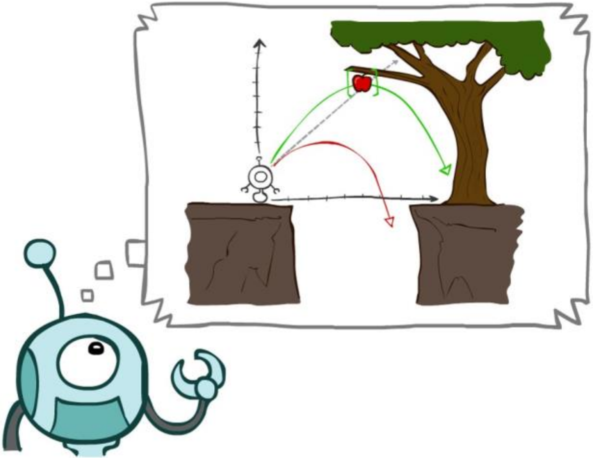
Artificial Intelligence
CSE 4617

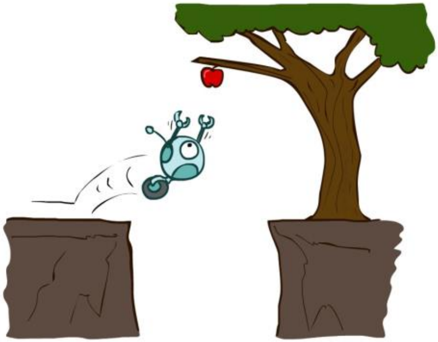
Ahnaf Munir
Assistant Professor
Islamic University of Technology

Agents That Plan



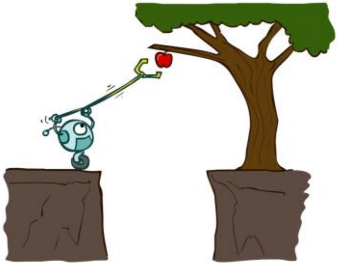
Reflex Agents

- Properties
  - Choose action based on current percept (and maybe memory)
  - May have memory or model of the world's current state
  - Do not consider future consequences
  - Consider how the world IS
  - Can be useful when quick decision is a must
- Can a reflex agent be rational?
  - Remember we only consider the outcome, not the process
  - Only if quick decision is optimal



Planning Agents

- Properties
  - Ask "what if"
  - Decision 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** → Achieve goal in minimum cost
  Complete → When there exists a solution, find it
- **Planning** vs **Replanning**

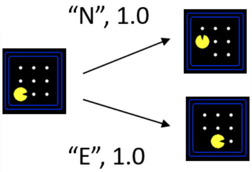


Search Problems

Search Problems

- State space → Set of possible scenarios

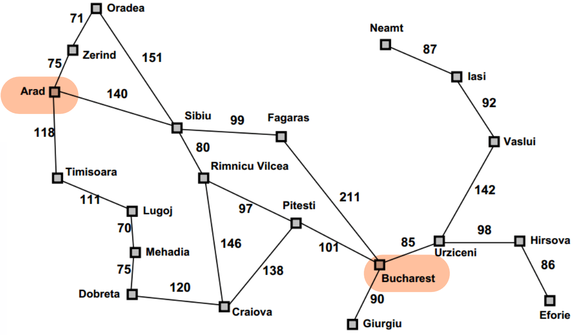
- A successor function (with actions, cost, etc.) → Consequent states for given state



- A start state and goal test
- Solution → A sequence of actions (a plan) which transforms the start state to goal state

Search Problems Are Models

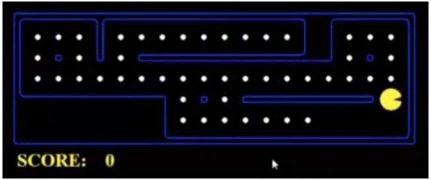
Example: Travelling in Romania



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

What’s in a State Space?

The *world state* includes every last detail of the environment



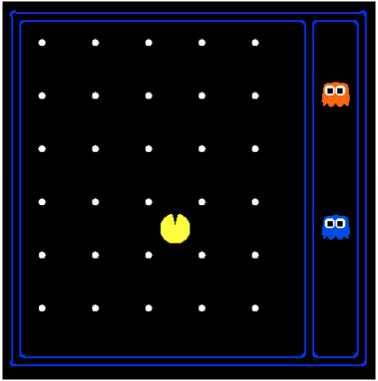
The *search state* keeps only the details needed for planning

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

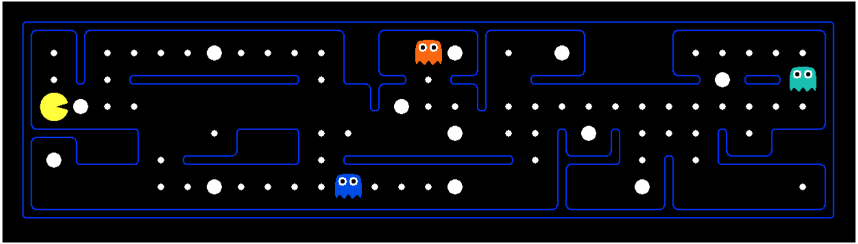
- Problem: Eat-All-Dots
  - States: {(x, y), dot booleans}
  - Actions: NSEW
  - Successor: update location and possibly a dot boolean
  - Goal test: dots all false

State Space Sizes

- Environment
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW
- How many?
  - World states?
    $120 \times 2^{30} \times 12^{2} \times 4$
  - Search states? $\longrightarrow$ Eat-All-Dots
    120×2^{30}
  - Search states? $\longrightarrow$ Pathing
    120

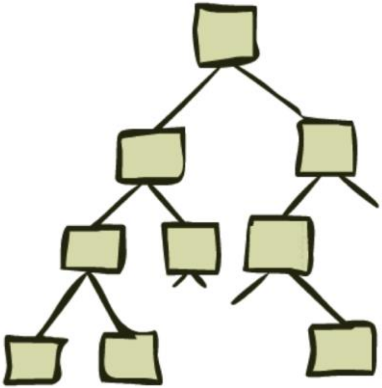


Quiz: Safe Passage



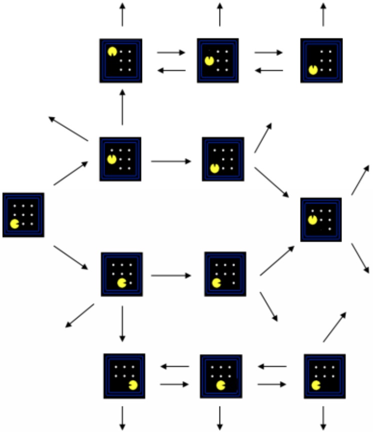
- Problem: Eat all dots while keeping the ghosts perma-scared
- State space? → Agent position, dot booleans, power pellet booleans, remaining scared time

State Space Graphs and Search Trees



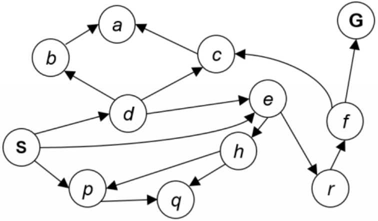
State Space Graphs

- A mathematical representation of a search problem
  - Nodes are (abstracted) world configuration
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)
- Each state occurs only once
- We can rarely build this full graph in memory

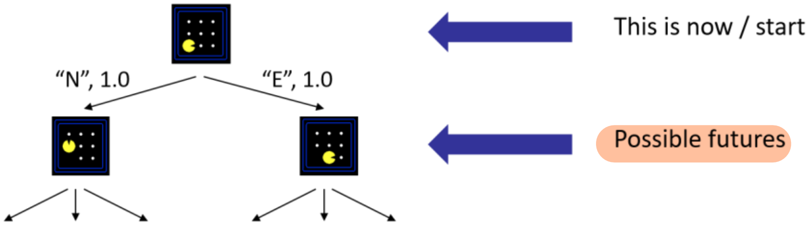


State Space Graphs

- A mathematical representation of a search problem
  - Nodes are (abstracted) world configuration
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)
- Each state occurs only once
- We can rarely build this full graph in memory

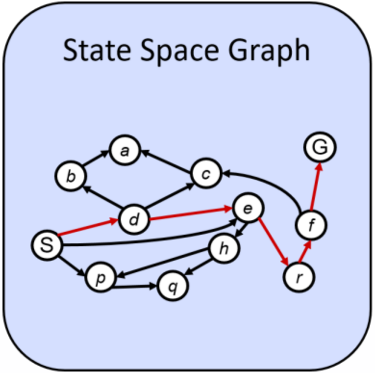


Search Trees



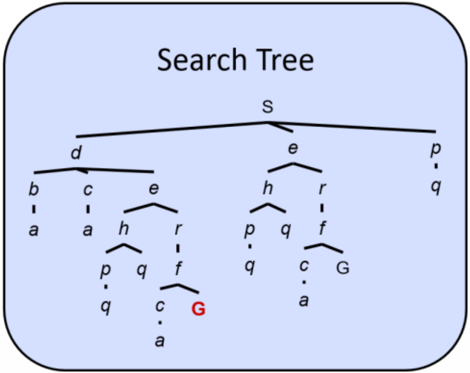
- A "what if" tree of plans and their outcomes
- The start state is the root node
- Children correspond to successors
- Nodes show states, but correspond to PLANS that achieve those states
- For most problems, we can never actually build the tree

State Space Graphs vs. Search Trees



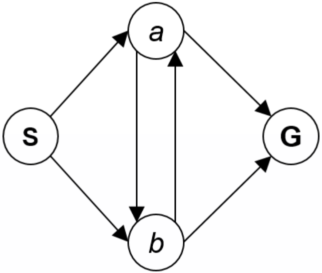
Each NODE in in the search tree is an entire PATH in the state space graph.

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



Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



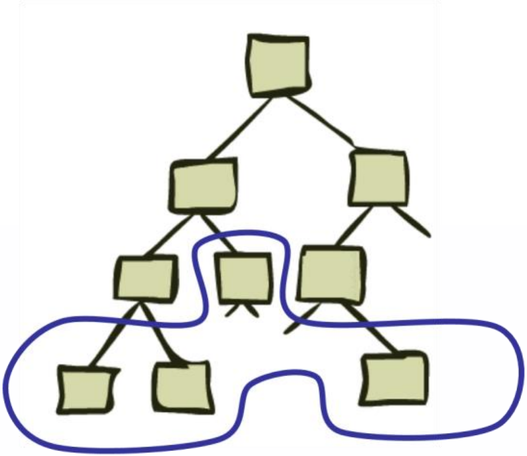
Let, s be the root.

How big is the search tree?

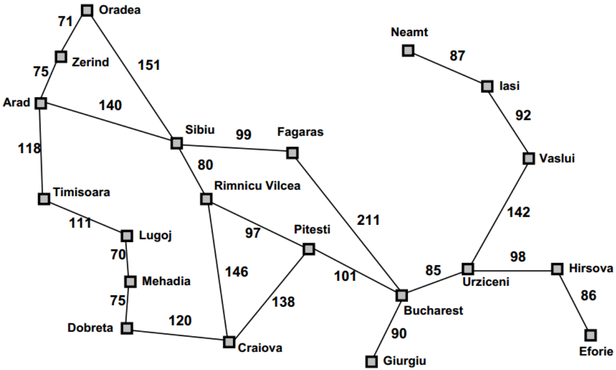
$$\infty$$

Lots of repeated structures in the search tree!

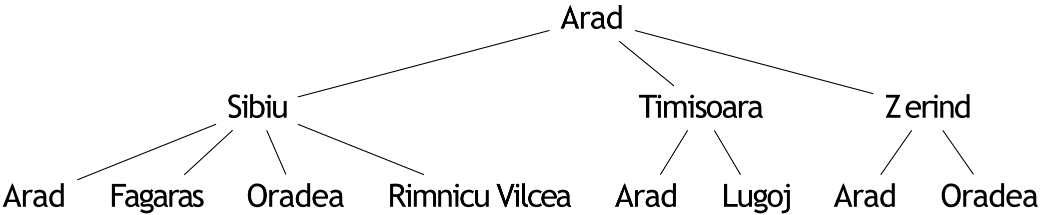
Tree Search



Search Example: Romania



Searching with a Search Tree



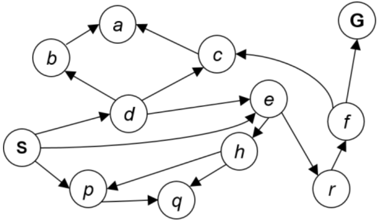
- Expand out potential plans
- Maintain a fringe (list) of partial plans under consideration
- Try to expand as few tree nodes as possible

General Tree Search

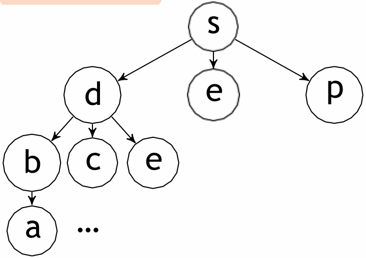
```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion
      then return failure
    choose a leaf node according to strategy
    if the node contains a goal state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

- Concepts: Fringe, Expansion, Exploration strategy

Example: Tree Search



Search Tree:



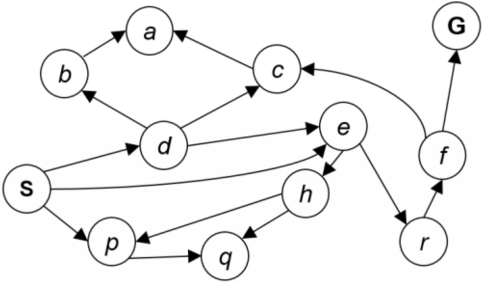
Fringe:

s → e, s → p,
s → d → c, s → d → e,
s → d → b → a ...

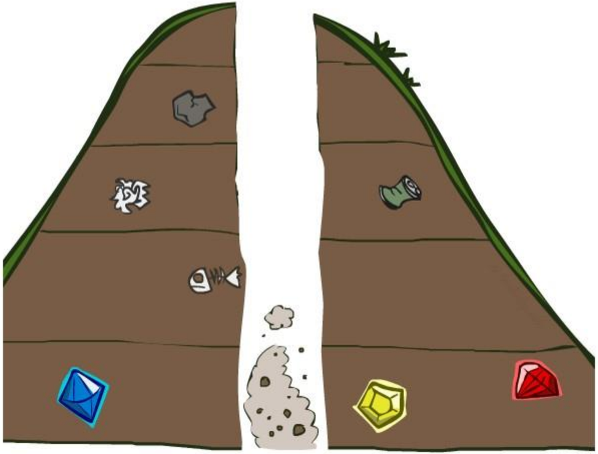
Depth-First Search

Depth-First Search

- Strategy: Expand the deepest node first
- Implementation: Fringe is a LIFO stack



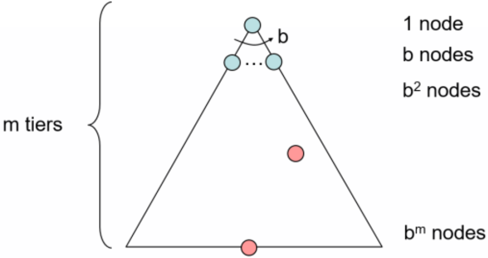
Search Algorithm Properties



Search Algorithm Properties

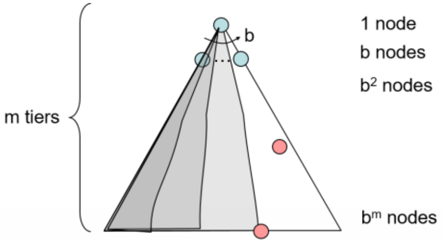
- Complete: Guaranteed to find solution if one exists?
- Optimal: Guaranteed to find the least cost path?

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

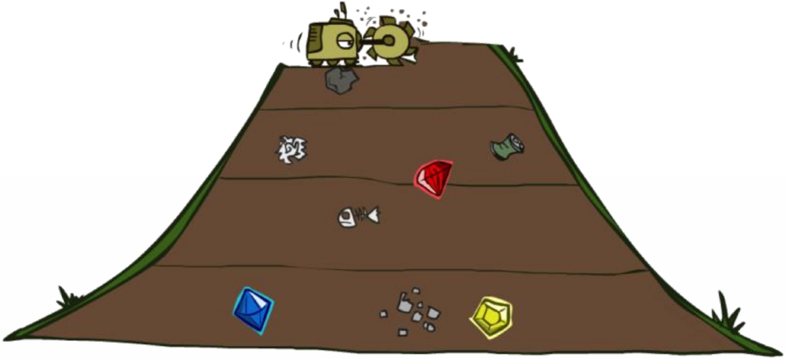


Depth-First Search (DFS) Properties

- What nodes does DFS expand?
  - Some left prefix of the tree
  - Could process the whole tree
  - If m is finite, takes time $O(b^m)$
- How much does a fringe take?
  - Only has siblings on path to root, $O(bm)$
- Is it complete?
  - m could be infinite, so only if we prevent cycles
- Is it optimal?
  - No, it finds the "leftmost" solution, regardless of the depth/cost

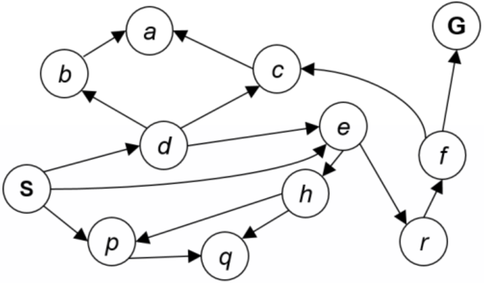


Breadth-First Search



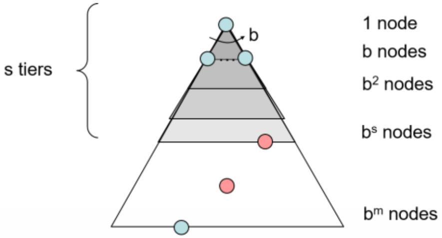
Breadth-First Search

- Strategy: Expand the shallowest node first
- Implementation: Fringe is a FIFO queue



Breadth-First Search (BFS) Properties

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



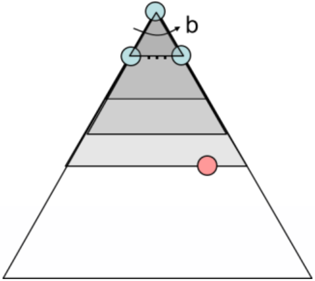
Quiz: DFS vs BFS

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

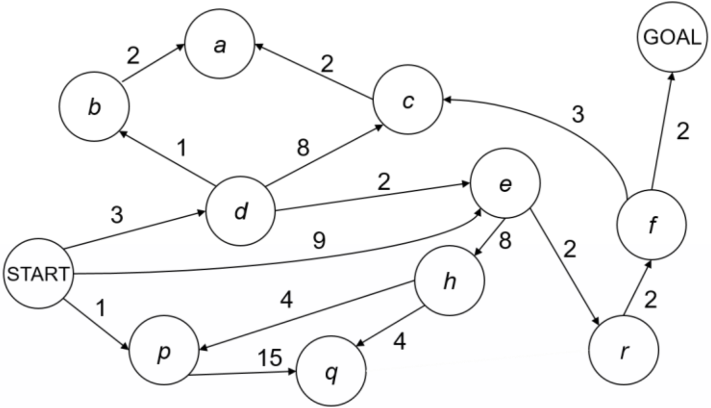
Video: Empty-BFS, Empty-DFS, MazeWater-BFS, MazeWater-DFS

Iterative Deepening

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run DFS with depth limit 1. If no solution…
  - Run DFS with depth limit 2. If no solution…
  - Run DFS with depth limit 3. …
- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!

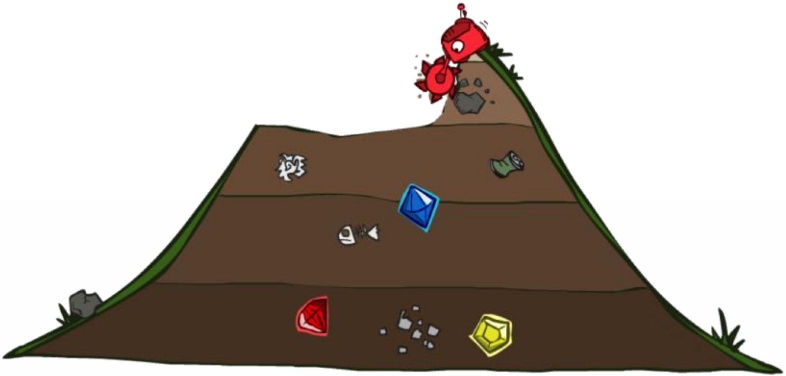


Cost-Sensitive Search



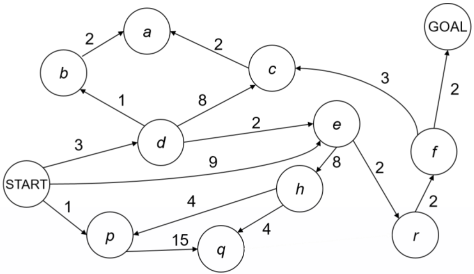
BFS is optimal in terms of the number of actions performed.
It does not find the least cost path.

Uniform Cost Search



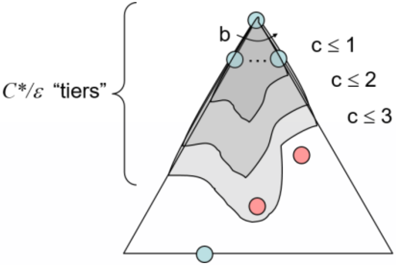
Uniform Cost Search

- Strategy: Expand the cheapest node first
- Implementation: Fringe is a priority queue
  - Priority: Cumulative cost



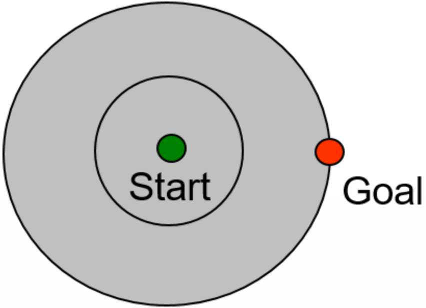
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Process all nodes with cost less than cheapest solution
  - If the solution costs C^* and arcs cost at least ϵ, then "effective depth" is roughly C^*/ϵ
  - Takes time $O(b^{C^*/\epsilon})$
- How much does a fringe take?
  - Has roughly the last tier, $O(b^{C^*/\epsilon})$
- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes
- Is it optimal?
  - Yes! (Proof, in future class)



Uniform Cost Issues

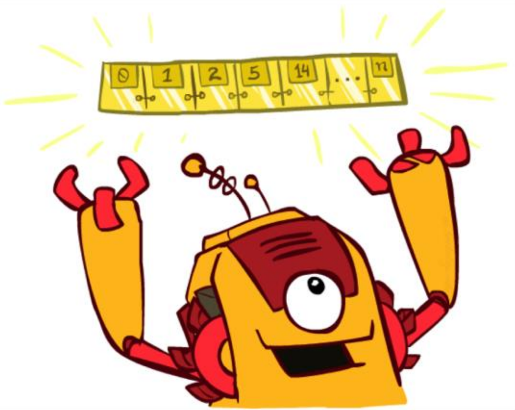
- Remember: UCS explores increasing cost contours
- The good: UCS is complete and optimal
- The bad
  - Explores options in every "direction"
  - No information about goal location



Video: MazeShallowDeep-UCS, MazeShallowDeep-BFS, MazeShallowDeep-DFS

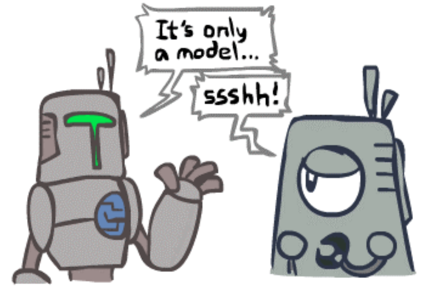
The One Queue

- 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…



Suggested Reading

- Russell & Norvig: Chapter: 3.1-3.4
- Poole & Mackworth: Chapter: 3.1-3.5