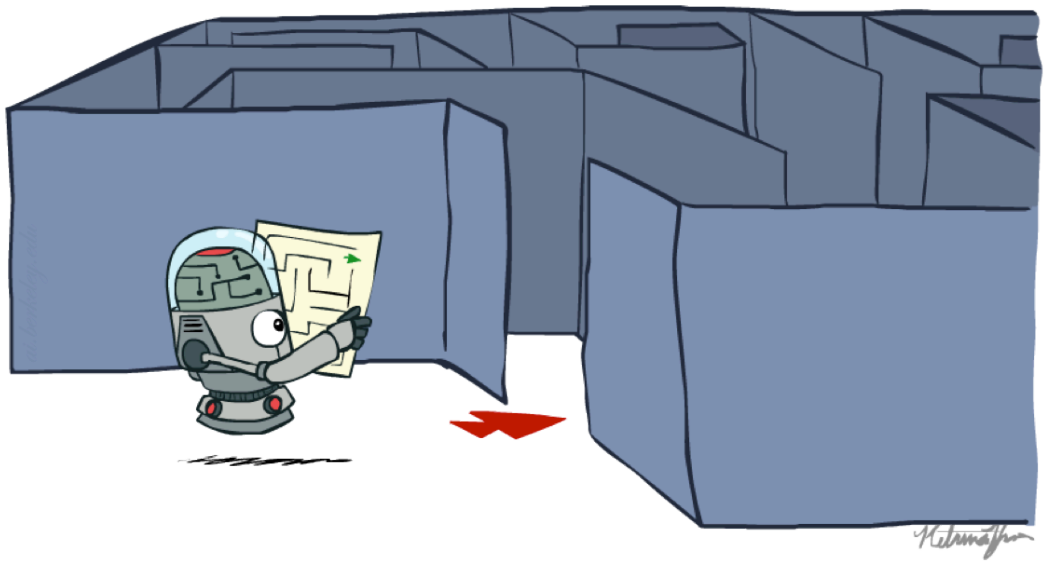
CS 4100 // artificial intelligence

instructor: byron wallace



Search I

Attribution: many of these slides are modified versions of those distributed with the *UC Berkeley CS188* materials
Thanks to John DeNero and Dan Klein

Quick announcements

We now have a piazza site (linked to on homepage)

- You should have received an email about this

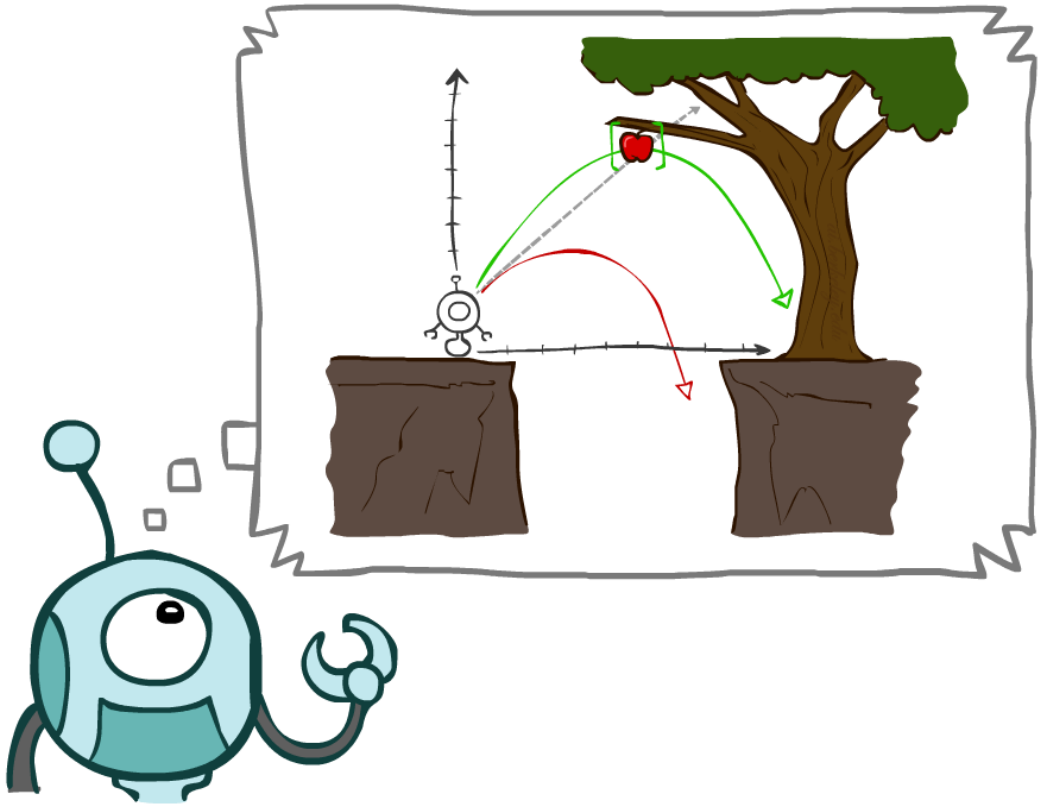
Instructions for setting up a git account are available and were sent out; this is how you will turn in assignments!

- If there is interest, Rohan (our TA) will run a crash course in the (very) basics of git – enough so that you can turn in the assignments!

Questions before we begin?

Search I

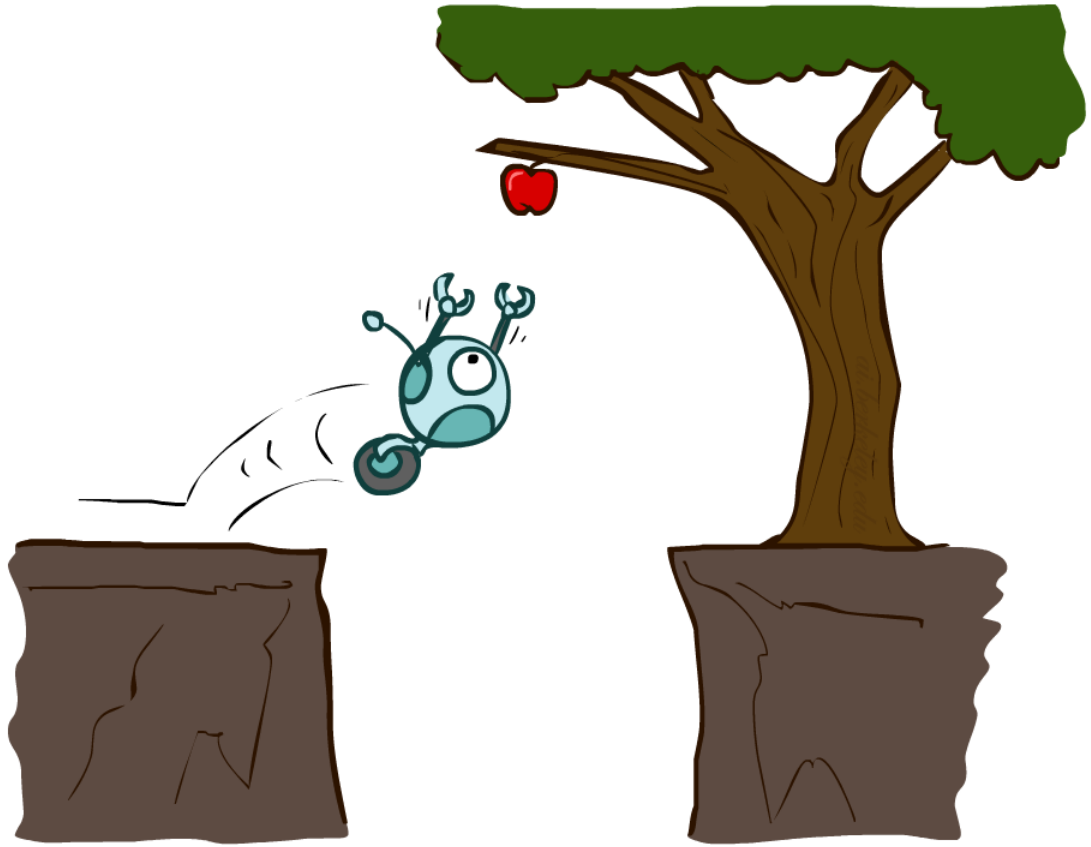
- *Agents* that Plan Ahead
- We will treat plans as *search problems*
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search



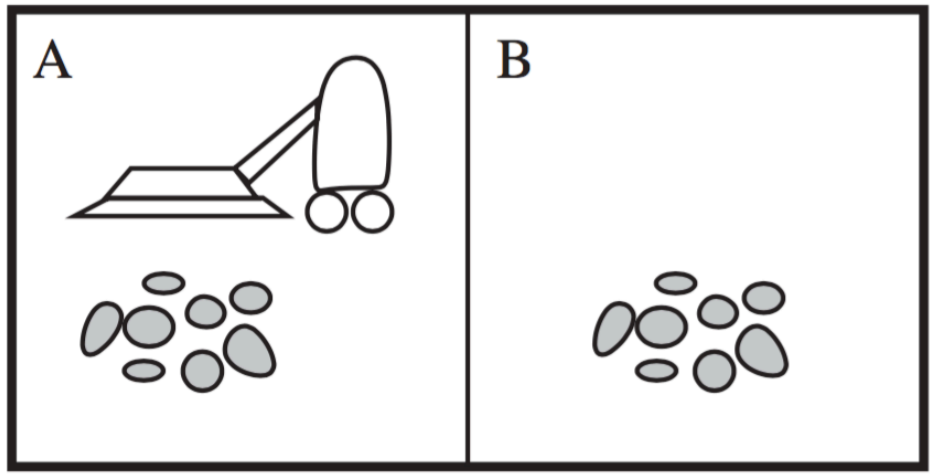
“Reflex agents”

- Reflex agents:
  - Choose action based on *current* percept only
  - May have memory or a model of the world’s current state
  - Do not consider the future consequences of their actions

- Can a reflex agent be rational?



Rational vacuum cleaner



Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Dirty]$	$Suck$
⋮	⋮
$[A, Clean], [A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Clean], [A, Dirty]$	$Suck$
⋮	⋮

The trouble with reflex agents

Problem solving agents

Goal-based: want to accomplish some goal, e.g., maximize *utility* or similar performance measure

Problem solving agents

Goal-based: want to accomplish some goal, e.g., maximize *utility* or similar performance measure

Simple problem-solving agents ***search*** *for sequences of actions that will realize their goals*

- This entails evaluating ‘future’ actions

Problem solving agents

Goal-based: want to accomplish some goal, e.g., maximize *utility* or similar performance measure

Simple problem-solving agents ***search*** *for sequences of actions that will realize their goals*

- This entails evaluating ‘future’ actions

Assuming S discrete states, finding the “best” sequence of T actions is non-trivial. Doing this brute force would require evaluating S^T sequences

- Motivates use of clever search algorithms

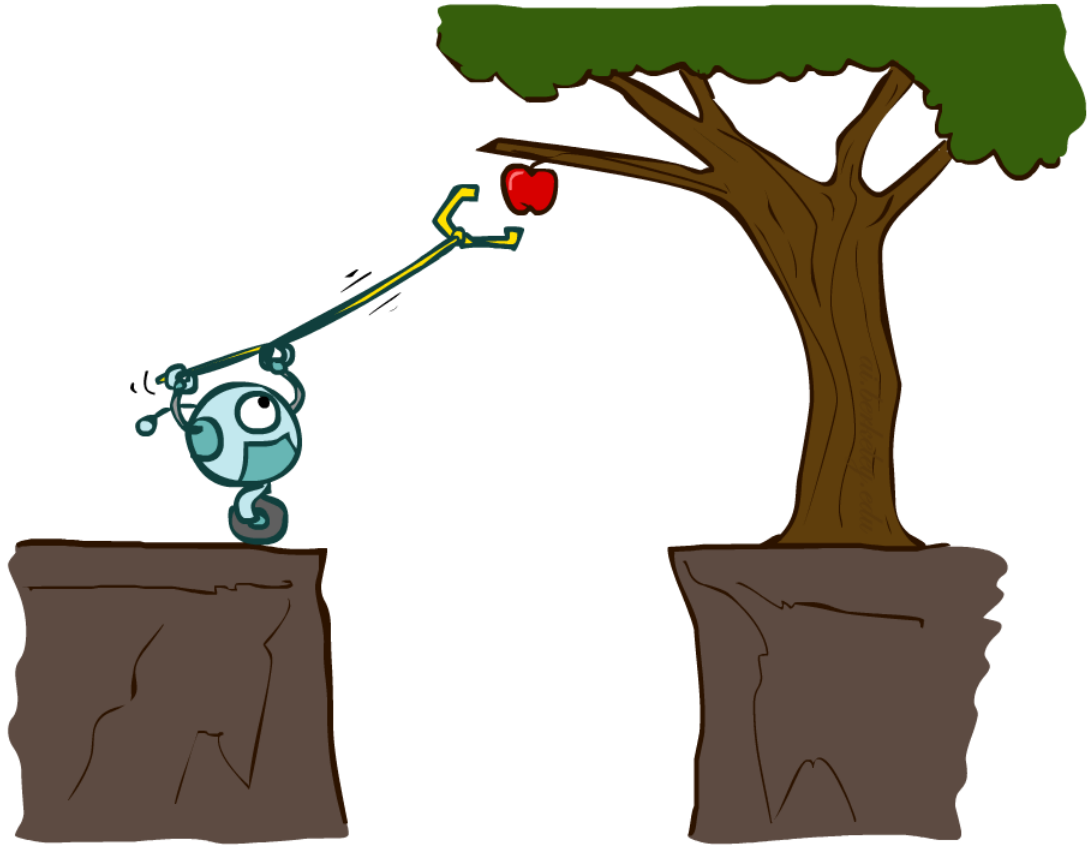
Planning agents

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

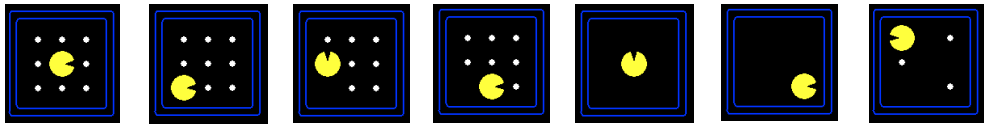


Formalizing search problems

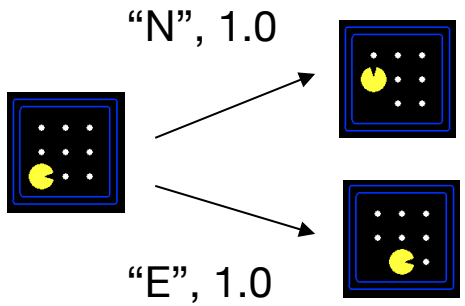
Search problems

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) that transforms the start state to a goal state

The advantage of abstraction

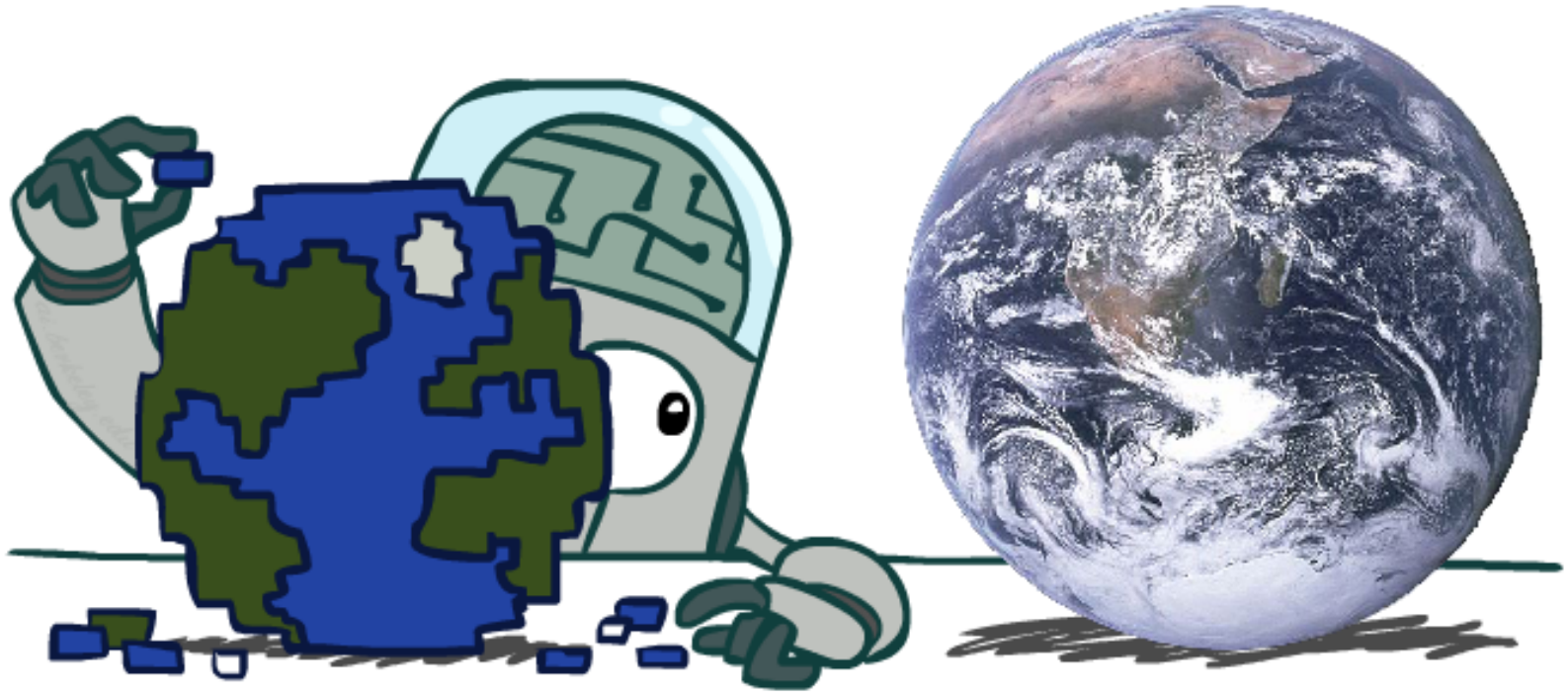
- Most of the work will often be spent formalizing your task into a search problem (designing the state space representation, successor functions and so on). Is it worth it?

The advantage of abstraction

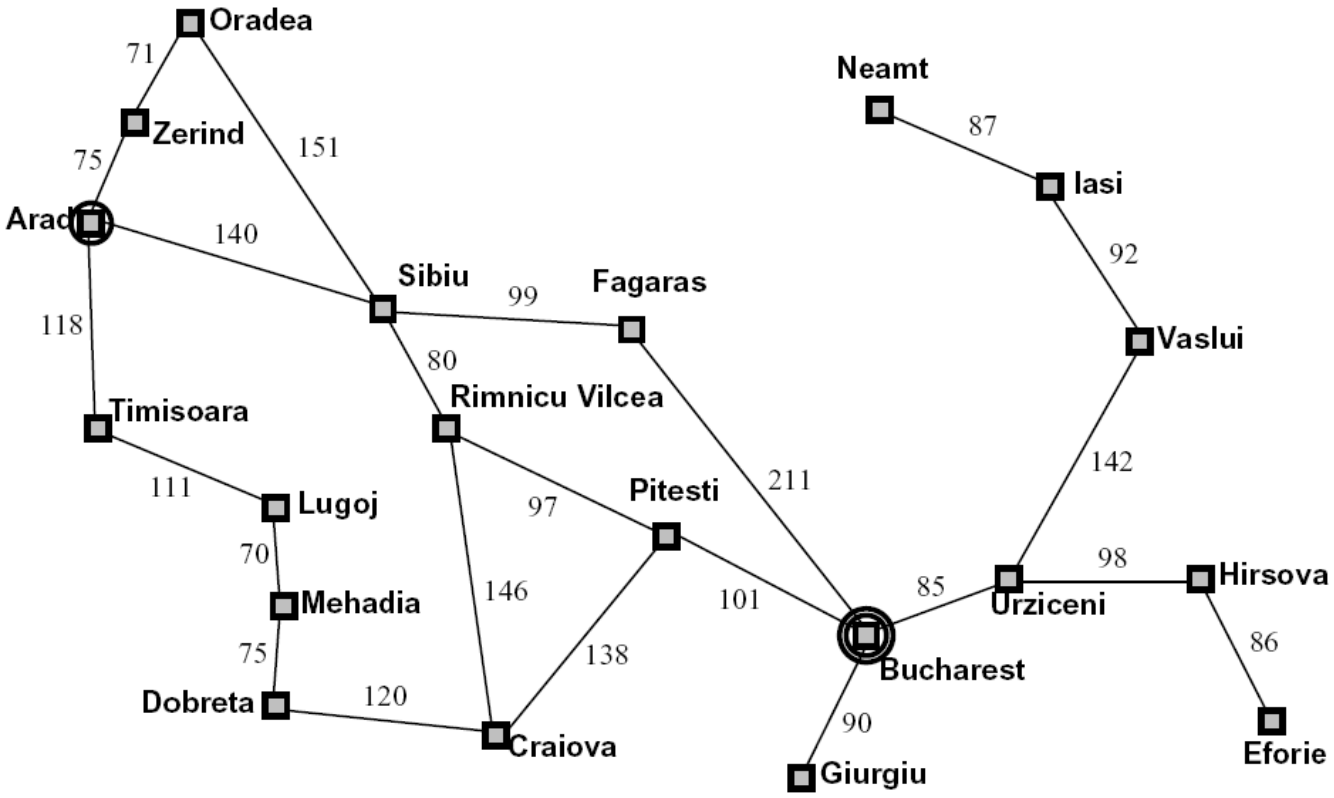
- Most of the work will often be spent formalizing your task into a search problem (designing the state space representation, successor functions and so on). Is it worth it?

- Yes. Because once we have formalized our problem in this way, we can hand it to off-the-shelf search algorithms to find a solution!

Search problems are models



Example from the text: traveling in Romania



State space

- Cities

Successor function

- Roads: Go to adjacent city with cost = distance

Start state

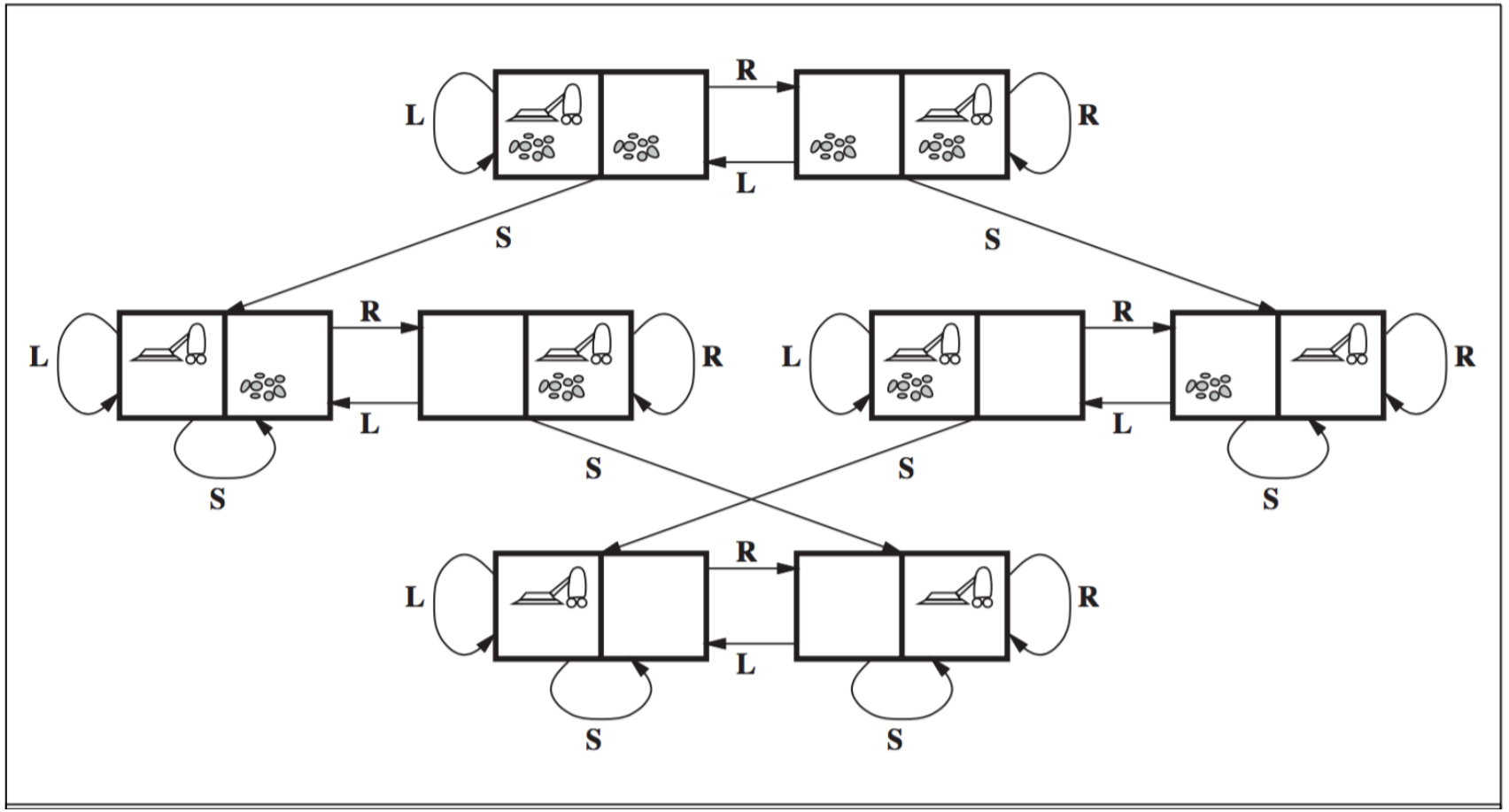
- Arad

Goal test

- Is state == Bucharest?

Solution?

Another example



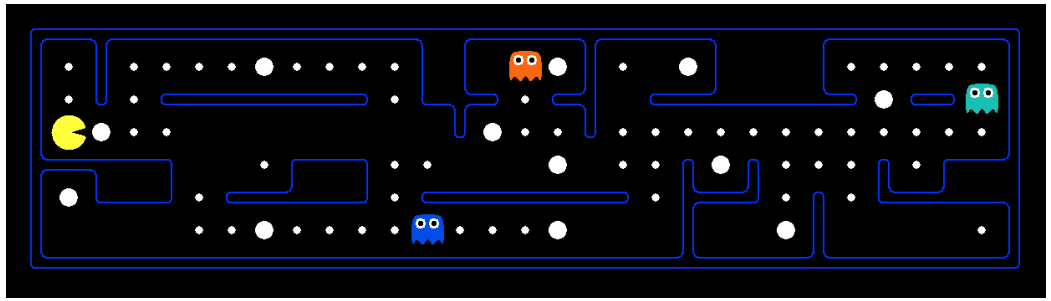
States?

Actions?

Goal?

World states v. search states

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 (if we eat food)
- Goal test: dots all false

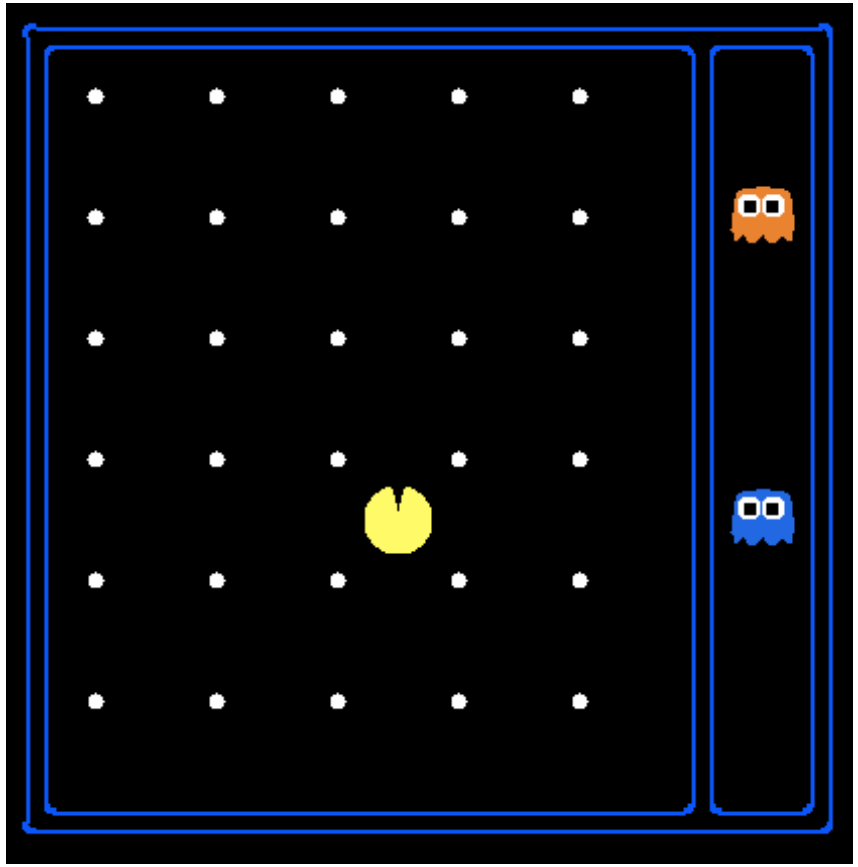
State space sizes?

World state:

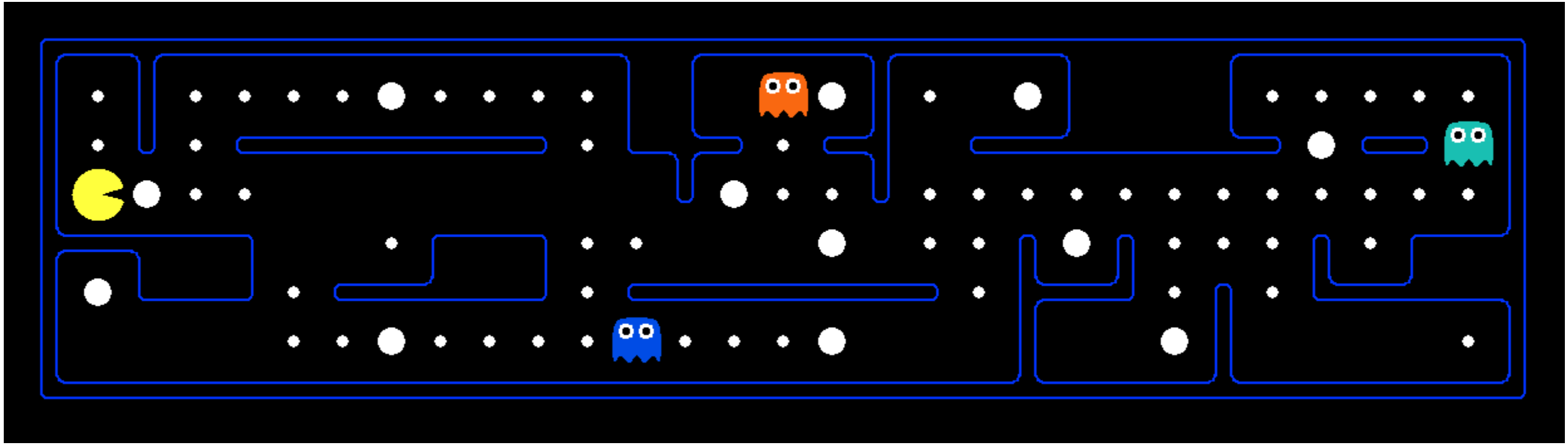
- Agent positions: 120 (assume 10x12 grid)
- Food (dots) count: 30
- Ghost positions: 12
- Agent facing: NSEW

How many

- World states?
  $120 \times 2^{30} \times 12^2 \times 4$
- States for *pathing*?
  120
- States for *eat-all-dots*?
  120×2^{30}



One more example: *safe passage*

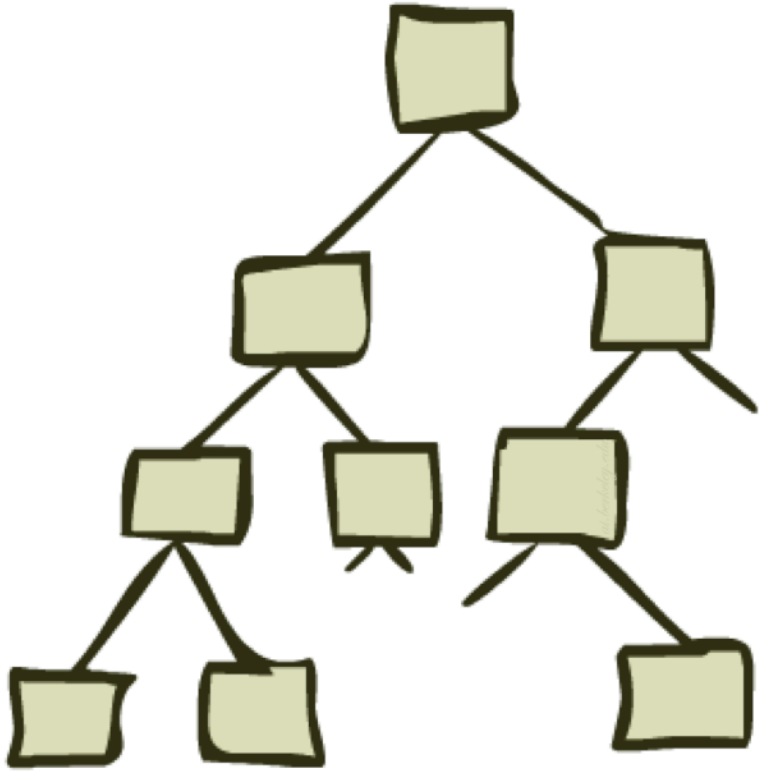


Problem: eat all dots while keeping the ghosts perma-scared

What does the state space have to specify?

- (agent position, dot booleans, power pellet booleans, remaining scared time)

State space graphs and search trees



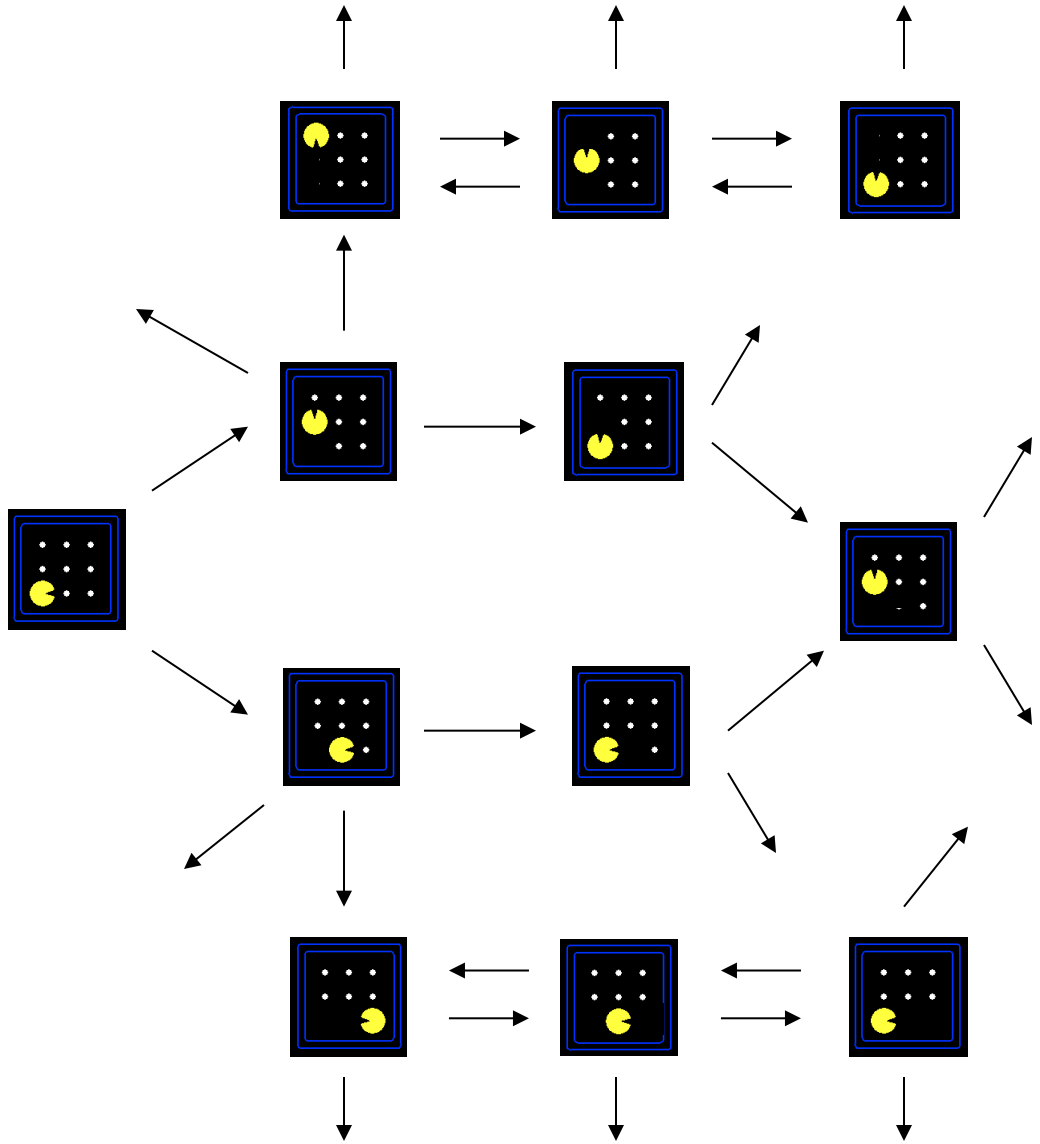
State space graphs

State space graph: A mathematical representation of a search problem

- Nodes are (abstracted) world configurations
- Arcs represent successor states (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/abstraction



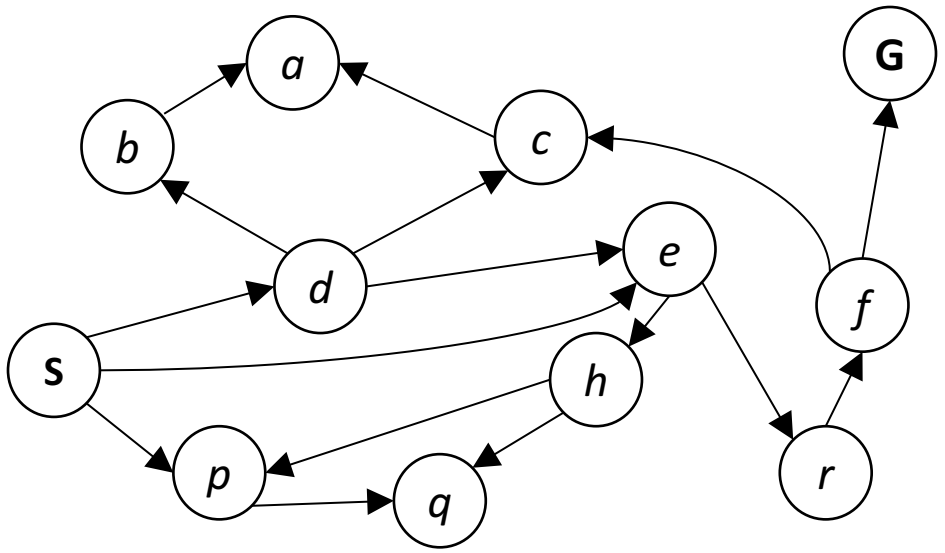
State space graphs

State space graph: A mathematical representation of a search problem

- Nodes are (abstracted) world configurations
- Arcs represent successor states (action results)
- The goal test is a set of goal nodes (maybe only one)

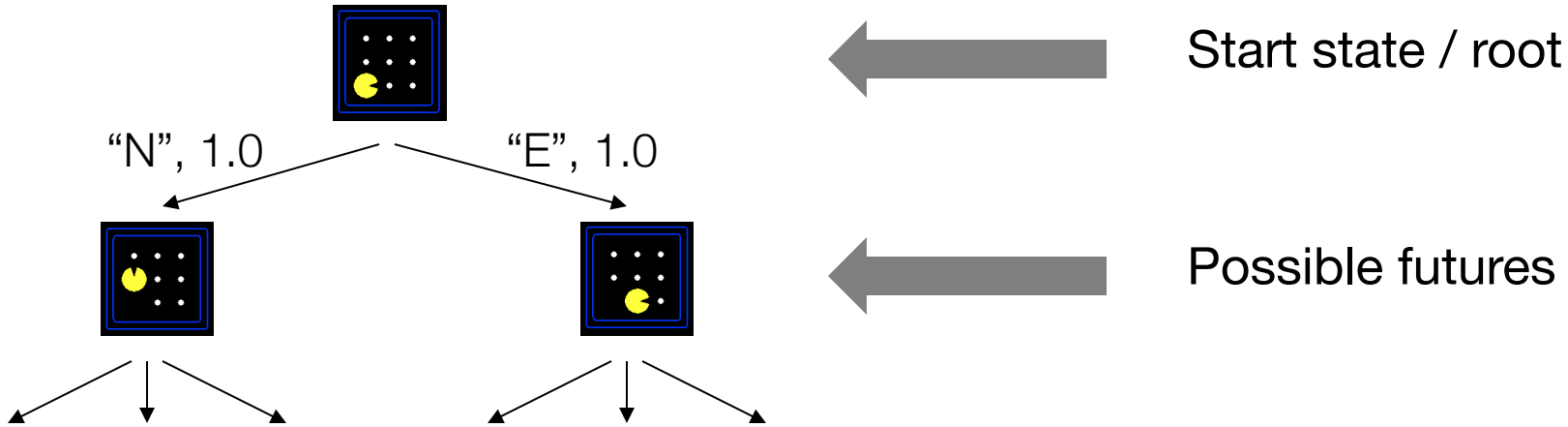
In a state space graph, each state occurs only once!

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



Tiny search graph for a tiny search problem

Search trees

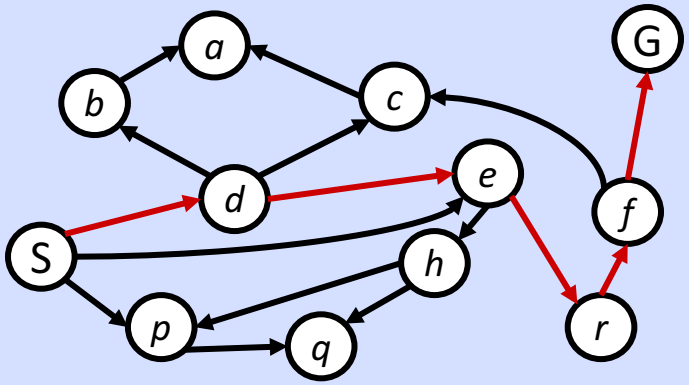


A search tree:

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

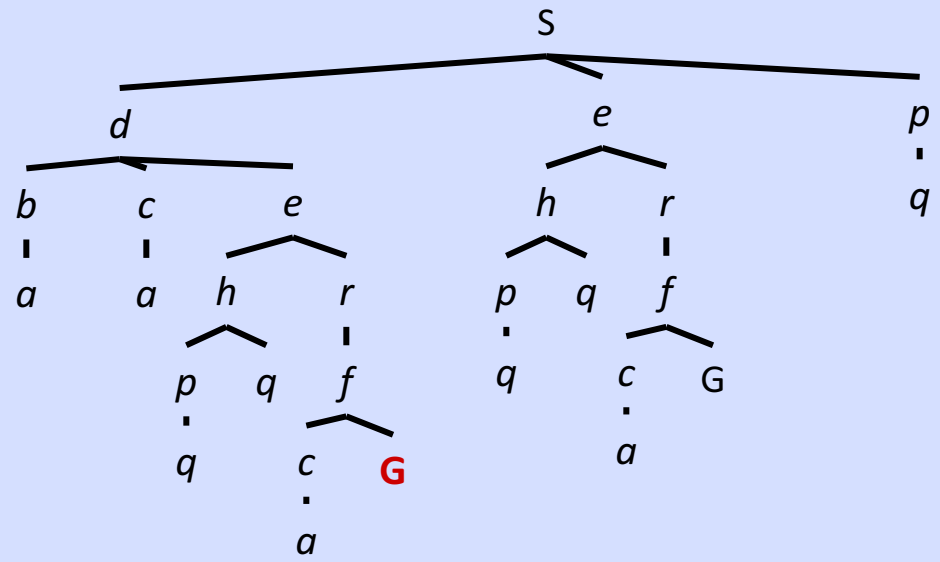
State space graphs vs. search trees

State space graph



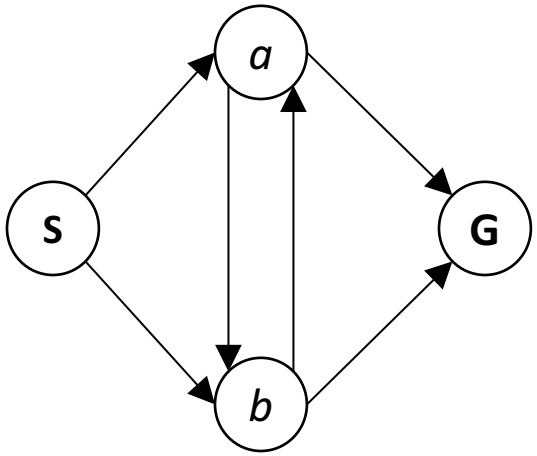
Each *node* in the **search tree** is an *entire path* in the **state space graph**.

Search tree



Pop Q: state space graphs vs. search trees

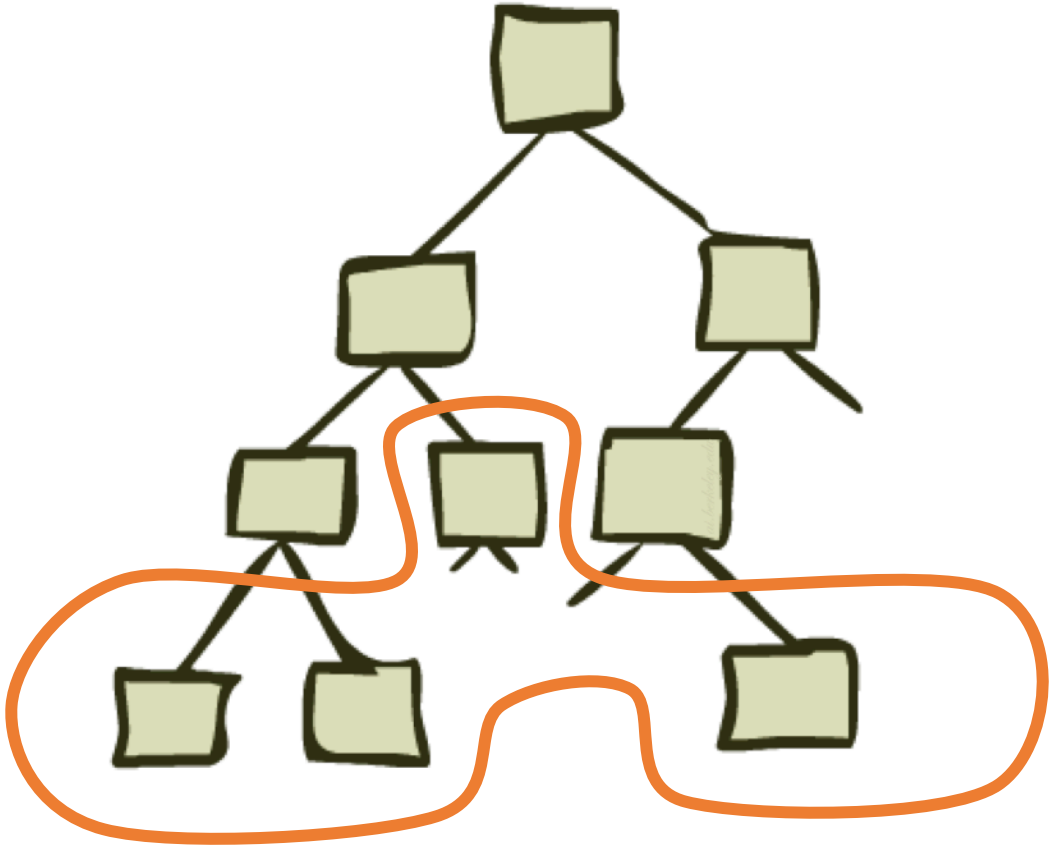
consider this 4-state graph



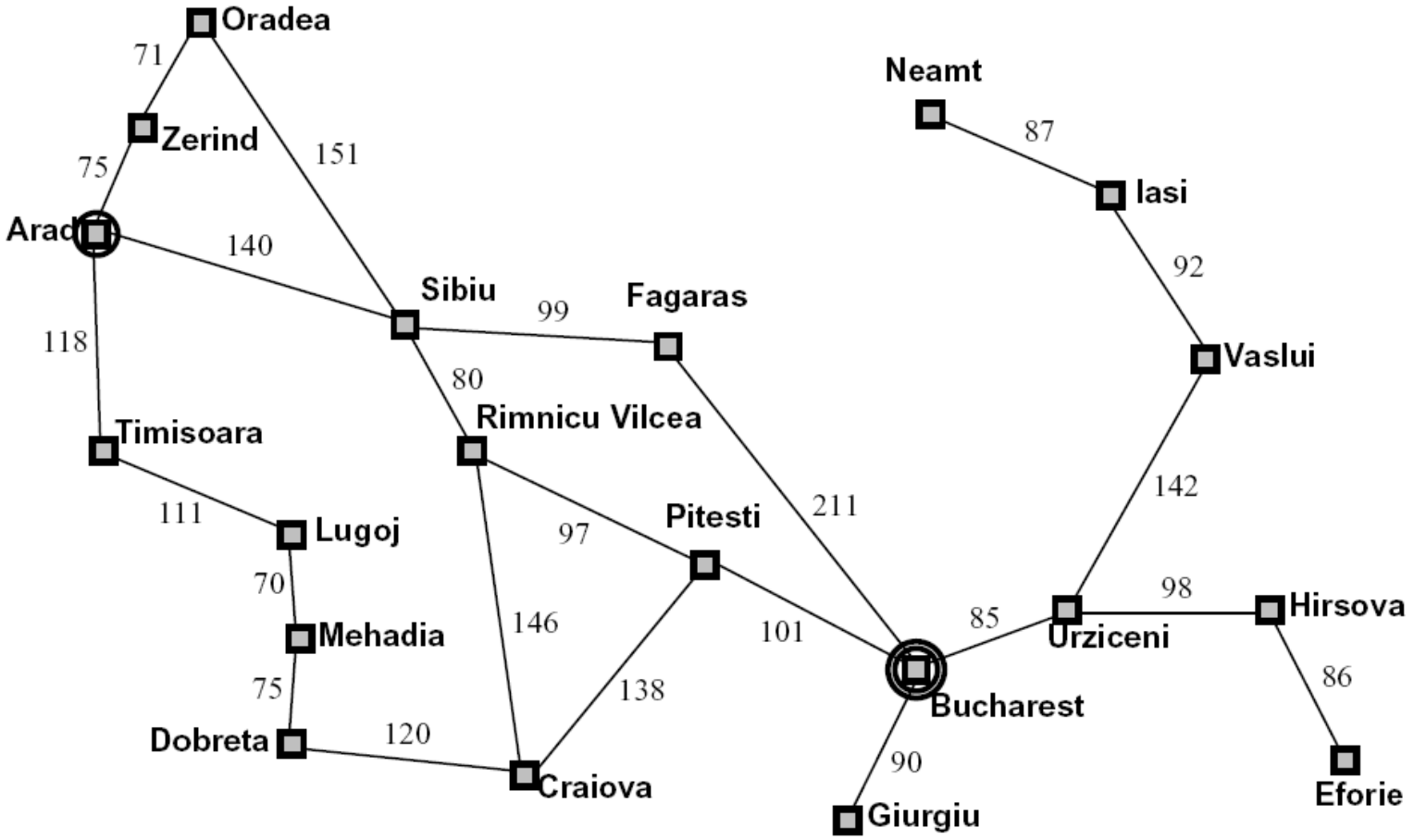
how big is its search tree (from S)?

Important: Lots of repeated structure in the search tree!

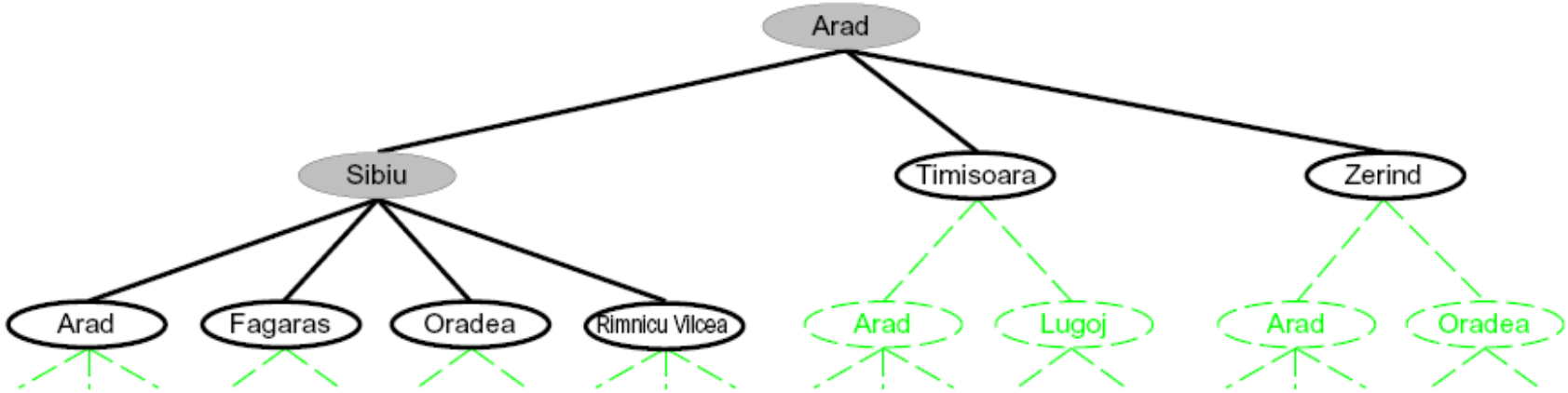
This brings us to *tree search*



Search example: Romania



Searching with a search tree



General strategy:

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

A bit more formally

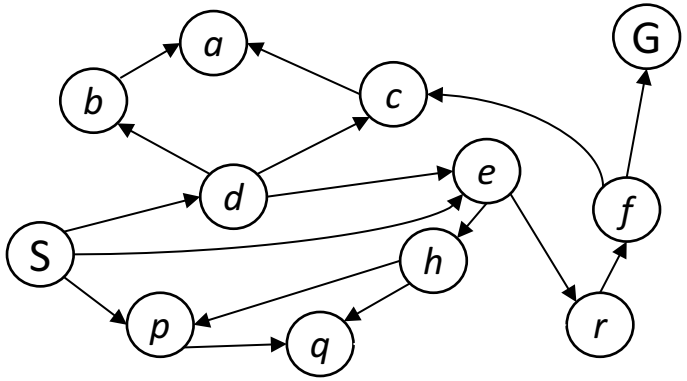
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the 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 (things that may yet work)
- Expansion (picking nodes out of fringe to expand)
- Exploration strategy (*how* to pick from the fringe)

Main question: which fringe nodes to explore?

Example: tree search



How shall we search?

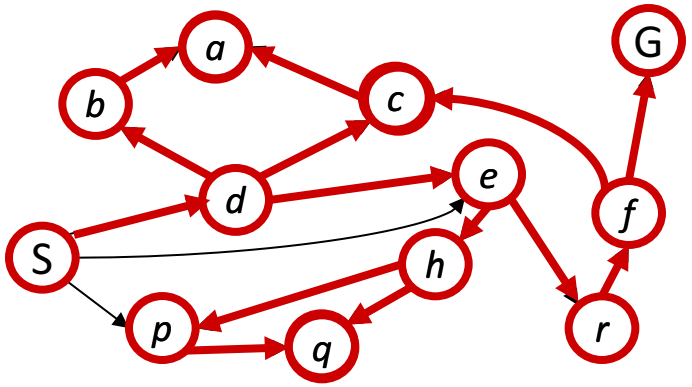
Today we will discuss a few simple strategies for node expansion: Depth First Search (DFS) and Breadth First Search (BFS)

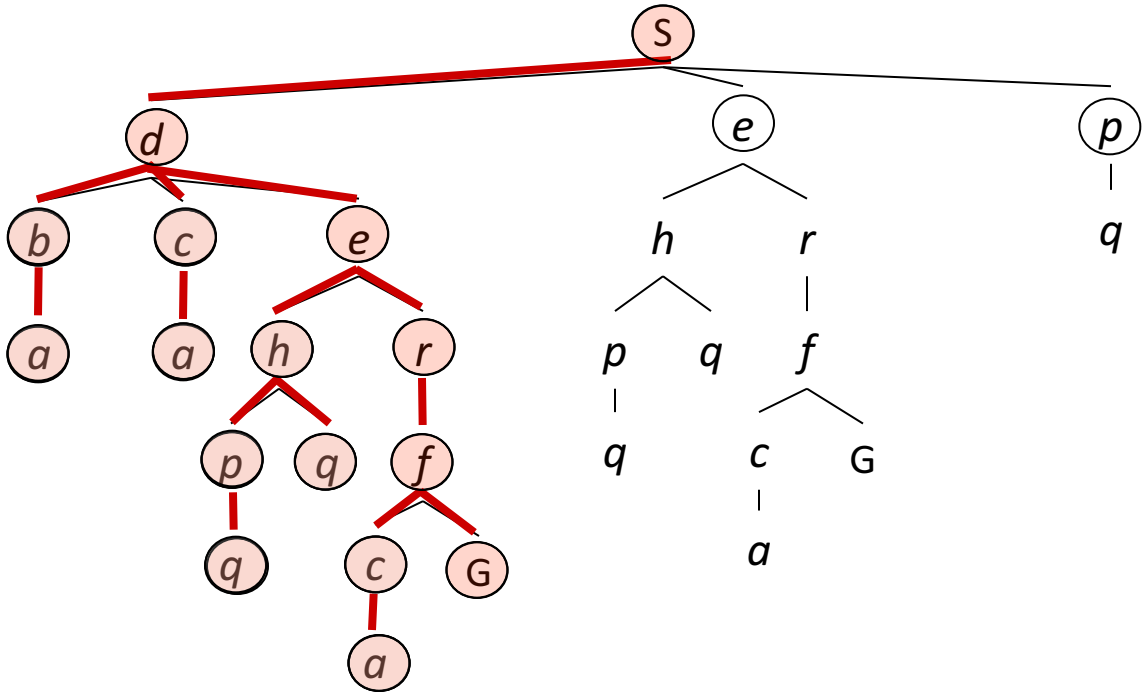
Depth-First Search (DFS)

Depth-First Search (DFS)

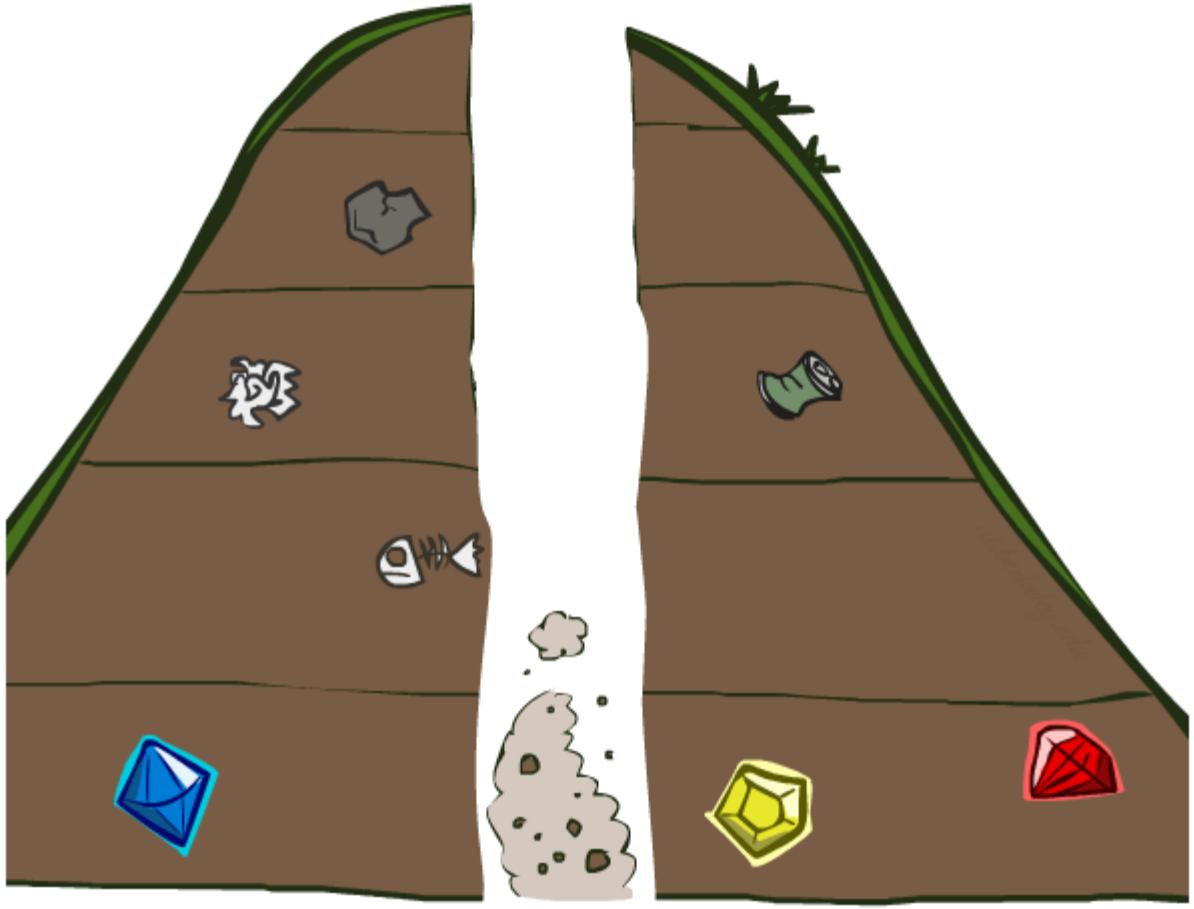
Strategy expand a deepest node first

Implementation Fringe is a stack (LIFO)





Search algorithm properties



Search algorithm properties

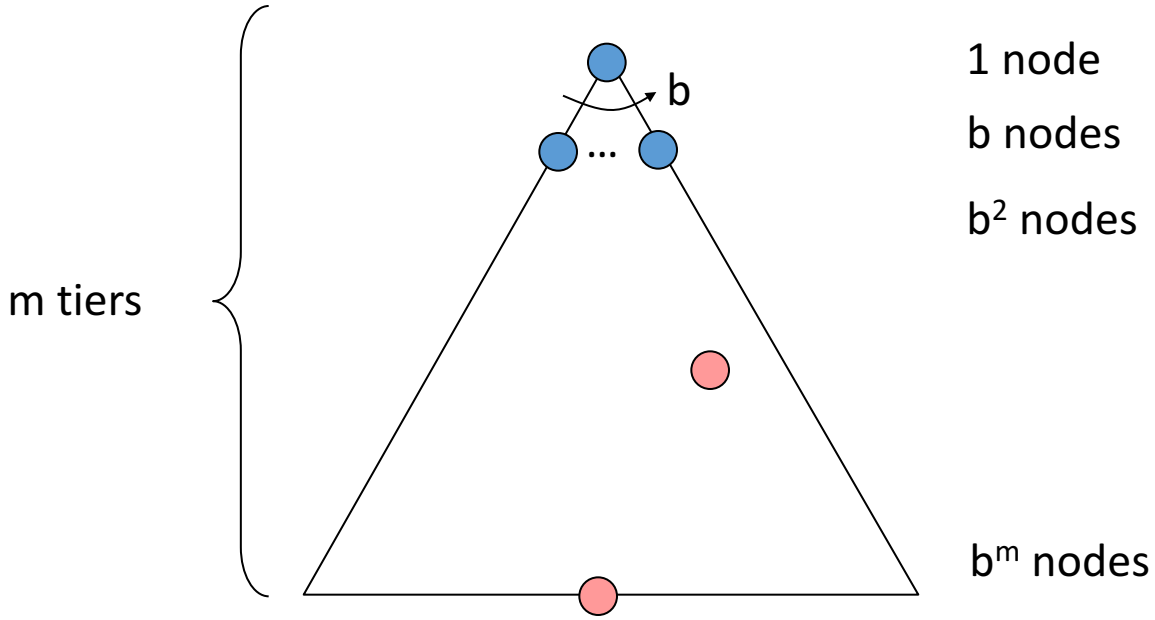
- **Complete**: Guaranteed to find a solution if one exists?
- **Optimal**: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

Cartoon of search tree:

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

Number of nodes in entire tree?

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



In sum: 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 space does the fringe take?

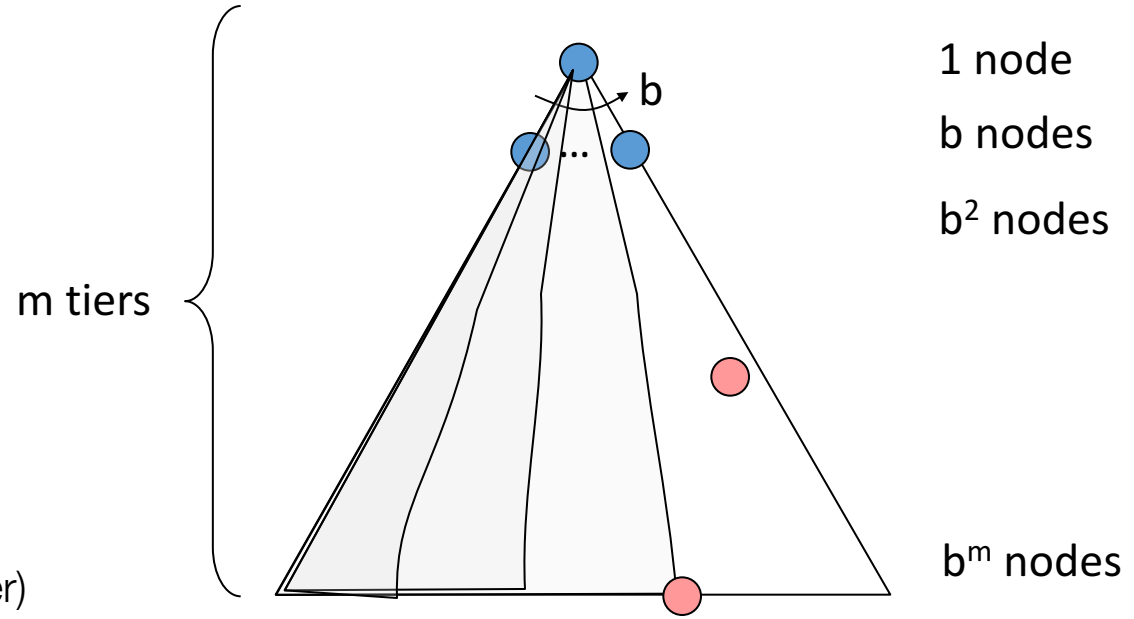
- Only has siblings on path to root, so O(bm)

Is it complete?

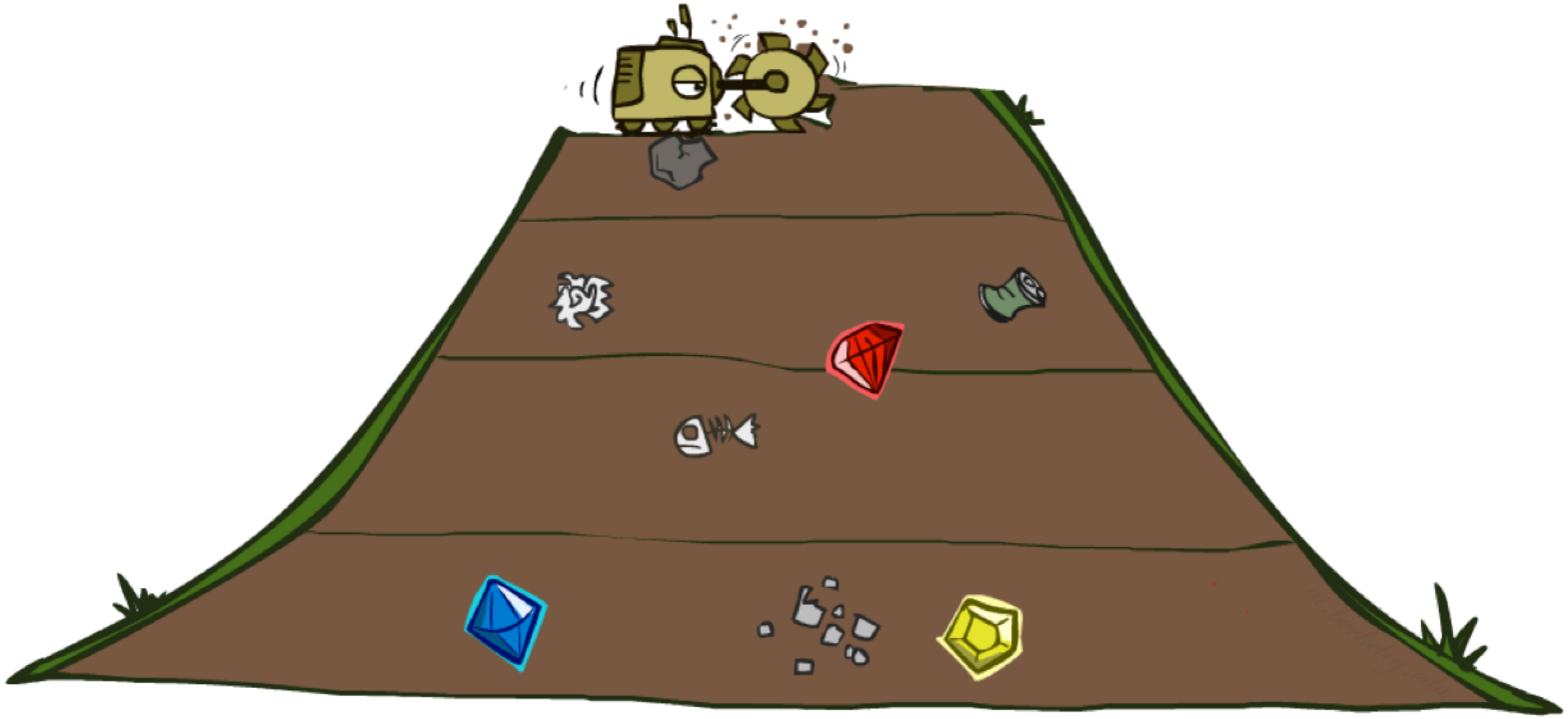
- m could be infinite, so only if we prevent cycles (more later)

Is it optimal?

- No, it finds the “leftmost” solution, regardless of depth or cost



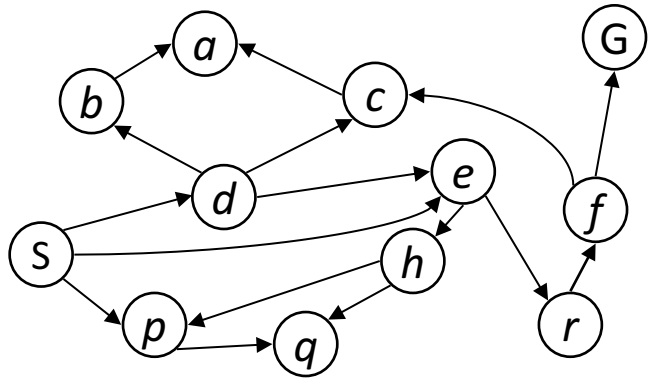
Breadth-First Search (BFS)

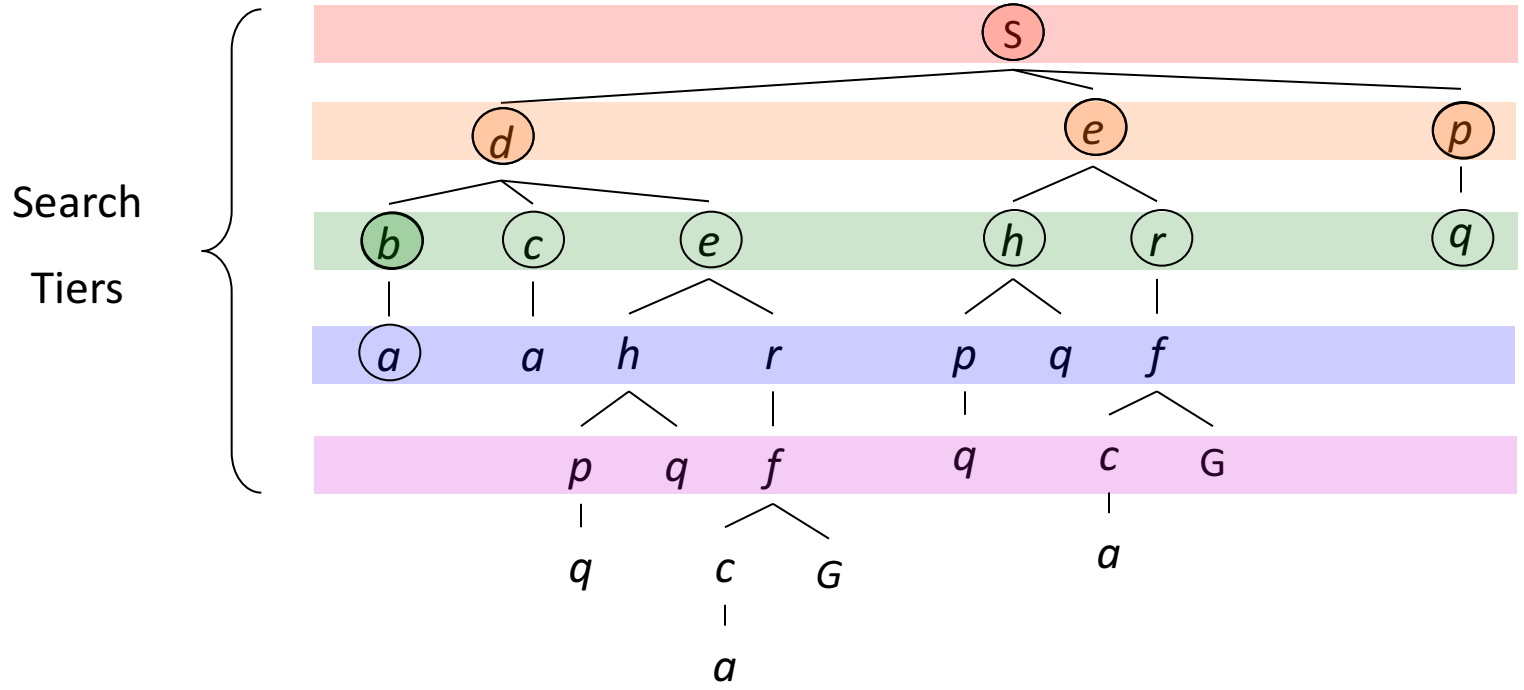


Breadth-First Search (BFS)

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 the depth of the shallowest solution be s
- Search takes time $O(b^s)$

How much space does the fringe take?

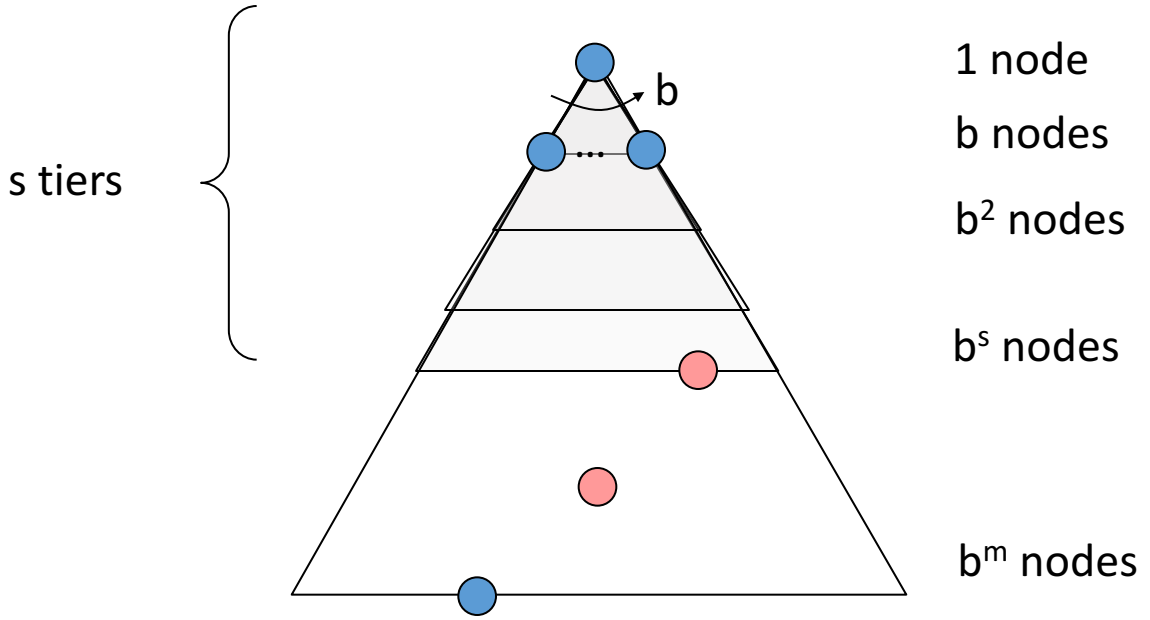
- Has roughly the last tier, so $O(b^s)$
- So in ”worst case” this is $O(b^m)$

Is it complete?

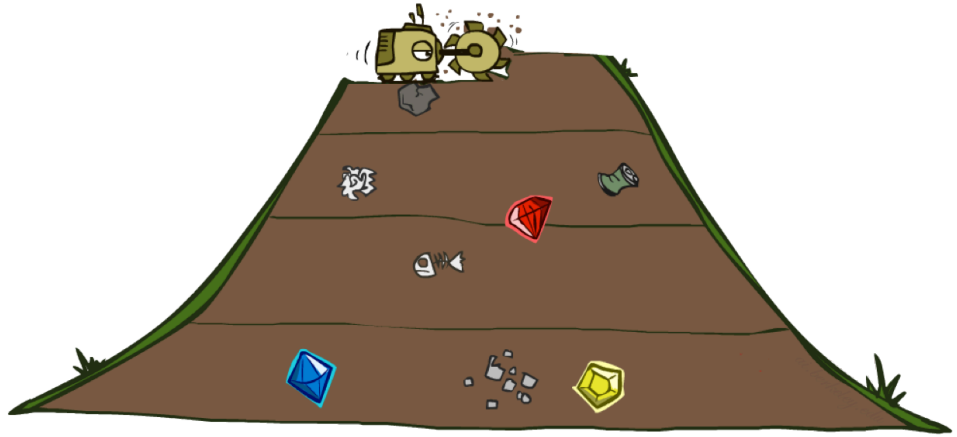
- s must be finite if a solution exists, so yes!

Is it optimal?

- Only if action costs are all 1 (more on costs later)



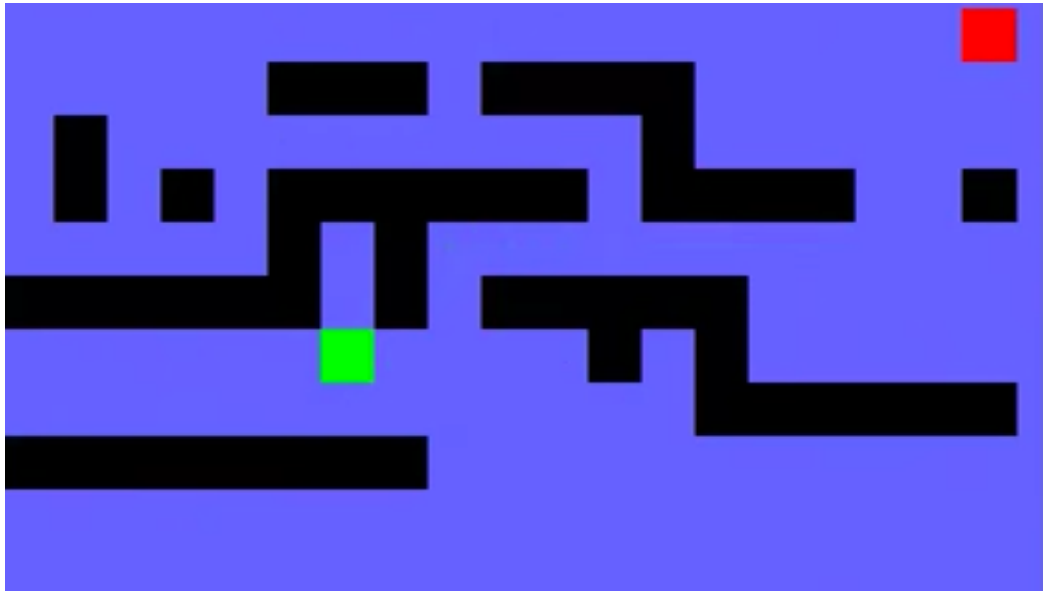
DFS vs BFS



DFS vs BFS

1. When will BFS outperform DFS?
2. When will DFS outperform BFS?

Some examples



BFS or DFS?

DFS

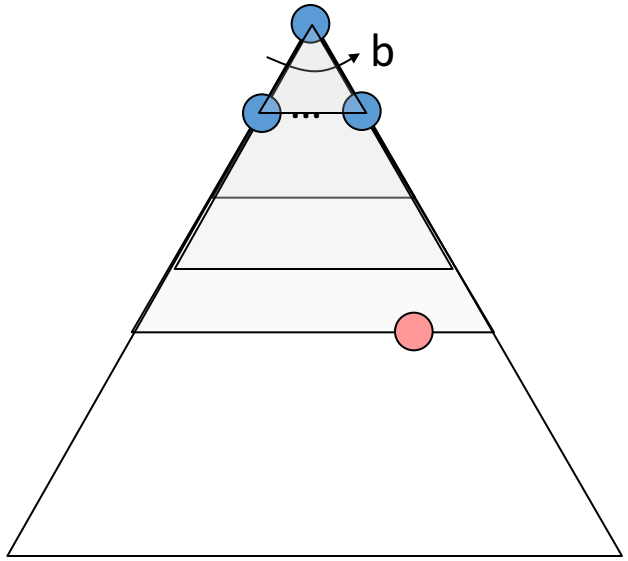
Hybrid approach: iterative deepening

Idea: get DFS's space advantage with BFS's time / shallow-solution advantages

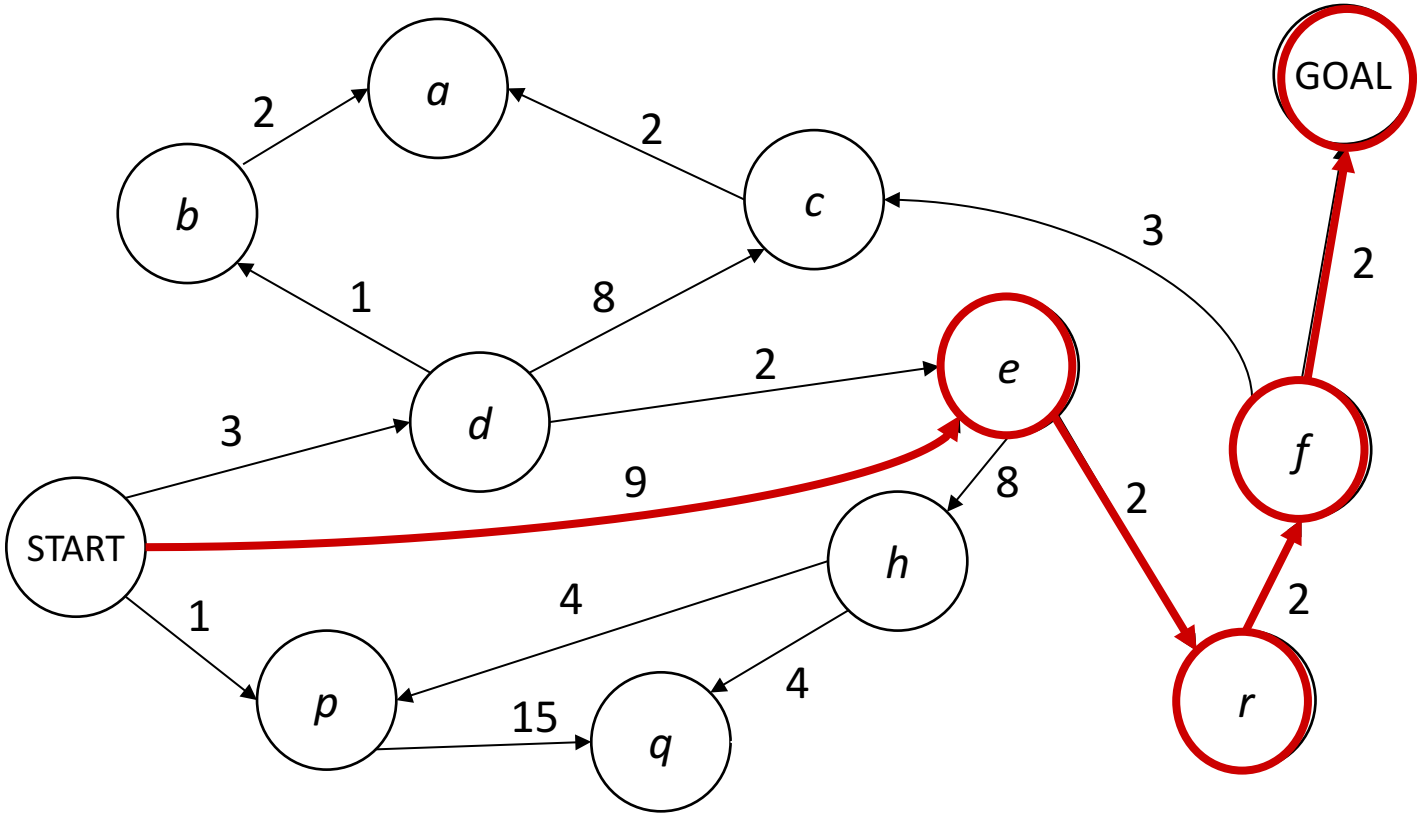
- Run a DFS with depth limit 1. If no solution…
- Run a DFS with depth limit 2. If no solution…
- Run a DFS with depth limit 3. …..

Isn't that wastefully redundant?

- Generally most work happens in the lowest level searched, so not so bad!

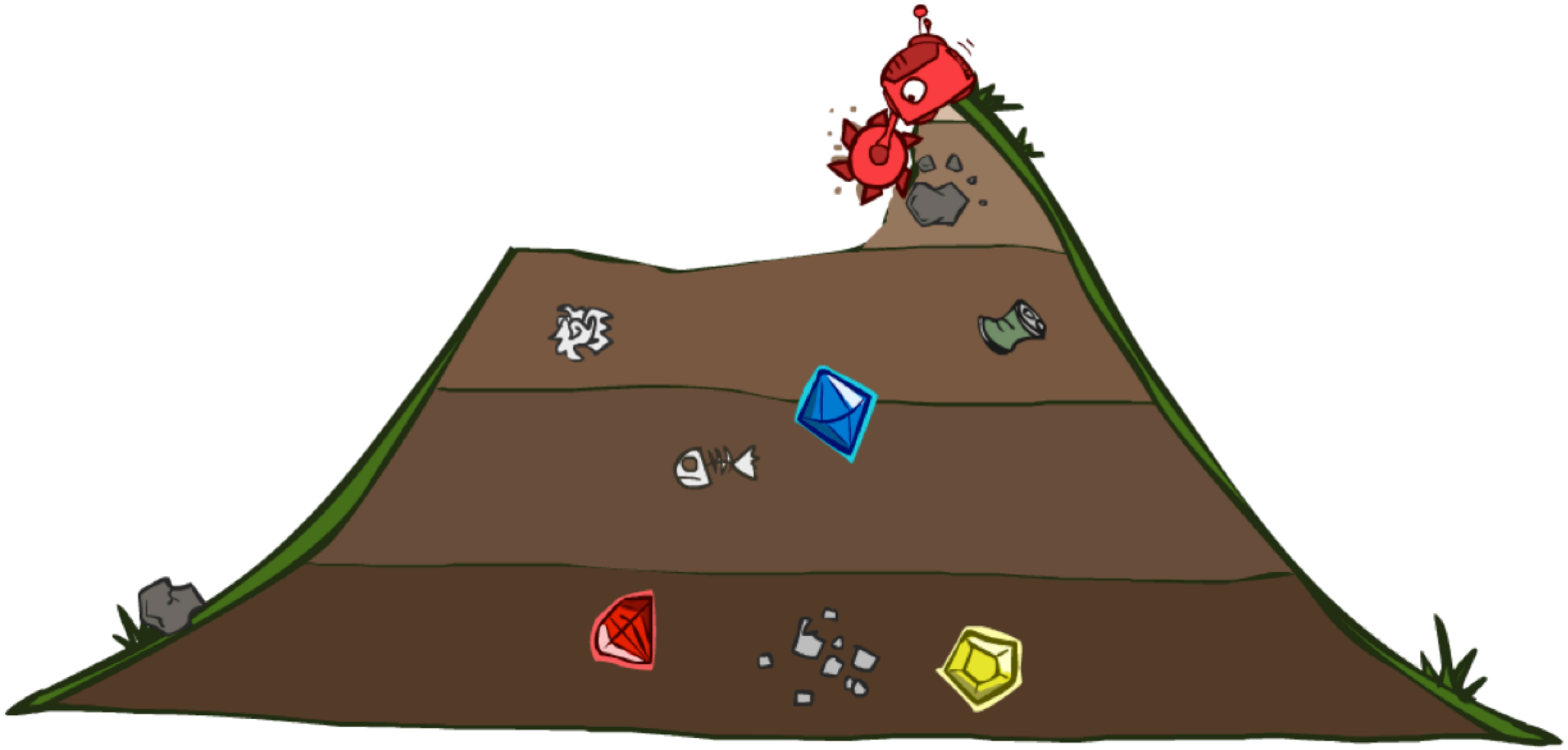


Cost-sensitive search



BFS finds the shortest path in terms of number of actions.
But it does not find the least-cost path.

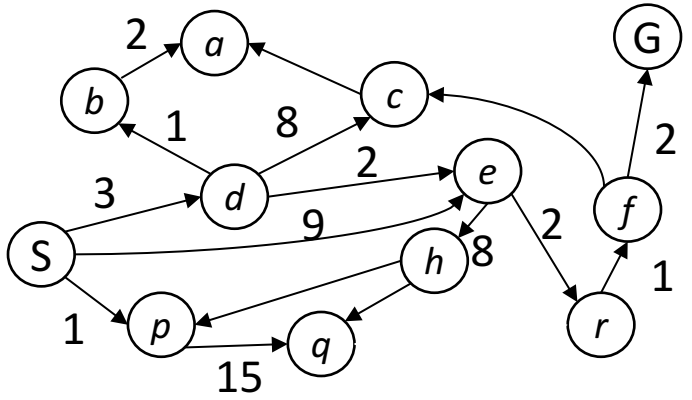
Uniform Cost Search (UCS)



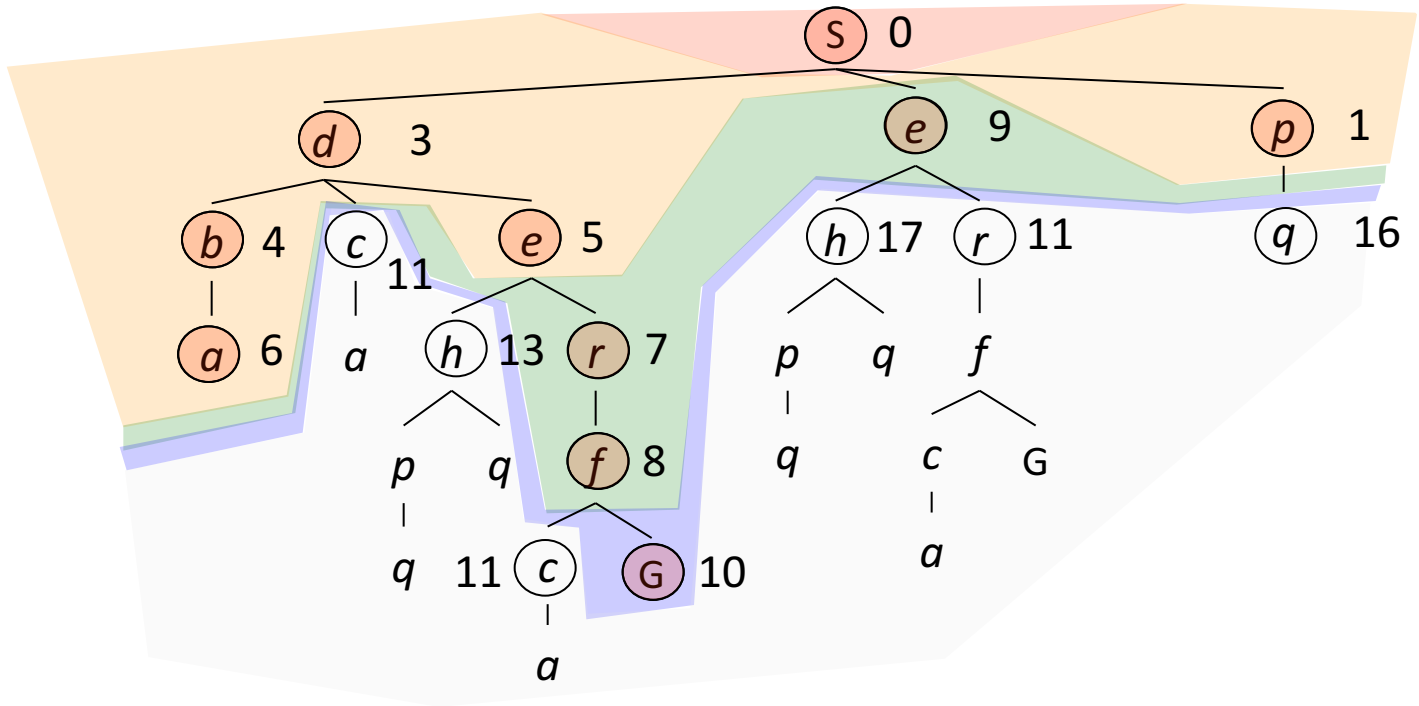
Uniform Cost Search (UCS)

Strategy *expand a cheapest node first:*

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



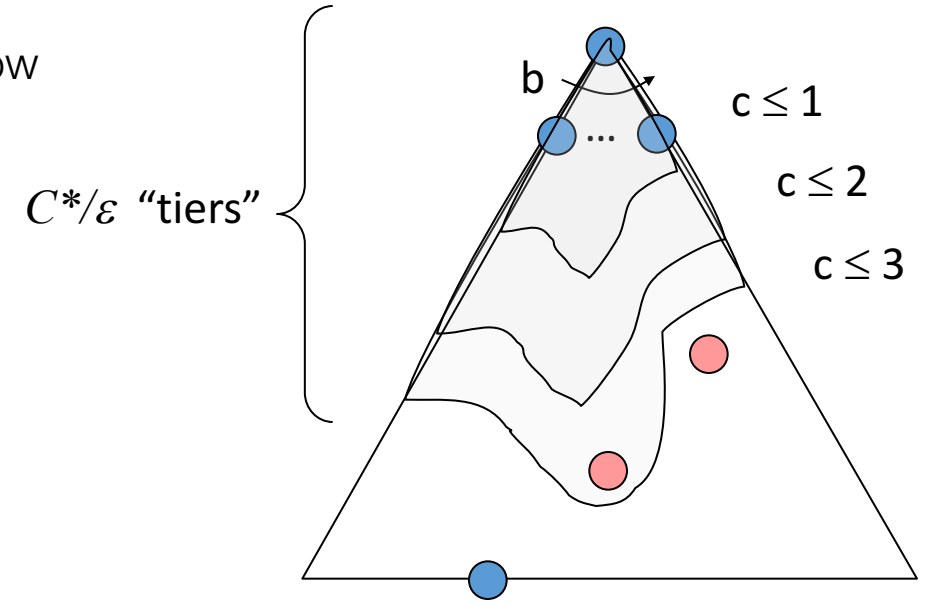
Cost contours



Uniform Cost Search (UCS) properties

What nodes does UCS expand?

- Processes **all nodes with cost less than cheapest solution**!
- **Assume** optimal solution costs C* and all arcs cost at least ε. How deep do we explore???
- "effective depth" is roughly C^*/ε
- So we'd expand $O(b^{C^*/\varepsilon})$ nodes (exponential in effective depth)



How much space does the fringe take?

- Has roughly the last tier, so $O(b^{C^*/\varepsilon})$

Is it complete?

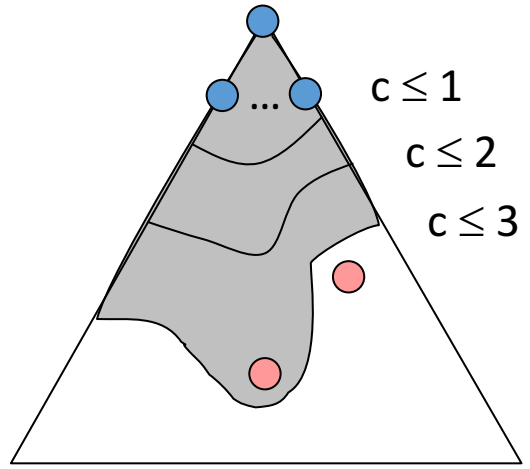
- Assuming best solution has a finite cost and minimum arc cost is positive, yes!

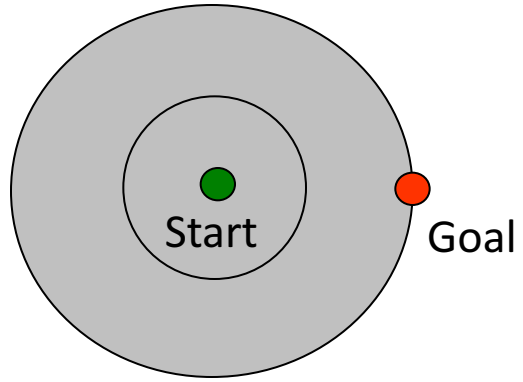
Is it optimal?

- 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!

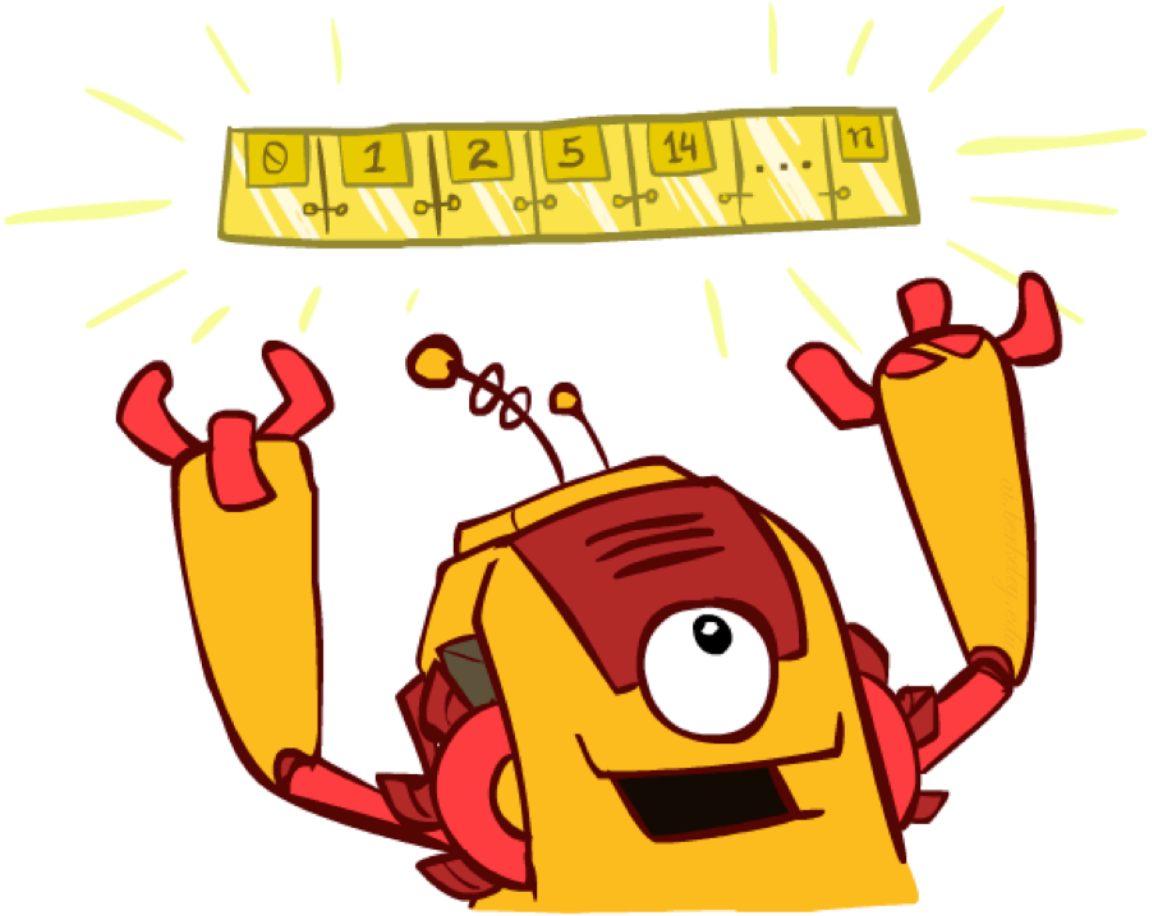




The one queue

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

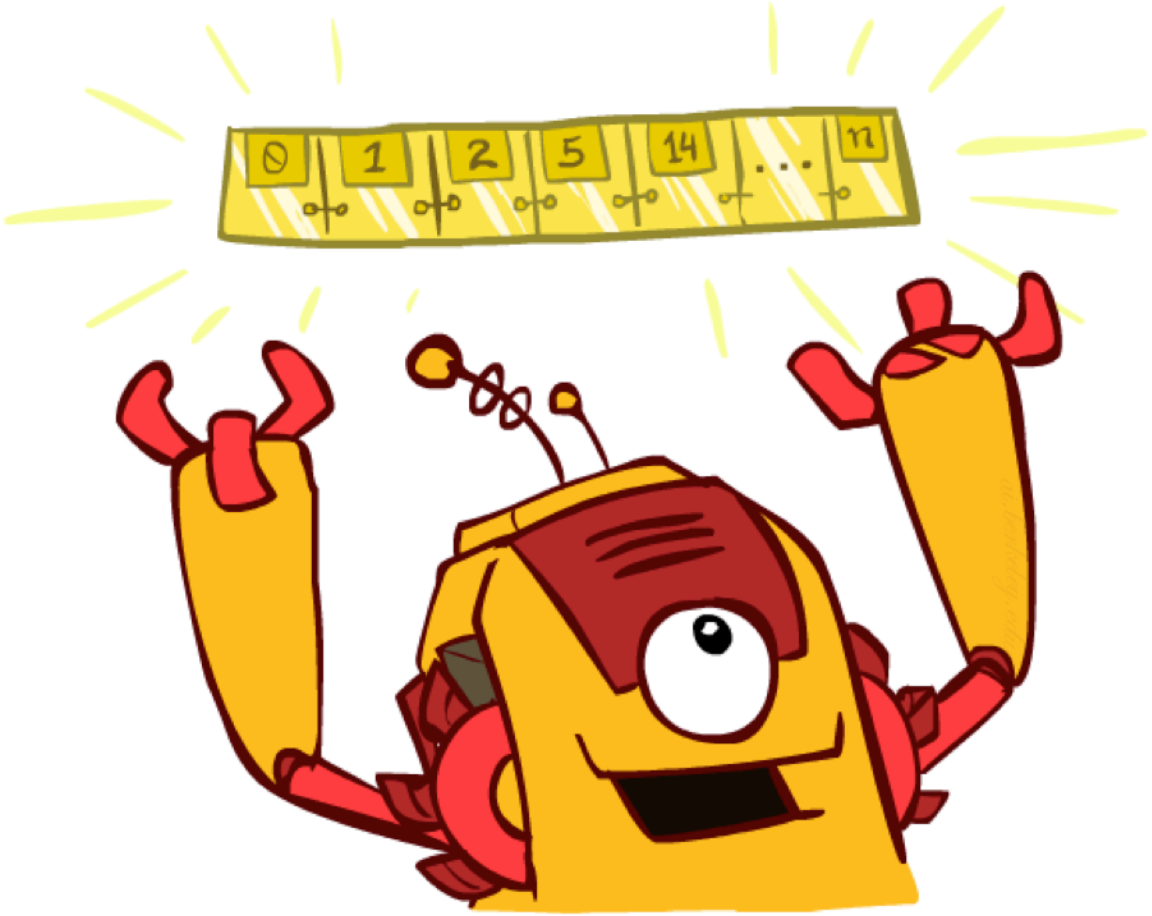


The one queue

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

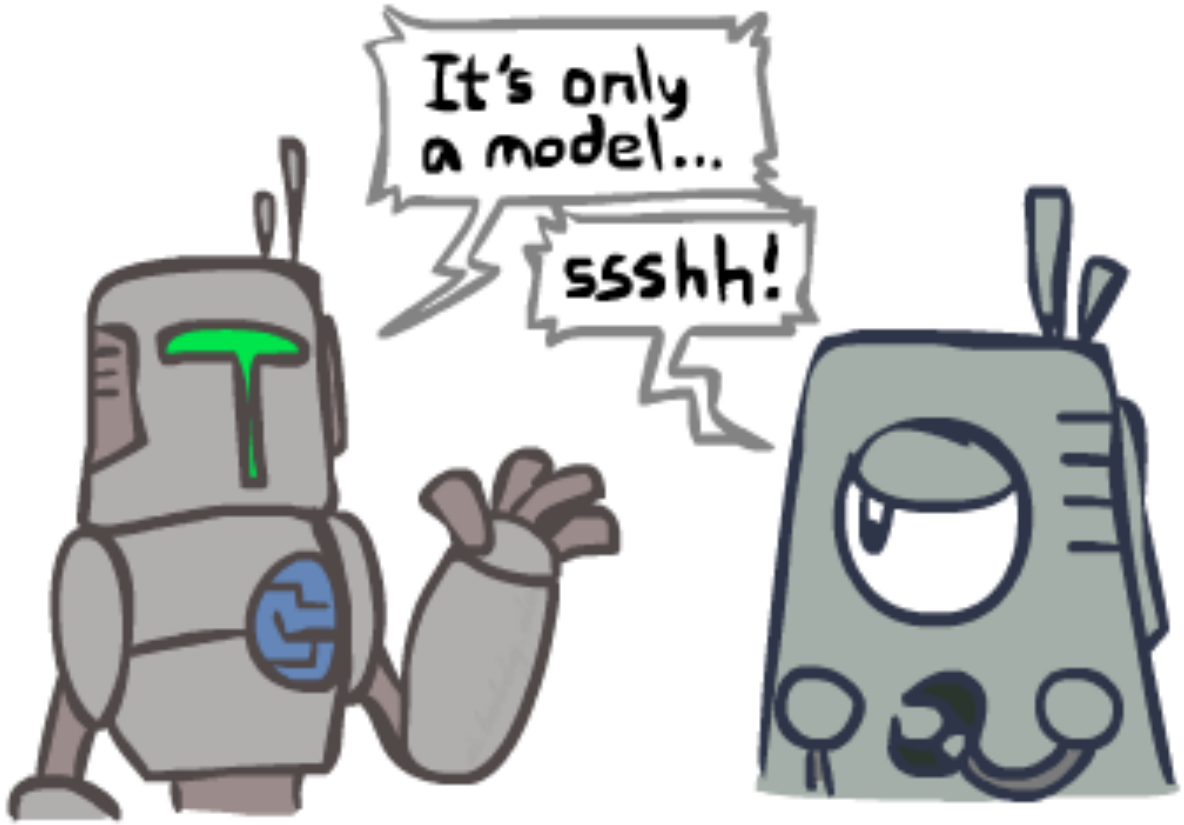
You'll learn this first-hand in HW 1!



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…



That’s it for today!

Next week: smarter search; *A** and beyond

HW1 is available on the website -- start early! Due in two weeks.

Rohan has office hours Thursdays: 3-5p in 462 WVH. I have office hours Tuesdays after class (476 WVH).