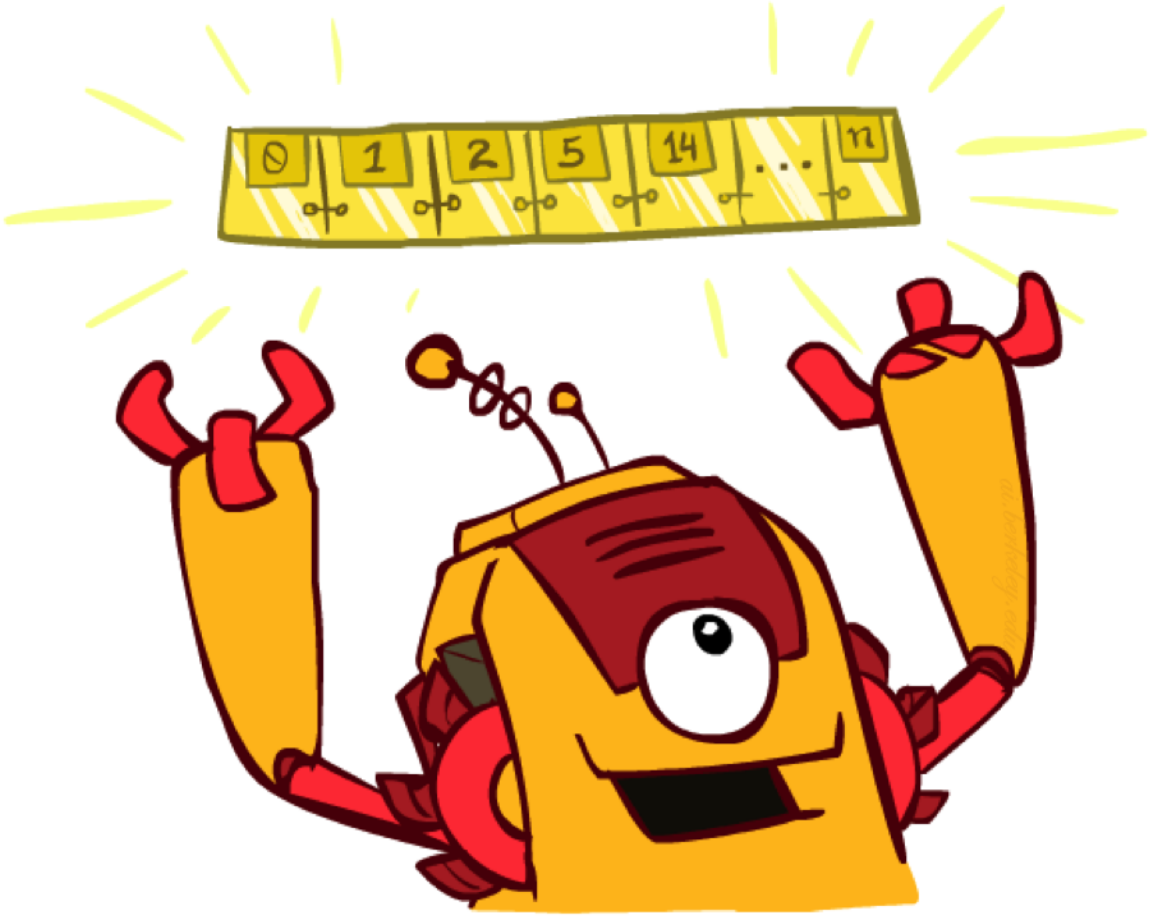


Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 3)



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
  - Can even code one implementation that takes a variable queuing object



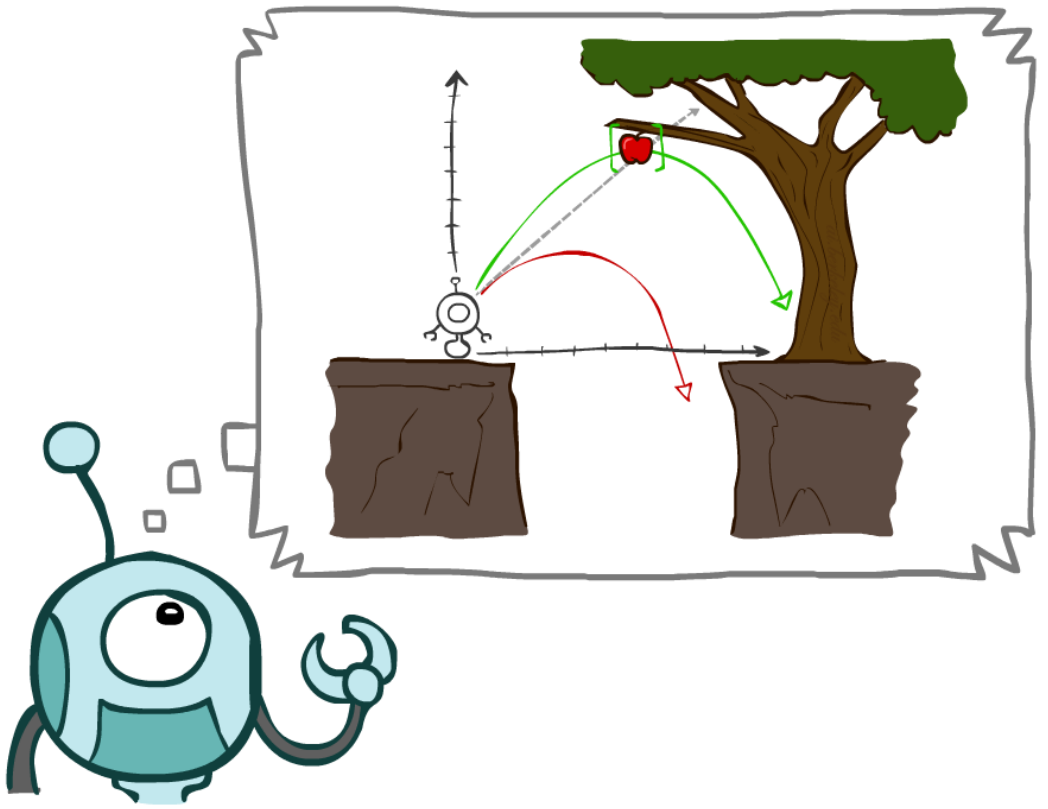
The One Queue

```
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
```

- DFS
  - Stack (LIFO)
- BFS
  - Queue (FIFO)
- UCS / Dijkstra's
  - min-Priority Queue
  - where priority is the cumulative cost from the start node

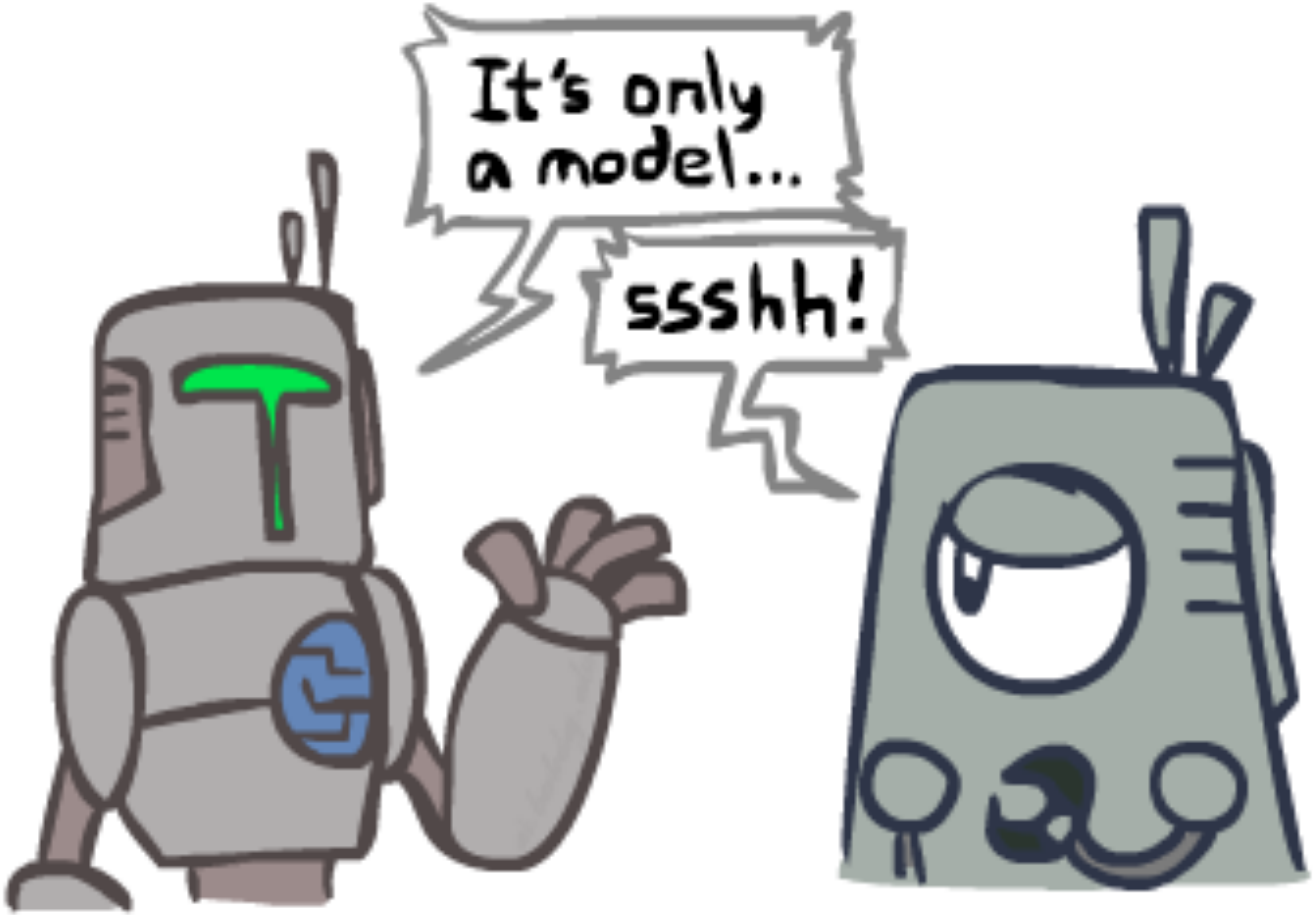
Recap

- Agents that Plan Ahead
  - Reflex agents vs planning agents
- Search Problems
  - State space representation
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search



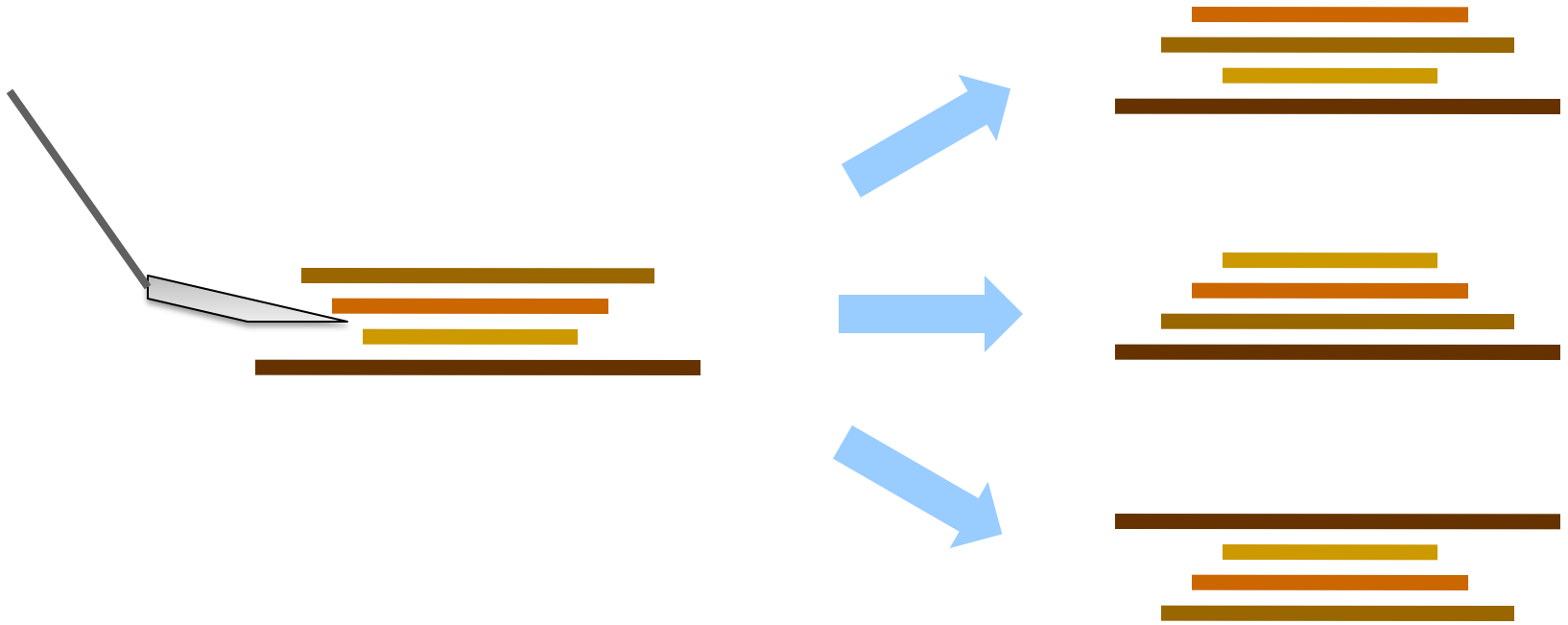
Search and Models

- 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?

Example: Pancake Problem



Cost: Number of pancakes flipped

Example: Pancake Problem

BOUNDS FOR SORTING BY PREFIX REVERSAL

William H. GATES

Microsoft, Albuquerque, New Mexico

Christos H. PAPADIMITRIOU*†

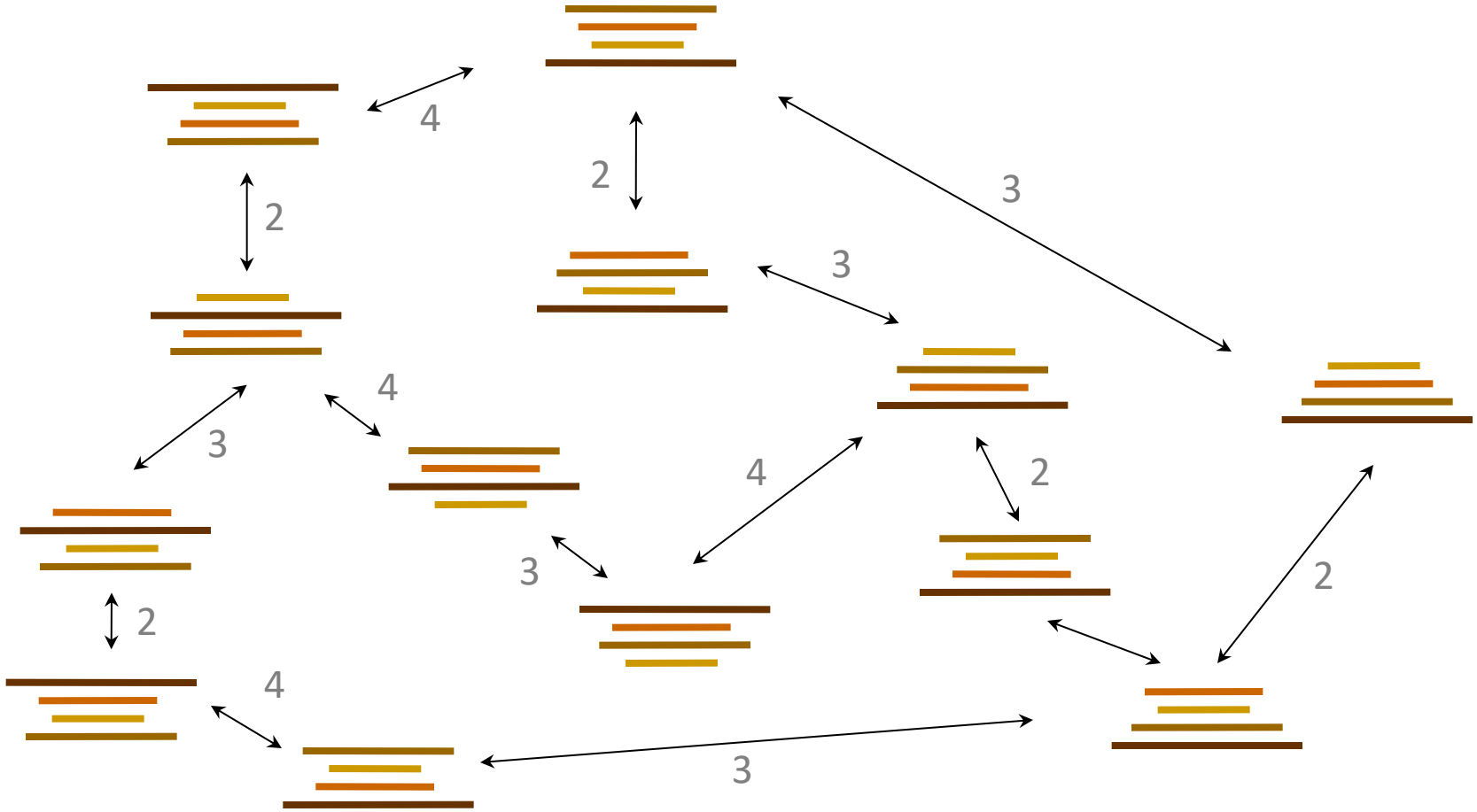
Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.

Received 18 January 1978
Revised 28 August 1978

For a permutation σ of the integers from 1 to n, let $f(\sigma)$ be the smallest number of prefix reversals that will transform σ to the identity permutation, and let $f(n)$ be the largest such $f(\sigma)$ for all σ in (the symmetric group) S_n. We show that $f(n) \leqslant (5n+5)/3$, and that $f(n) \geqslant 17n/16$ for n a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function $g(n)$ is shown to obey $3n/2 - 1 \leqslant g(n) \leqslant 2n+3$.

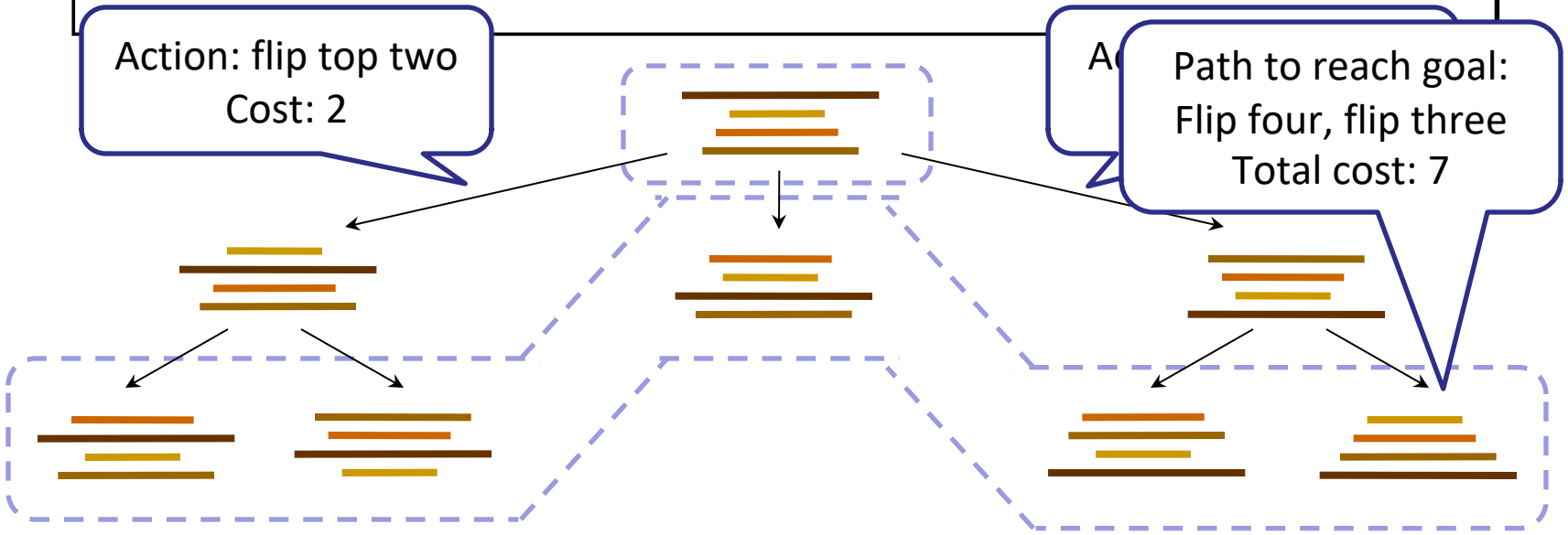
Example: Pancake Problem

State space graph with costs as weights



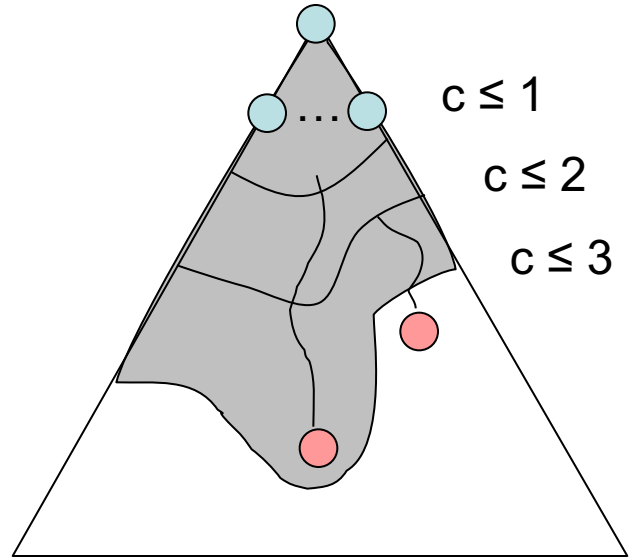
General Tree Search

```
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
```



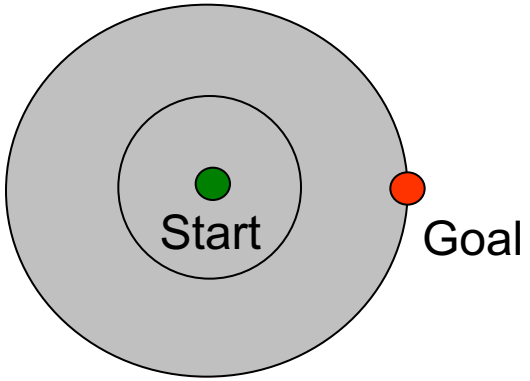
Uniform Cost Search

- Strategy: expand lowest path cost



- The good: UCS is complete and optimal!

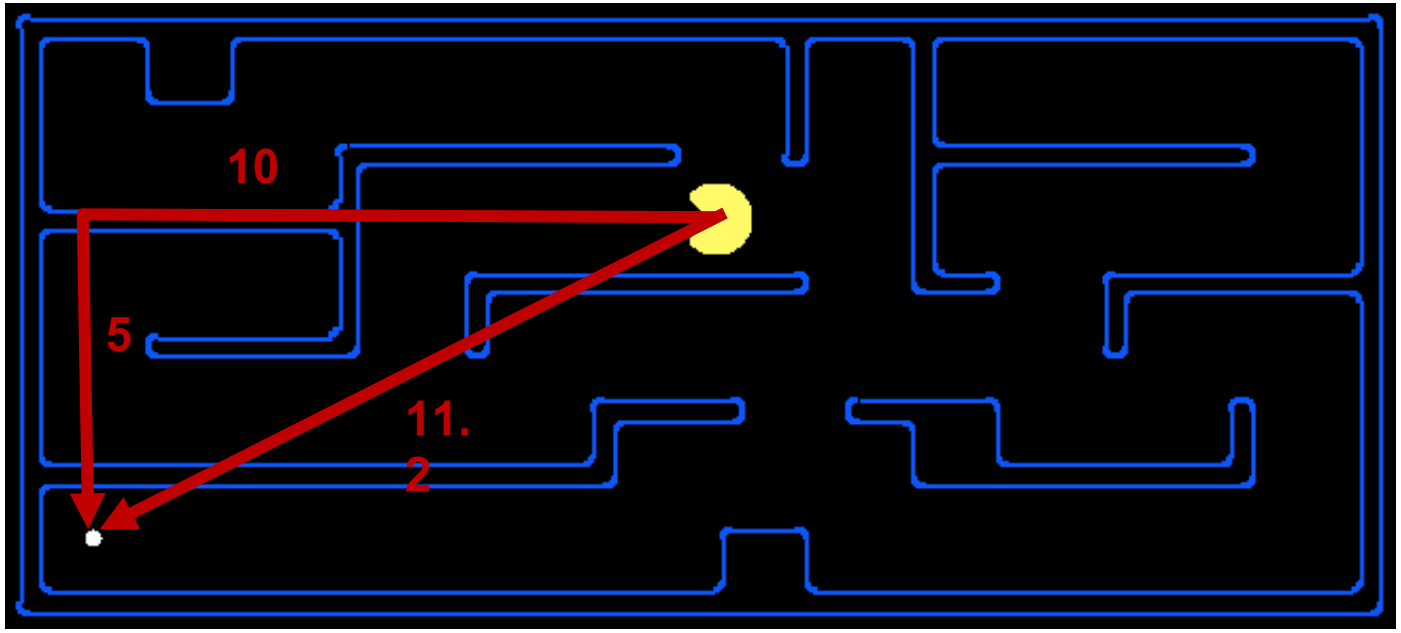
- The bad:
  - Explores options in every “direction”
  - No information about goal location

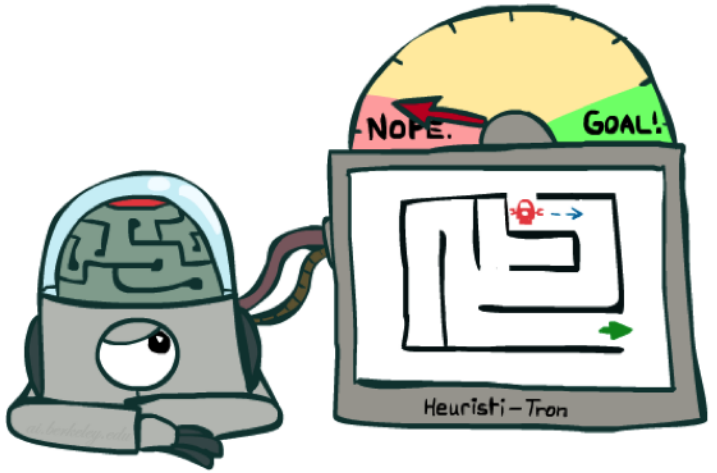


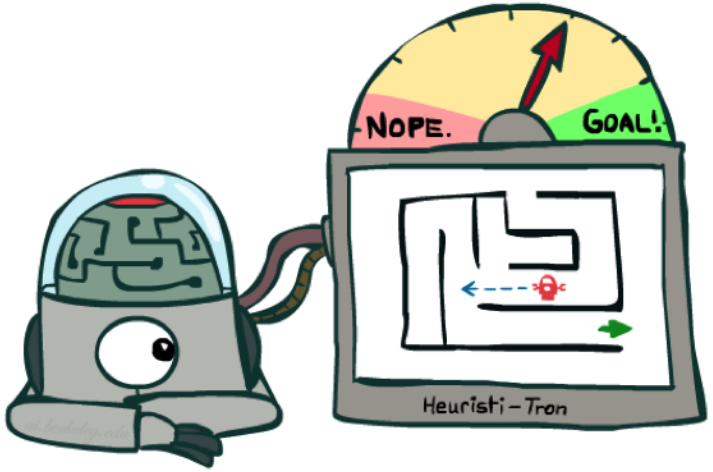
Informed Search

Search Heuristics

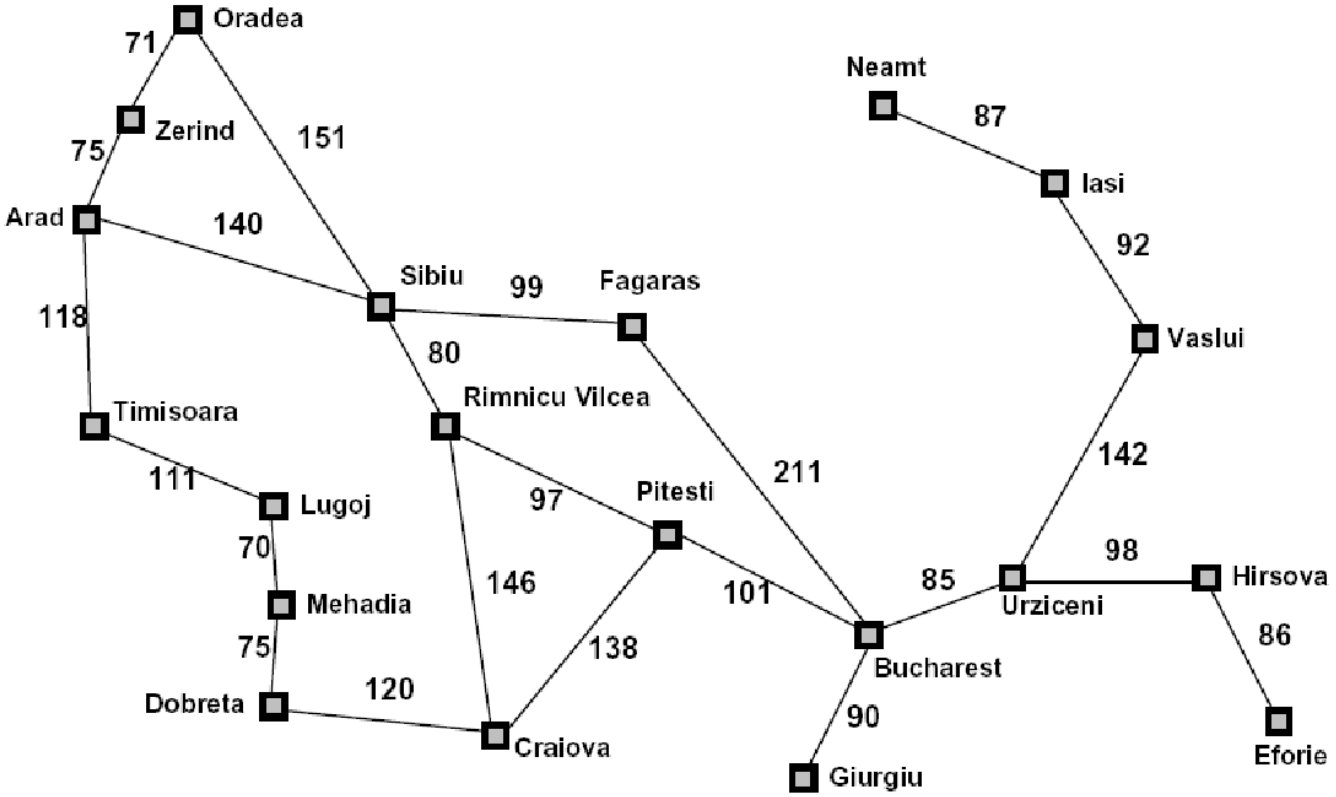
- A heuristic is:
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing







Example: Heuristic Function

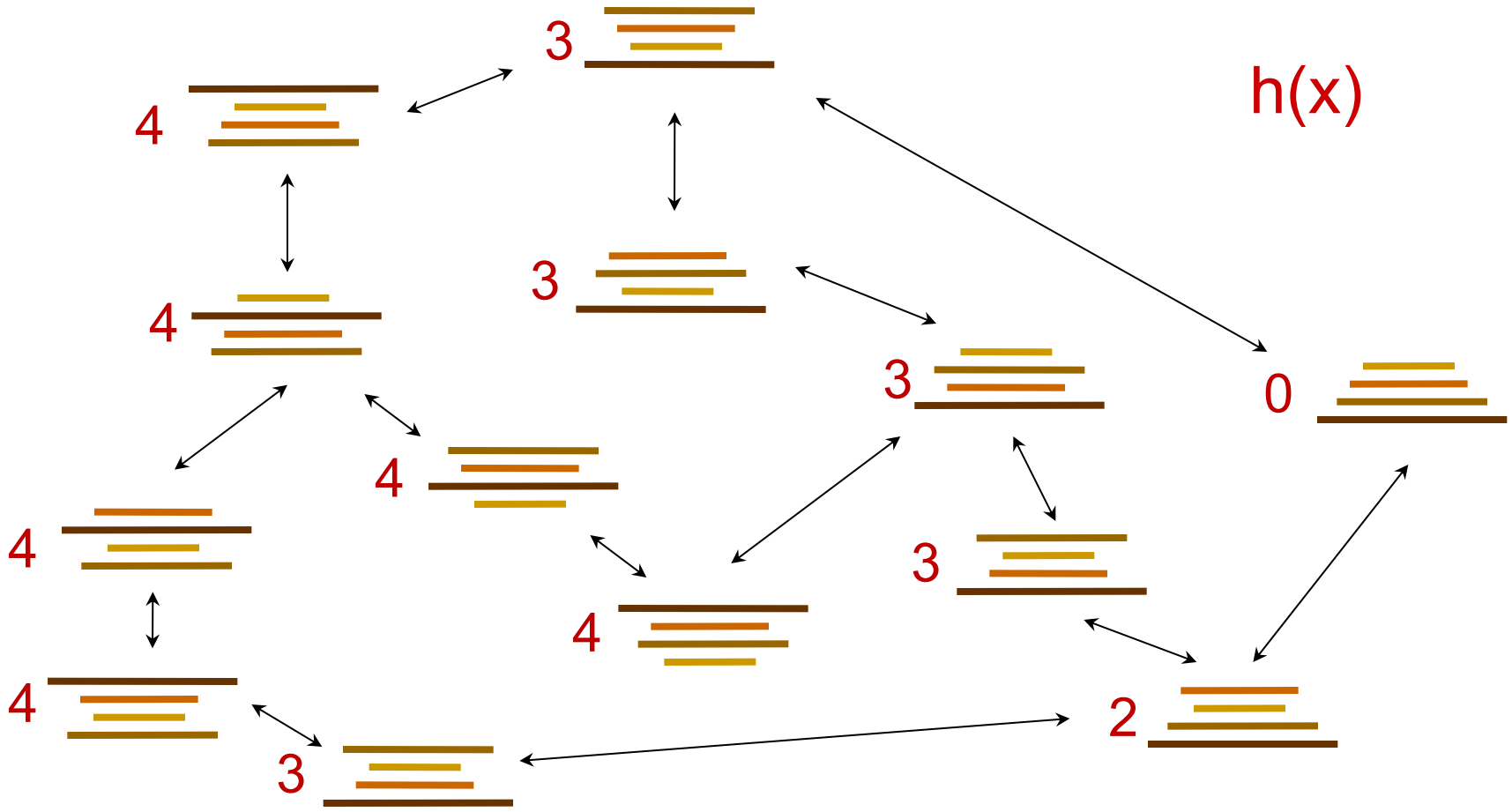


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

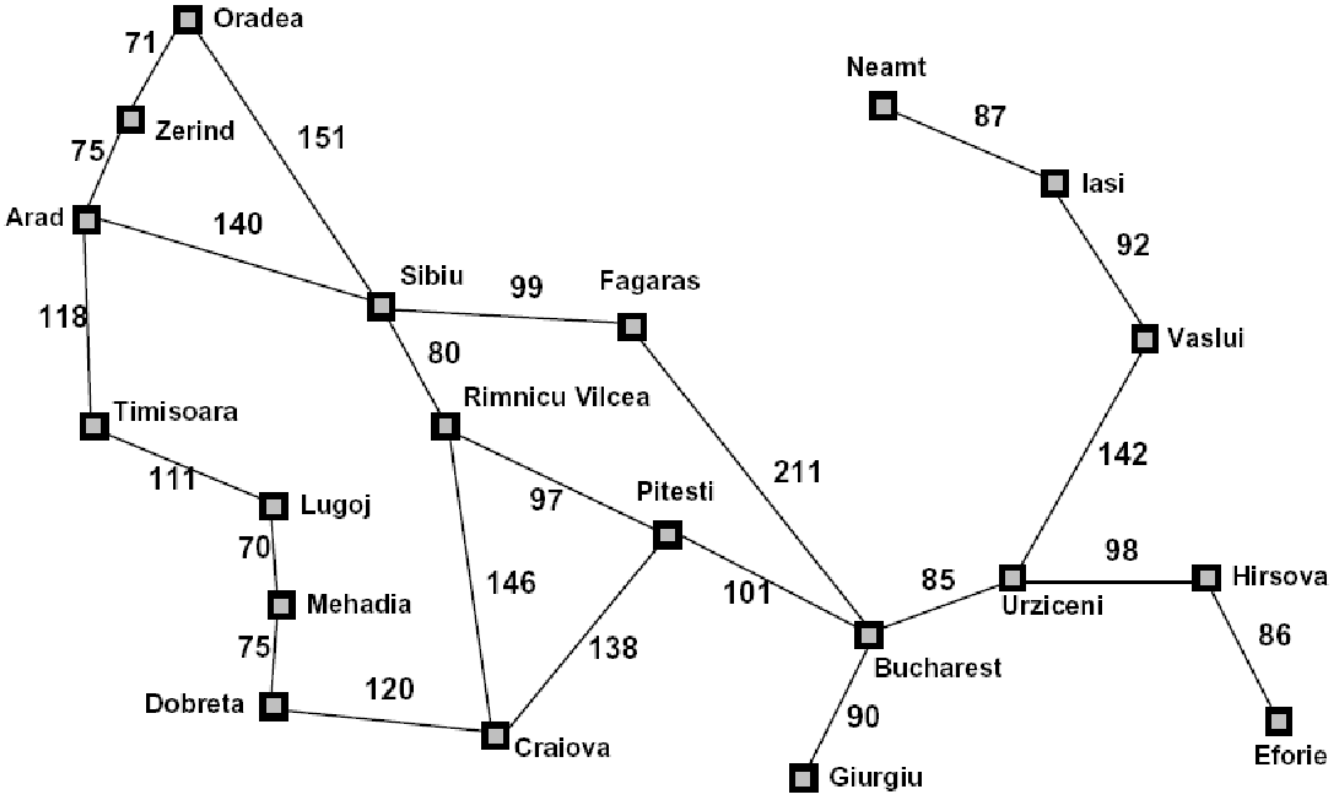
Example: Heuristic Function

Heuristic: the number of the largest pancake that is still out of place



Greedy Search

Example: Heuristic Function

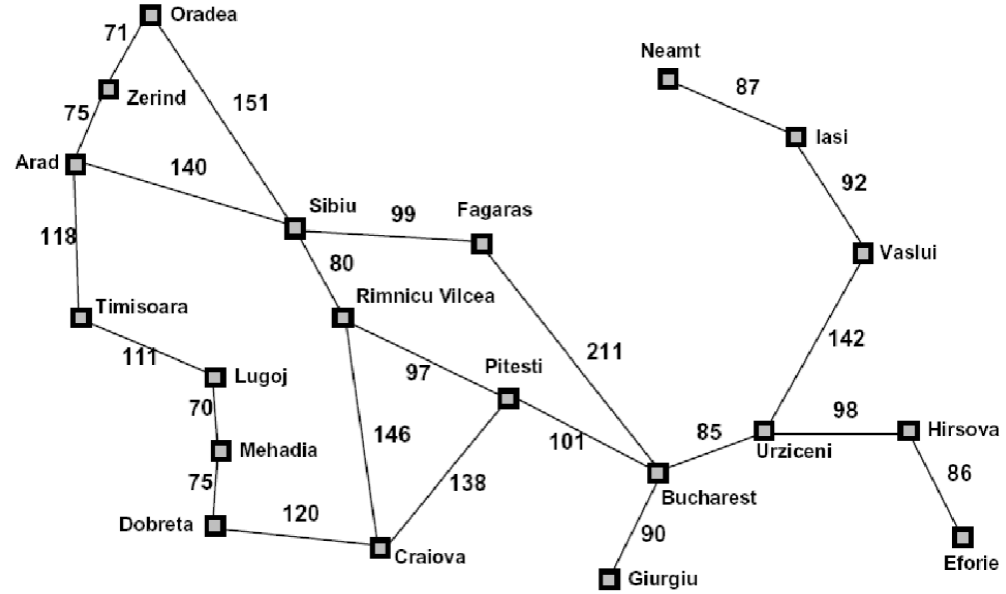


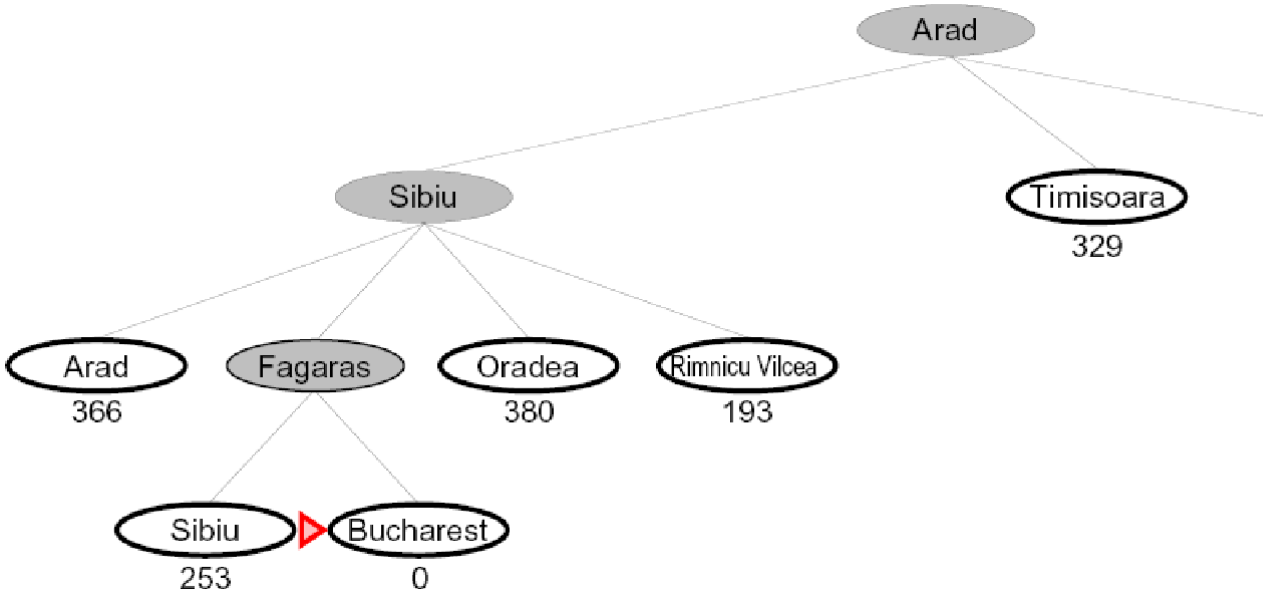
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

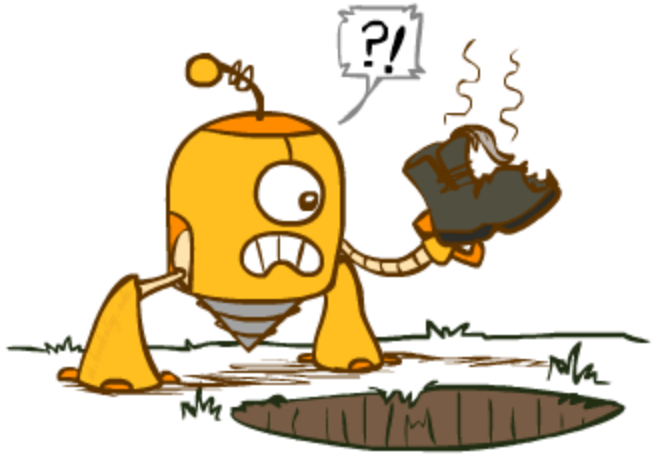
Greedy Search

- Expand the node that seems closest…



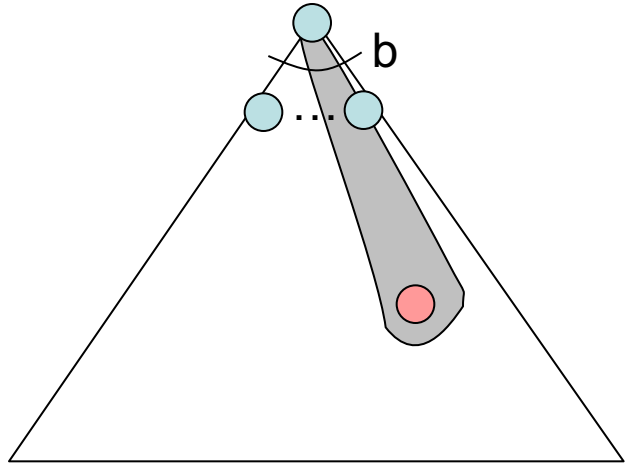


- What can go wrong?

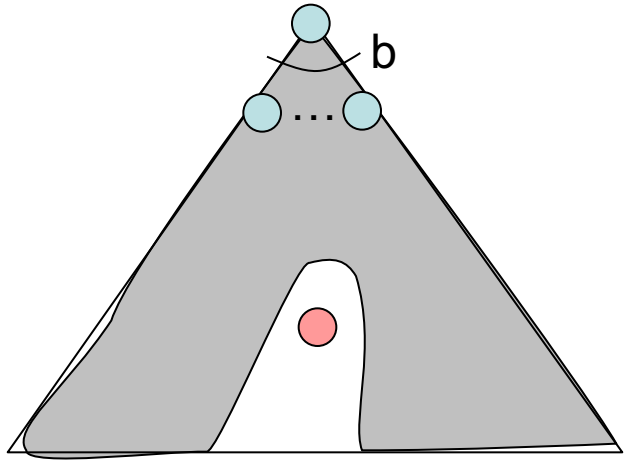


Greedy Search

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state



- A common case:
  - Best-first takes you straight to the (wrong) goal



- Worst-case: like a badly-guided DFS

[Demo: contours greedy empty (L3D1)]
[Demo: contours greedy pacman small maze (L3D4)]

Video of Demo Contours Greedy (Empty)

Video of Demo Contours Greedy (Pacman Small Maze)