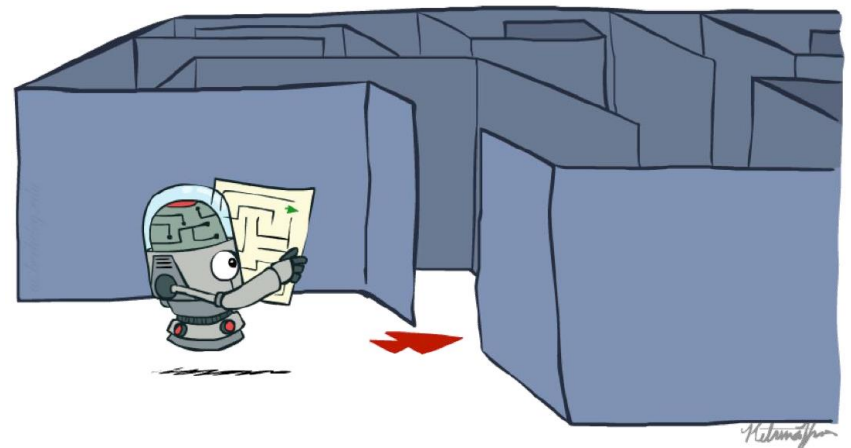
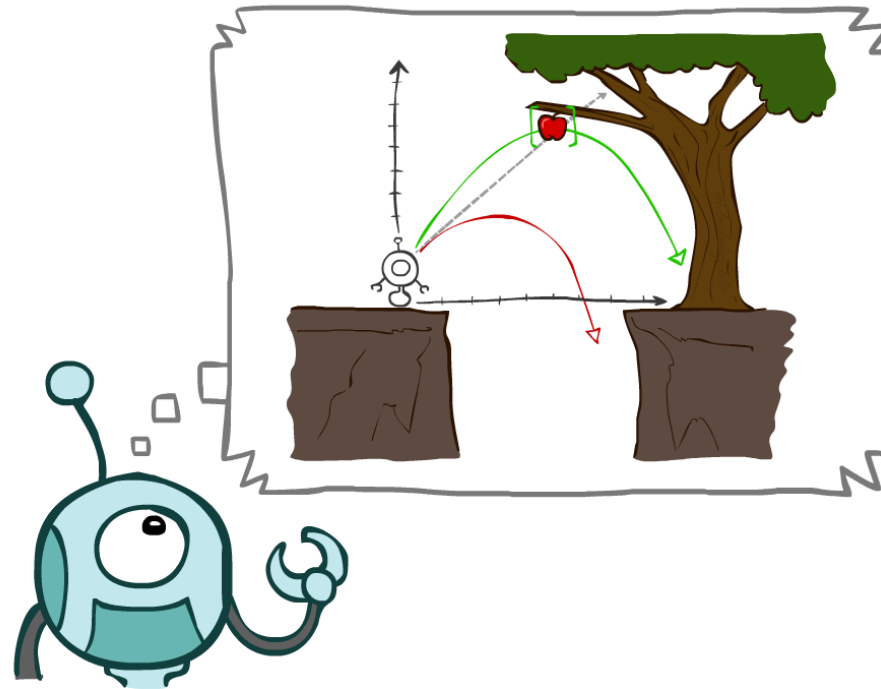


Uninformed Search

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

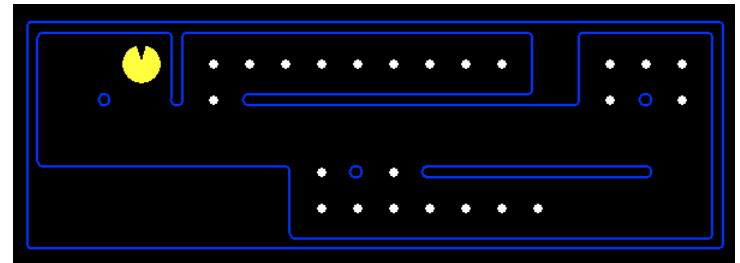
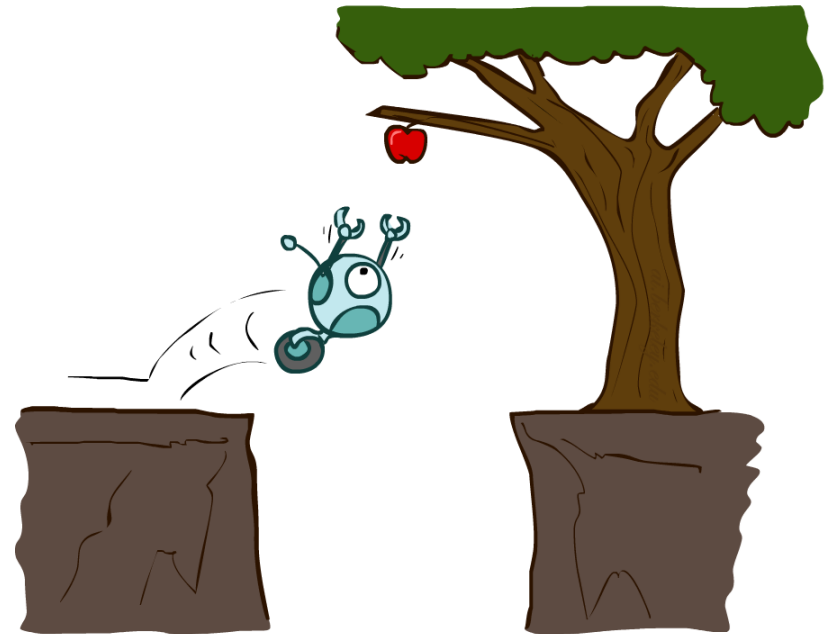


Agents that Plan

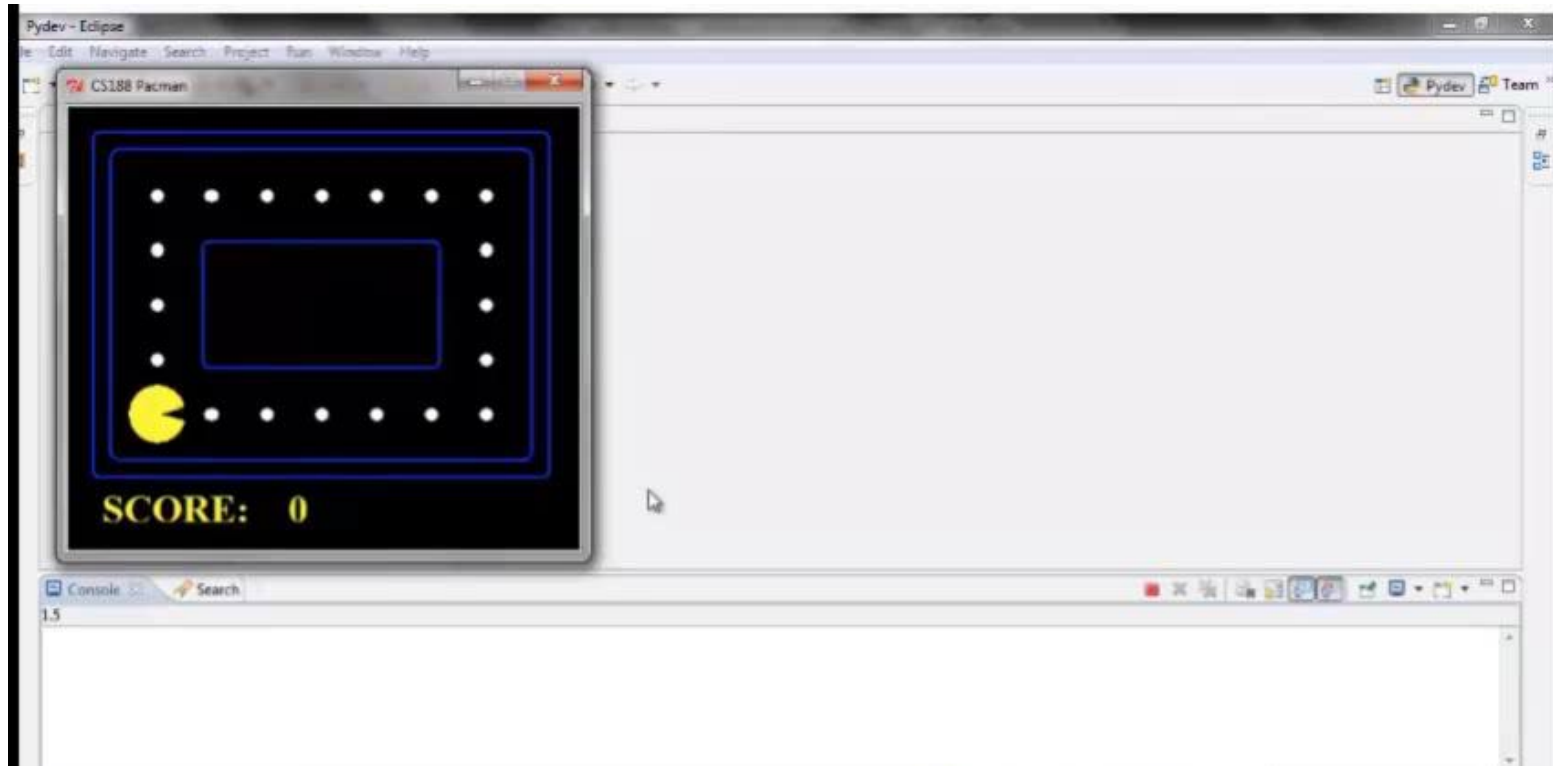


Reflex Agents

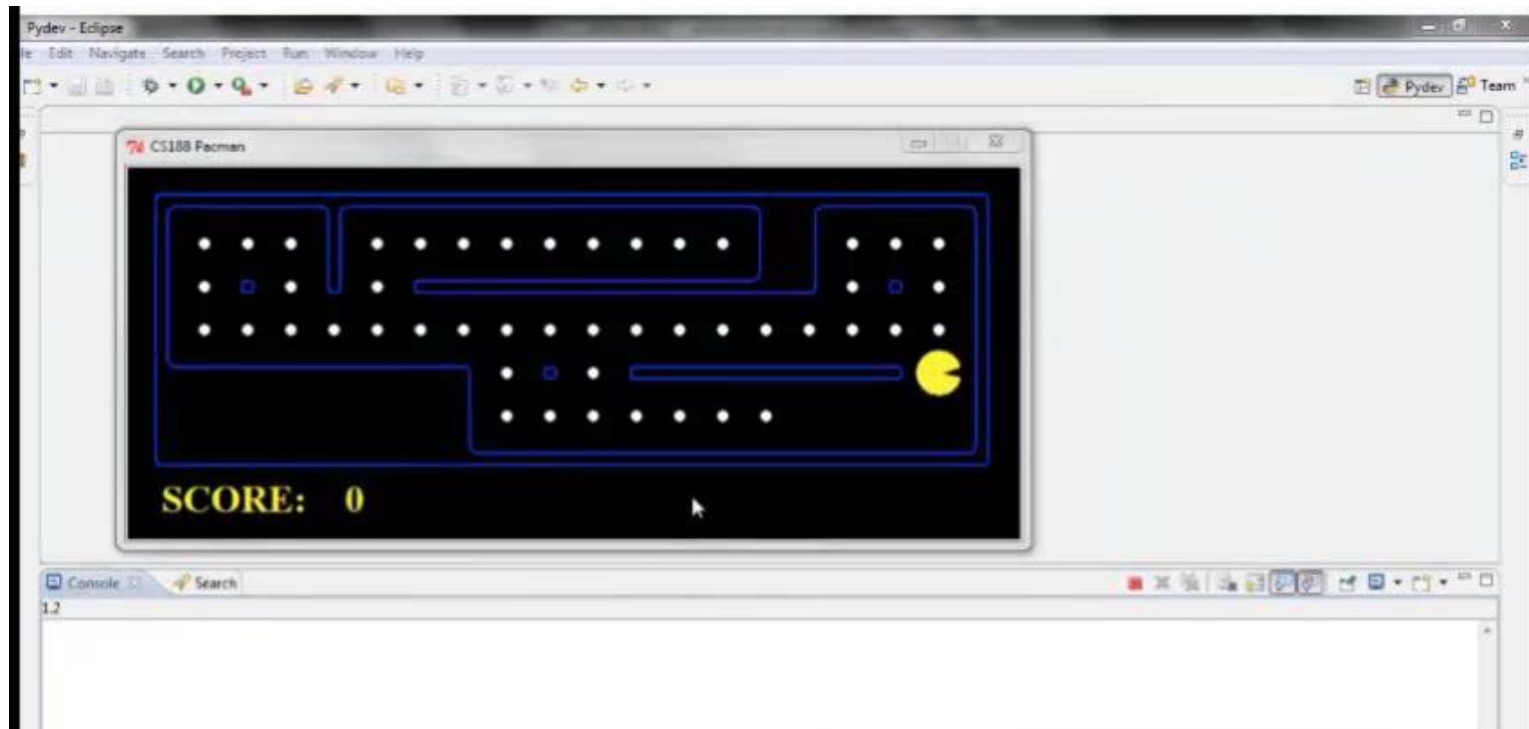
- Reflex agents:
 - Choose action based on current percept (and maybe memory)
 - May have memory or a model of the world's current state
 - Do not consider the future consequences of their actions
 - Consider how the world IS
- Can a reflex agent be rational?



Video of Demo Reflex Optimal

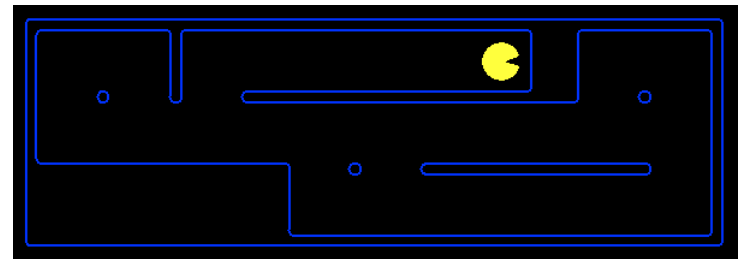
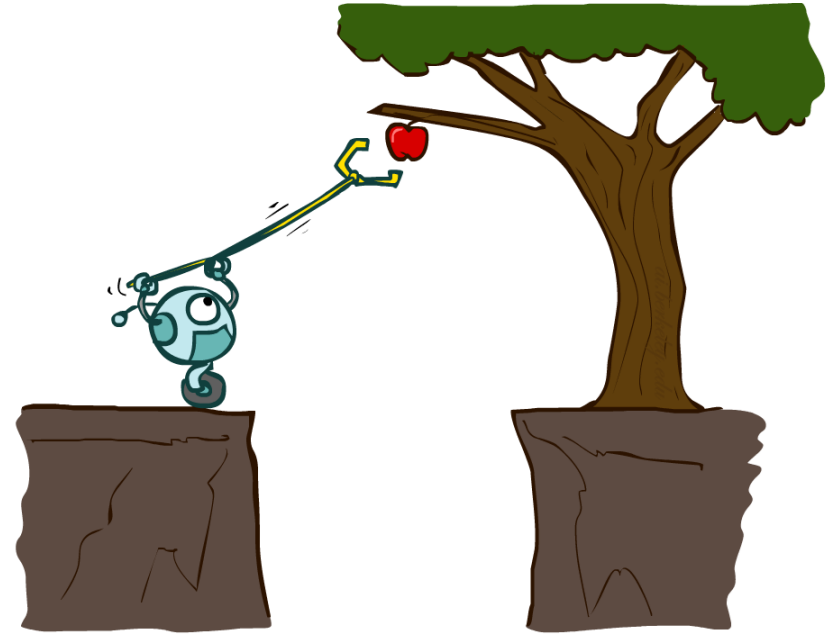


Video of Demo Reflex Odd

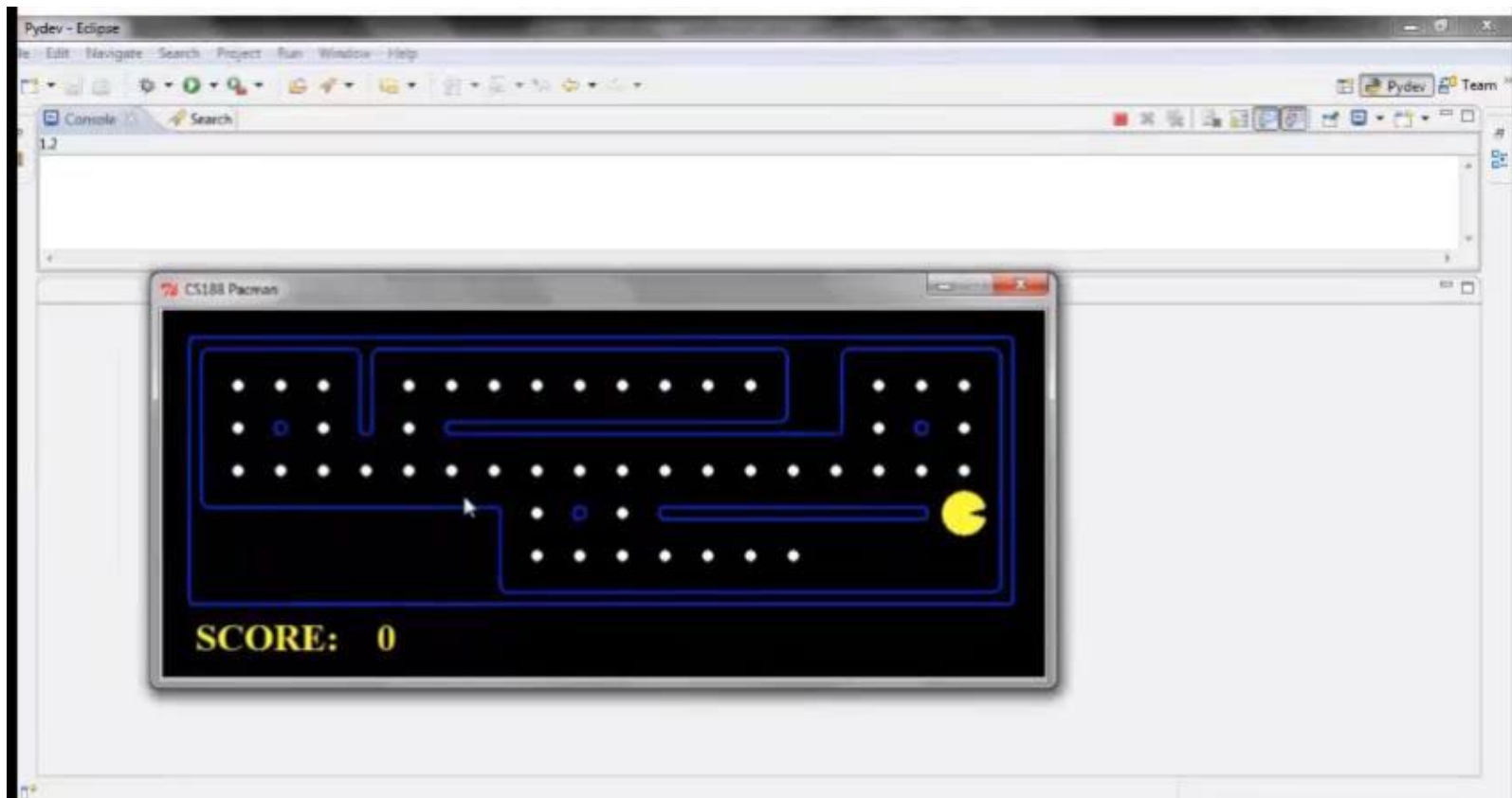


Planning Agents

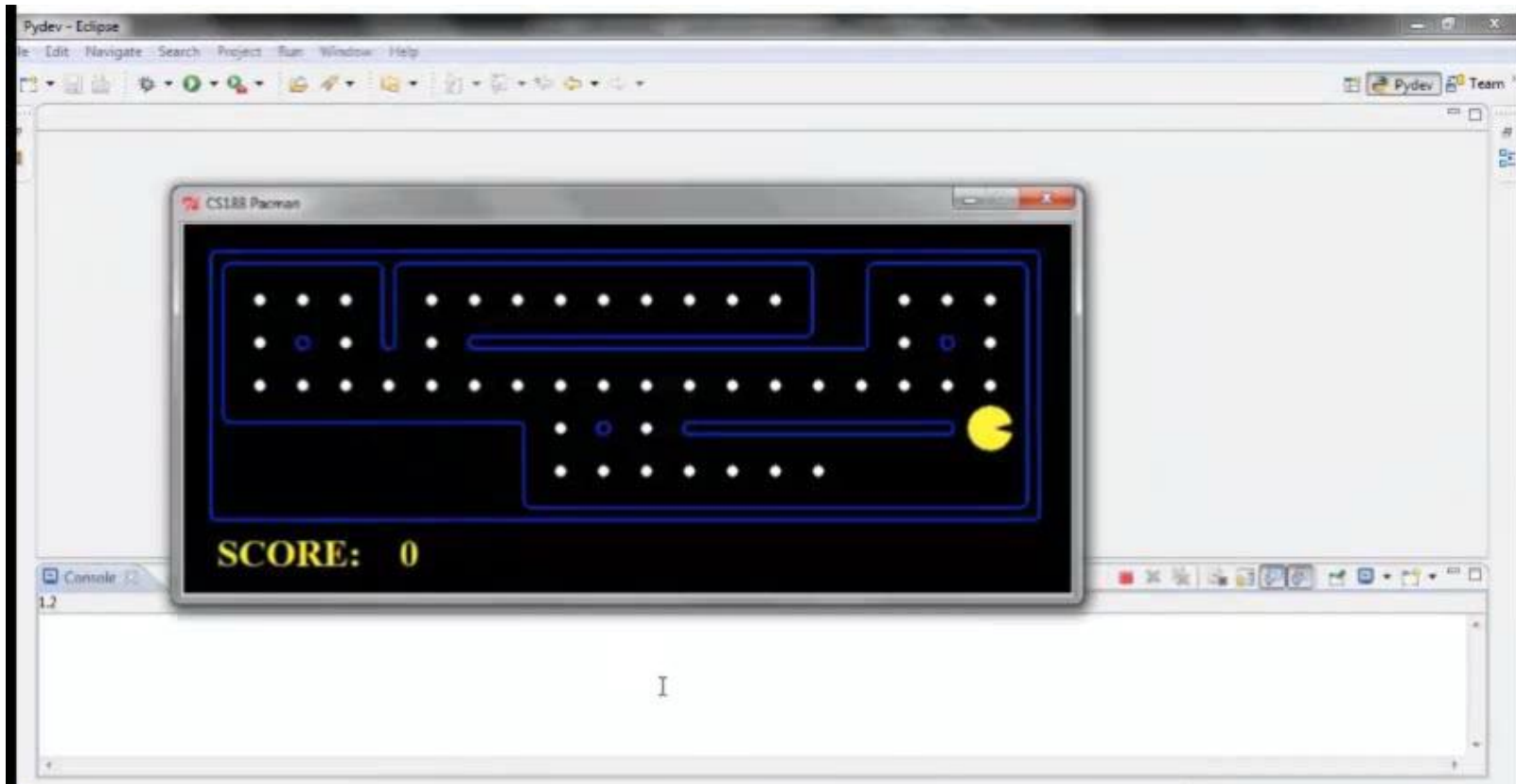
- Planning agents:
 - Ask “what if”
 - Decisions based on (hypothesized) consequences of actions
 - Must have a model of how the world evolves in response to actions
 - Must formulate a goal (test)
 - Consider how the world **WOULD BE**
- Optimal vs. complete planning
- Planning vs. replanning



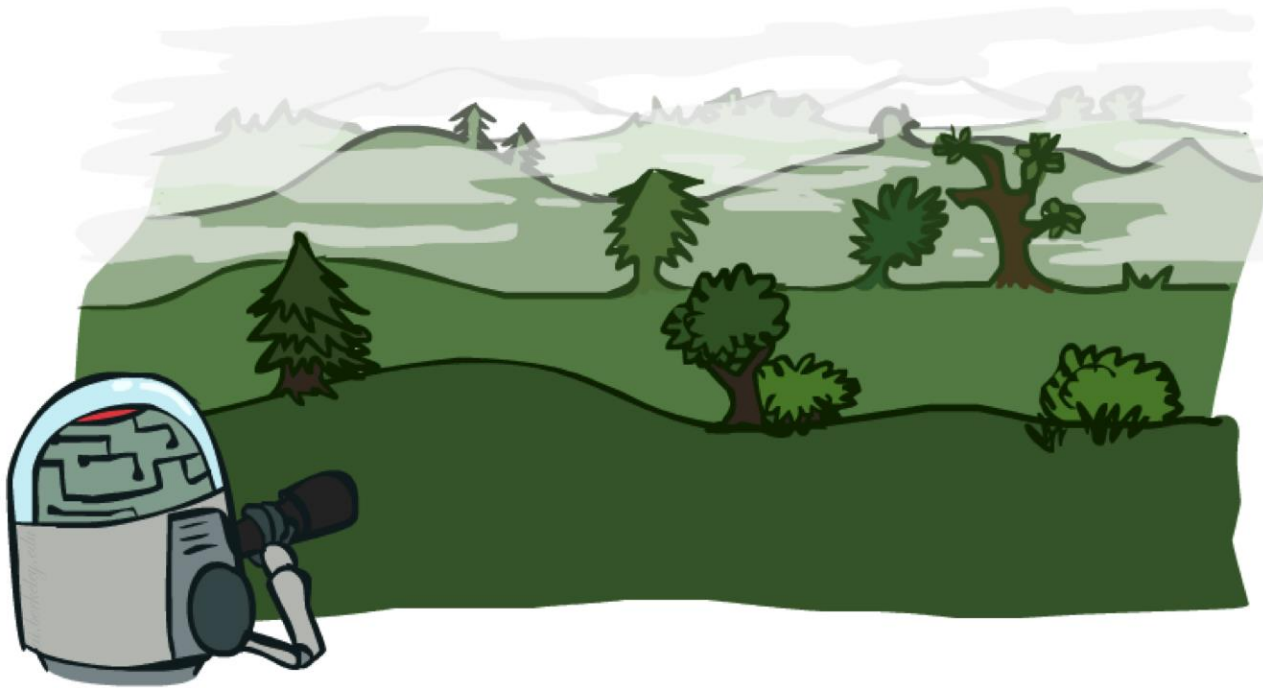
Video of Demo Mastermind



Video of Demo Replanning



Search Problems



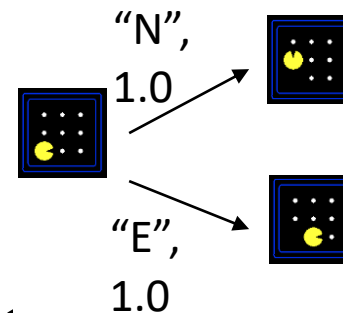
Search Problems

- A *search problem* consists of:

- A state space



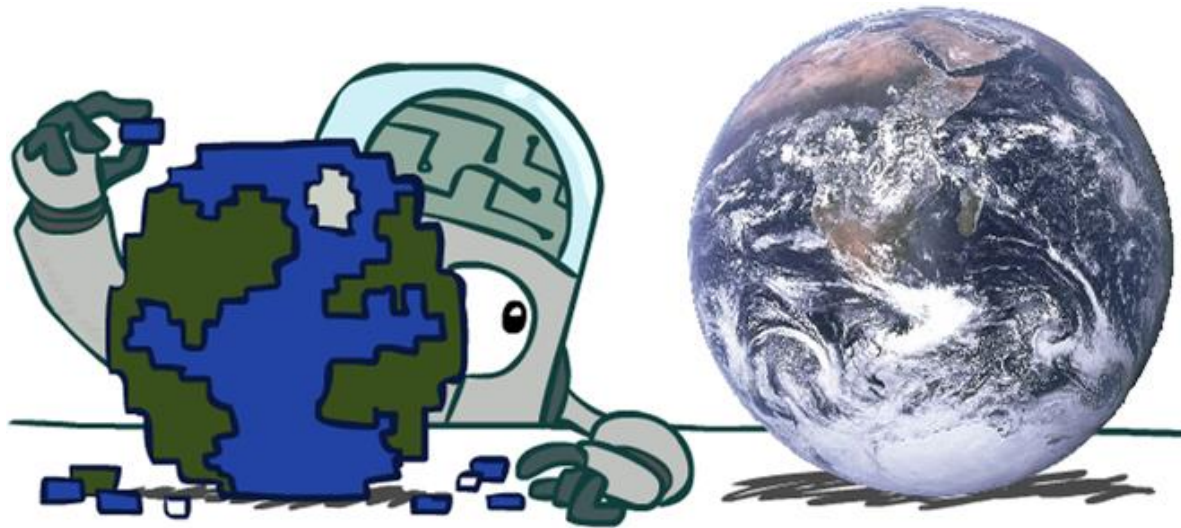
- A successor function
(with actions, costs)



- 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

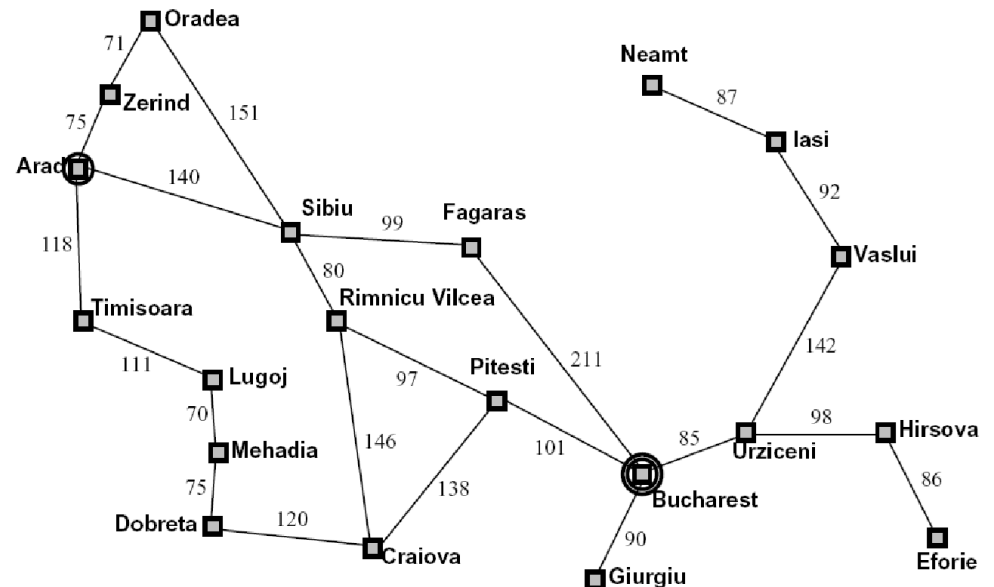


Example: Traveling in Romania

- On holiday in Romania; currently in Arad. Flight leaves tomorrow from Bucharest
- Formulate goal:
 - be in Bucharest
- Formulate problem:
 - states: various cities actions: drive between cities
- Find solution:
 - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

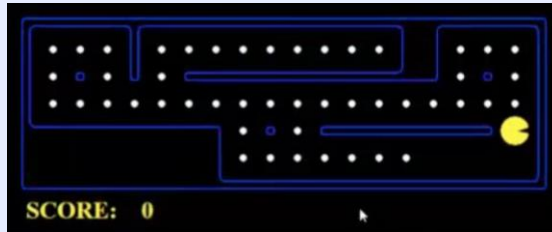
Example: Traveling in Romania

- 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

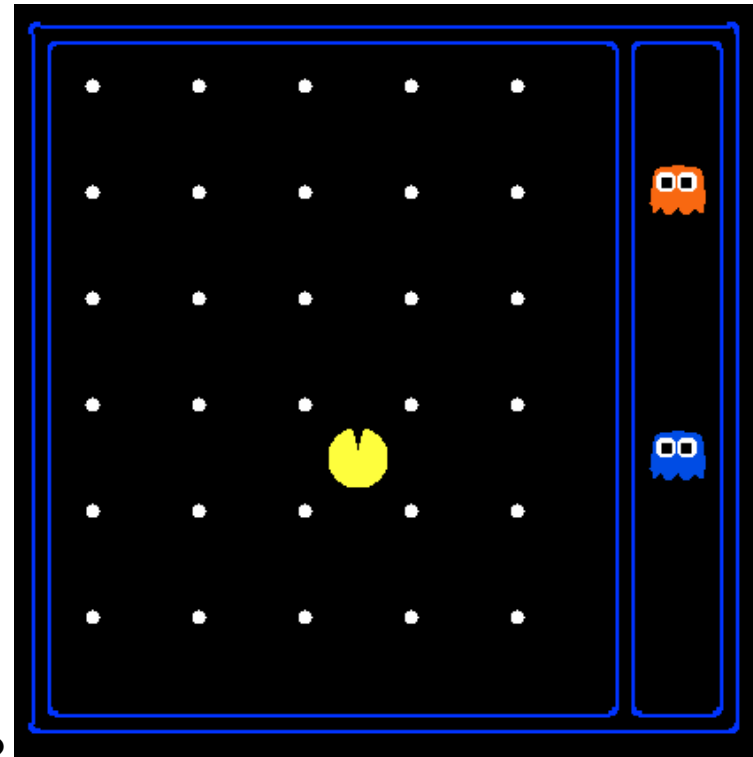
■ Problem: Eat-All-Dots

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

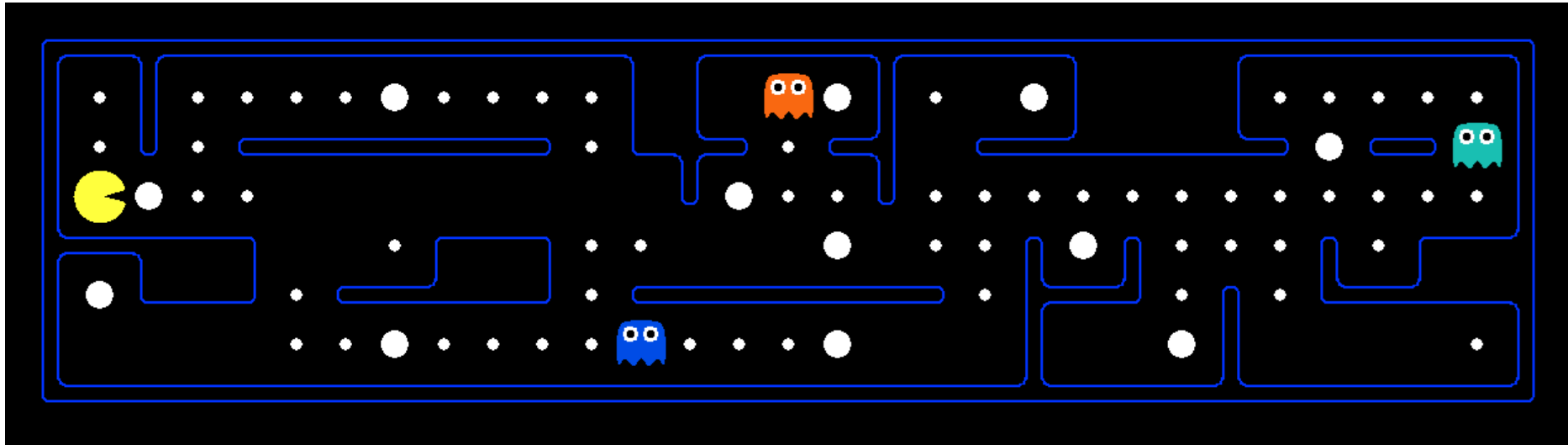
State Space Sizes?

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

- How many
 - World states?
 $120 \times (2^{30}) \times (12^2) \times 4$
 - States for Pathing Problem?
 120
 - States for Eat-All-Dots Problem?
 $120 \times (2^{30})$

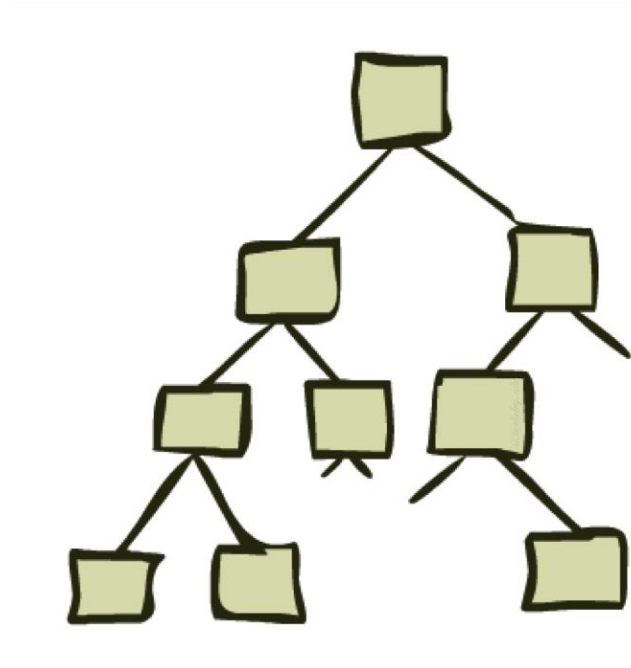


Quiz: Safe Passage



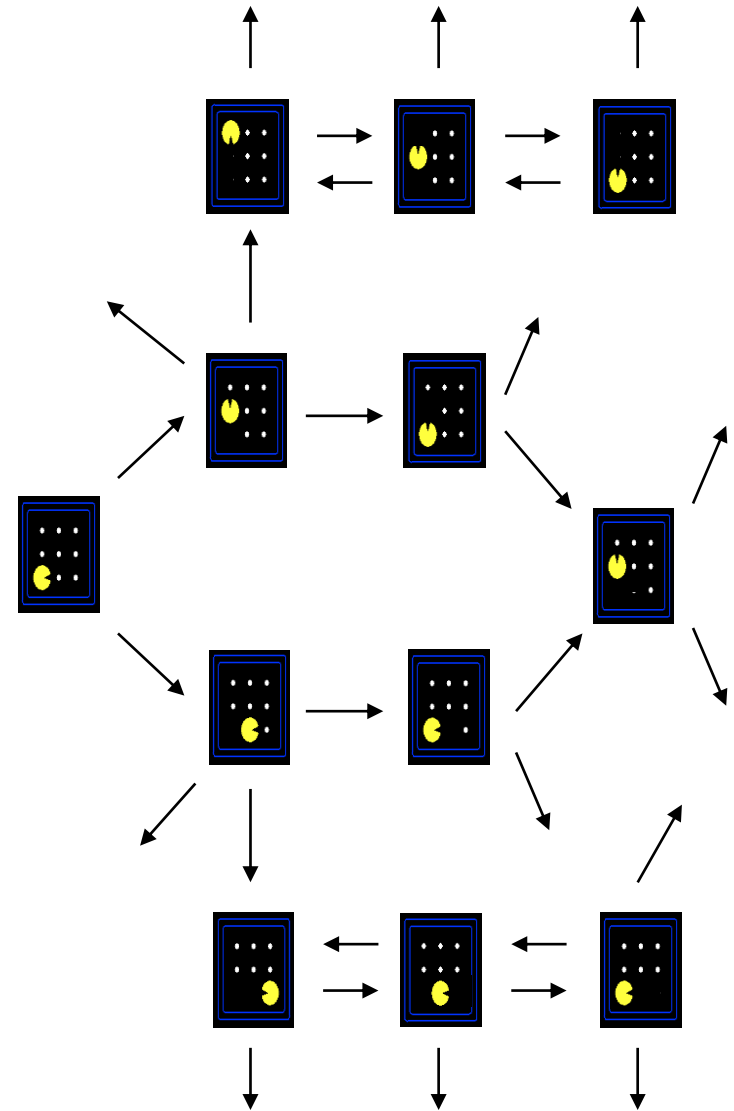
- Problem: eat all dots while keeping the ghosts perma-scared
- What does the state space have to specify?

State Space Graphs and Search Trees



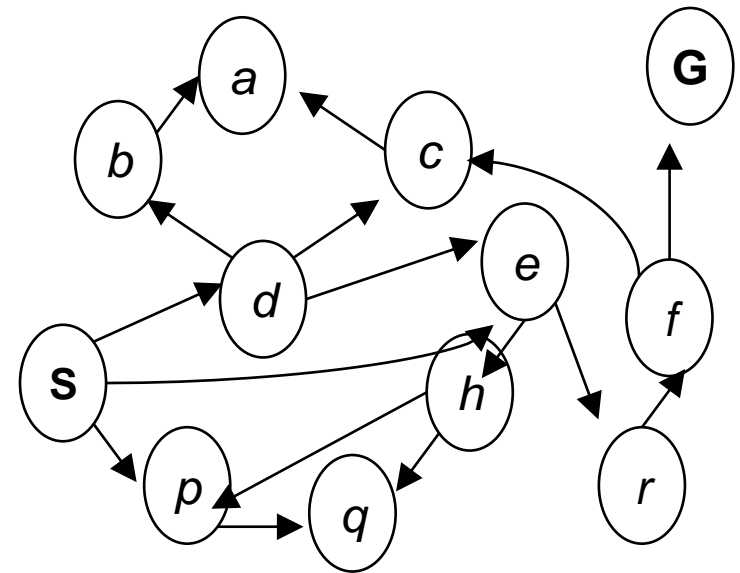
State Space Graphs

- 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



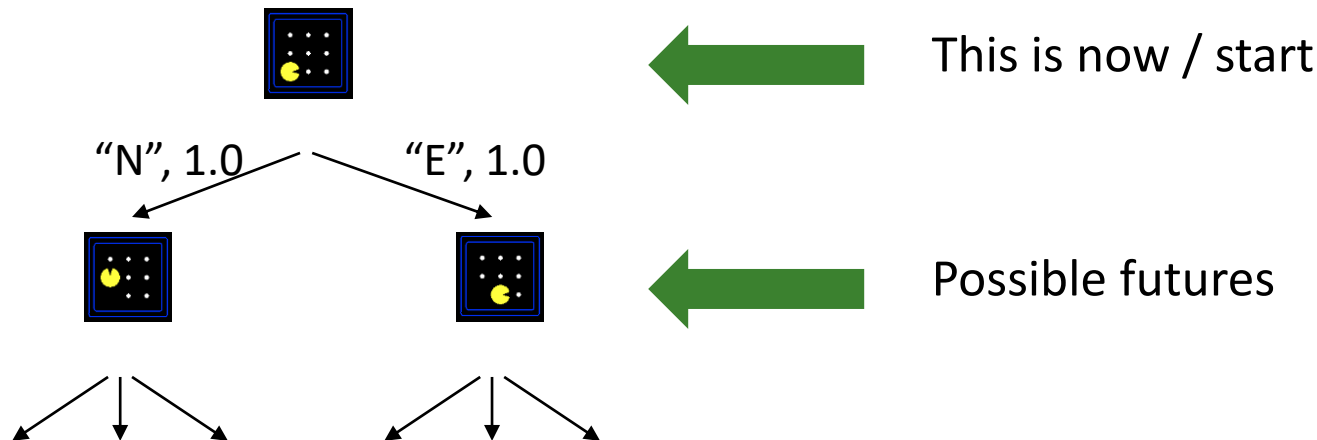
State Space Graphs

- 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

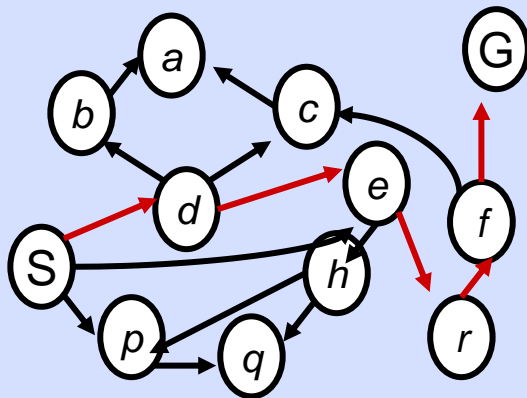


■ A search tree:

- 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

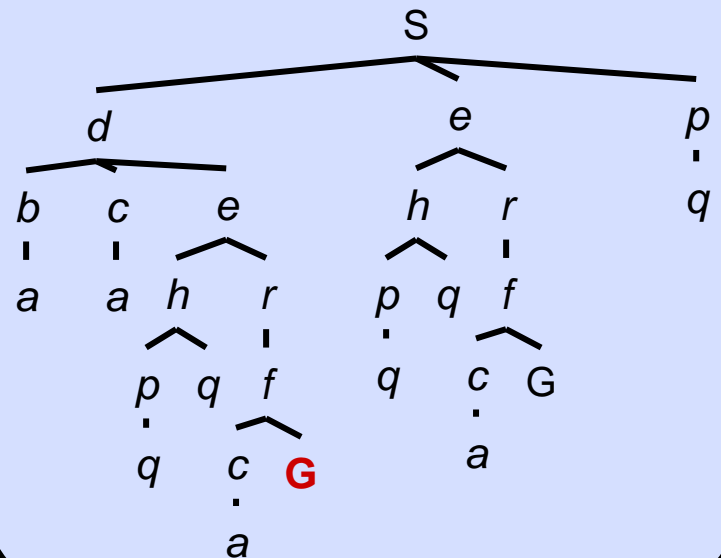
State Space Graph



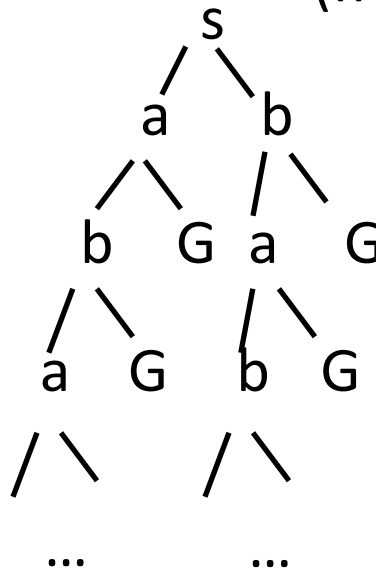
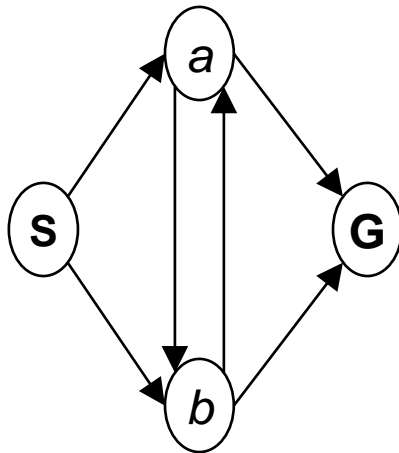
Each NODE in the search tree is an entire PATH in the state space graph.

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

Search Tree

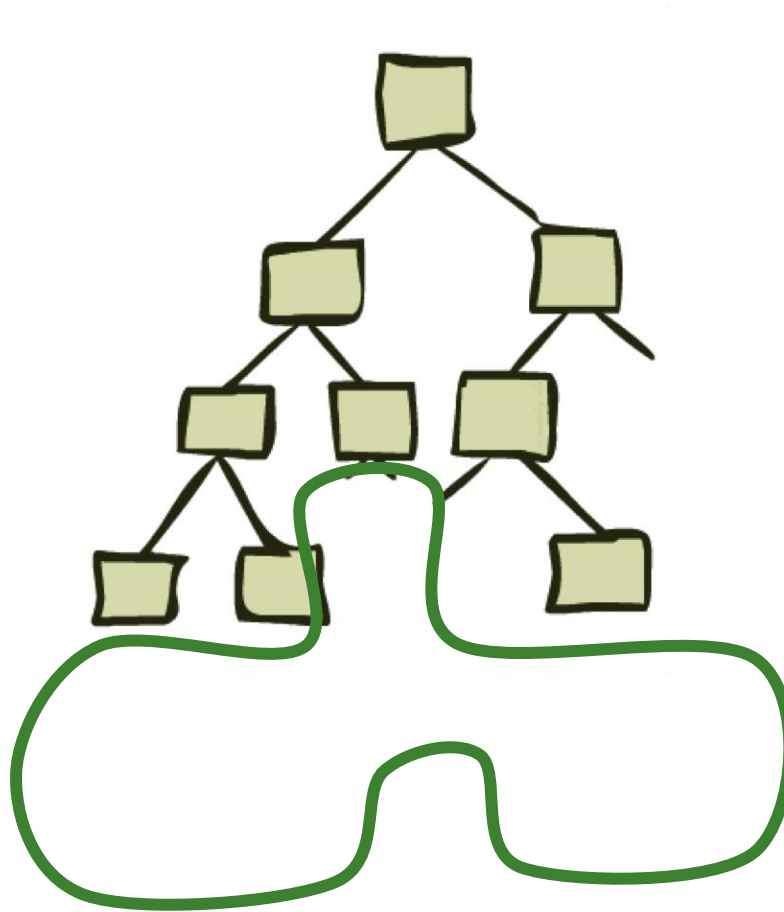


How big is its search tree (from S)?

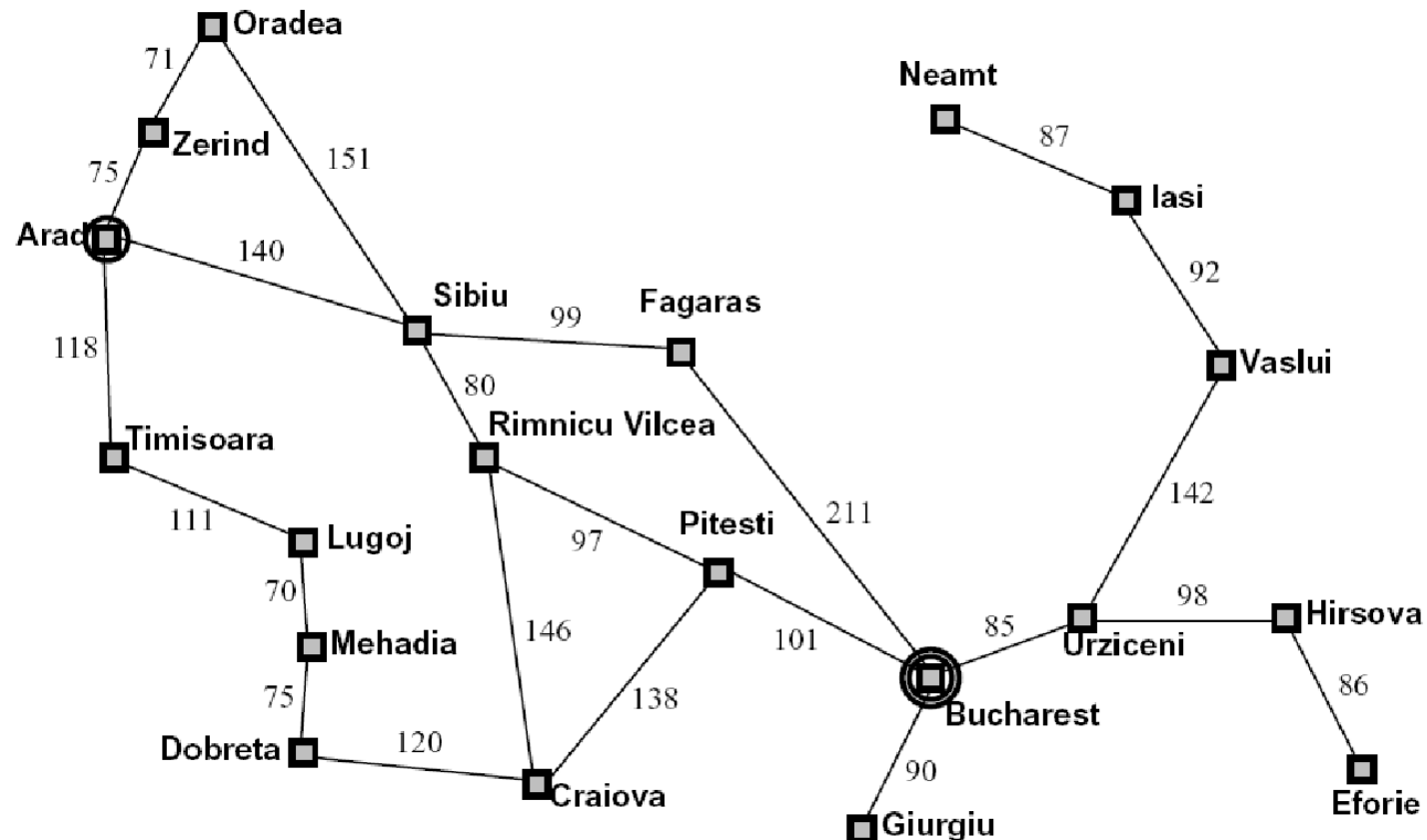


Important: Lots of repeated structure in the search tree!

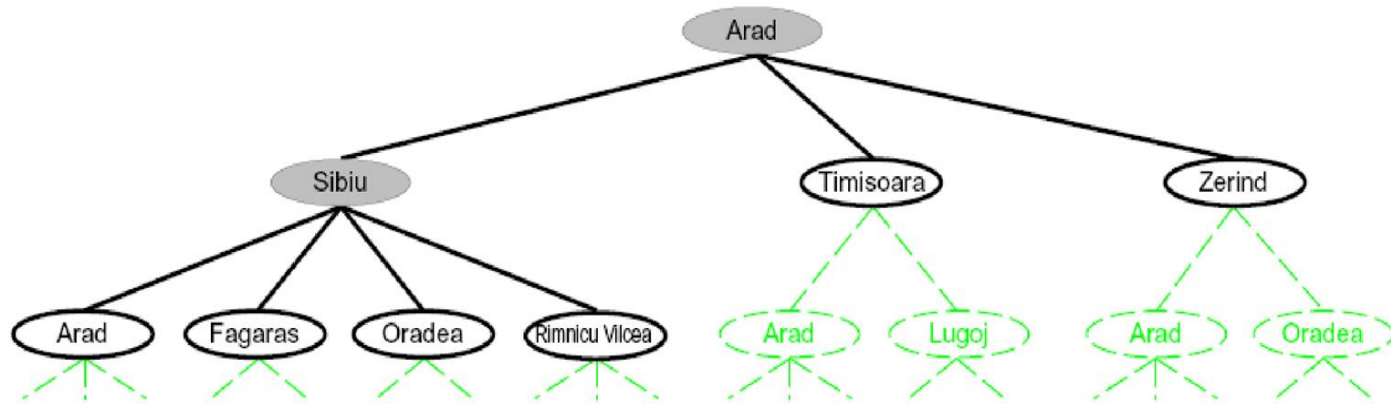
Tree Search



Search Example: Romania



Searching with a Search Tree



■ Search:

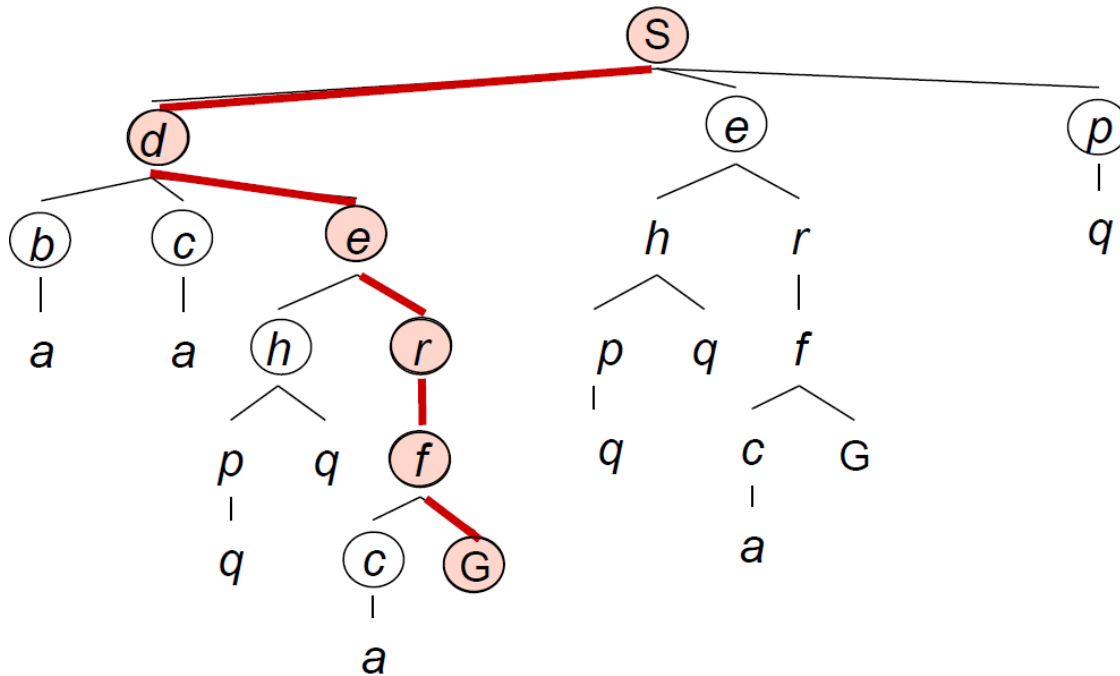
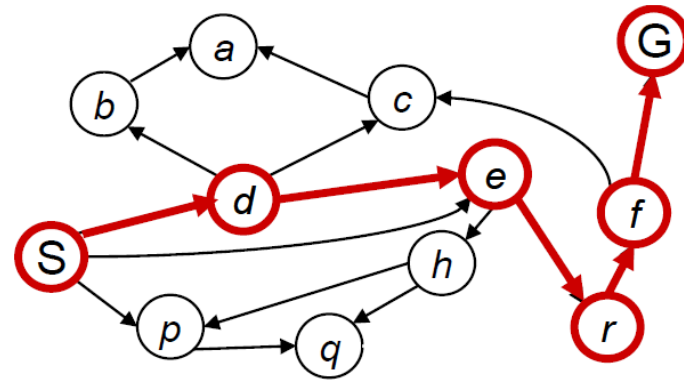
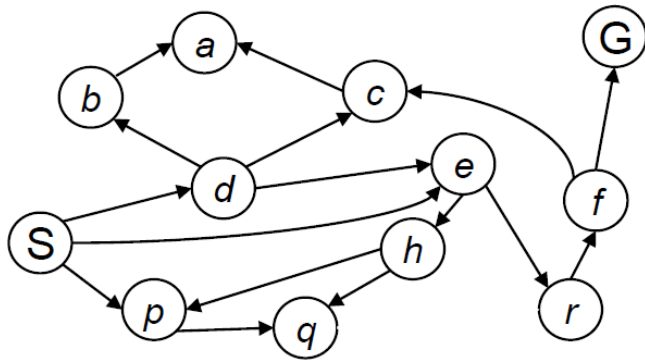
- ❑ Expand out potential plans (tree nodes)
- ❑ Maintain a **fringe** of partial plans under consideration
- ❑ Try to expand as few tree nodes as possible

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



~~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~~

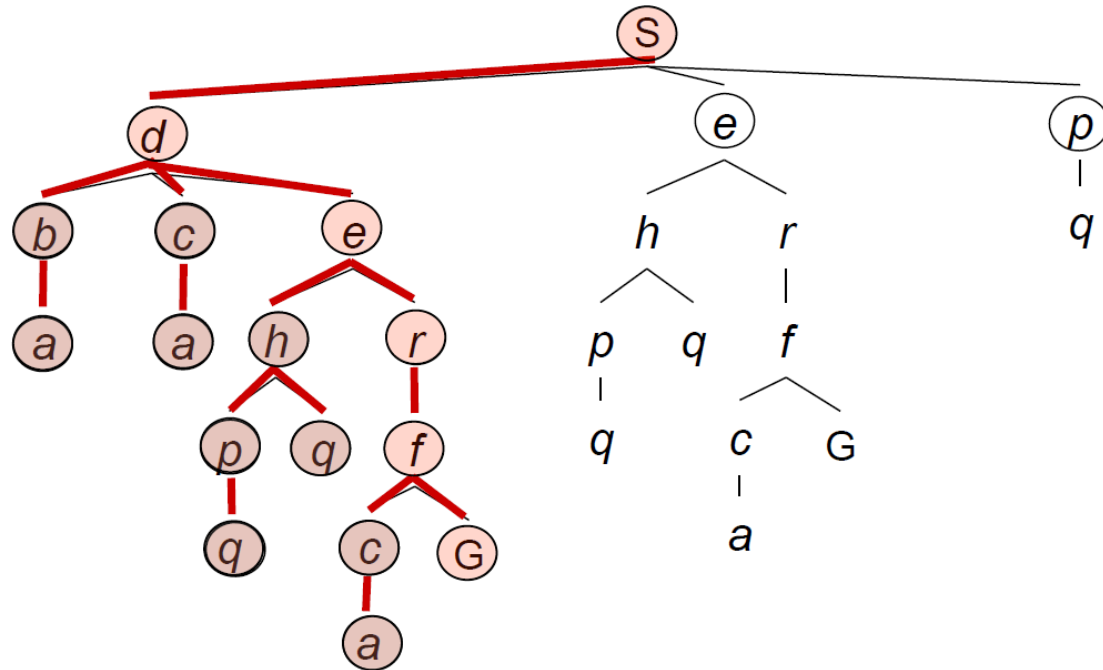
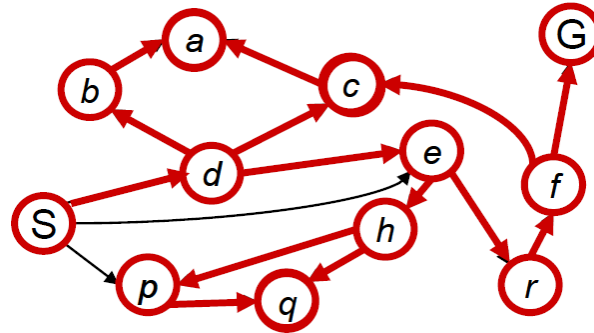
Depth-First Search



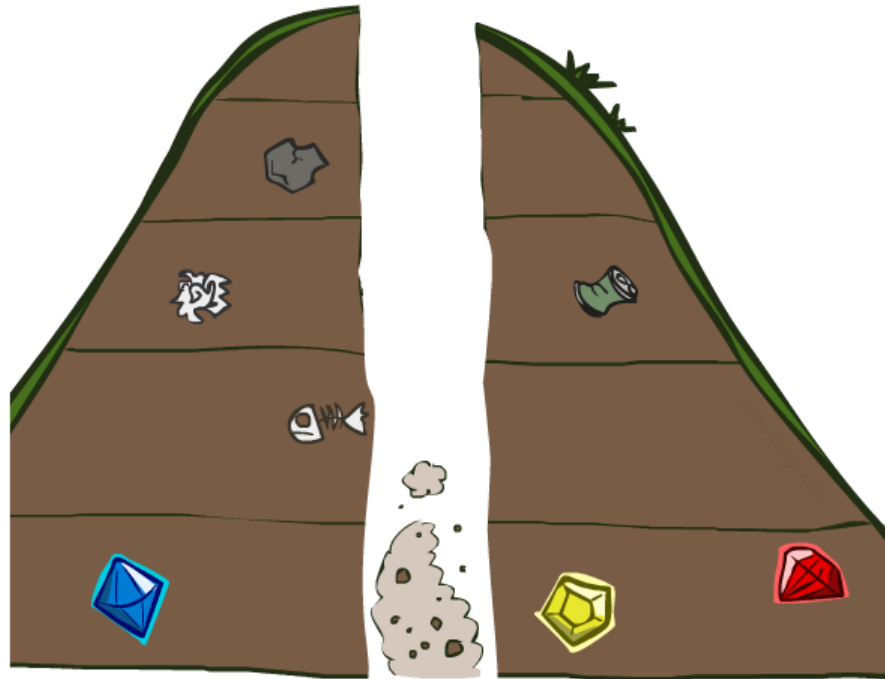
Depth-First Search

Strategy: expand a deepest node first

*Implementation:
Fringe is a LIFO stack*



Search Algorithm Properties



Search Algorithm Properties

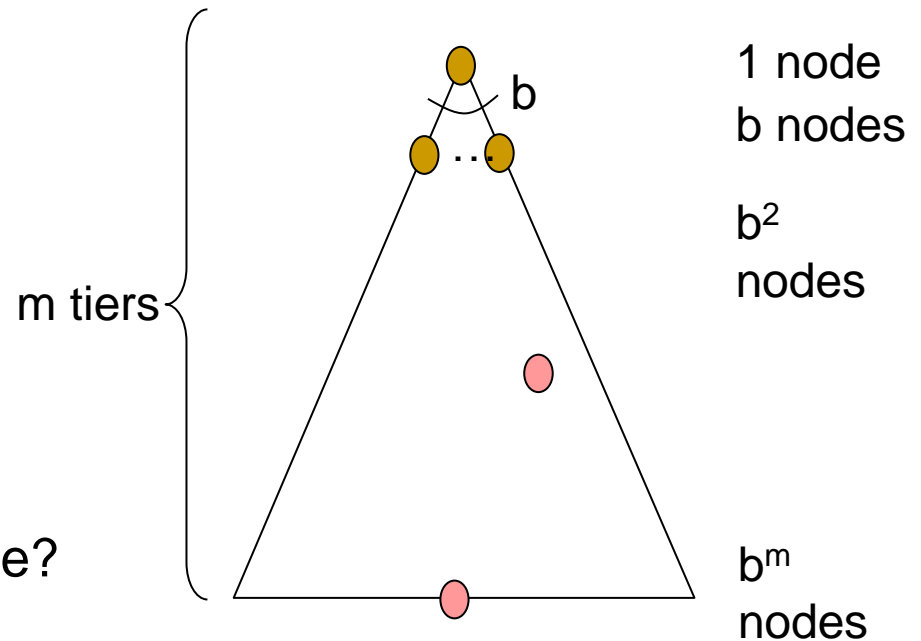
- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

- Cartoon of search tree:

- b is the branching factor
- m is the maximum depth
- solutions at various depths

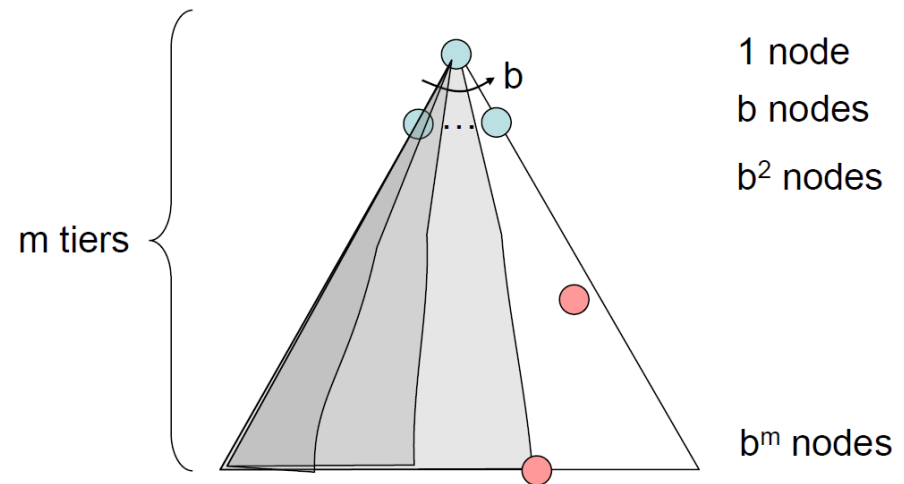
- Number of nodes in entire tree?

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

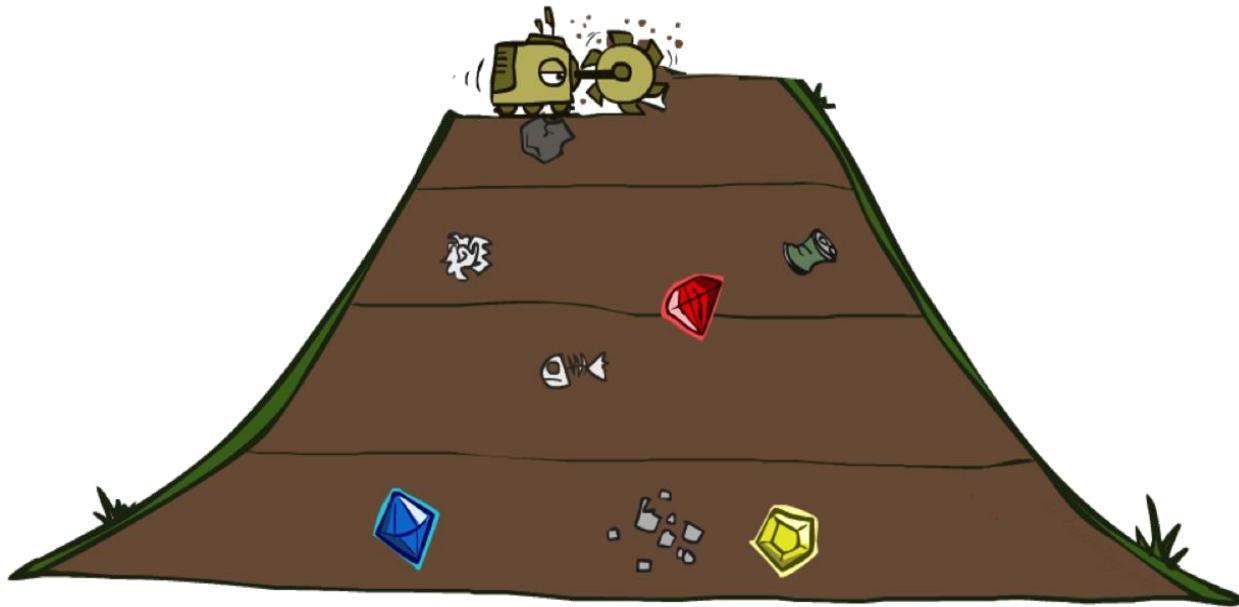


Depth-First Search (DFS) Properties

- 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



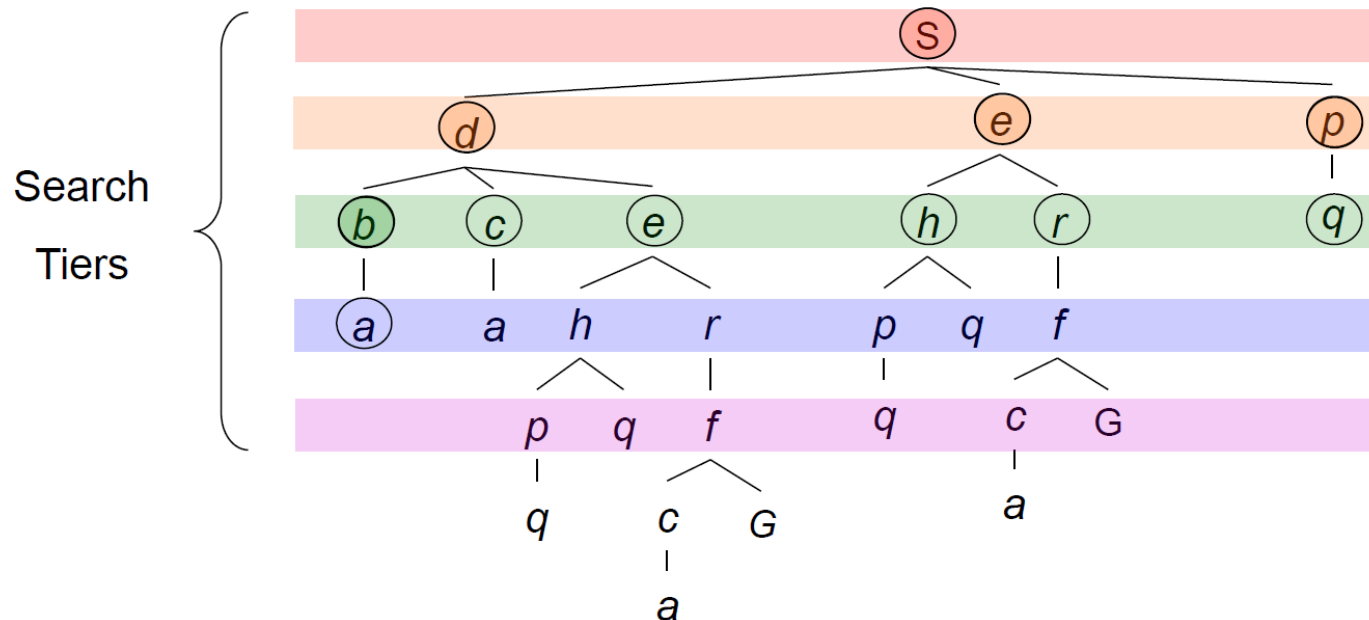
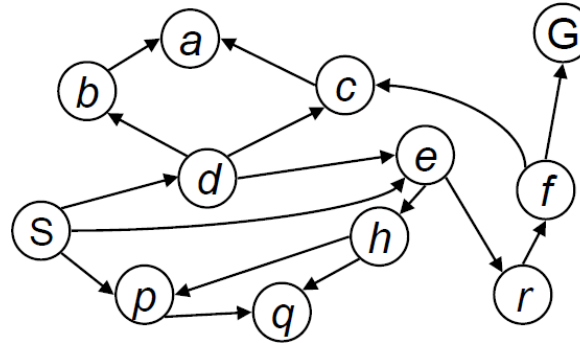
Breadth-First Search



Breadth-First Search

*Strategy: expand
a shallowest
node first*

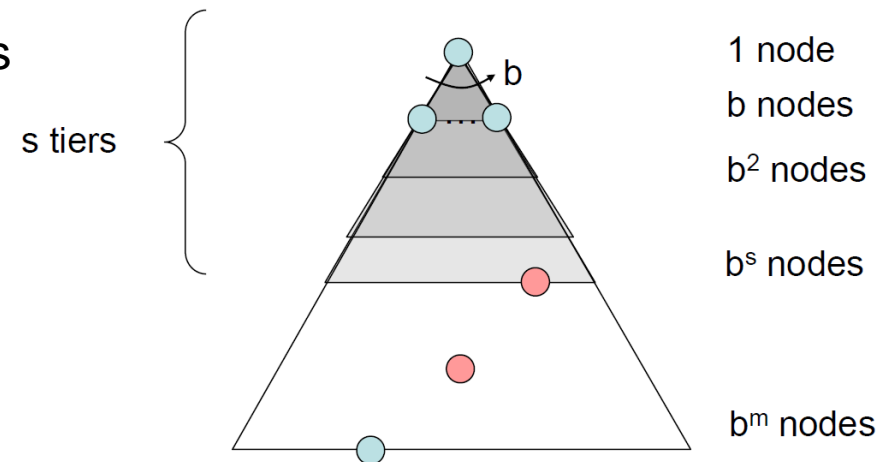
*Implementation:
Fringe is a FIFO
queue*



Breadth-First Search (BFS)

Properties

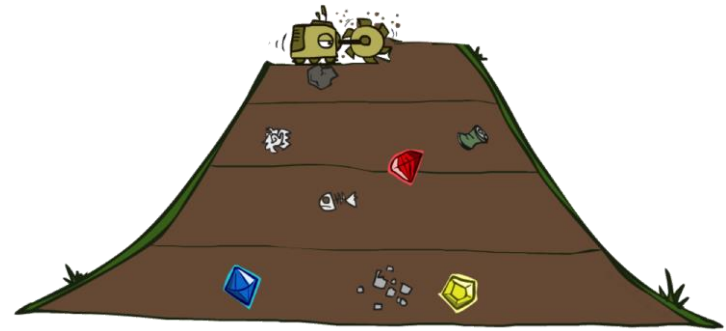
- What nodes BFS expand?
 - Processes all nodes above shallowest solution
 - Let depth of shallowest solution be s
 - Search takes time $O(b^s)$
- How much space does the fringe take?
 - Has roughly the last tier, so $O(b^s)$
- Is it complete?
 - s must be finite if a solution exists, so yes!
- Is it optimal?
 - Only if costs are all 1 (more on costs later)



Quiz: DFS vs BFS

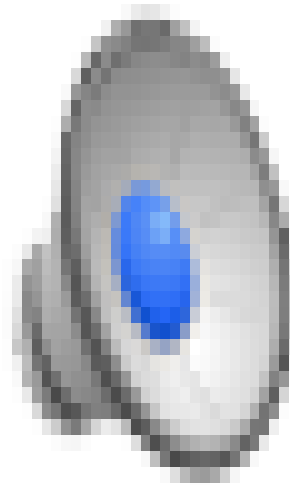


VS

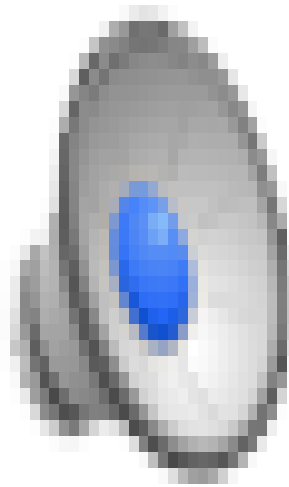


- When will BFS outperform DFS?
- When will DFS outperform BFS?

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

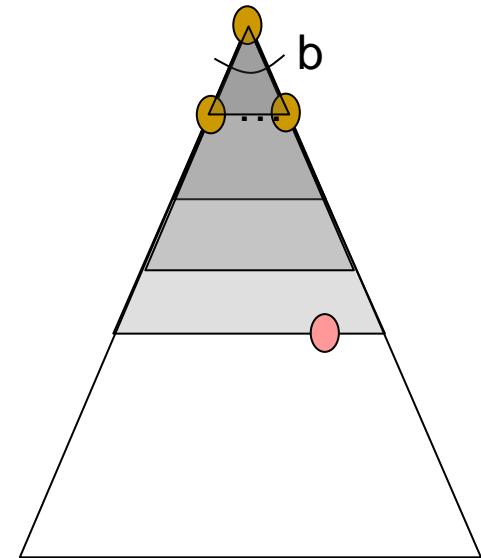


Video of Demo Maze Water DFS/BFS (part 2)

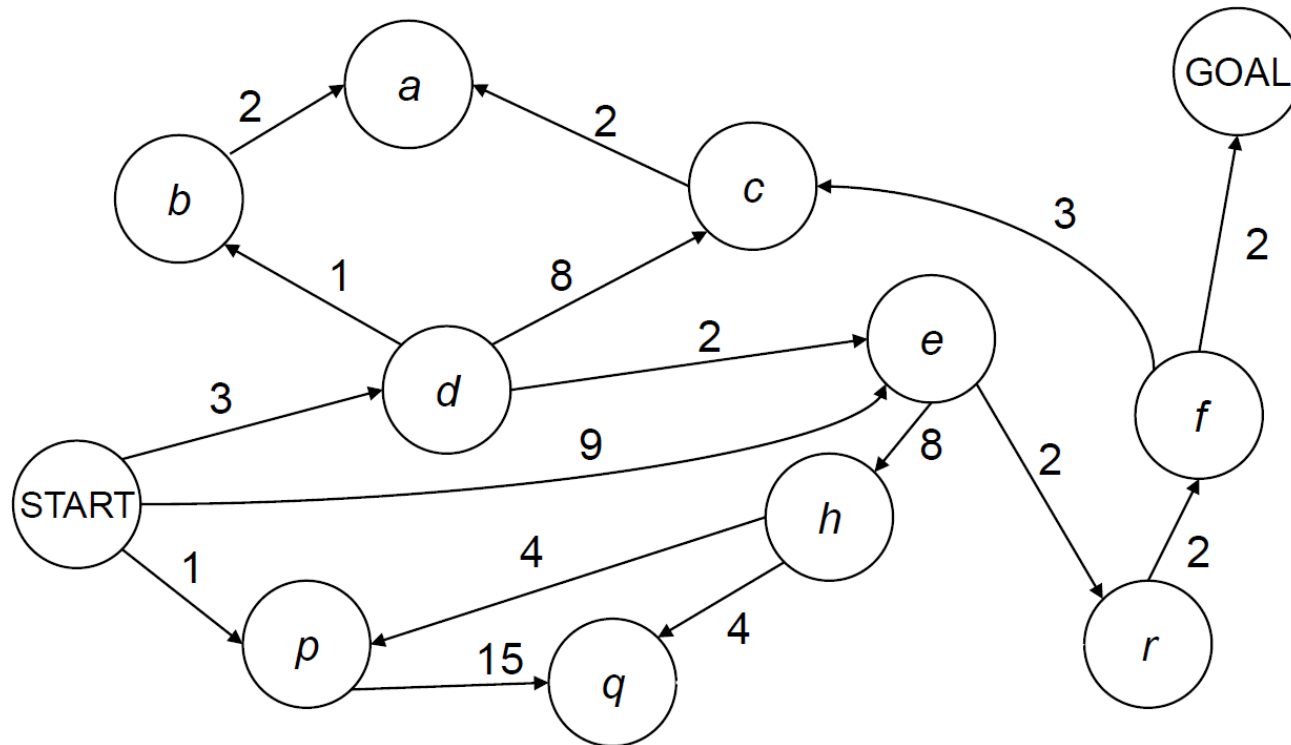


Iterative Deepening

- 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!

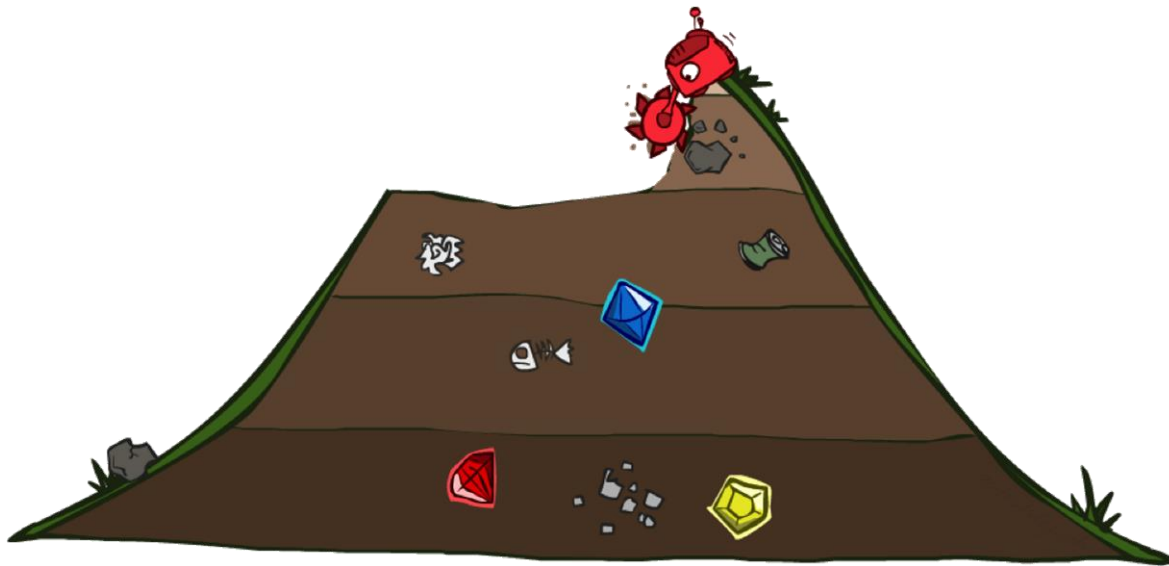


Cost-Sensitive Search



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.

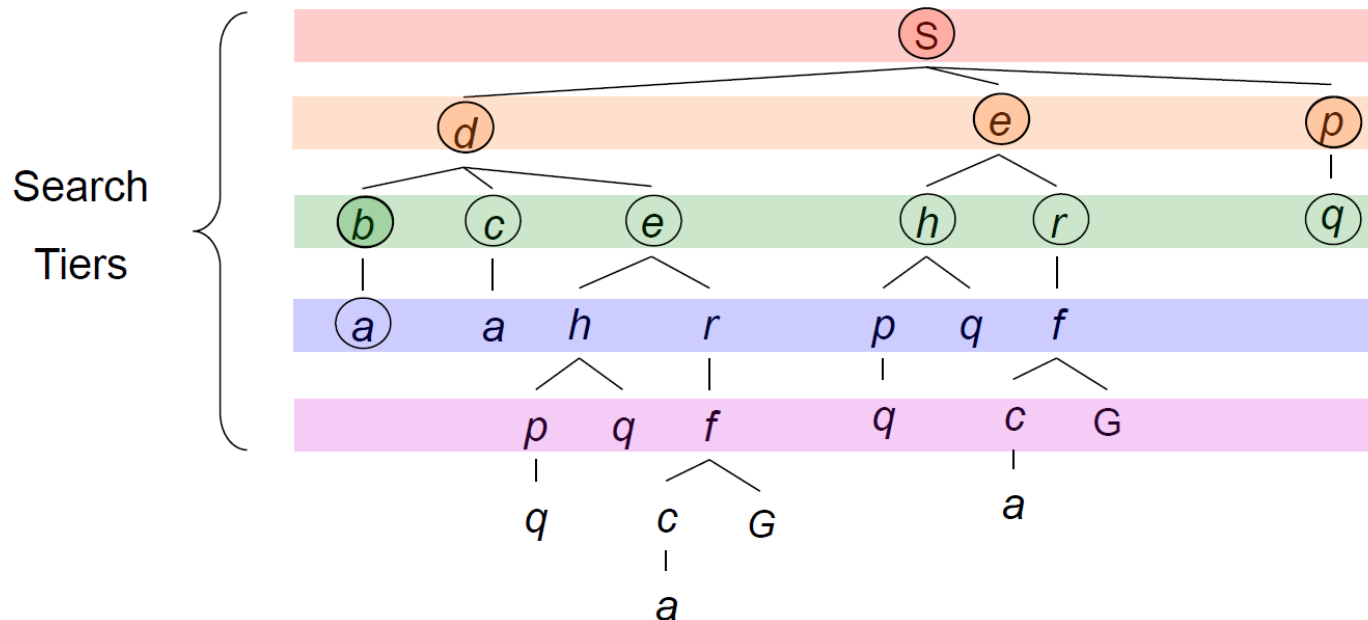
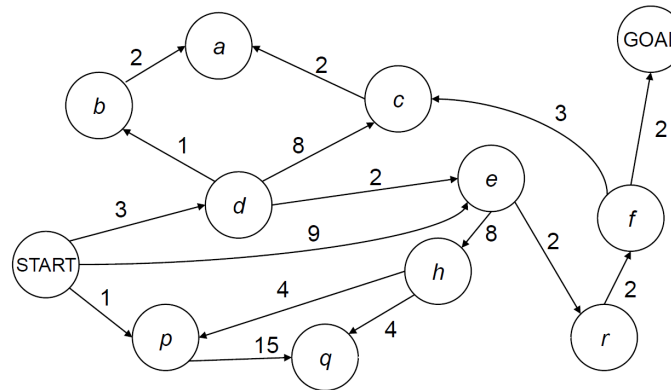
Uniform Cost Search



Uniform Cost Search

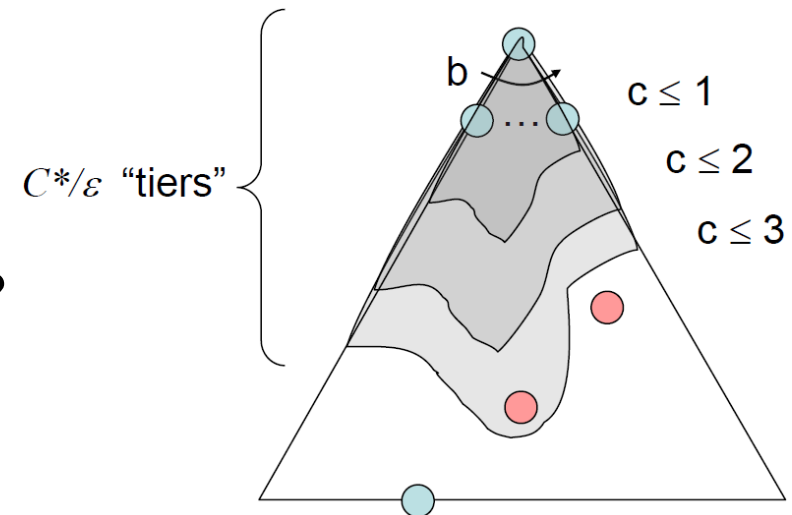
Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)



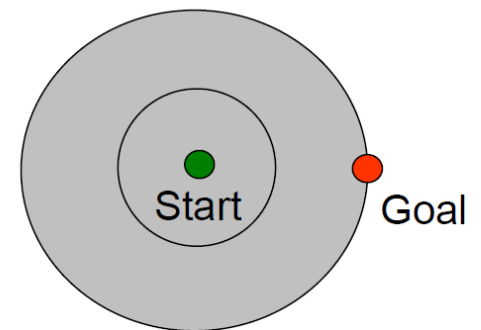
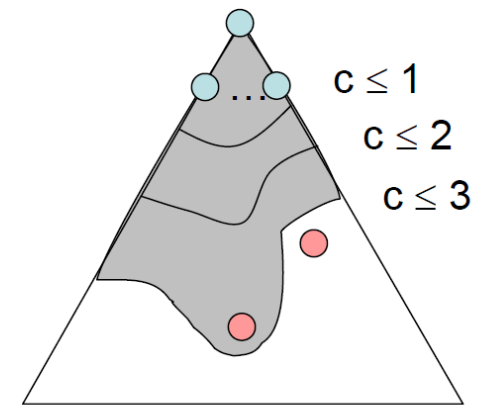
Uniform Cost Search (UCS) Properties

- 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)
- 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!

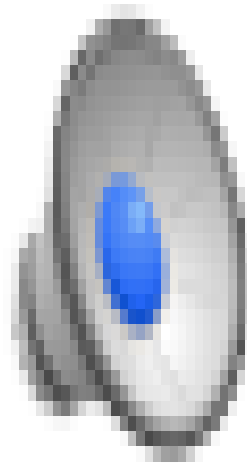


Uniform Cost Issues

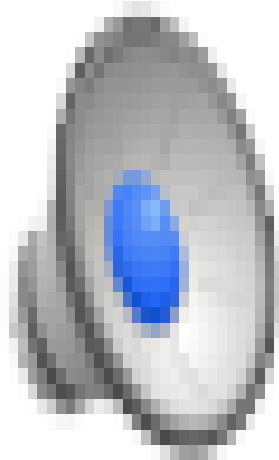
- Remember: UCS explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
 - Explores options in every “direction”
 - No information about goal location



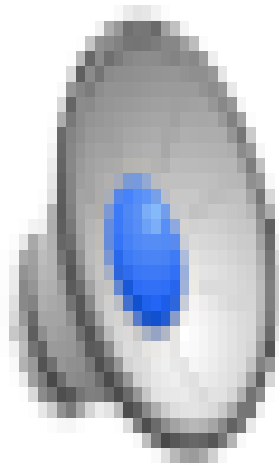
Video of Demo Empty UCS



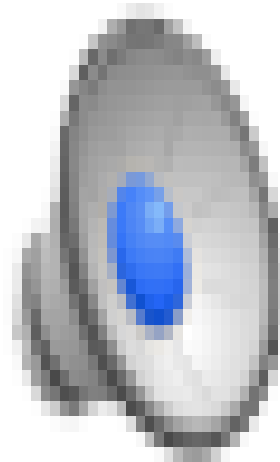
Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 1)



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

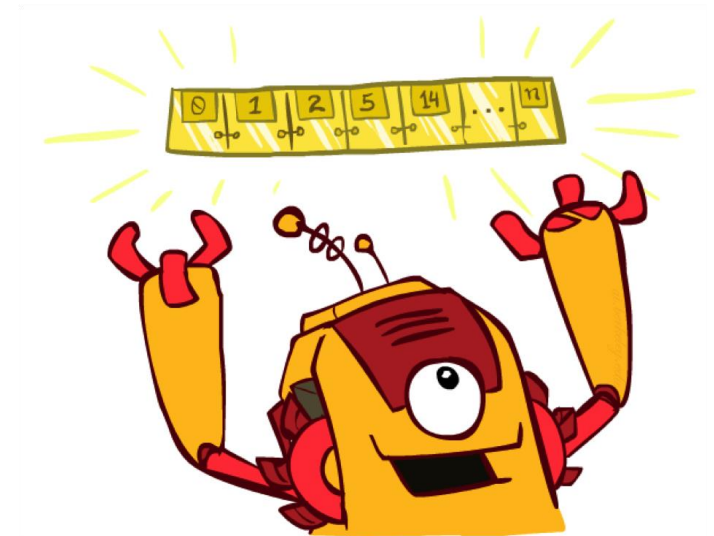


Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 3)

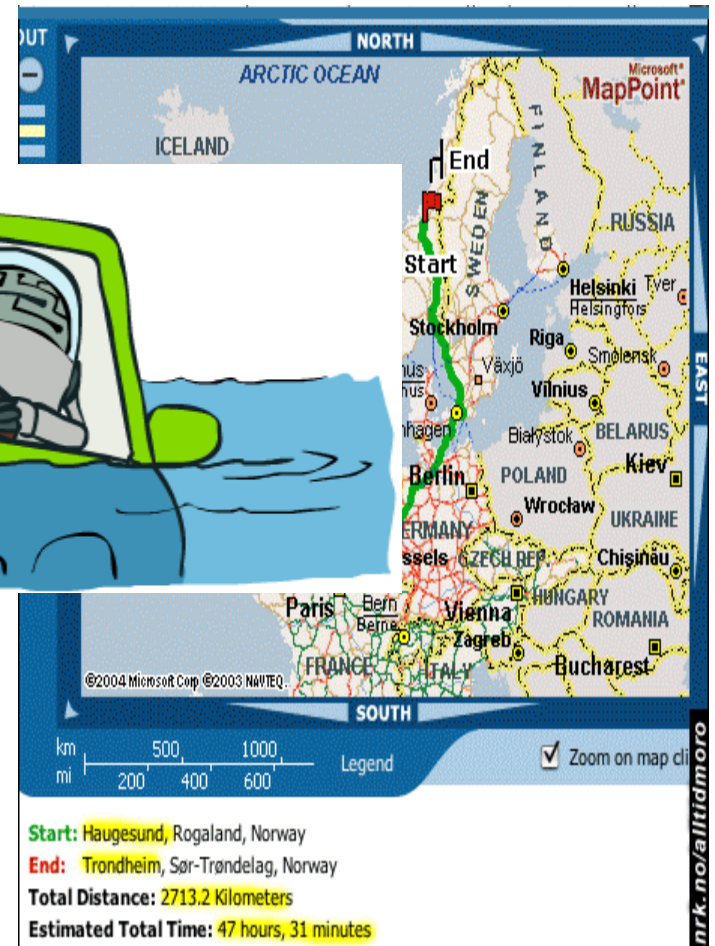
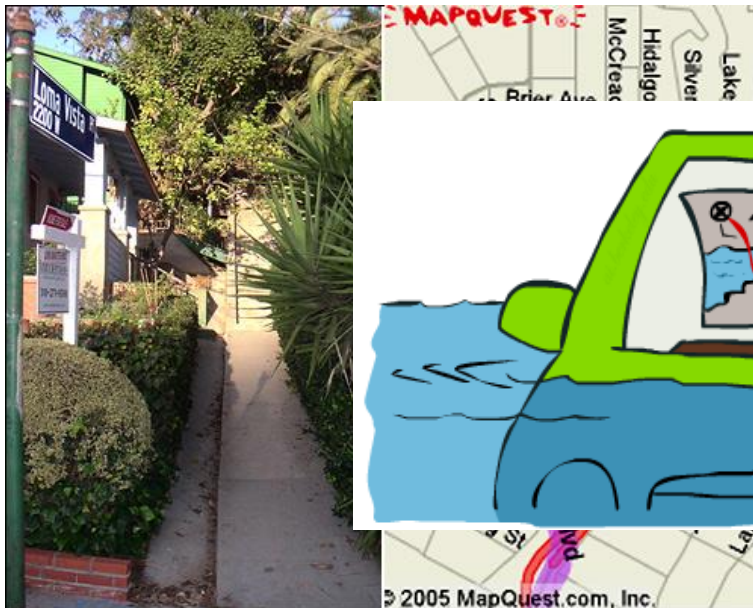


The One Queue

- 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



Search Gone Wrong?



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...

