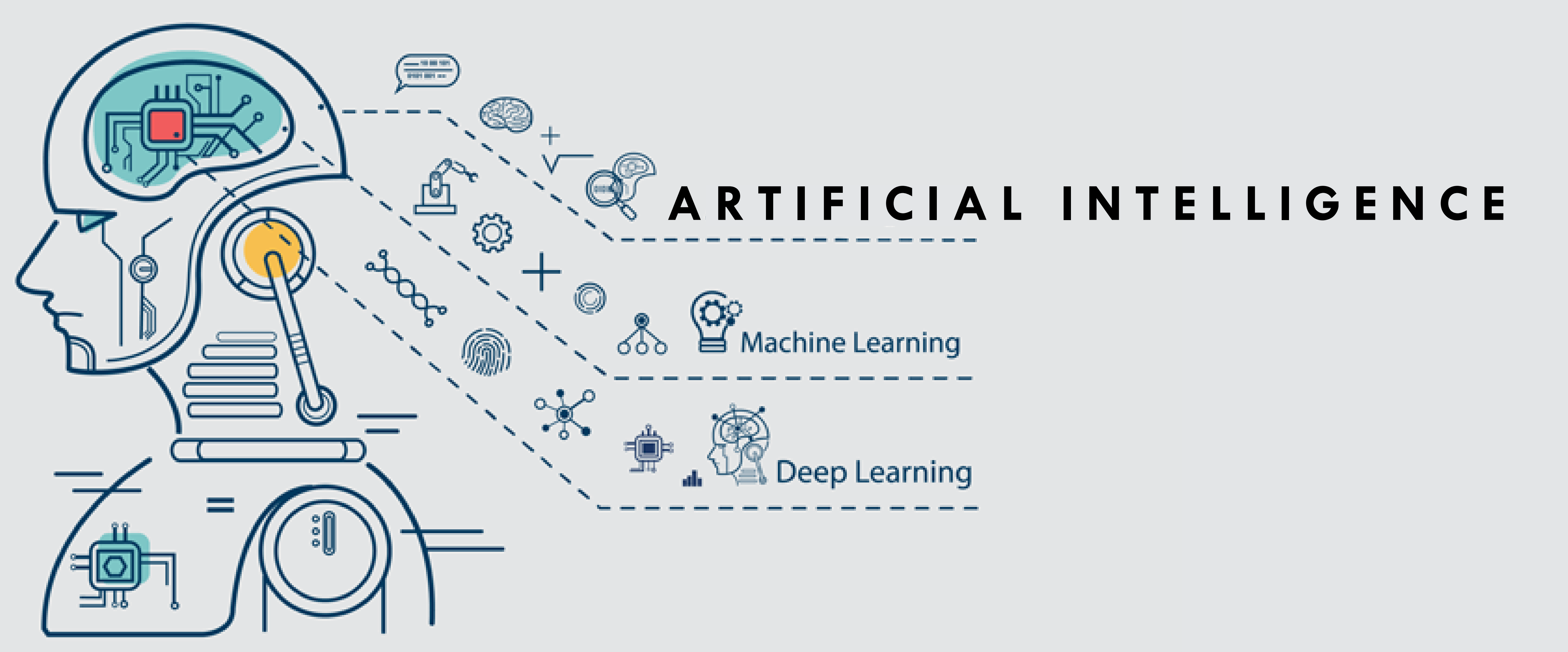


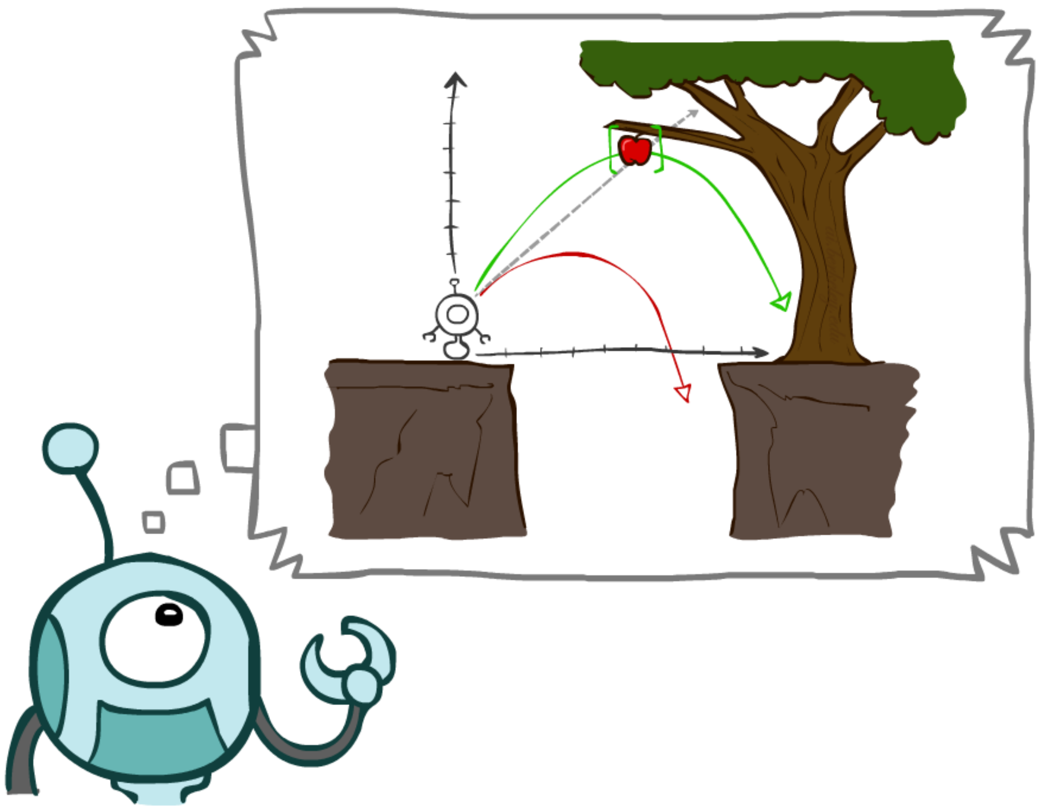
Dr. Seemab latif

Lecture 2

17th Sept 2024

Today

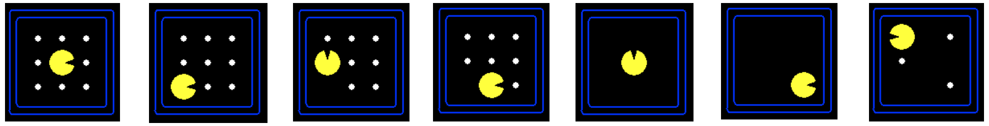
- Agents and Environment (Recap)
- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search



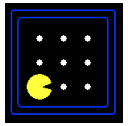
Search Problems

Search Problems

- A search problem consists of:
  - A state space

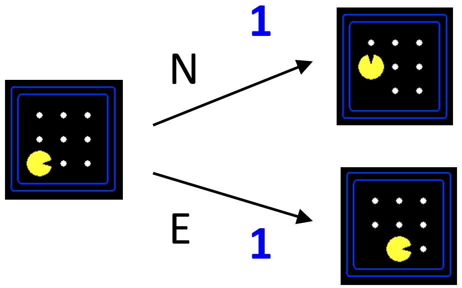


  - For each state, a set Actions(s) of allowable actions



{N, E}

  - A transition model Result(s,a)

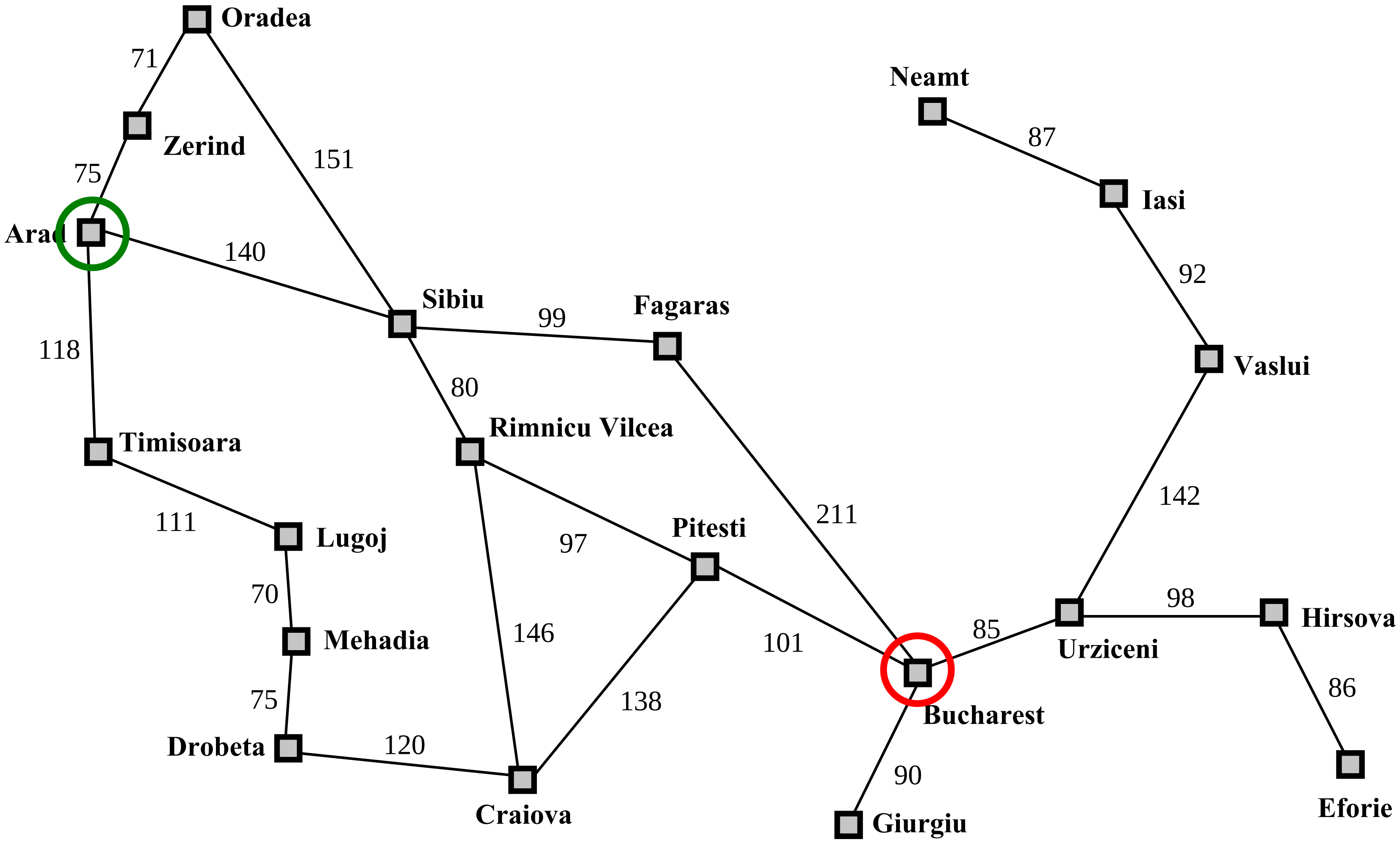


  - A step cost function c(s,a,s')
  - A start state and a goal test

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

Search Problems Are Models

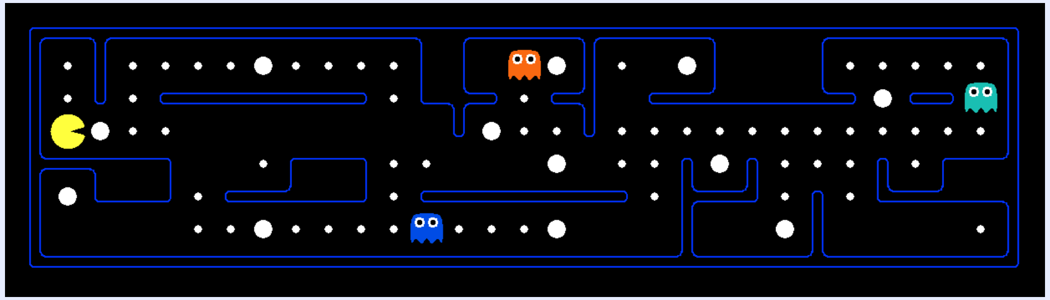
Example: Travelling in Romania



- State space:
  - Cities
- Actions:
  - Go to adjacent city
- Transition model
  - Result(A, Go(B)) = B
- Step cost
  - Distance along road link
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?

What’s in a State Space?

The real-world state includes every detail of the environment



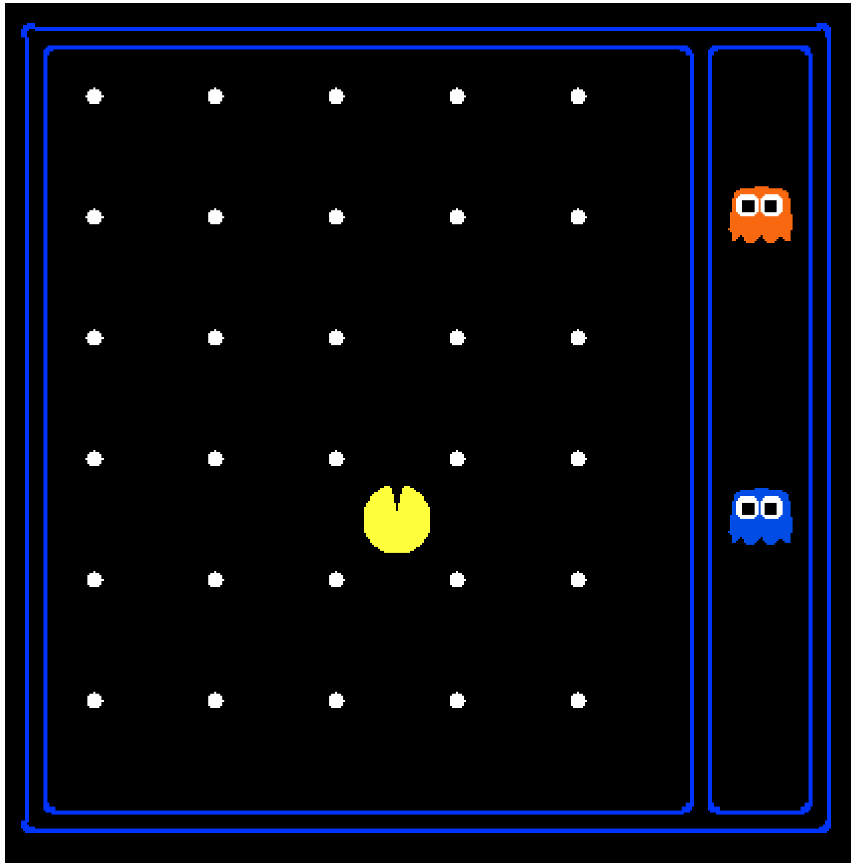
A search state abstracts away details not needed to solve the problem

- Problem: Pathing
  - State representation: (x,y) location
  - Actions: NSEW
  - Transition model: update location
  - Goal test: is (x,y)=END
- Problem: Eat-All-Dots
  - State representation: {(x,y), dot booleans}
  - Actions: NSEW
  - Transition model: update location and possibly a dot boolean
  - Goal test: dots all false

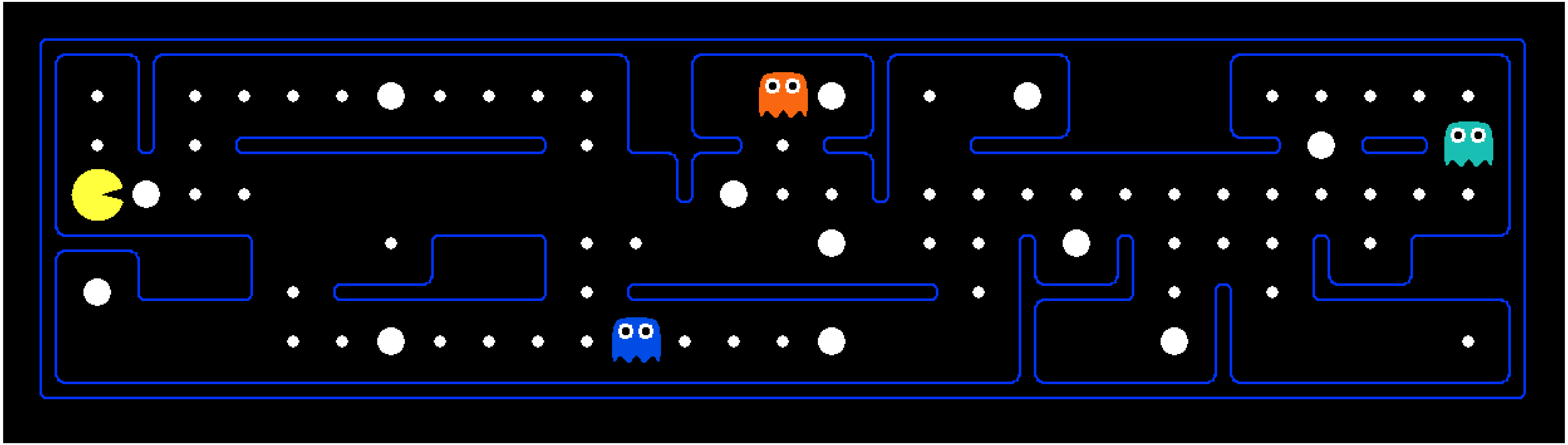
State Space Sizes?

- 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?
    120
  - States for eat-all-dots?
    $120 \times (2^{30})$

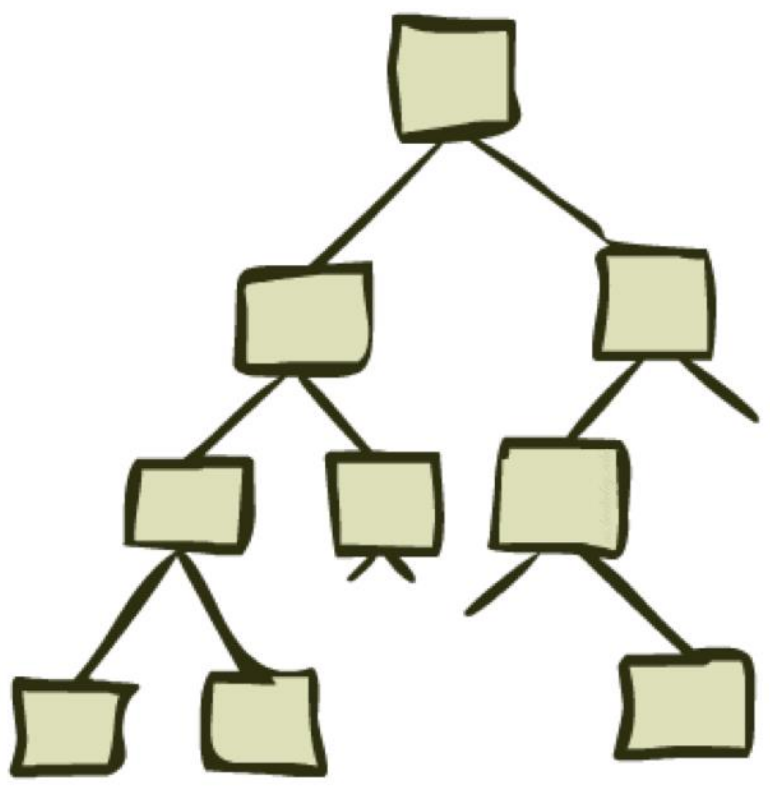


Safe Passage



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

State Space Graphs and Search Trees

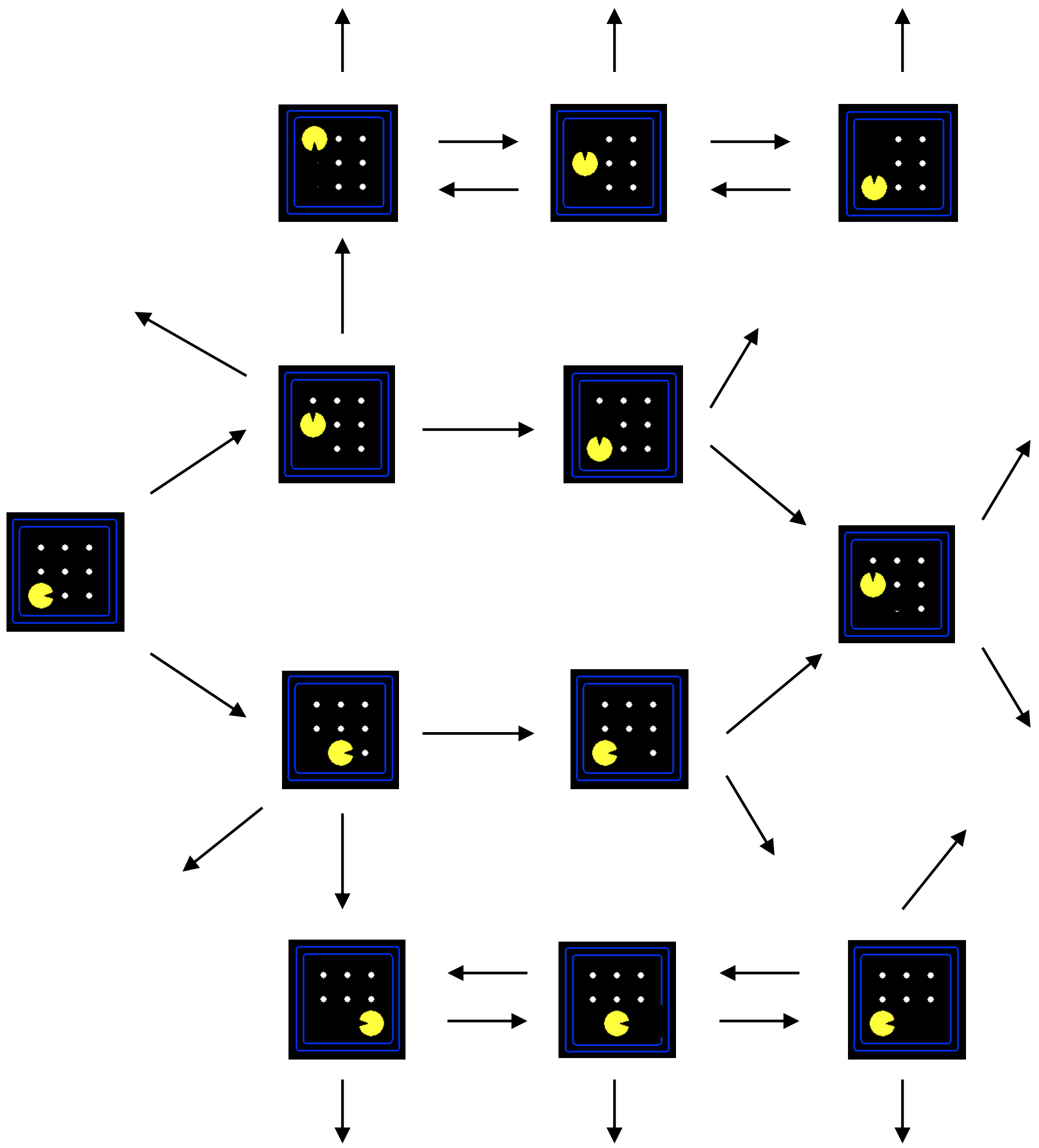


State Space Graphs

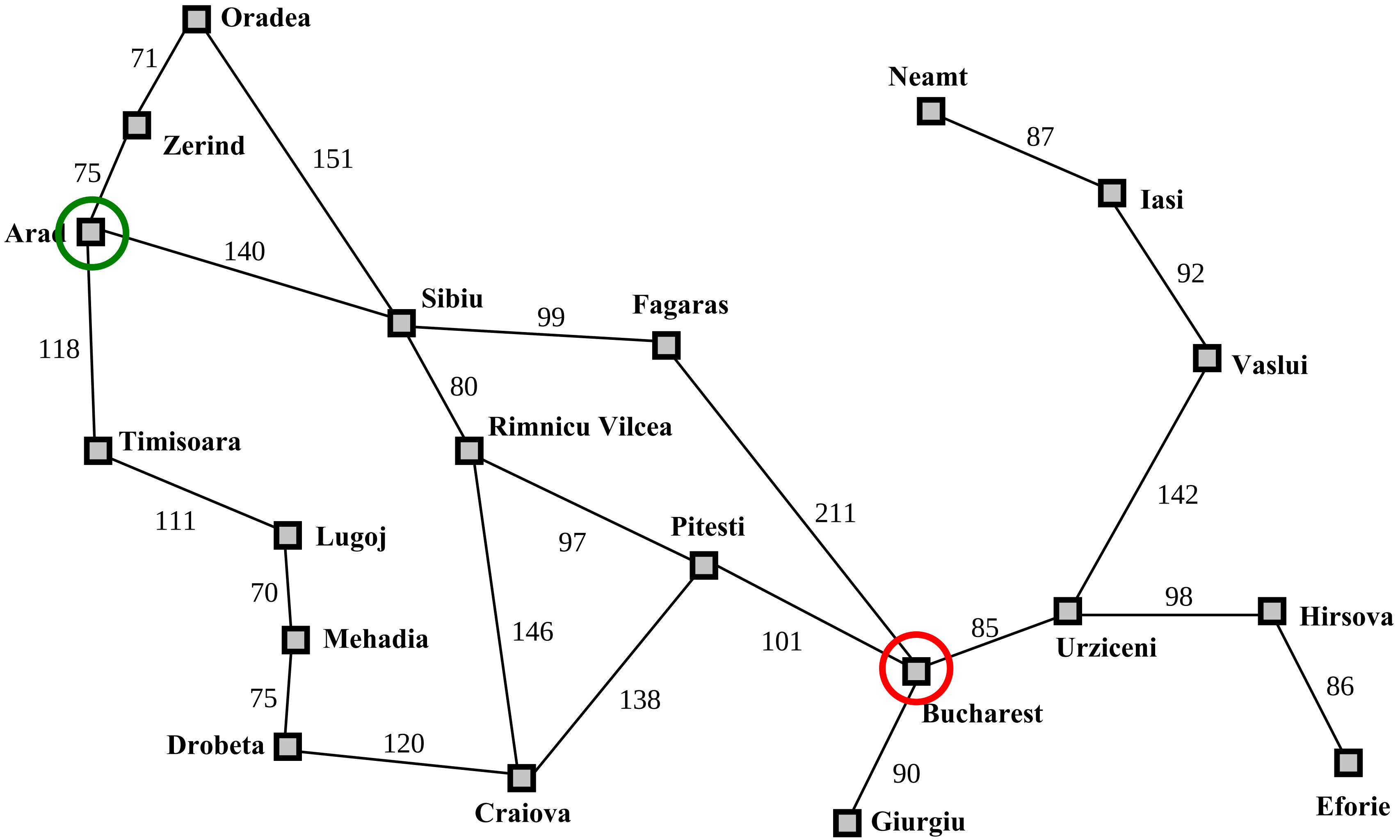
- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent transitions resulting from actions
  - The goal test is a set of goal nodes (maybe only one)

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

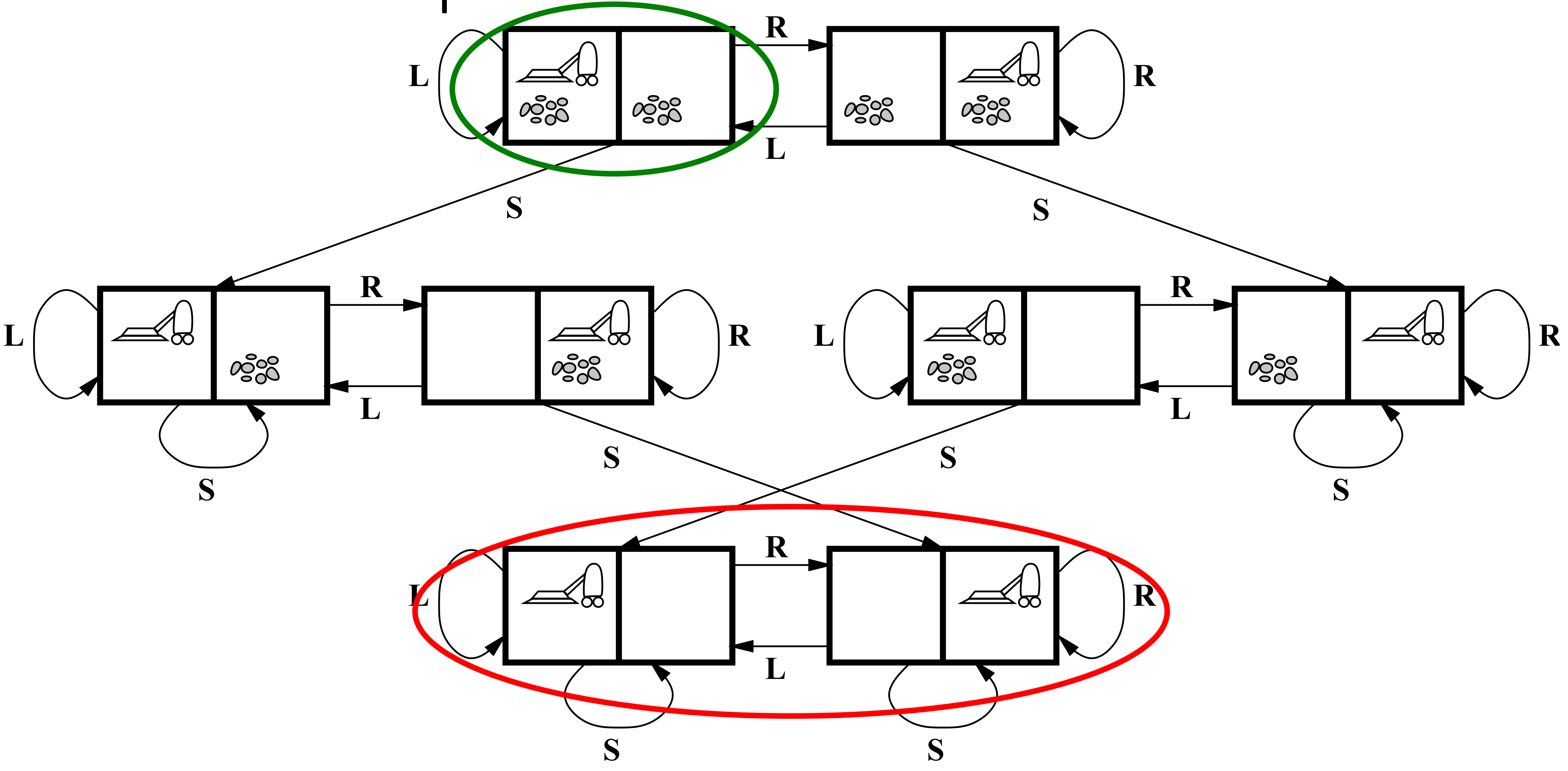
- We can rarely build this full graph in memory (it's too big), but it's a useful idea



More Examples

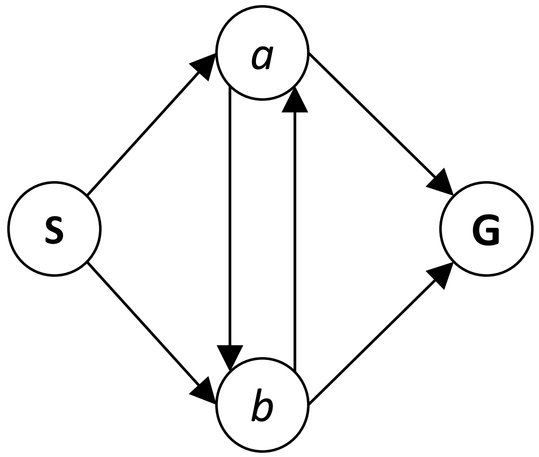


More Examples

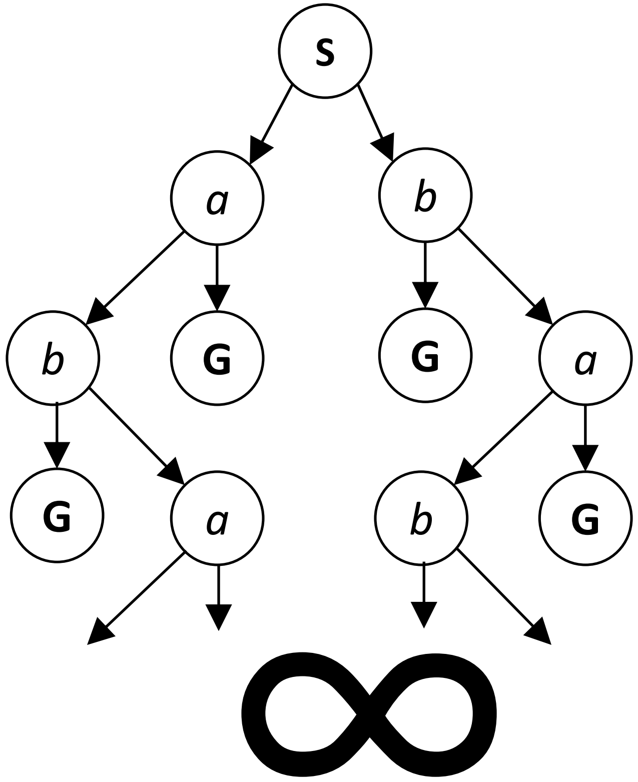


State Space Graphs vs. Search Trees

Consider this 4-state graph:

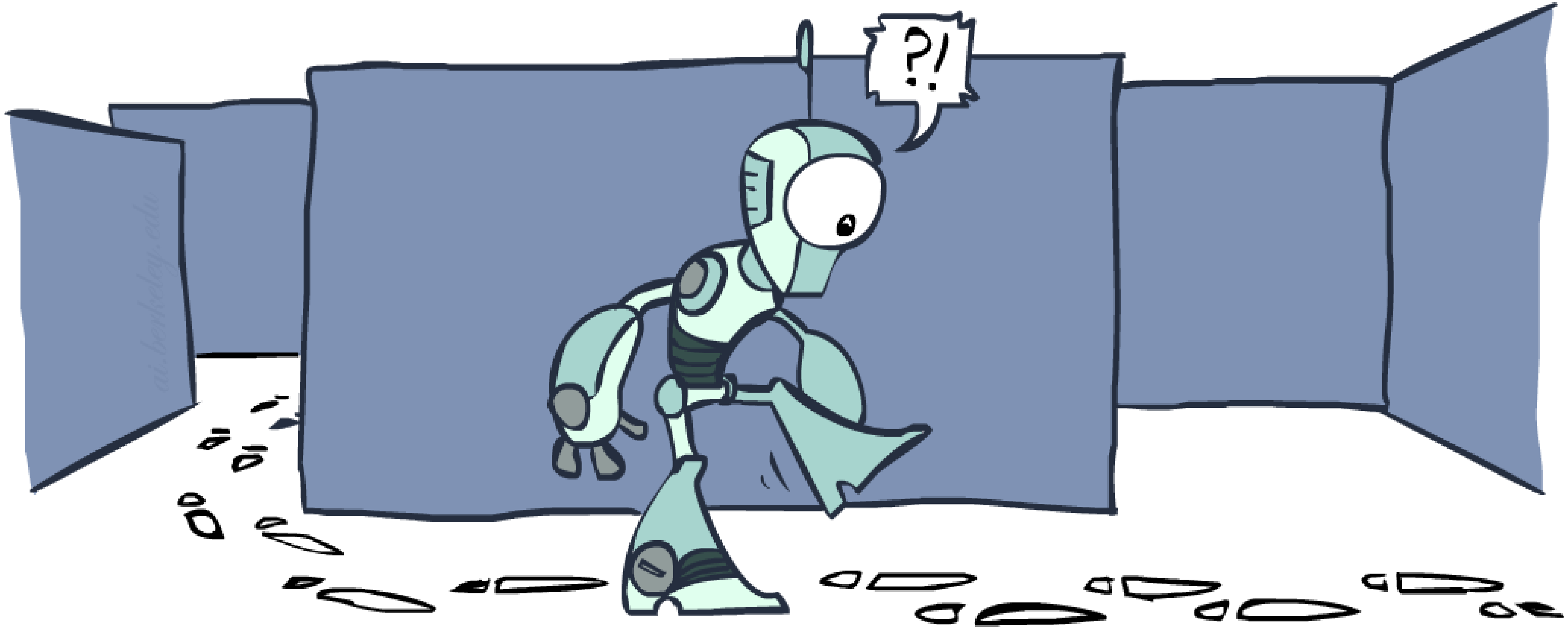


How big is its search tree (from S)?



Important: Lots of repeated structure in the search tree!

Tree Search vs Graph Search

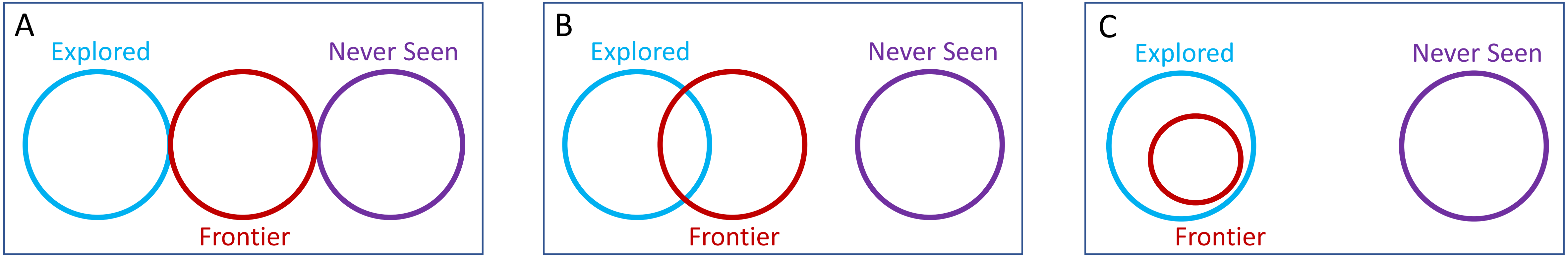


```
function TREE_SEARCH(problem) returns a solution, or failure

    initialize the frontier as a specific work list (stack, queue, priority queue)
    add initial state of problem to frontier
    loop do
        if the frontier is empty then
            return failure
        choose a node and remove it from the frontier
        if the node contains a goal state then
            return the corresponding solution

        for each resulting child from node
            add child to the frontier
```

```
function GRAPH_SEARCH(problem) returns a solution, or failure
    initialize the explored set to be empty
    initialize the frontier as a specific work list (stack, queue, priority queue)
    add initial state of problem to frontier
    loop do
        if the frontier is empty then
            return failure
        choose a node and remove it from the frontier
        if the node contains a goal state then
            return the corresponding solution
        add the node state to the explored set
        for each resulting child from node
            if the child state is not already in the frontier or explored set then
                add child to the frontier
```

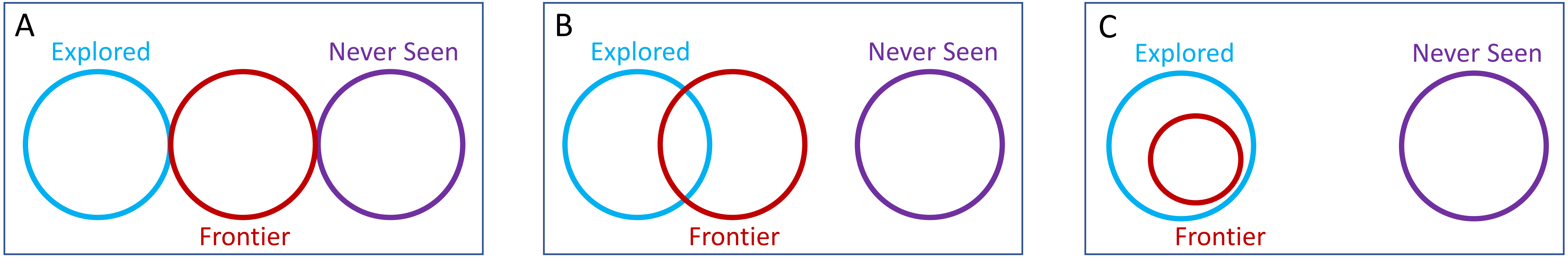
2

- What is the relationship between these sets of states after each loop iteration in GRAPH_SEARCH?
- (Loop invariants!!!)



POLL 2-answer

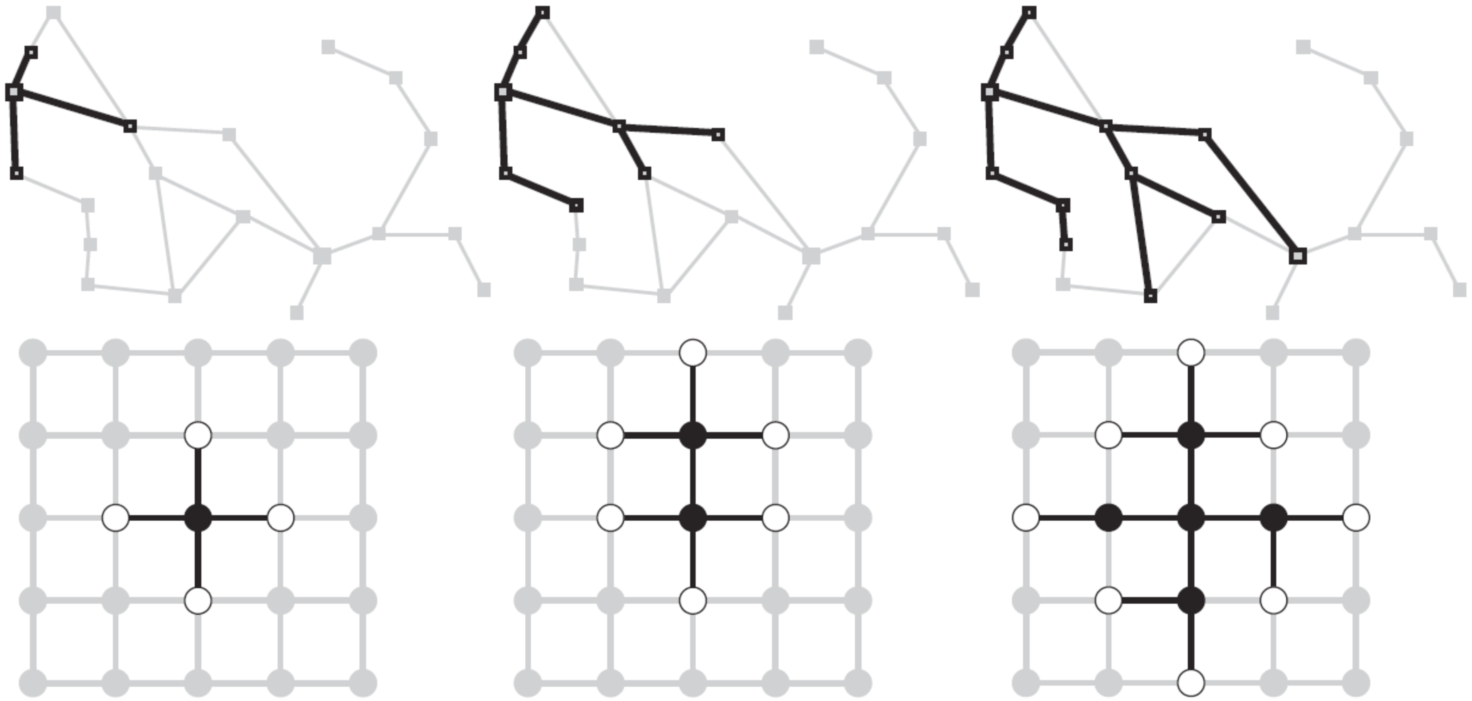
- What is the relationship between these sets of states after each loop iteration in GRAPH_SEARCH?
- (Loop invariants!!!)



- The frontier states separate the explored states from never seen states
- Frontier is sub-set of Explored, as loop progresses, number of explored states will be more than the number of states infrontier
- Nodes that are not explored (Never Seen) are distinct from the other two

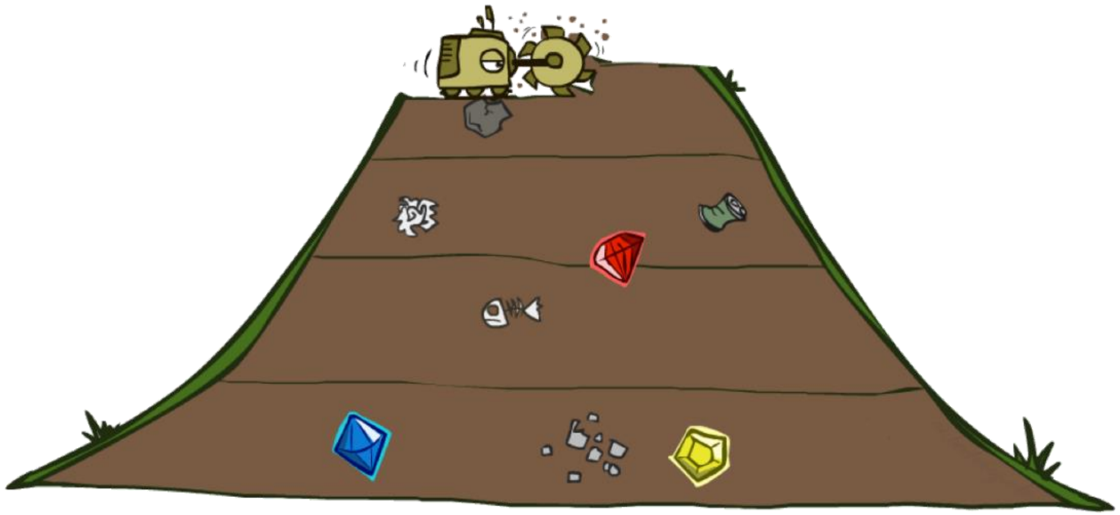
Graph Search

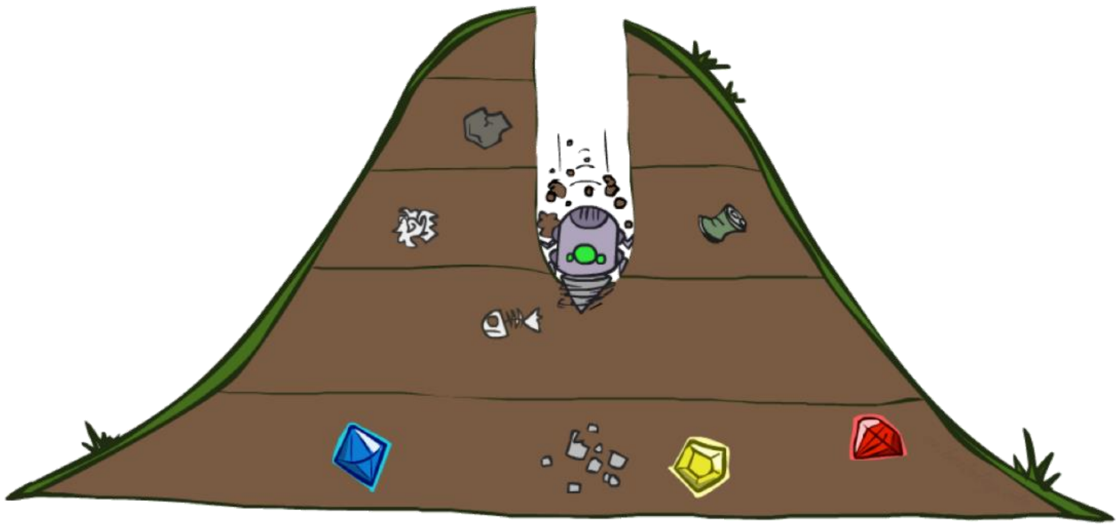
- This graph search algorithm overlays a tree on a graph
- The frontier states separate the explored states from never seen states



Images: AIMA, Figure 3.8, 3.9

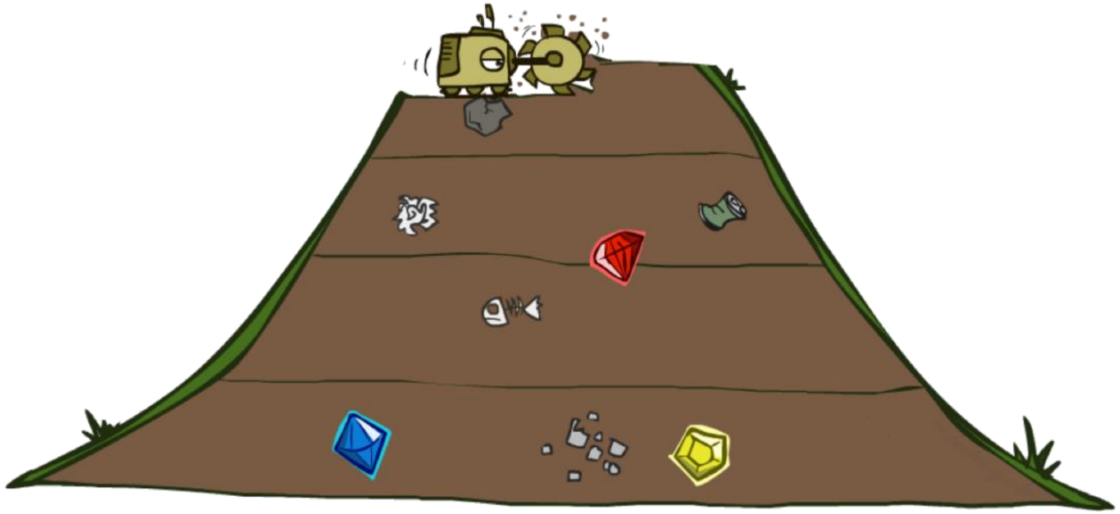
BFS vs DFS

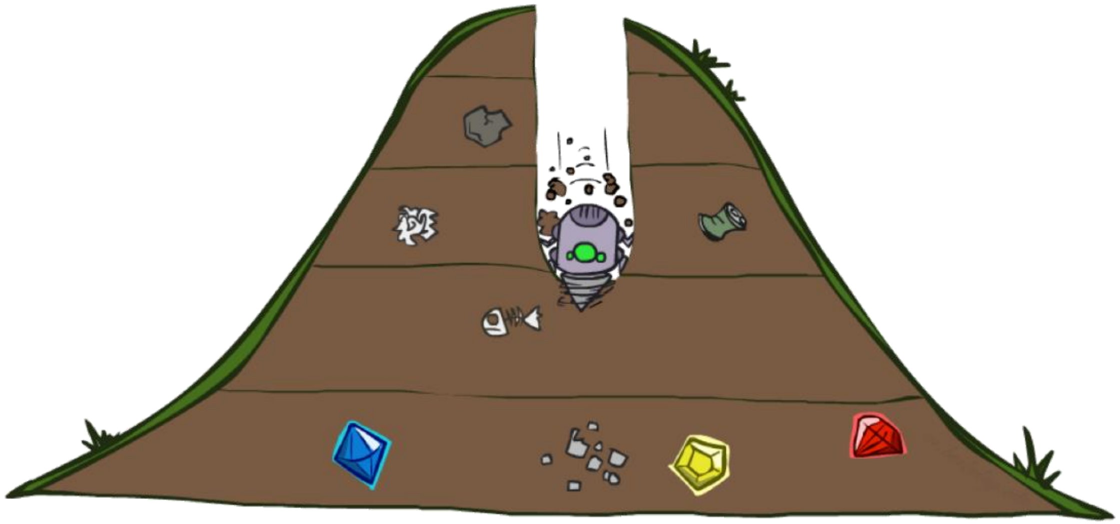




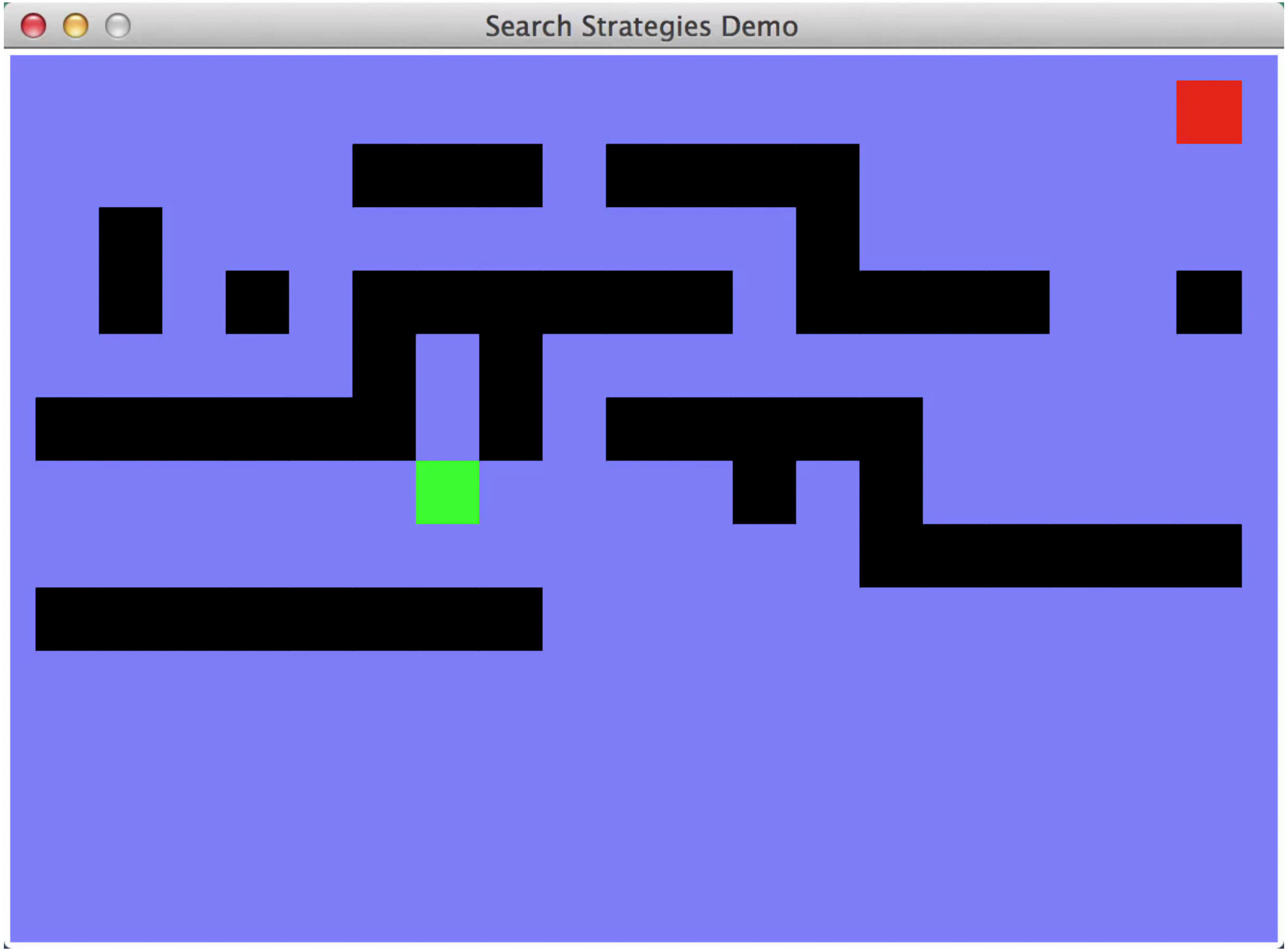
POLL 3

- Is the following demo Part 1 using BFS or DFS

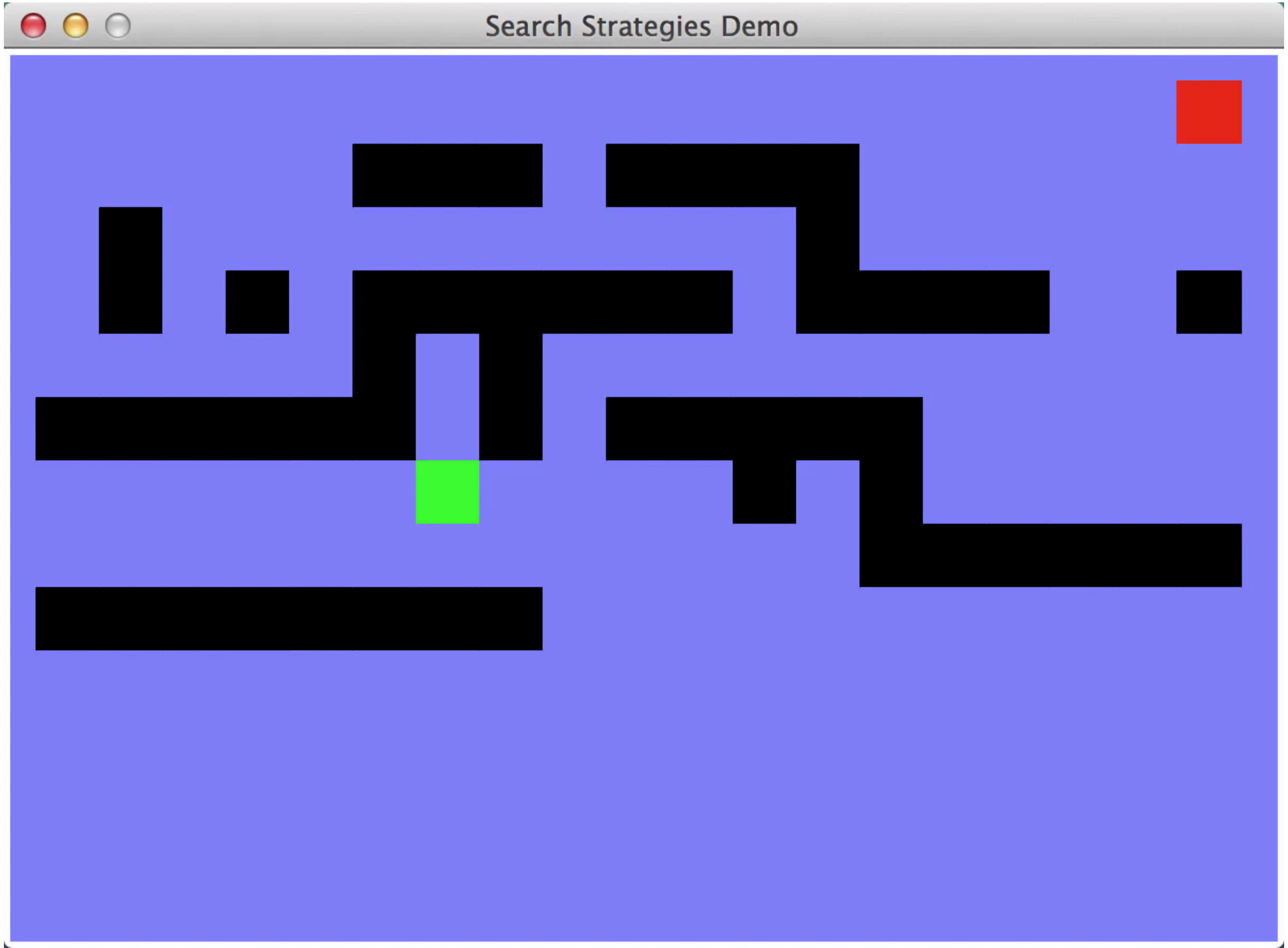




Video of Demo Maze Water DFS/BFS (part 1)



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



A Note on Implementation

Nodes have

state, parent, action, path-cost

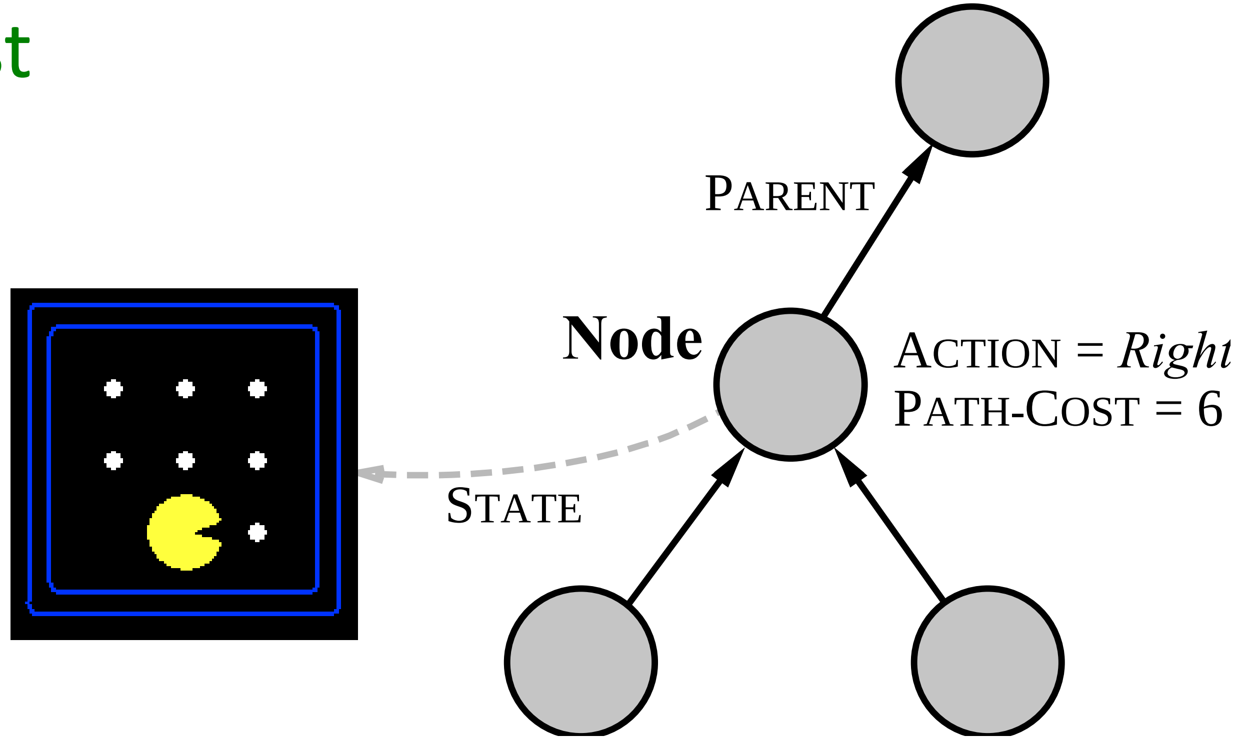
A child of node by action *a* has

state = result(node.state,*a*)

parent = node

action = *a*

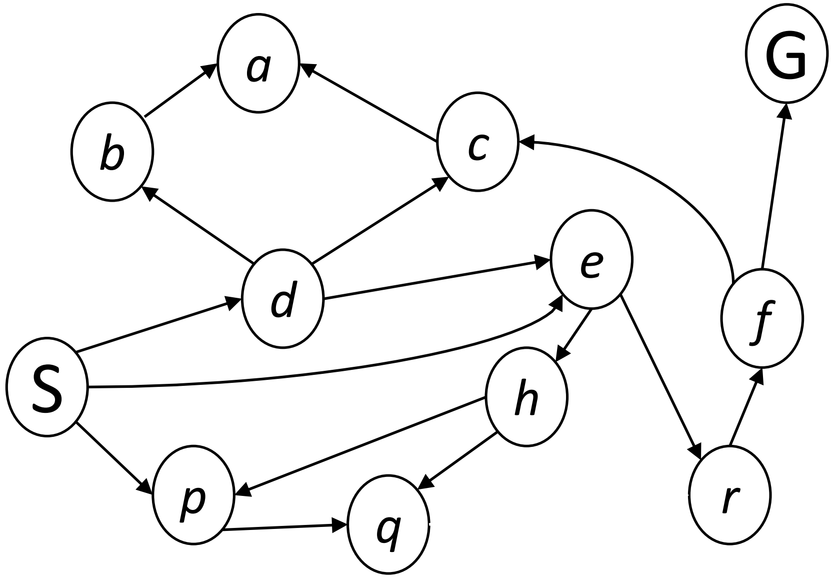
path-cost = node.path_cost + step_cost(node.state, *a*, self.state)



Extract solution by tracing back parent pointers, collecting actions

Q,Q,R,E,P	P	S,D,B,A,C,E,H	S-D-B-A-C-E-H-P
Q,R,E,P	Q	S,D,B,A,C,E,H,P,	S-D-B-A-C-E-H-P-Q
R,E,P	R	S,D,B,A,C,E,H,P,Q	S,D,B,A,C,E,H,P,Q,R
F,E,P	F	S,D,B,A,C,E,H,P,Q,F	S,D,B,A,C,E,H,P,Q,R,F
G,E,P	G=Goal		

ch

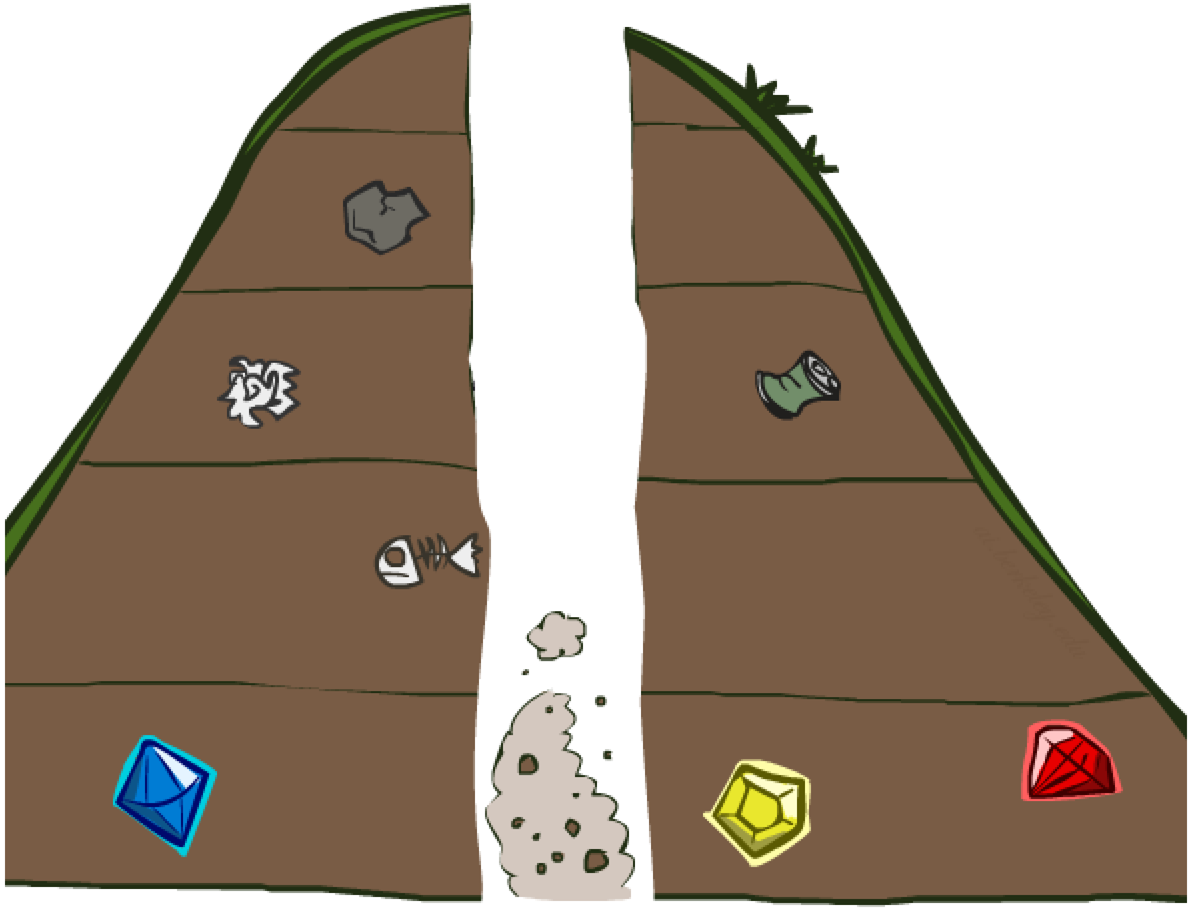


BFS vs DFS

- When will BFS outperform DFS?

- When will DFS outperform BFS?

Search Algorithm Properties

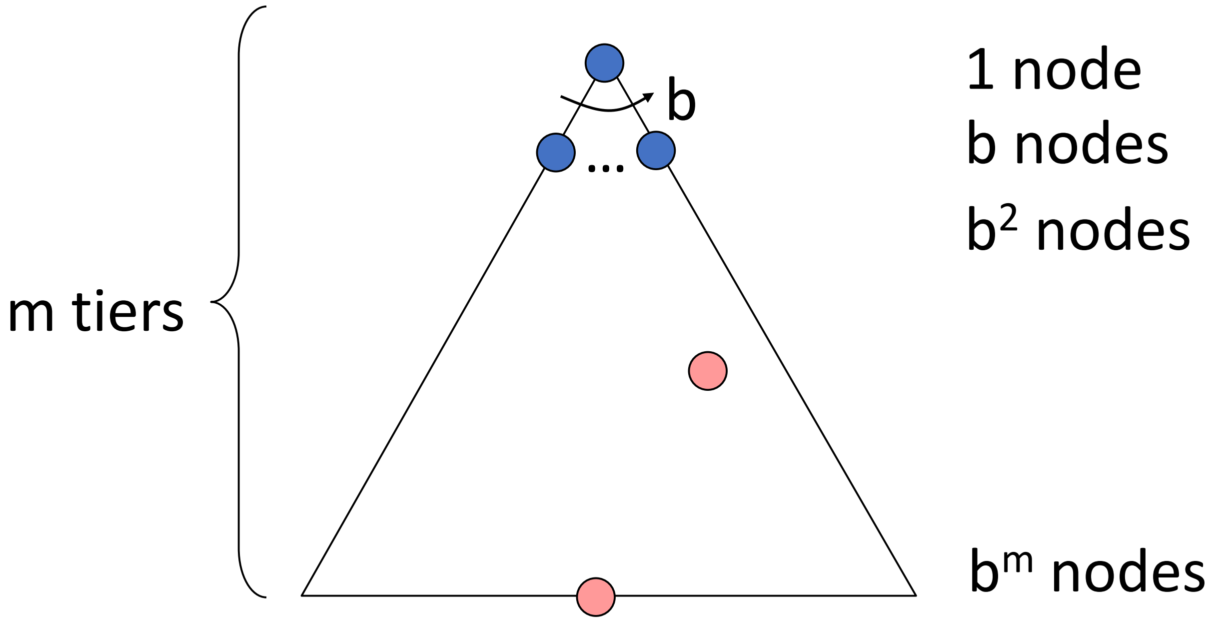


Search Algorithm Properties

- 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

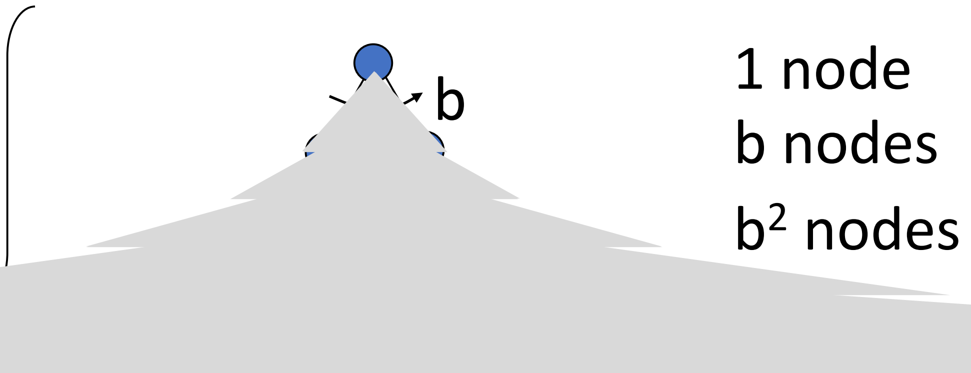
- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots.\ b^m = O(b^m)$



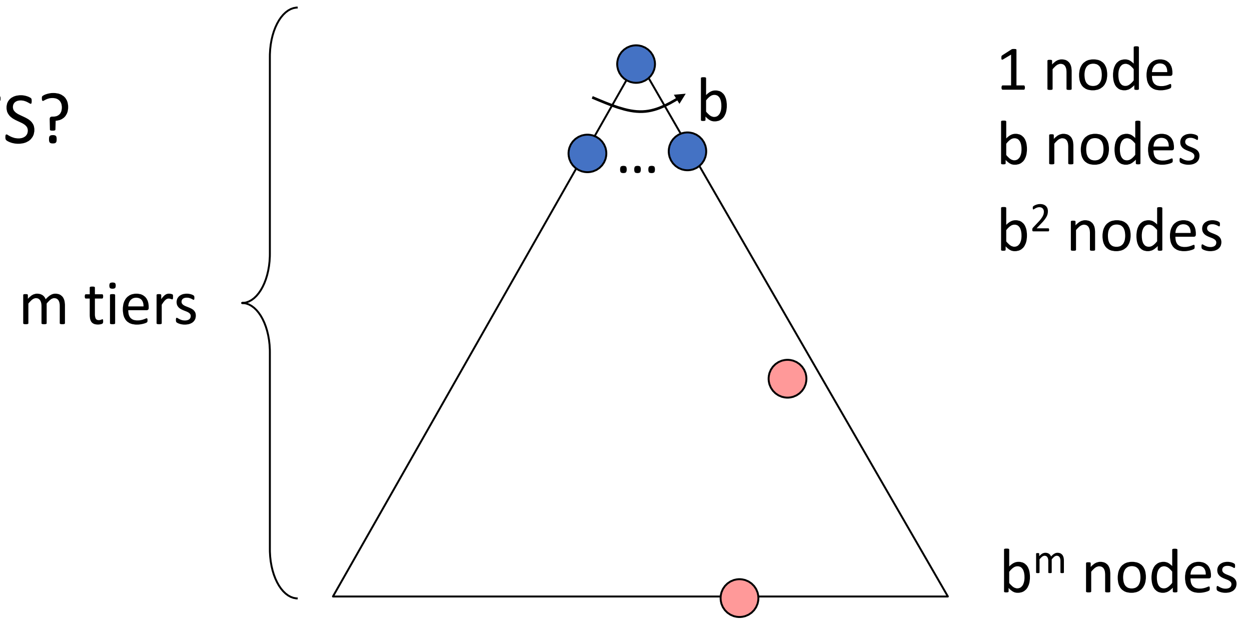
Search Algorithm Properties

- 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

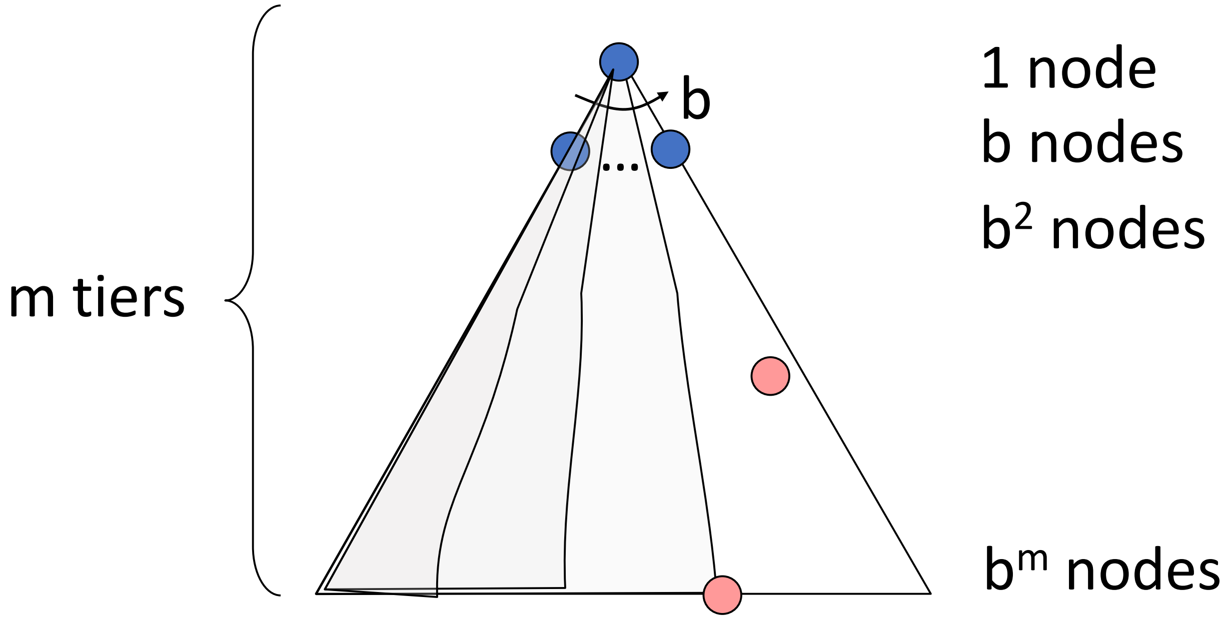


- Are these the properties for BFS or DFS?
  - Takes $O(b^m)$ time
  - Uses $O(bm)$ space on frontier
  - Complete with graph search
  - Not optimal unless all goals are in the same level (and the same step cost everywhere)



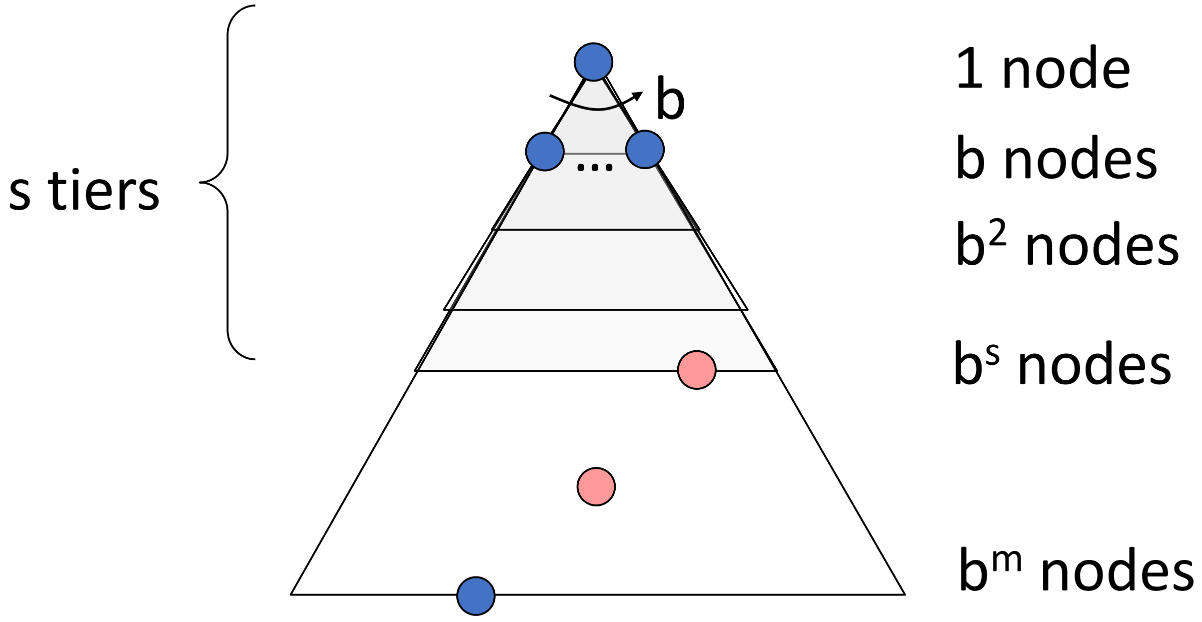
Depth-First Search (DFS) Properties

- 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 space does the frontier take?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?
  - m could be infinite, so only if we prevent cycles (graph search)
- Is it optimal?
  - No, it finds the “leftmost” solution, regardless of depth or cost



Breadth-First Search (BFS) Properties

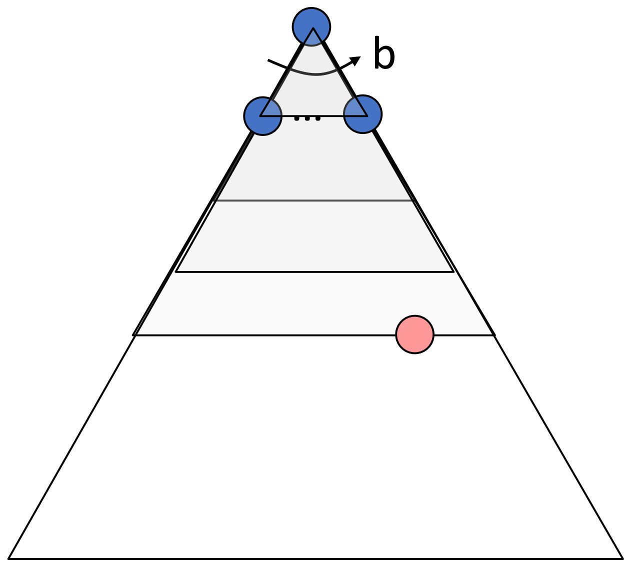
- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?
  - s must be finite if a solution exists, so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



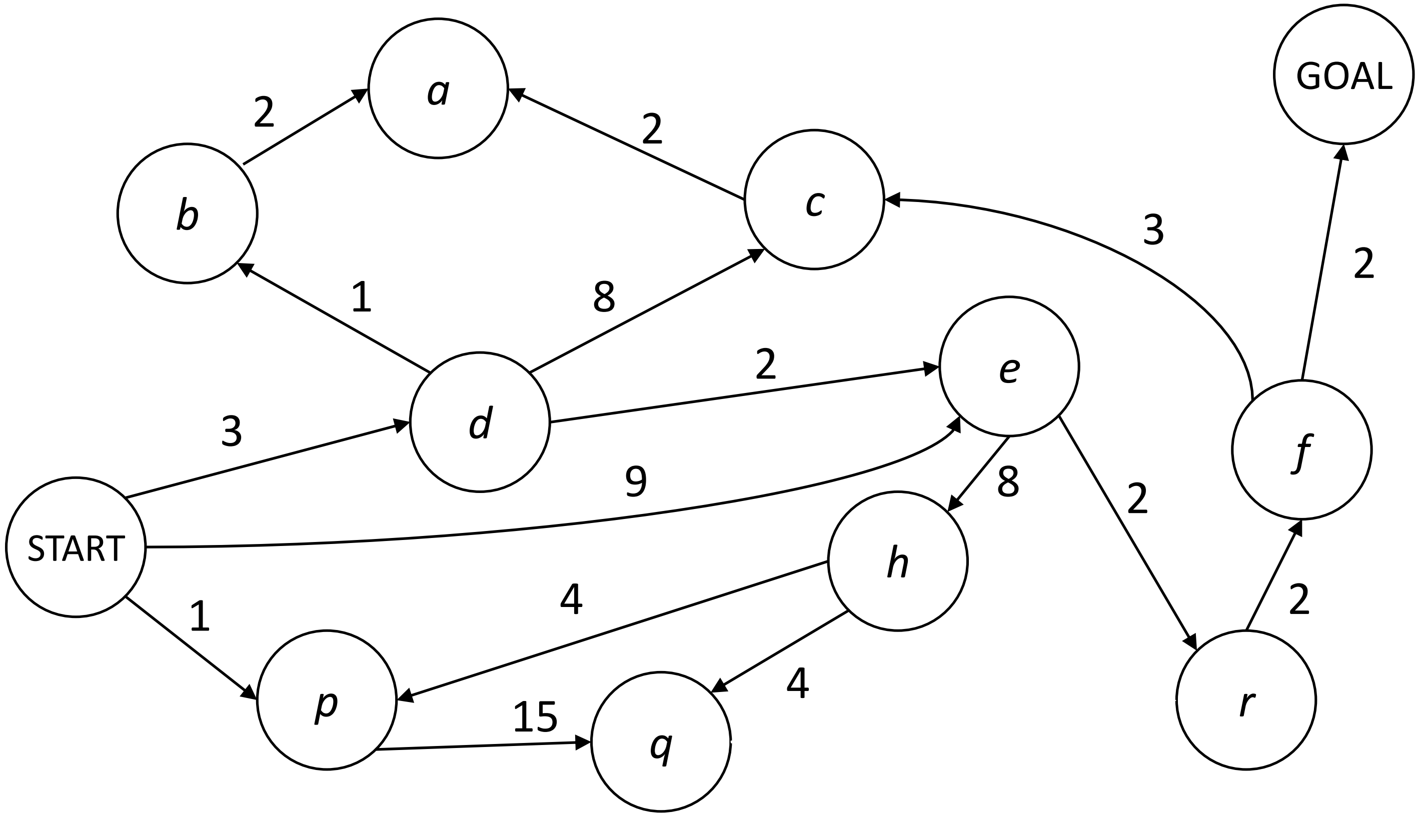
Iterative Deepening

- 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!



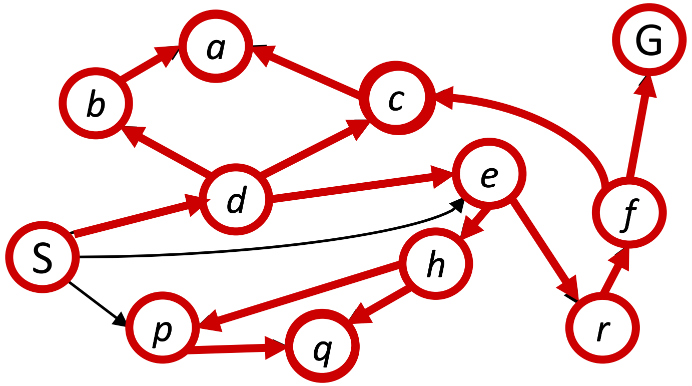
Finding a Least-Cost Path

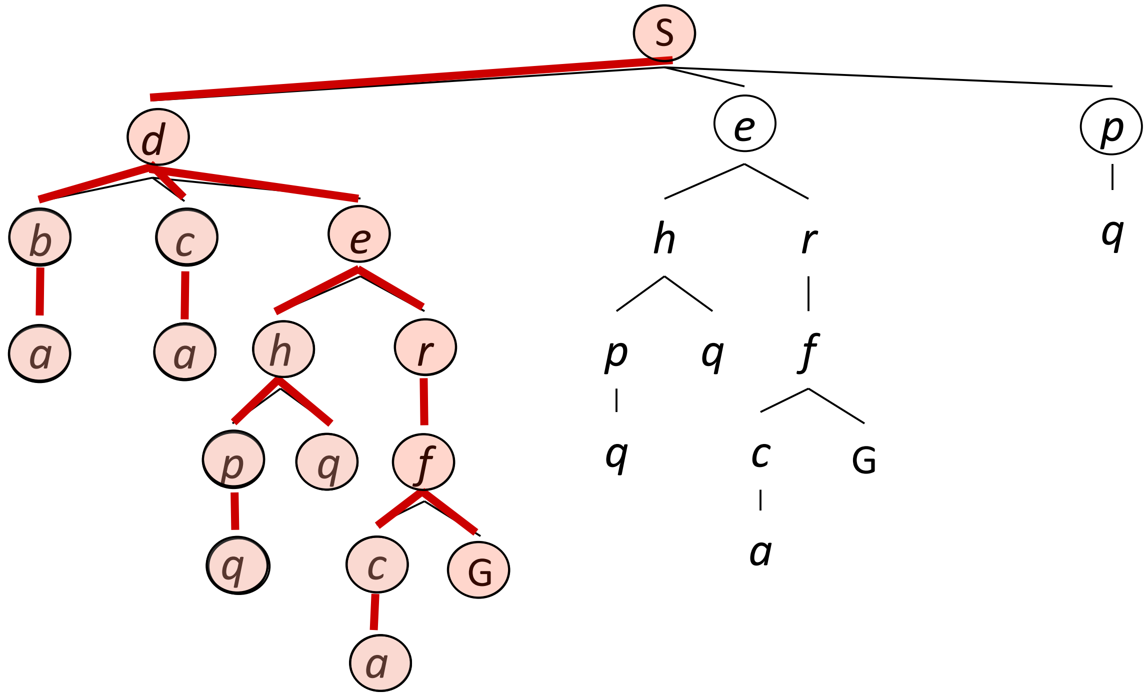


Depth-First (Tree) Search

Strategy: expand a deepest node first

Implementation:
Frontier is a LIFO stack

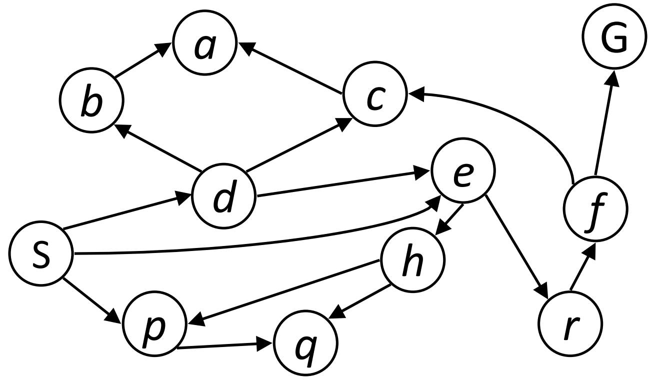


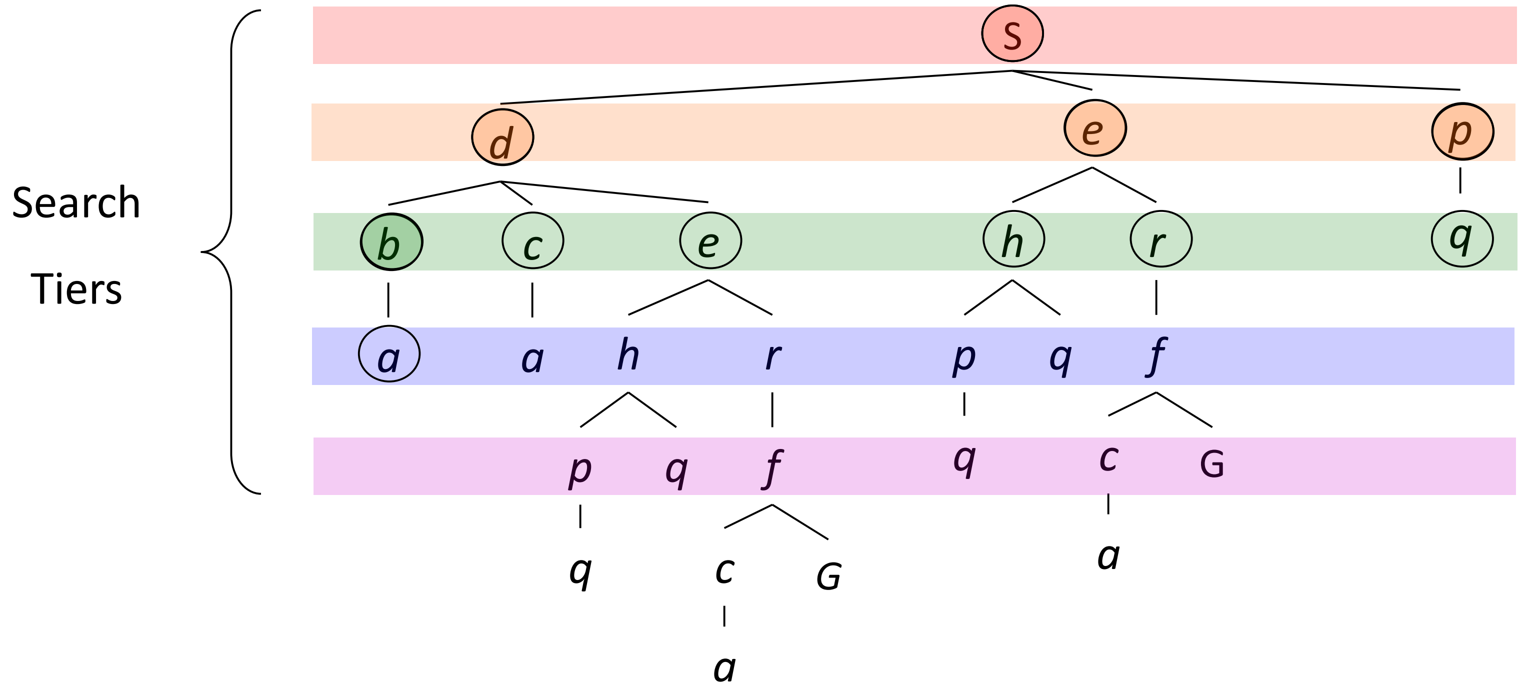


Breadth-First (Tree) Search

Strategy: expand a shallowest node first

Implementation:
Frontier is a FIFO queue

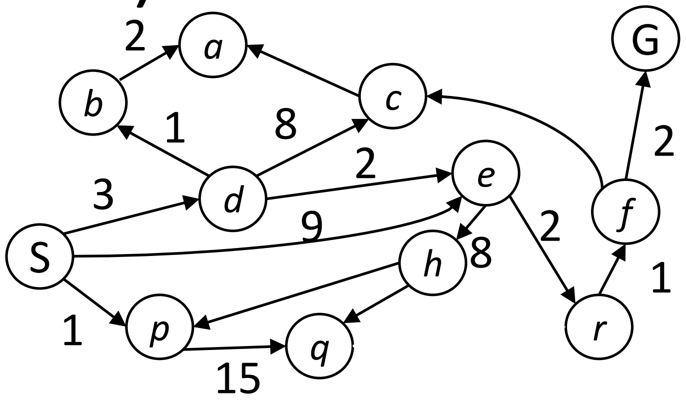




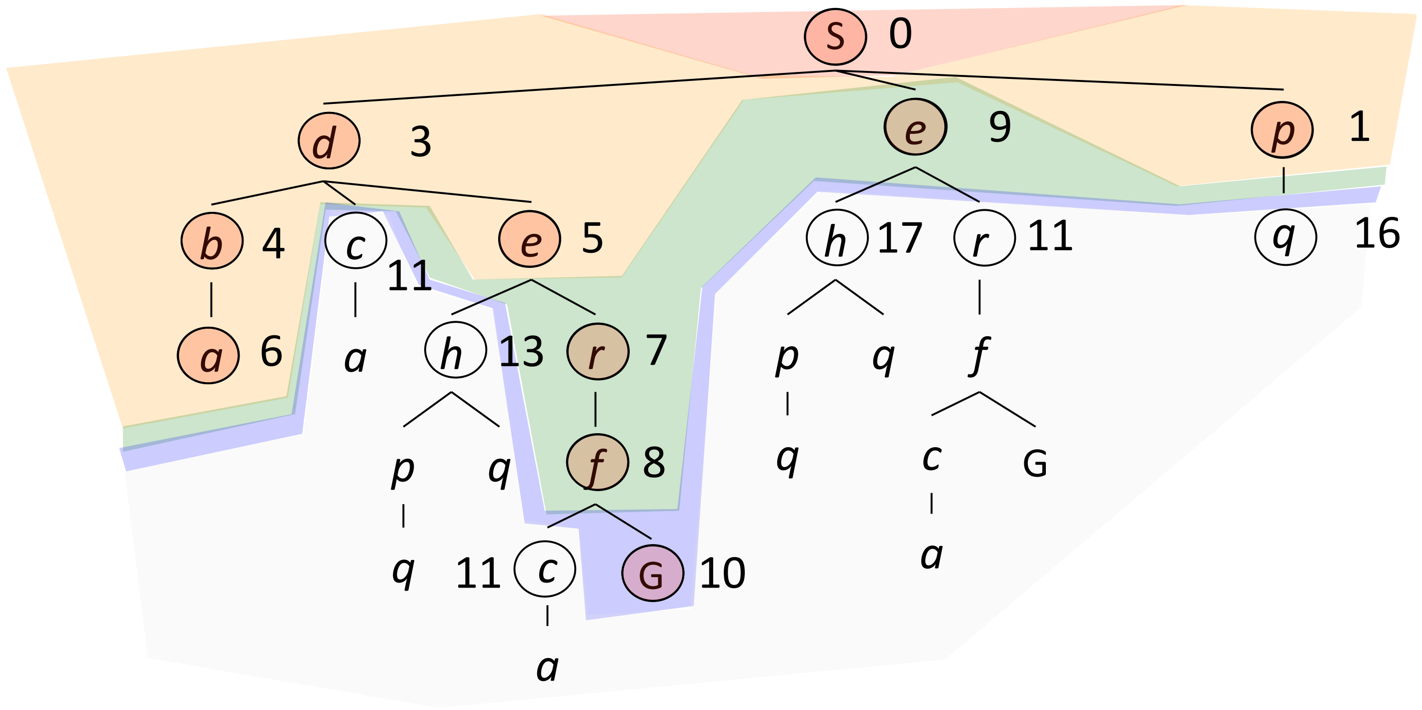
Uniform Cost (Tree) Search

Strategy: expand a cheapest node first:

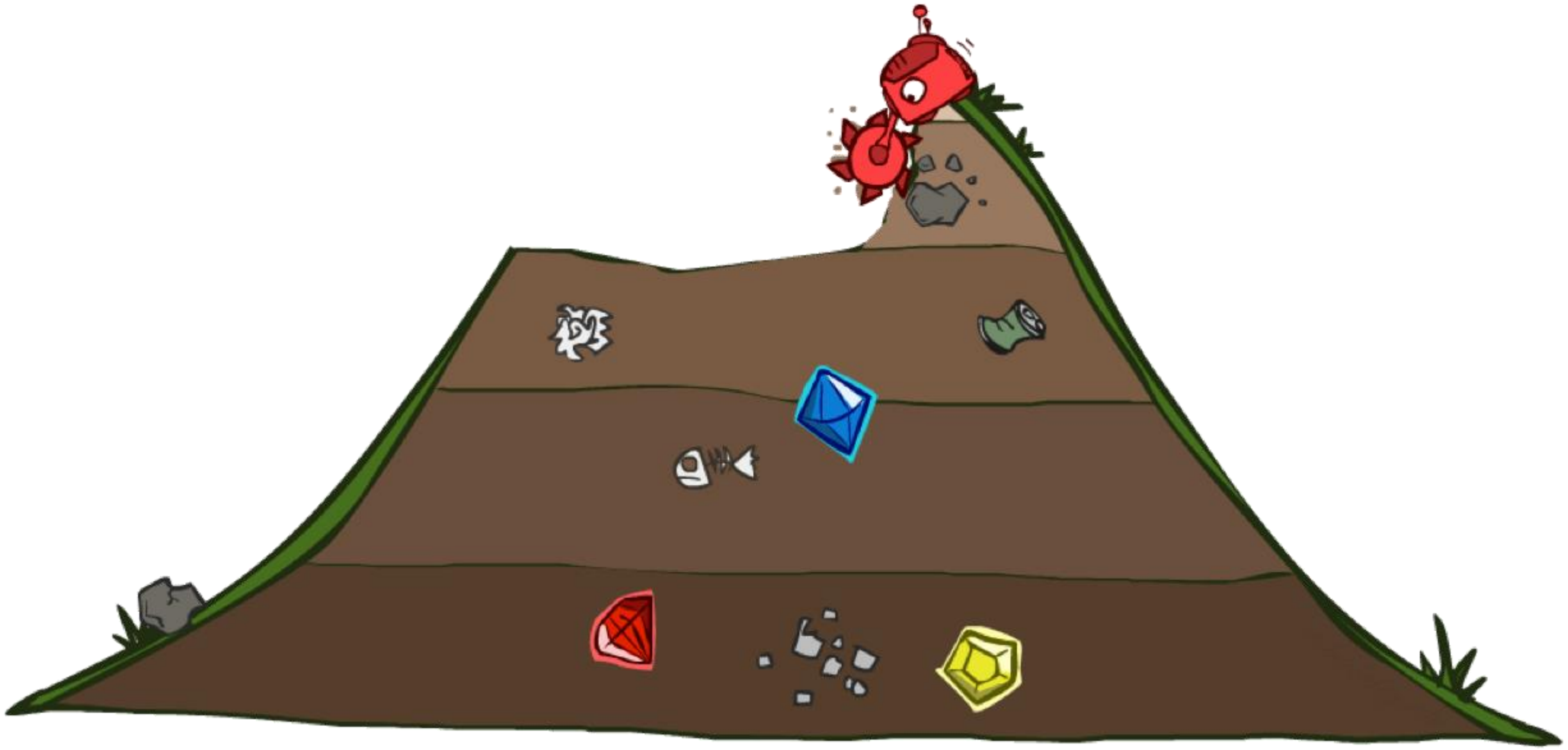
Frontier is a priority queue (priority: cumulative cost)



Cost contours



Uniform Cost Search



```
function GRAPH_SEARCH(problem) returns a solution, or failure
    initialize the explored set to be empty
    initialize the frontier as a specific work list (stack, queue, priority queue)
    add initial state of problem to frontier
    loop do
        if the frontier is empty then
            return failure
        choose a node and remove it from the frontier
        if the node contains a goal state then
            return the corresponding solution
        add the node state to the explored set
        for each resulting child from node
            if the child state is not already in the frontier or explored set then
                add child to the frontier
```

function UNIFORM-COST-SEARCH(problem) returns a solution, or failure

initialize the explored set to be empty

initialize the frontier as a **priority queue using node path_cost as the priority**

add initial state of problem to frontier **with path_cost = 0**

loop do

if the frontier is empty then

return failure

choose a node and remove it from the frontier

if the node contains a goal state then

return the corresponding solution

add the node state to the explored set

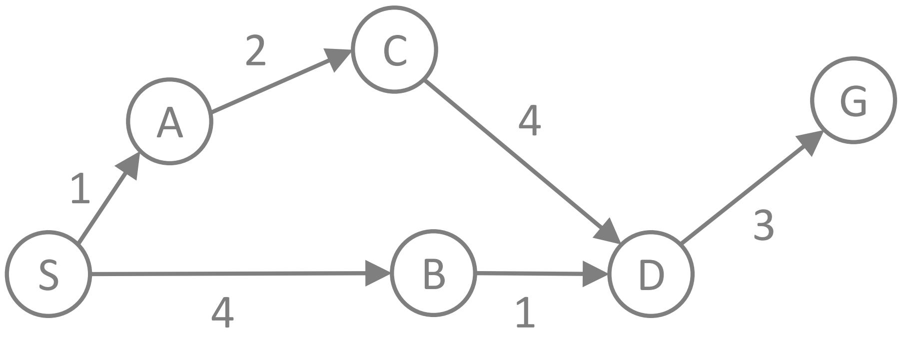
for each resulting child from node

if the child state is not already in the frontier or explored set then

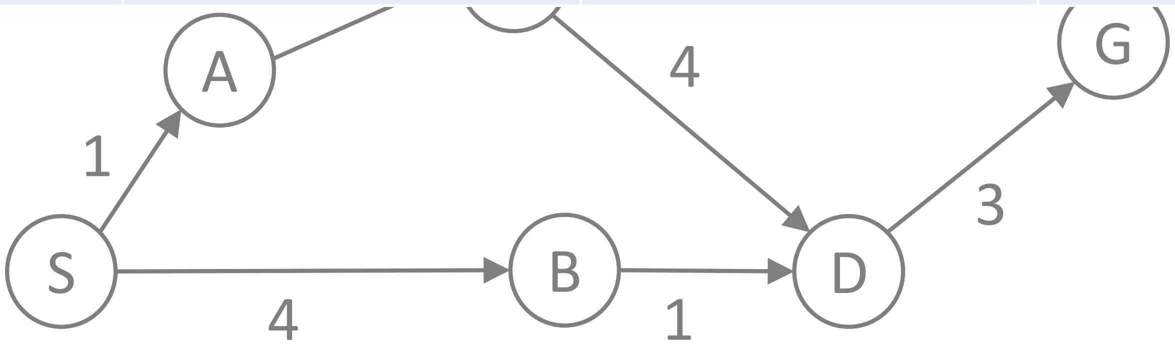
add child to the frontier

else if the child is already in the frontier with higher path_cost then

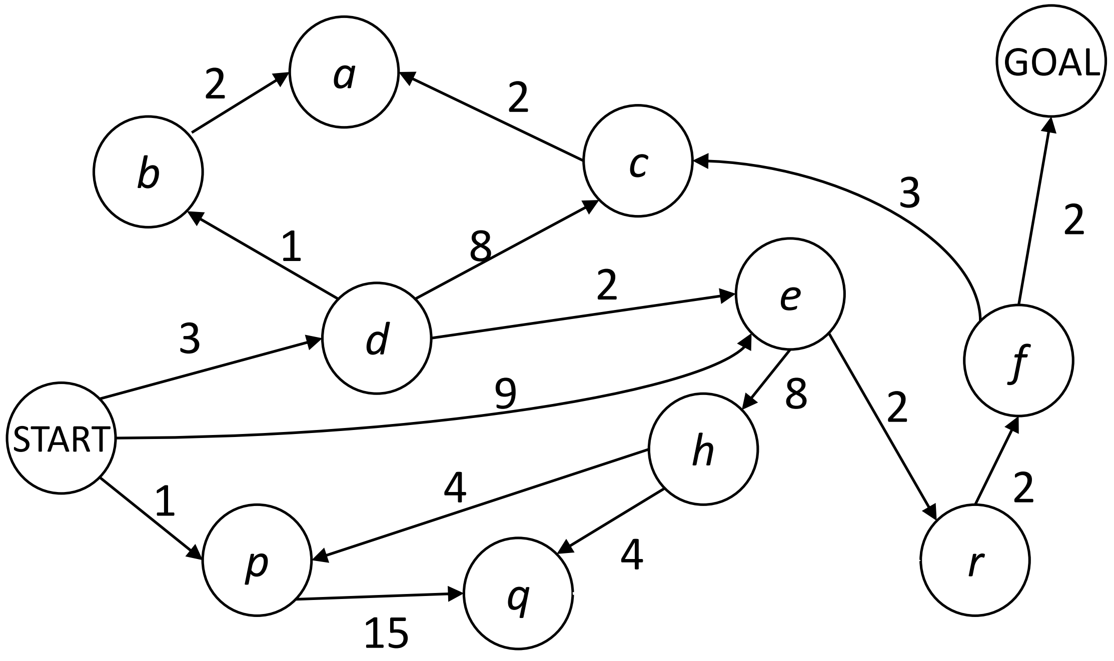
replace that frontier node with child



Priotity Queue	Current Node	Explored\Close List	Path followed
A(1),B(4)	S		S
B(4), C(3)	A(1)	S	S-A
B(4), D(7)	C(3)	S,A	S-A-C
D(7),D(5)	B(4)	S,A,C	S-A-C-B
D(7), G(8)	D(5)	S,A,C,B	S-A-C-B-D
G(8)		S,A,C,B,D	
	G(8) = goal	S,A,C,B,D,G	S-A-C-B-D-G
Path = S-B-D-G Cost = 8			

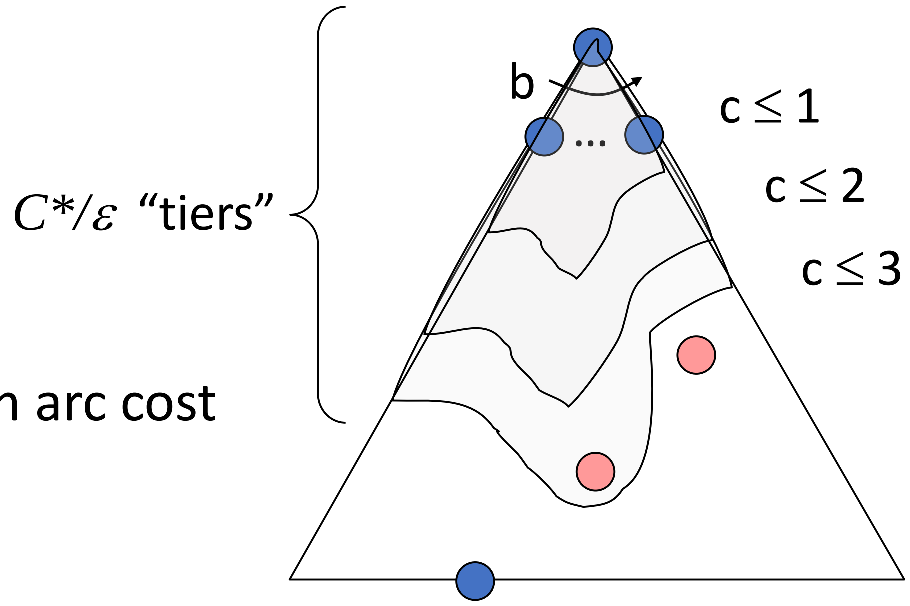


Walk-through UCS



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 frontier 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*)



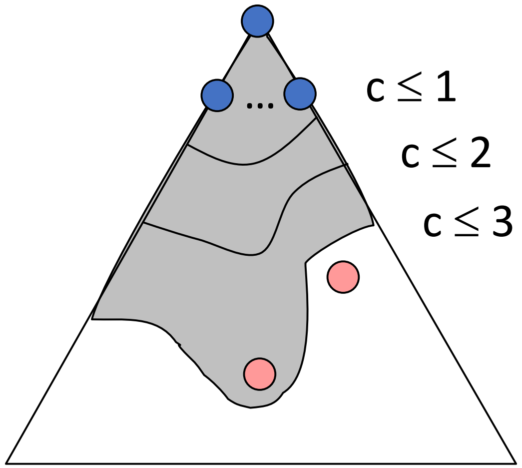
Uniform Cost Issues

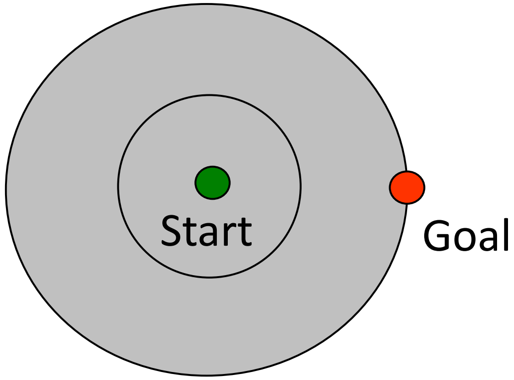
- Remember:
  - UCS explores increasing cost contours

- The good:
  - UCS is complete and optimal!

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

- We’ll fix that soon!





Recommended Reading

- Notes added on LMS
- https://www.oreilly.com/library/view/graph-algorithms/9781492047674/ch04.html


QUESTIONS


We become what we behold.
We shape our tools,
and thereafter our tools shape us.
- Marshall McLuhan