CSC 665: Artificial Intelligence

Search

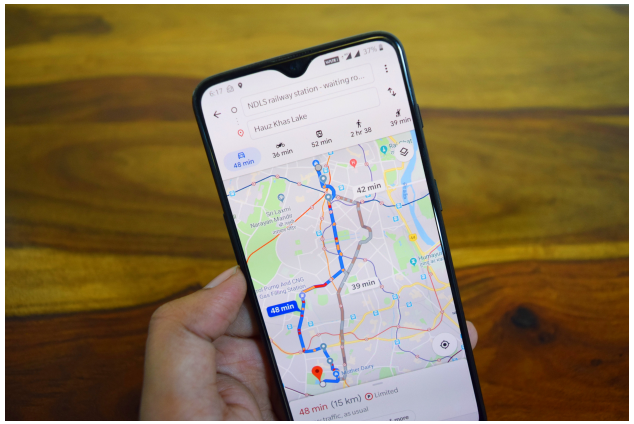
Pooyan Fazli

San Francisco State University

Today

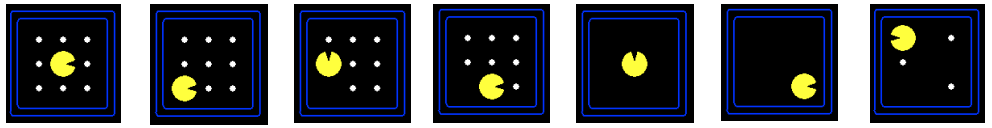
- Search Problems: Intro
- Generic Search Algorithm
- Uninformed Search Algorithms:
  - BFS
  - DFS
  - UCS

Search is a powerful tool in AI

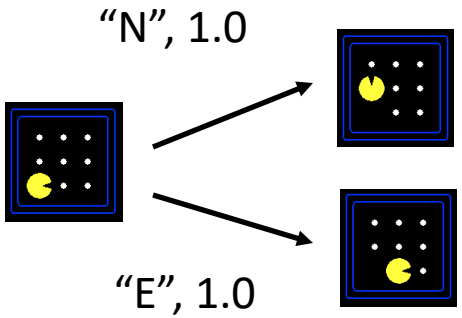


Search Problems

- A search problem consists of:
  - A state space:
    A set of states that a problem can be in



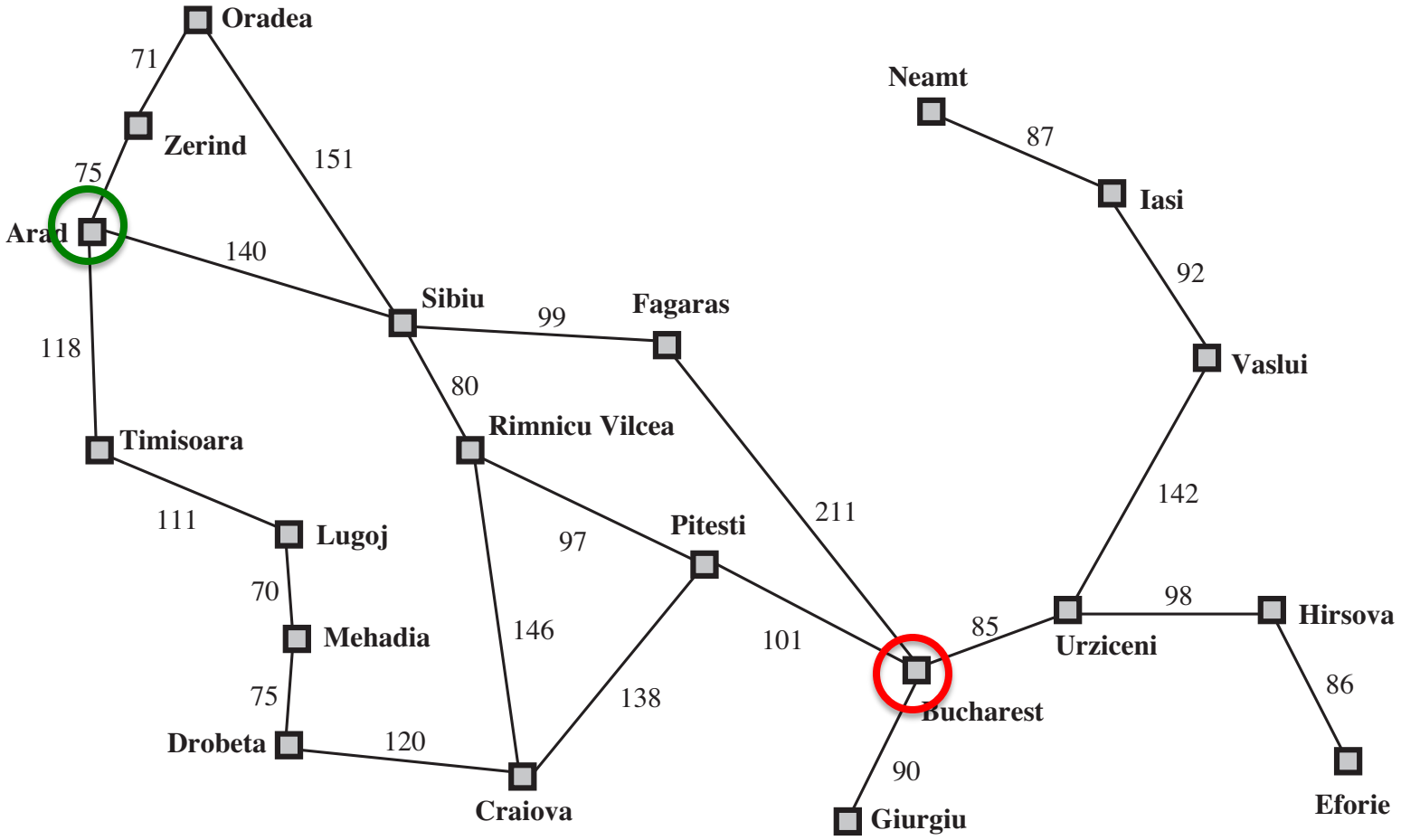
  - A successor function
    (with actions, costs)



  - 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

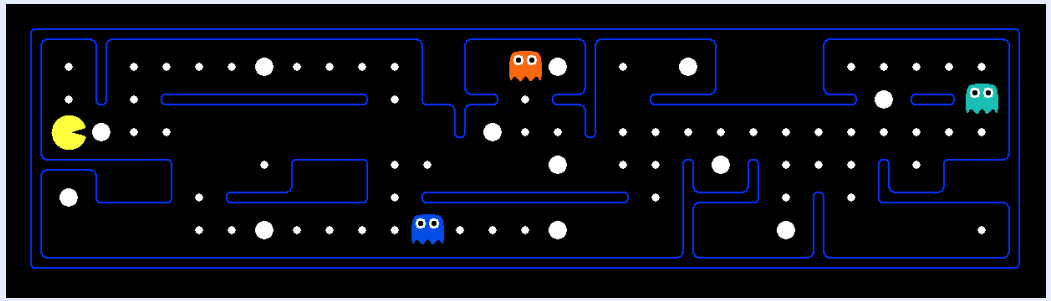
Example: Traveling in Romania



- 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 detail of the environment



A search state keeps only the details needed for planning (abstraction)

- Problem: Path Planning
  - 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?

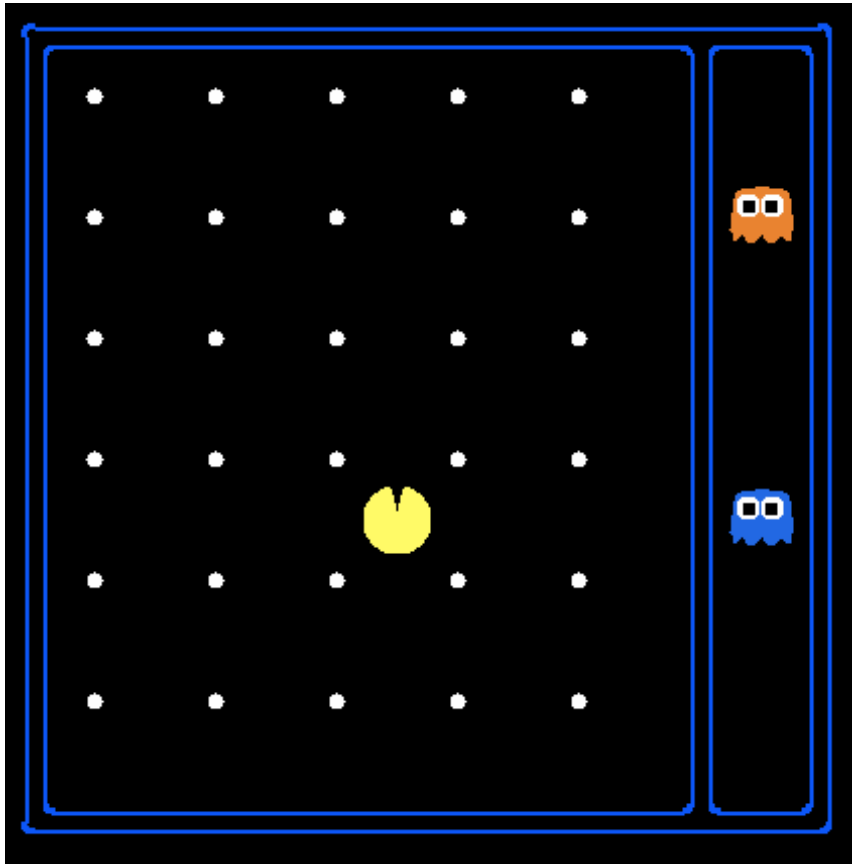
- World State:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

- How many
  - World states?

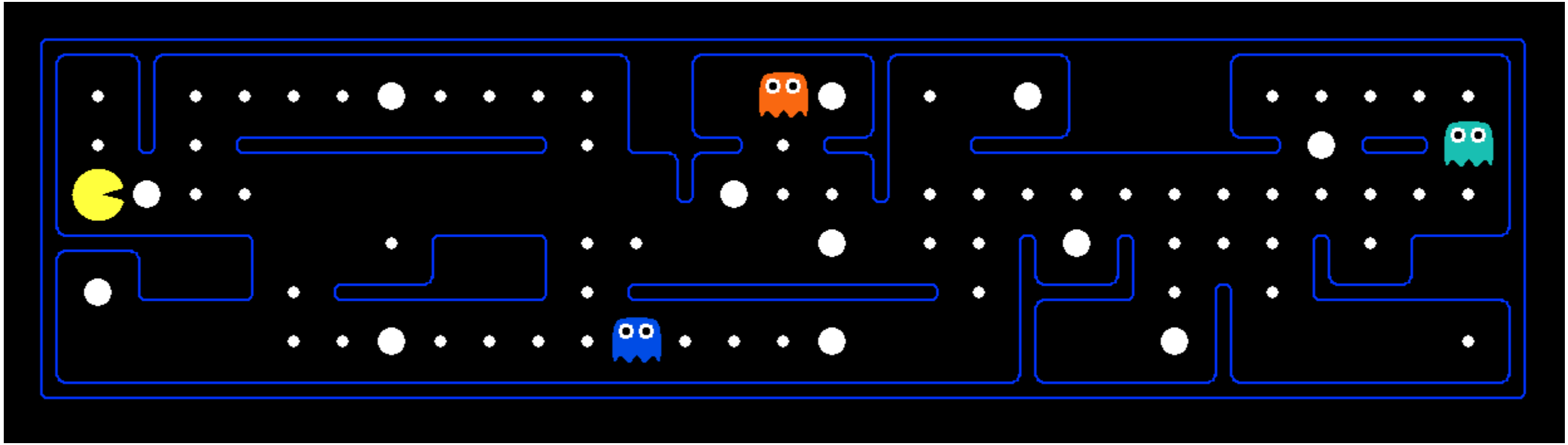
    $120x(2^{30})x(12^2)x4$
  - States for pathing?

    120
  - States for eat-all-dots?

    $120x(2^{30})$



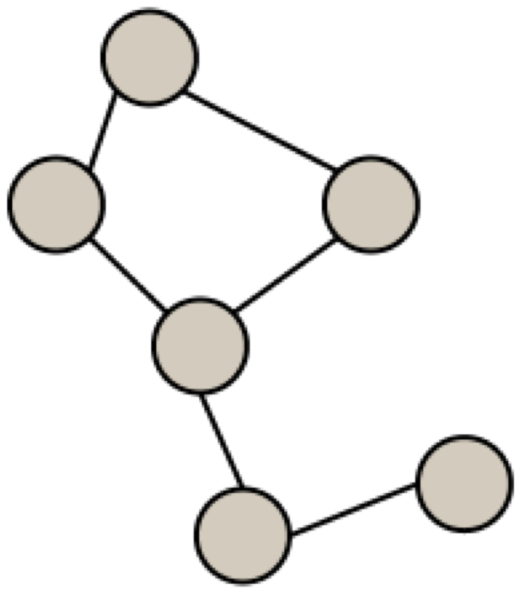
Quiz: Safe Passage



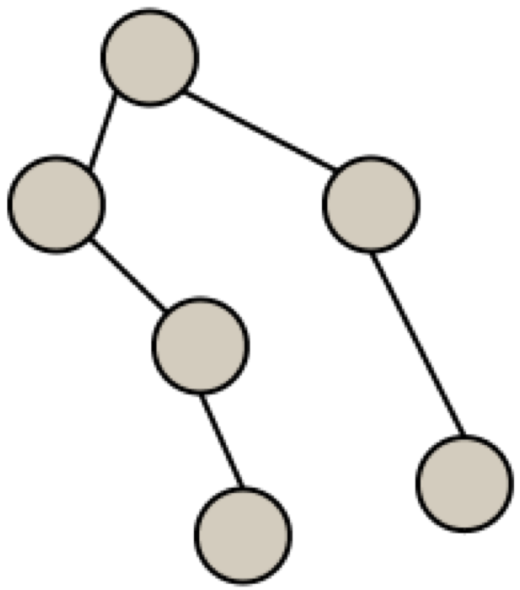
- Problem: eat all dots while keeping the ghosts permanently scared
- What does the state representation have to specify?
  - (pacman position, dot booleans, power pellet booleans, remaining scared time)

Background

Graphs & Trees



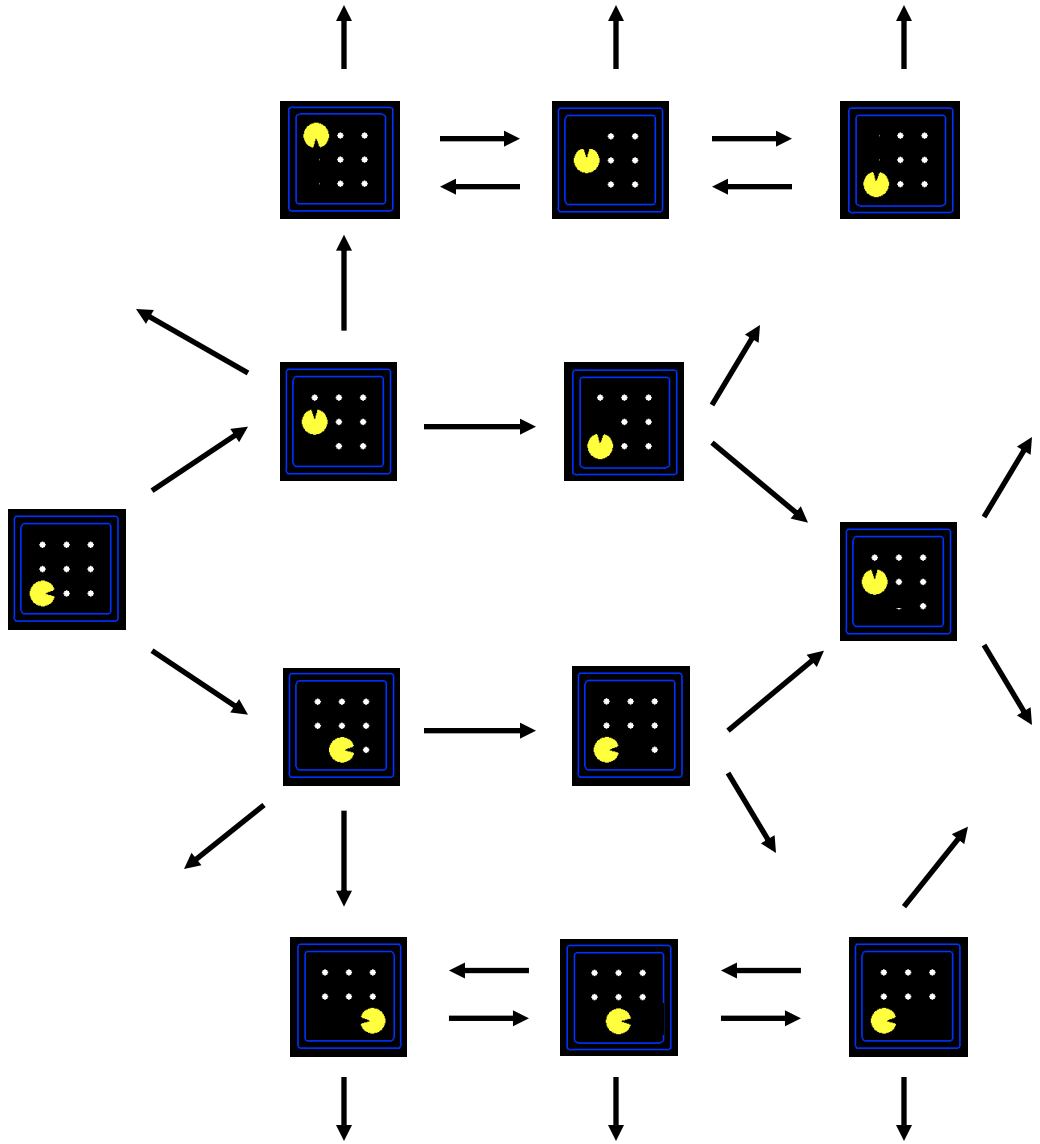
Graph



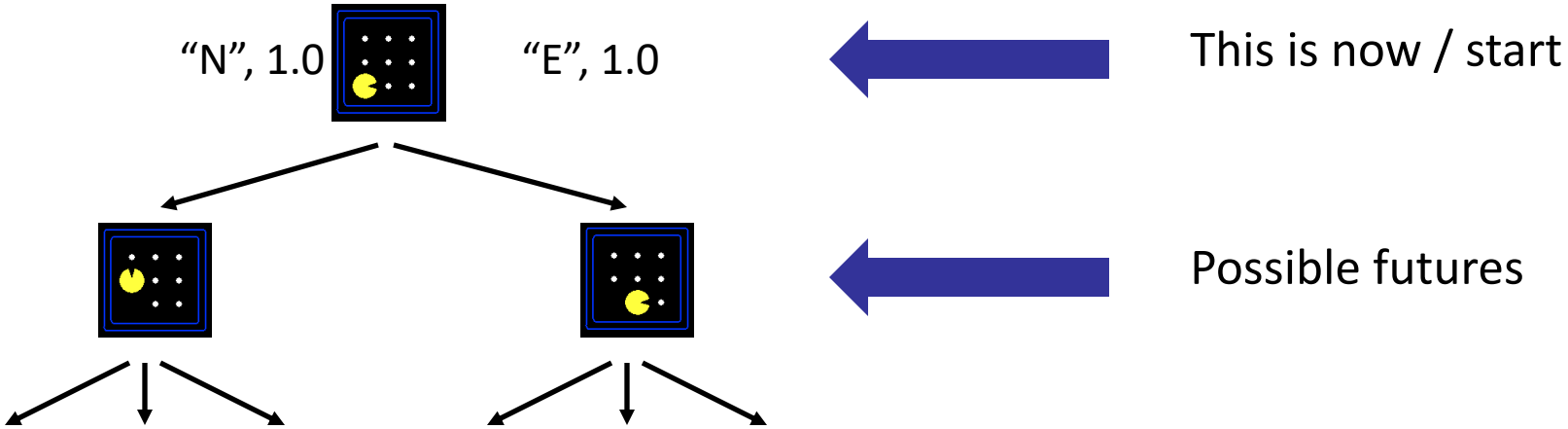
Tree

State Space Graph

- 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 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



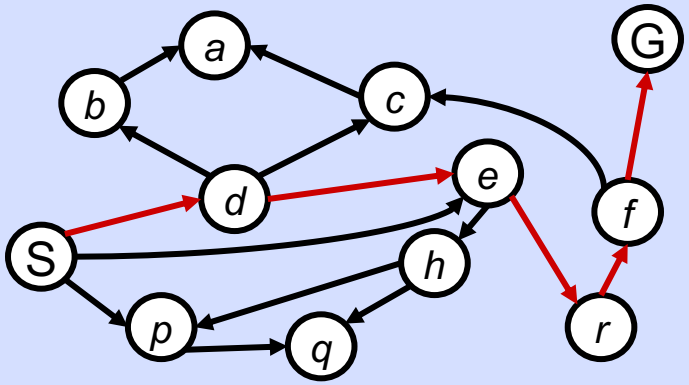
Search Trees



- A search tree:
  - 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

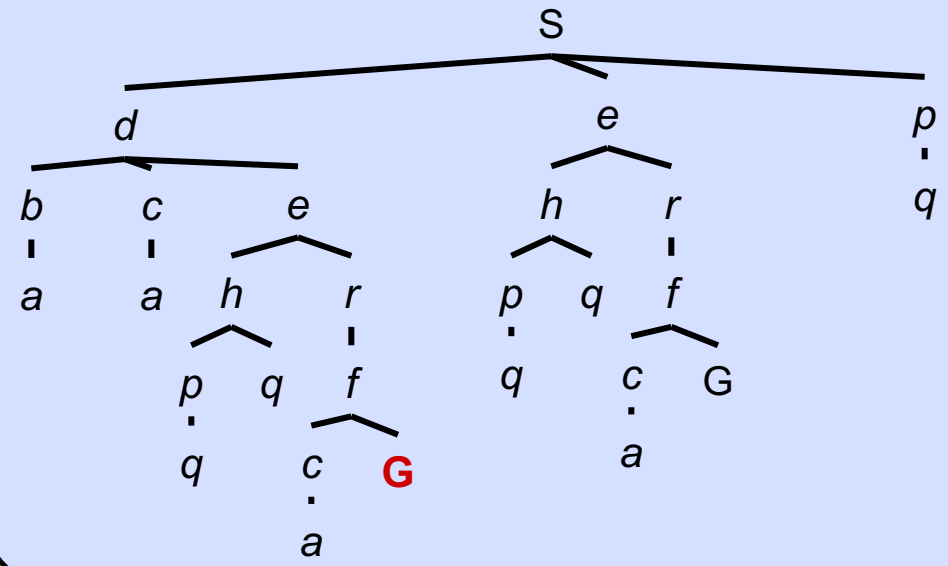
State Space Graphs vs. Search Trees

State Space Graph



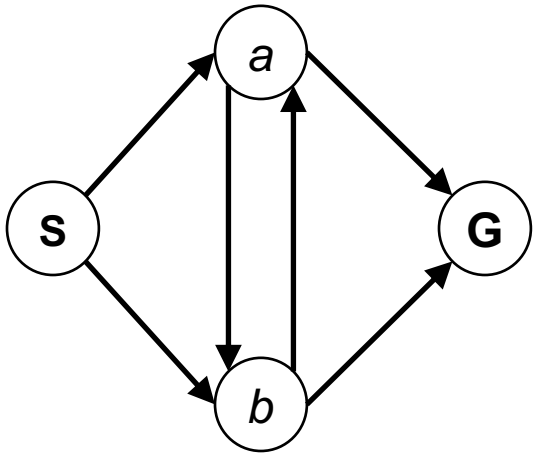
Each NODE in the search tree is an entire PATH in the state space graph.

Search Tree

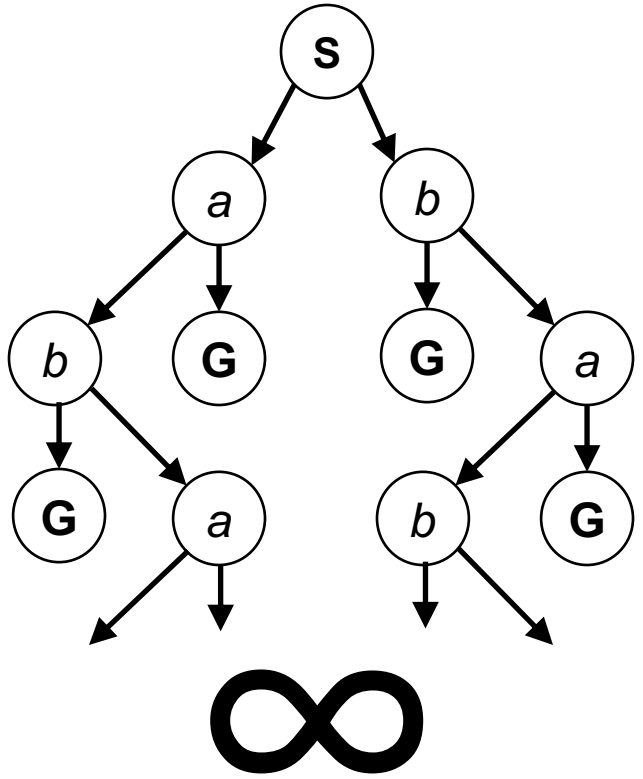


Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

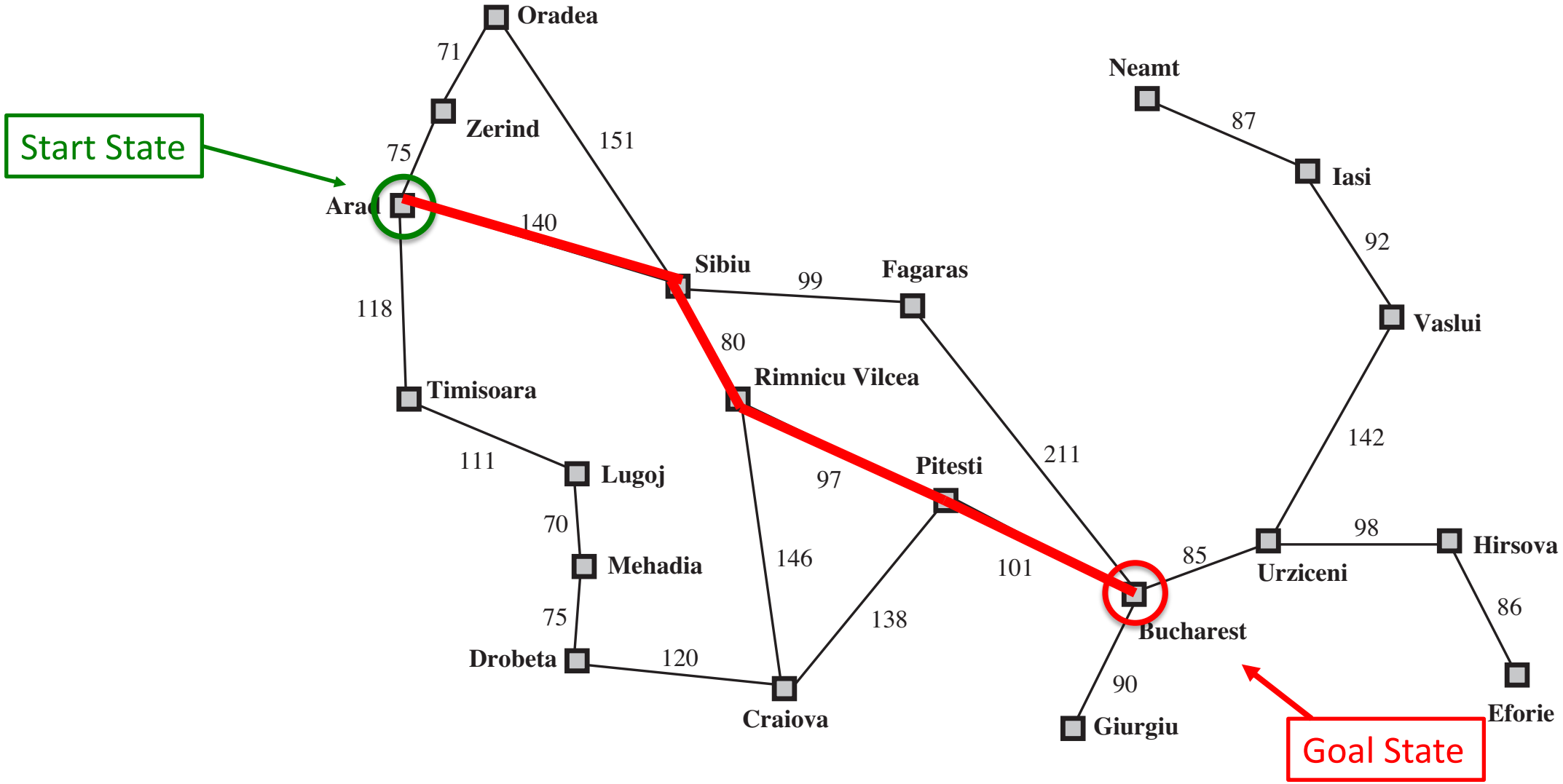


How big is its search tree (from S)?

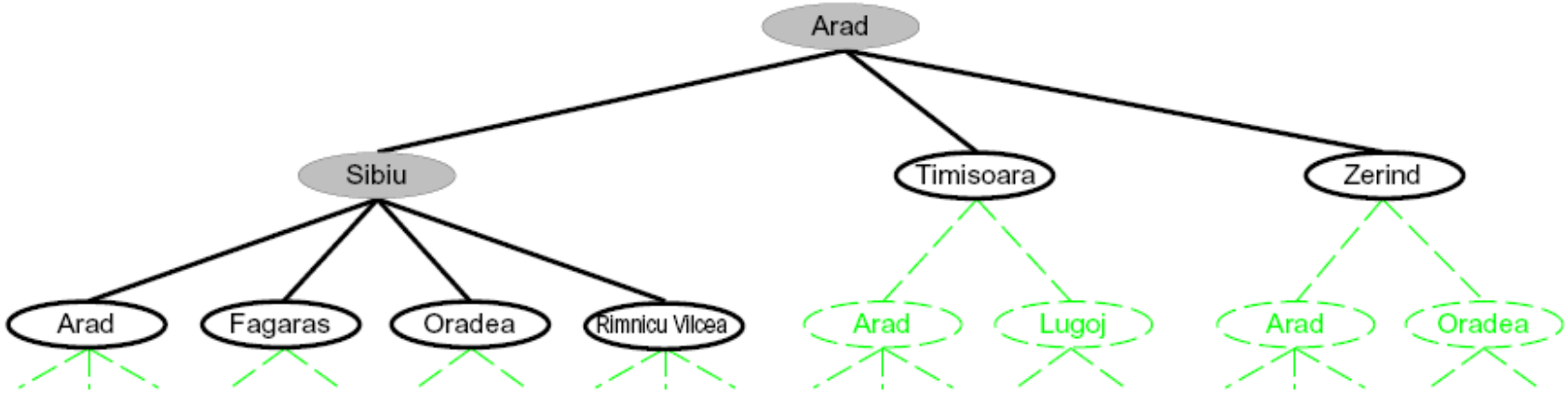


Important: Lots of repeated structure in the search tree!

State Space Graph

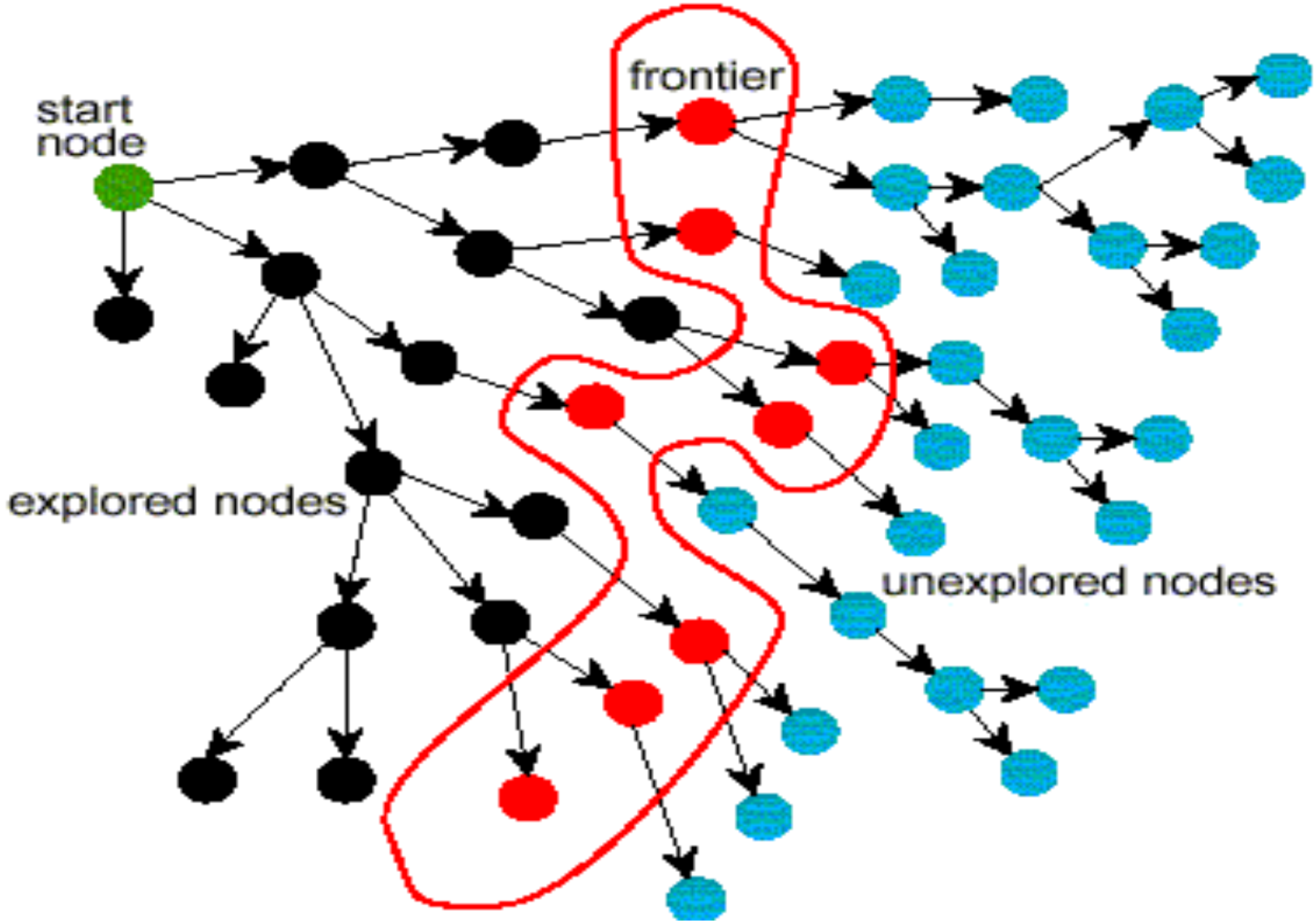


Searching with a Search Tree



- Search:
  - Expand out tree nodes
  - Maintain a frontier (e.g., stack, queue) of potential plans

Searching with a Search Tree



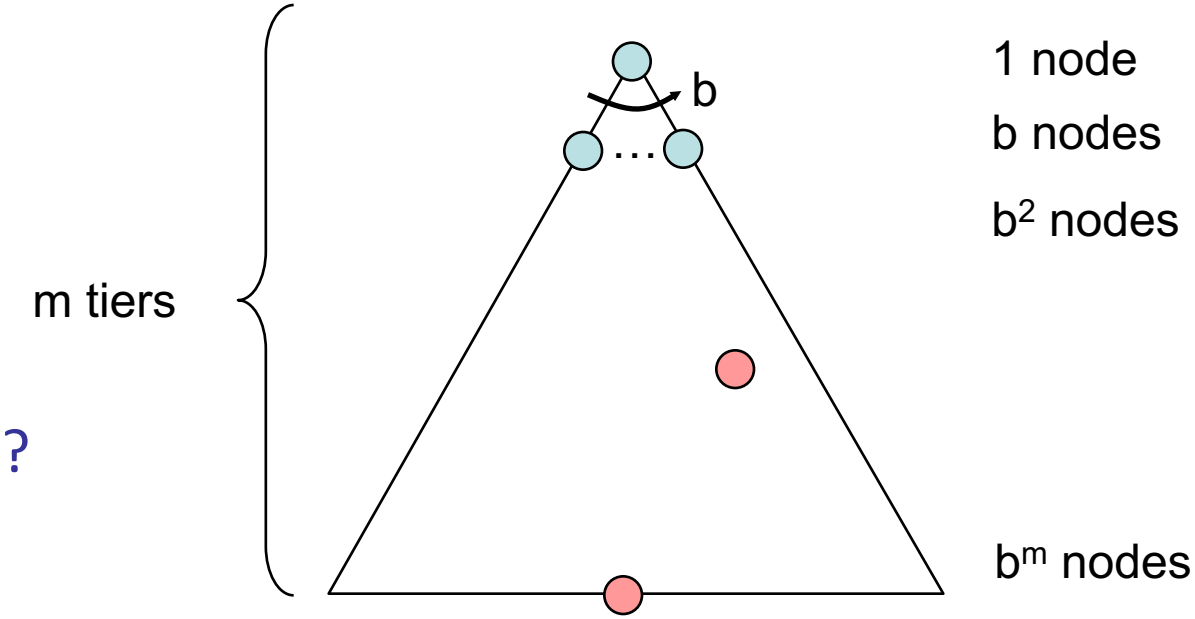
General Tree Search

```
function TREE-SEARCH(problem) returns a solution or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then
            return failure
        else choose a node and remove it from the frontier
        if the node contains a goal state then
            return solution
        else expand the chosen node, adding all the neighboring nodes to the frontier
```

- Important ideas:
  - Expansion
  - Frontier
  - Exploration strategy
- Main question: which frontier nodes to explore?

Background

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



Background

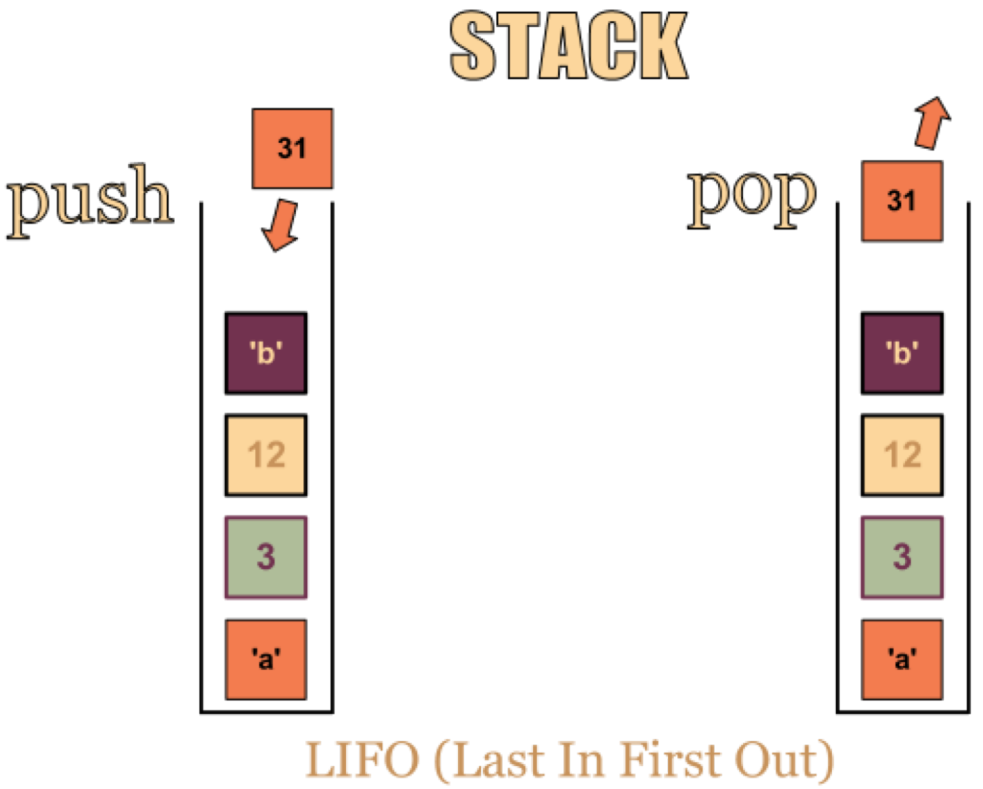


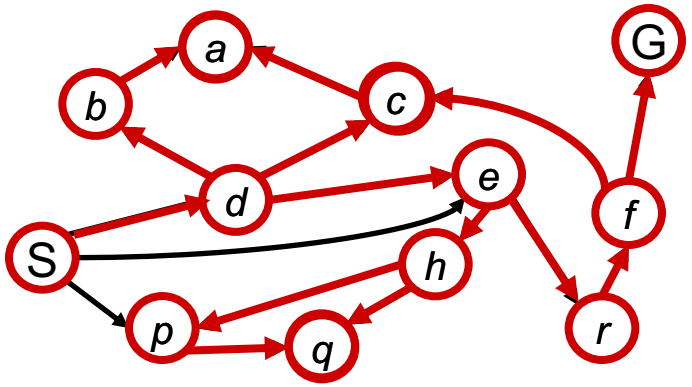
Image Reference: http://www.stoimen.com/blog/2012/06/05/computer-algorithms-stack-and-queue-data-structure/

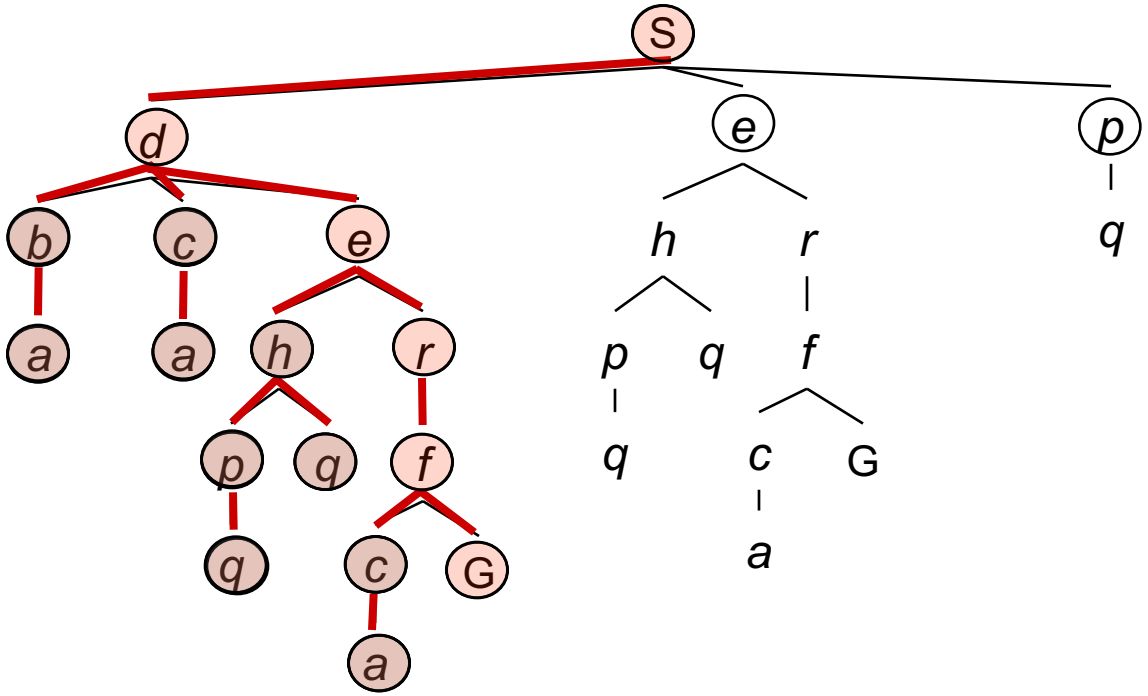
Depth-First Search

Depth-First Search

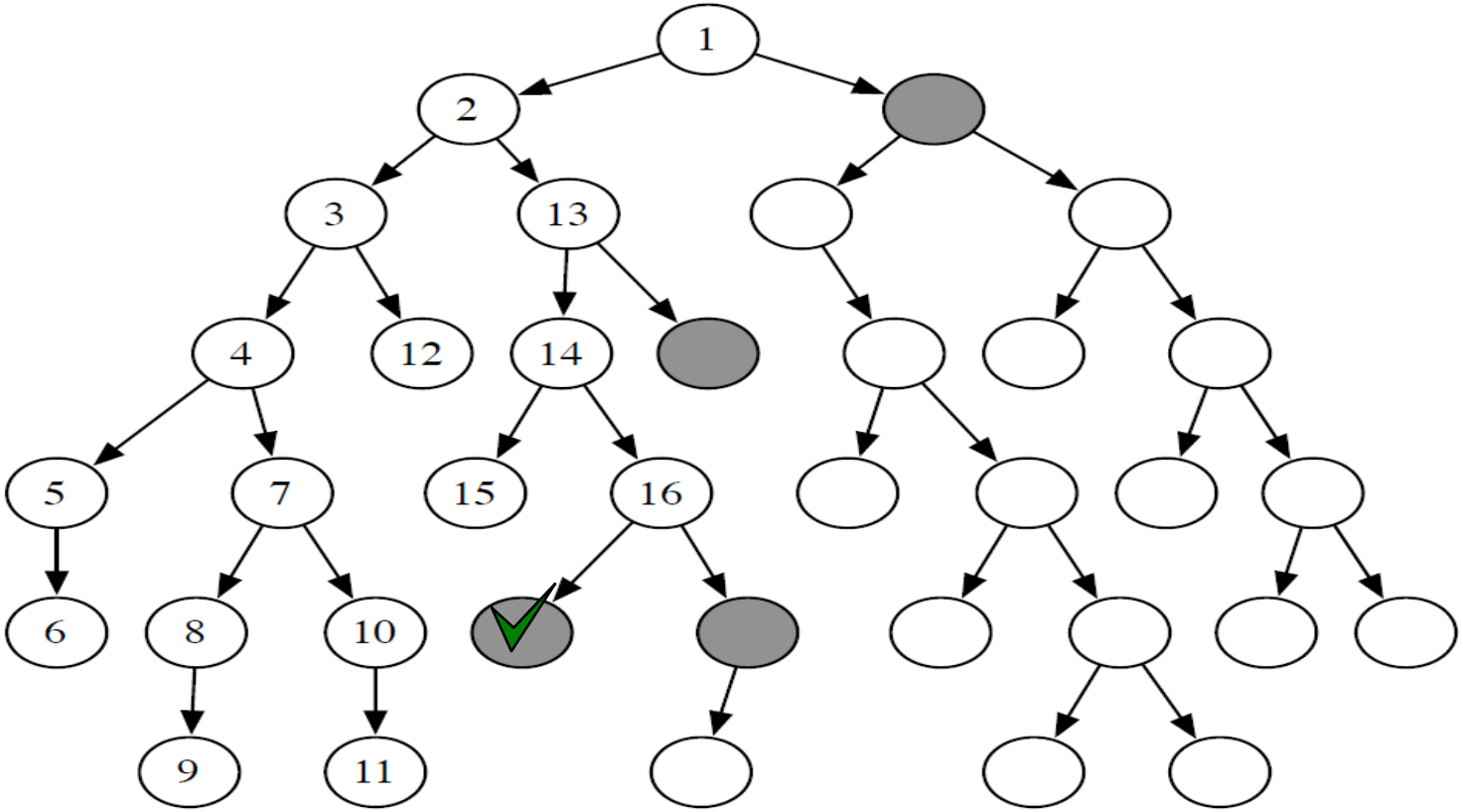
Strategy: expand a deepest node first

Implementation:
Frontier is a LIFO stack





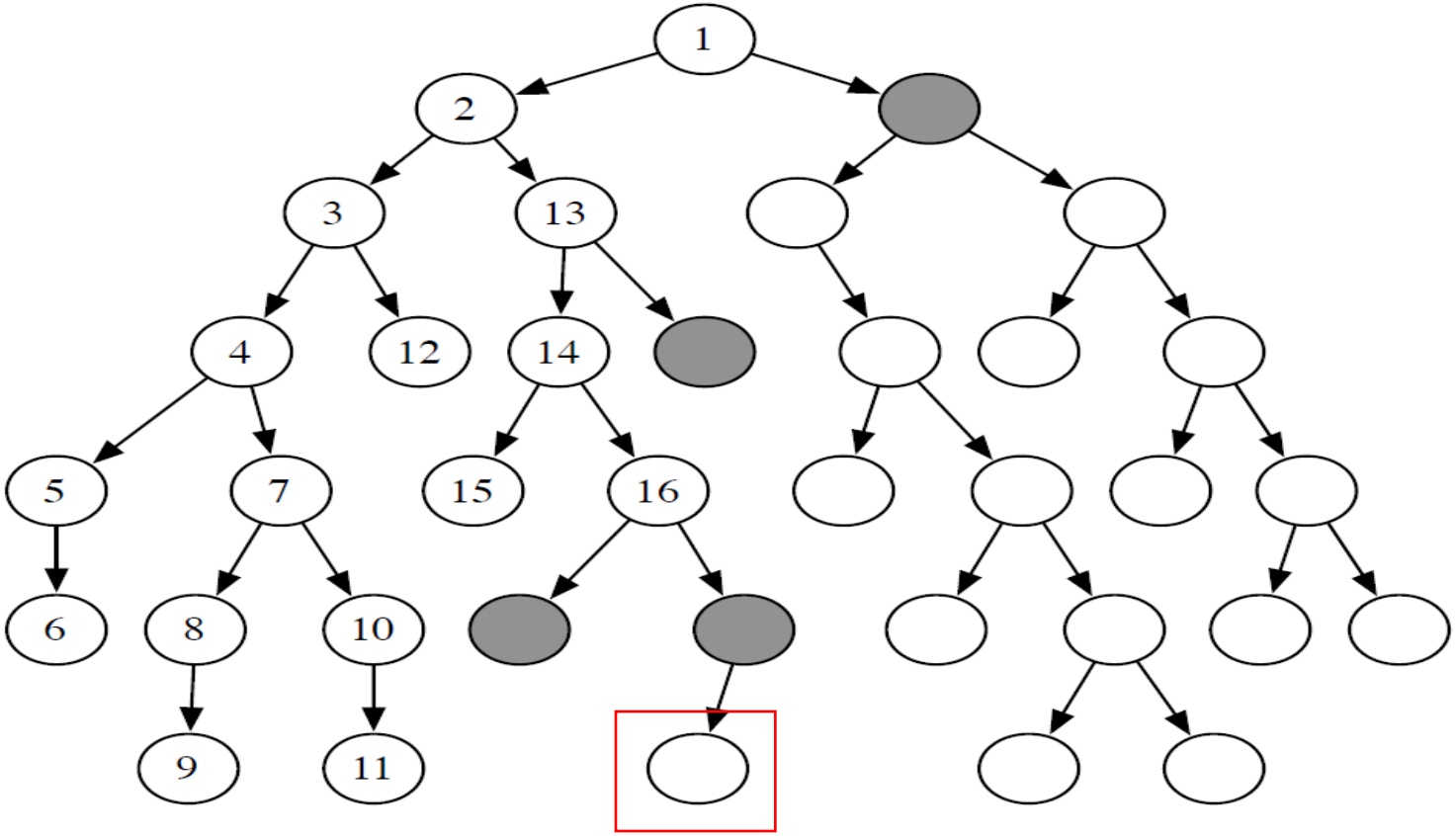
Depth-First Search: Quiz



- Frontier (Stack): shaded nodes
- Which node will be expanded next?

Depth-First Search: Quiz

You only return once the goal is being expanded! Not when a goal is put onto the frontier!



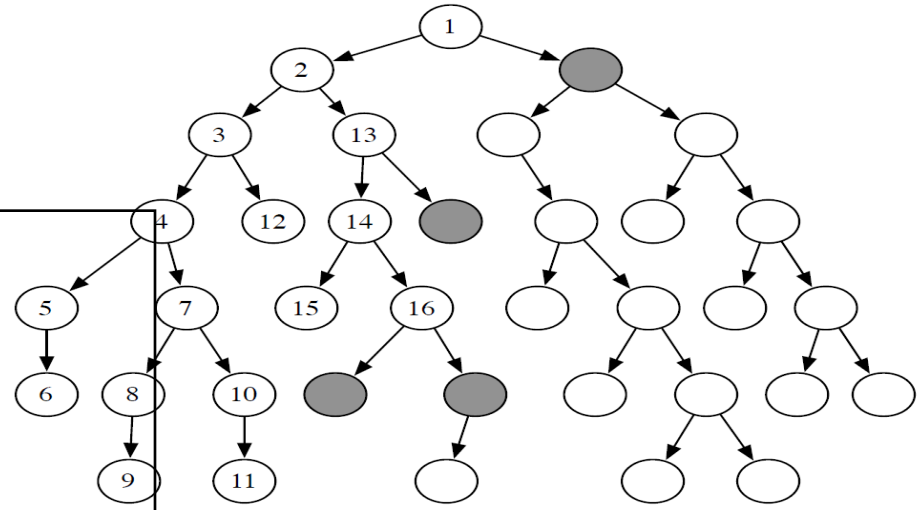
- Say, node in red box is a goal
- How many more nodes will be expanded?

1 2 3 4

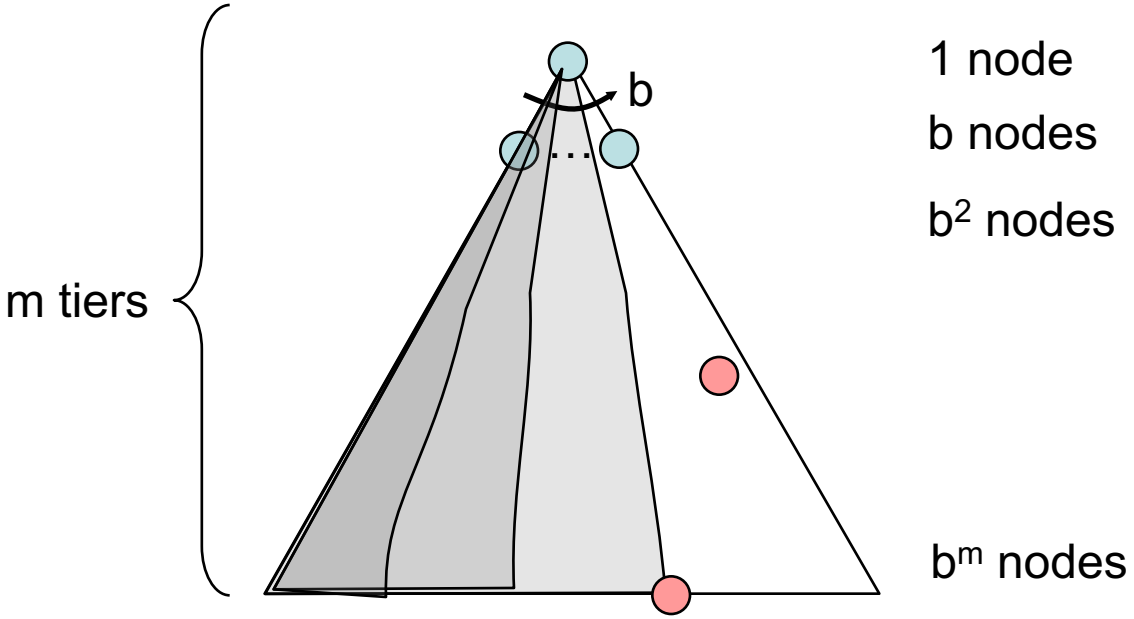
DFS as an Instantiation of the Generic Tree Search Algorithm

```
function TREE-SEARCH(problem) returns
    initialize the frontier using the initial s
    loop do
        if the frontier is empty then
            return failure
        else choose a node and remove it from the frontier
        if the node contains a goal state then
            return solution
        else expand the chosen node, adding all the neighboring nodes to the frontier
```

In DFS, the frontier is a last-in-first-out stack



Depth-First Search (DFS) Properties



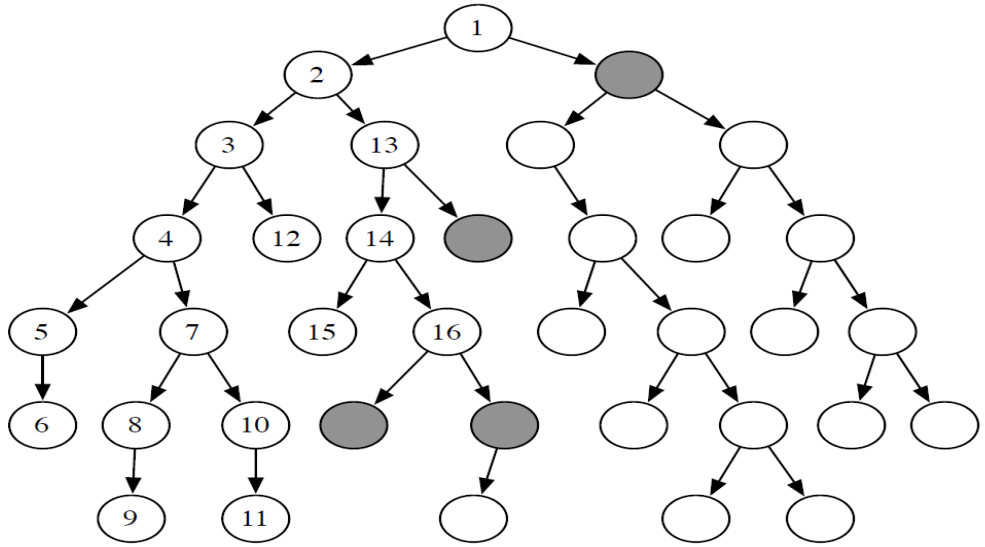
Depth-First Search (DFS) Properties

Definition: A search algorithm is complete if whenever there is at least one solution, the algorithm is guaranteed to find it within a finite amount of time.

Is DFS Complete?

Yes

No



Depth-First Search (DFS) Properties

Definition: A search algorithm is optimal if when it finds a solution, it is the best one

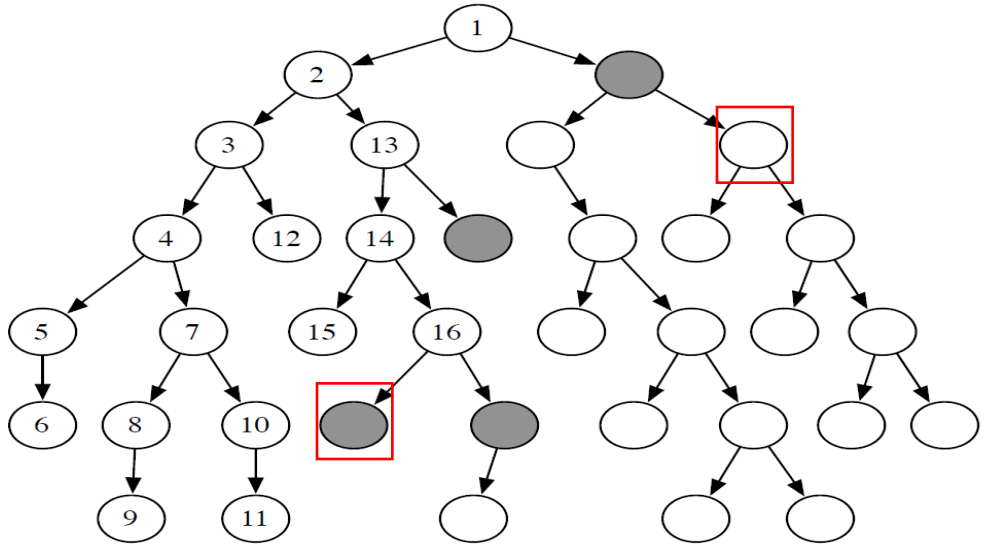
Is DFS Optimal?

Yes

No

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

E.g., goal nodes: red boxes

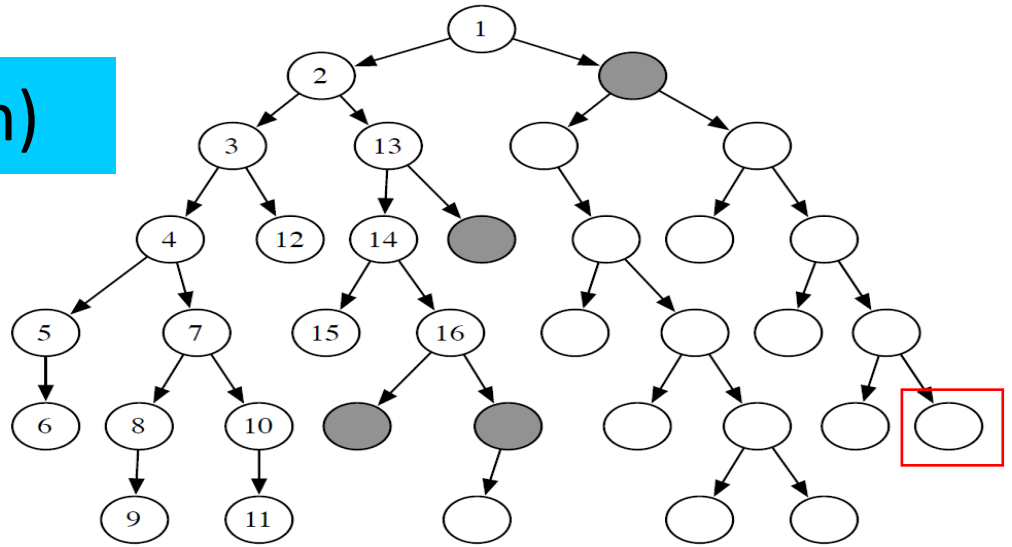


Depth-First Search (DFS) Properties

Definition: The time complexity of a search algorithm is the worst-case amount of time it will take to run, expressed in terms of
- maximum path length *m*
- maximum branching factor *b*.

- What is DFS's time complexity, in terms of m and b ?

$O(b^m)$ ✓ $O(m^b)$ $O(bm)$ $O(b+m)$



- E.g., single goal node: red box

Depth-First Search (DFS) Properties

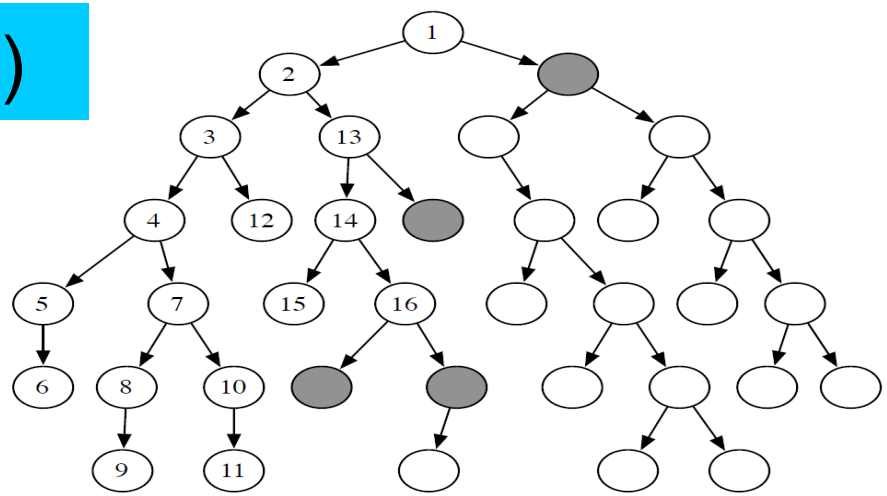
Definition: The space complexity of a search algorithm is the worst-case amount of memory that the algorithm will use (i.e., the maximal number of nodes on the frontier), expressed in terms of

- maximum path length *m*
- maximum branching factor *b*.

- What is DFS's space complexity, in terms of m and b ?

$O(b^m)$ $O(m^b)$ $O(bm)$ ✓ $O(b+m)$

- The longest possible path is m, and for every node in that path must maintain a frontier of size b

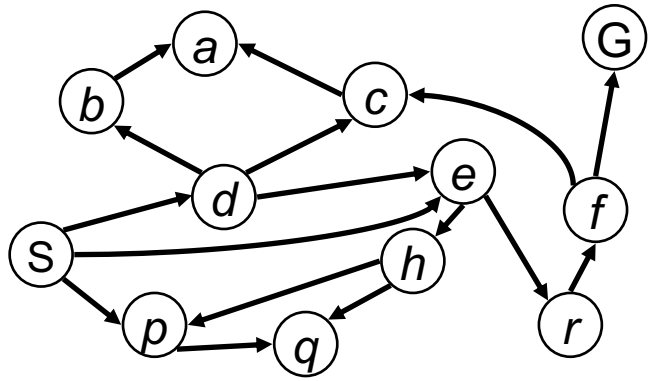


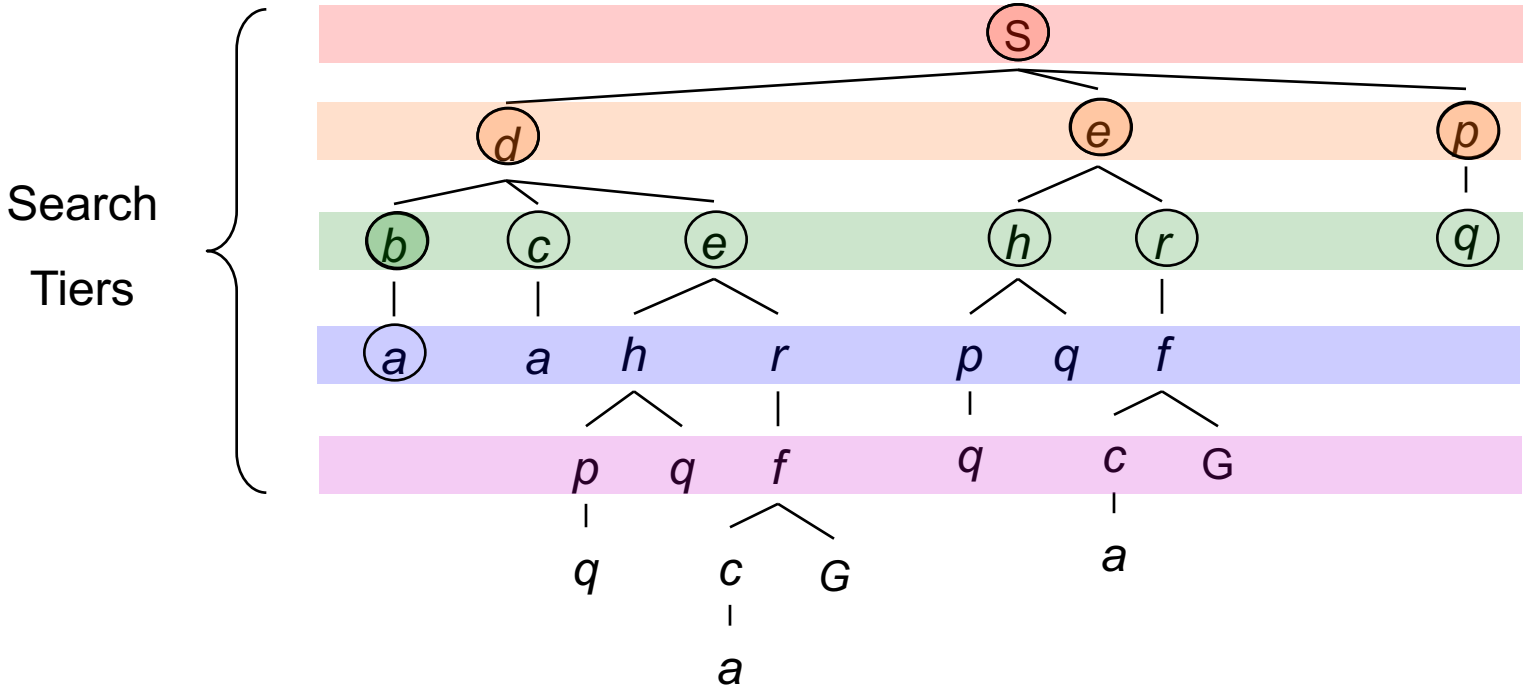
Breadth-First Search

Breadth-First Search

Strategy: expand a shallowest node first

Implementation:
Frontier is a FIFO queue

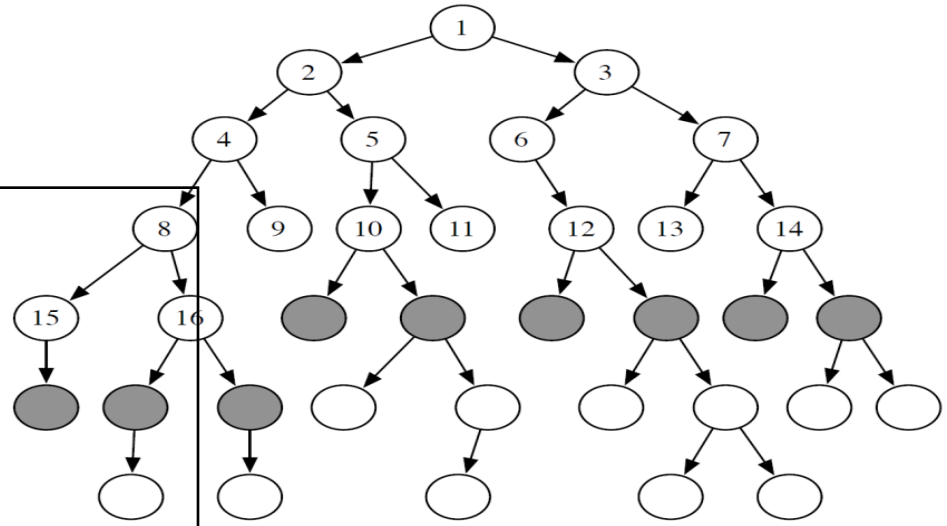




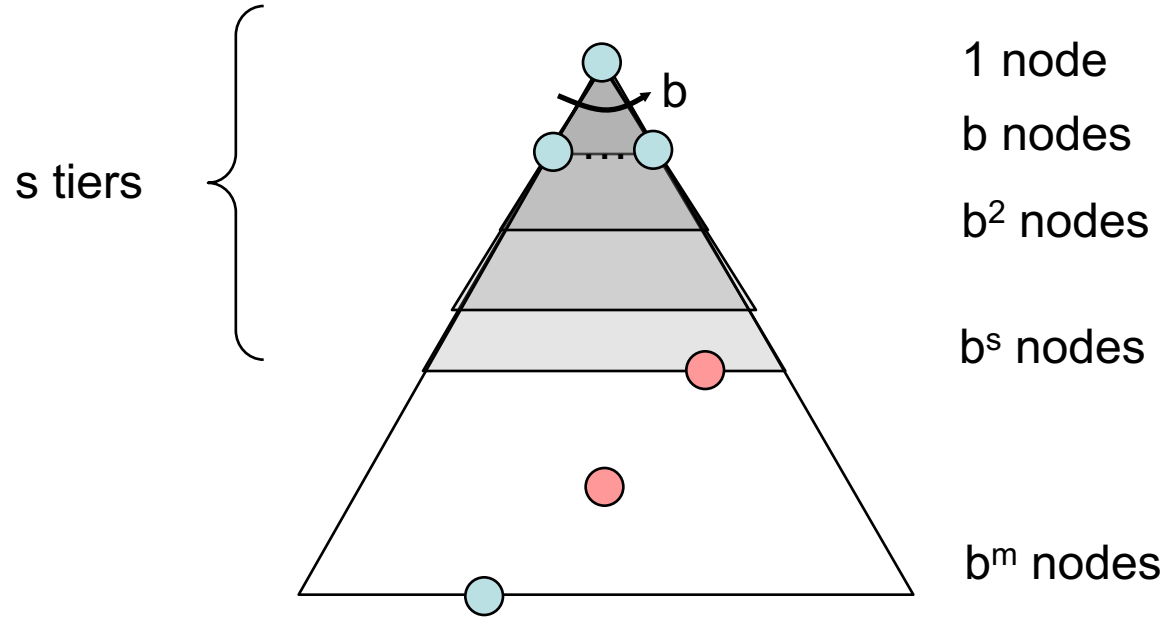
BFS as an Instantiation of the Generic Tree Search Algorithm

```
function TREE-SEARCH(problem) returns
    initialize the frontier using the initial s
    loop do
        if the frontier is empty then
            return failure
        else choose a node and remove it from the frontier
        if the node contains a goal state then
            return solution
        else expand the chosen node, adding all the neighboring nodes to the frontier
```

In BFS, the frontier is a first-in-first-out queue



Breadth-First Search (BFS) Properties



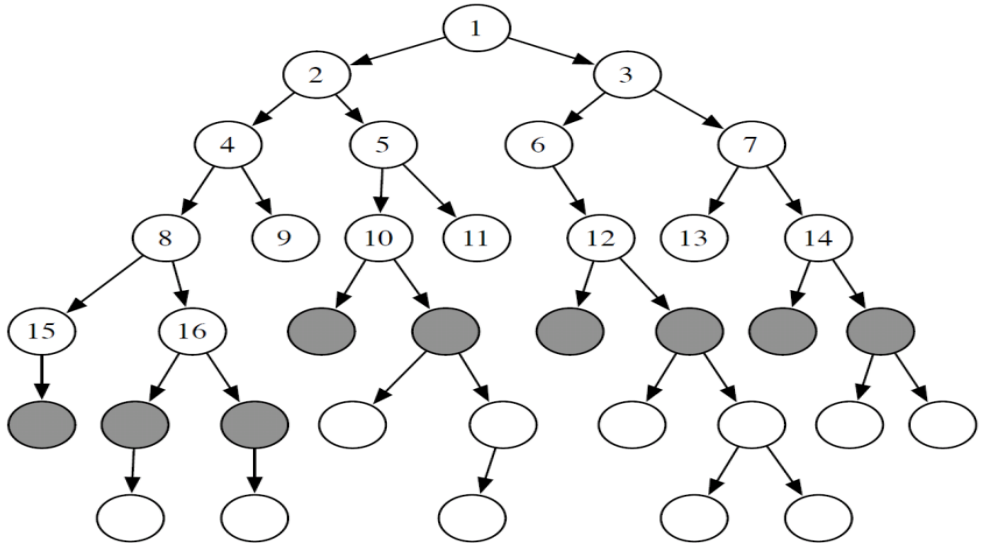
Breadth-First Search (BFS) Properties

Definition: A search algorithm is complete if whenever there is at least one solution, the algorithm is guaranteed to find it within a finite amount of time.

Is BFS Complete?

Yes

No



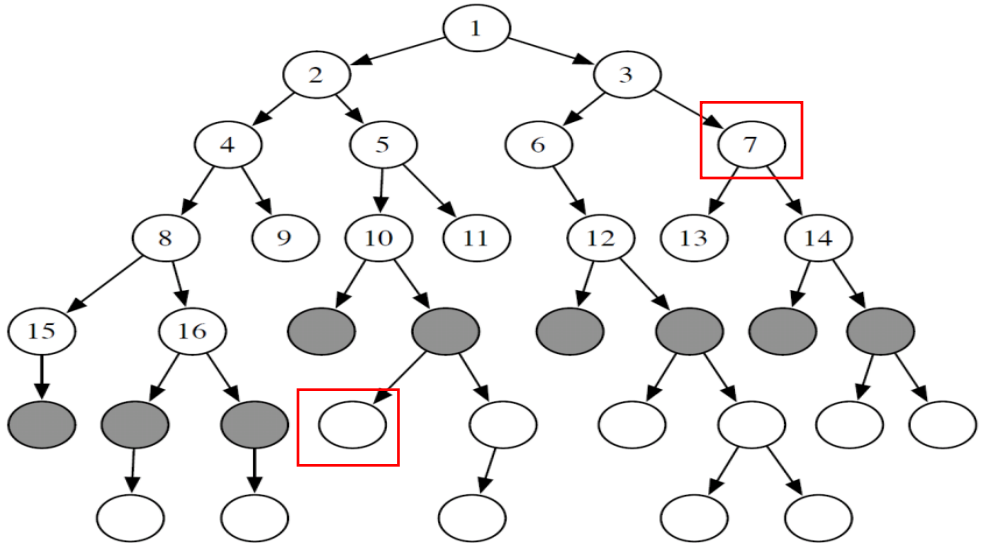
Breadth-First Search (BFS) Properties

Definition: A search algorithm is optimal if when it finds a solution, it is the best one

Is BFS Optimal?

Yes ✓

No

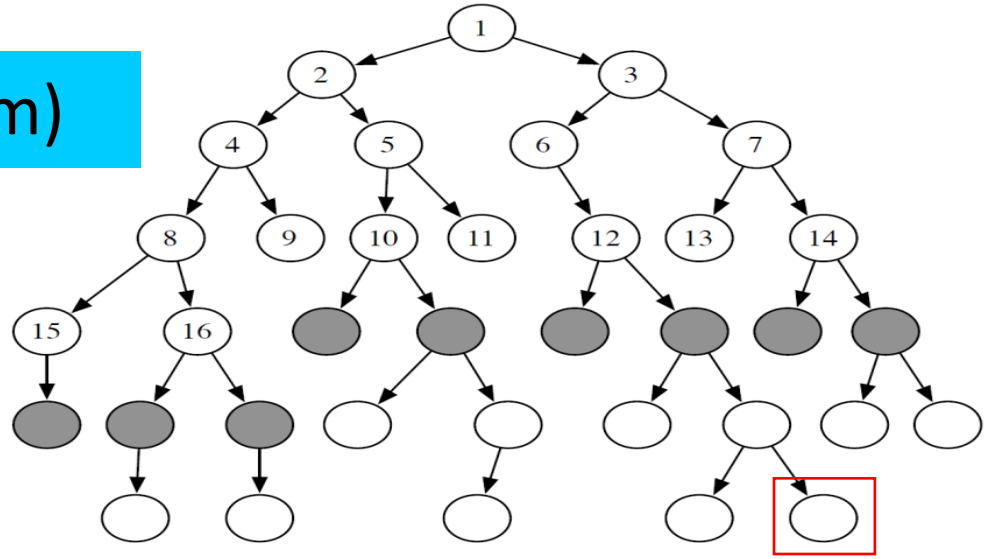


Breadth-First Search (BFS) Properties

Definition: The time complexity of a search algorithm is the worst-case amount of time it will take to run, expressed in terms of
- maximum path length *m*
- Maximum branching factor *b*.

- What is BFS's time complexity, in terms of m and b ?

$O(b^m)$ ✓ $O(m^b)$ $O(bm)$ $O(b+m)$



- E.g., single goal node: red box

Breadth-First Search (BFS) Properties

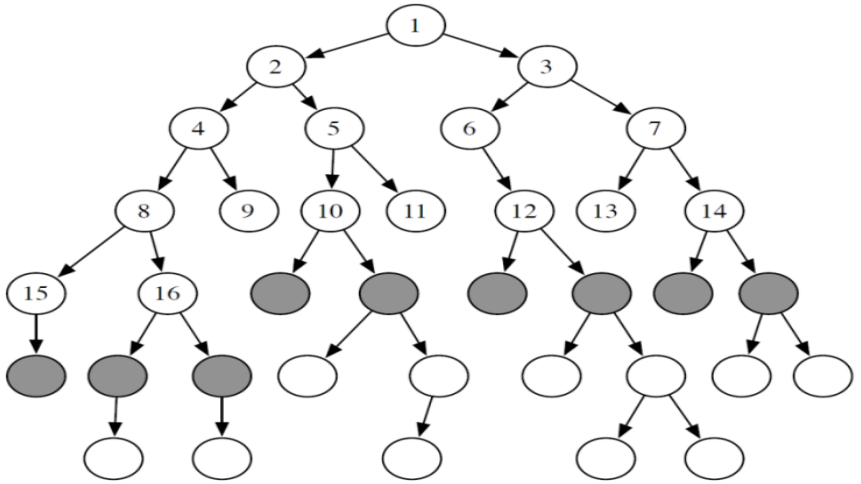
Definition: The space complexity of a search algorithm is the worst-case amount of memory that the algorithm will use (i.e., the maximal number of nodes on the frontier), expressed in terms of

- maximum path length *m*
- Maximum branching factor *b*.

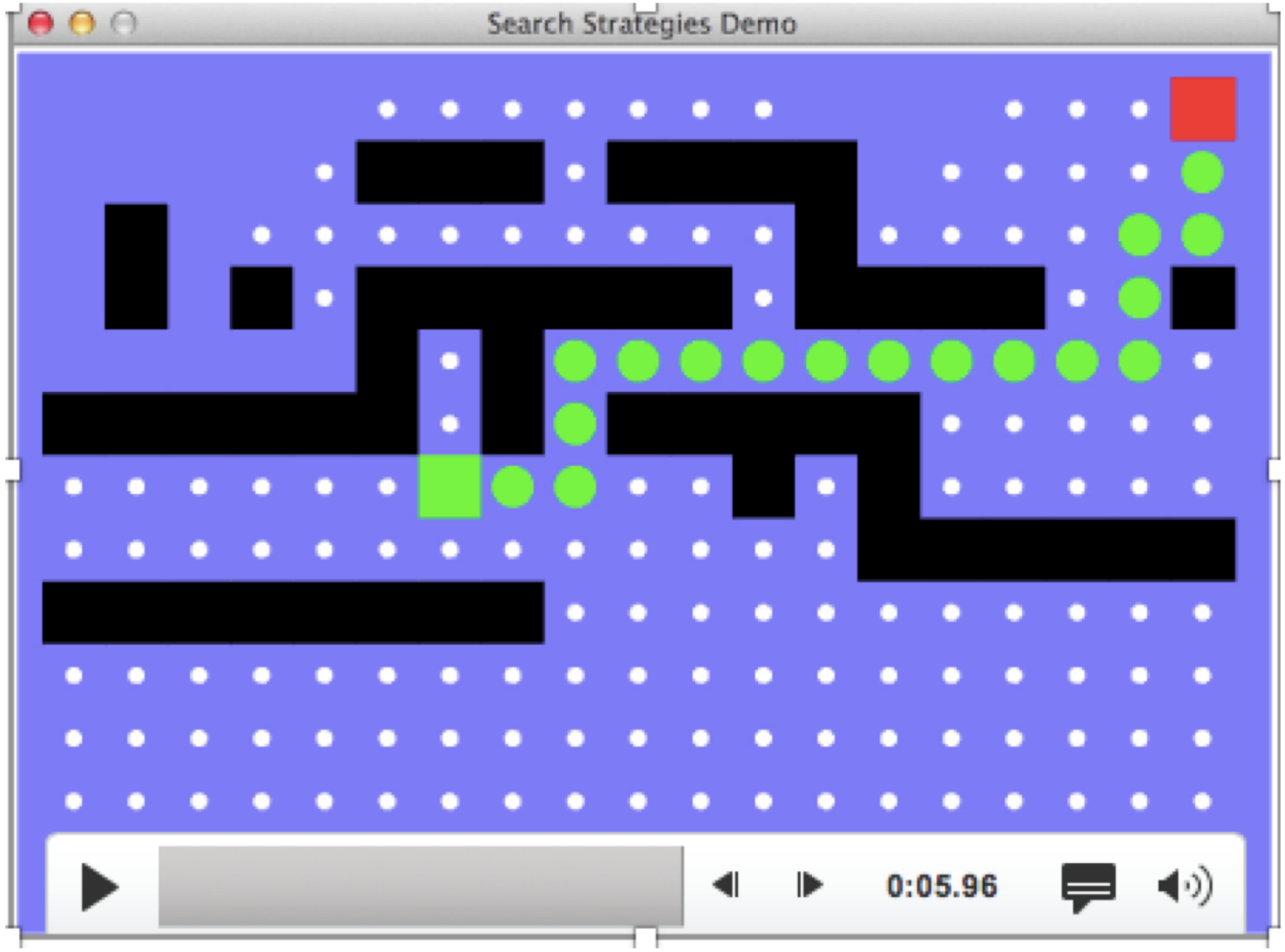
- What is BFS's space complexity, in terms of m and b ?

$O(b^m)$ ✓ $O(m^b)$ $O(bm)$ $O(b+m)$

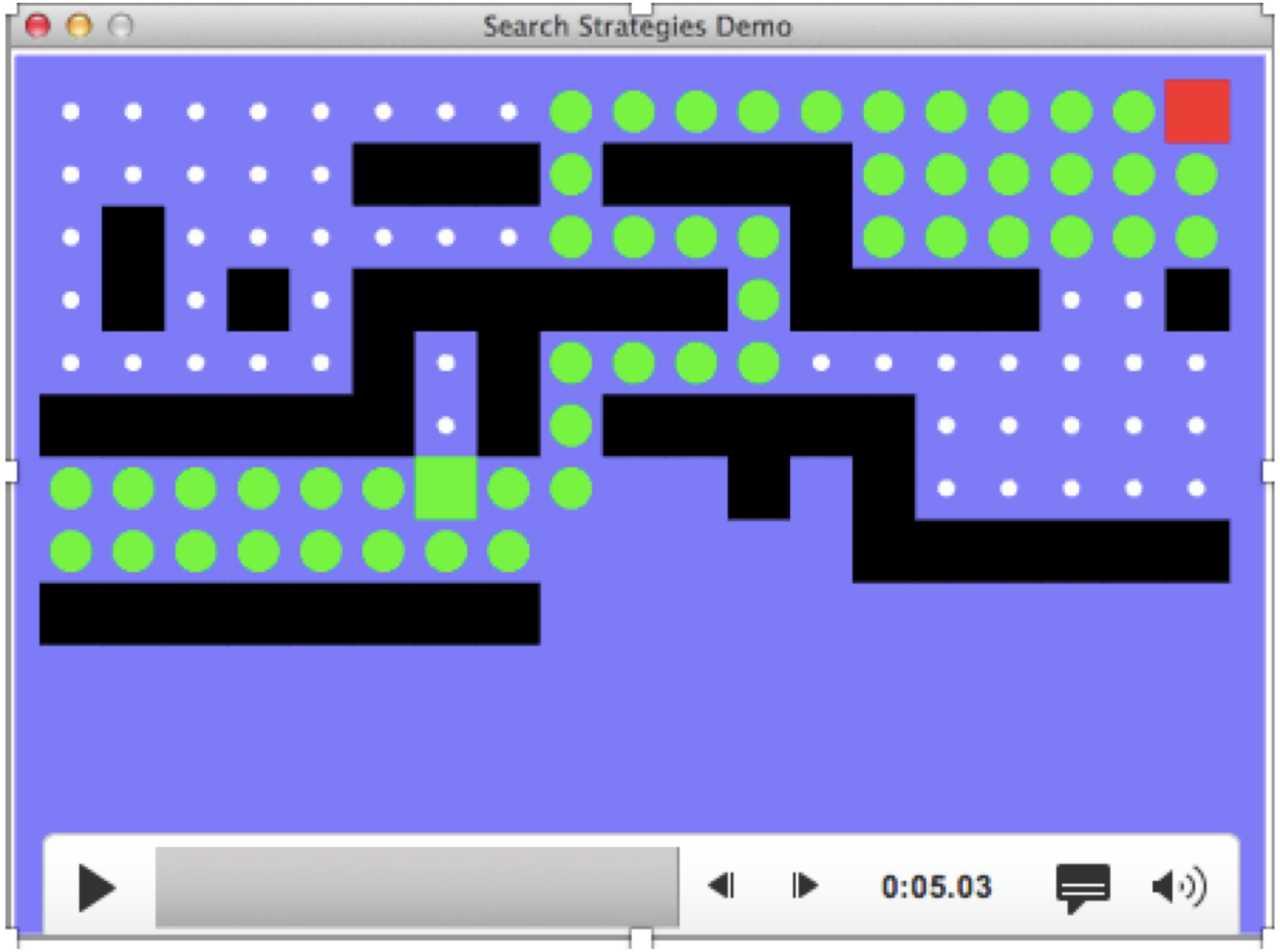
- How many nodes at depth m?



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



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



When to Use BFS vs. DFS?

- The search tree is infinite

BFS ✓ DFS

- We need the shortest path to a solution

BFS ✓ DFS

- There are only solutions at great depth

BFS DFS ✓

- There are some solutions at shallow depth

BFS ✓ DFS

- Memory is Limited

BFS DFS ✓

Real Example: Solving Sudoku

Sudoku Puzzle

	9	3	6	2	8	1	4	
	6						5	
	3			1			9	
	5		8		2		7	
	4			7			6	
	8						3	
	1	7	5	9	3	4	2	

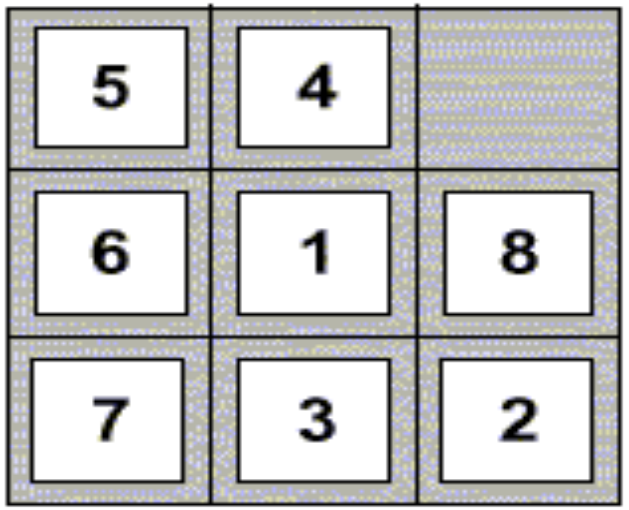
- Start state on the left
- Actions: fill in an allowed number
- Solution: all numbers filled in, with constraints satisfied

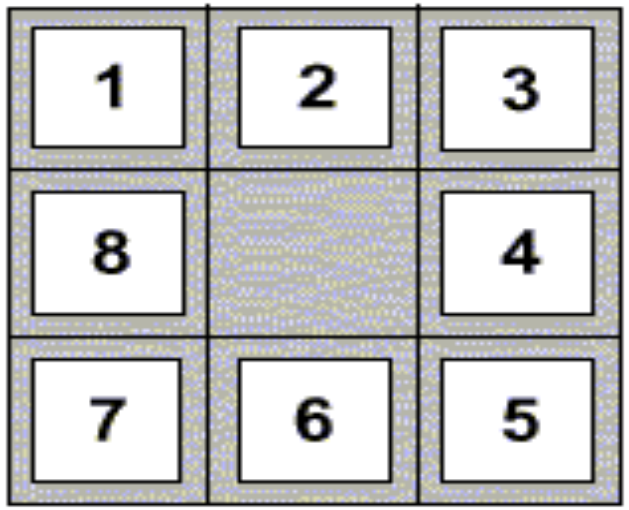
- Which method would you rather use?

BFS

DFS ✓

Real Example: 8-Puzzle

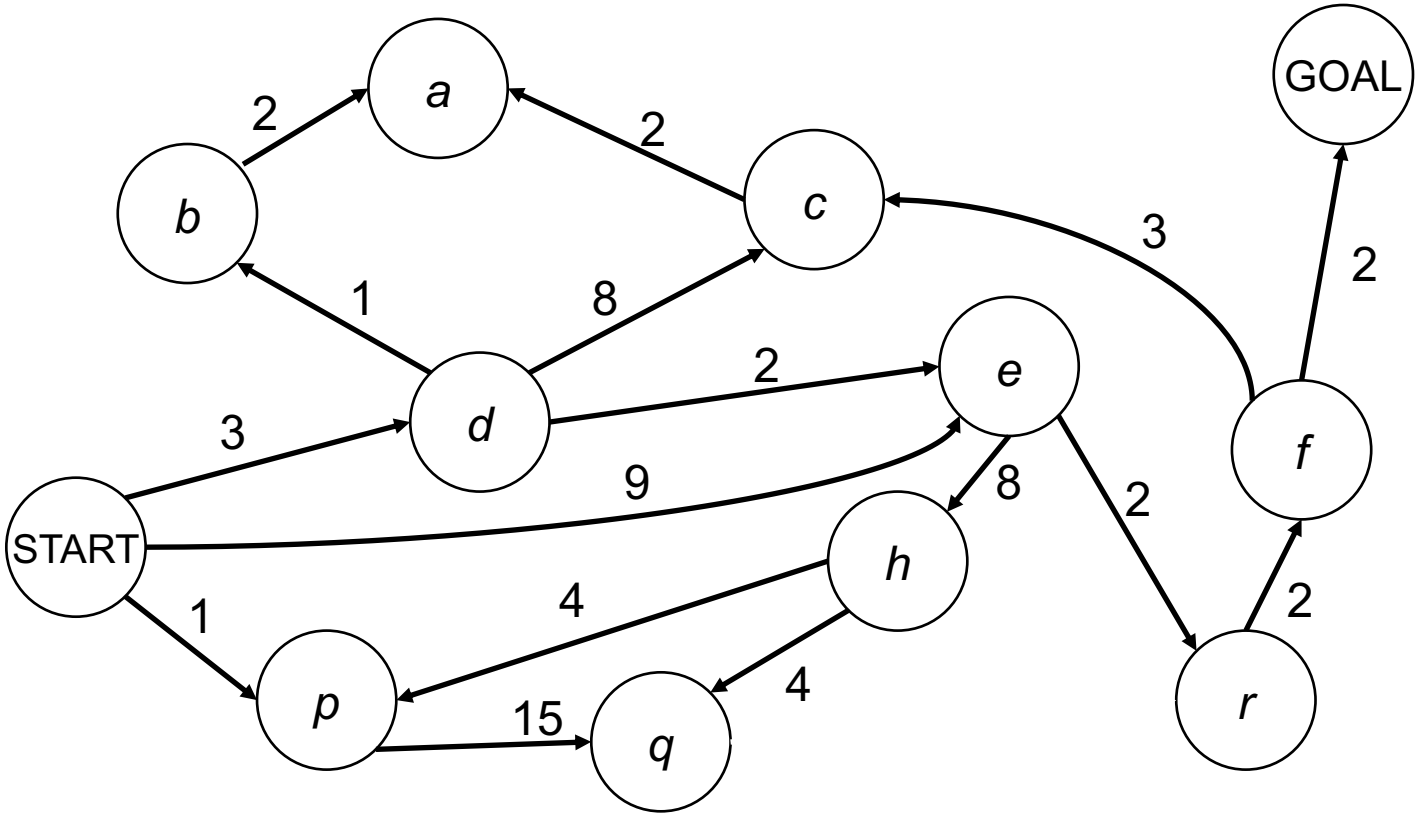




- Which method would you rather use?

BFS ✓ DFS

Cost-Sensitive Search



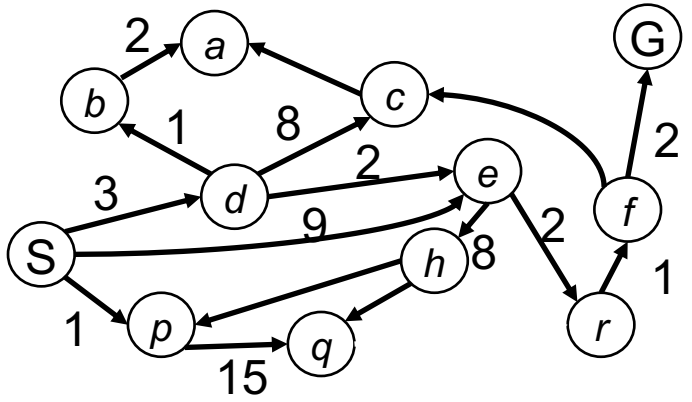
BFS finds the shortest path in terms of number of actions.
It does not find the least-cost path. We will now cover
a similar algorithm which does find the least-cost path.

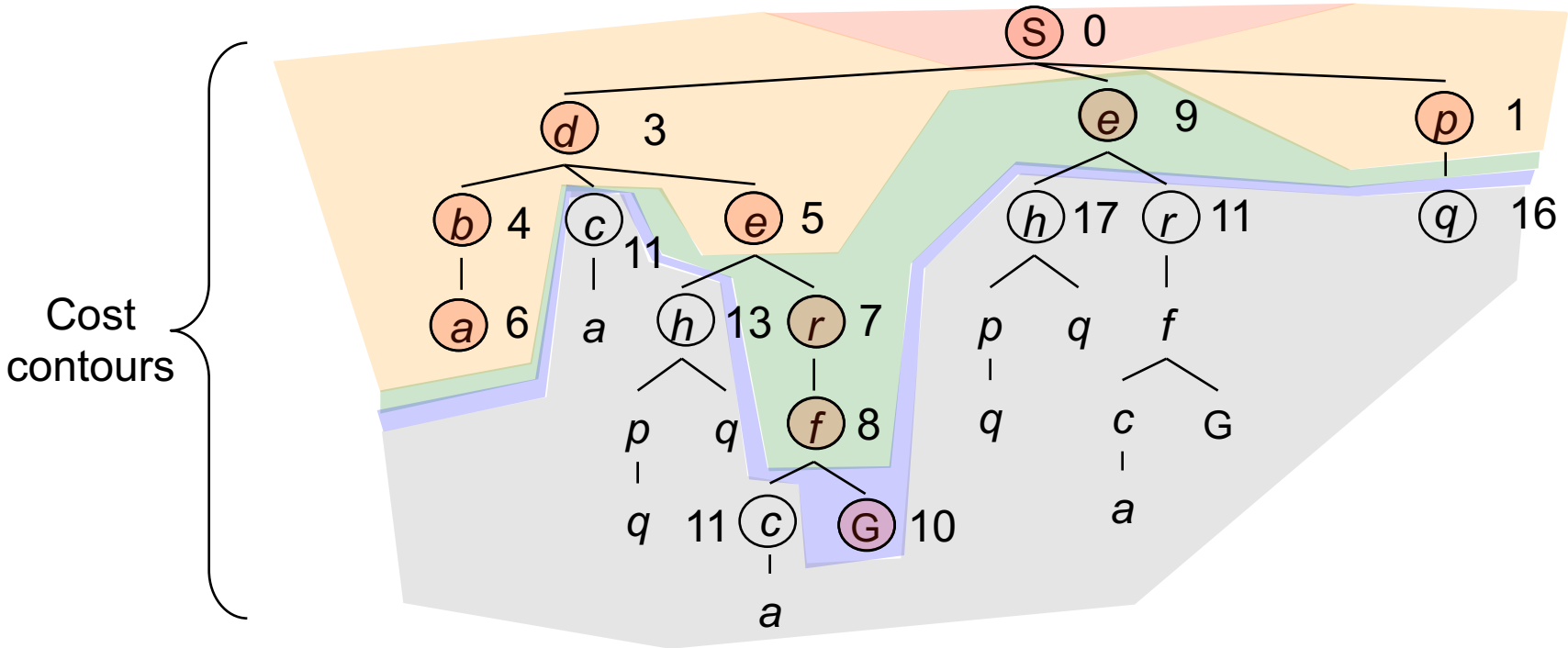
Uniform Cost Search (aka Lowest Cost First Search)

Uniform Cost Search (aka Lowest Cost First Search)

Strategy: expand a cheapest node first:

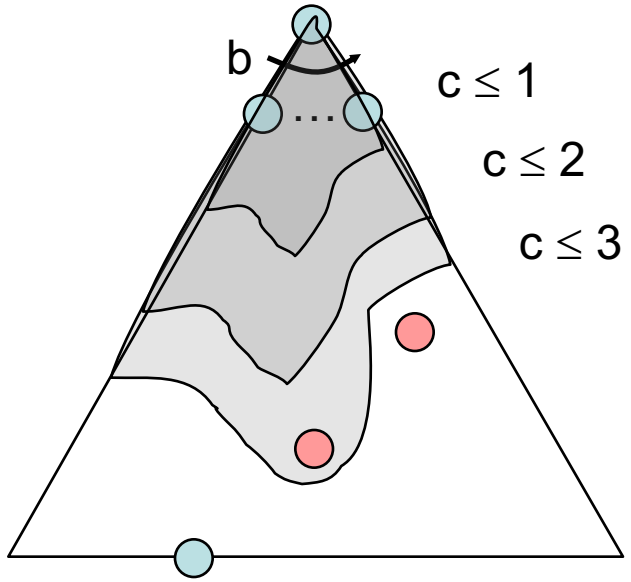
Frontier is a priority queue (priority: cumulative cost)





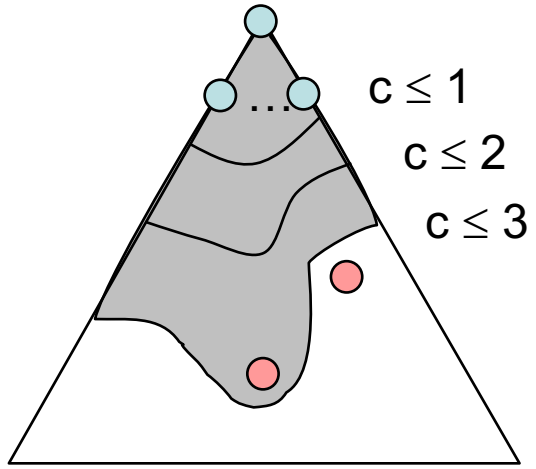
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
- Is it complete?
  - Assuming best solution has a finite cost, minimum arc cost is positive, and the branching factor is finite, yes!
- Is it optimal?
  - Assuming minimum arc cost is positive, yes!

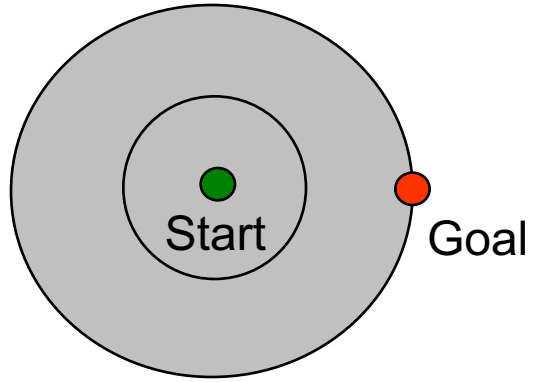


UCS Issues

- 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!

When arc costs are equal, UCS is equivalent to?

DFS

BFS

None of the above

Analysis of Uniform Cost Search

- What is the time complexity of UCS if the maximum path length is *m* and the maximum branching factor is *b*?

  Knowing costs doesn't help here; worst case: all nodes

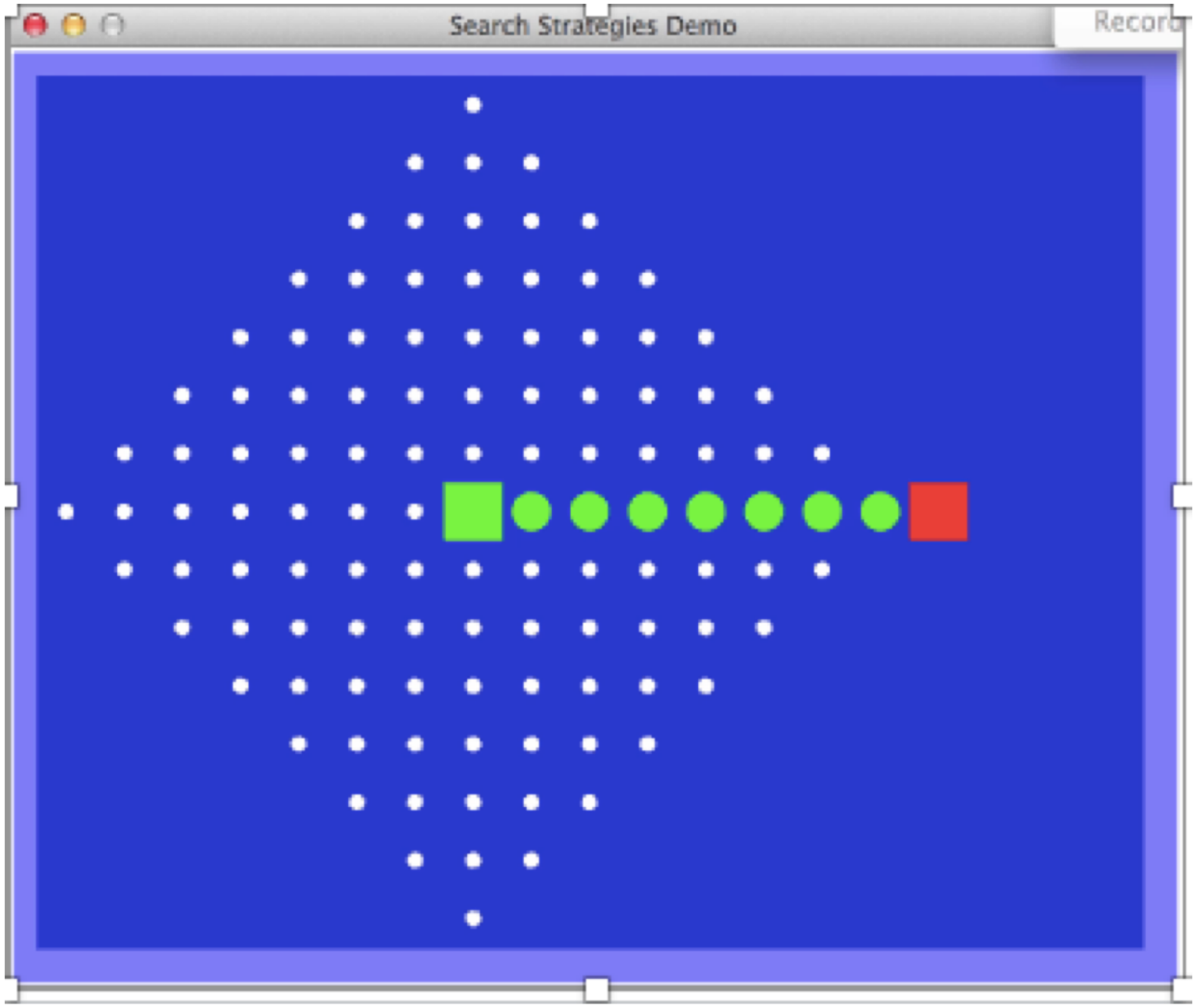
  $O(b^m)$ ✓ $O(m^b)$ $O(bm)$ $O(b+m)$

- What is the space complexity?

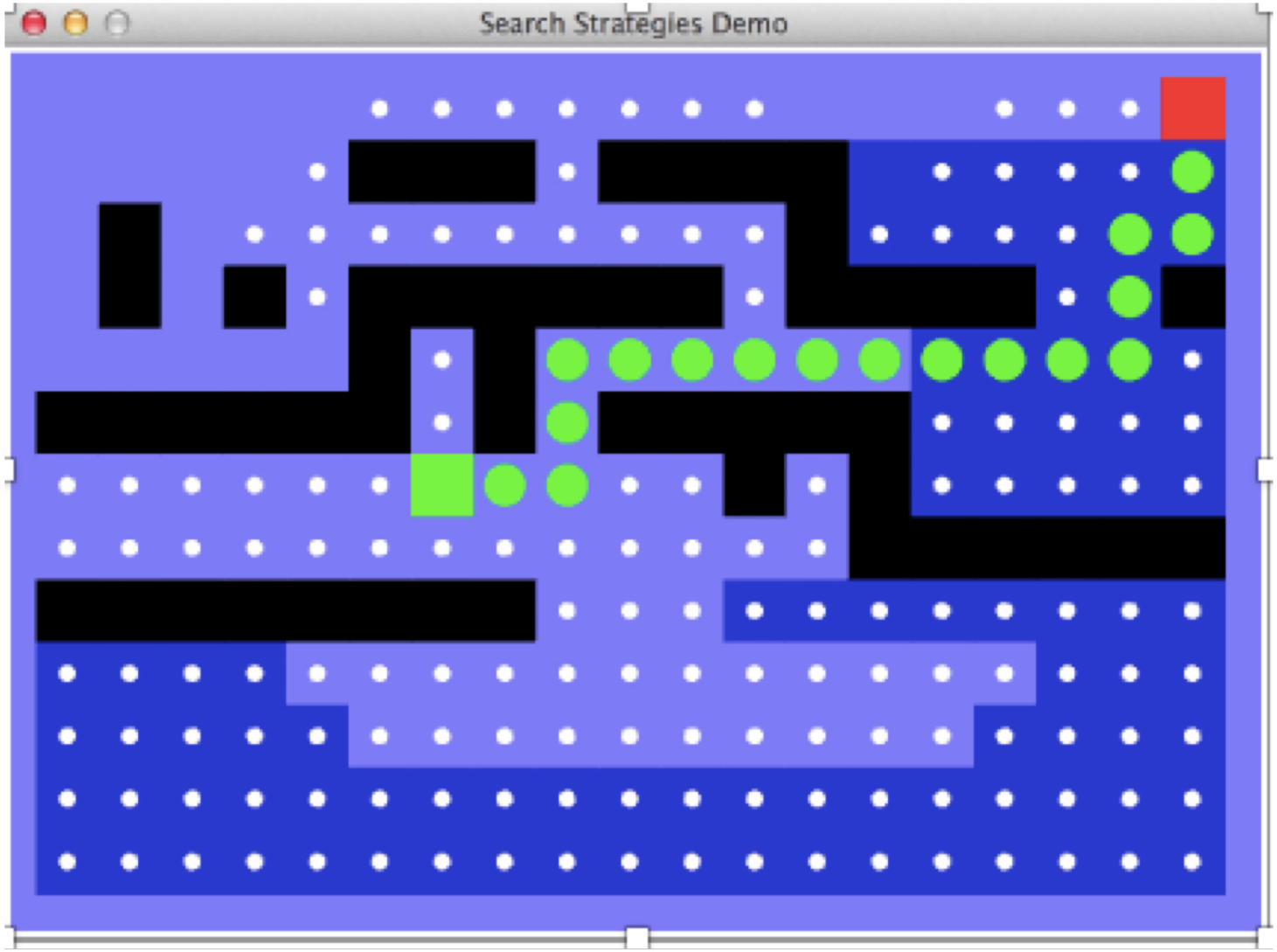
  E.g. uniform cost: just like BFS, in worst case frontier has to store all nodes in the last tier

  $O(b^m)$ ✓ $O(m^b)$ $O(bm)$ $O(b+m)$

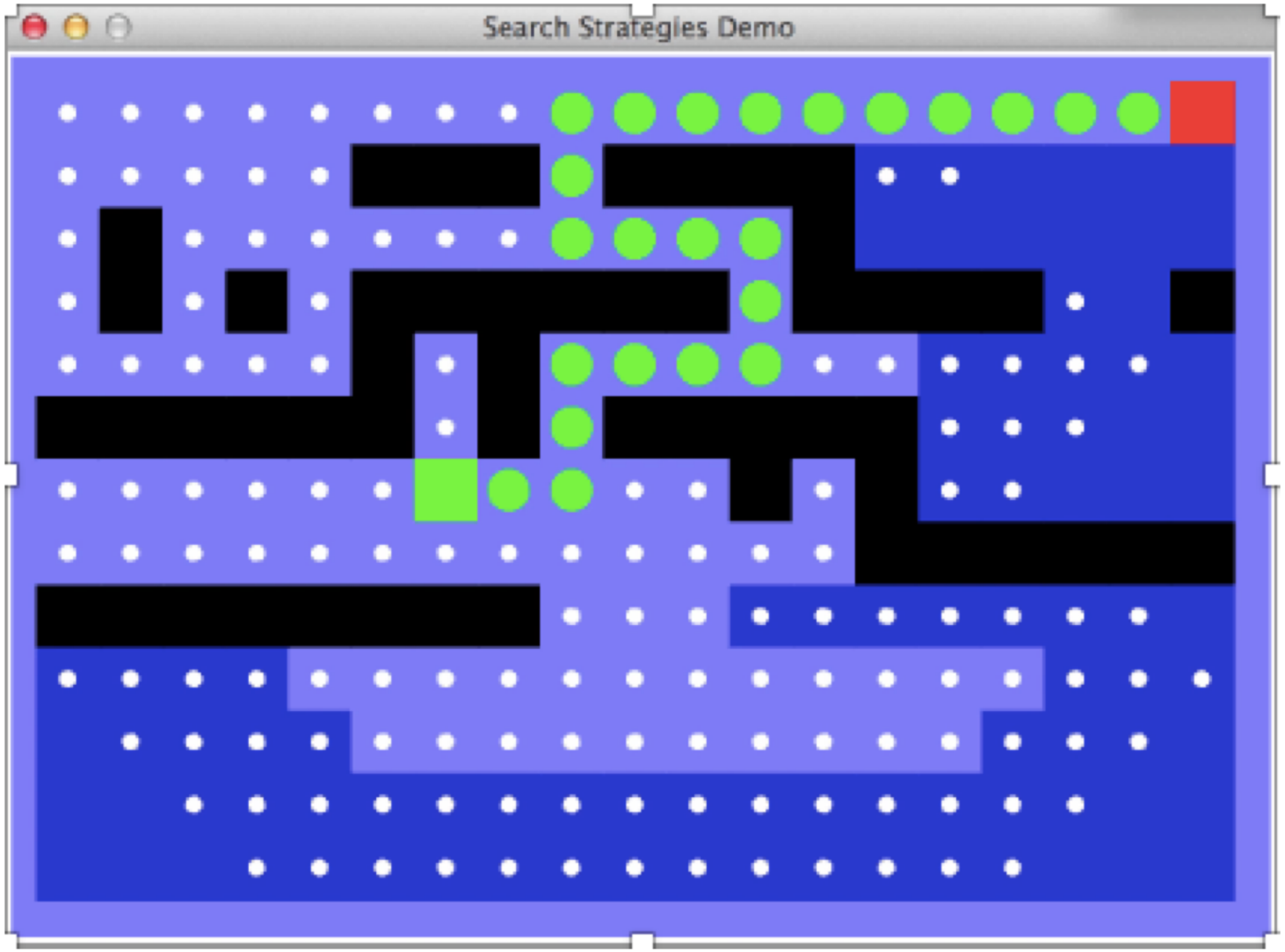
Video of Demo Empty UCS



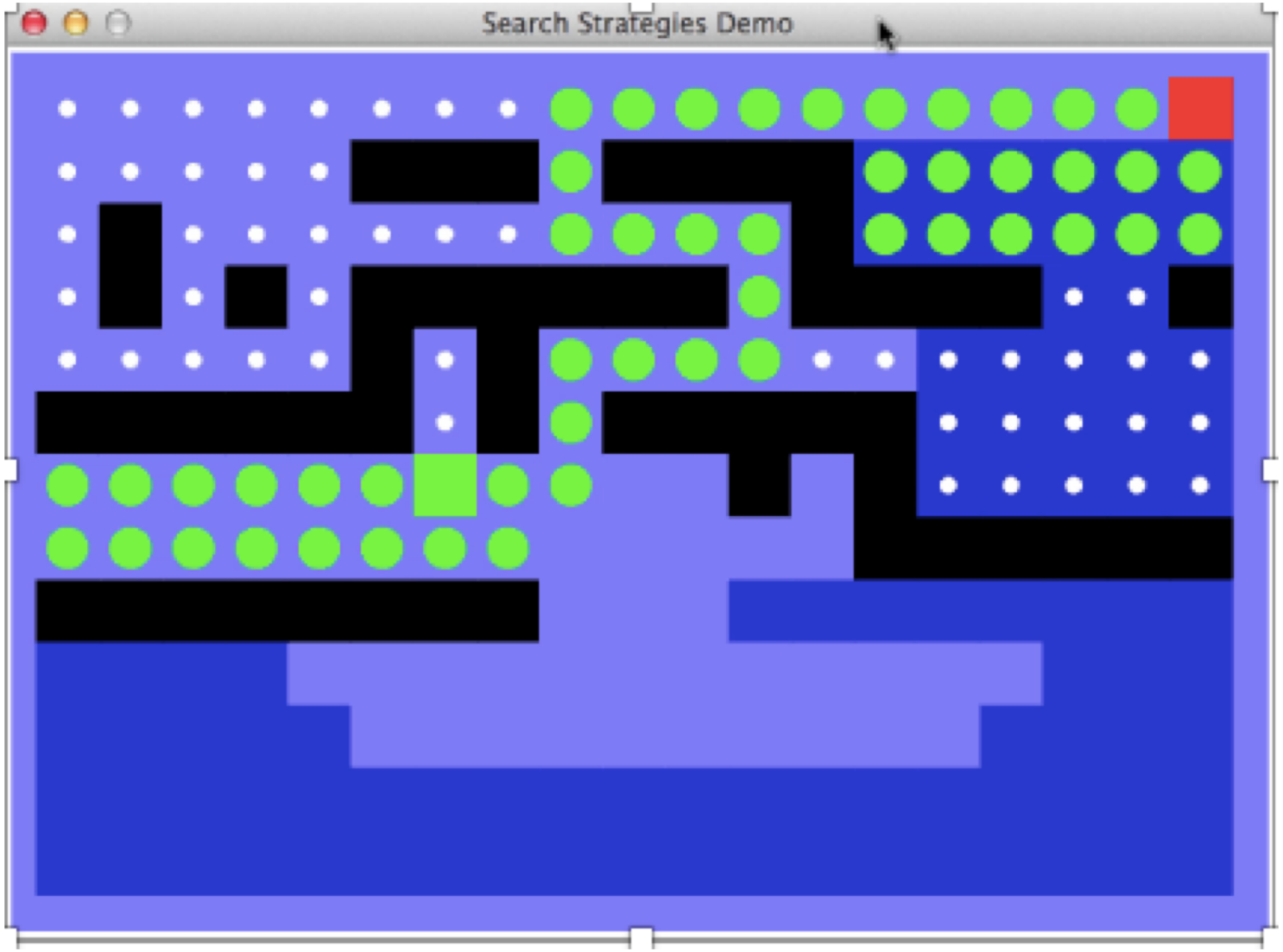
Video of Demo Maze with Deep/Shallow Water --- DFS, BFS, or UCS? (part 1)



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)



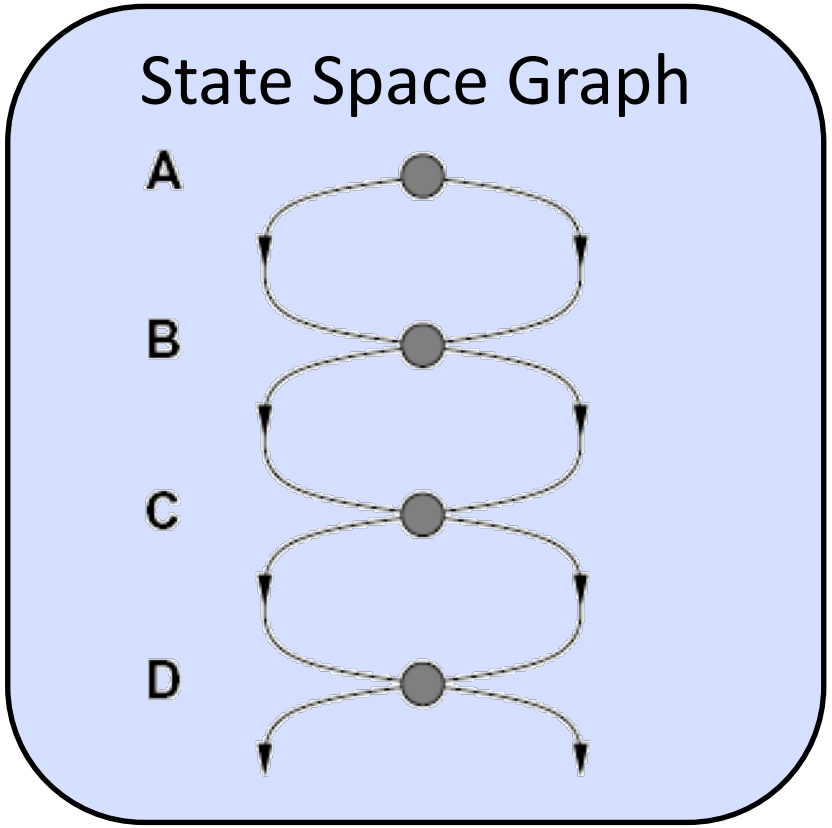
Uninformed Search: DFS, BFS, UCS

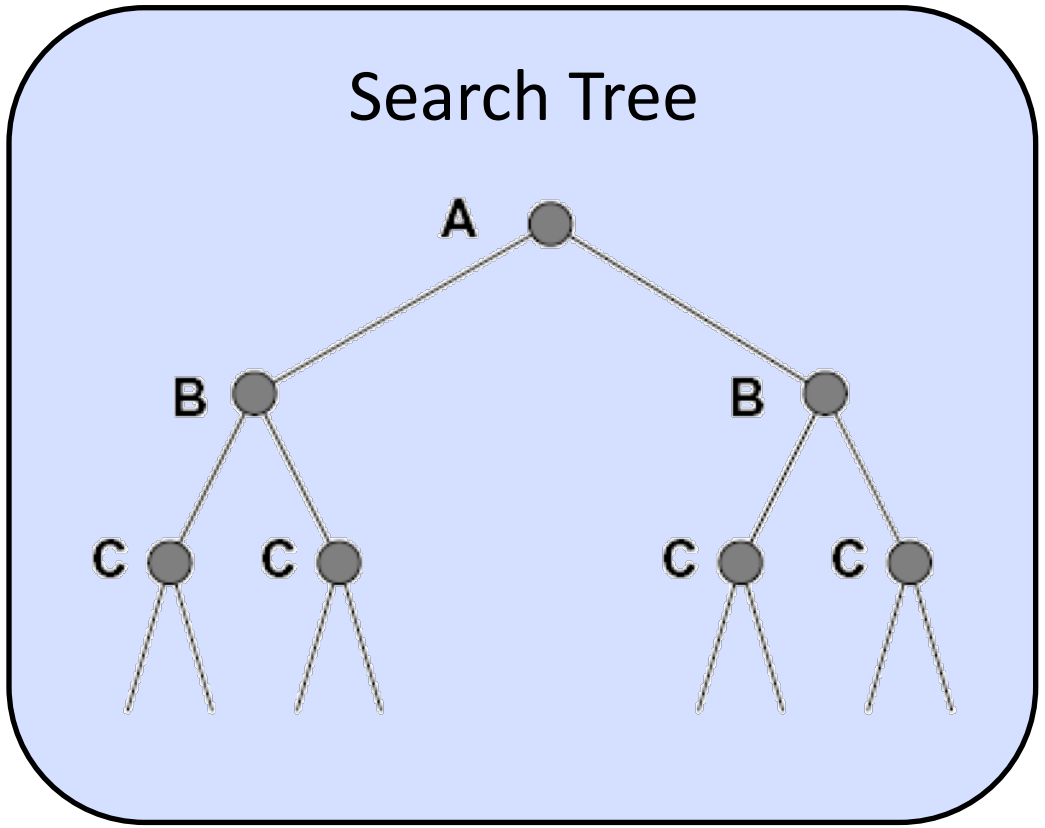
- Why are all these strategies called uninformed?
  - Because they do not consider any information about the states and the goals to decide which path to expand first on the frontier
  - They are blind to the goal

Graph Search vs Tree Search

Tree Search: Extra Work!

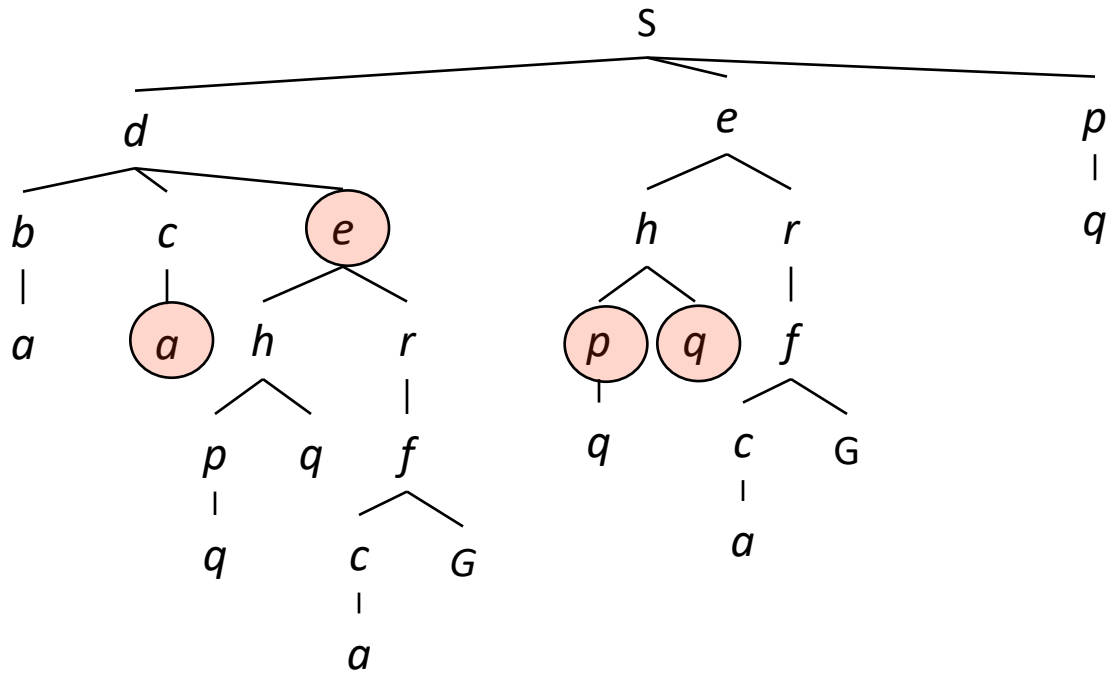
- Failure to detect repeated states can cause exponentially more work.





Graph Search

- In BFS, for example, we shouldn't bother expanding the circled nodes (why?)



Graph Search

- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to the set of expanded states

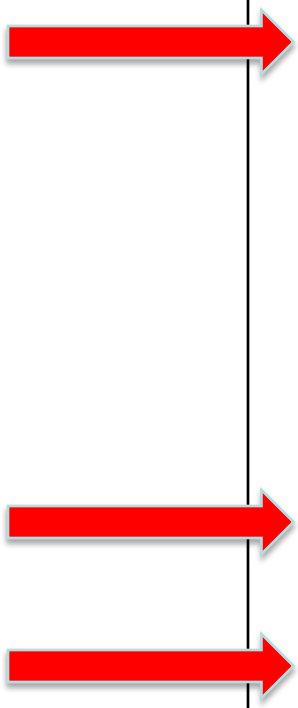
Tree Search Algorithm

```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem

    loop do
        if the frontier is empty then
                return failure
        else choose a node and remove it from the frontier
        if the node contains a goal state then
                return solution

        else expand the chosen node, adding all the neighboring nodes to the frontier
```

Graph Search Algorithm



```
function GRAPH-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    initialize the explored set to be empty
    loop do
        if the frontier is empty then
                return failure
        else choose a node and remove it from the frontier
        if the node contains a goal state then
                return solution
        else add the node to the explored set
            expand the chosen node, adding all the neighboring nodes to the frontier
                but only if the child is not already in the explored set
```

Theorem: each state appears at most once in the search tree constructed

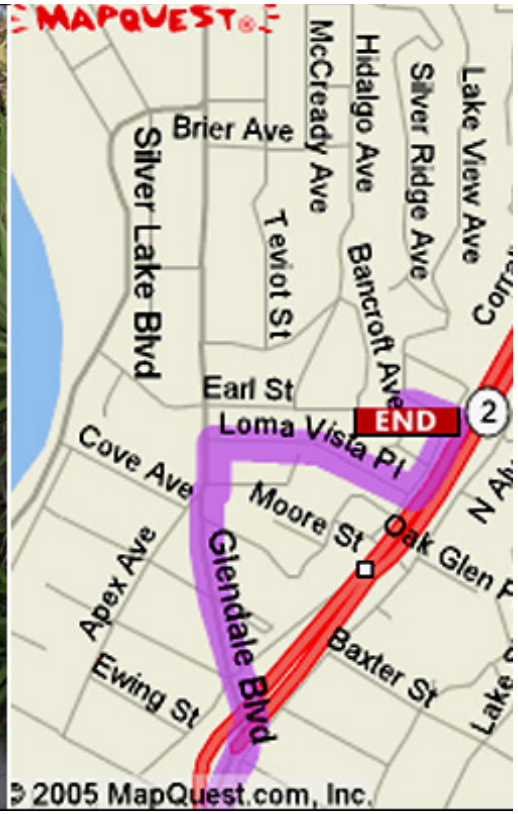
Tree Search vs Graph Search

- Graph search
  - Avoids infinite loops
  - Eliminates exponentially many redundant paths
  - Requires memory proportional to its runtime!

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?



Search methods so far

	Complete	Optimal	Time	Space
DFS	N (Y if no cycles)	N	$O(b^m)$	$O(mb)$
BFS	Y	Y	$O(b^m)$	$O(b^m)$
UCS	Y Arc Costs > 0	Y Arc Costs > 0	$O(b^m)$	$O(b^m)$

Reading

- Chapters 3.1-4 in the AIMA textbook