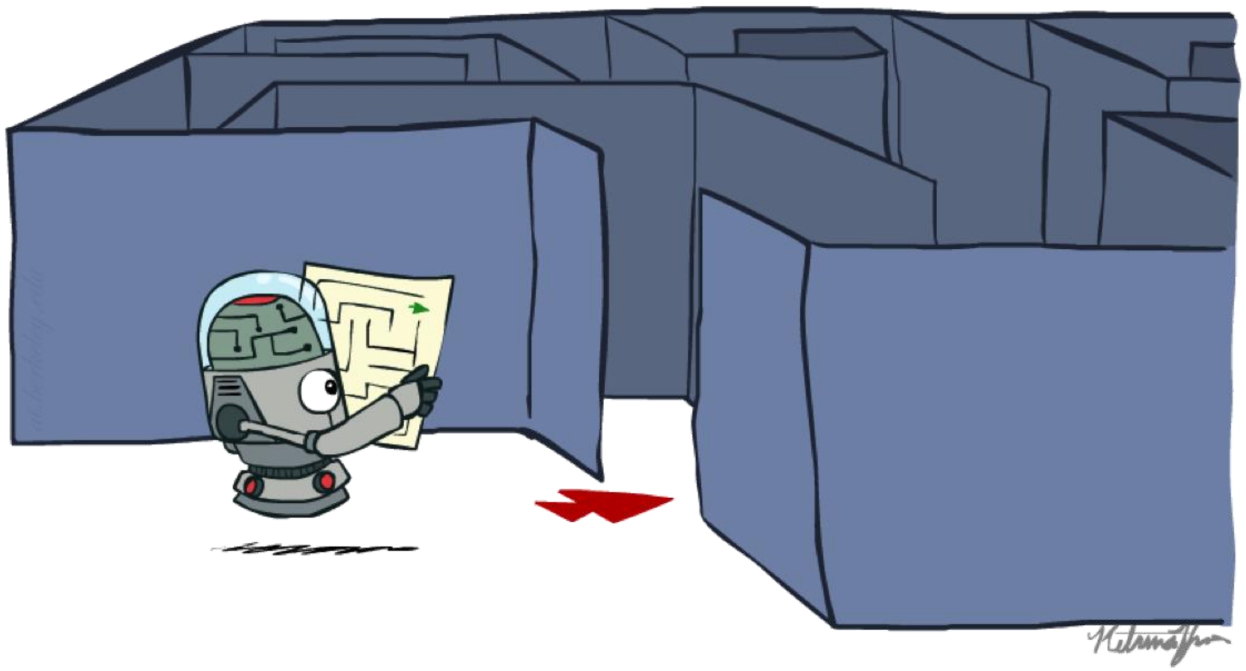
CS 188: Artificial Intelligence

Search



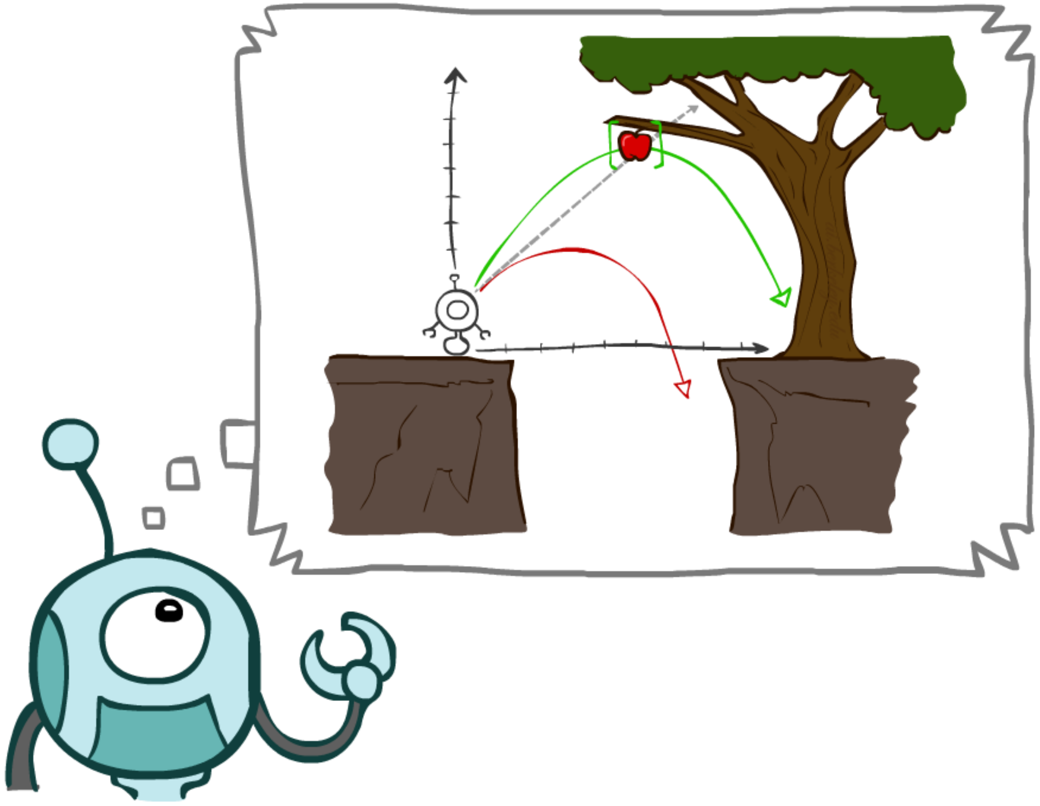
Instructor: Anca Dragan

University of California, Berkeley

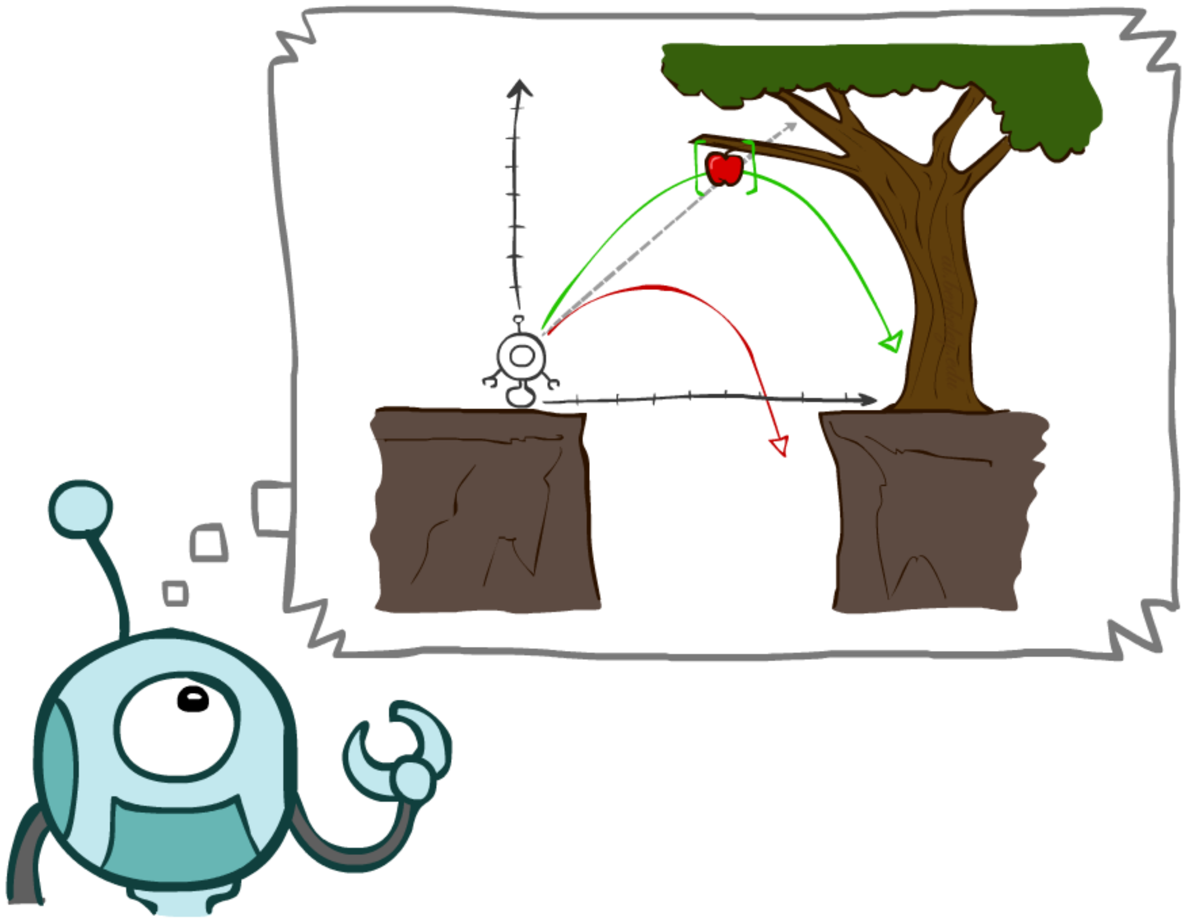
[These slides adapted from Dan Klein and Pieter Abbeel]

Today

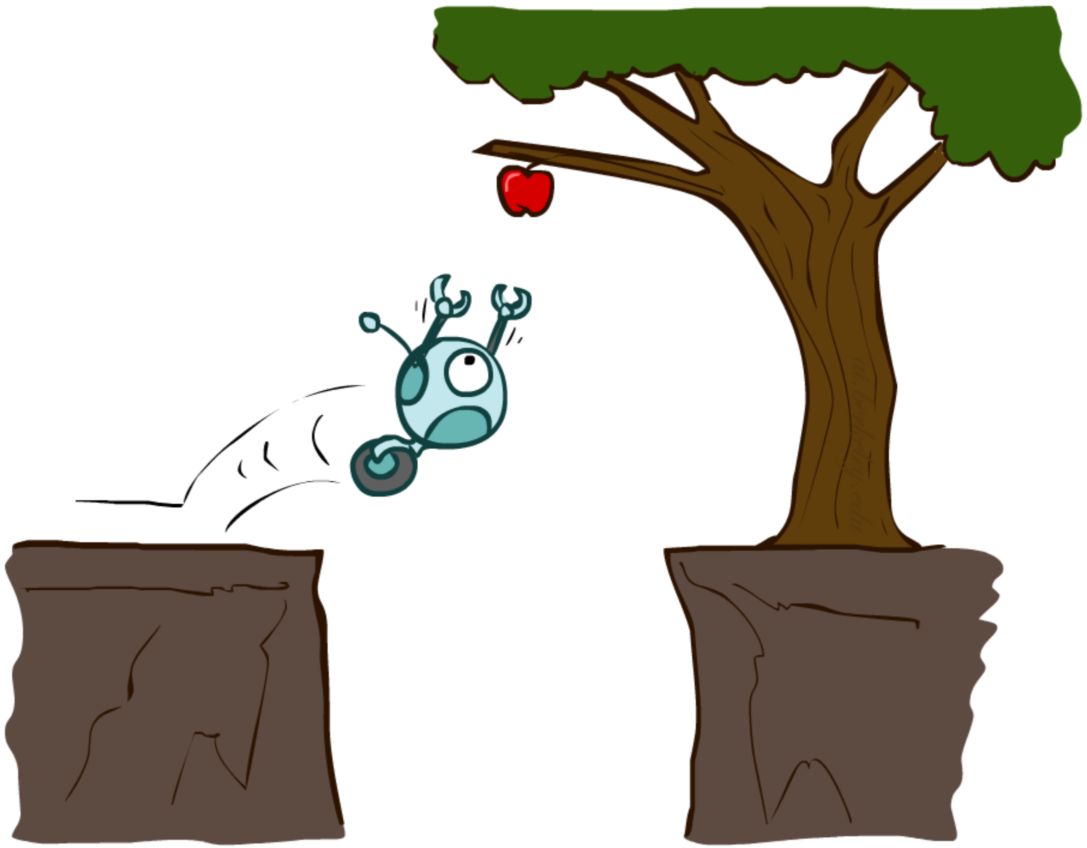
- Agents that Plan Ahead
- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search



Agents that Plan



Reflex Agents

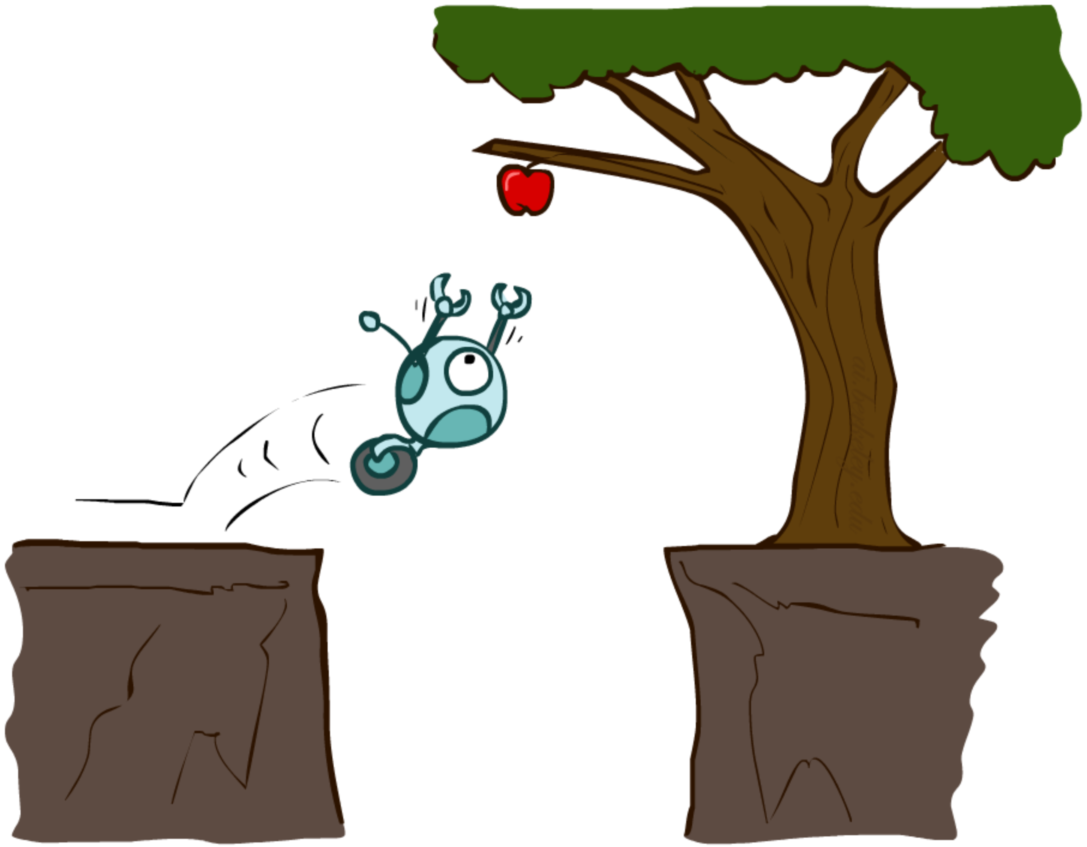


[Demo: reflex optimal (L2D1)]
[Demo: reflex optimal (L2D2)]

Reflex Agents

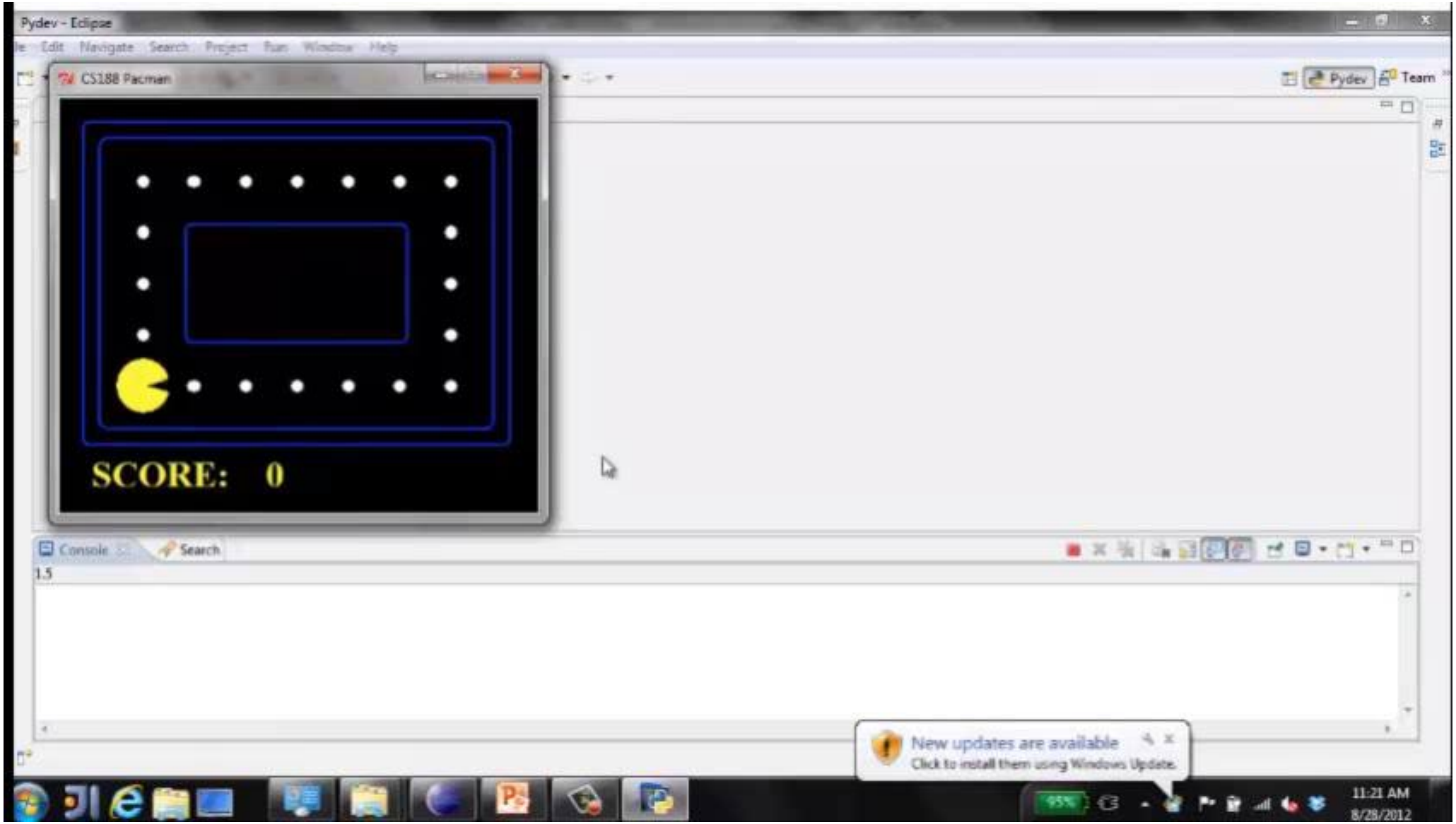
- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - May have memory or a model of the world' s current state
  - **Do not consider** the **future consequences** of their actions
  - Consider how the world IS

- *Can a reflex agent be rational?*

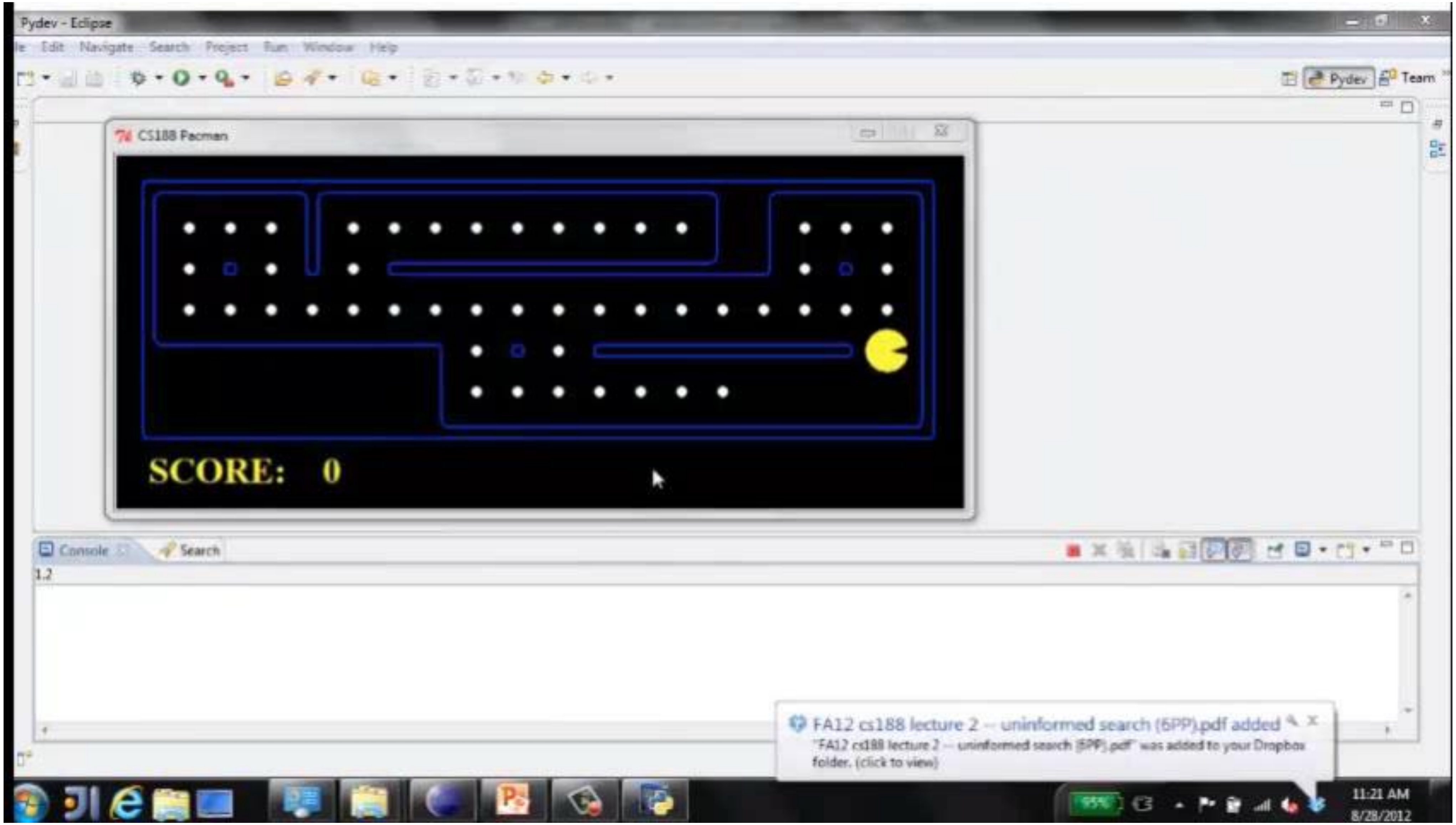


[Demo: reflex optimal (L2D1)]
[Demo: reflex optimal (L2D2)]

Video of Demo Reflex Optimal

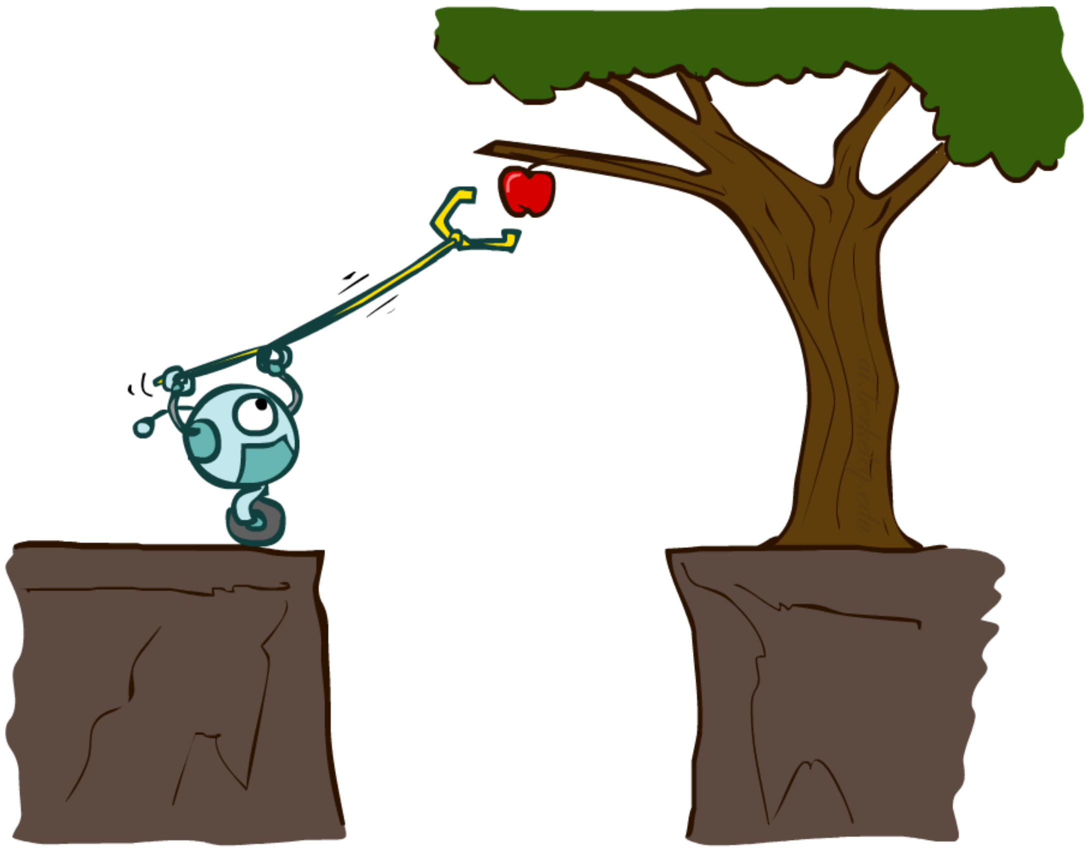


Video of Demo Reflex Odd



Planning Agents

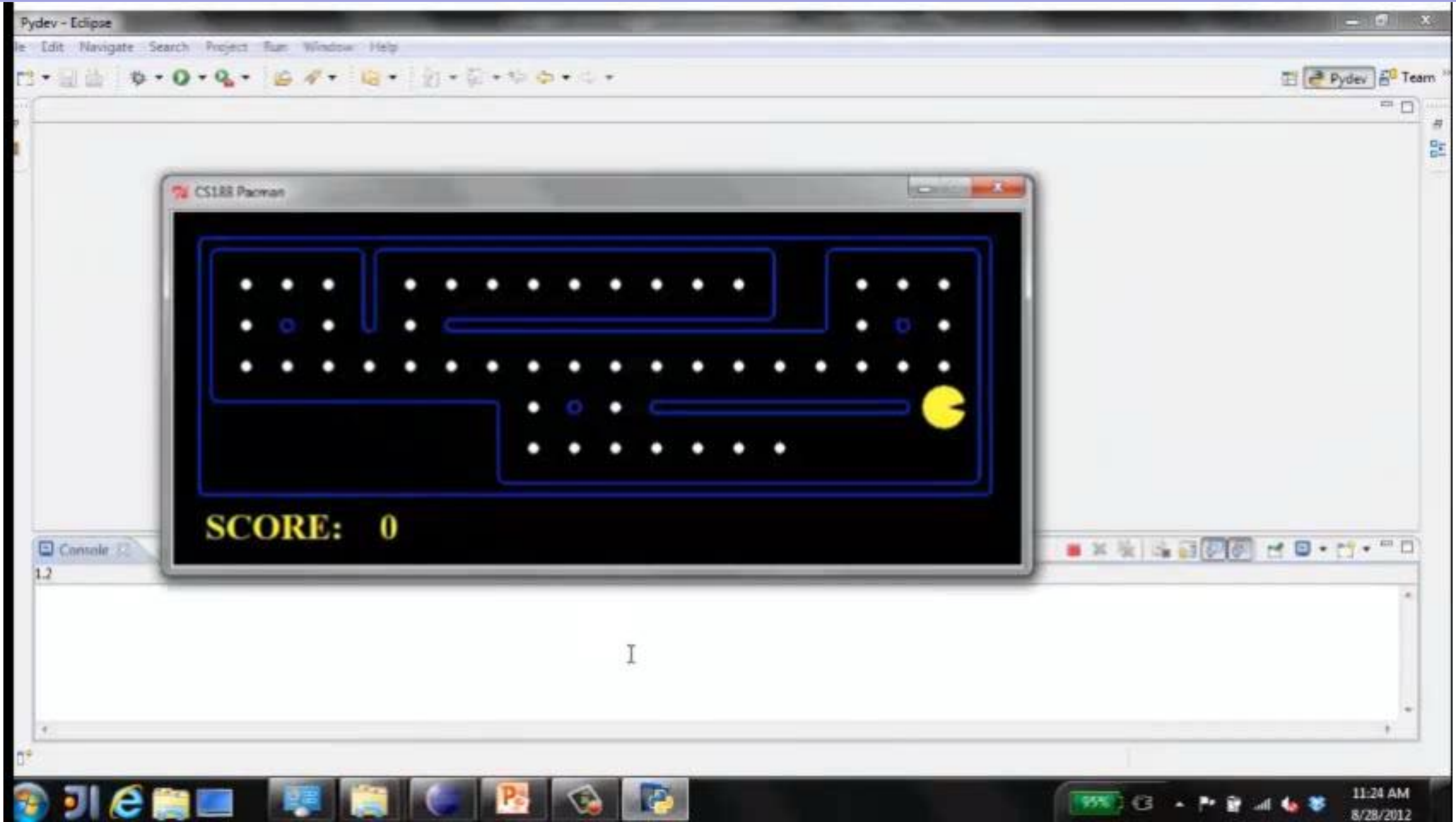
- Planning agents:
  - Ask "what if"
  - Decisions based on (hypothesized) consequences of actions
  - **Must have a model of how the world evolves in response to actions**
  - Must formulate a goal (test)
  - Consider how the world WOULD BE
- Optimal vs. complete planning
- Planning vs. Replanning>>Replanning(Advans: may get the optimal solution;Disadvans: need some time to do the replanning)



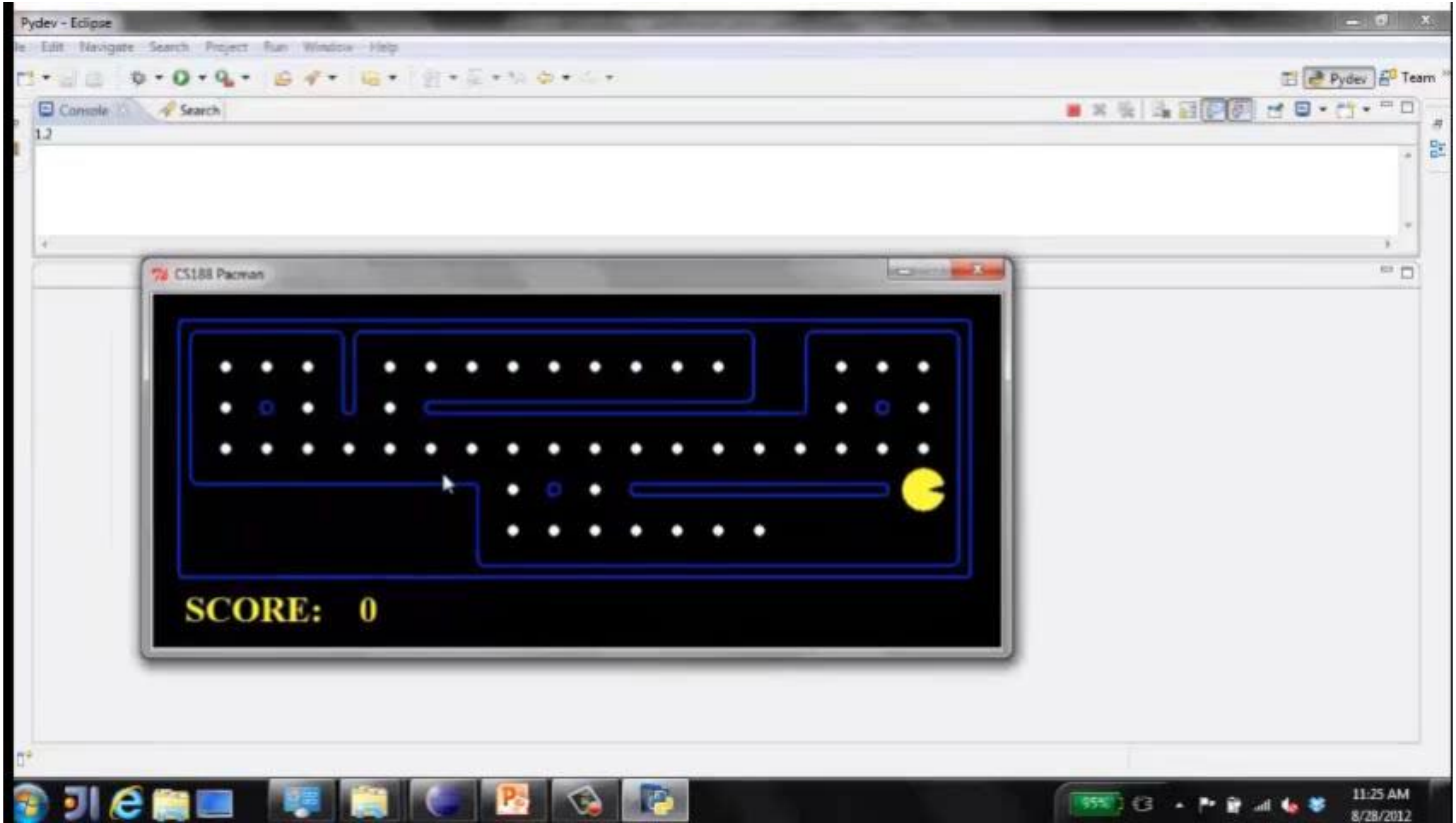
[Demo: re-planning (L2D3)]

[Demo: mastermind (L2D4)]

Video of Demo Replanning



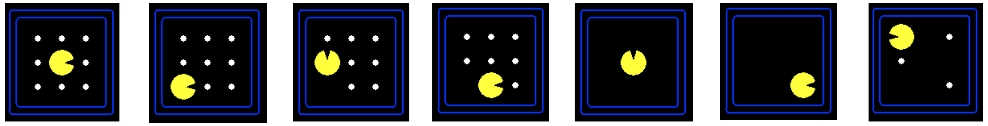
Video of Demo Mastermind



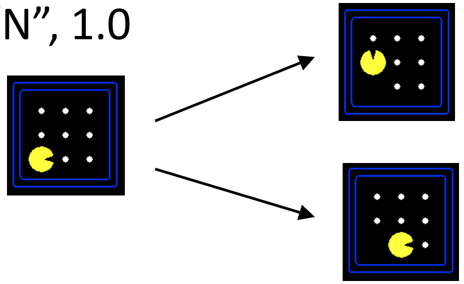
Search Problems

Search Problems

- A search problem consists of:
  - A state space



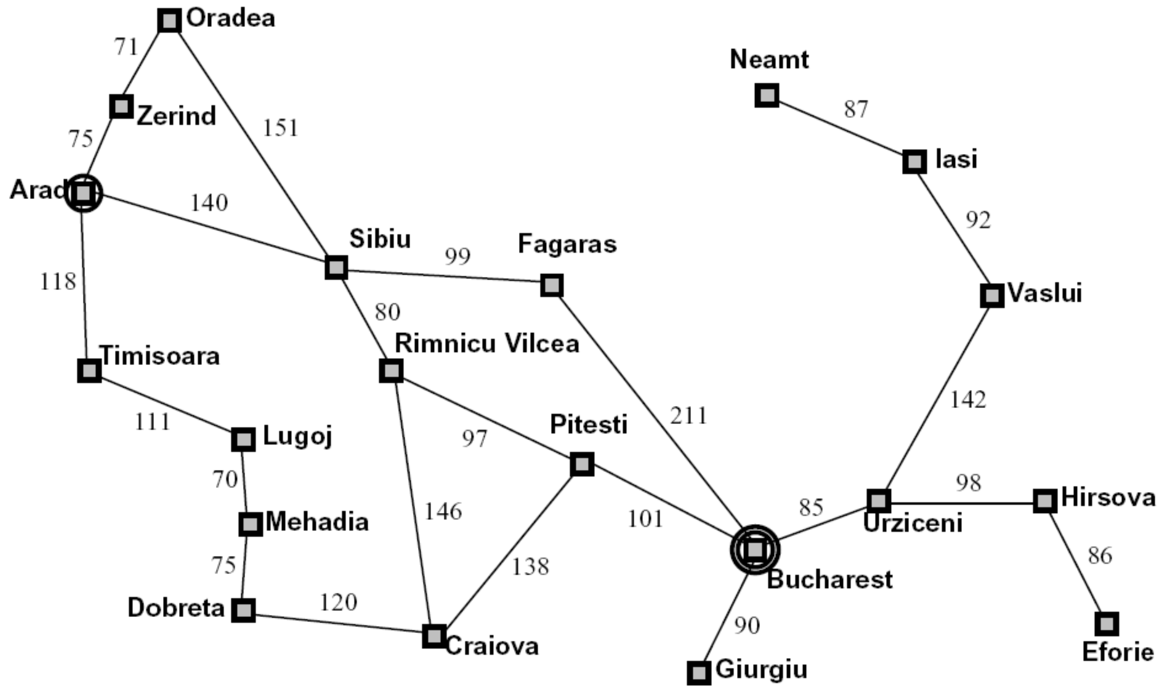
  - A successor function
    (with actions, costs)
  - A start state and a goal test
  (or called end point|| terminating state)



- A solution is a sequence of actions (a plan) which transforms the start state to a goal state， 也就是一条path

Search Problems Are Models

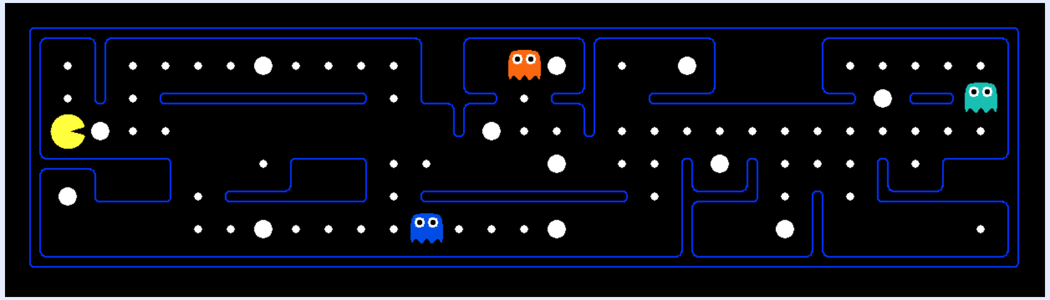
Example: Traveling in Romania



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

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)

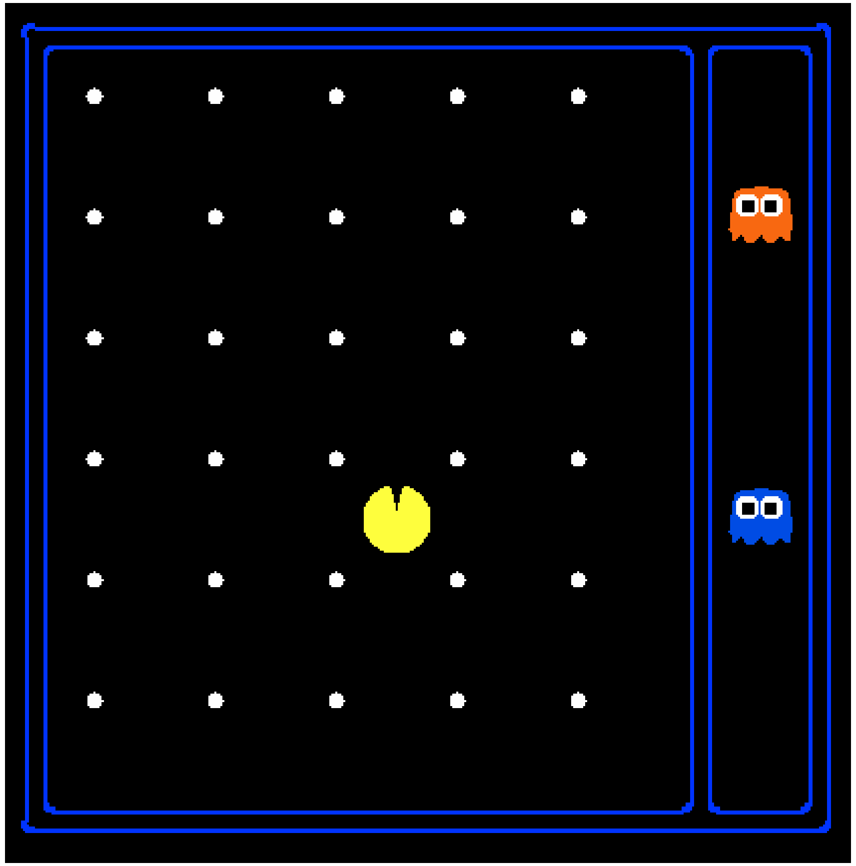
- Problem: Pathing
  - States: (x, y) location
  - Actions:NSEW(North,South ,East,West)
  - Successor: update location only
  - Goal test: is (x,y)=END

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

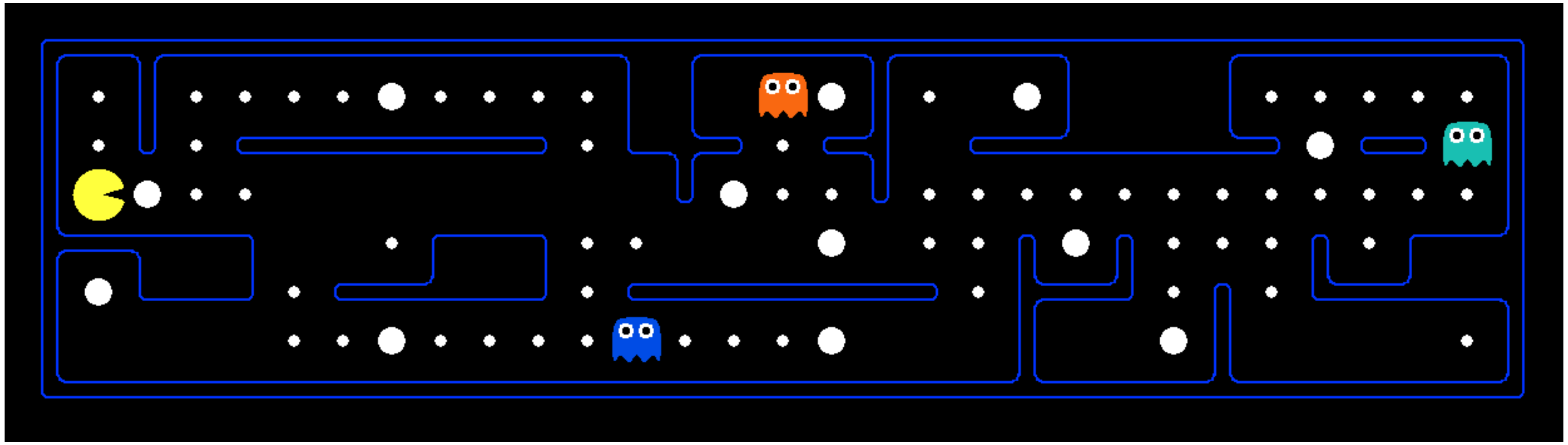
State Space Sizes?

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

- How many
  - World states?
    $120 x (2^{30}) x (12^2) x 4$
  - States for pathing?
    120
  - States for eat-all-dots?
    $120 x (2^{30})$

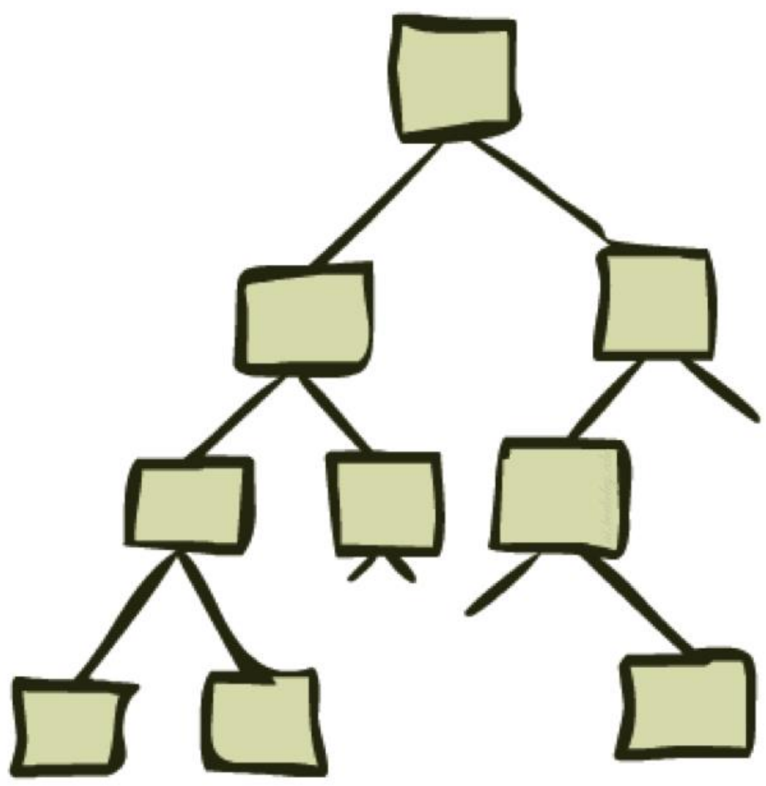


Safe Passage



- 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)-> Not include ghost position, because for this specific program, the information above is enough to fill the game.

State Space Graphs and Search Trees

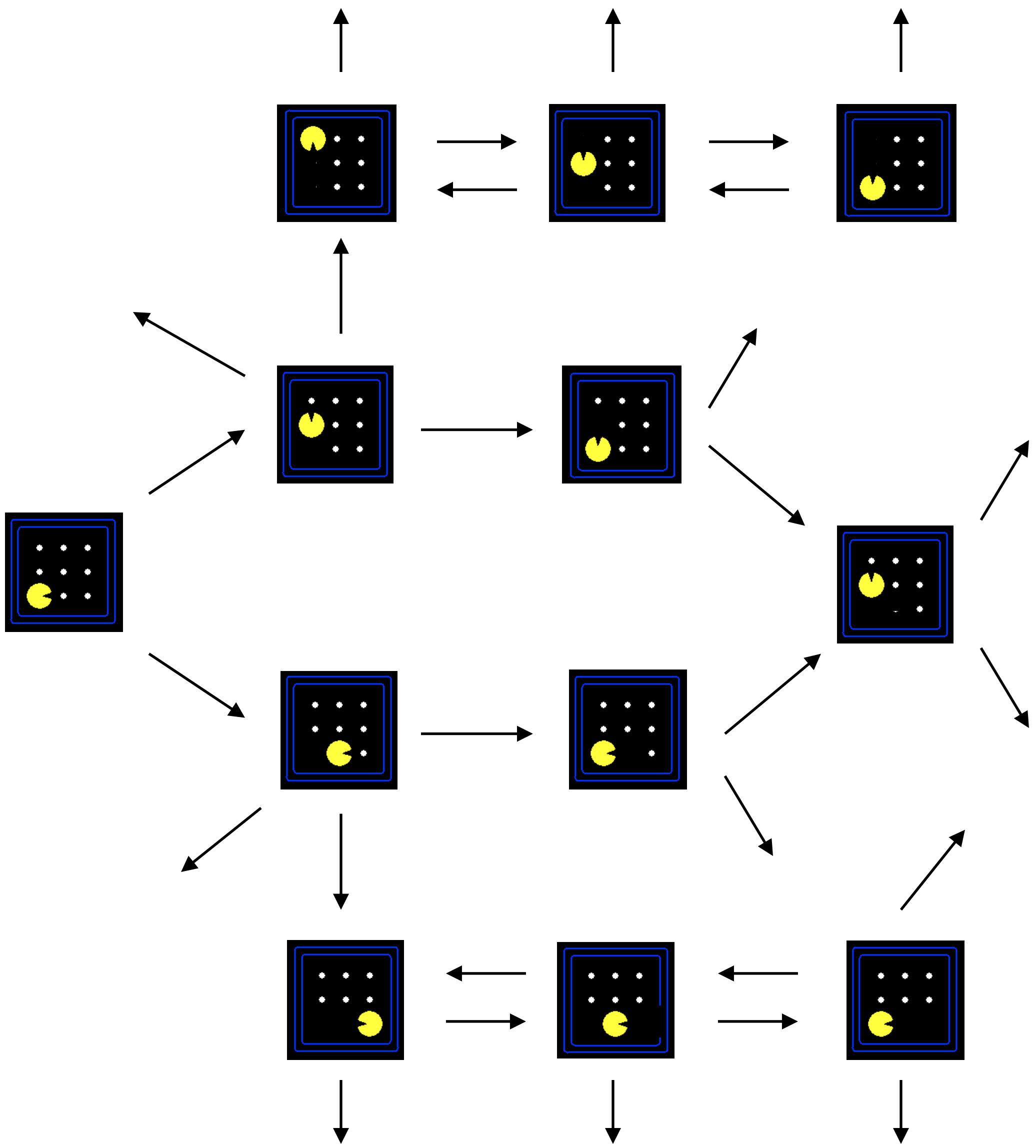


State Space Graphs

- 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

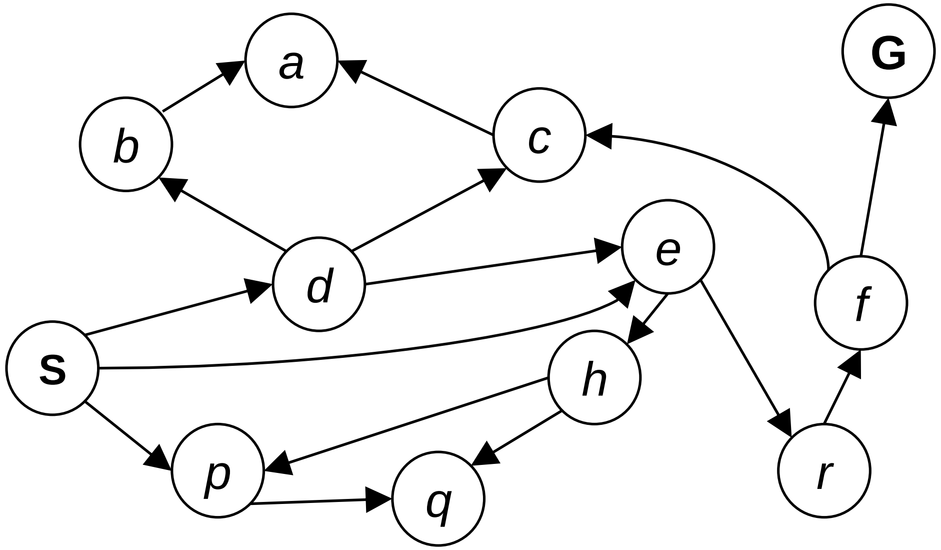


State Space Graphs

- 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)

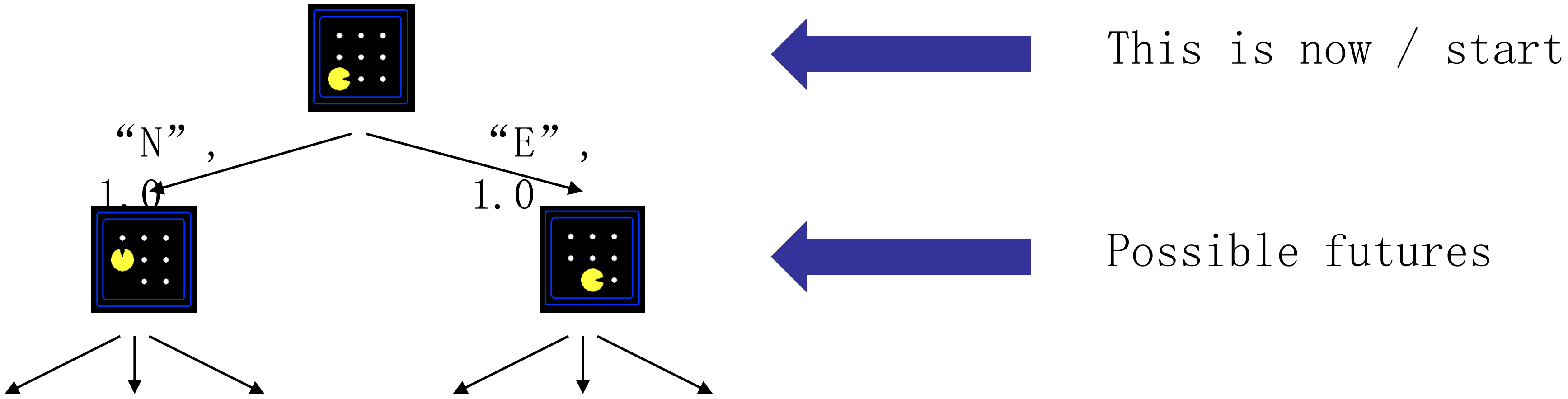
- In a search graph, each state occurs only once!

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



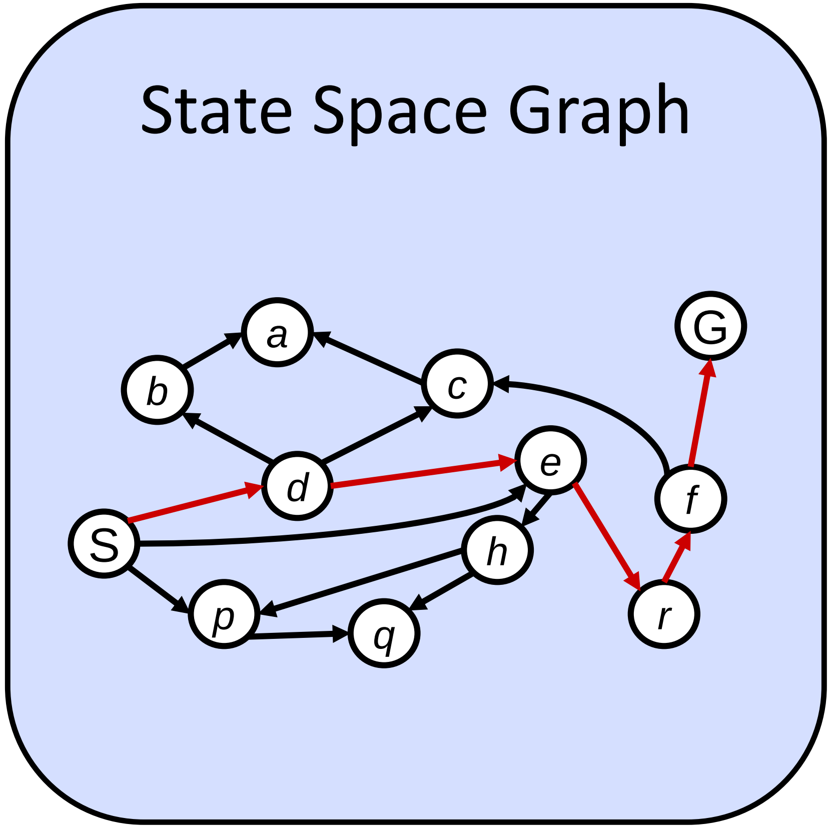
Tiny search graph for a tiny search problem

Search Trees



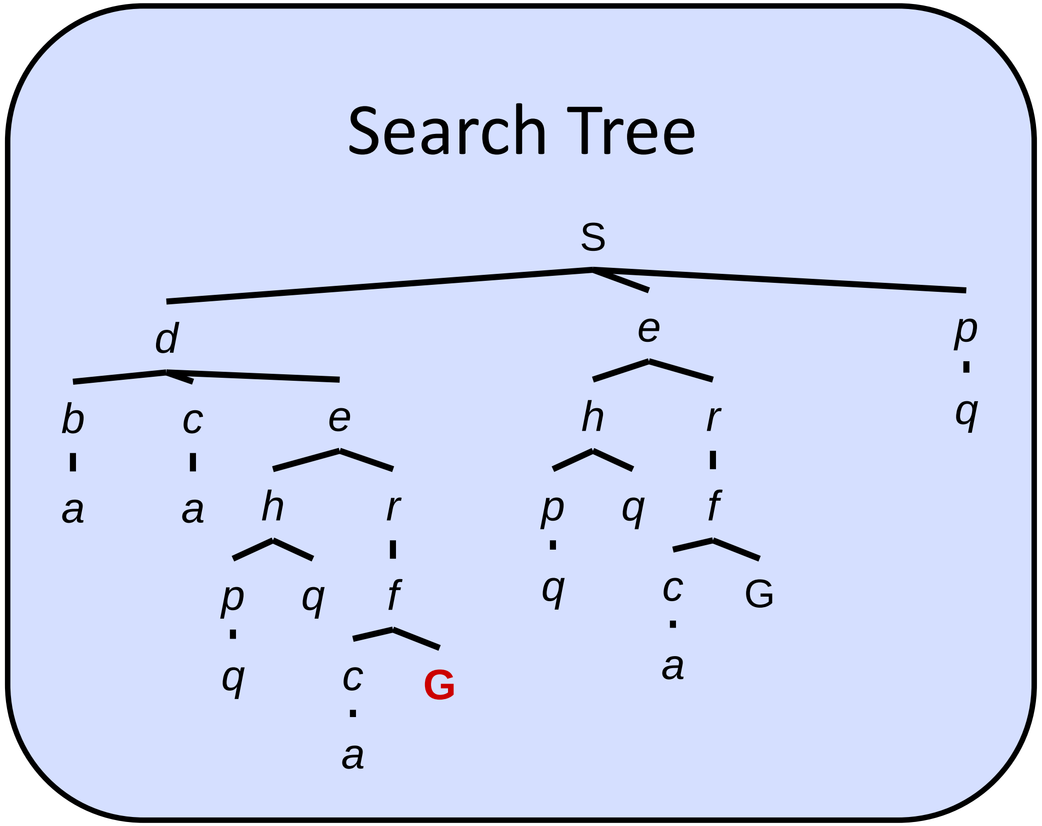
- A search tree:
  - A "what if" tree of plans and their outcomes
  - The start state is the root node
  - Children correspond to successors
  - Nodes show states, but correspond to PLANS that achieve those states
  - For most problems, we can never actually build the whole tree

State Space Graphs vs. Search Trees



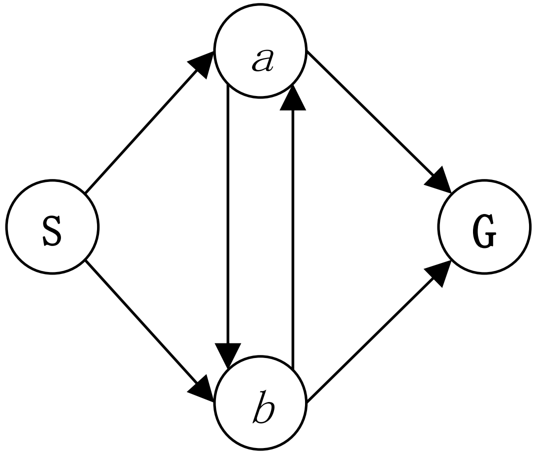
Each NODE in in the search tree is an entire PATH in the state space graph.

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

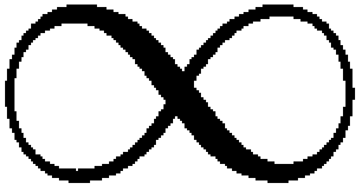


State Space Graphs vs. Search Trees

Consider this 4-state graph:

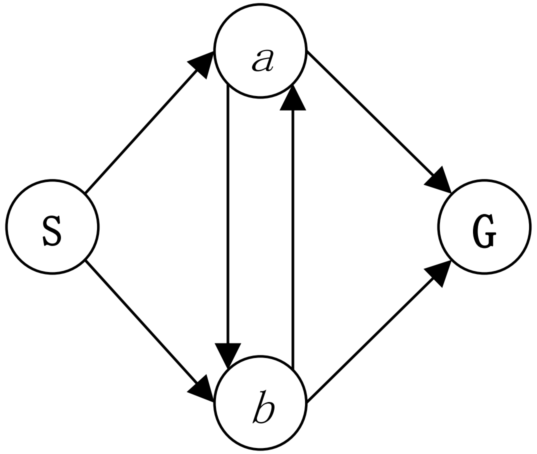


How big is its search tree (from S)?

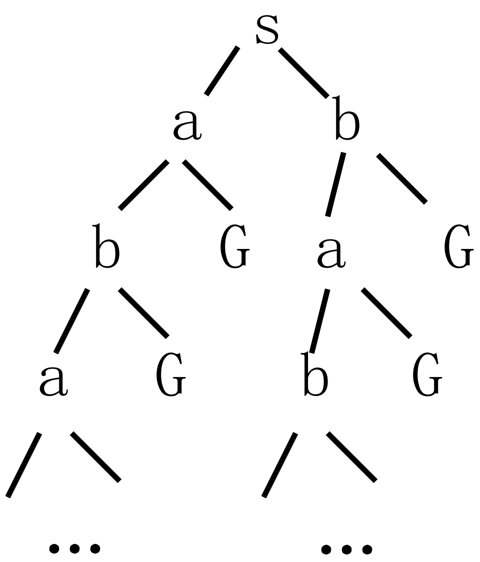


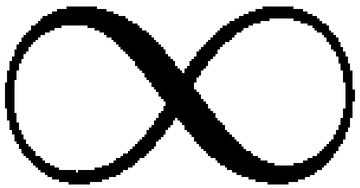
State Space Graphs vs. Search Trees

Consider this 4-state graph:



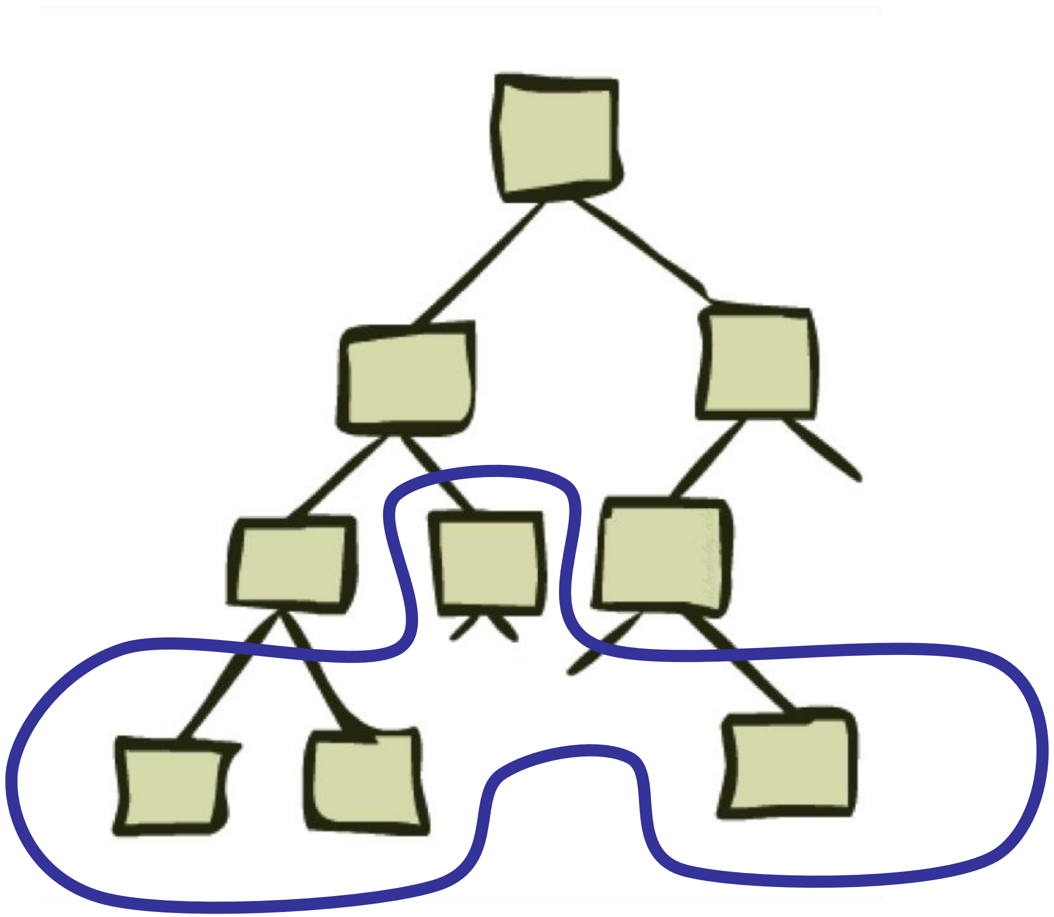
How big is its search tree (from S)?



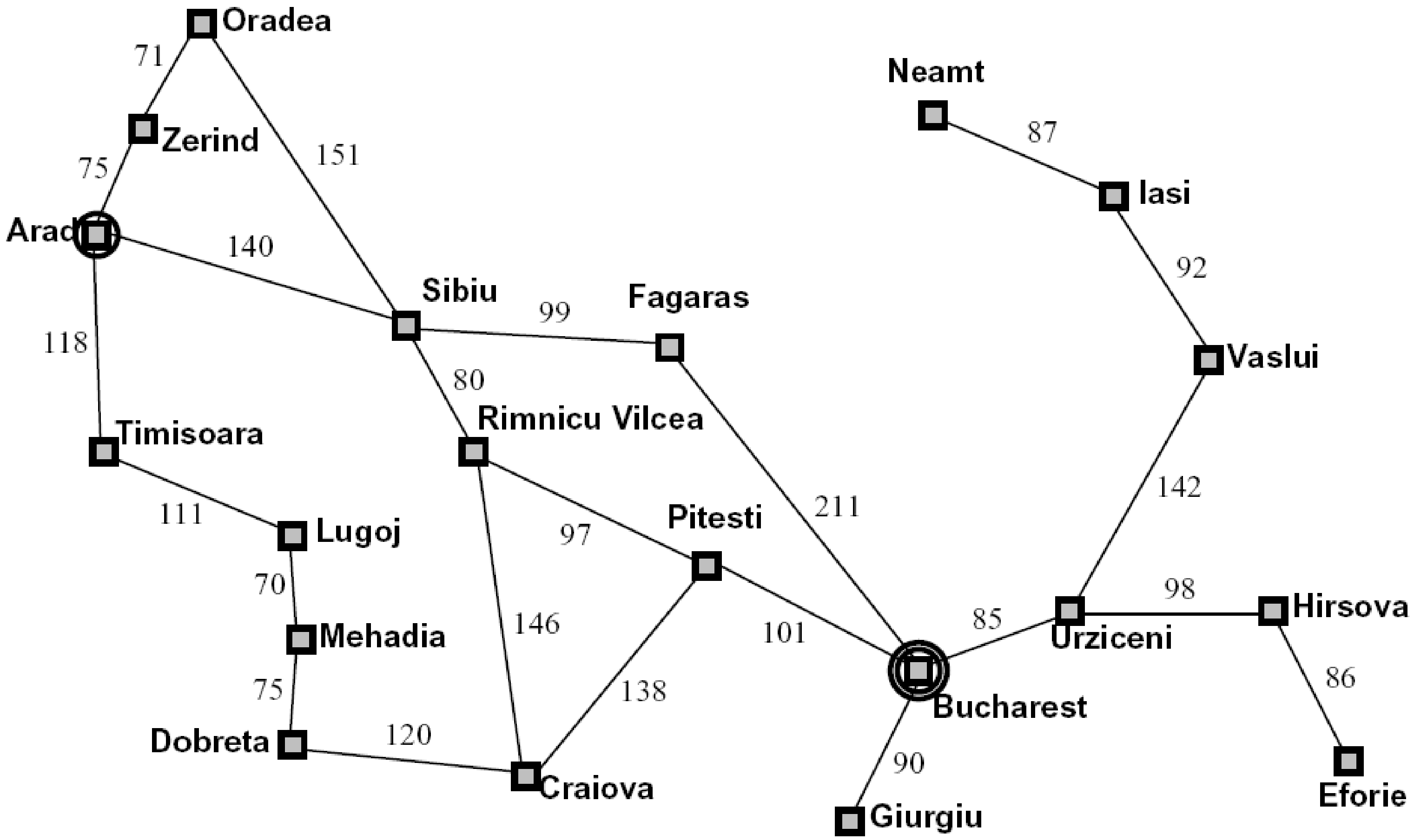


Important: Lots of repeated structure in the search tree!

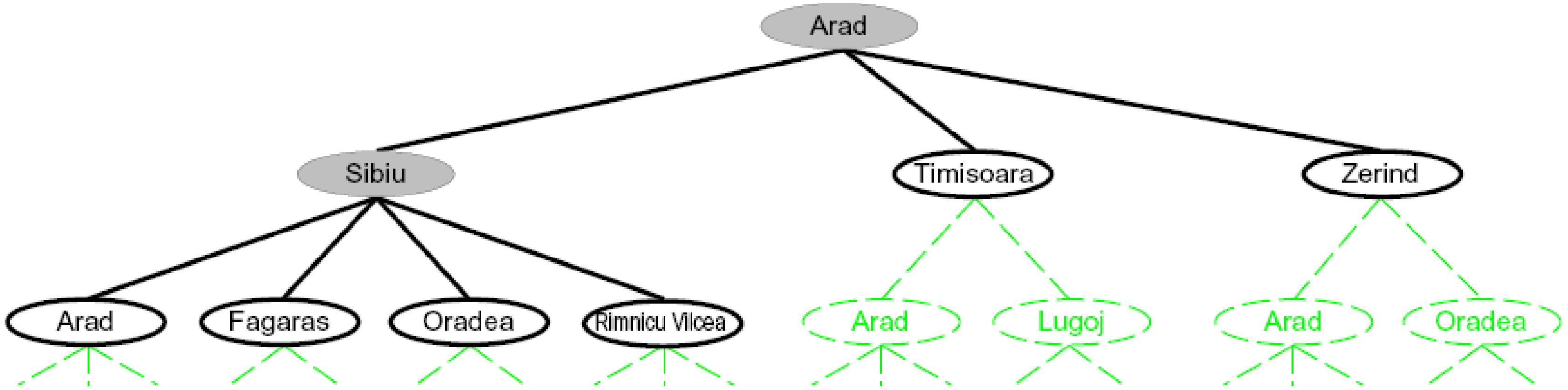
Tree Search



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

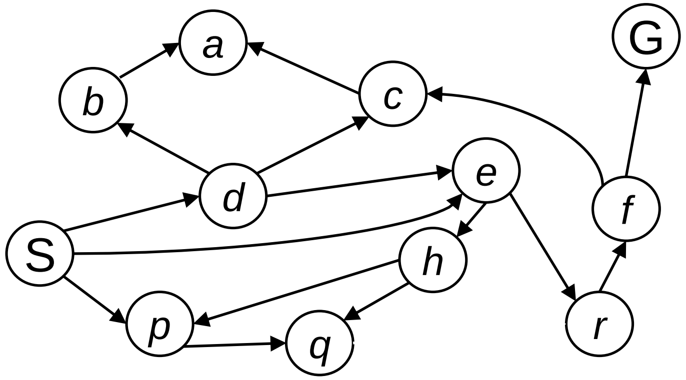
General Tree Search

```
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
```

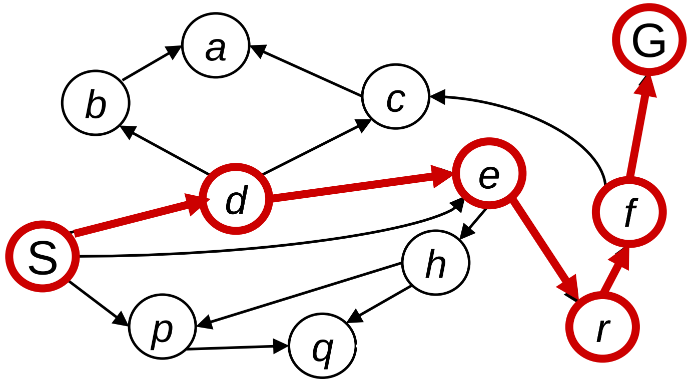
- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy

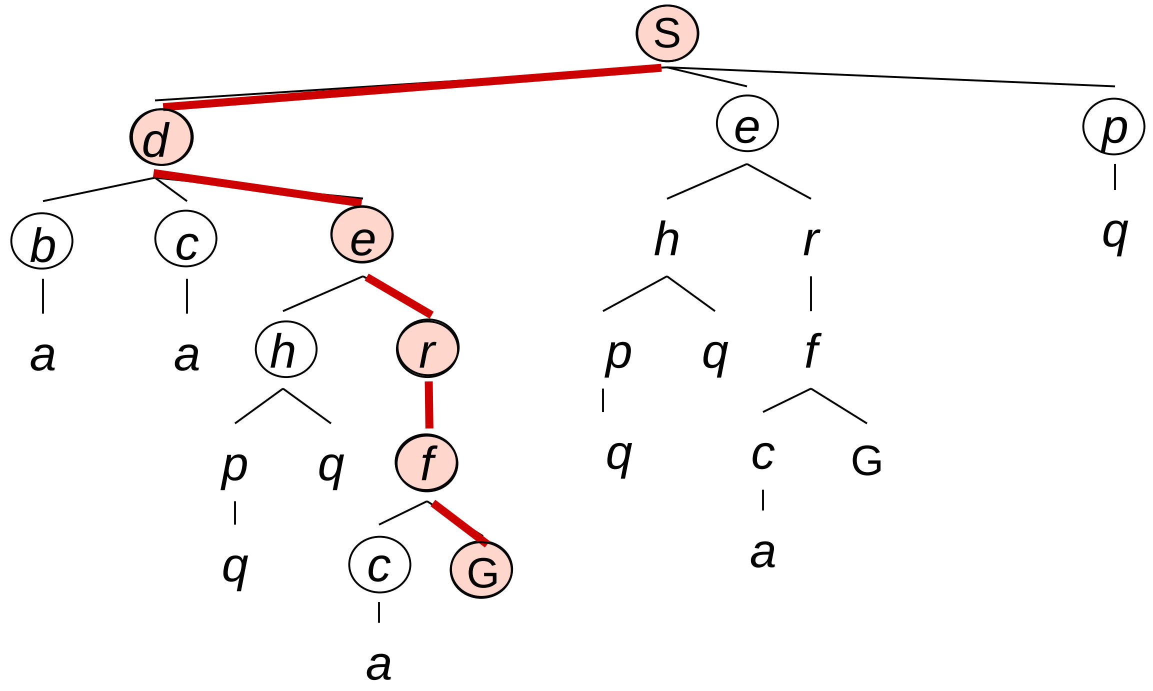
- Main question: which fringe nodes to explore?

Example: Tree Search



Example: Tree Search





~~s~~

~~s → d~~

s → e

s → p

s → d → b

s → d → c

~~s → d → e~~

s → d → e → h

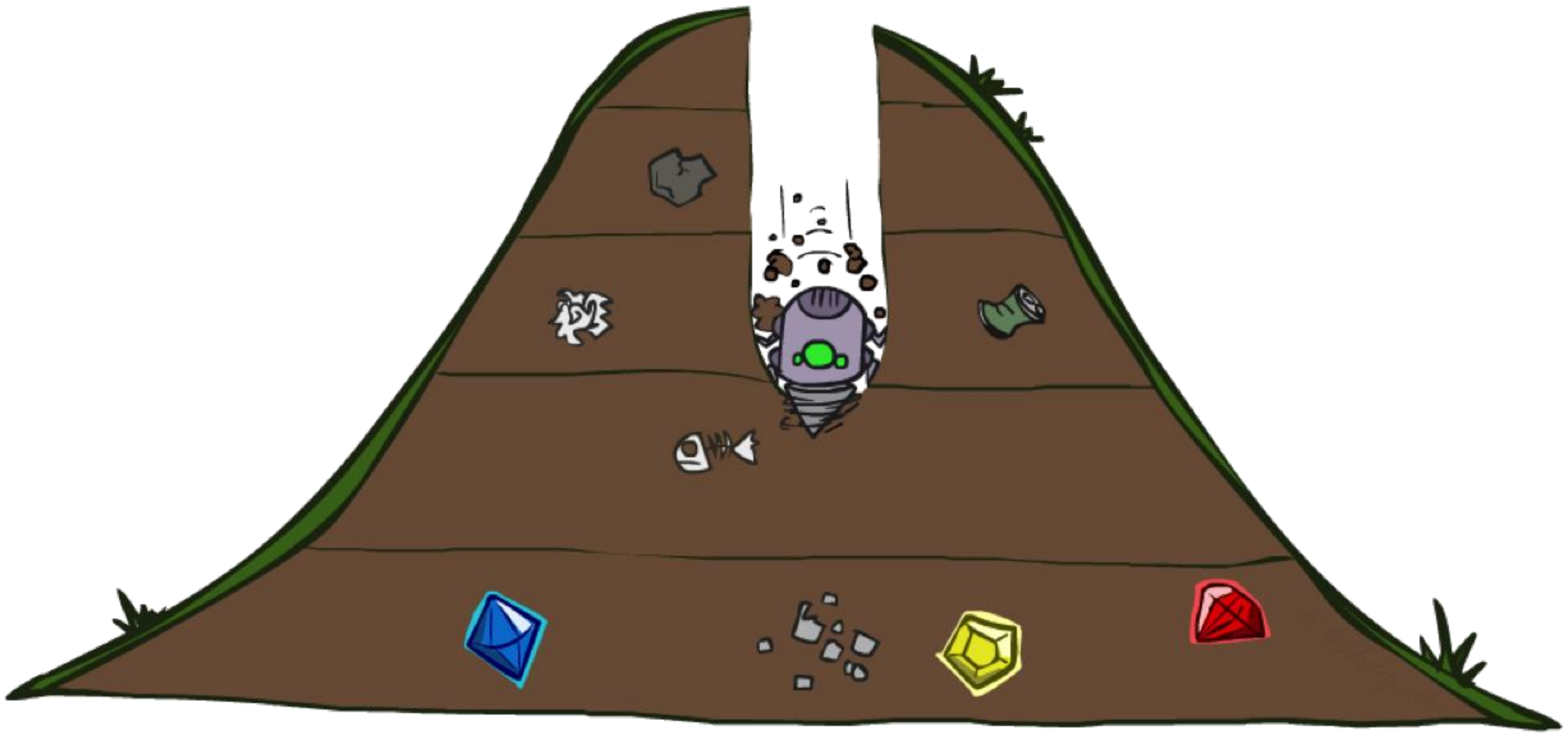
~~s → d → e → r~~

~~s → d → e → r → f~~

s → d → e → r → f → c

~~s → d → e → r → f → G~~

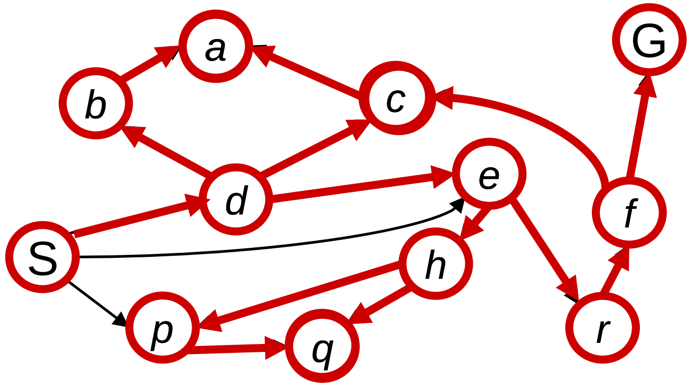
Depth-First Search

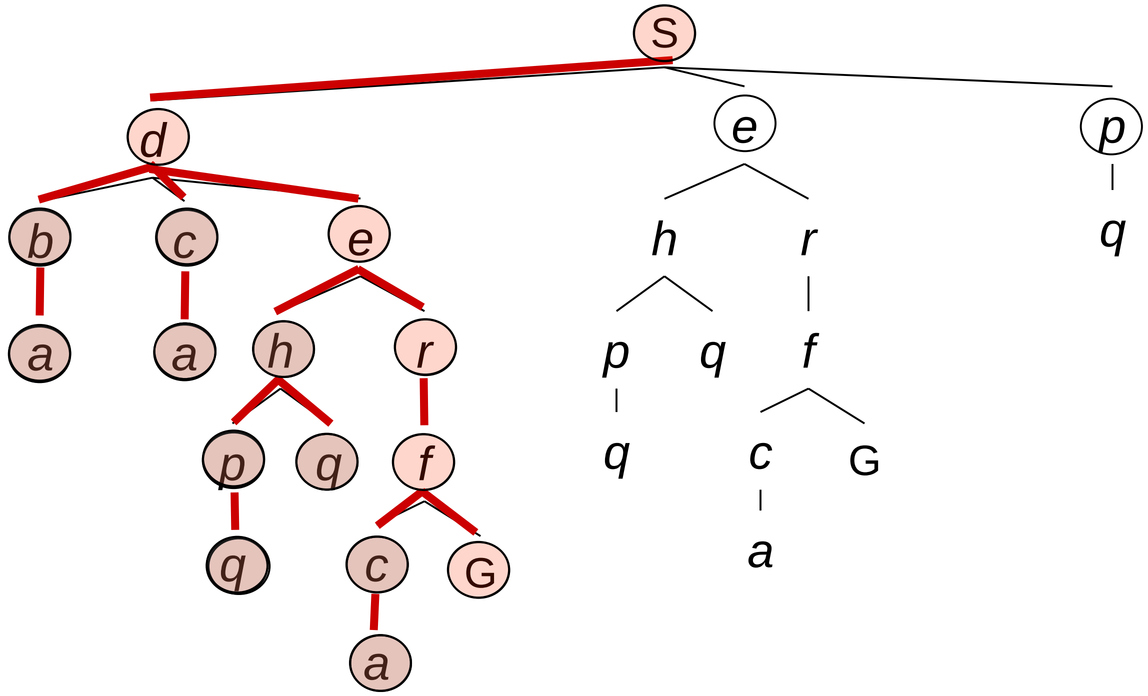


Depth-First Search

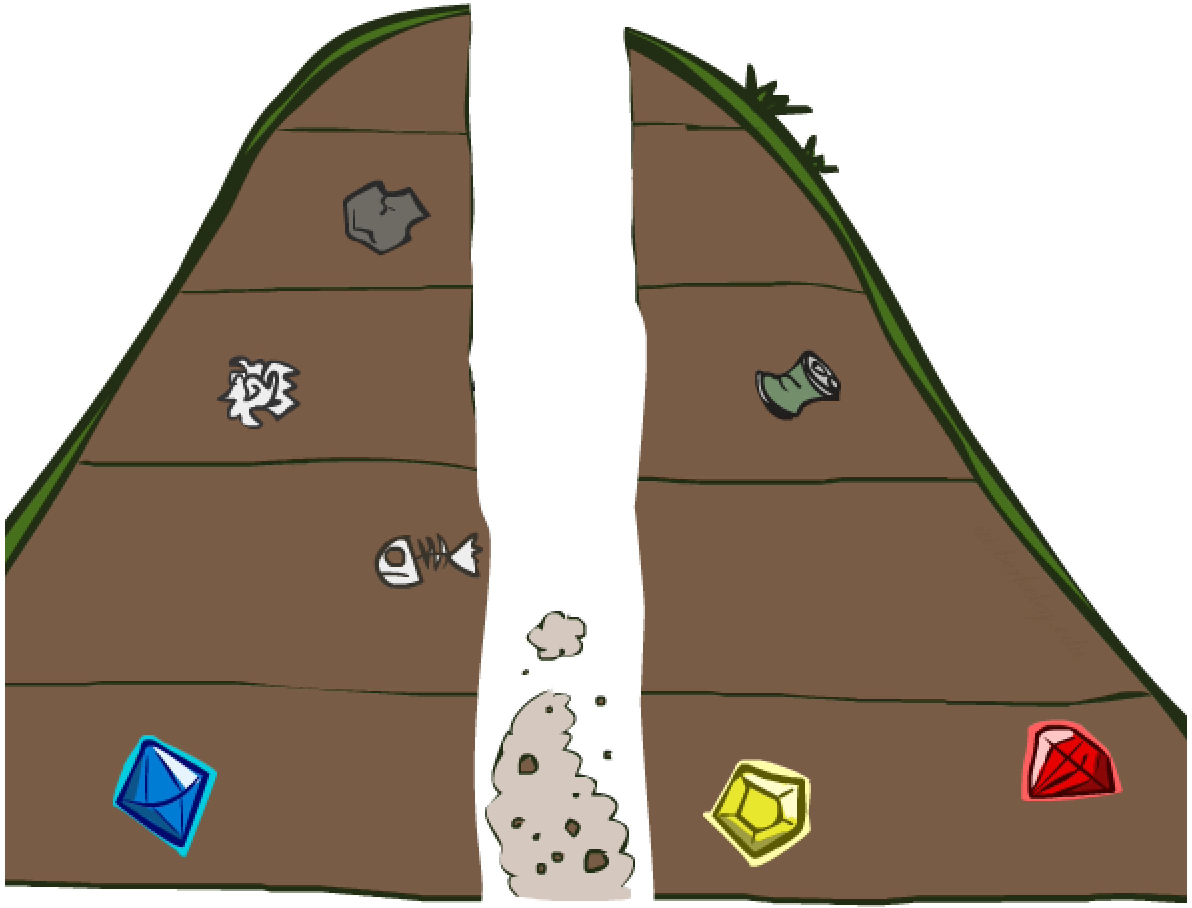
Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack





Search Algorithm Properties

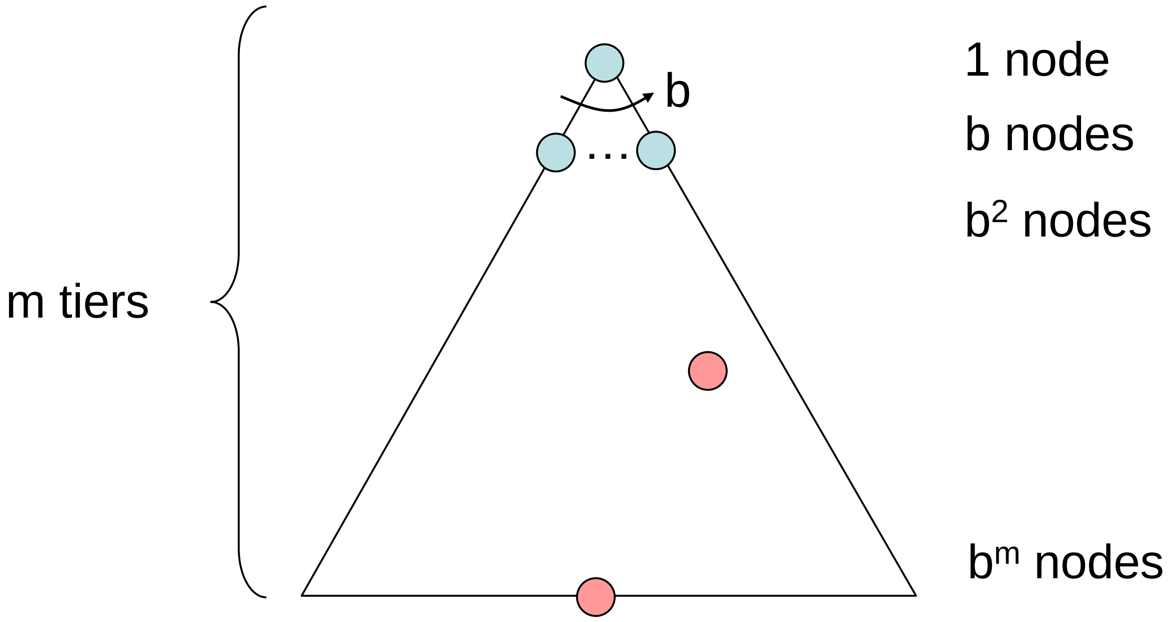


Search Algorithm Properties

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

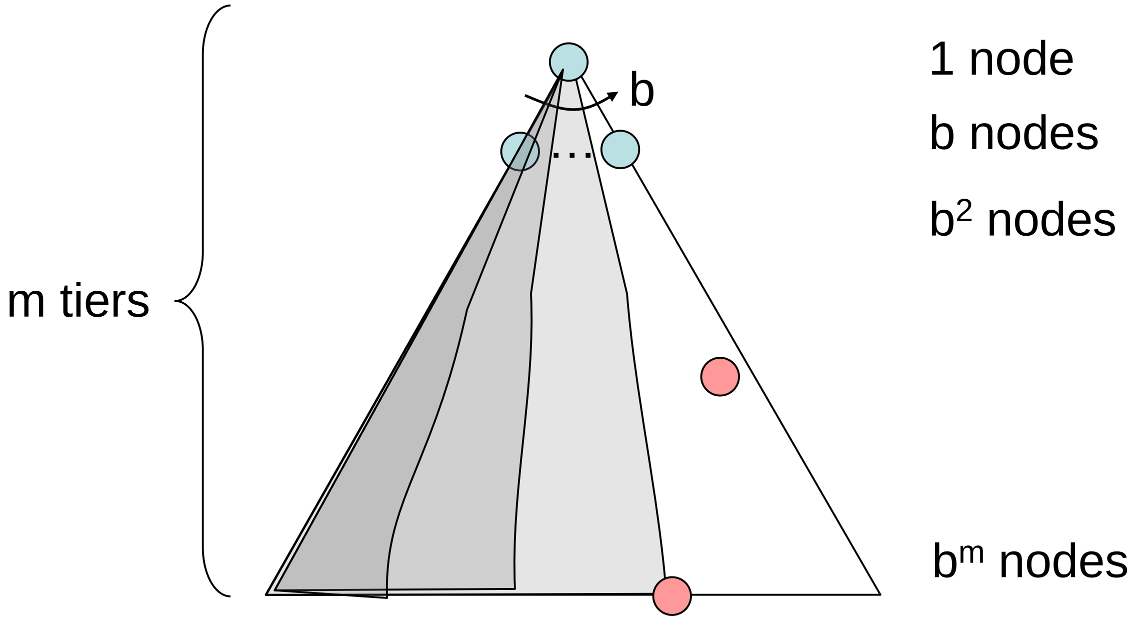
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths

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

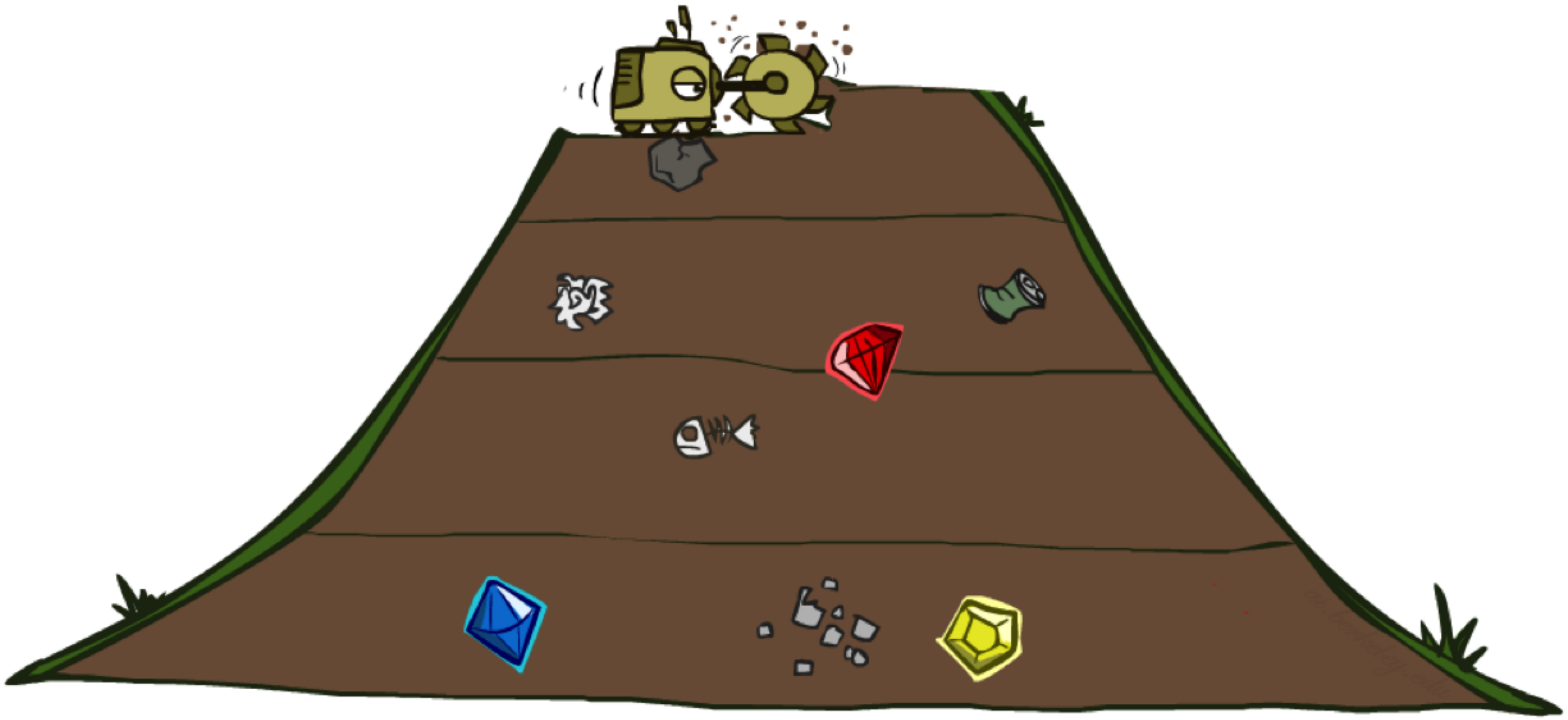


Depth-First Search (DFS) Properties

- 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(make search tree grows infinitely)** (more later)
- Is it optimal?
  - No, it finds the “leftmost” solution, regardless of depth or cost



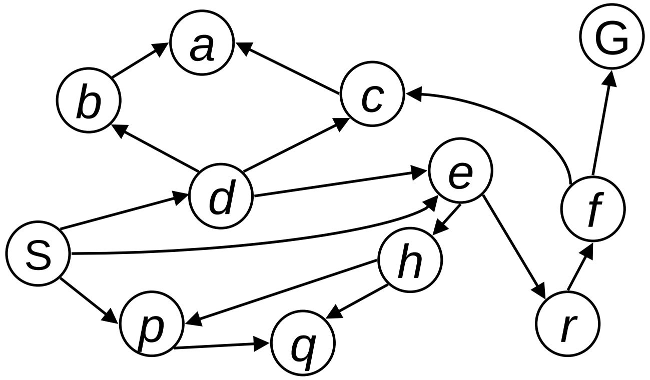
Breadth-First Search

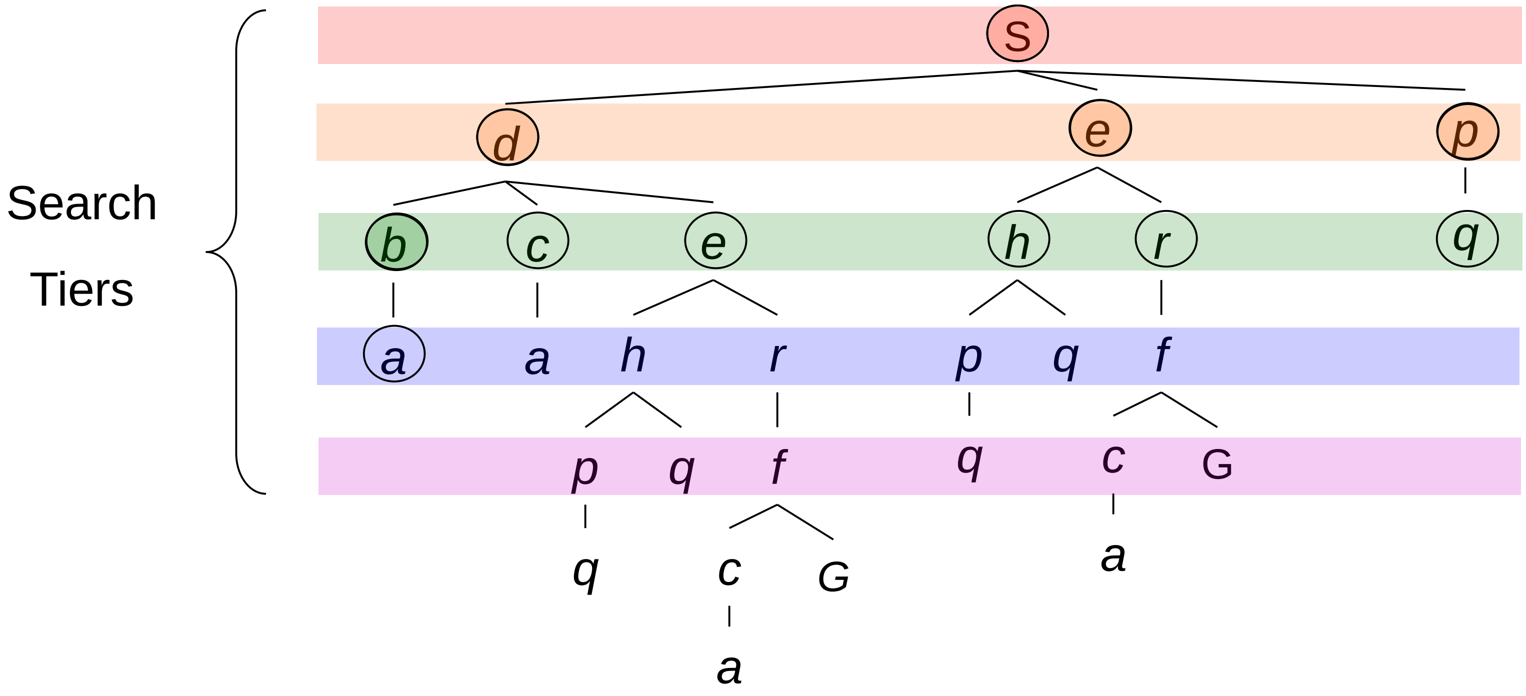


Breadth-First Search

Strategy: expand a shallowest node first

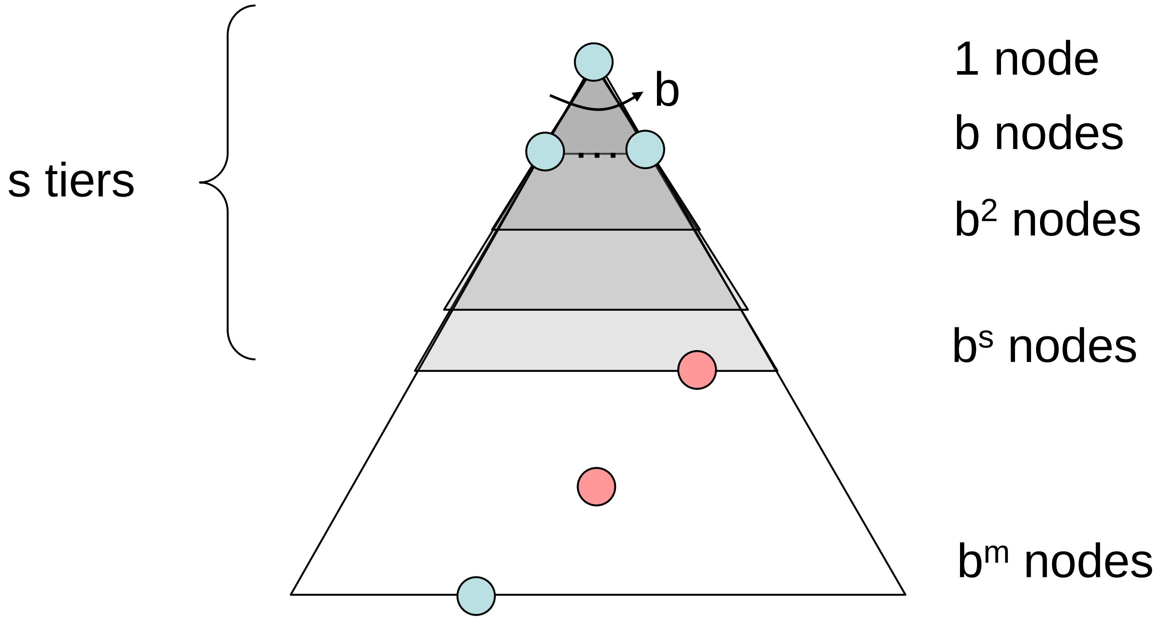
Implementation: Fringe is a FIFO queue



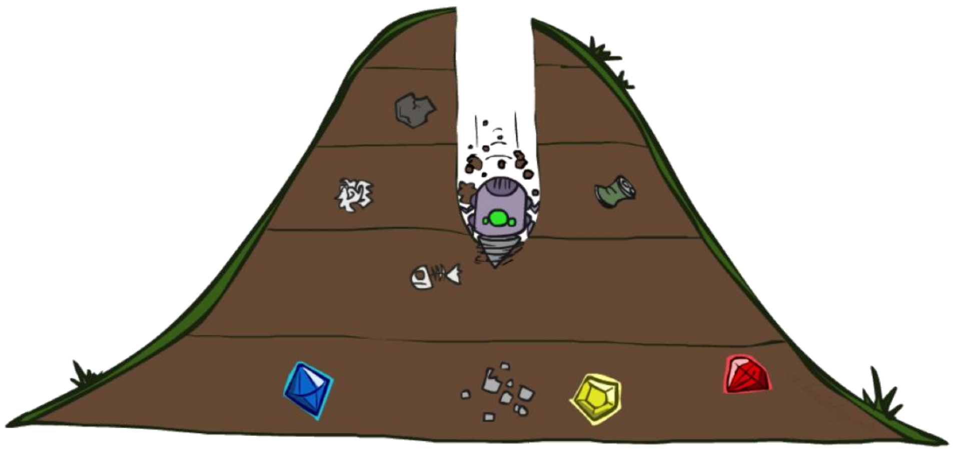


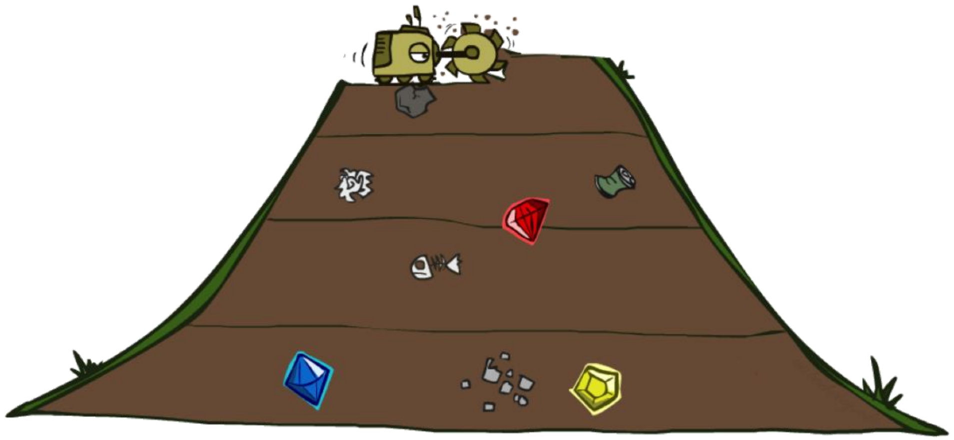
Breadth-First Search (BFS) Properties

- 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
- Is it optimal?
  - Only if costs are all 1 (more on costs later), if the cost are positive values, then we use Dijkstra Algorithm. If the cost have negative values, we use Bellman-Ford Algorithm. If we want to find all pair shortest path, we use Floyed-Warshall Algorithm or Johnson Algorithm.



Quiz: DFS vs BFS



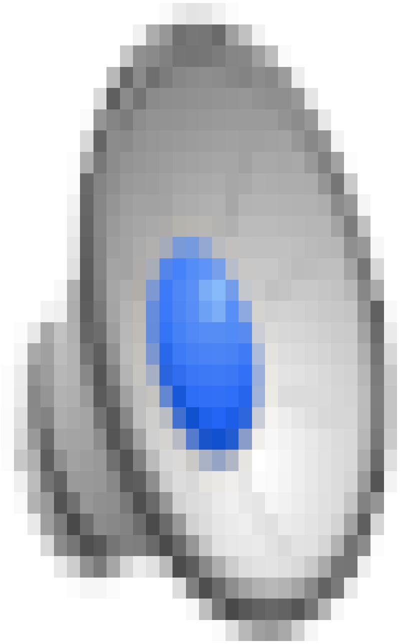


DFS vs BFS

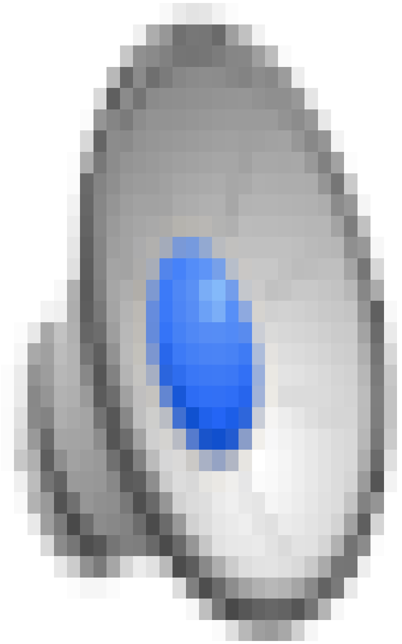
- When will BFS outperform DFS?
- -> When the goal test is on the right hand side of the state space tree and on the shallow layer of the tree.
- When will DFS outperform BFS?
- -> When the goal test is on the left hand side and deep layer of the tree.

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

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

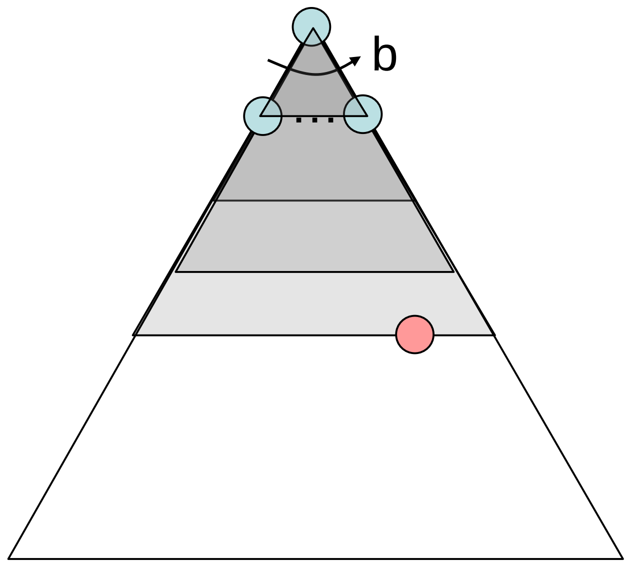


Video of Demo Maze Water DFS/BFS (part 2)

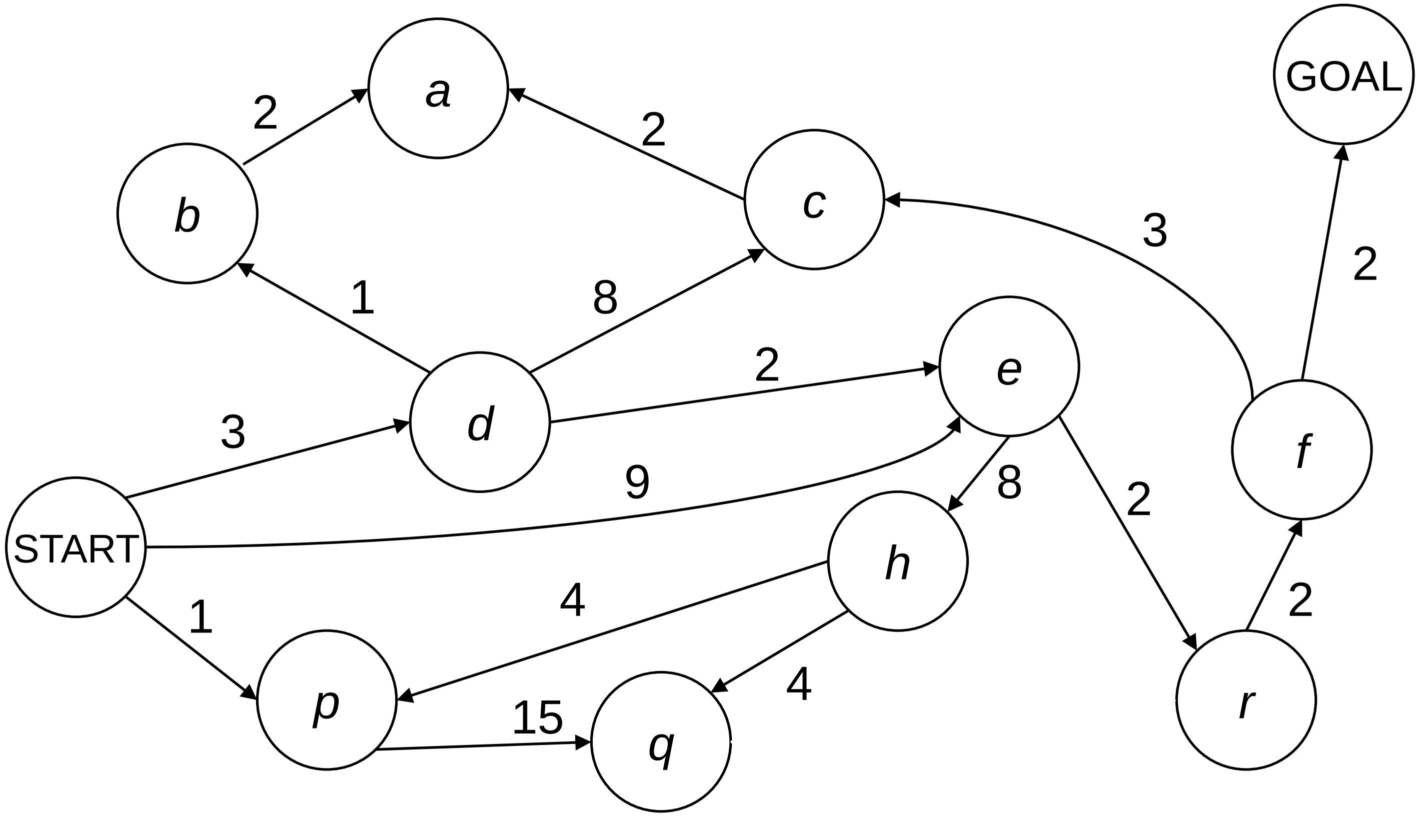


Iterative Deepening(迭代加深算法)

- 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. …..
  - Note that limit can be set by user.
- Isn’ t that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!



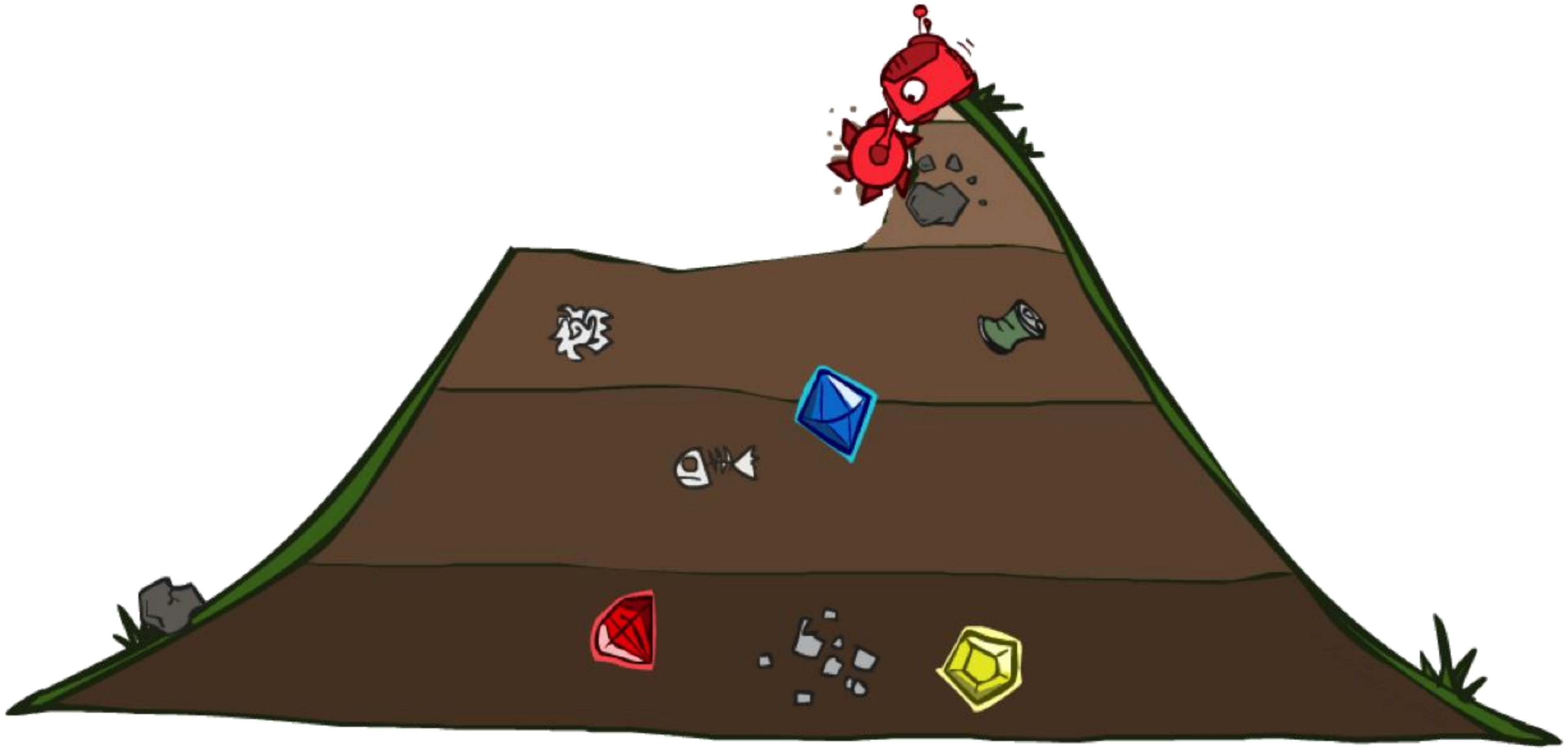
Cost-Sensitive Search



How?->Dijkstra Algorithm

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.

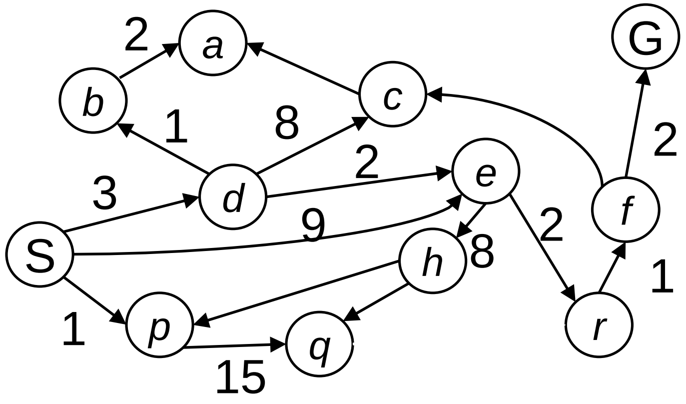
Uniform Cost Search

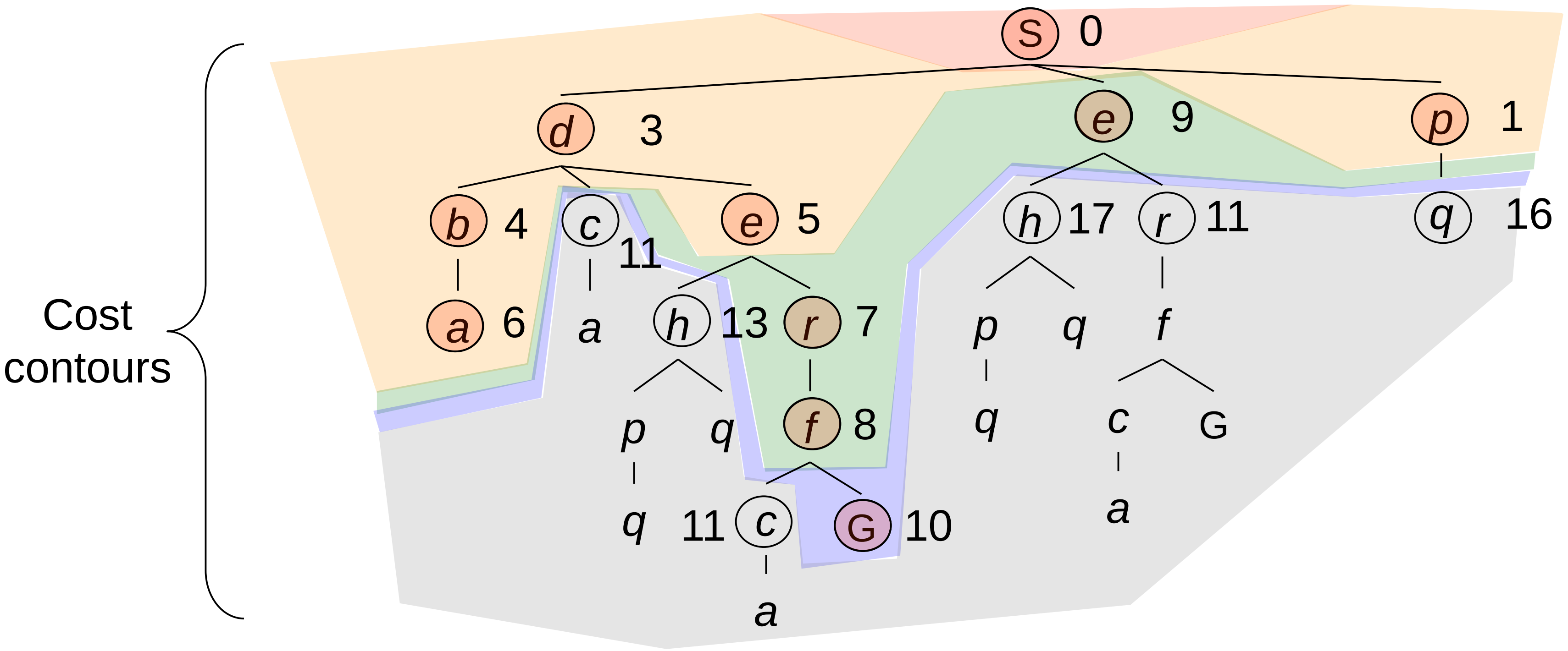


Uniform Cost Search

Strategy: expand a cheapest node first:

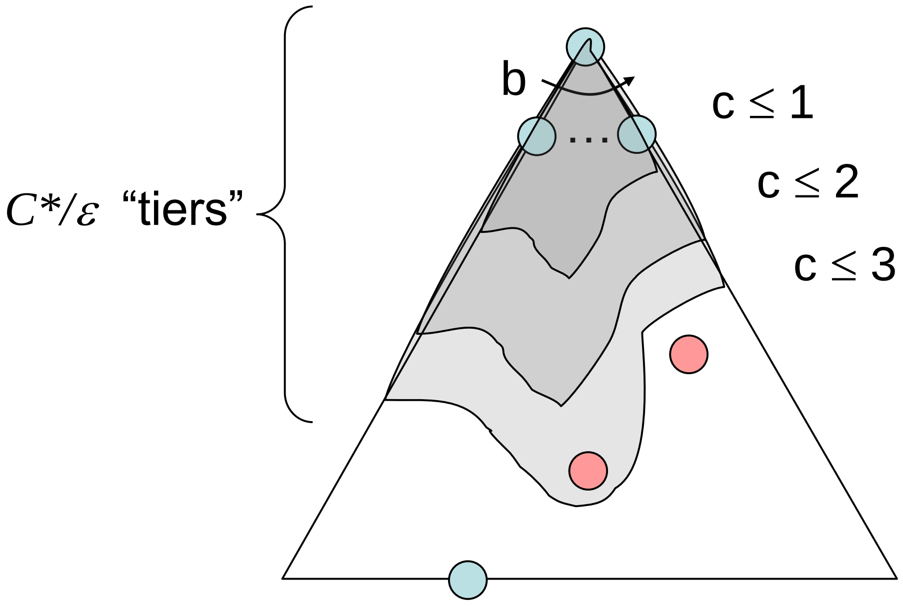
*Fringe is a priority queue (**priority: cumulative cost**)*





Uniform Cost Search (UCS) Properties

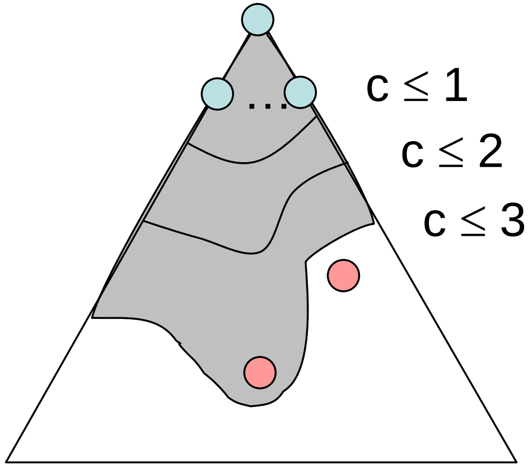
- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "**effective depth**" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- How much space does the fringe 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(For the fact that this UCS strategy is a greedy strategy, thus the current optimal must lead to global optimal), yes!
- Is it optimal?
  - Yes! (Proof next lecture via A*)

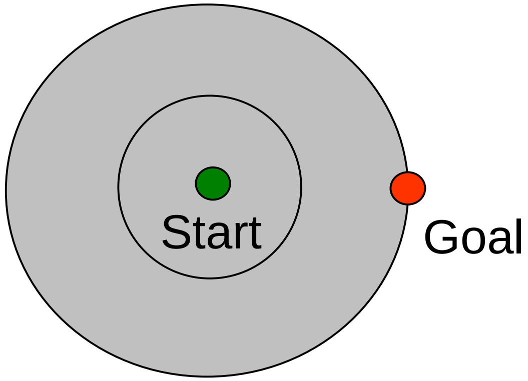


Uniform Cost Issues

- 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

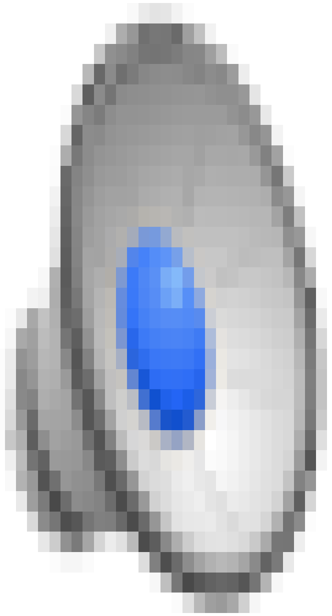
  Add the information-> A* Algorithm, a heuristic algorithm.
- We'll fix that soon!



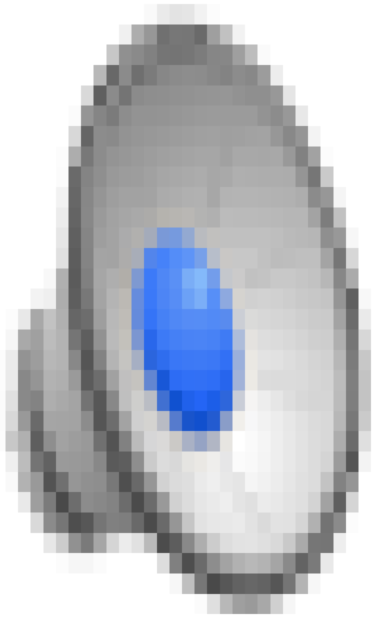


[Demo: empty grid UCS (L2D5)]
[Demo: maze with deep/shallow water DFS/BFS/UCS (L2D7)]

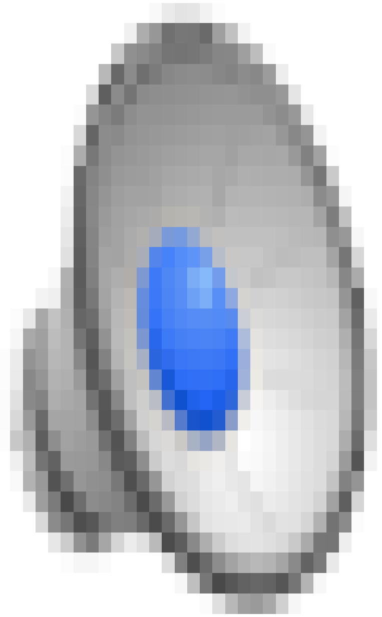
Video of Demo Empty UCS



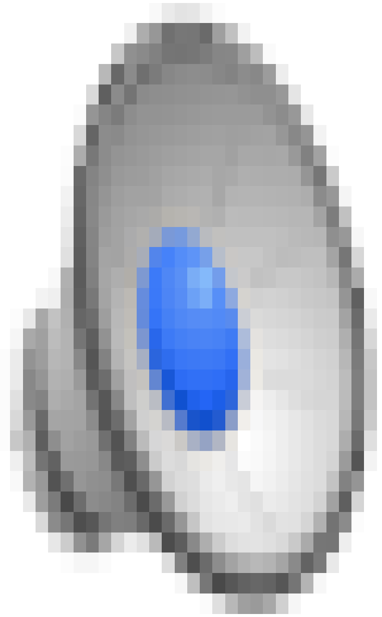
Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 1)



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

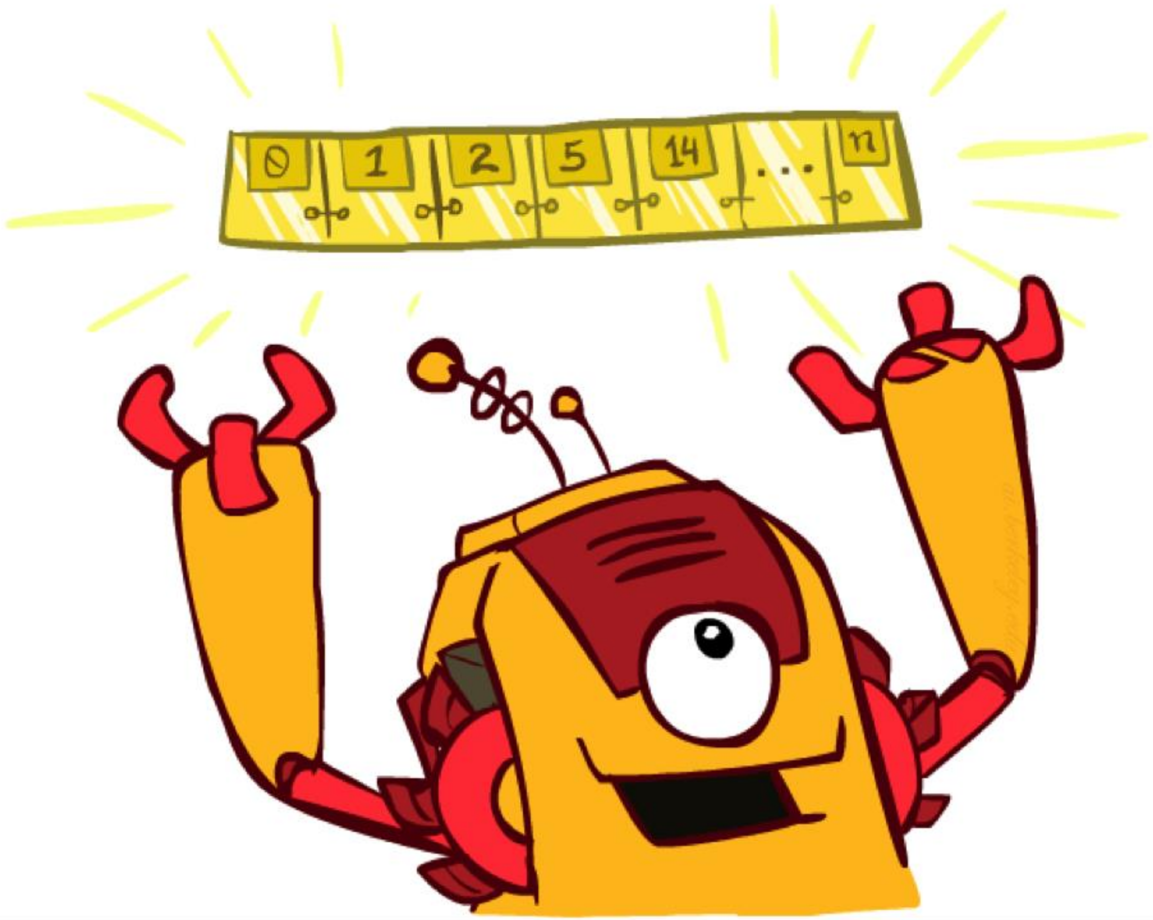


Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 3)



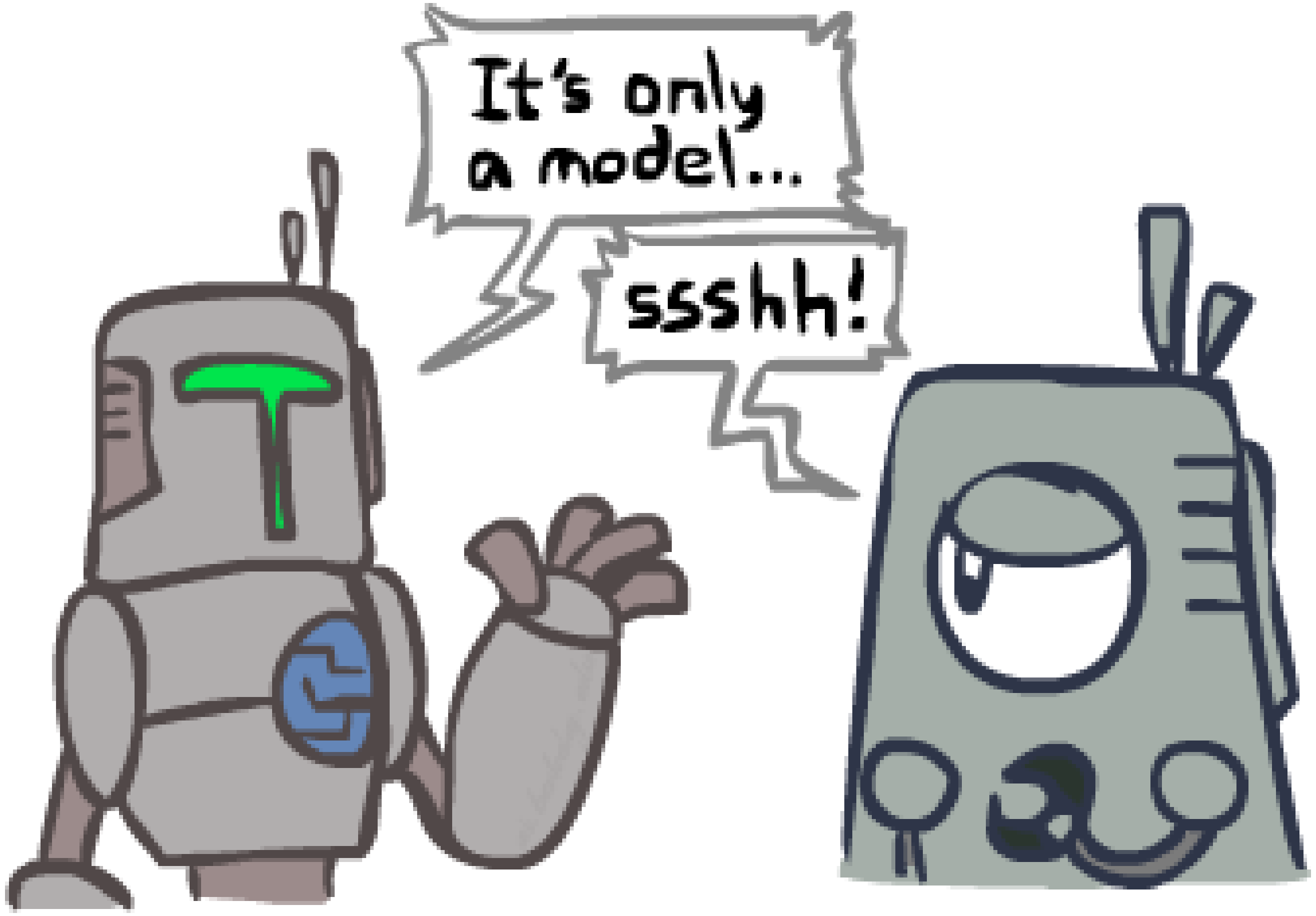
The One Queue

- 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



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?-> If the model is wrong