Fundamentals of Artificial Intelligence

Lecture-2
Problem Solving

Debela Desalegn

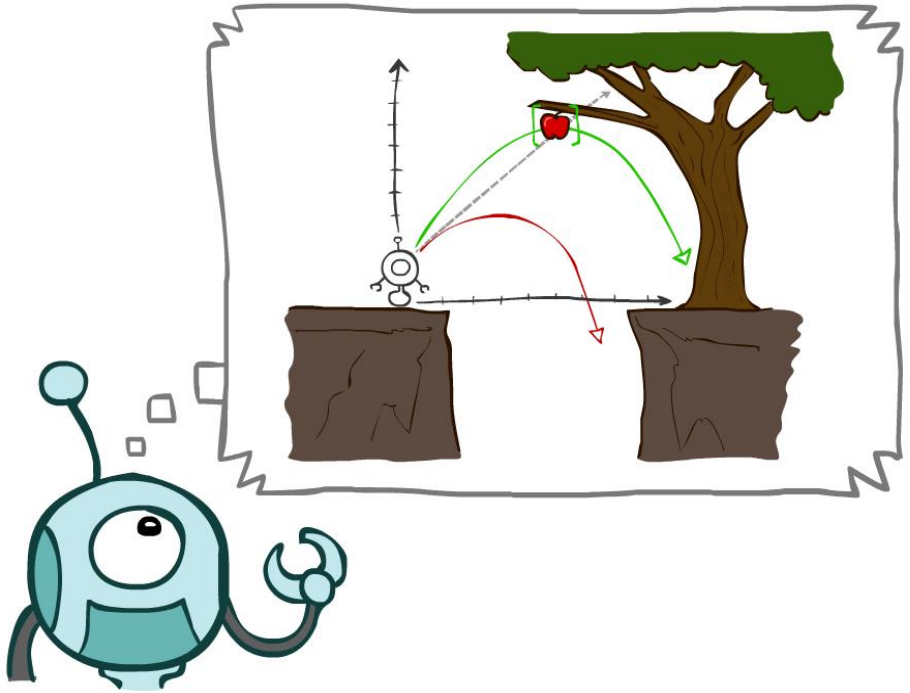
March 16, 2025

Today we will see:

- Search Problems
- Uninformed Search Methods
  - Depth-First Search (DFS)
  - Breadth-First Search (BFS)
  - Uniform-Cost Search(UCS)

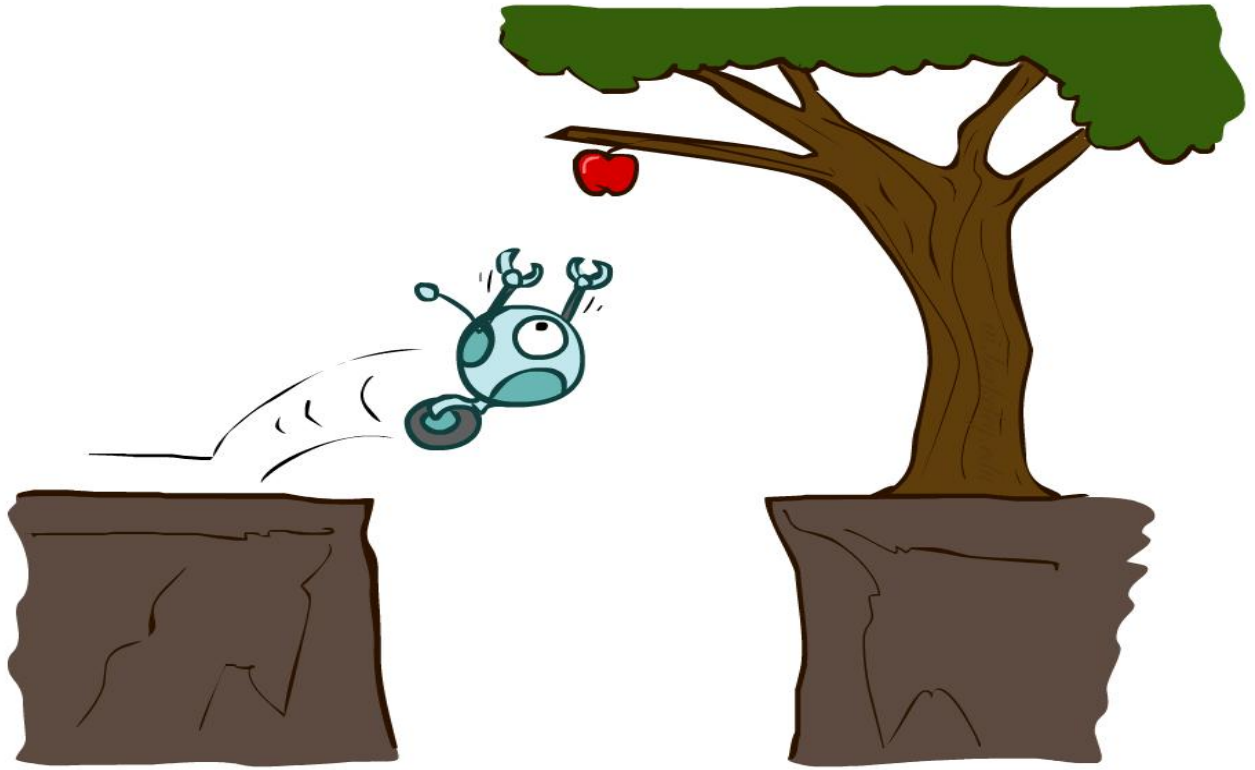
Intro

- Problem Solving Agent: An intelligent agent that finds solutions by searching for the best sequence of actions.
- Search Algorithms: Informed vs Uninformed
- Assumption: Known world
- Problem Solving Phases
  - Goal Formulation
  - Problem Formulation (Abstraction)
  - Search (simulated)
  - Execution (Closed Loop vs Open Loop)



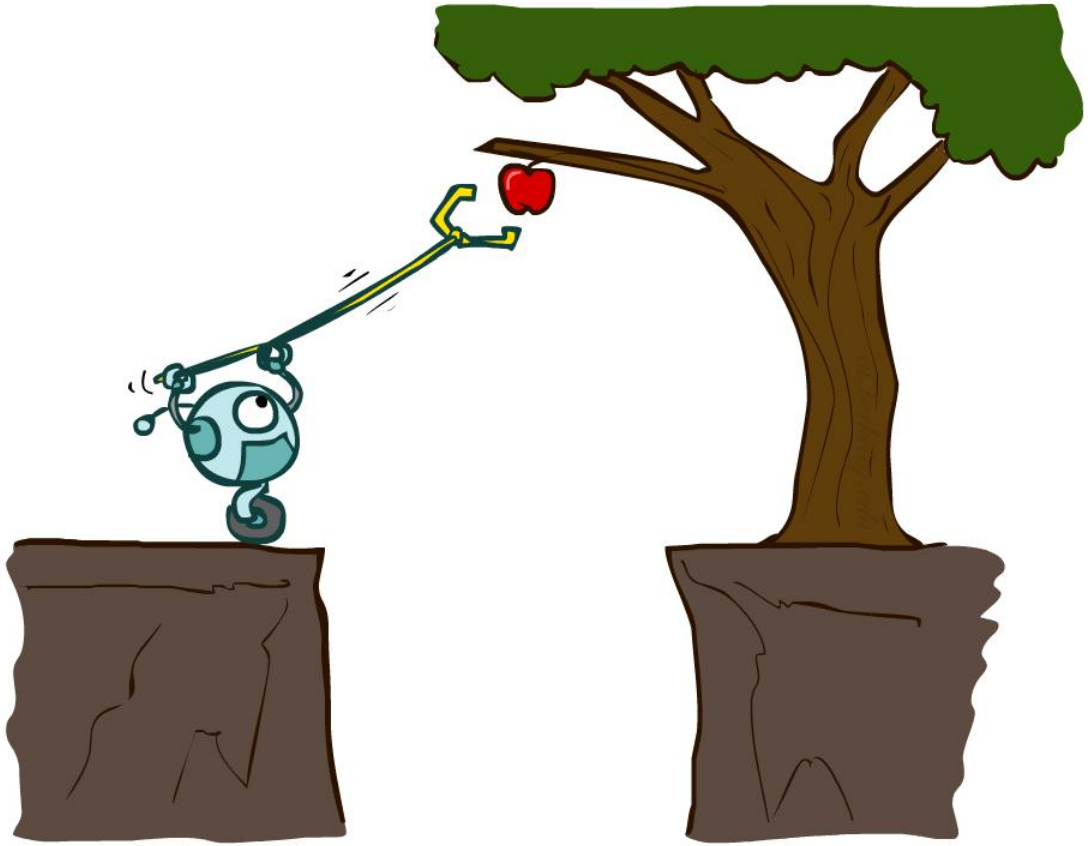
Intro

- 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?



Intro

- 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



Search Problems

State: Represents a configuration of the problem.

- Initial State: Where the agent starts.
- Goal State: The desired outcome.

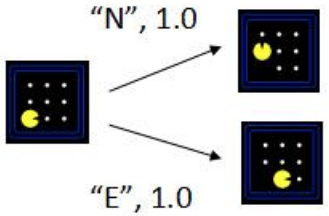
- **State Space:** The set of all possible states reachable from the initial state.
- **Actions:** The possible moves the agent can take (e.g., moving a piece in a puzzle, turning left in a navigation problem).
- **Transition Model:** Defines how an action transforms the current state into a new state.
- **ACTION-COST**(s, a, s') or c(s, a, s'): Measures the cost of performing action a in state s to reach state s'.
- **Path:** A sequence of actions leading from the initial state to the other state.
- **Solution:** a sequence of actions (a plan/path) which transforms the start state to a goal state
- **Optimal Solution** == Lowest ACTION-COST

Search Problems

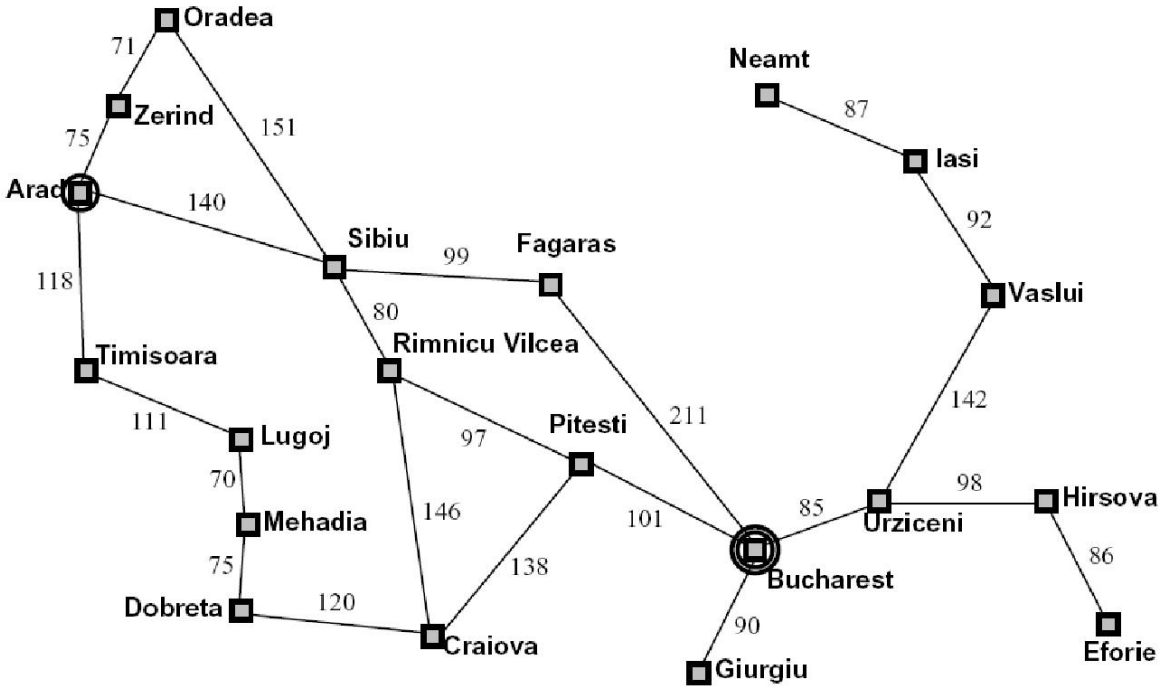
A search problem consists of:

- ✓ A state space

- ✓ A successor function (with actions, costs)



- ✓ A start state and a goal test



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 last detail of the environment.
A search state/search tree 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
- Goal test: dots all false

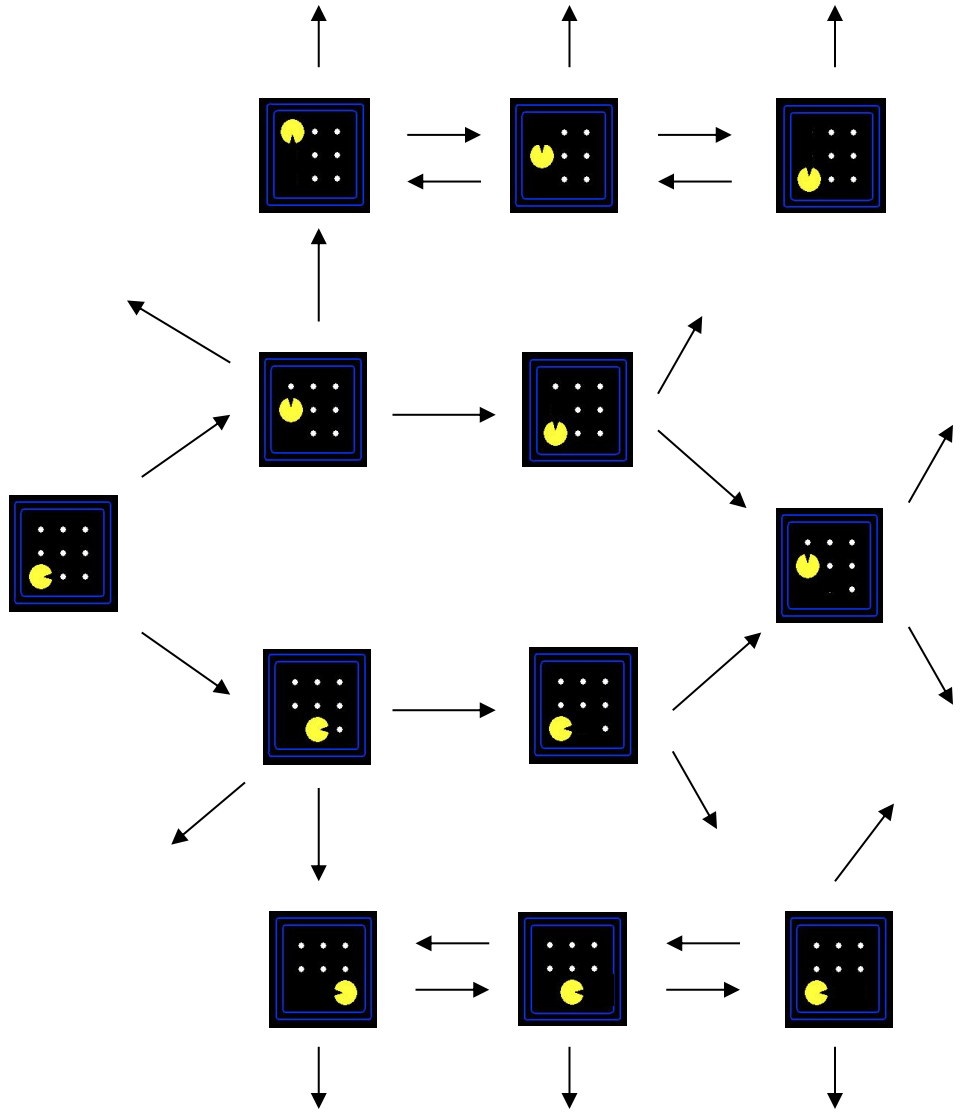
State Space Graphs

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



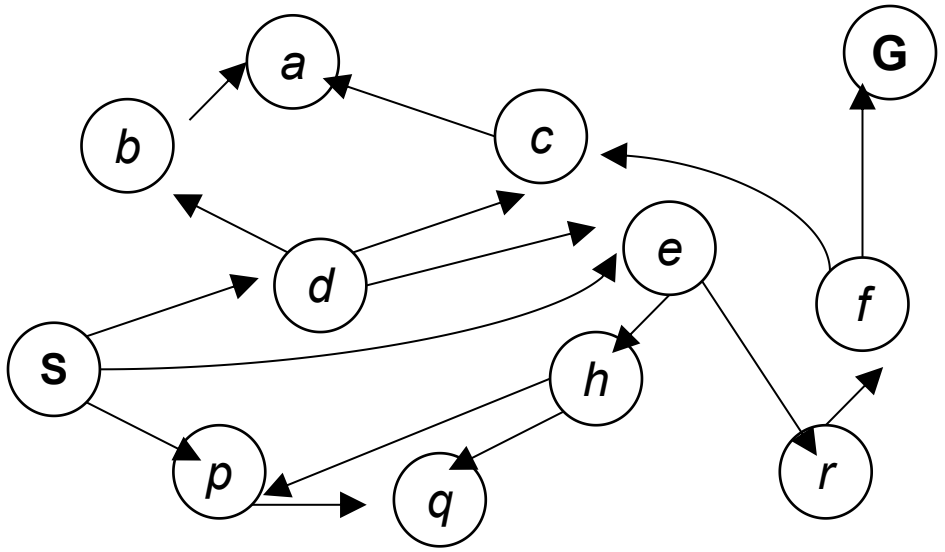
State Space Graphs

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)

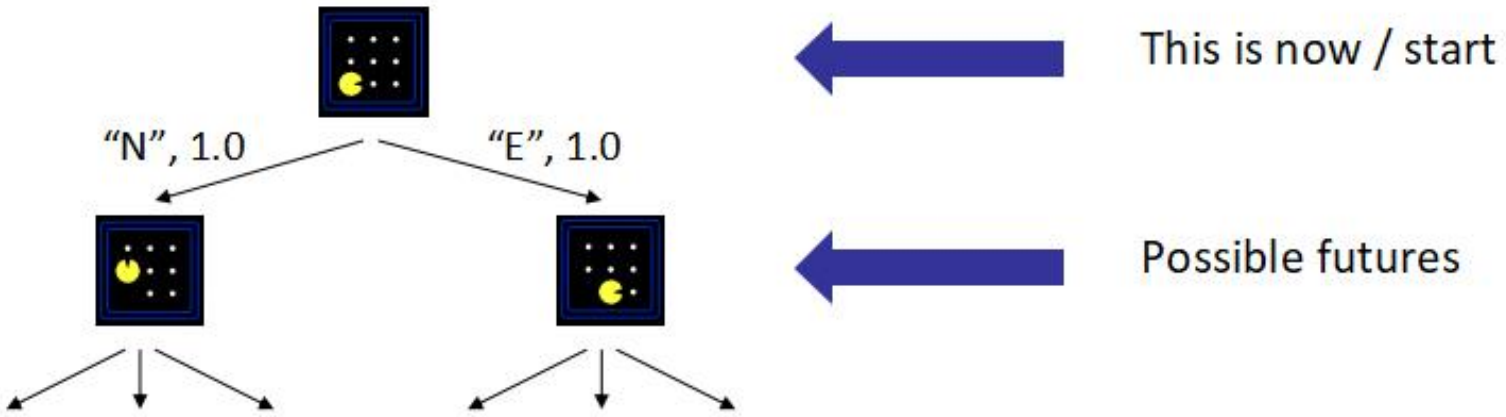
- In a state space graph, each state occurs only once!

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



Tiny state space graph for a tiny search problem

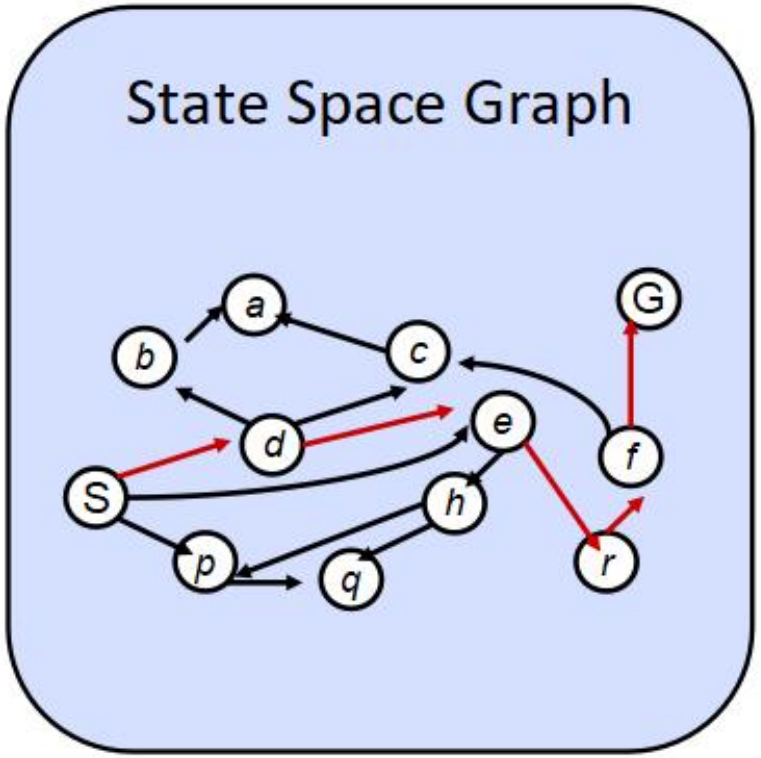
Search Trees



A search tree:

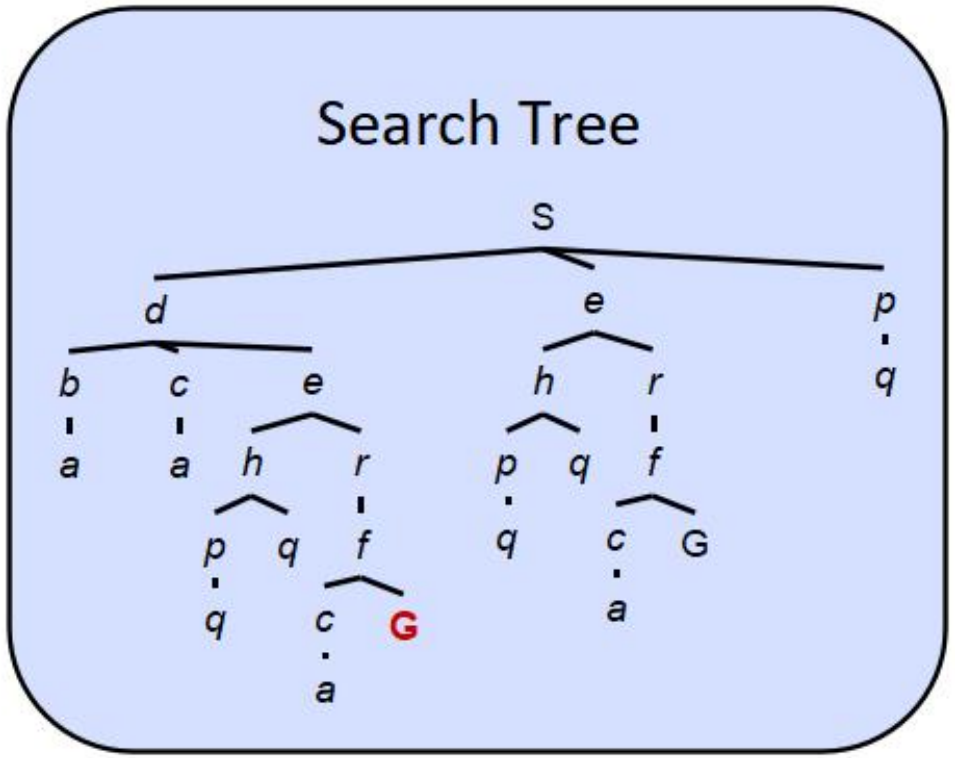
- A "what if" tree of plans and their outcomes
- The start state is the root node
- Children correspond to successors
- Nodes show states, but correspond to PLANS that achieve those states
- For most problems, we can never actually build the whole tree

State Space Graphs vs. Search Trees



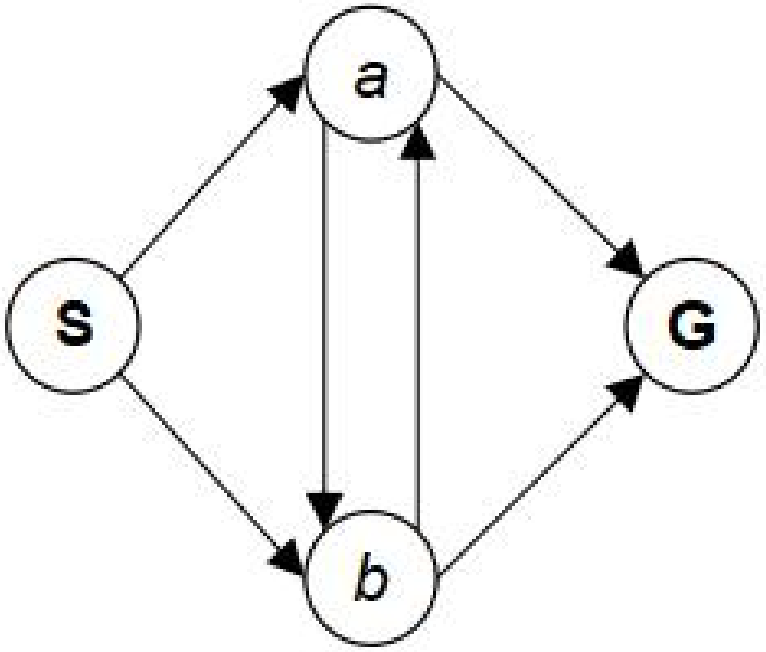
Each NODE in the search tree is an entire PATH in the state space graph.

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

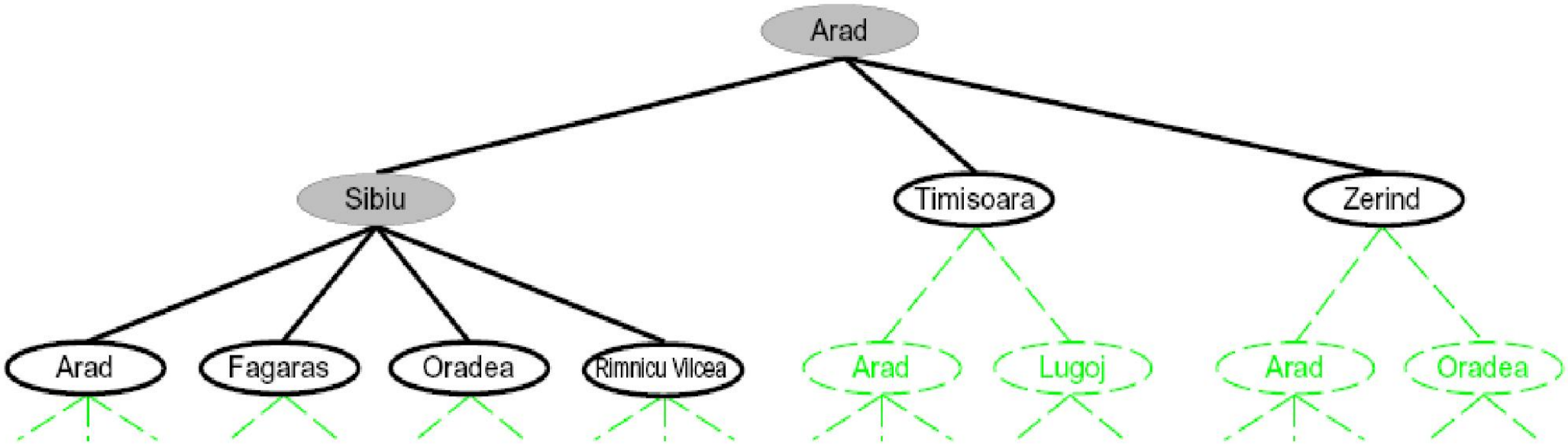


Quiz-1

How big is its search tree (from S)?



Tree Search



Search

- Expand out potential plans (tree nodes)
- Maintain a fringe/frontier 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
```

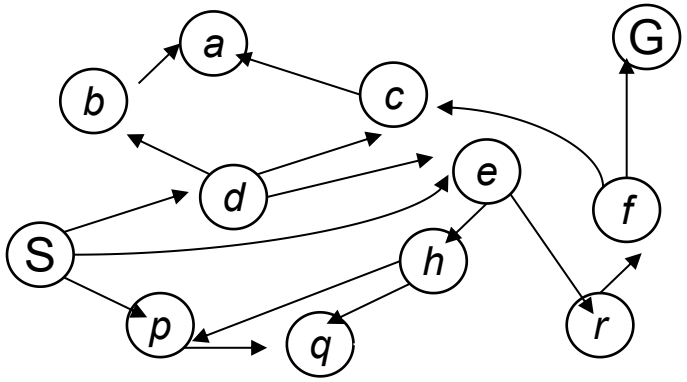
Important ideas:

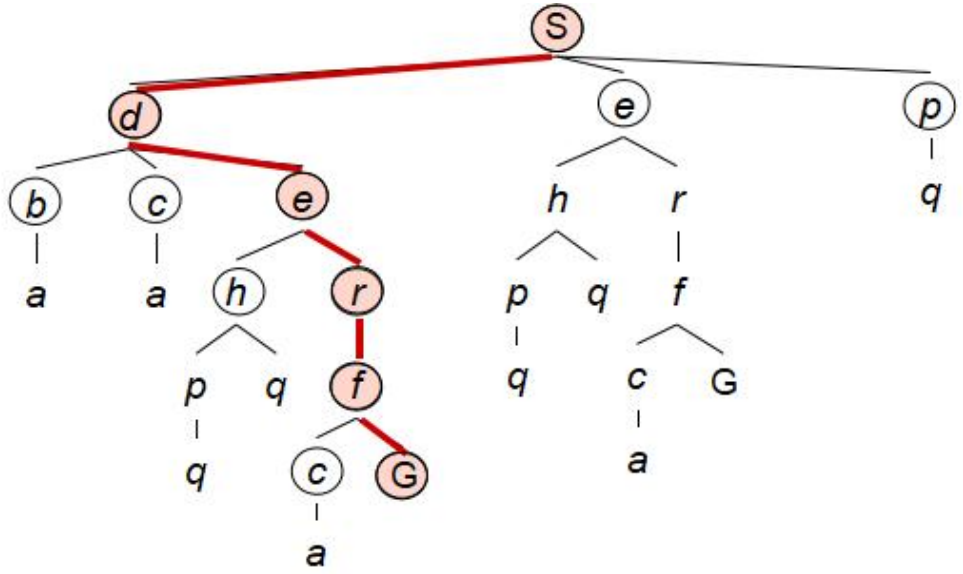
- Fringe/Frontier
- Expansion
- Exploration strategy

Main question: Which fringe nodes to explore?

The essence of 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~~

Search Data structure

- node.STATE: the state to which the node corresponds;
- node.PARENT: the node in the tree that generated this node;
- node.ACTION: the action that was applied to the parent's state to generate this node;
- node.PATH-COST: the total cost of the path from the initial state to this node. In mathematical formulas, we use as a synonym for PATH-COST.

- A data structure for a frontier could be a Queue (Priority Queue, FIFO Queue, LIFO queue (stack)).

Uninformed Search Strategies

No clue about how close a state is to the goal(s).

Have information on how to traverse or visit the nodes in the tree.

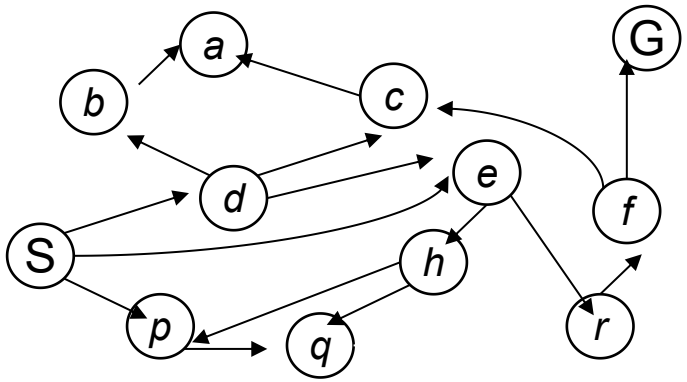
- ✓ Depth-First Search (DFS)
- ✓ Breadth-First Search (BFS)
- ✓ Uniform-Cost Search(UCS)

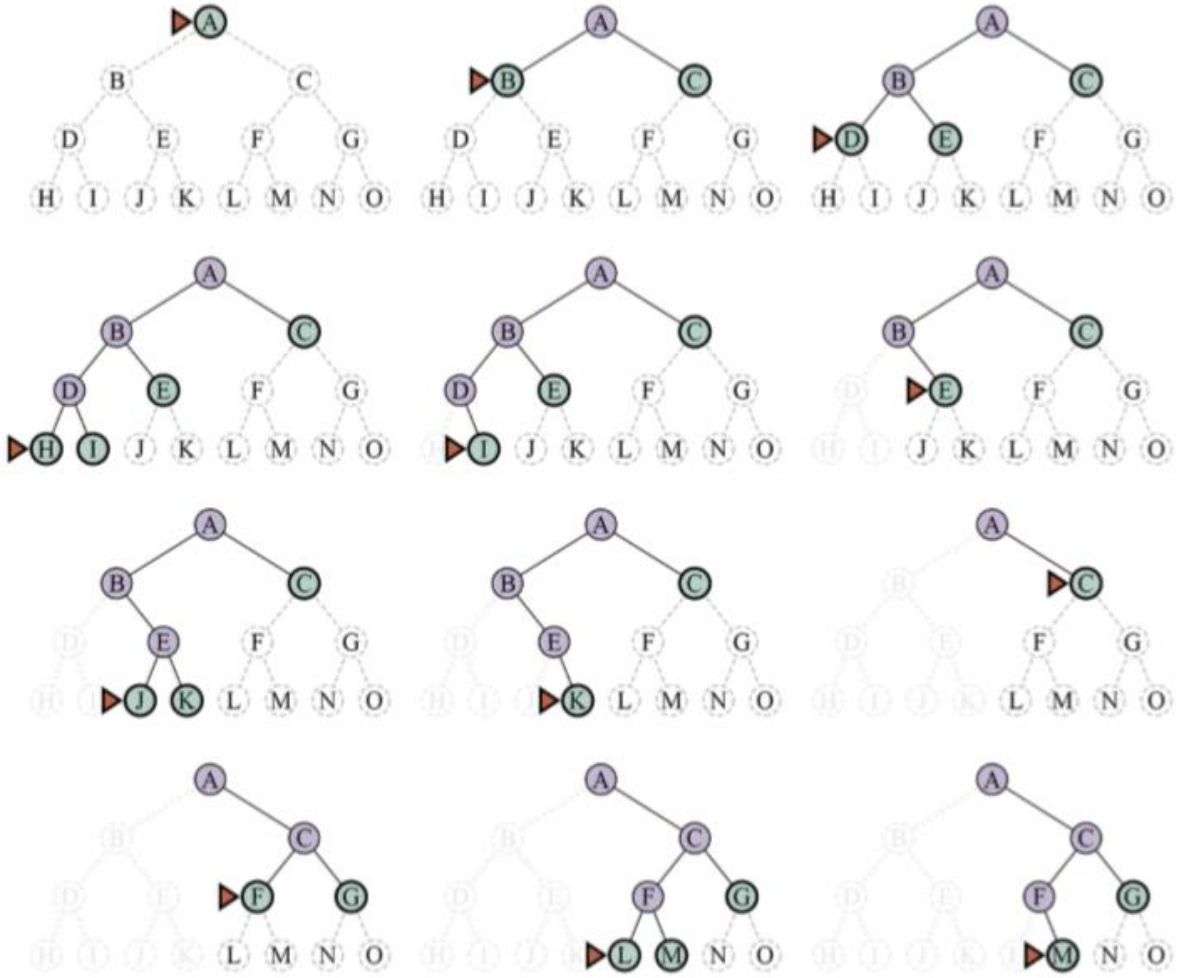
Depth-First Search (DFS)

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack

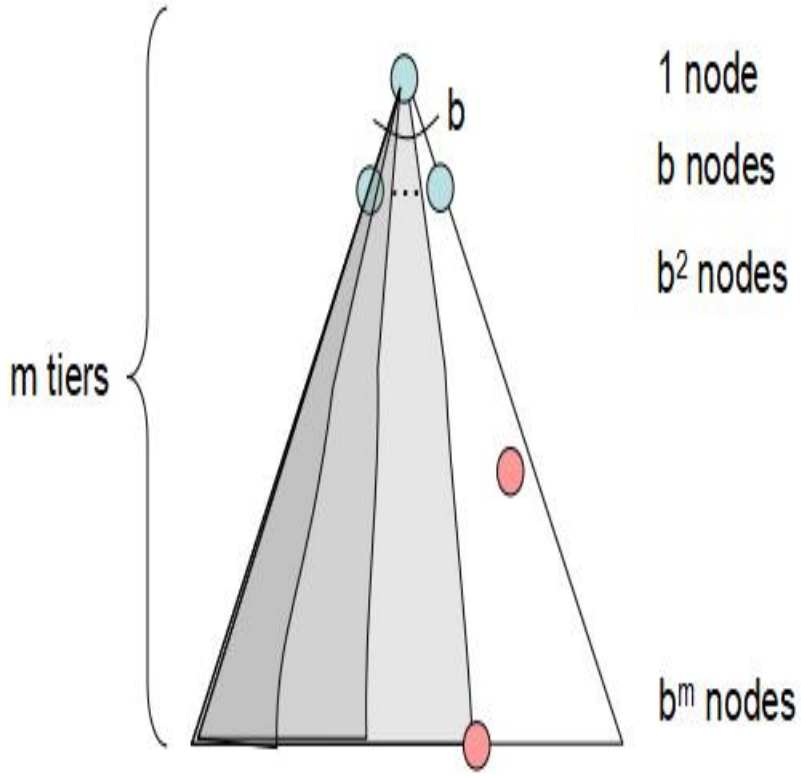
Depth-First Search (DFS)





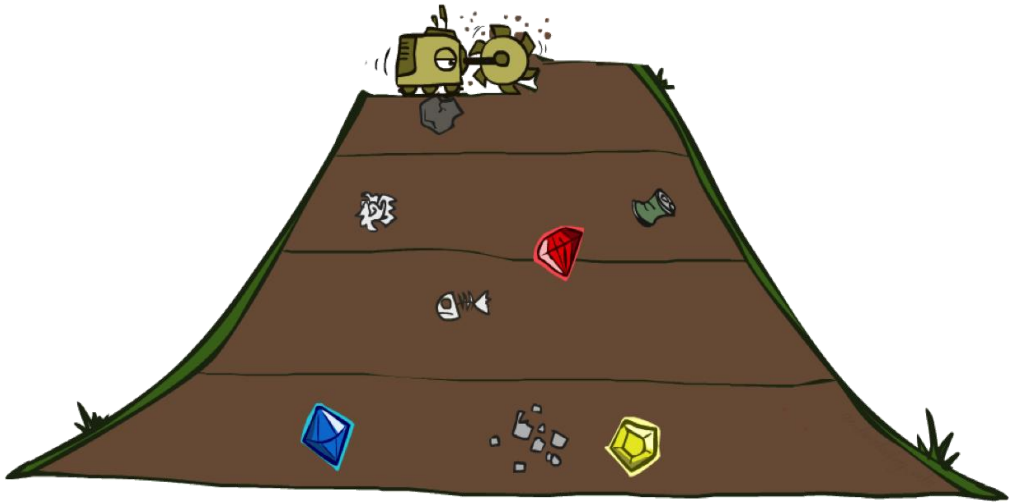
Depth-First Search (DFS) Properties

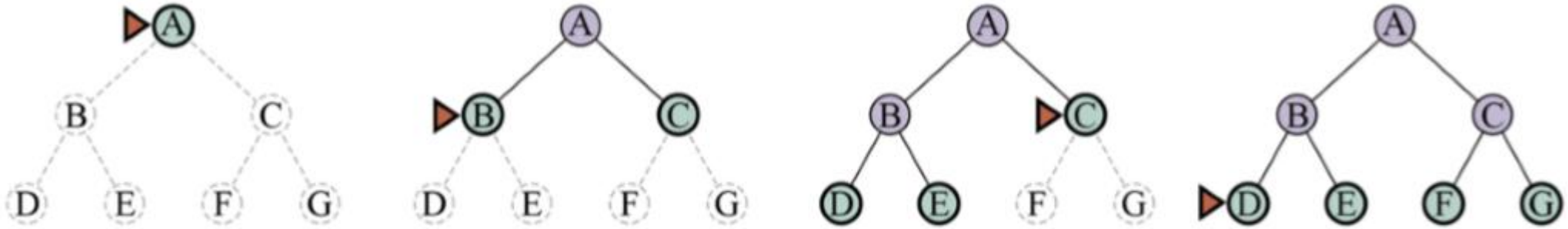
- What nodes DFS expand?
  - Some left prefix of the tree.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the fringe take?
  - Only has siblings on path to root, so O(bm)
- Is it complete?
  - m could be infinite, so only if we prevent cycles (more later)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



Breadth-First Search (BFS)

- Optimal when all action cost is the same.
- Node Expansion sequence
  - Root, successors of the root, next successors
  - Complete in infinite state spaces.
  - Is Best first search where f(n) is the depth.
  - Good efficiency → FIFO queue



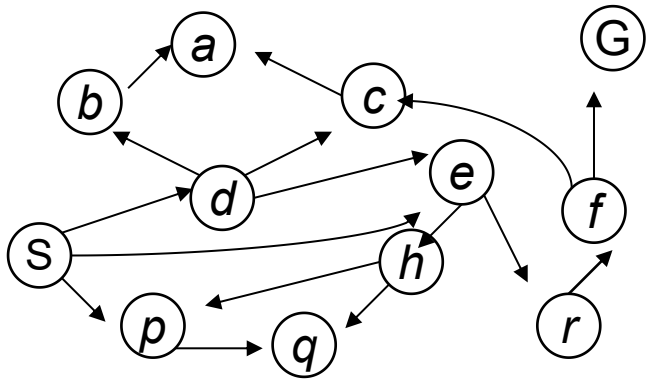


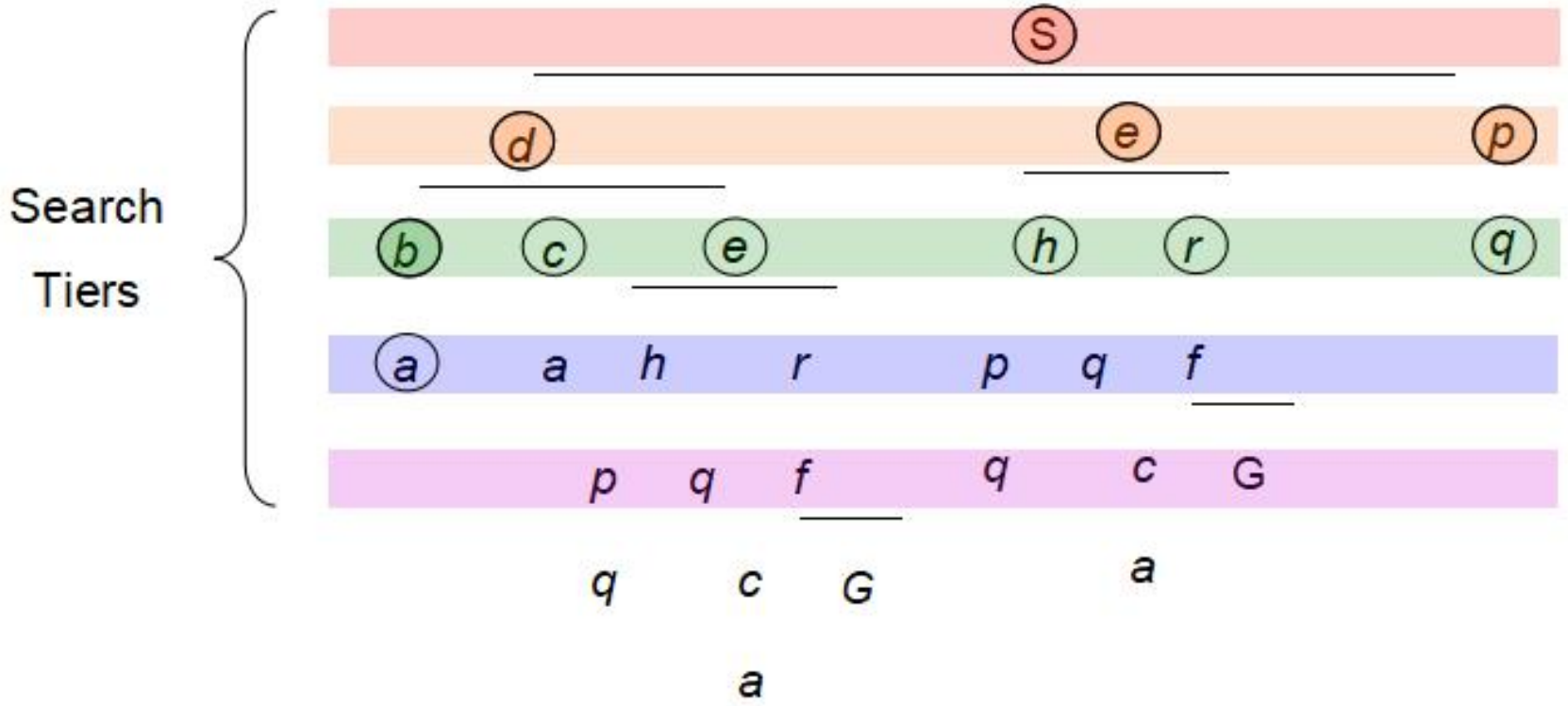
Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by the triangular marker.

Breadth-First Search (BFS)

Strategy: *expand a shallowest node first*

Implementation: *Fringe is a FIFO queue*



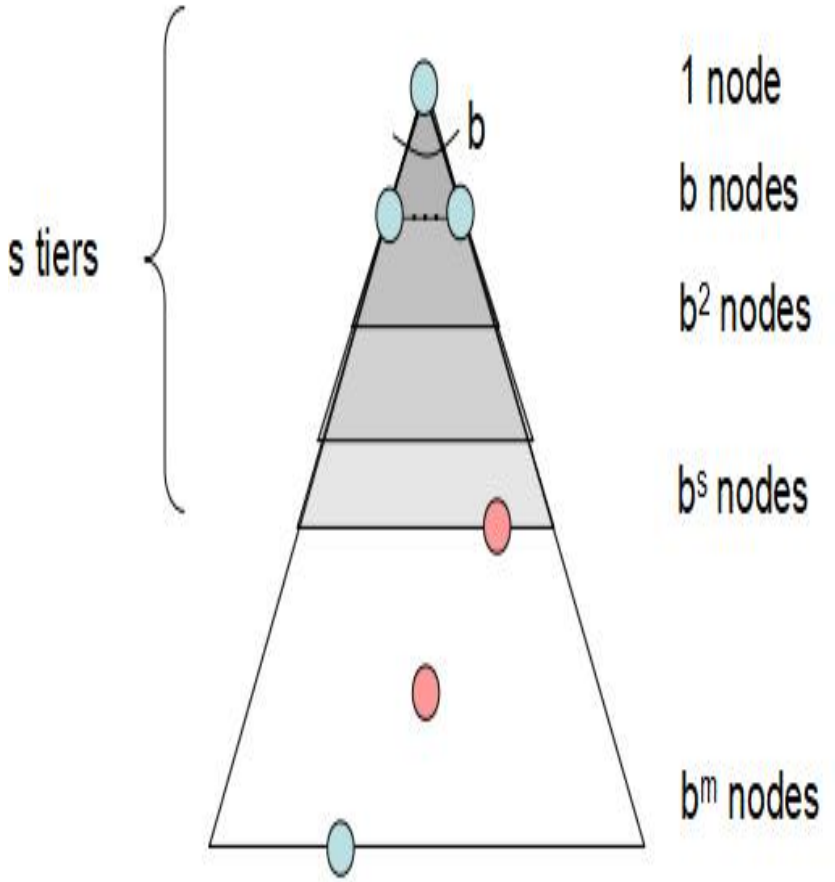


Breadth-First Search (BFS)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
    node ← NODE(problem.INITIAL)
    if problem.IS-GOAL(node.STATE) then return node
    frontier ← a FIFO queue, with node as an element
    reached ← {problem.INITIAL}
     while not IS-EMPTY(frontier) do
        node ← POP(frontier)
        for each child in EXPAND(problem, node) do
            s ← child.STATE
            if problem.IS-GOAL(s) then return child
            if s is not in reached then
                add s to reached
                add child to frontier
    return failure
```

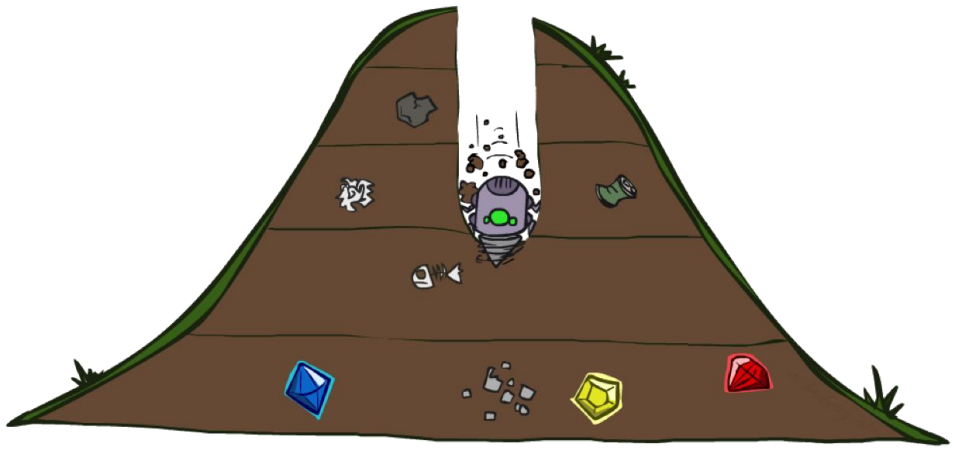
Breadth-First Search (BFS)

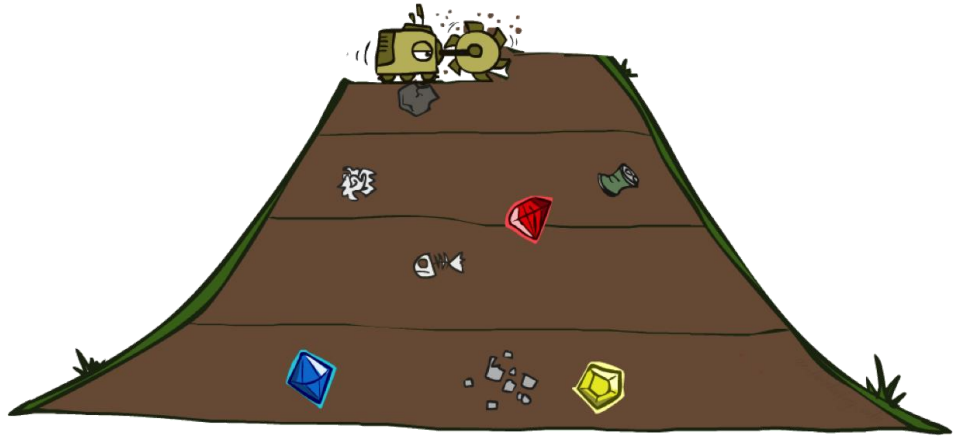
- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time **$O(b^s)$**
- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^s)$
  - The memory requirements are higher than time requirements.
- Is it complete?
  - s must be finite if a solution exists, so yes!
- Is it optimal?
  - Only if costs are all 1 (more on costs later)



Quiz 2: DFS vs BFS

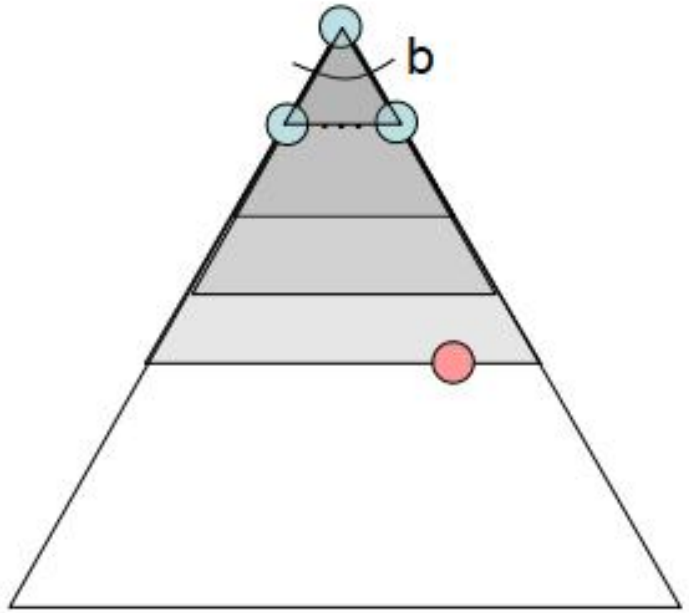
- When will BFS outperform DFS?
- When will DFS outperform BFS?





Iterative Deepening

- Depth Limited Depth-First Search
  - Limit depth to some specific level "l" and get a time and space complexity of $O(b^l)$ and O(bl) respectively.
- 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!
  - Good if the diameter of the problem is known



Iterative Deepening

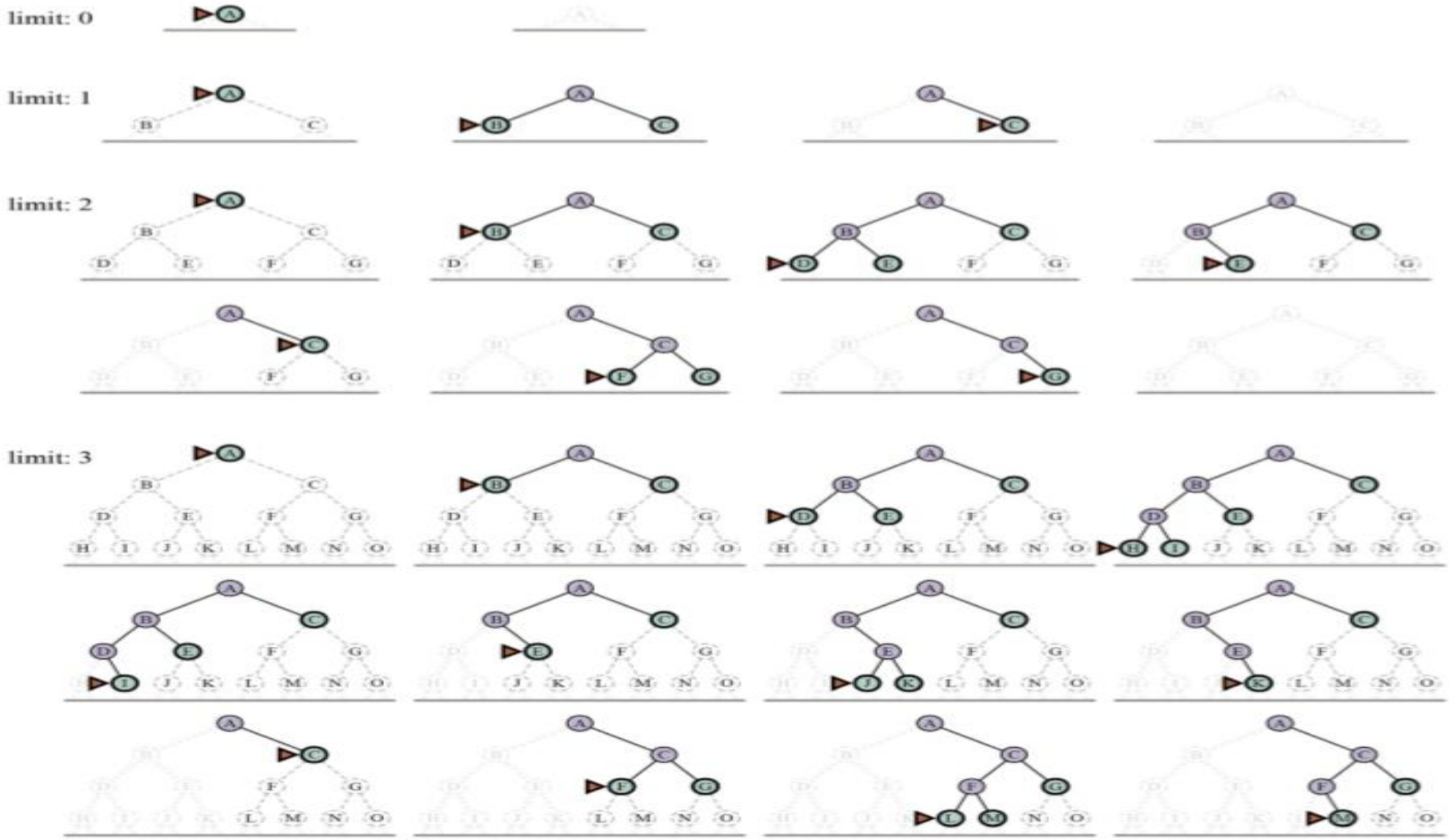
Time Complexity: $O(b^d)$ when there is a solution and $O(b^M)$ when there is none.
Space Complexity: O(bd) when there is a solution and O(bm) when there is none.

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution node or failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result

function DEPTH-LIMITED-SEARCH(problem, ℓ) returns a node or failure or cutoff
  frontier ← a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result ← failure
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node) > ℓ then
      result ← cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
  return result
```

Iterative Deepening

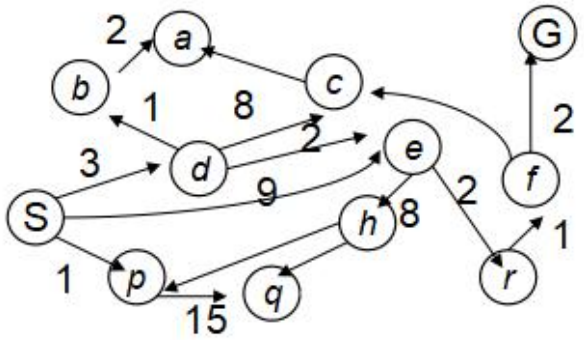
Goal to M

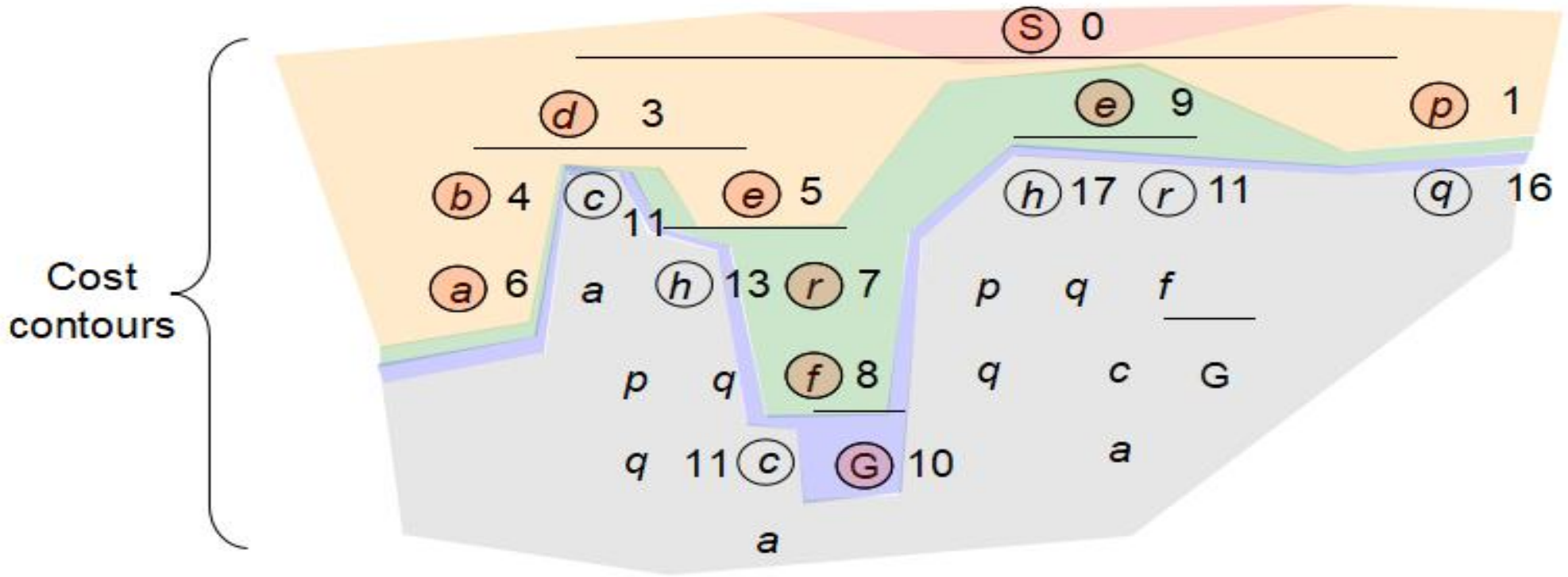


Dijkstra's algorithm or Uniform Cost Search

Strategy: expand a cheapest node first:

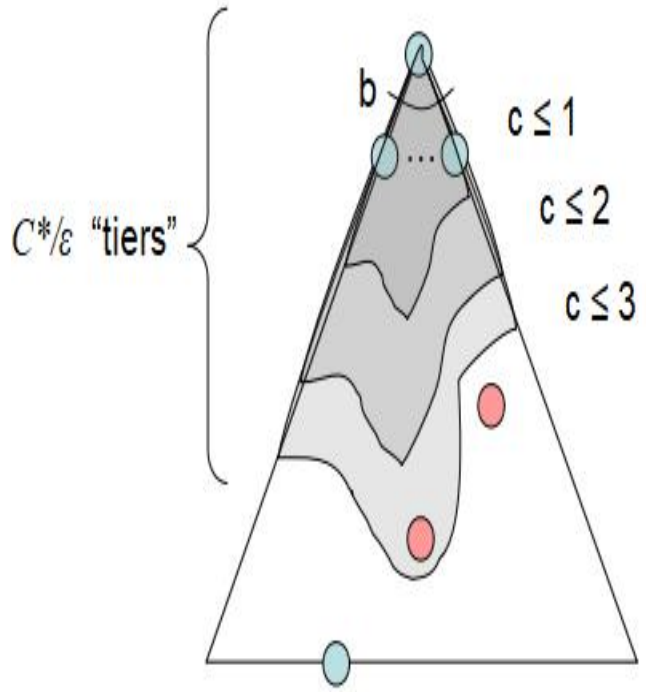
Fringe is a priority queue (priority: cumulative cost)





Dijkstra's algorithm or Uniform Cost Search

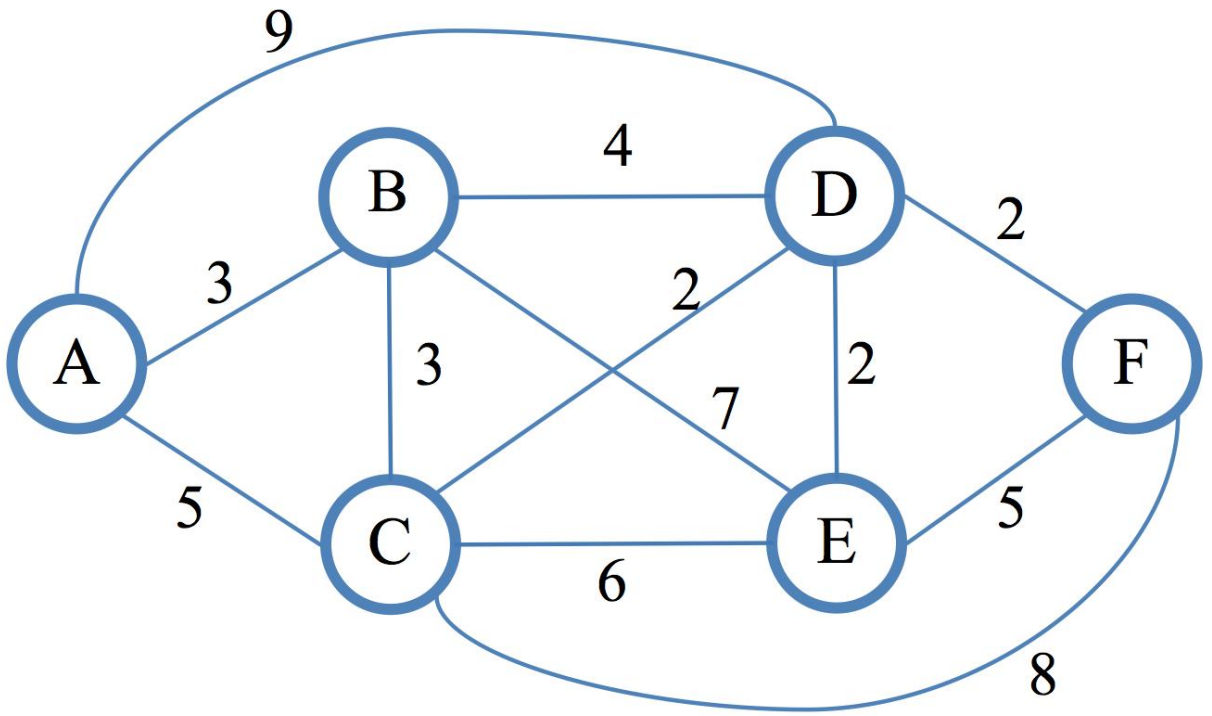
- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)



- 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, yes!
- Is it optimal?
  - Yes! (Proof next lecture via A*)

Example

Find the shortest path from state A to state F.



Next Lecture: Searching Problem

Informed Search

- Heuristics (Greedy Search and A* Search)
- Graph Search