

Introduction to Artificial Intelligence

IFT3335 Lecture 3: Uninformed Search

Bang Liu, Jian-Yun Nie

2 Certain Slides Adapted From or Referred To...

- © Slides from **UC Berkeley CS188, Dan Klein and Pieter Abbeel**
 - Uninformed Search: <https://inst.eecs.berkeley.edu/~cs188/su21/>

3 Plan

- ⊙ Agents to solve problems
 - Types of problem
 - Problem formulation
- ⊙ Sample search problems
- ⊙ Problem solving by searching
 - Breadth-First Search
 - Depth-First Search
 - Uniform-Cost Search

Agents to Solve Problems

5 Problem-Solving Agents

- ◎ **Problem-Solving Agent** is a Goal-Based Agent that decides what to do by finding sequences of actions that lead to desirable states.
- ◎ The computational process it undertakes is called **search**.
- ◎ **Problem-solving agents** use **atomic** representations: states of the world are considered as wholes
 - **Planning agents** use **factored** or **structured** representations of states
- ◎ **Informed** v.s. **Uninformed** algorithm: whether the agent can estimate how far it is from the goal

6 Example: Travel in Romania

● On vacation in Romania; currently in Arad.

● **Formulate the goal:**

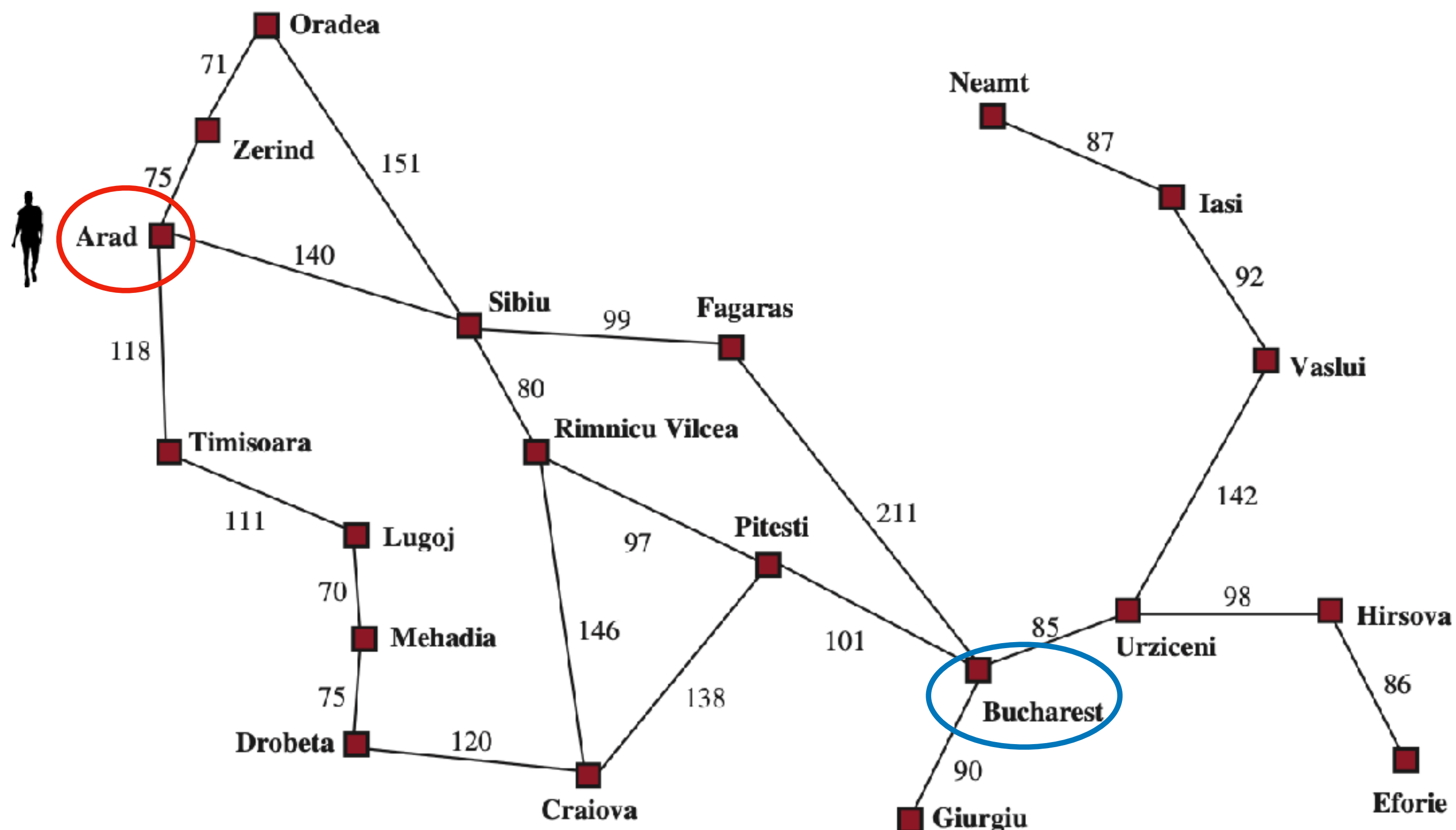
- Be in Bucharest

● **Formulate the problem:**

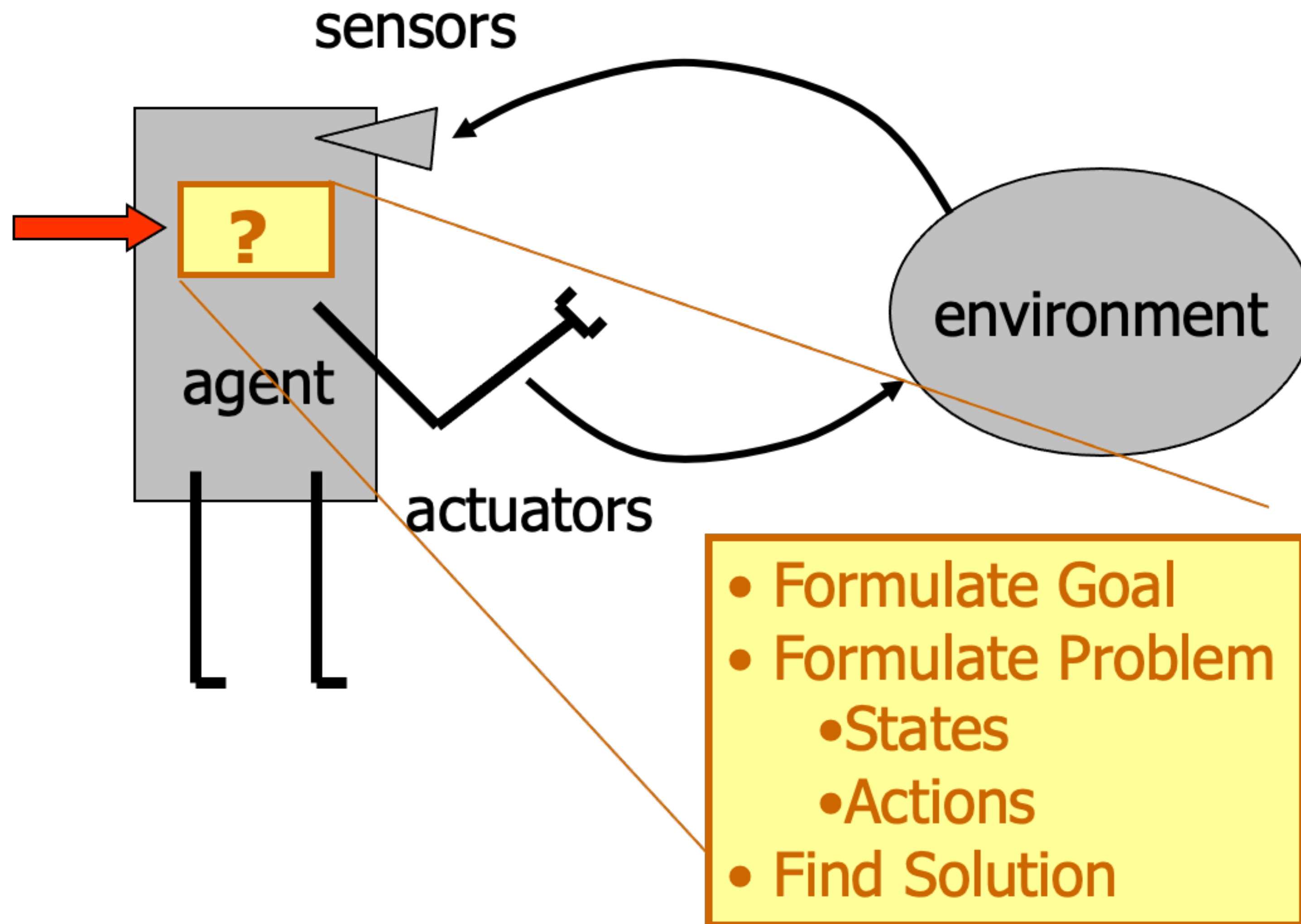
- **States:** be in different cities
- **Actions:** driving between cities

● **Find a solution:**

- Sequence of cities, e.g.,
Arad, Sibiu, Fagaras, Bucharest



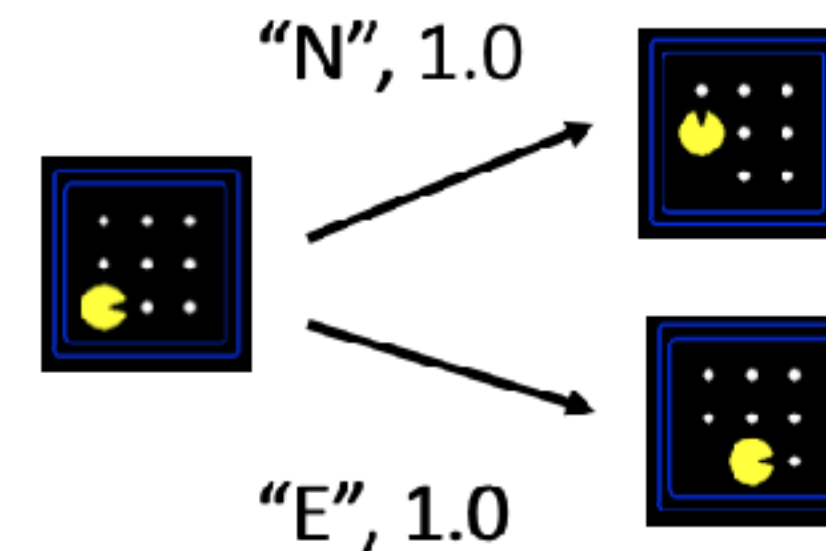
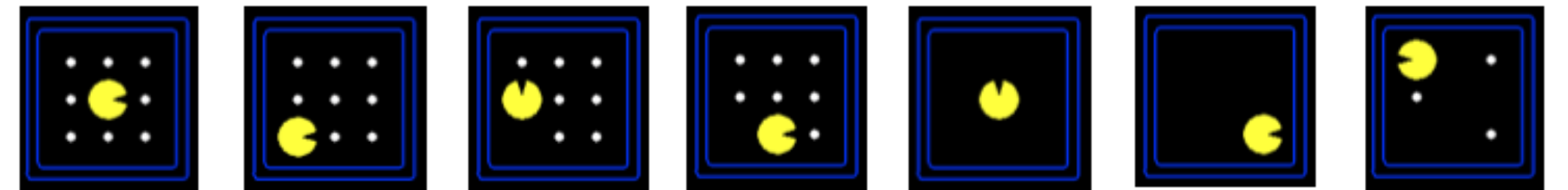
7 Problem-Solving Agents



Problem formulation

A **search problem** is defined by:

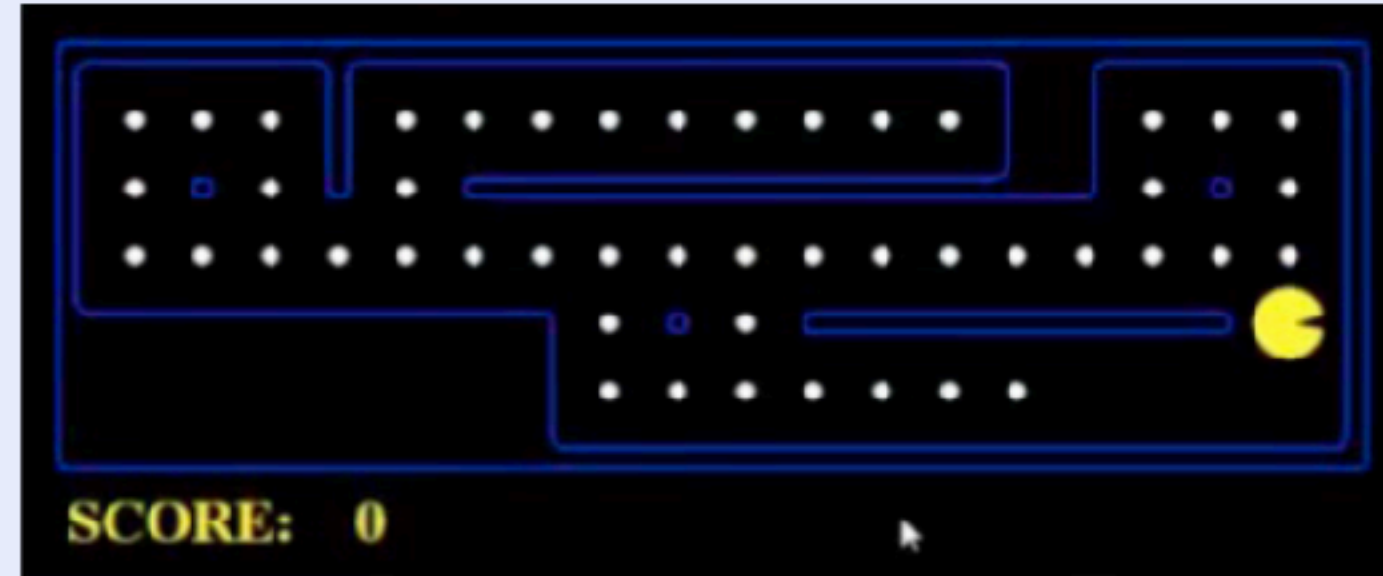
1. **State space**
A set of possible states the environment can be in.
2. **Initial state**
Where the agent starts in
3. **Goal test**
Usually a condition, sometimes the description of a state
4. **Successor function**
Include action and cost: describe how state changes with an action, and the cost of applying action a to transit from state s to s'



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

What is in a State Space?

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 $(x,y)=END$

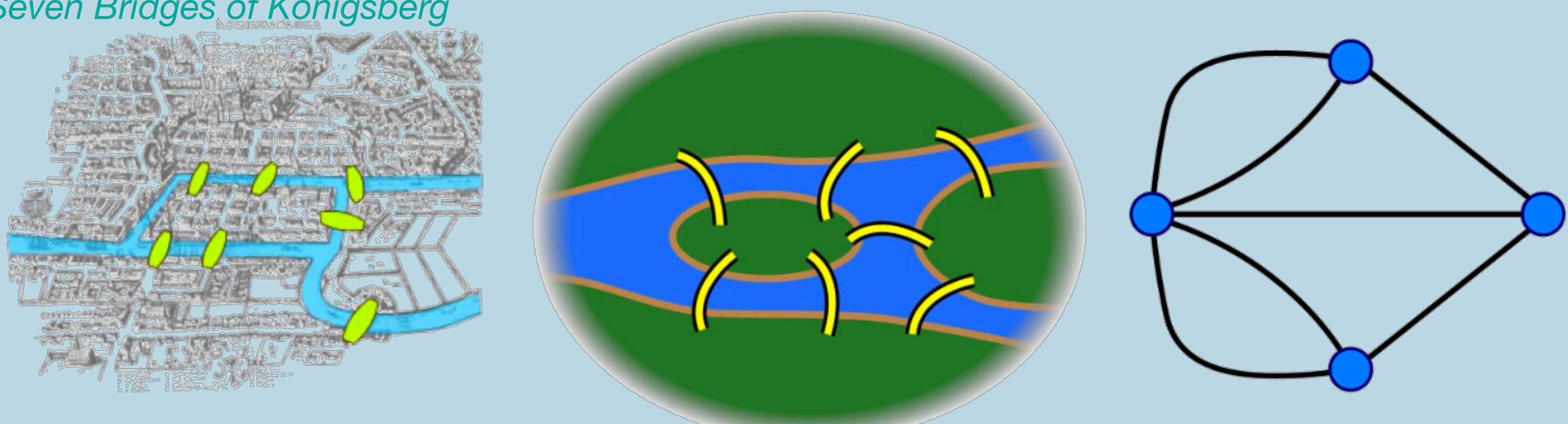
■ Problem: Eat-All-Dots

- States: $\{(x,y), \text{dot booleans}\}$
- Actions: NSEW
- Successor: update location and possibly a dot boolean
- Goal test: dots all false

10 Abstraction

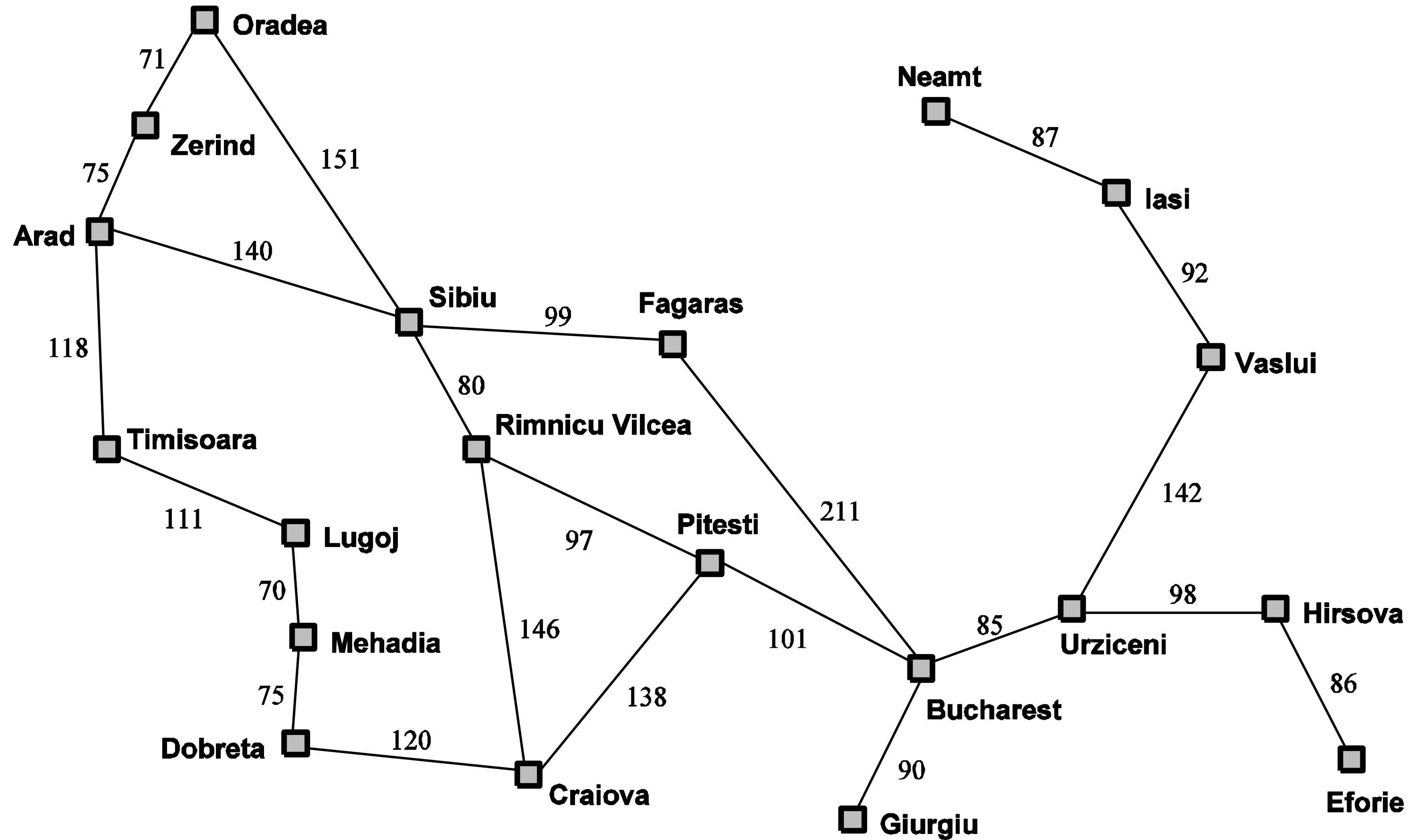
- Our formulation of the problem (e.g., getting to Bucharest) is a **model**—an abstract mathematical description—and not the real thing.
- The process of removing detail from a representation is called **abstraction**.

Seven Bridges of Königsberg



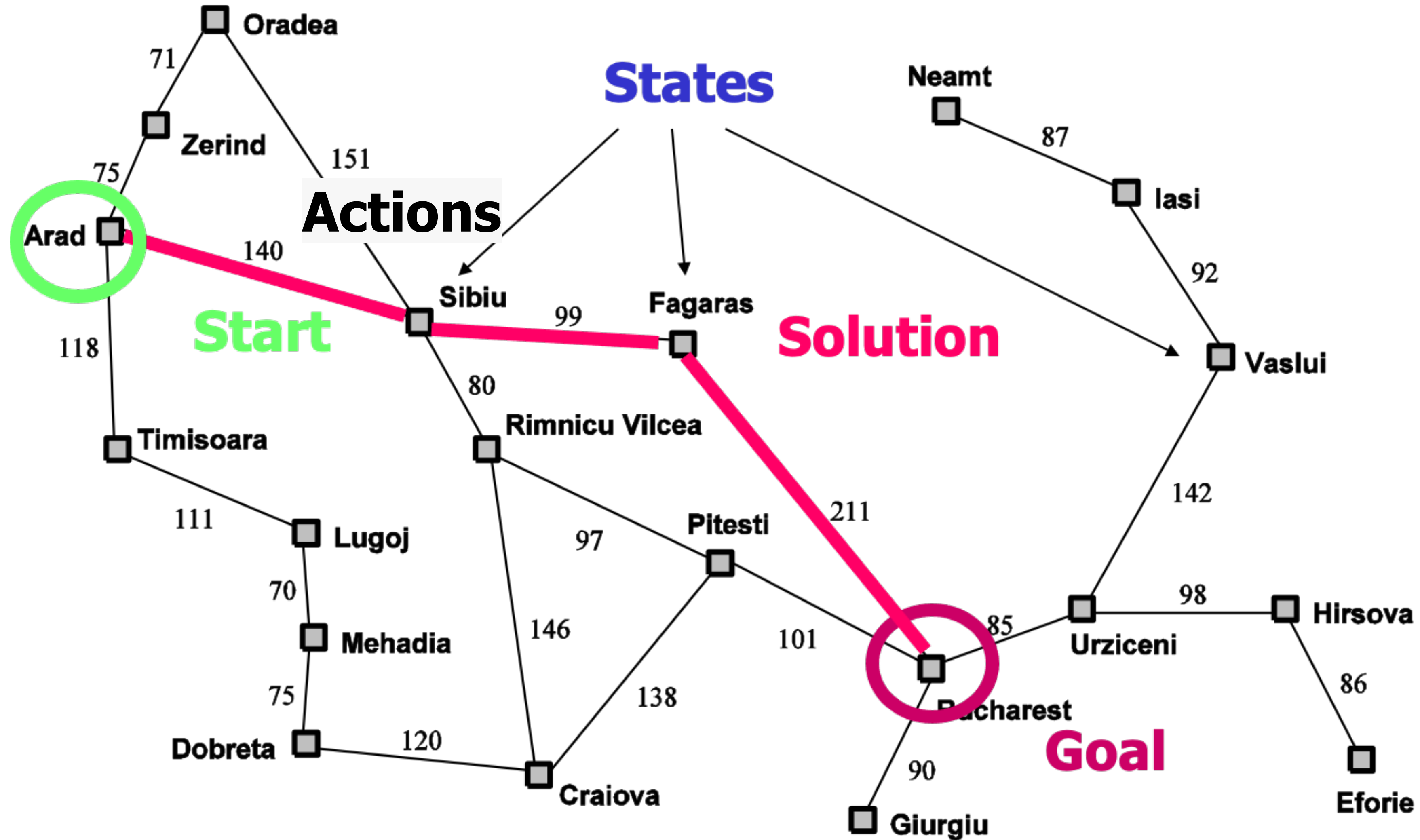
Go around the city taking each bridge once and only once?
In 1735, Euler showed that this is impossible using graph theory and topology.
Solution in graph theory: for this to be possible, each node must have an even degree.

11 Example: Route finding



Problem: find path from **Arad** to **Bucharest**

12 Example: Route finding



Example: Route finding

● **Formulate goal:**

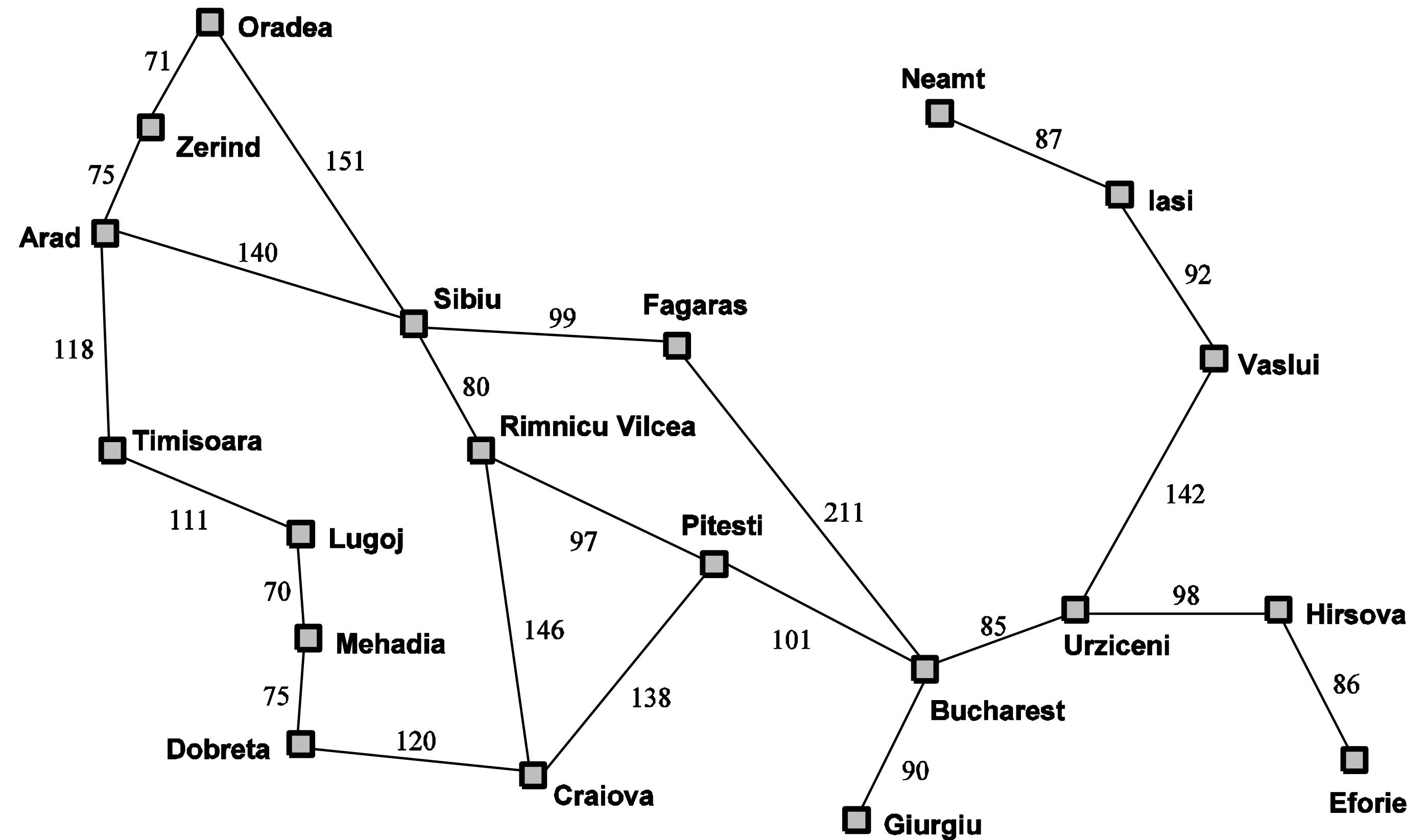
- Be in Bucharest

● **Formulate problem:**

- **States:** various cities
- **Actions:** go to adjacent city
- **Goal test:** whether in Bucharest
- **Path cost:** distance

● **Find solution:**

- Sequence of cities
- e.g., Arad, Sibiu, Fagaras, Bucharest



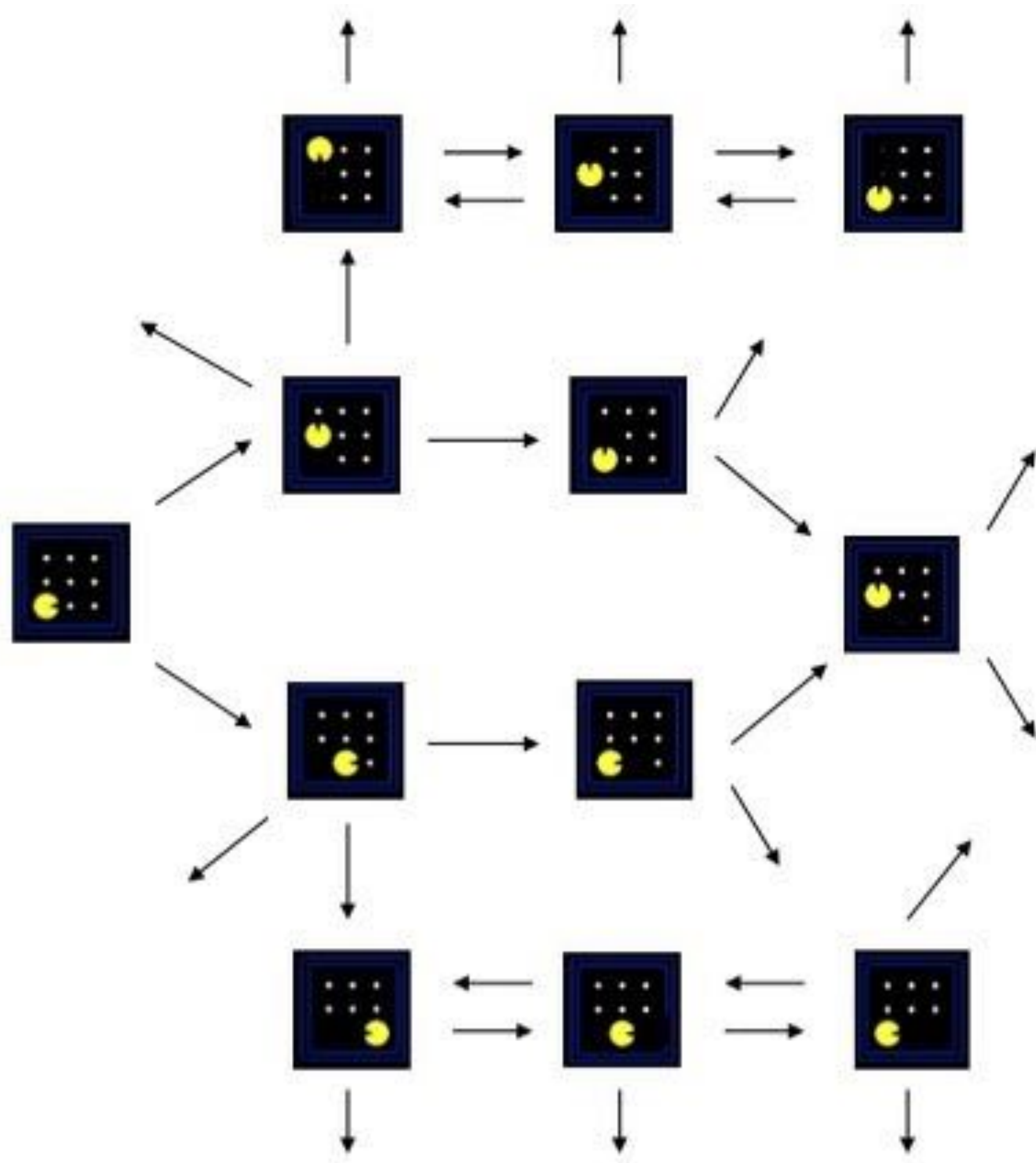
Formulate tasks as **searching**

Where is the **Map?**
State Space Graphs
and **Search Trees**

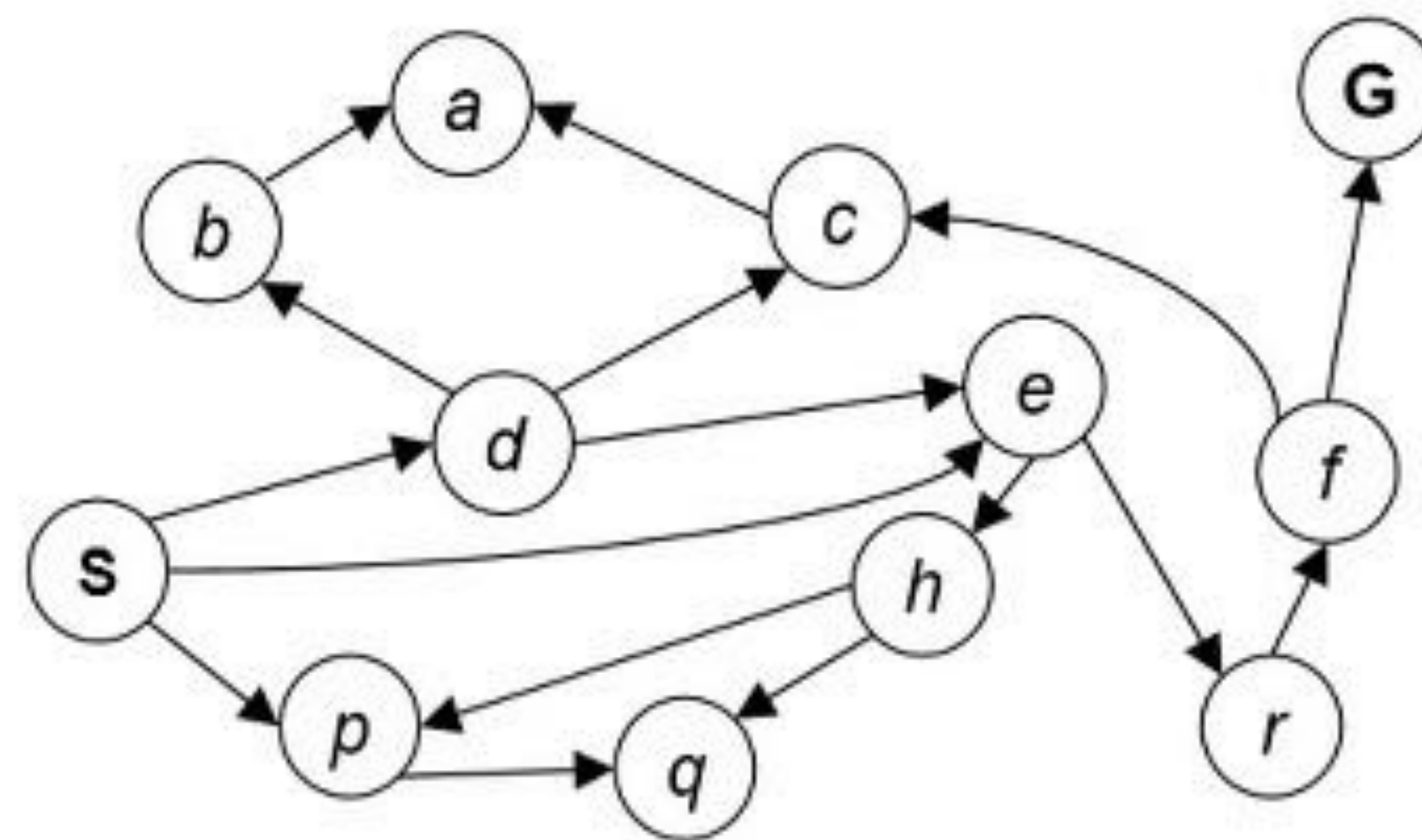
16 State space graph

- 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 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

17 State space graph

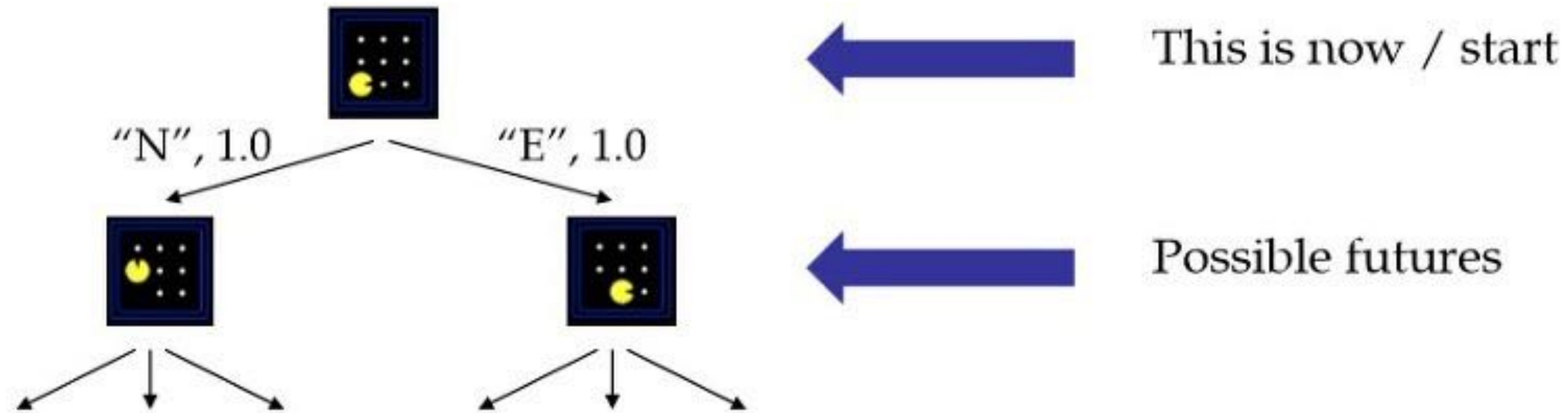


*State space graph for Pacman:
eat all the dots*



*Tiny state space graph for a tiny
search problem*

18 Search tree

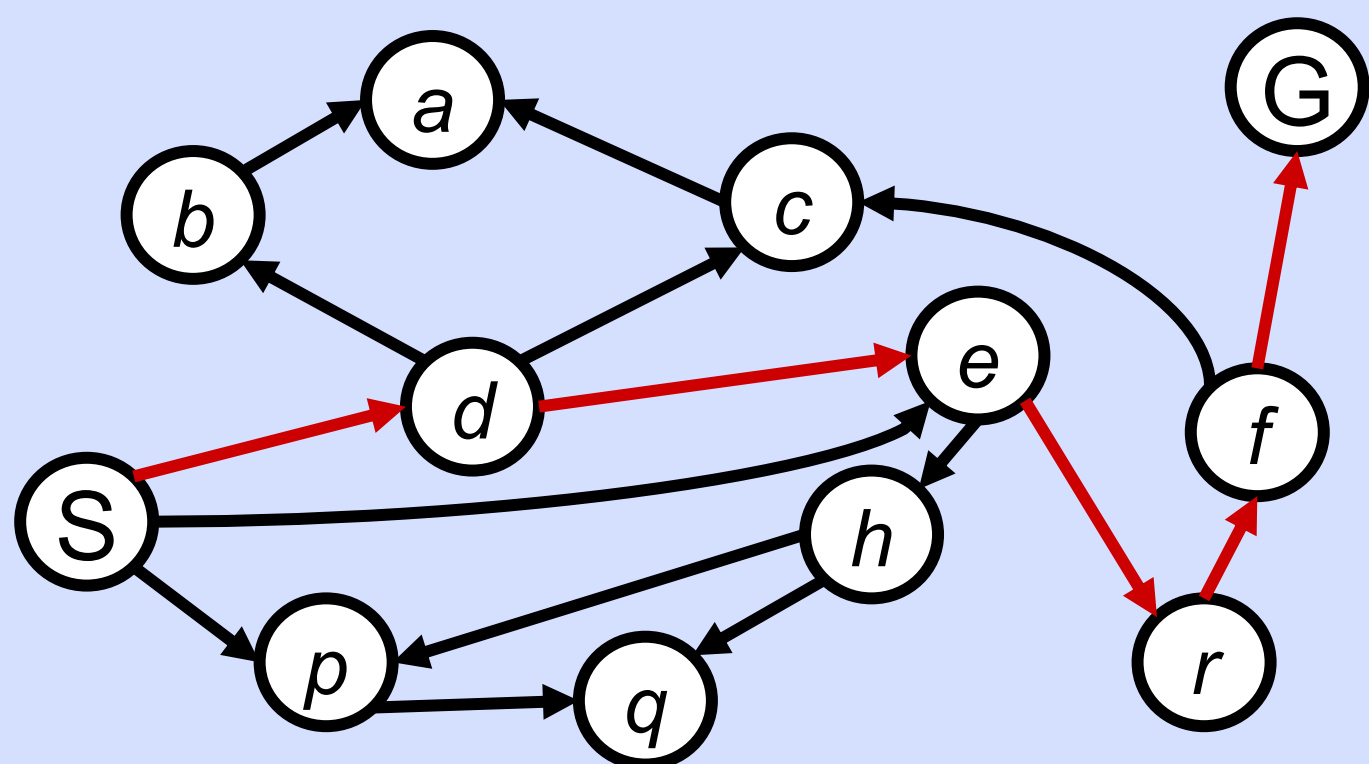


● A search tree:

- 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

19 State Space Graphs vs. Search Trees

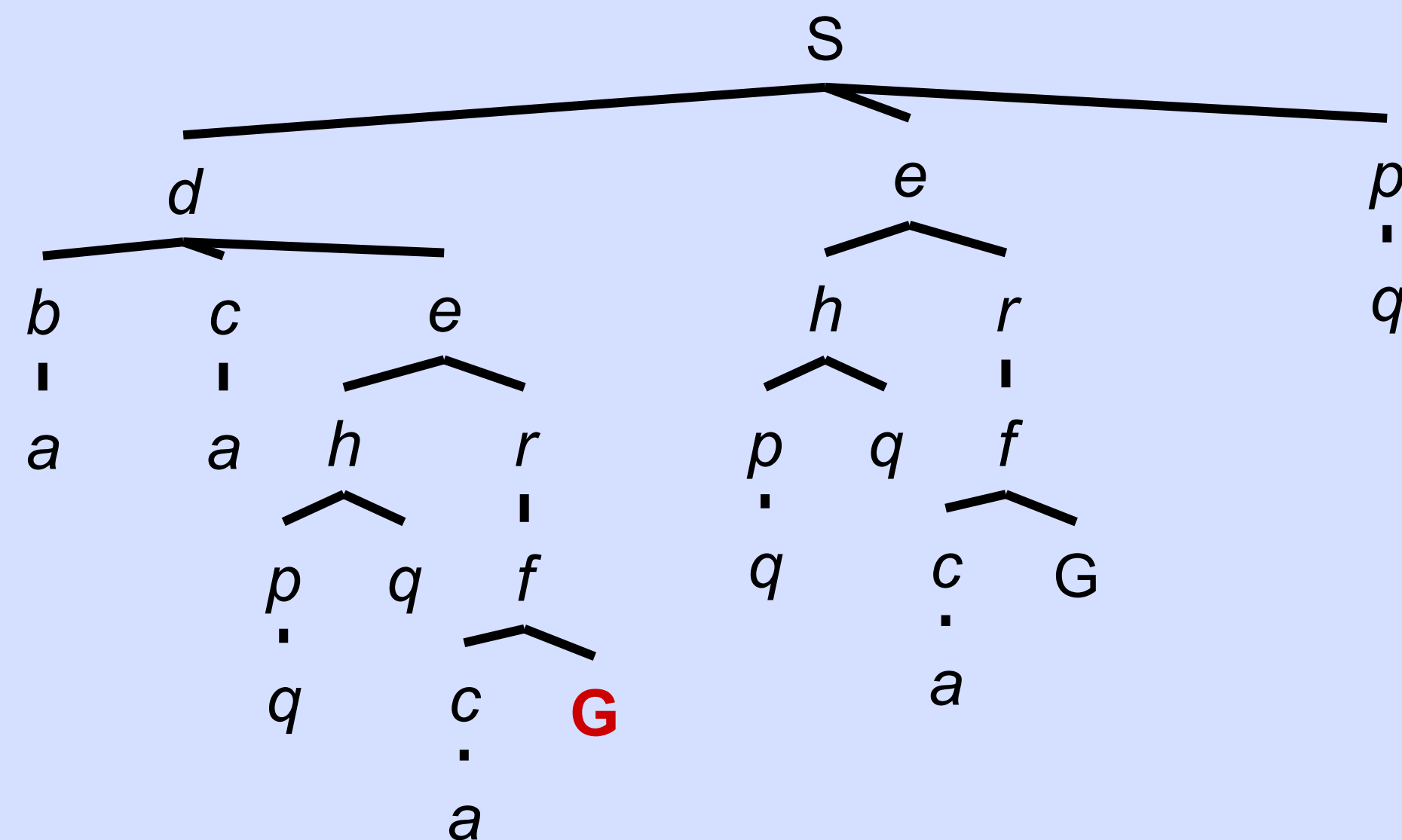
State Space Graph



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

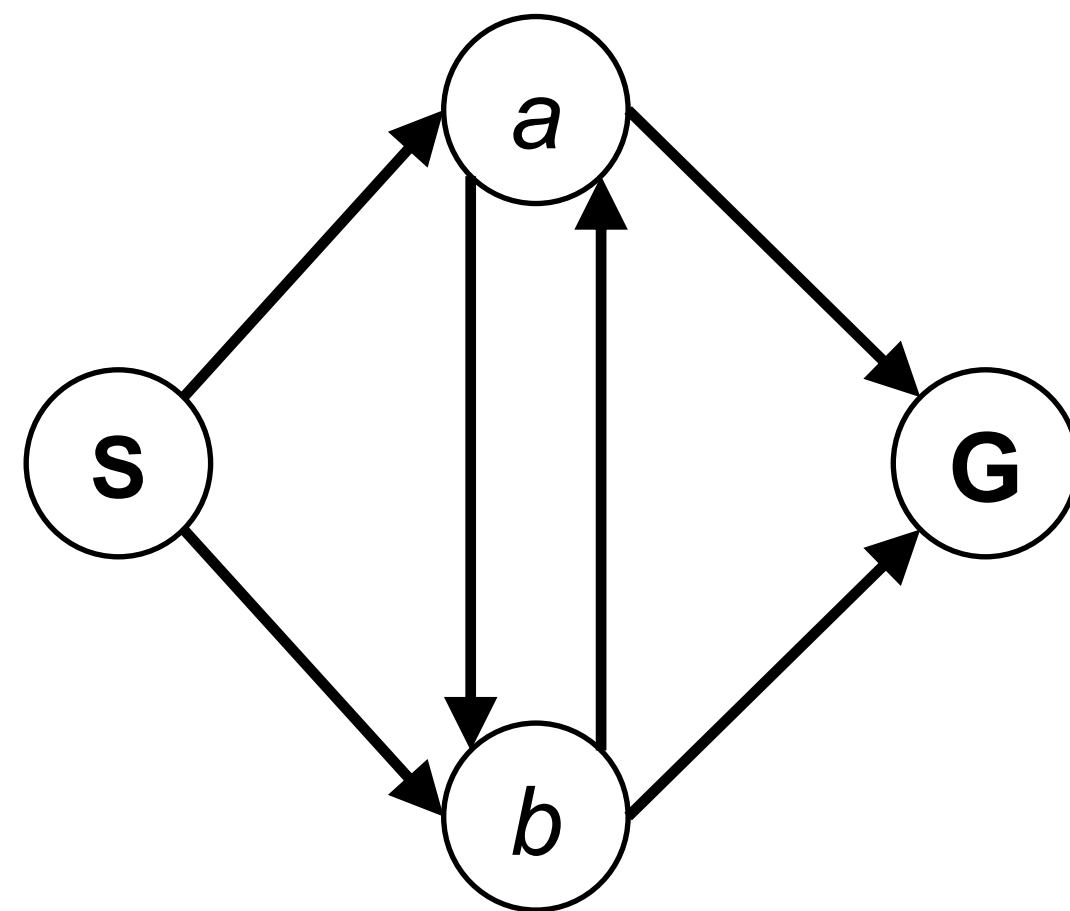
We construct both on demand – and we construct as little as possible.

Search Tree



20 State Space Graphs vs. Search Trees

Consider this 4-state graph:

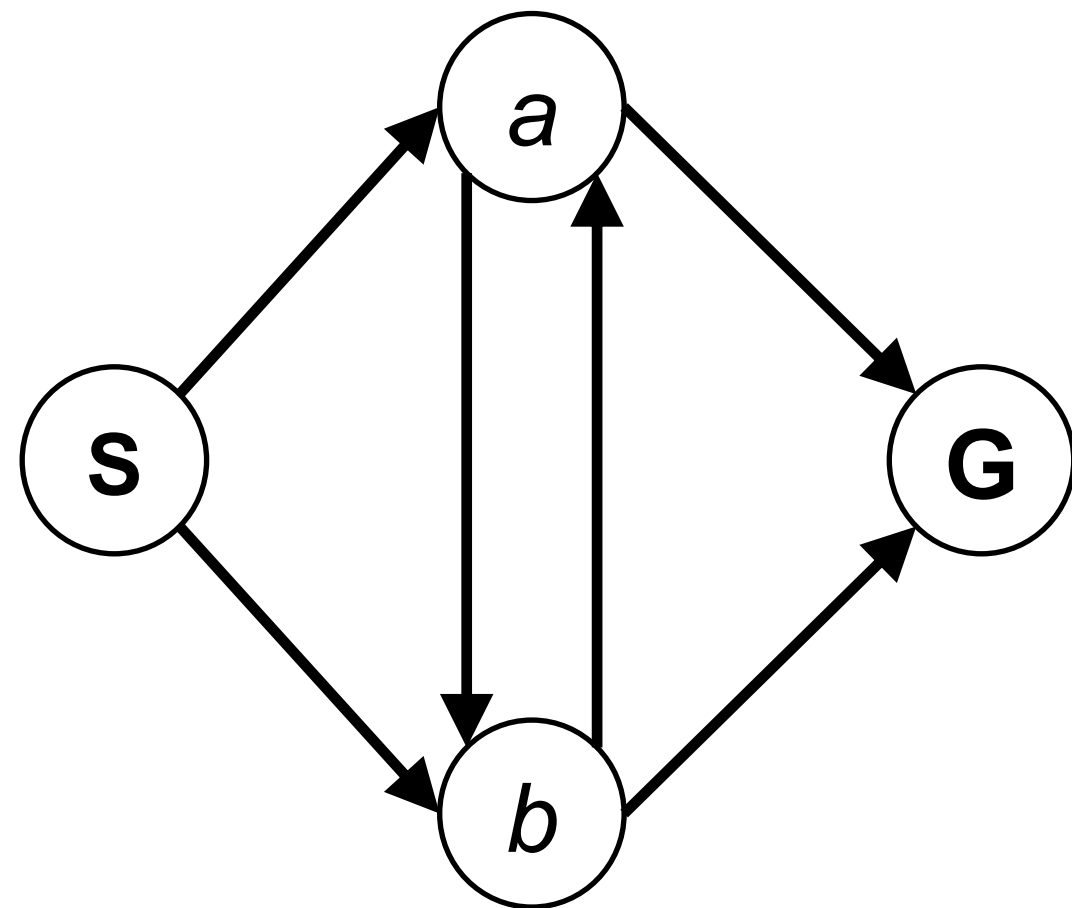


How big is its search tree (from S)?



21 State Space Graphs vs. Search Trees

Consider this 4-state graph:

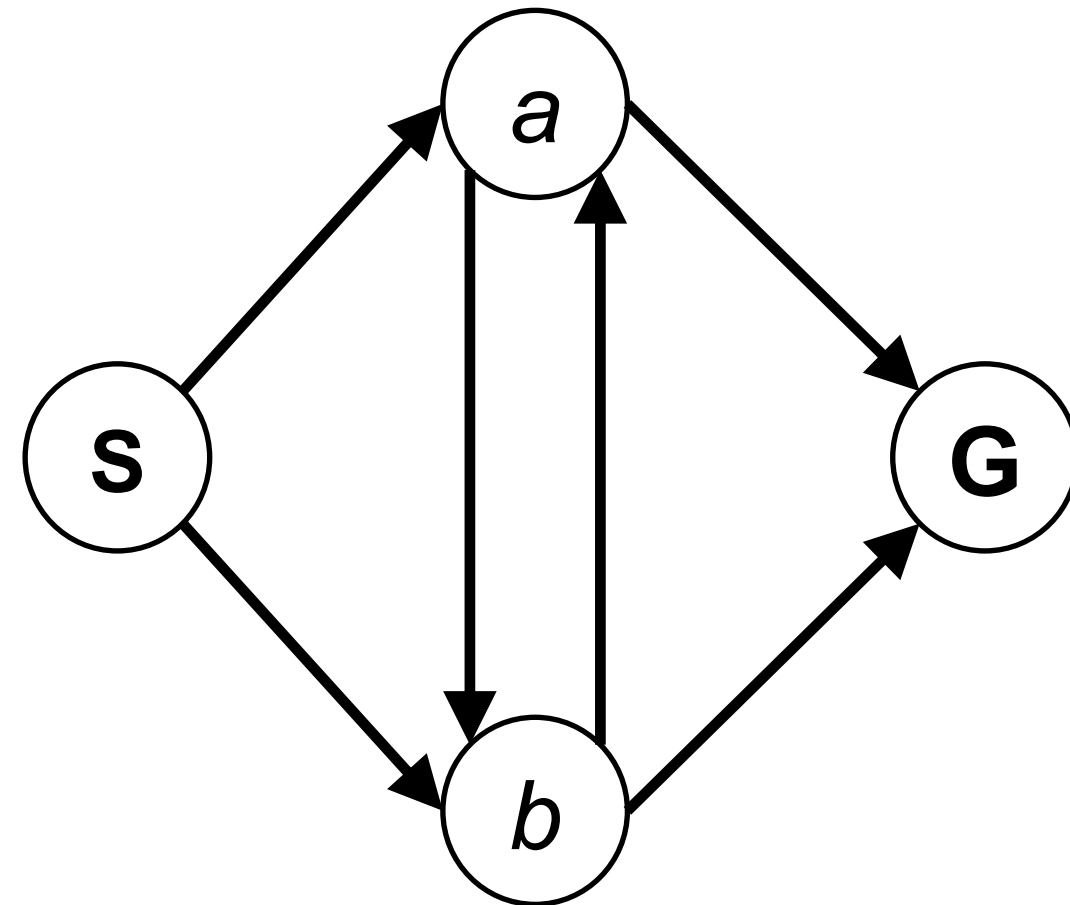


How big is its search tree (from S)?

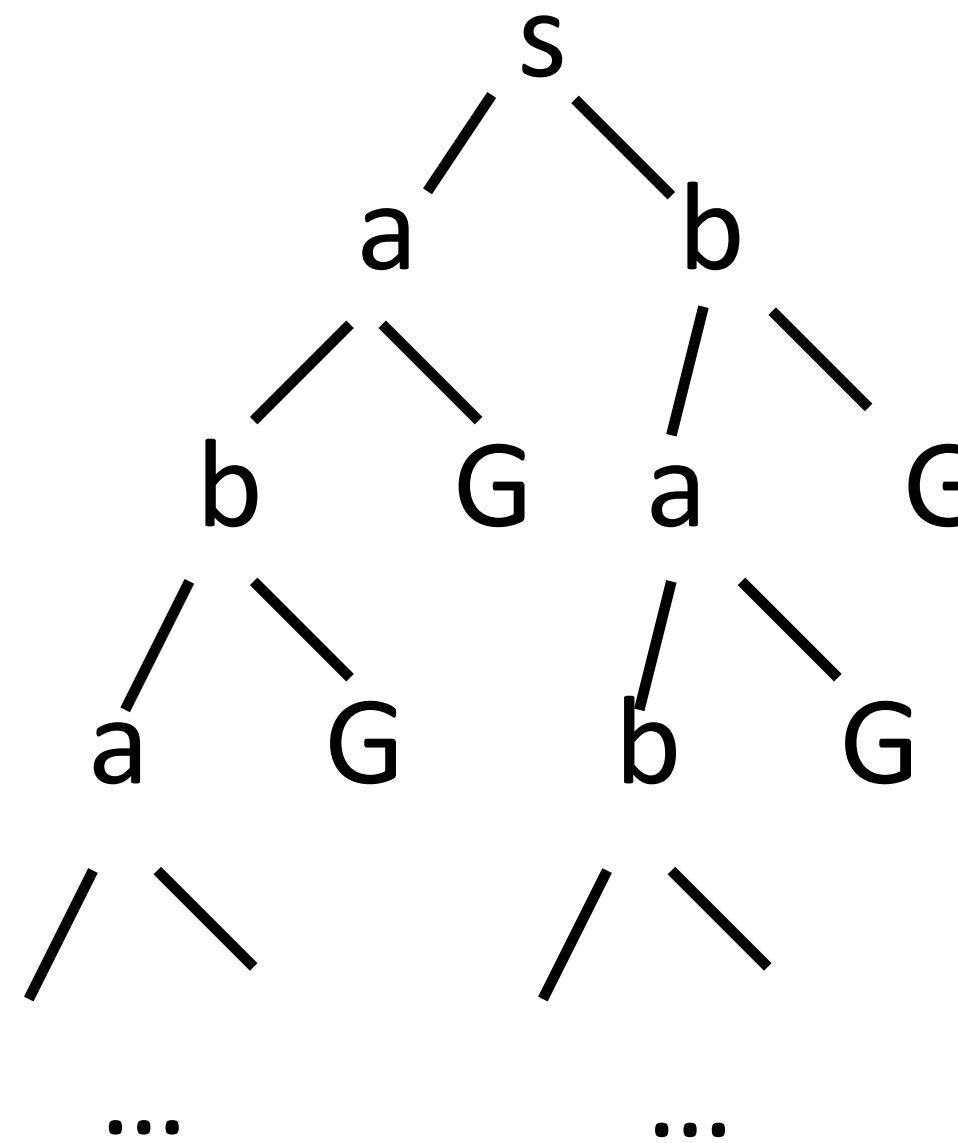


22 State Space Graphs vs. Search Trees

Consider this 4-state graph:



How big is its search tree (from S)?



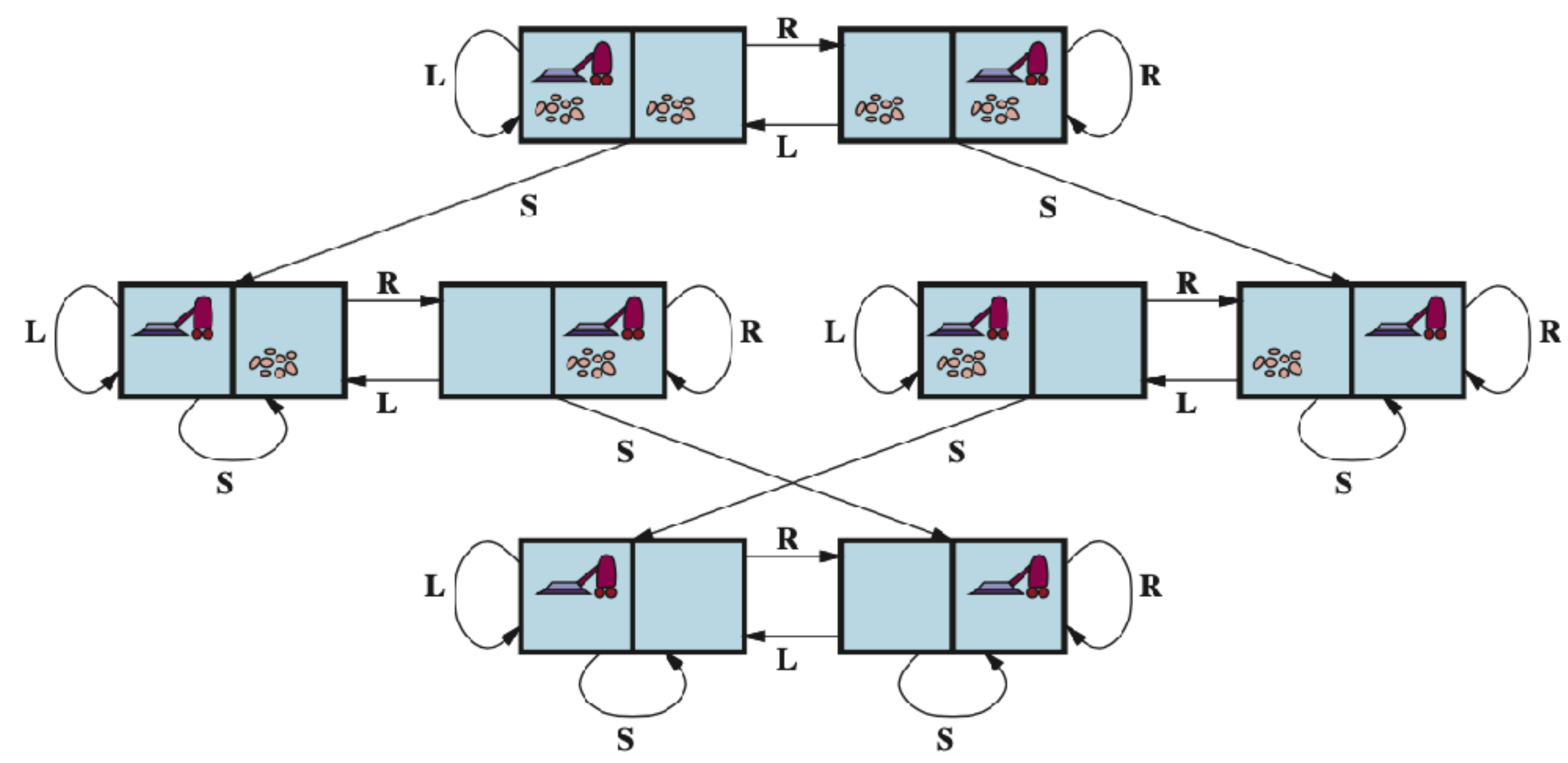
Important: Lots of repeated structure in the search tree!

Sample Search Problems

Example: Vacuum World

● **Formulate problem:**

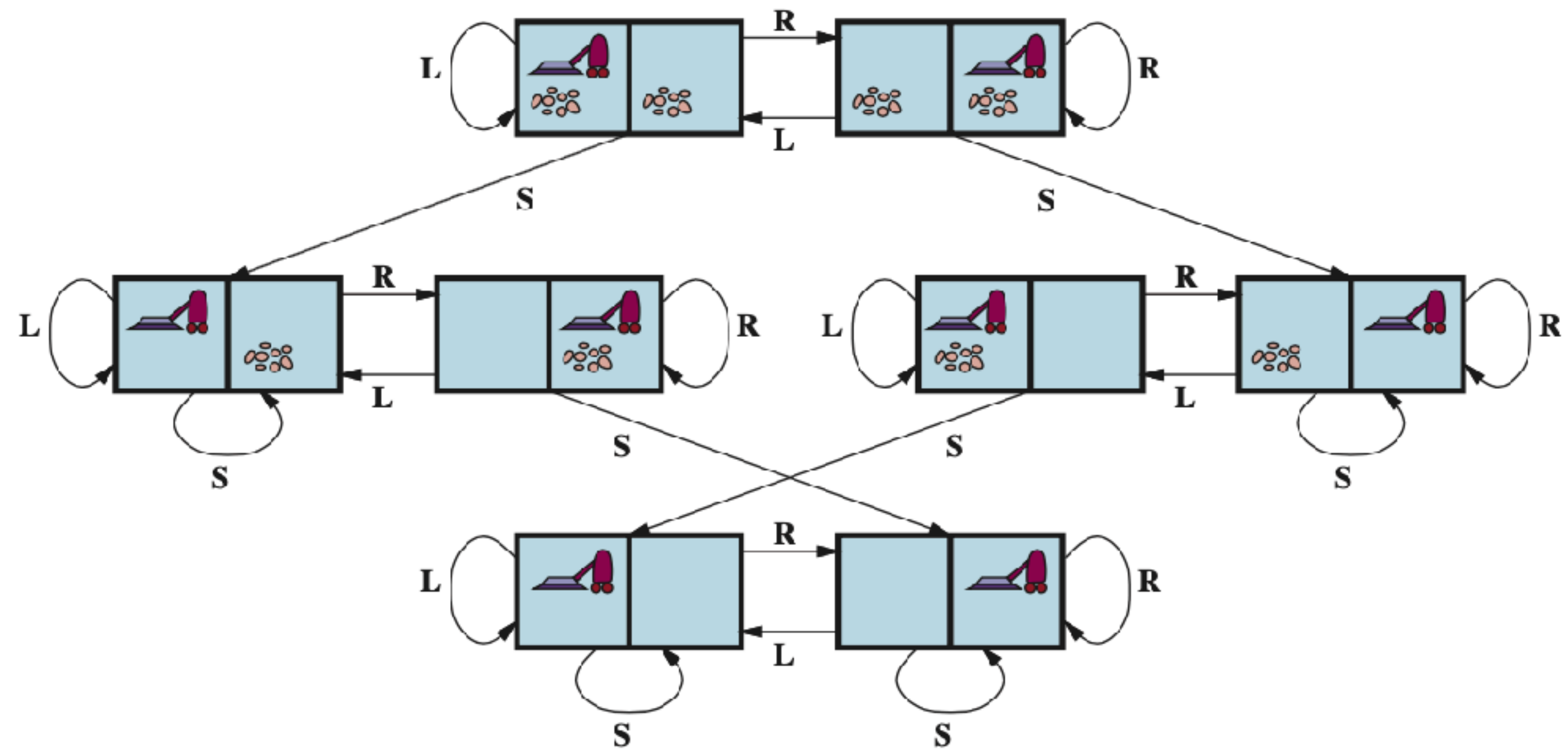
- **States:** ?
- **Actions:** ?
- **Goal test:** ?
- **Path cost:** ?



Example: Vacuum World

● **Formulate problem:**

- **States:** dirt and robot location
- **Actions:** left (L), right (R), suck (S)
- **Goal test:** no dirt at all locations
- **Path cost:** 1 per action



Example: The 8-puzzle

8	2	
3	4	7
5	1	6

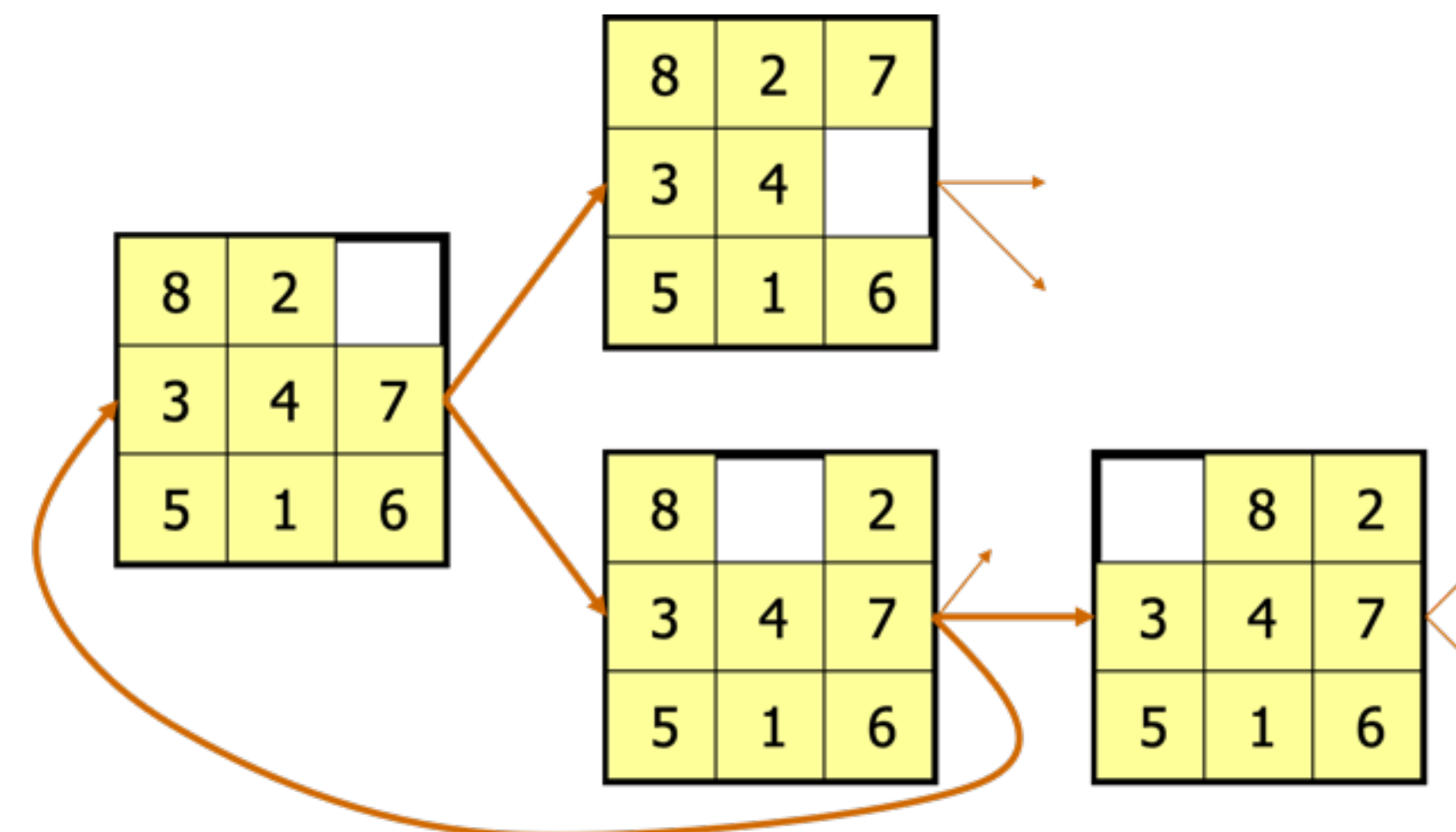
Initial state

1	2	3
4	5	6
7	8	

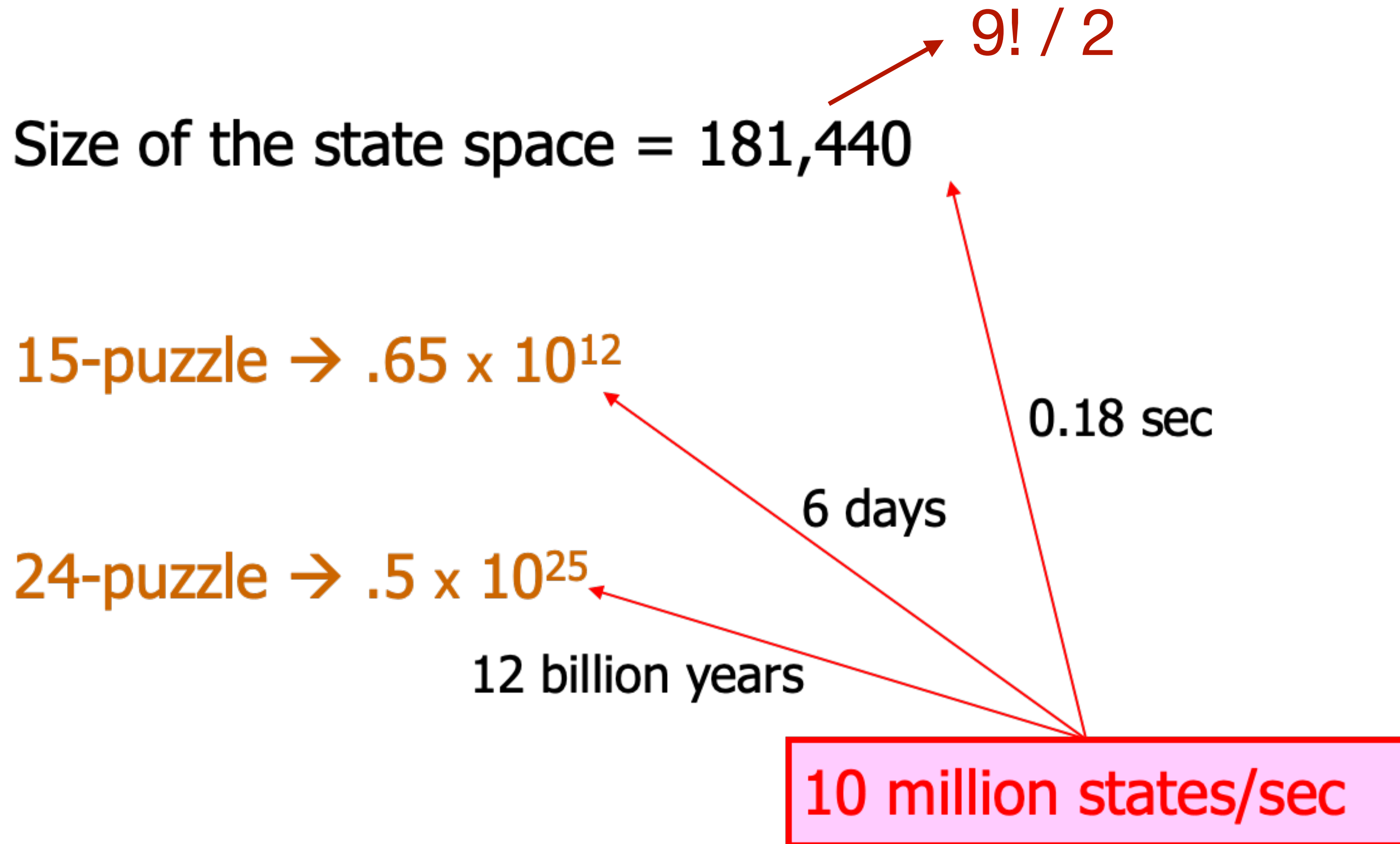
Goal state

● **Formulate problem:**

- **States:** location of tiles
- **Actions:** move blank left, right, up, down
- **Goal test:** given goal state
- **Path cost:** 1 per move



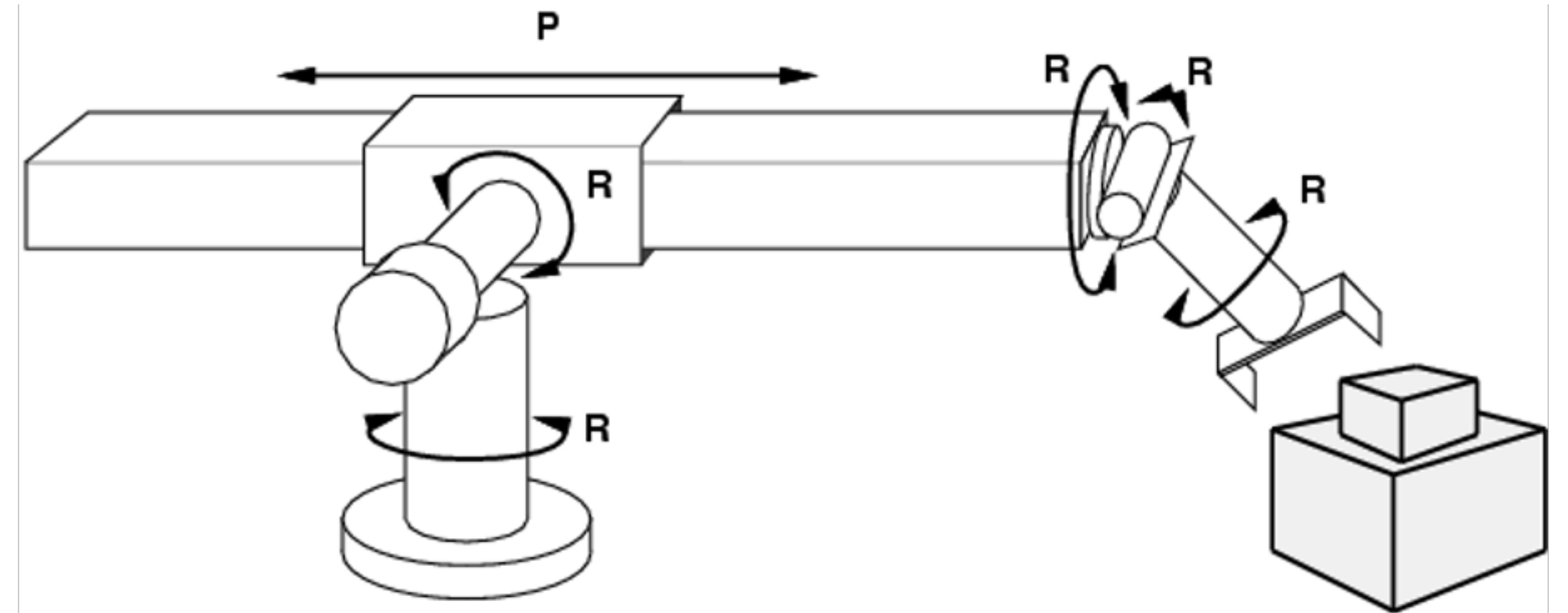
27 Example: The 8-puzzle



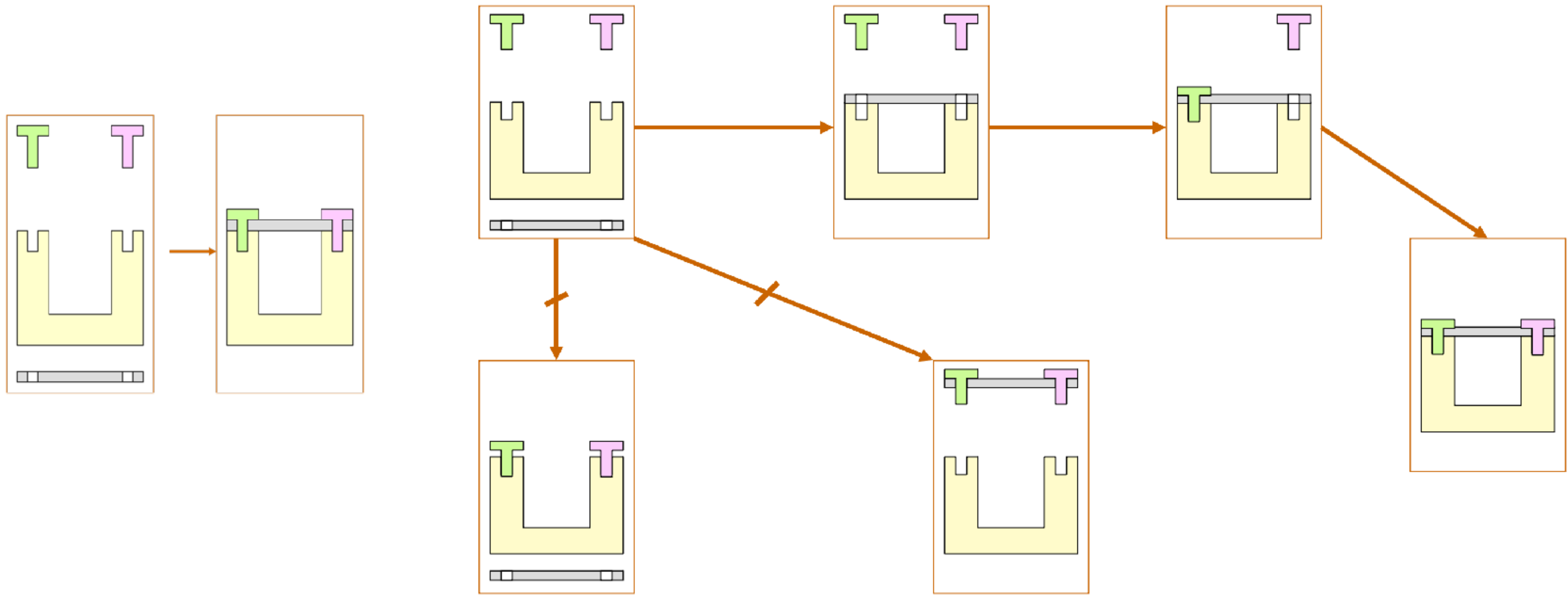
Example: Robotic Assembly

Formulate problem:

- **States:** real-valued coordinates of robot joint angles parts of the object to be assembled
- **Actions:** continuous motions of robot joints
- **Goal test:** complete assembly
- **Path cost:** time to execute



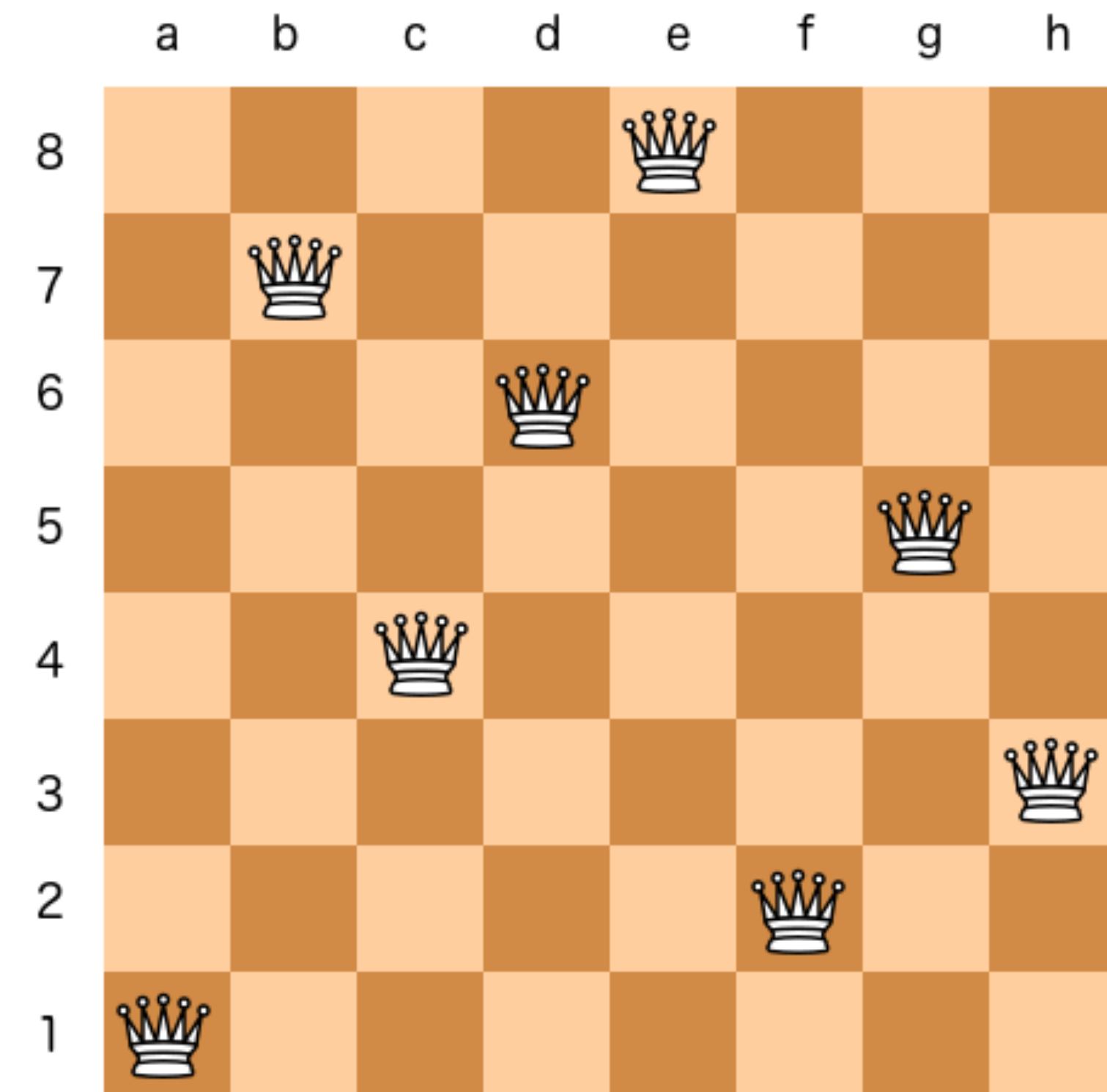
Example: Robotic Assembly



30 Example: The 8-Queens

Formulate problem:

- **States:** ?
- **Actions:** ?
- **Goal test:** ?
- **Path cost:** ?



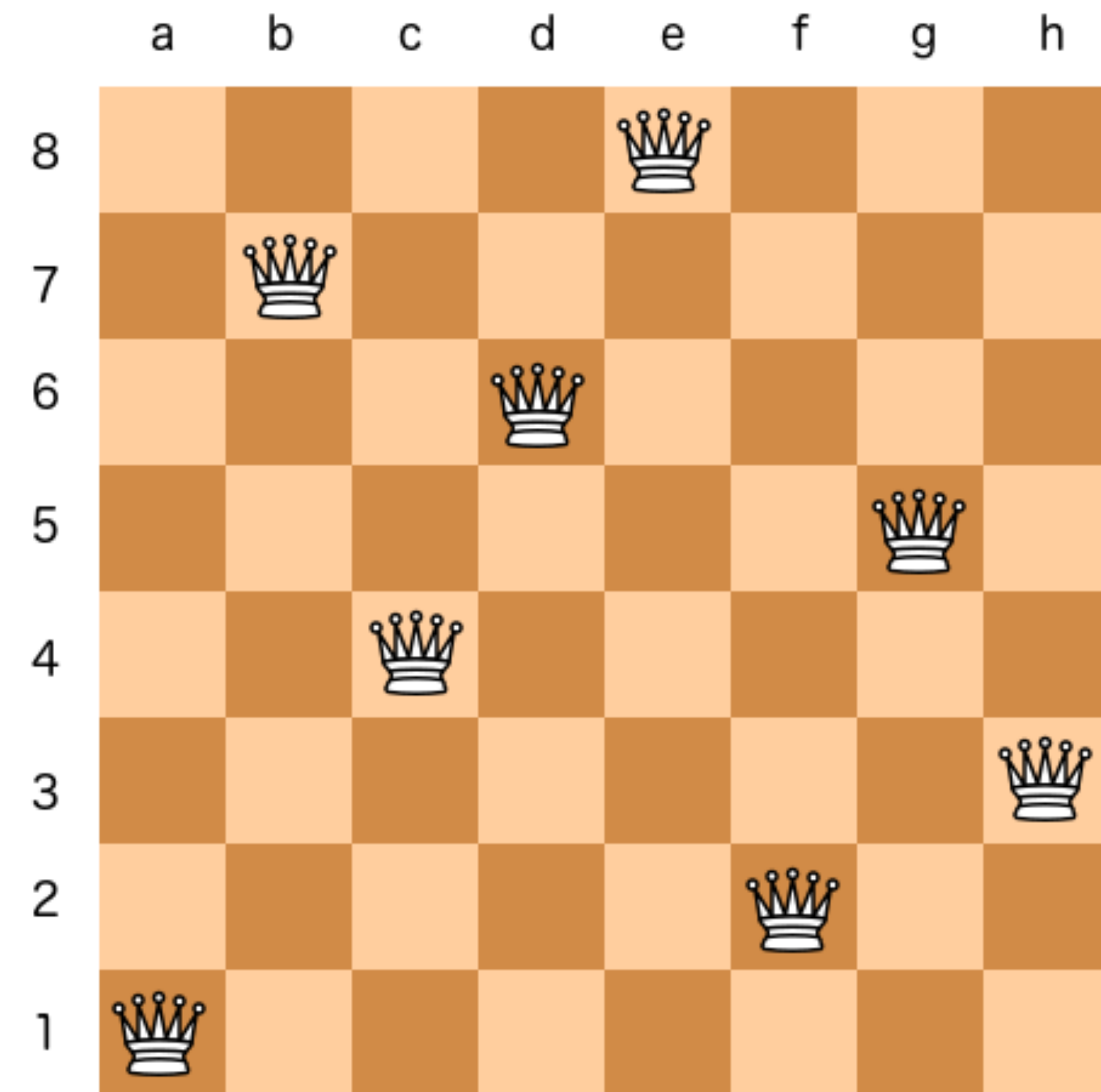
Place 8 queens in a chessboard so that no two queens are in the same row, column, or diagonal.

Example: The 8-Queens

Formulation 1:

- **States:** Any arrangement of 0 to 8 queens on the board.
- **Actions:** Add a queen in any square.
- **Goal test:** 8 queens on the board, none attacked.
- **Path cost:** 1 per move.

→ 64⁸ states with 8 queens



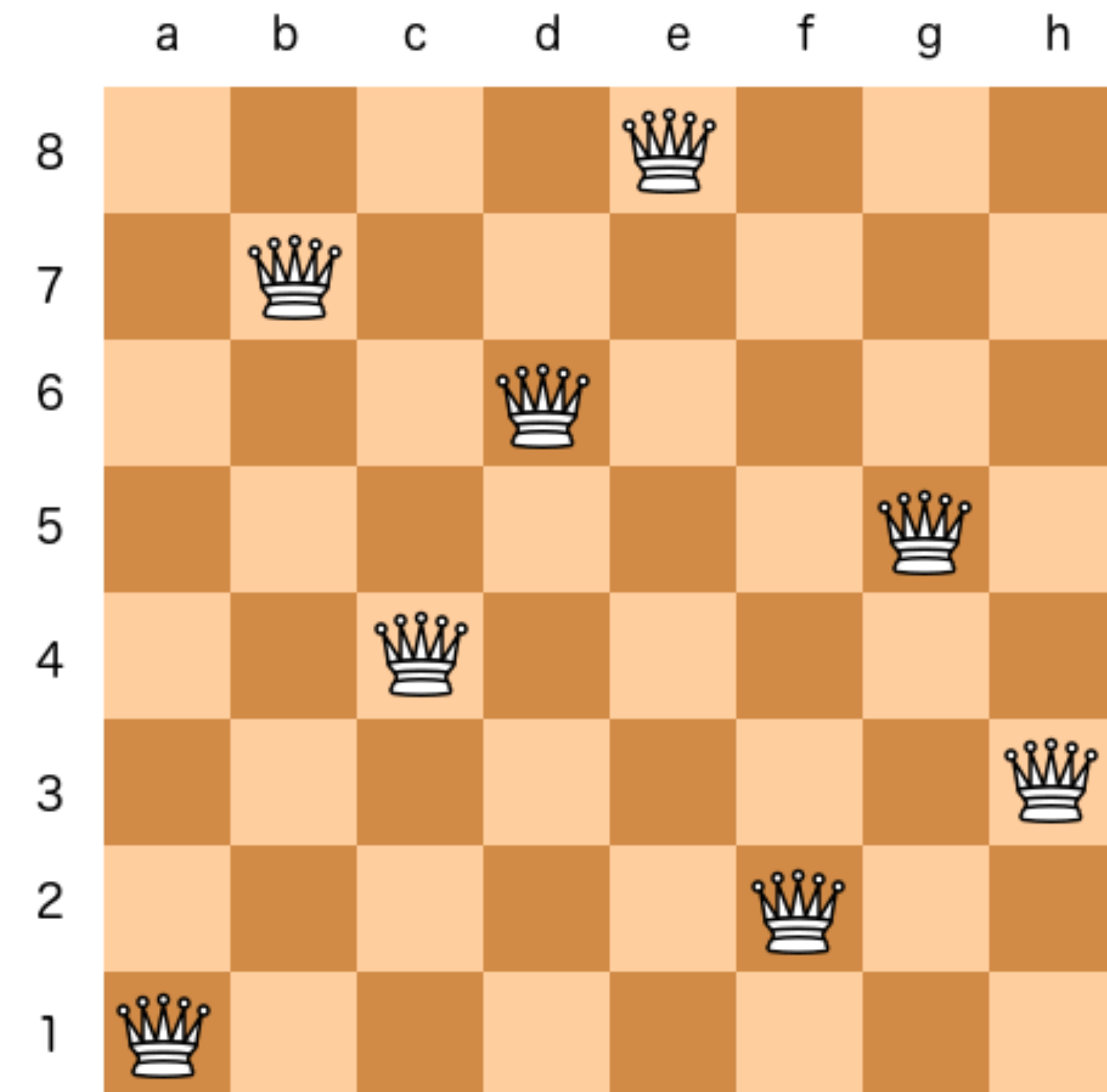
Place 8 queens in a chessboard so that no two queens are in the same row, column, or diagonal.

Example: The 8-Queens

Formulation 2:

- **States:** Any arrangement of $k = 0$ to 8 queens in the k leftmost columns with none attacked.
- **Actions:** Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen.
- **Goal test:** 8 queens on the board, none attacked
- **Path cost:** 1 per move

→ 2067 states



Place 8 queens in a chessboard so that no two queens are in the same row, column, or diagonal.

Exercise: River Crossing

A Riddle:

A Wolf, a sheep and a cabbage
need to cross the river.

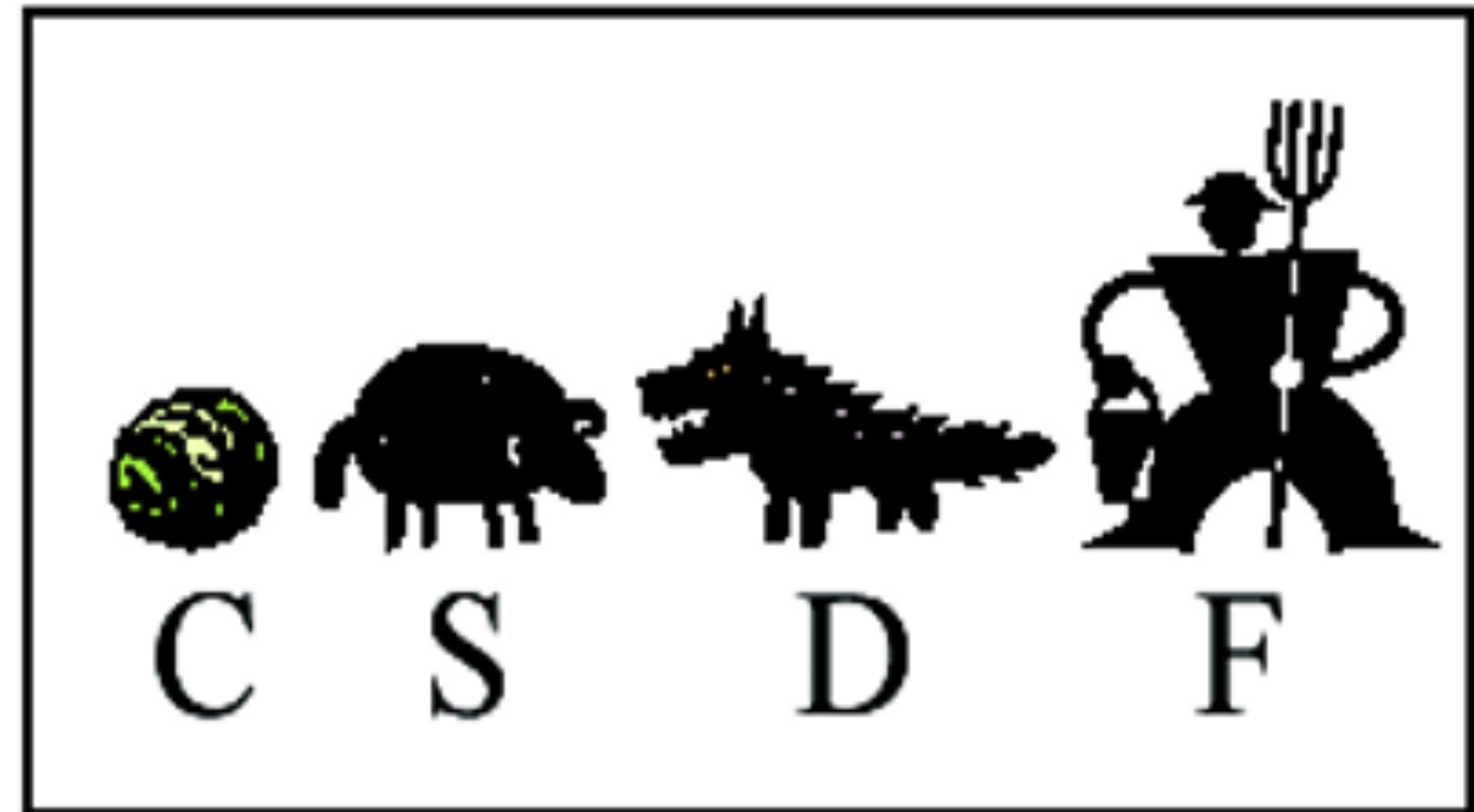
How can you bring them across, one by one,
without the sheep eating the cabbage,
nor the wolf eating the sheep?



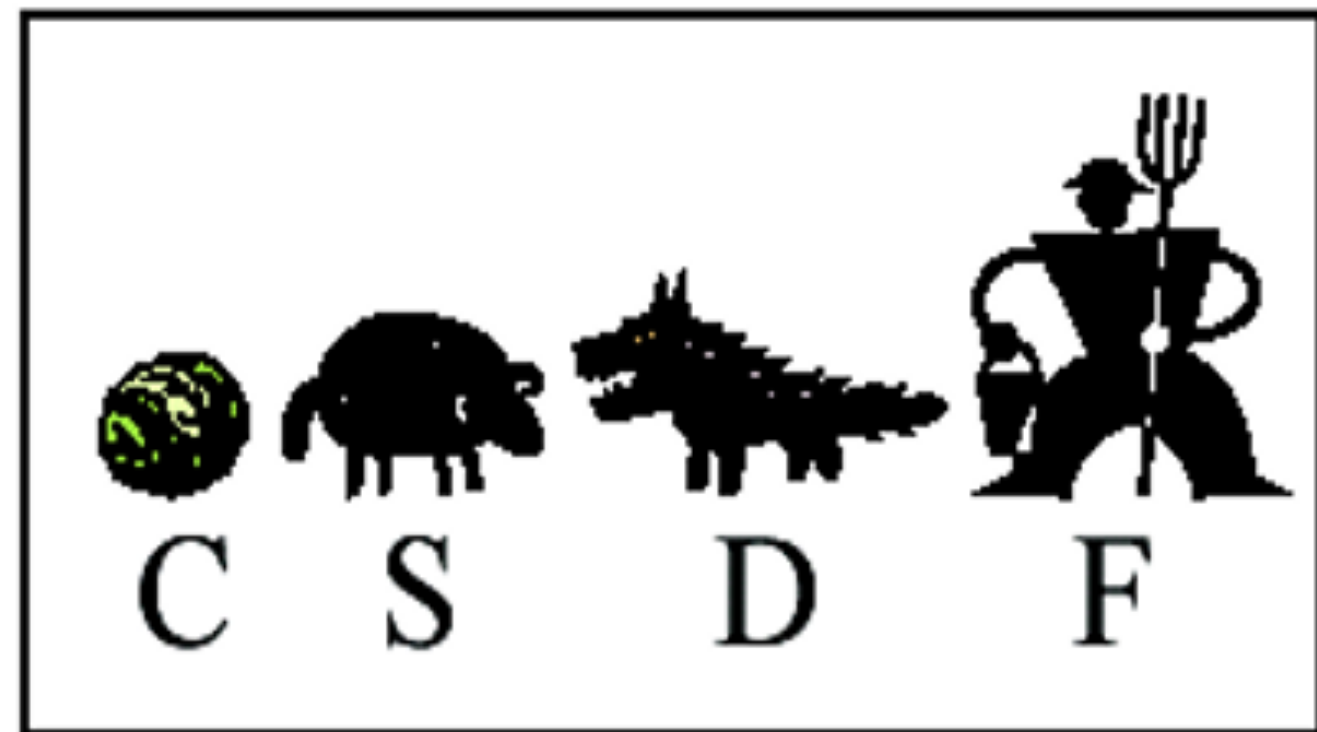
Answer: River Crossing

State space S: all valid configurations

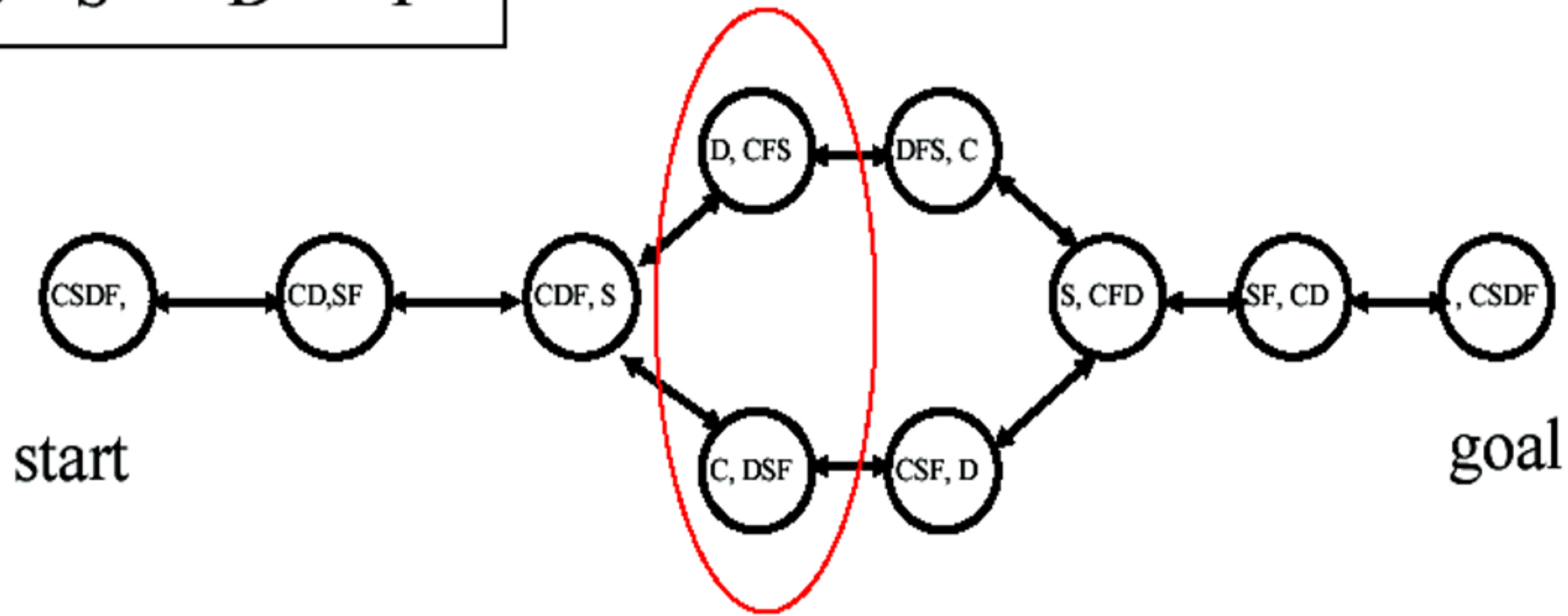
- **Initial state:** $I = \{(CSDF,)\}$
- **Goal state:** $G = \{(),CSDF\}$
- **Actions:** states reachable in one step from S
 - $\text{Succs}((CSDF,)) = \{(CD, SF)\}$
 - $\text{Succs}((CD, SF)) = \{(CD, FS), (D, CFS), (C, DFS)\}$
- **Path cost:** 1 for each transition



Answer: River Crossing



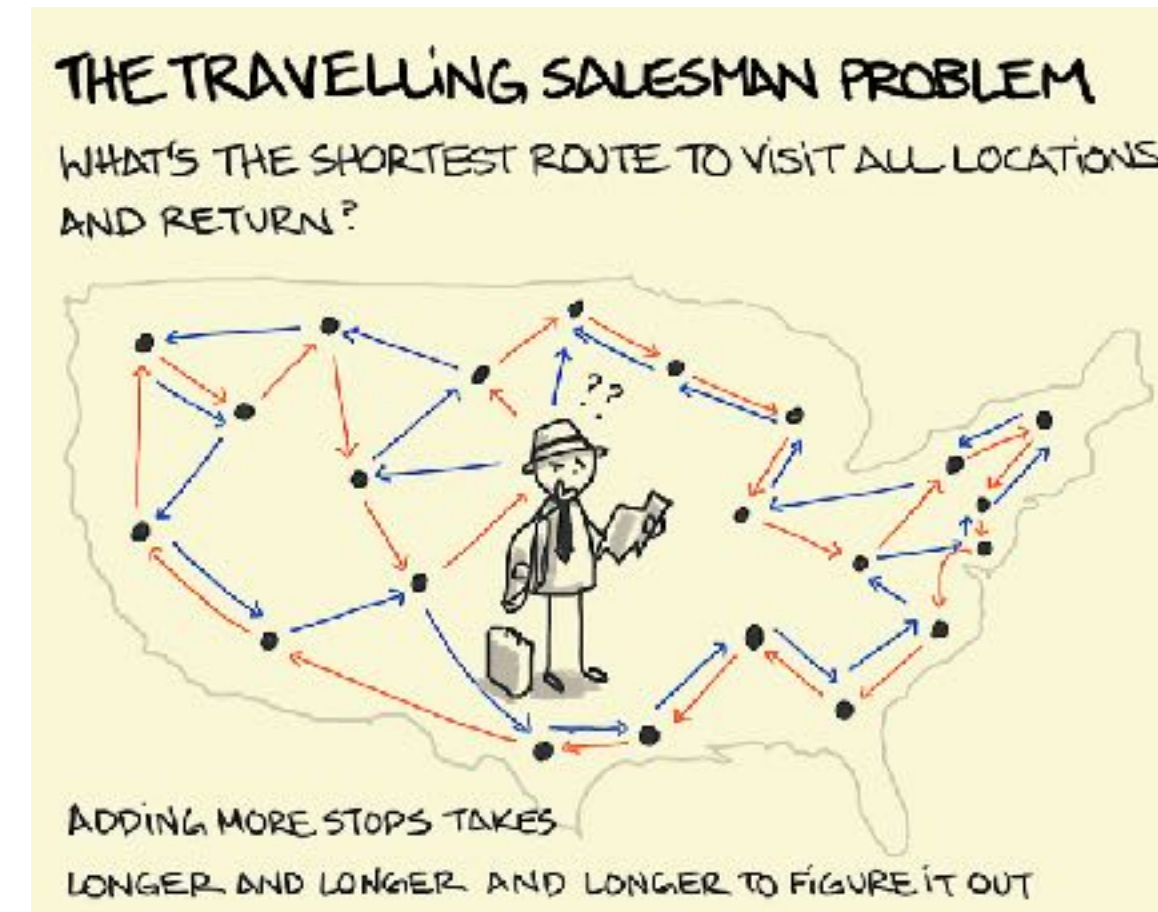
A directed graph in state space



Other example problems

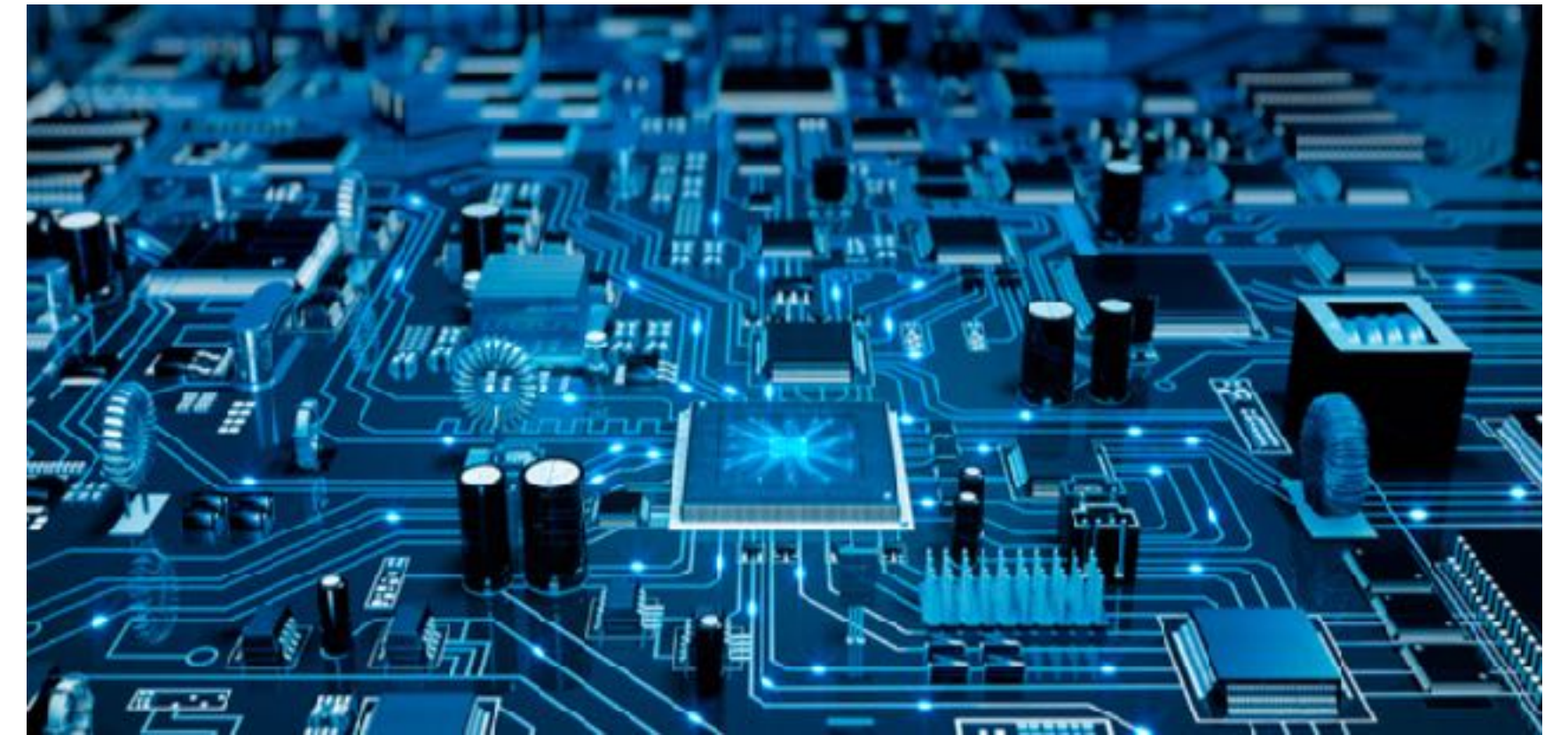
● Travelling Salesperson Problem (TSP)

- Each city must be visited exactly once – find the shortest tour



● VLSI (Very Large Scale Integration) Layout

- Given schematic diagram comprising components (chips, resistors, capacitors, etc) and interconnections (wires), find optimal way to place components on a printed circuit board, under the constraint that only a small number of wire layers are available (and wires on a given layer cannot cross !)



● Robot navigation

- A generalization fo the route-finding problem.



Problem Solving by Searching

38 Searching as problem solving technique

Problem solving by searching:

- **Searching** is the process of looking for the solution of a problem through a set of possibilities (state space).
- **Search** conditions include :
 - **Current state** - where one is;
 - **Goal state** - the solution reached; check whether it has been reached;
 - **Cost** of obtaining the solution.
- The **Solution** is a path from the current state to the goal state.

39 Searching as problem solving technique

Process of Searching:

- ◎ **Searching** proceeds as follows:
 - Check the **current state**;
 - Execute **allowable actions** to move to the next state;
 - Check if the **new state** is the **solution state**; if it is not, then the new state becomes the current state and the process is repeated until a solution is found or the state space is exhausted.

40 Explore the search tree to find solution(s)

Assumptions in basic search:

- The environment is **Static**
- The environment is **Discretizable**
- The environment is **Observable**
- The actions are **Deterministic**

General tree search

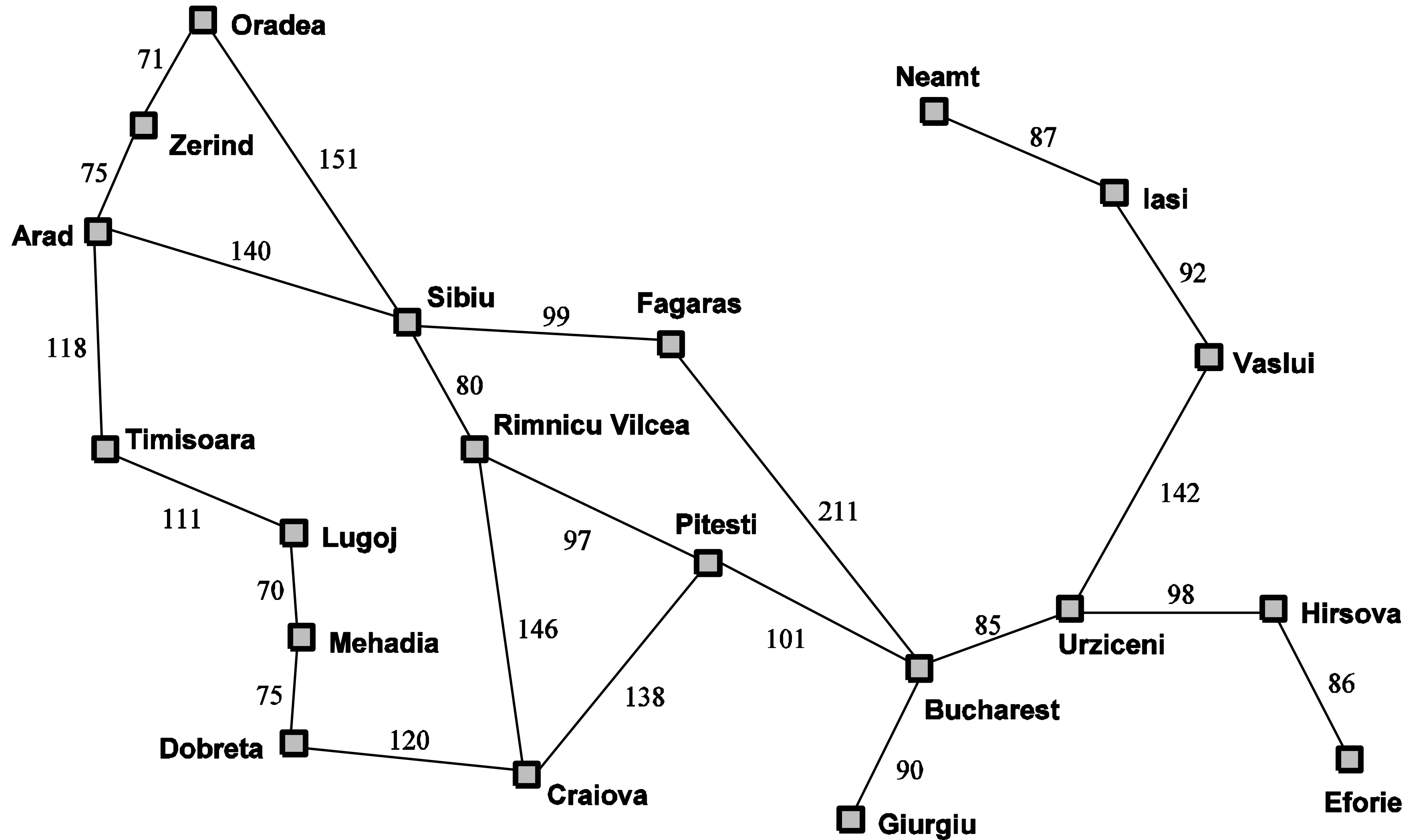
```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal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

● Basic idea:

- offline, simulated exploration of state space by generating successors of already-explored states (a.k.a. ~**expanding** states)

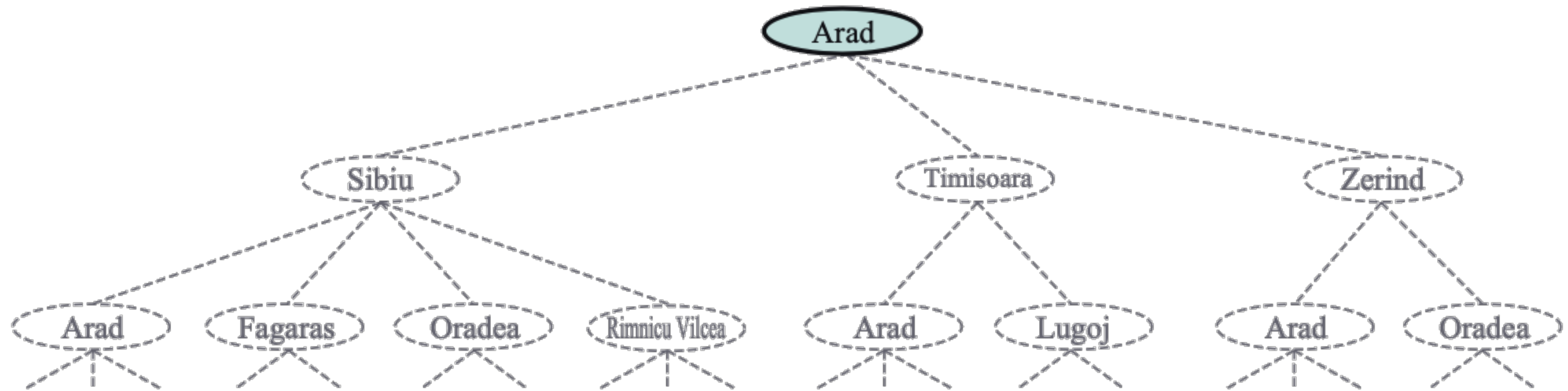
● Main question: which fringe (or frontier) nodes to explore?

Example: Route finding

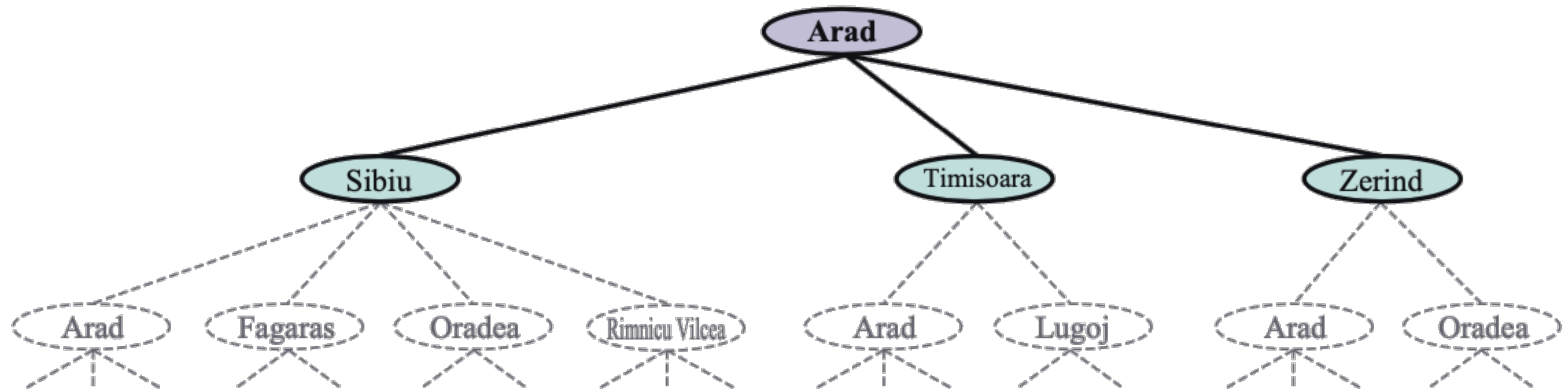


Problem: find path from **Arad** to **Bucharest**

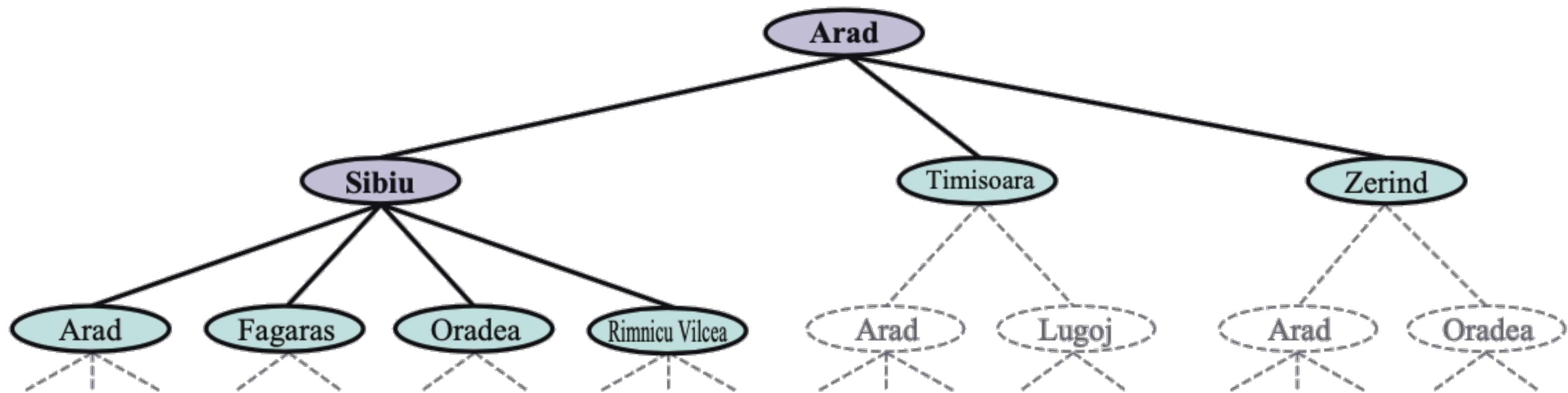
Example: Route finding



44 Example: Route finding



Example: Route finding



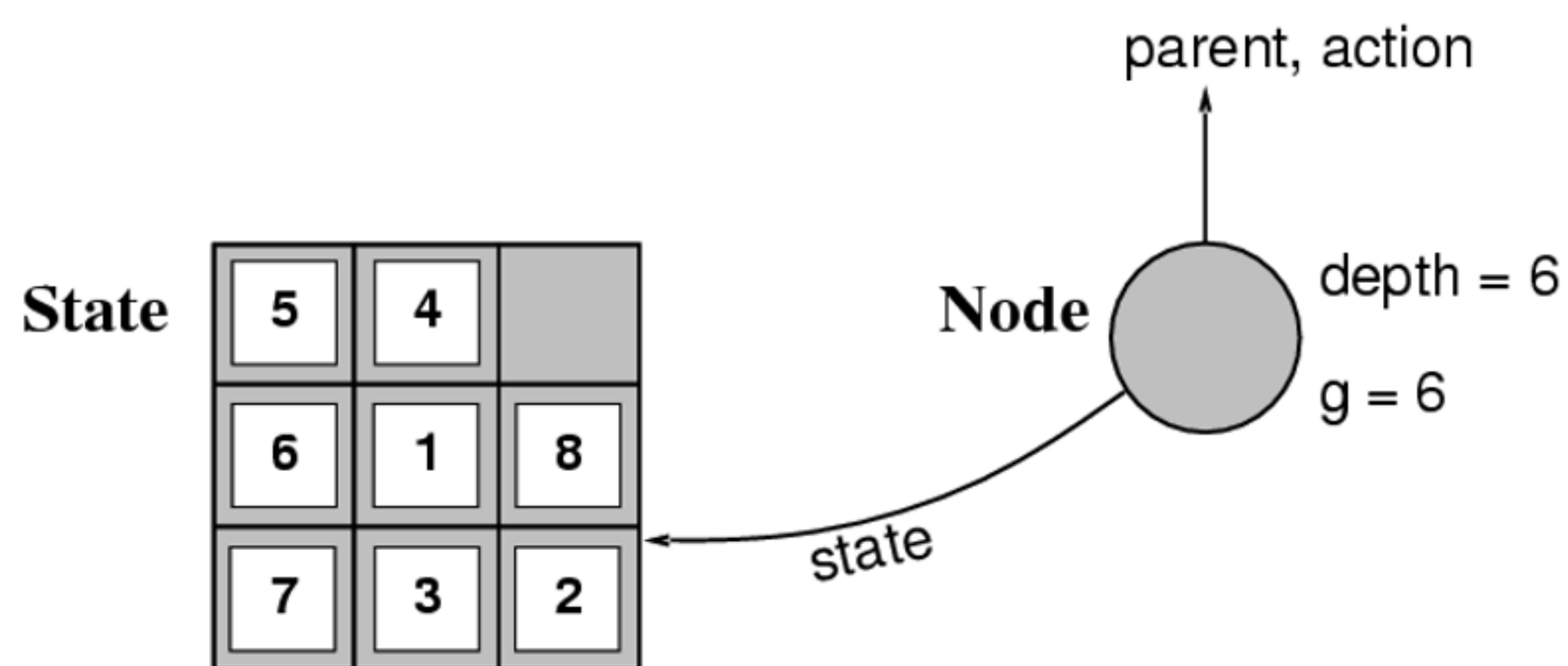
Implementation: general tree search

```
function TREE-SEARCH(problem, fringe) returns a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND(node, problem) returns a set of nodes
  successors ← the empty set
  for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
    s ← a new NODE
    PARENT-NODE[s] ← node; ACTION[s] ← action; STATE[s] ← result
    PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
    DEPTH[s] ← DEPTH[node] + 1
    add s to successors
  return successors
```

47 State vs. Node

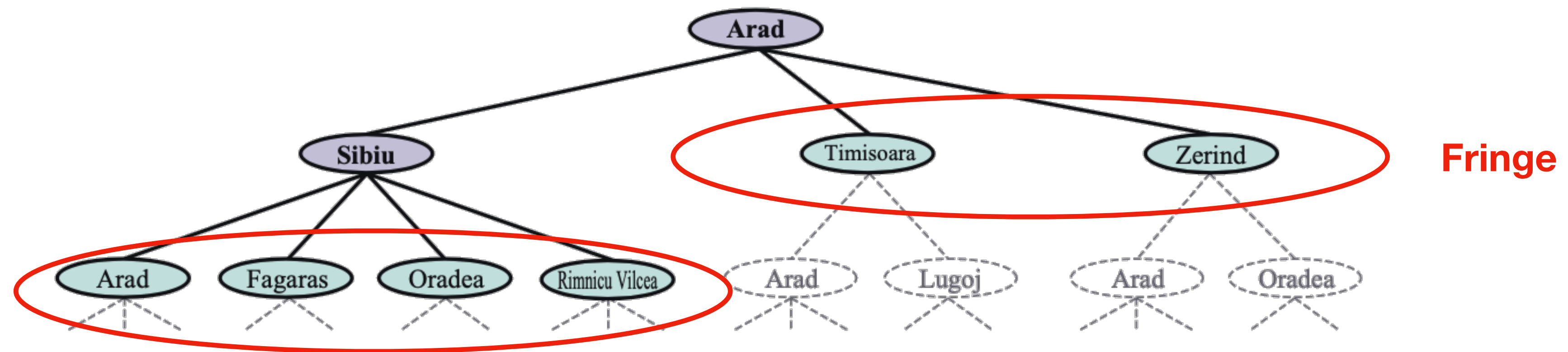
- A **state** is a representation of a physical configuration
- A **node** is a data structure constituting part of a search tree, includes **state, parent node, action, path cost $g(x)$, depth**



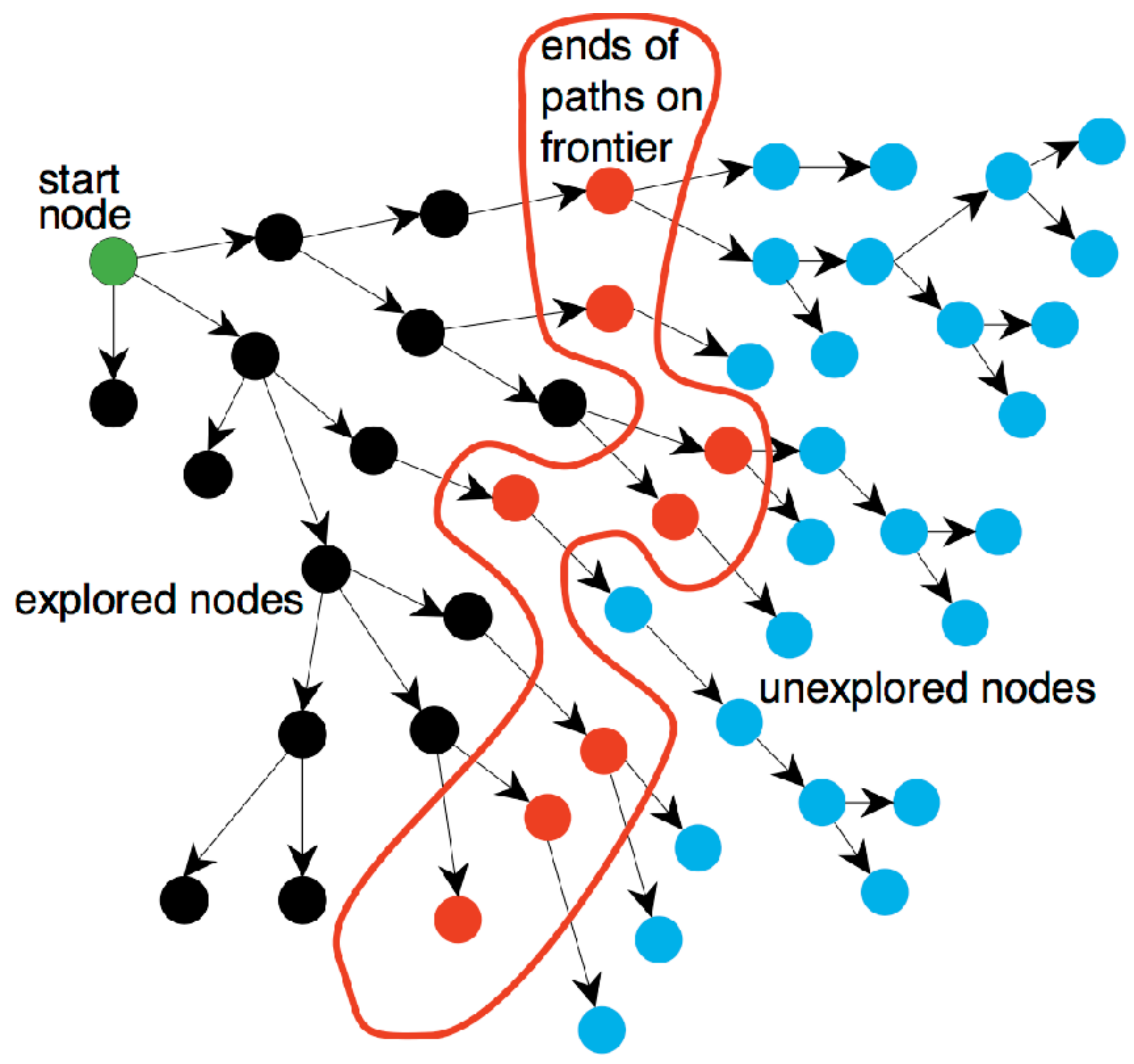
- The Expand function creates new nodes, filling in the various fields and using the SuccessorFn of the problem to create the corresponding states.

Fringe (Frontier)

- If you're performing a tree (or graph) search, then the set of all nodes at the end of all visited paths is called the *fringe*, *frontier* or *border*.
- Implemented as a queue FRINGE.
 - INSERT (node, FRINGE)
 - REMOVE (FRINGE)
- The ordering of the nodes in FRINGE defines the search strategy.



Fringe (Frontier)



Search strategies

- ⦿ A search strategy is defined by picking the **order of node expansion**
- ⦿ Strategies are evaluated along the following dimensions:
 - **Completeness**: Is the algorithm guaranteed to find a solution when there is one, and to correctly report failure when there is not?
 - **Cost optimality**: Does it find a solution with the lowest path cost of all solutions?
 - **Time complexity**: How long does it take to find a solution? This can be measured in seconds, or more abstractly by the number of states and actions considered.
 - **Space complexity**: How much memory is needed to perform the search?
- ⦿ Time and space complexity are measured in terms of
 - *b*: maximum branching factor of the search tree
 - *d*: depth of the least-cost solution
 - *m*: maximum depth of the state space (may be ∞)

51 Uninformed vs. Informed Strategies

- **Uninformed** (or **blind**) strategies have no clue about how close a state is to the goal(s).
- **Informed** (or **heuristic**) strategies exploits such information to assess that one node is “more promising” than another.

Uninformed search	Informed search
<p>➤ look for solutions by <u>systematically</u> generating new states and checking each of them against the goal.</p> <ol style="list-style-type: none">1. It is very <u>inefficient</u> in most cases.2. Most successor states are “obviously” a bad choice.3. Such strategies do not use problem-specific knowledge	<ol style="list-style-type: none">1. They are almost always <u>more efficient</u> than uninformed strategies.2. May reduce time and space complexities.3. Evaluation function $f(n)$ measures distance to the goal.4. Order nodes in Frontier according to $f(n)$ and decide which node to expand next.

Uninformed Strategies

Uninformed search strategies use only the information available in the problem definition.

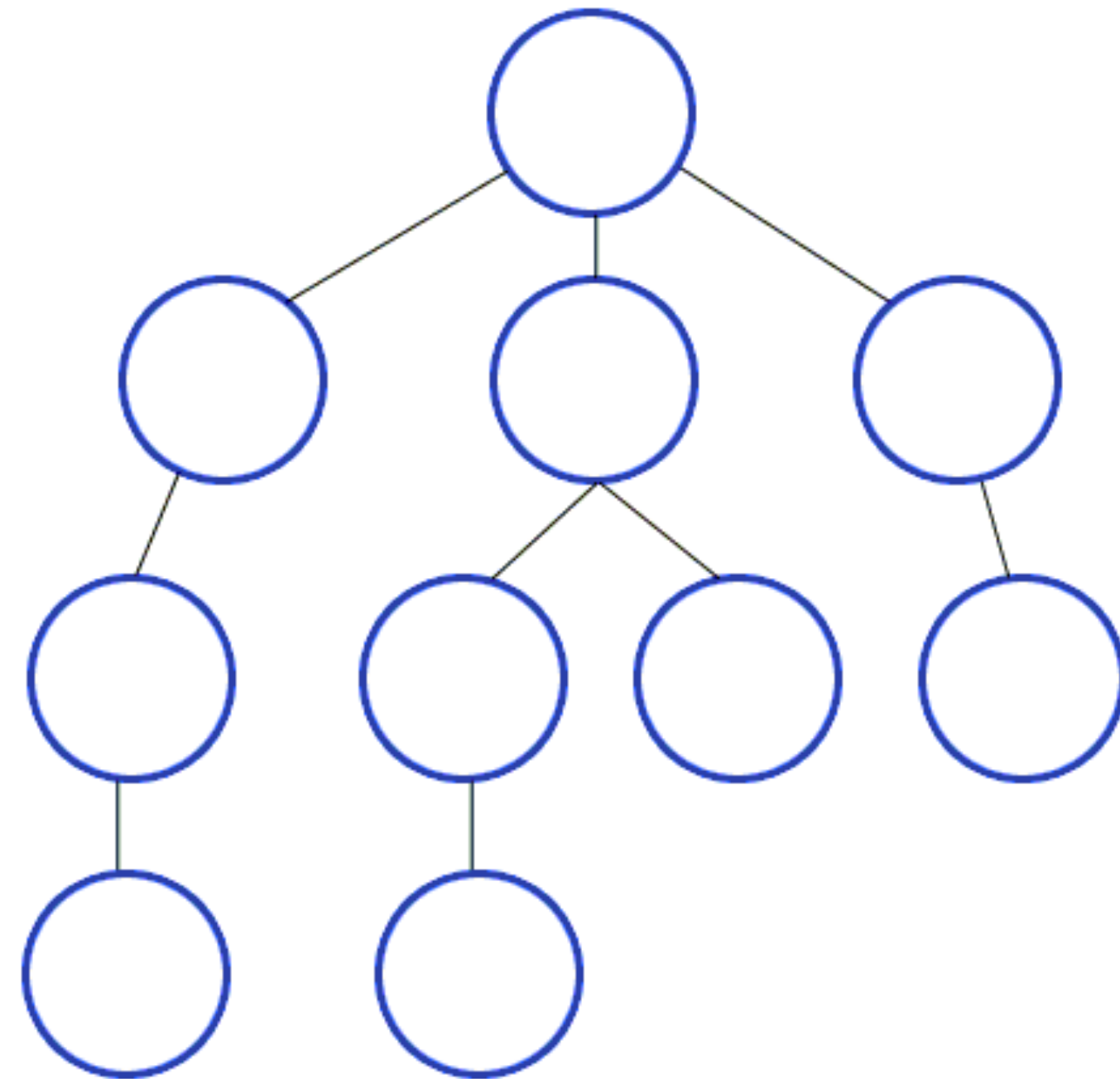
- **Breadth-First Search**
- **Depth-First Search**
- **Uniform-Cost Search**
- Depth-Limited Search
- Iterative Deepening Search

Breadth-First Search



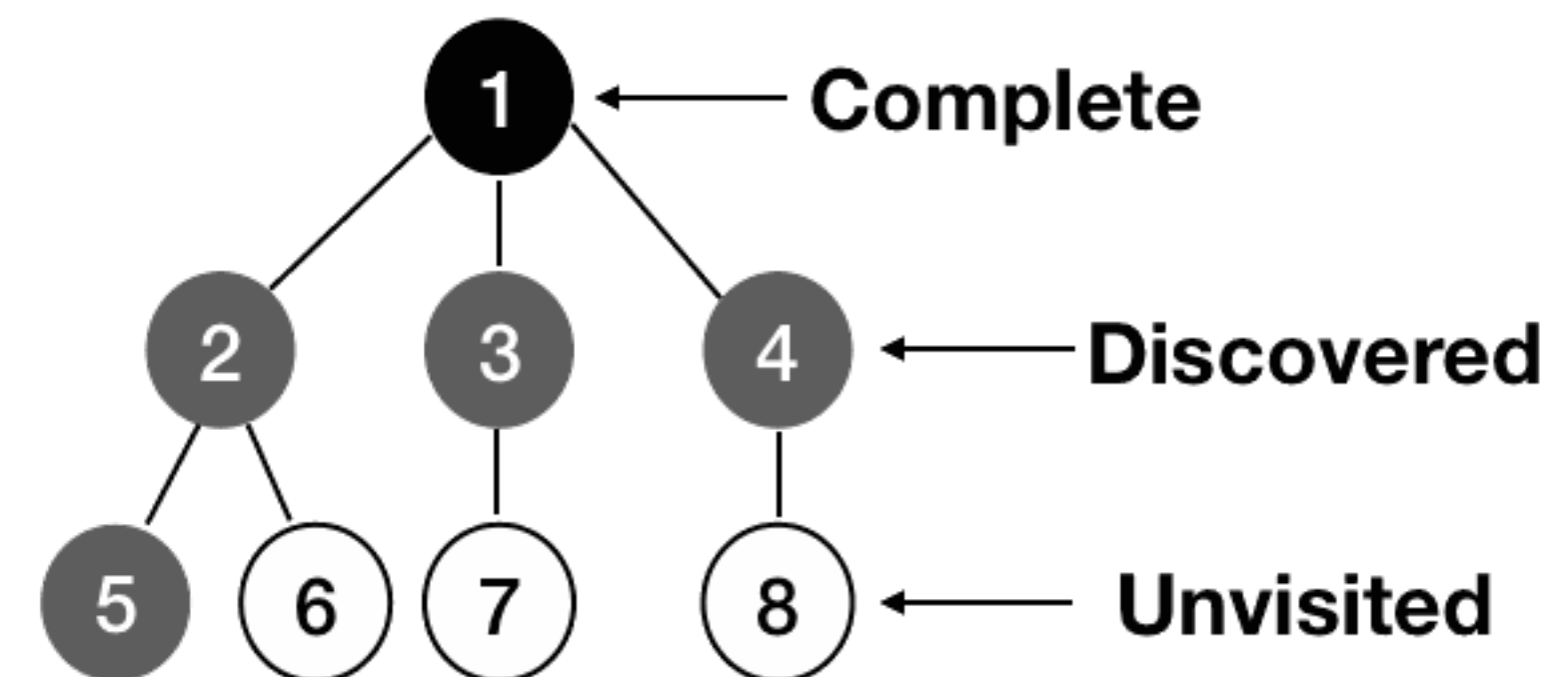
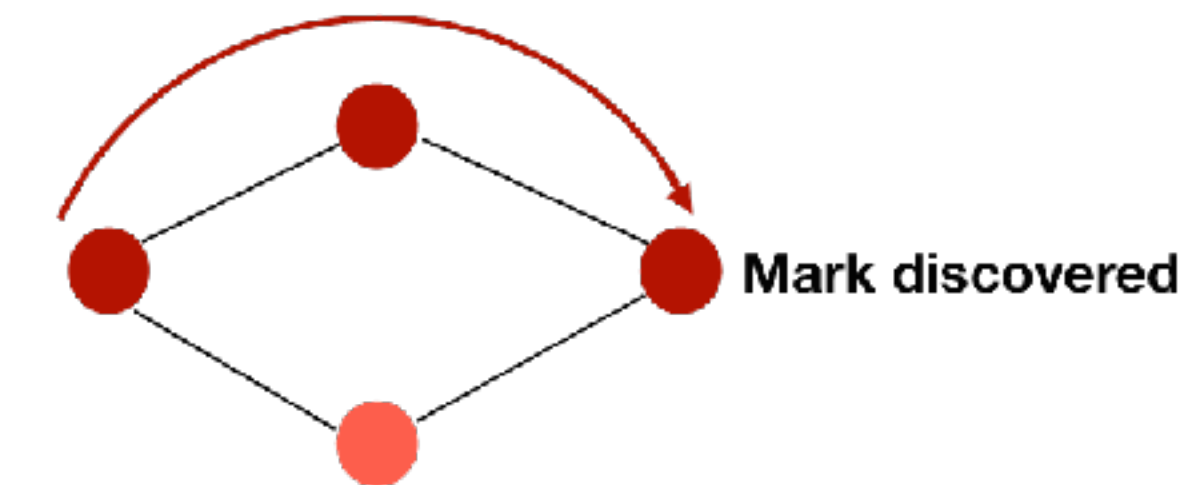
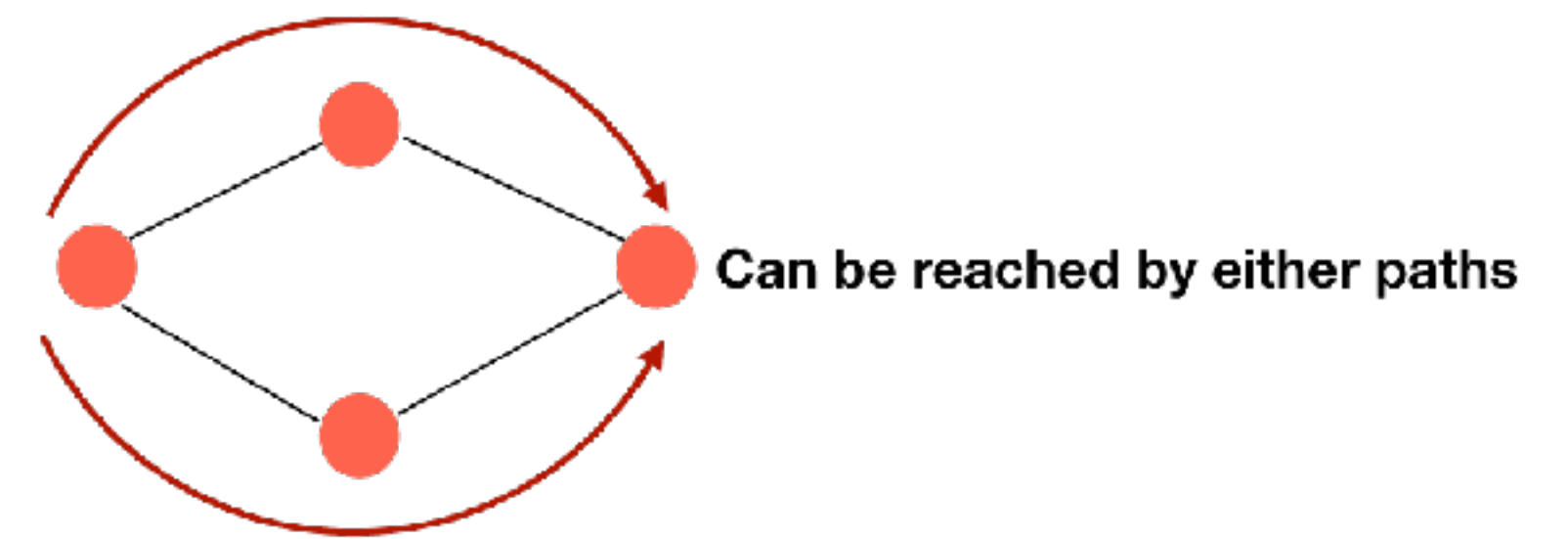
Breadth-First Search (BFS)

- BFS first visit all the nodes at the same depth first and then proceed visiting nodes at a deeper depth (strategy: expand a shallowest node first)



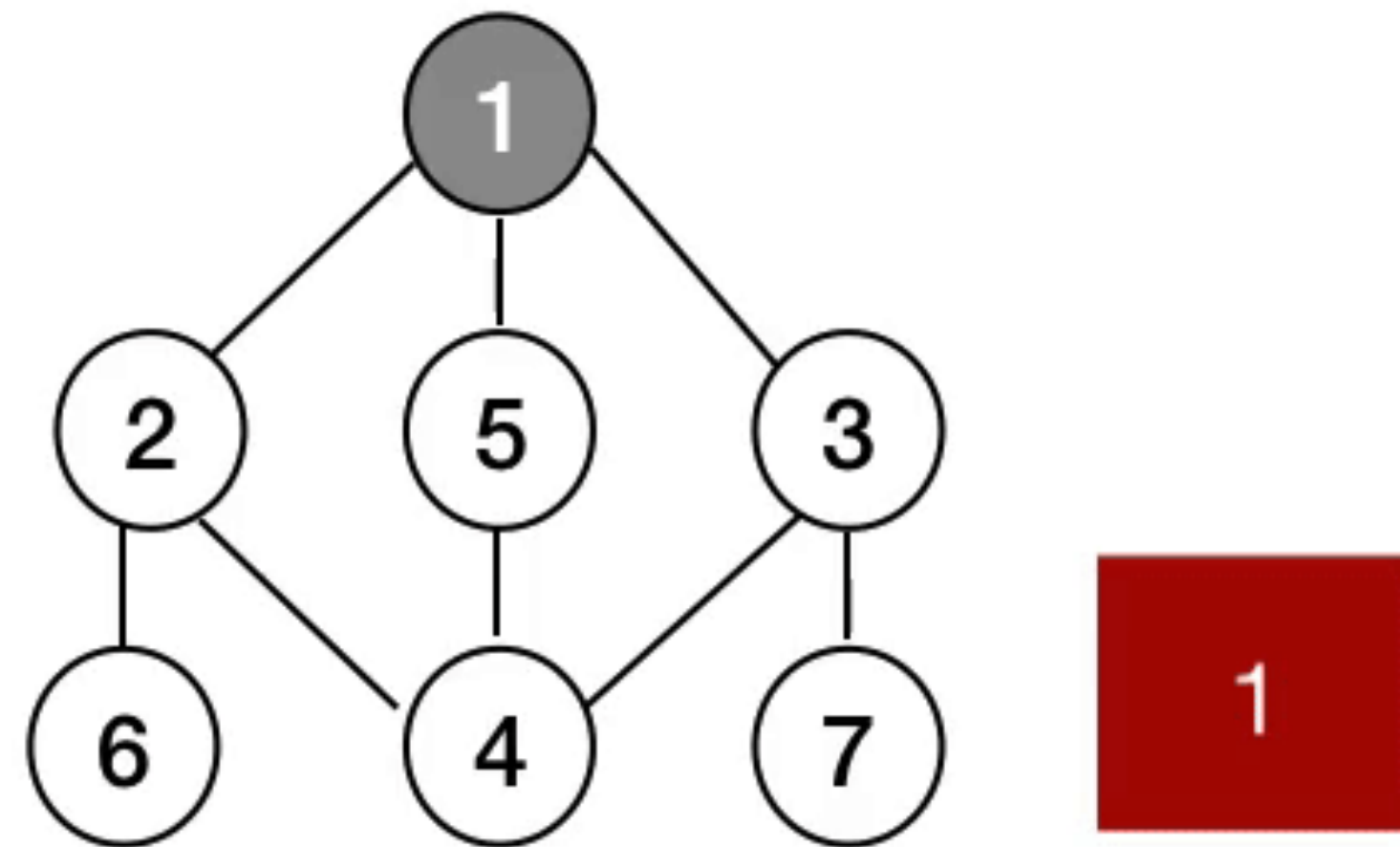
Breadth-First Search

- A node can be reached from different nodes using different paths
- But we need to visit each node only once.
- So, we mark each node differently into 3 categories - **unvisited, discovered** and **complete**.
 - Initially, all nodes are unvisited. After visiting a node for the first time, it becomes discovered.
 - A node is complete if all of its adjacent nodes have been visited.
 - Thus, all the adjacent nodes of a complete node are either discovered or complete.

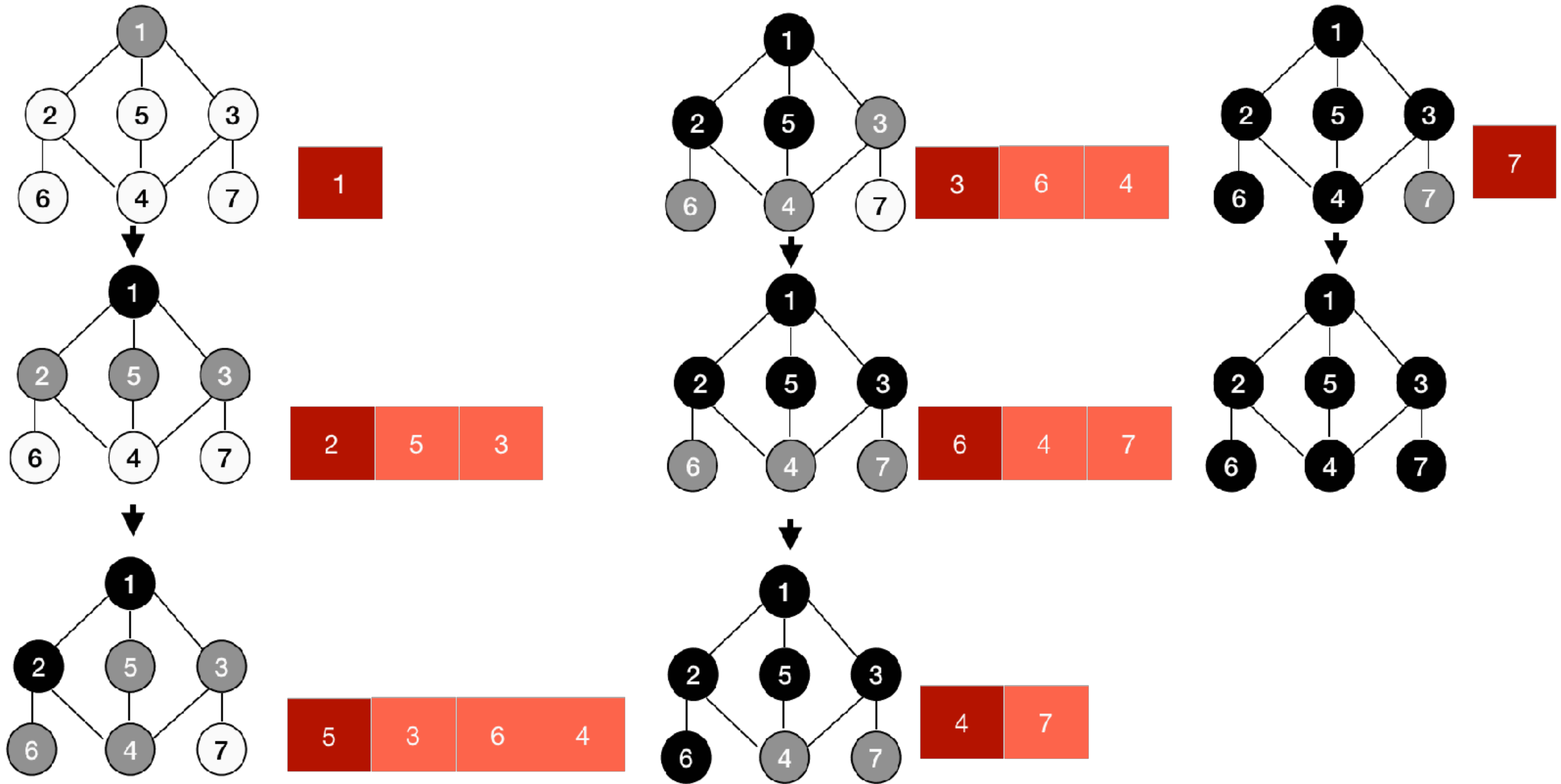


Breadth-First Search

- ⦿ Thus after visiting a node, we first visit all its sibling and then their children.
- ⦿ We can use a first in first out (FIFO) queue to achieve this (Fringe is a FIFO queue).
- ⦿ Starting from the source, we can put all its adjacent nodes in a queue

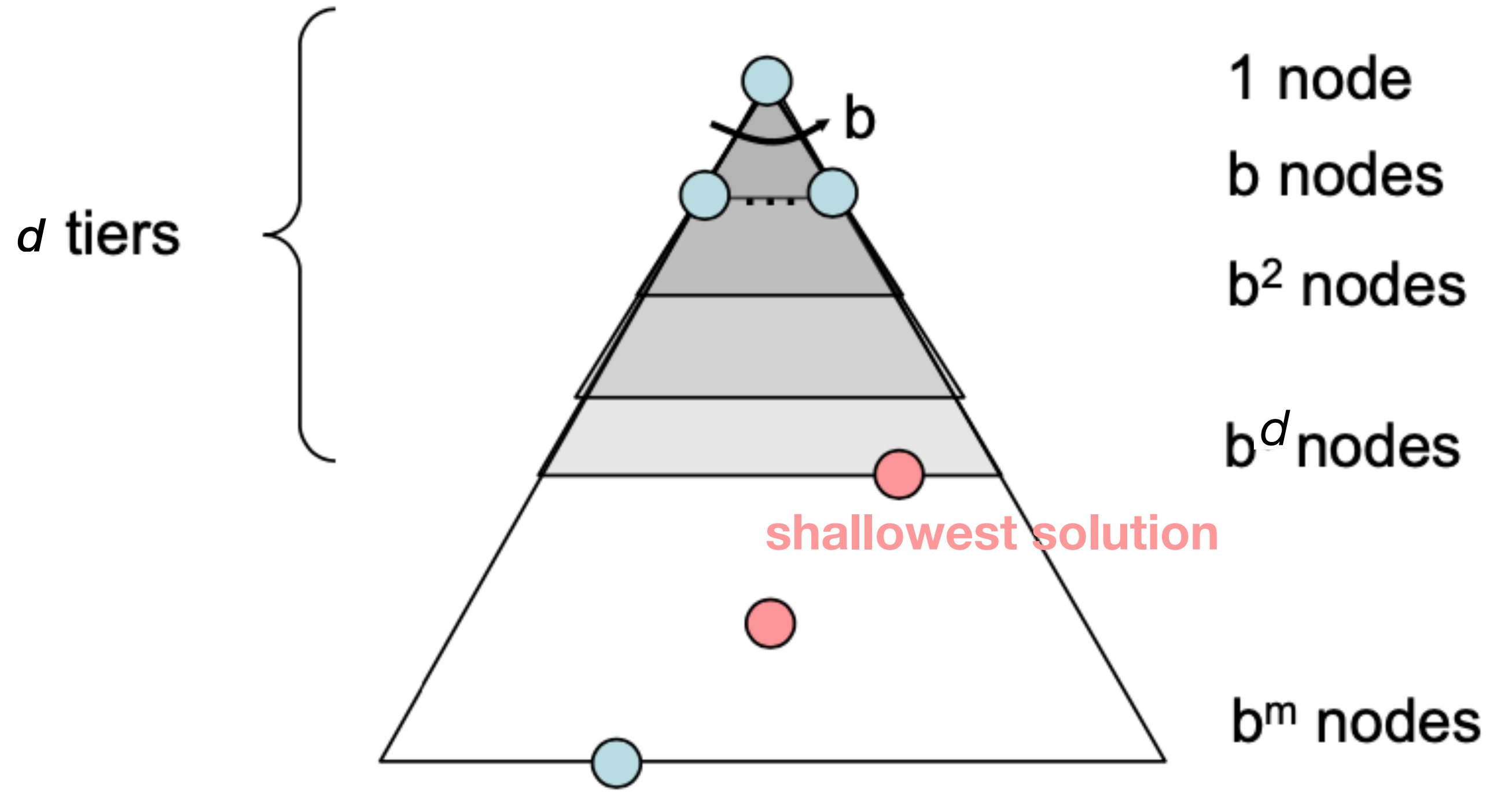


Breadth-First Search



58 Breadth-First Search (BFS) Properties

- Is it complete?
 - Yes. d must be finite if a solution exists.
- Is it optimal?
 - Only if costs are all 1.
- Search time?
 - $1 + b + b^2 + b^3 + \dots + b^d = O(b^d)$
- Fringe space?
 - Has roughly the last tier, so $O(b^d)$

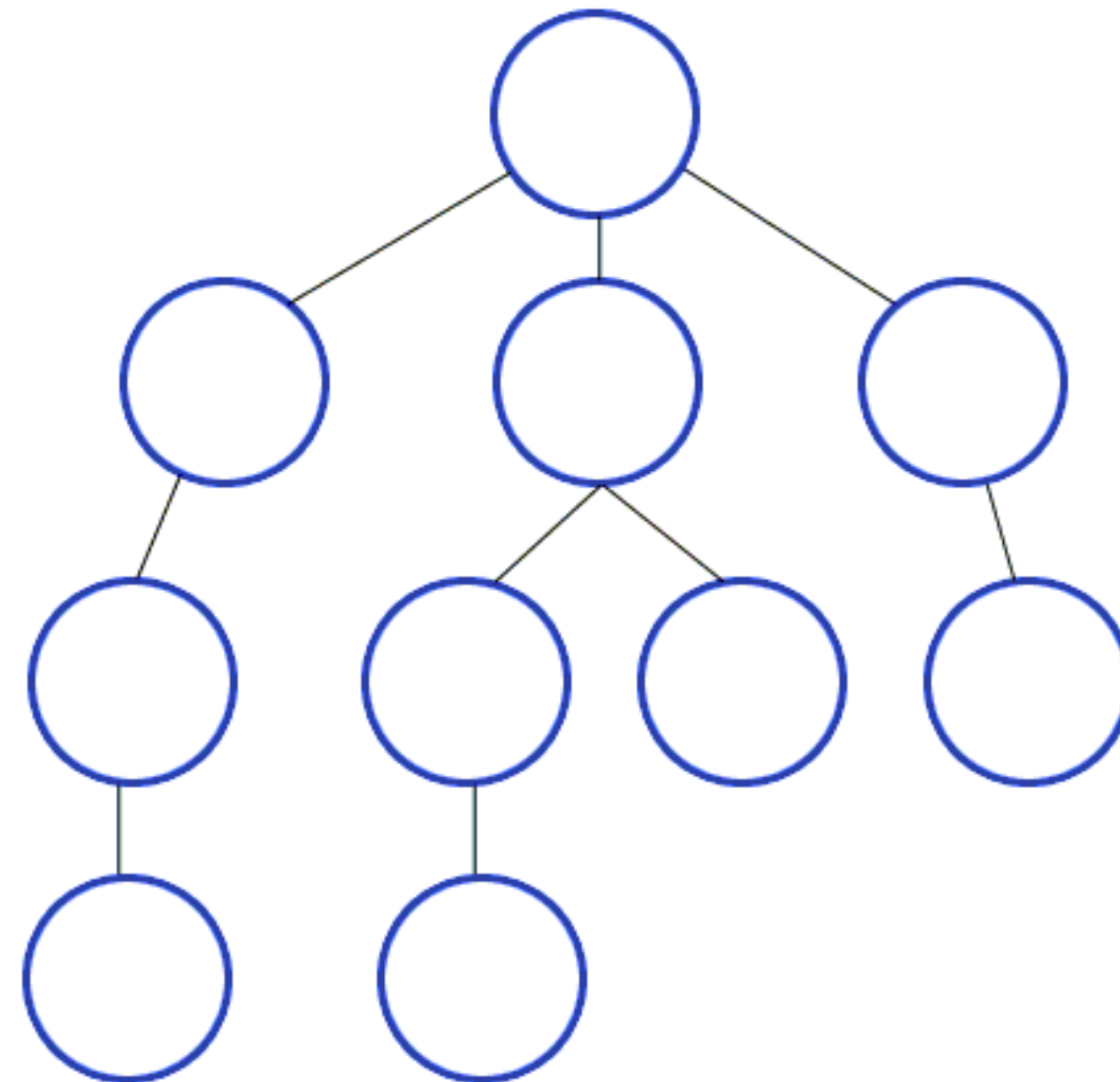


Depth-First Search



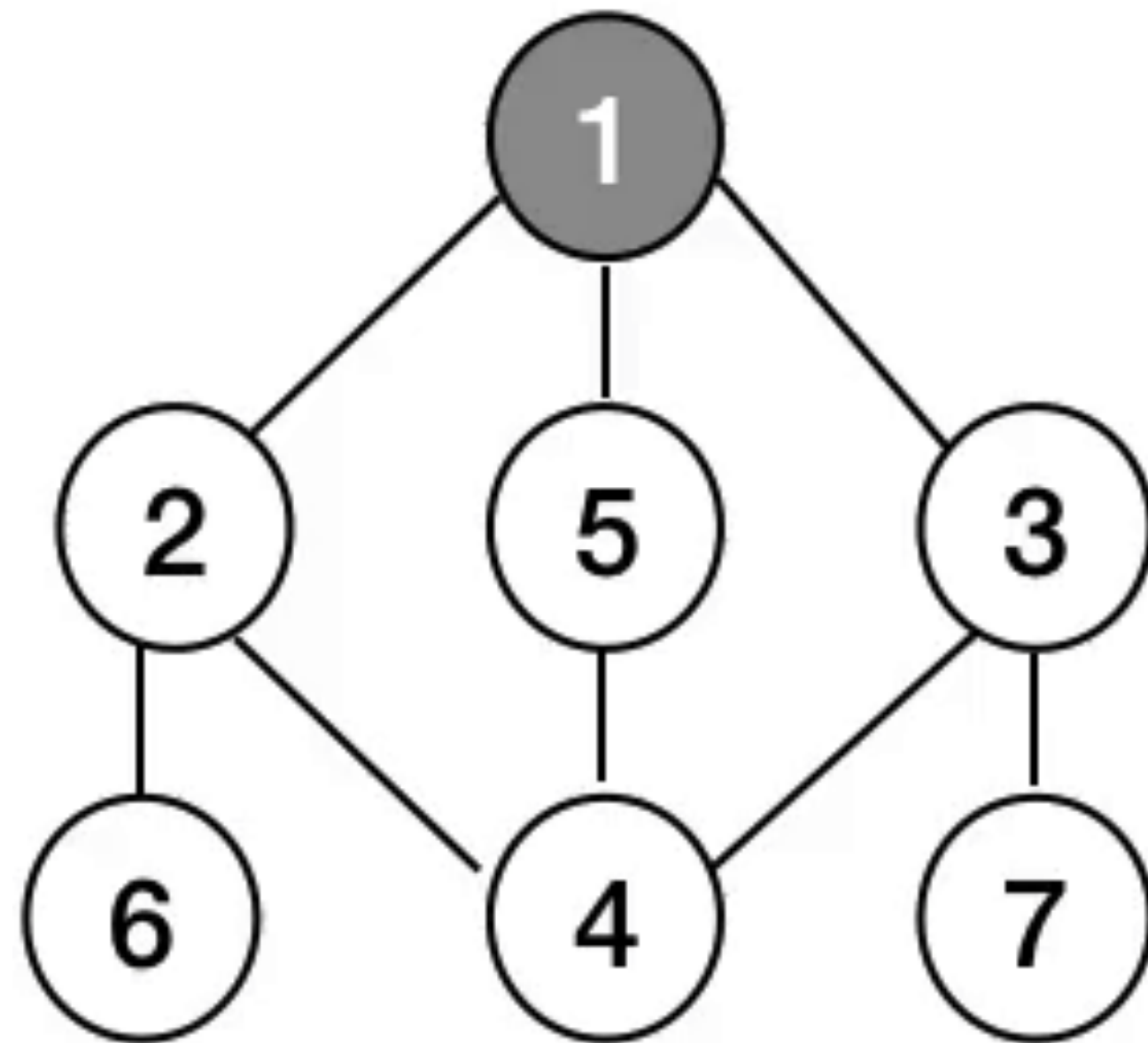
60 Depth-First Search (DFS)

- DFS first explores the depth of the graph before the breadth i.e., it traverses along the increasing depth and upon reaching the end, it backtracks to the node from which it was started and then do the same with the sibling node.

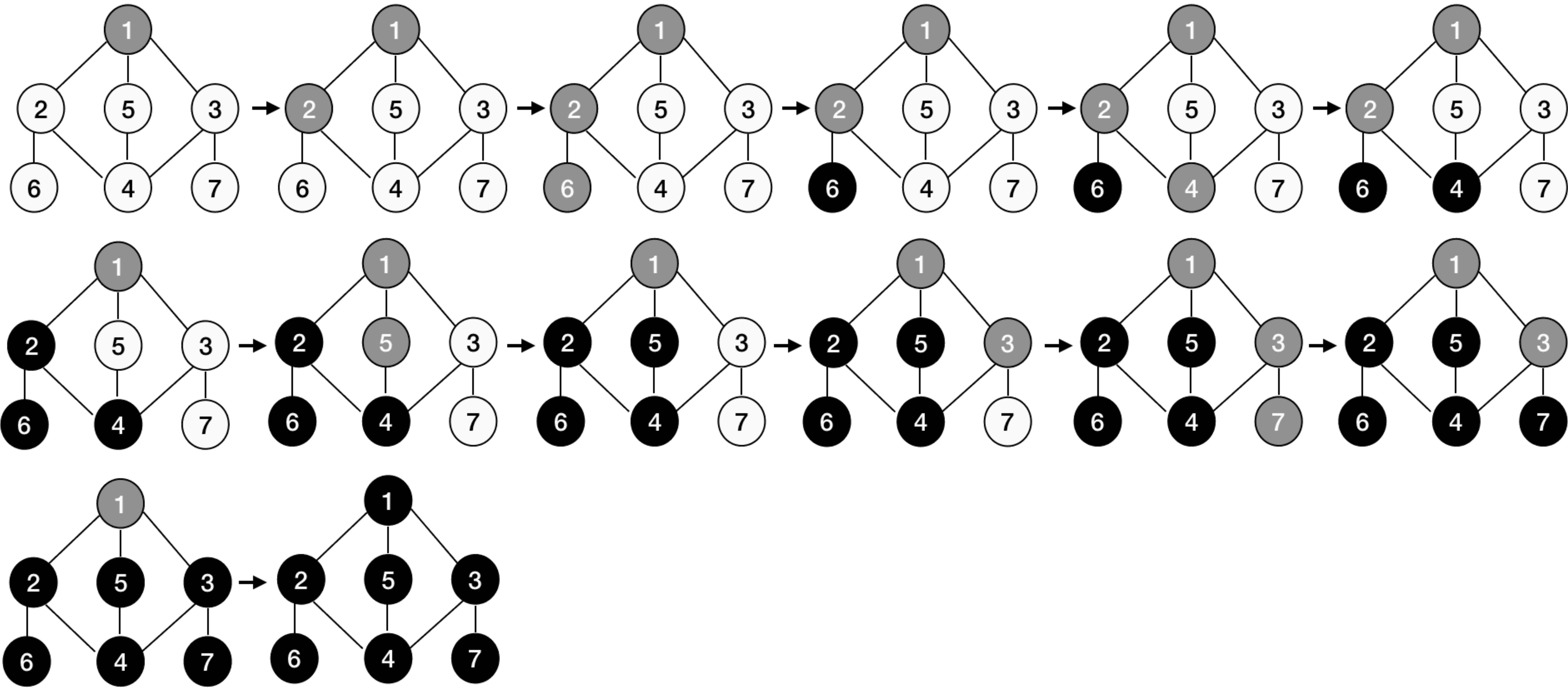


61 Depth-First Search

- Similar to the BFS, we also mark the vertices white, gray and black to represent unvisited, discovered and complete respectively.
- Fringe = LIFO queue, i.e., a stack that put successors at front.

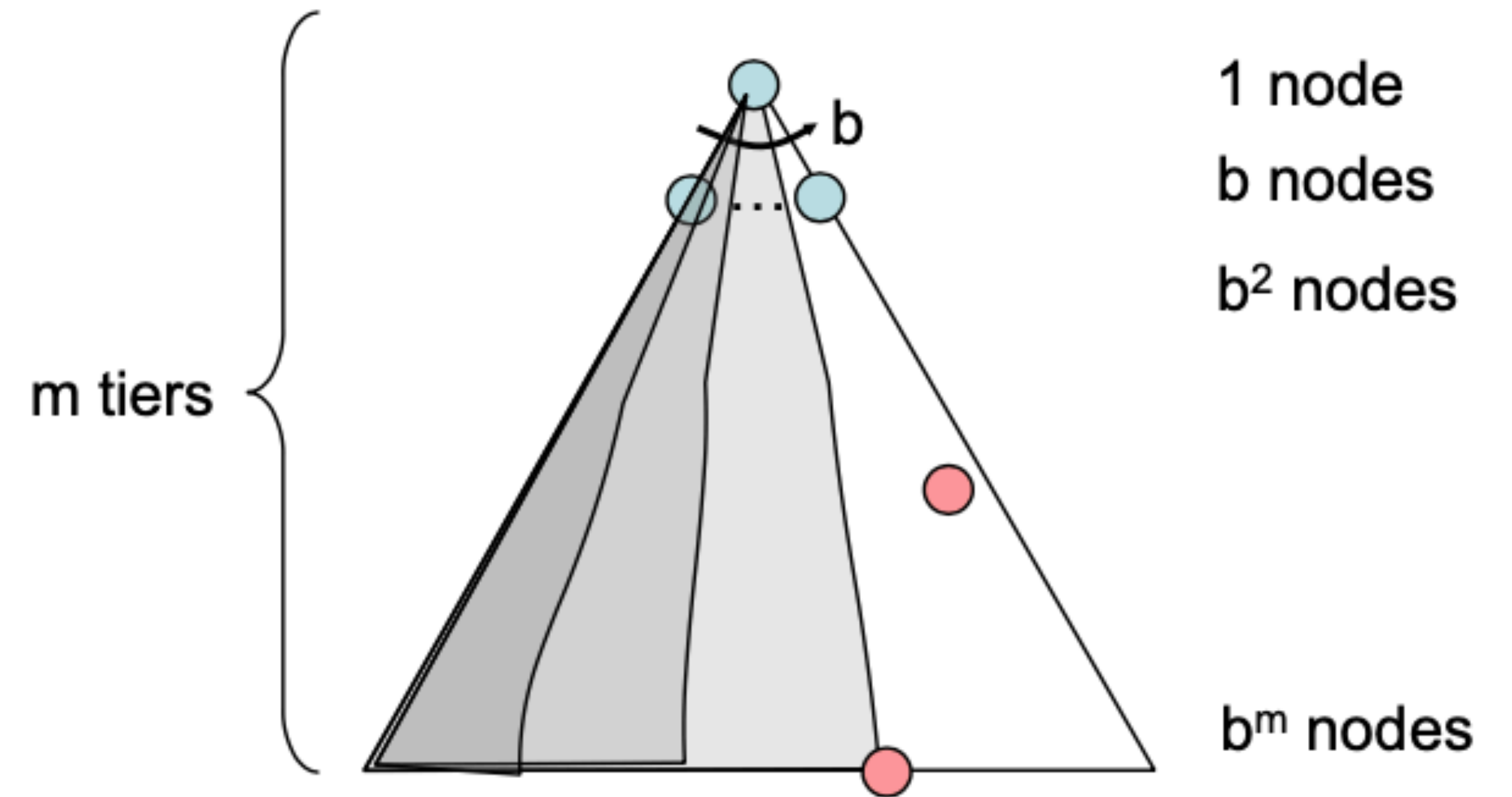


62 Depth-First Search



63 Depth-First Search (DFS) Properties

- Is it complete?
 - m could be infinite, so only if we prevent cycles
- Is it optimal?
 - No, it finds the “leftmost” solution, regardless of depth or cost
- Search time?
 - $O(b^m)$ Terrible if m is much larger than d , but if solutions are dense, may be much faster than breadth-first.
- Fringe space?
 - Only has siblings on path to root, so $O(bm)$, i.e., linear space!

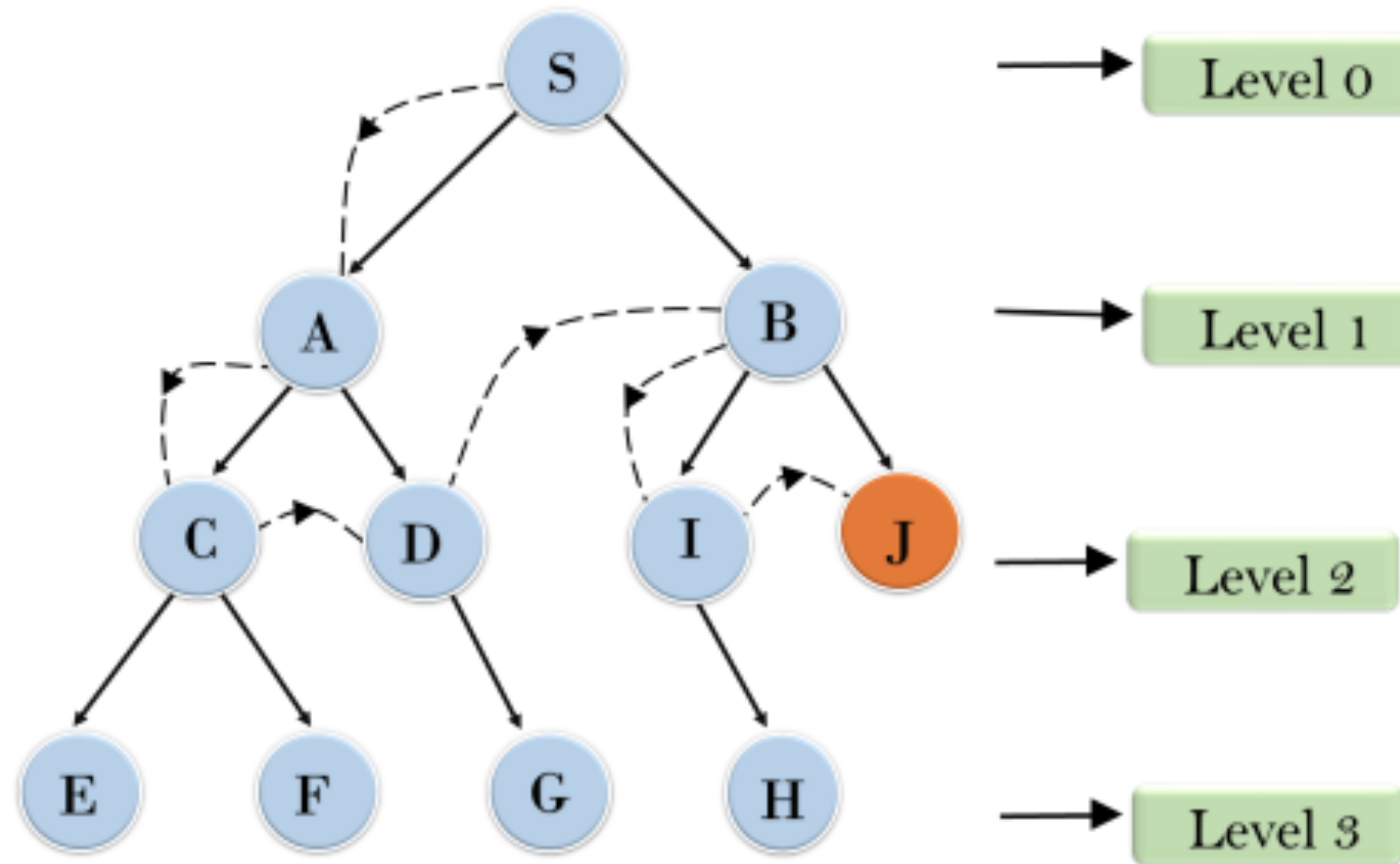


Exercise: DFS vs. BFS

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

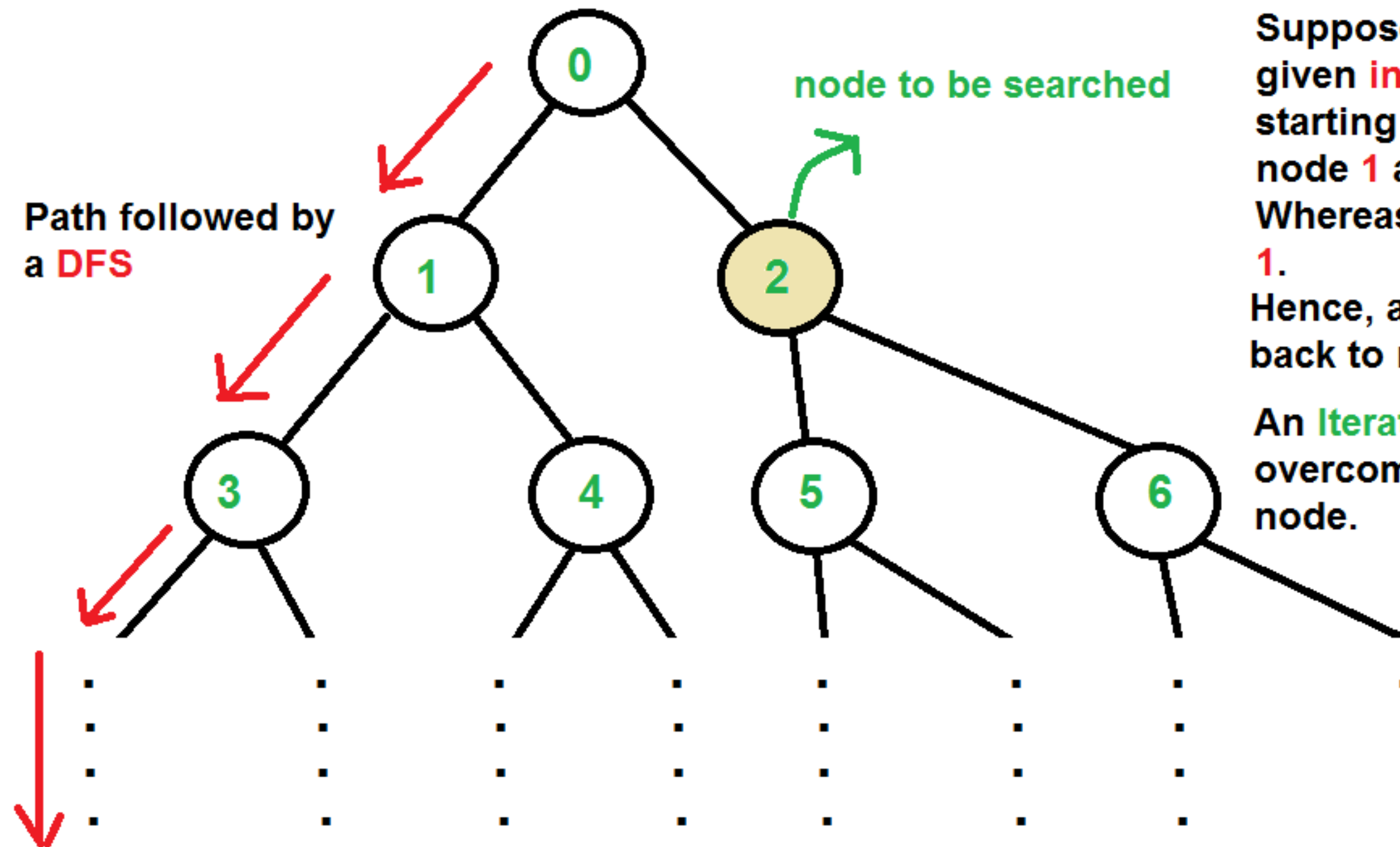
65 Depth-Limited Search

⦿ = depth-first search with depth limit L , i.e., nodes at depth L have no successors



Iterative Deepening Search

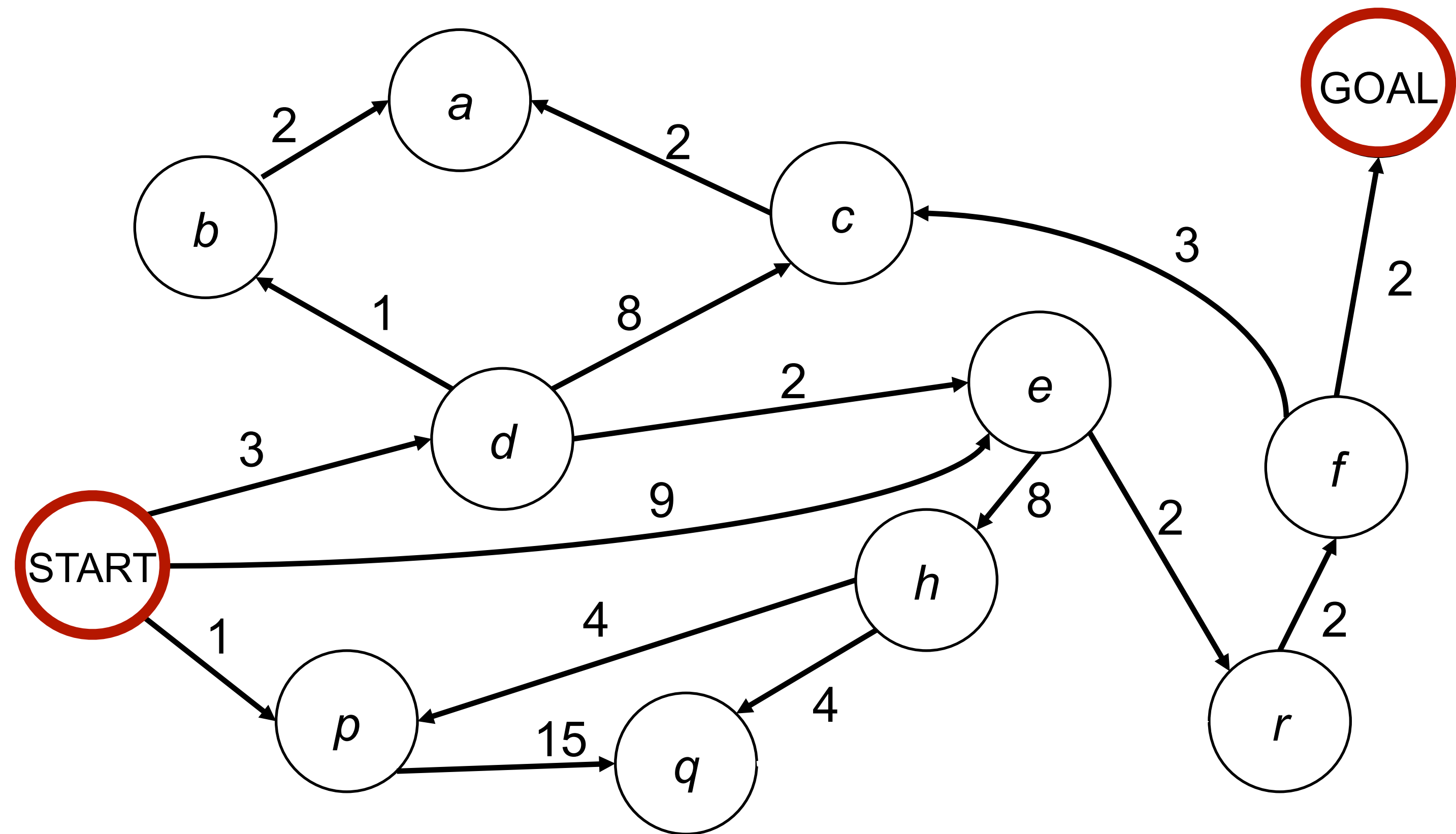
- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
 - Run a DFS with depth limit 1. If no solution...
 - Run a DFS with depth limit 2. If no solution...
 - Run a DFS with depth limit 3.



Suppose, we want to find node- '2' of the given infinite undirected graph/tree. A DFS starting from node- 0 will dive left, towards node 1 and so on. Whereas, the node 2 is just adjacent to node 1. Hence, a DFS wastes a lot of time in coming back to node 2.

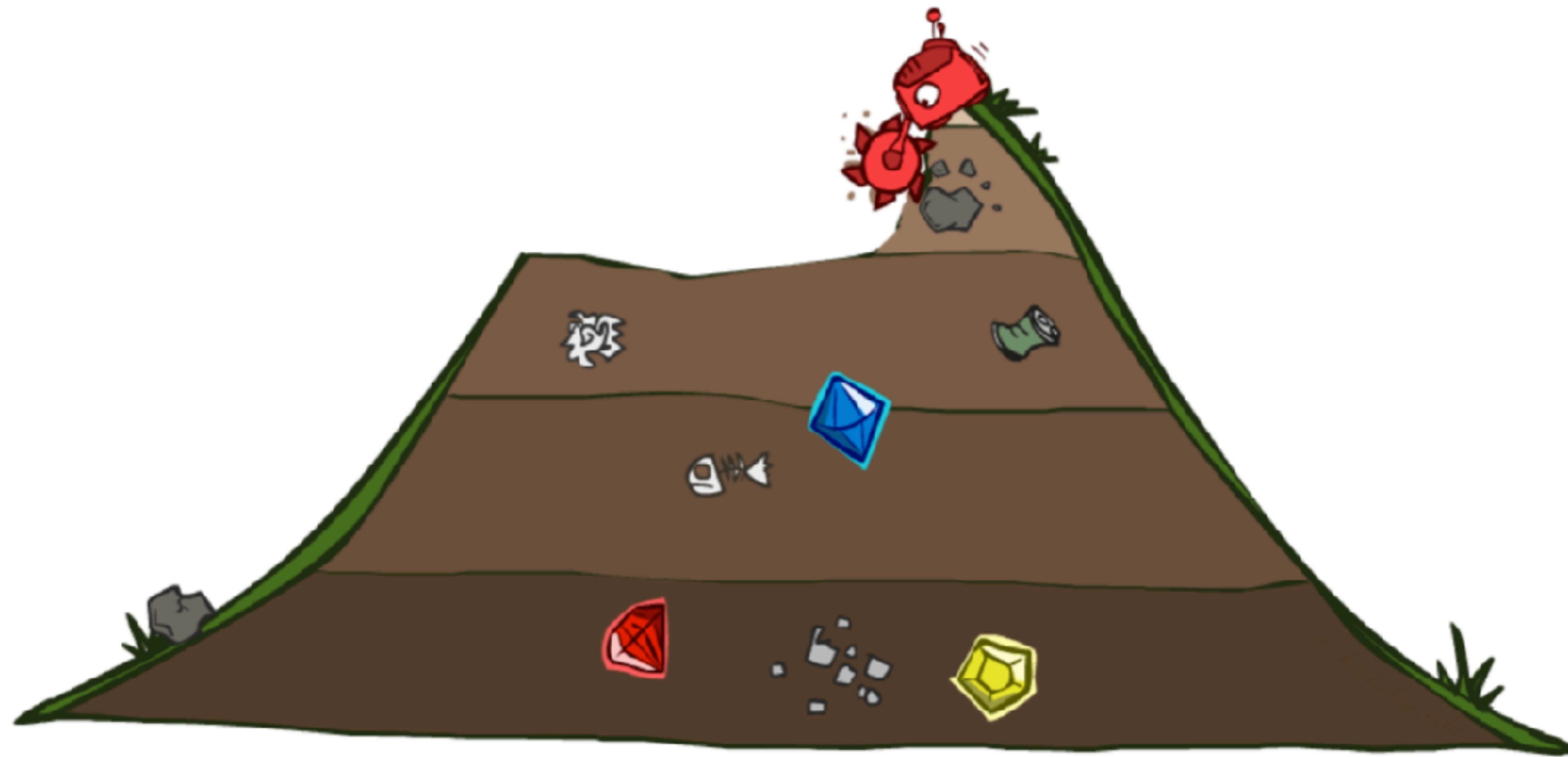
An Iterative Deepening Depth First Search overcomes this and quickly find the required node.

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.

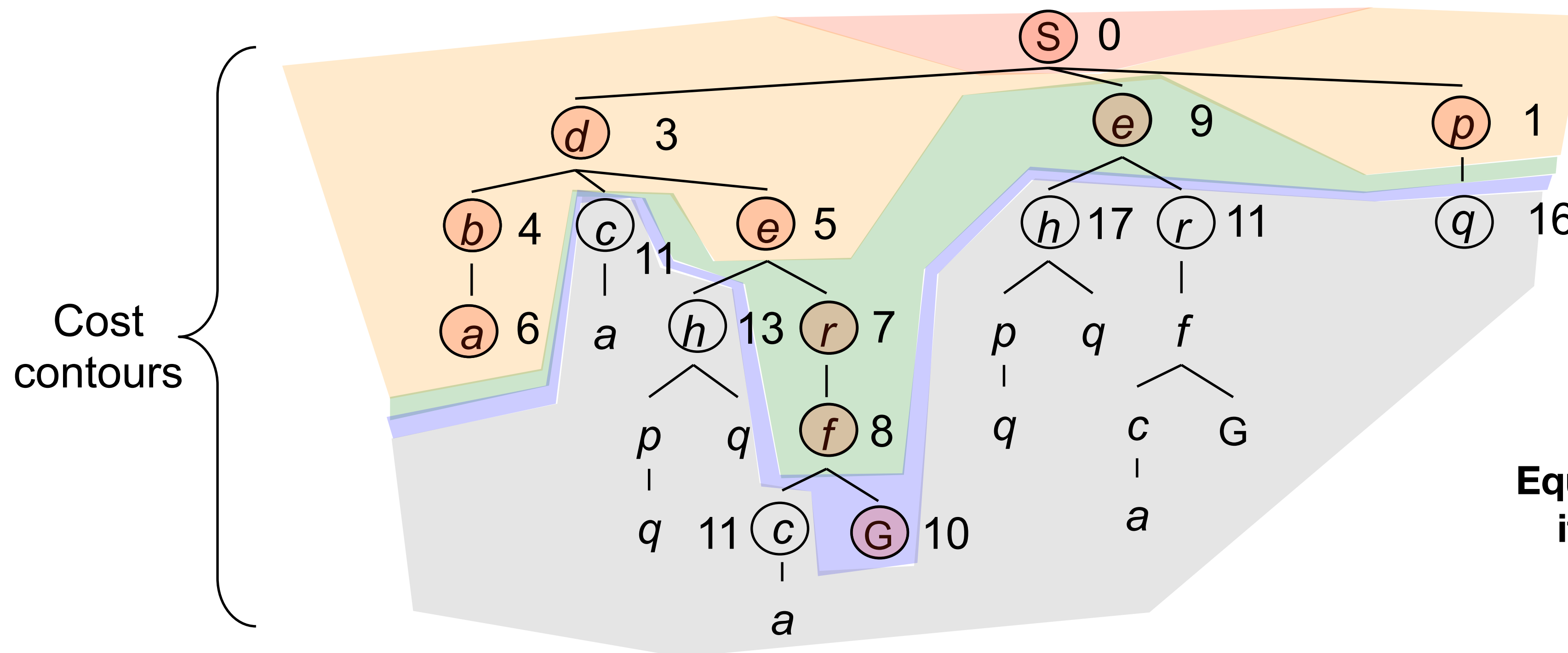
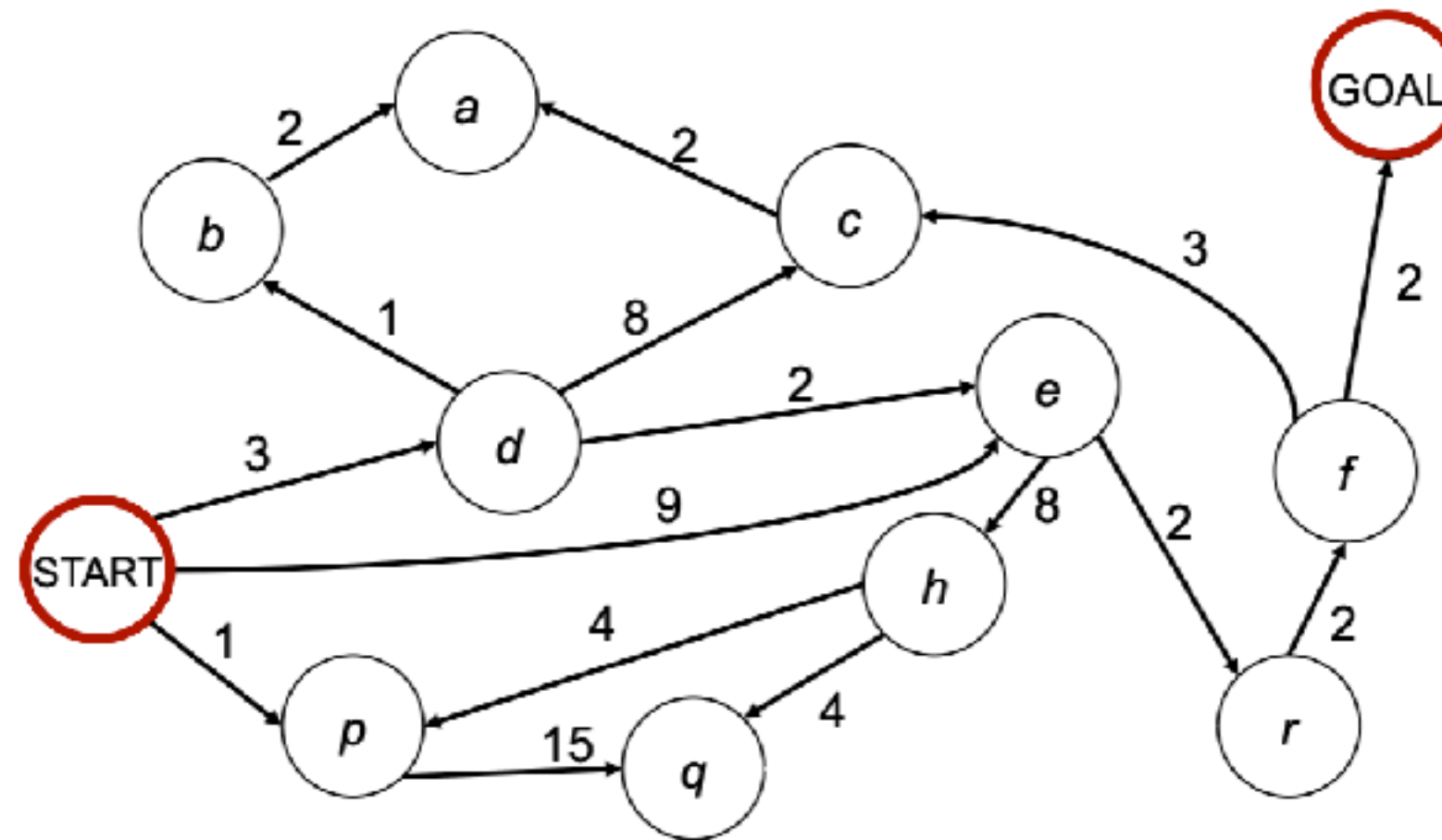
Uniform-Cost Search



69 Uniform-Cost Search (Dijkstra's algorithm)

Strategy: expand a cheapest node first:

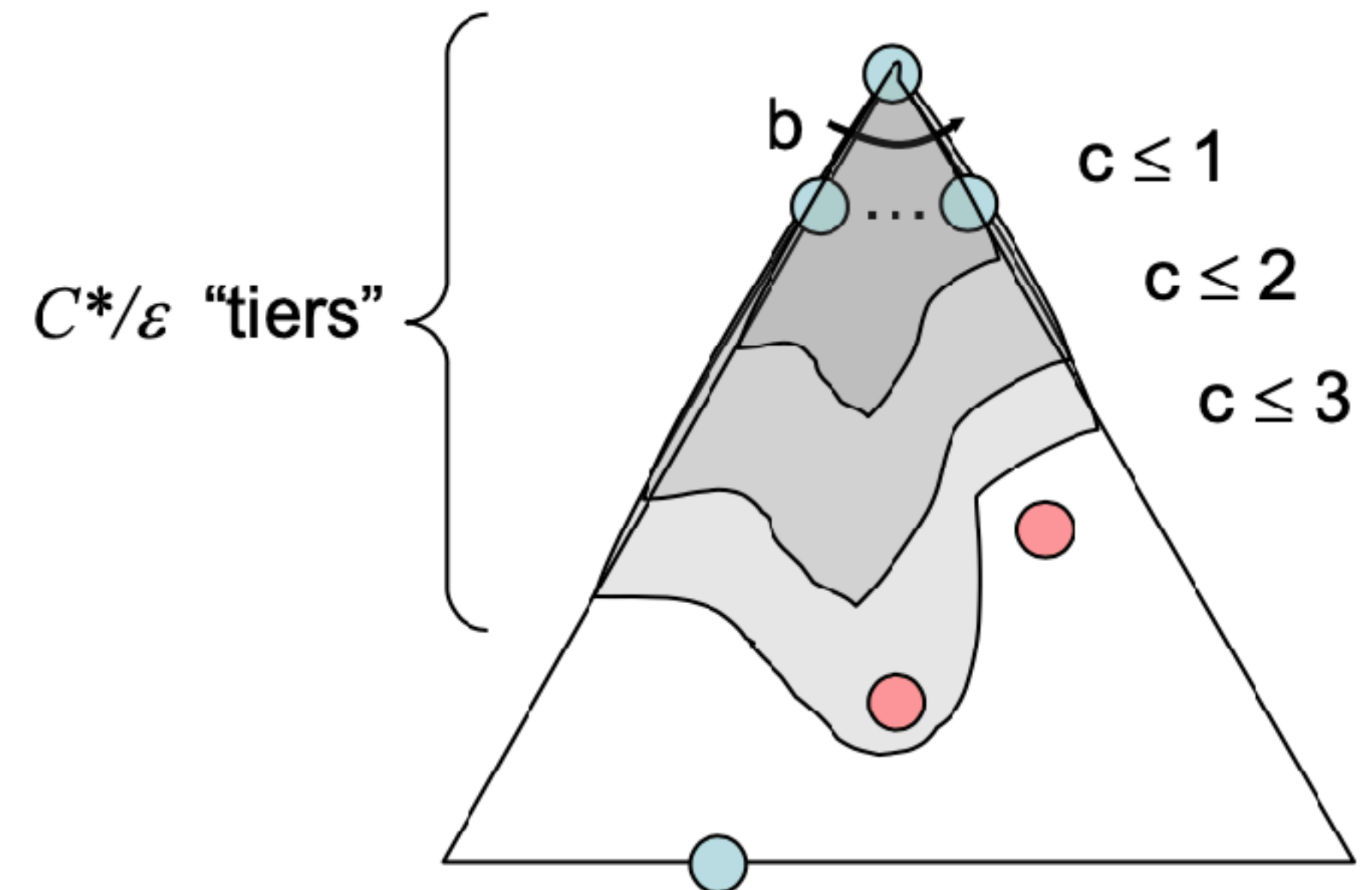
Fringe is a priority queue (priority: cumulative cost)



Equivalent to breadth-first if step costs all equal.

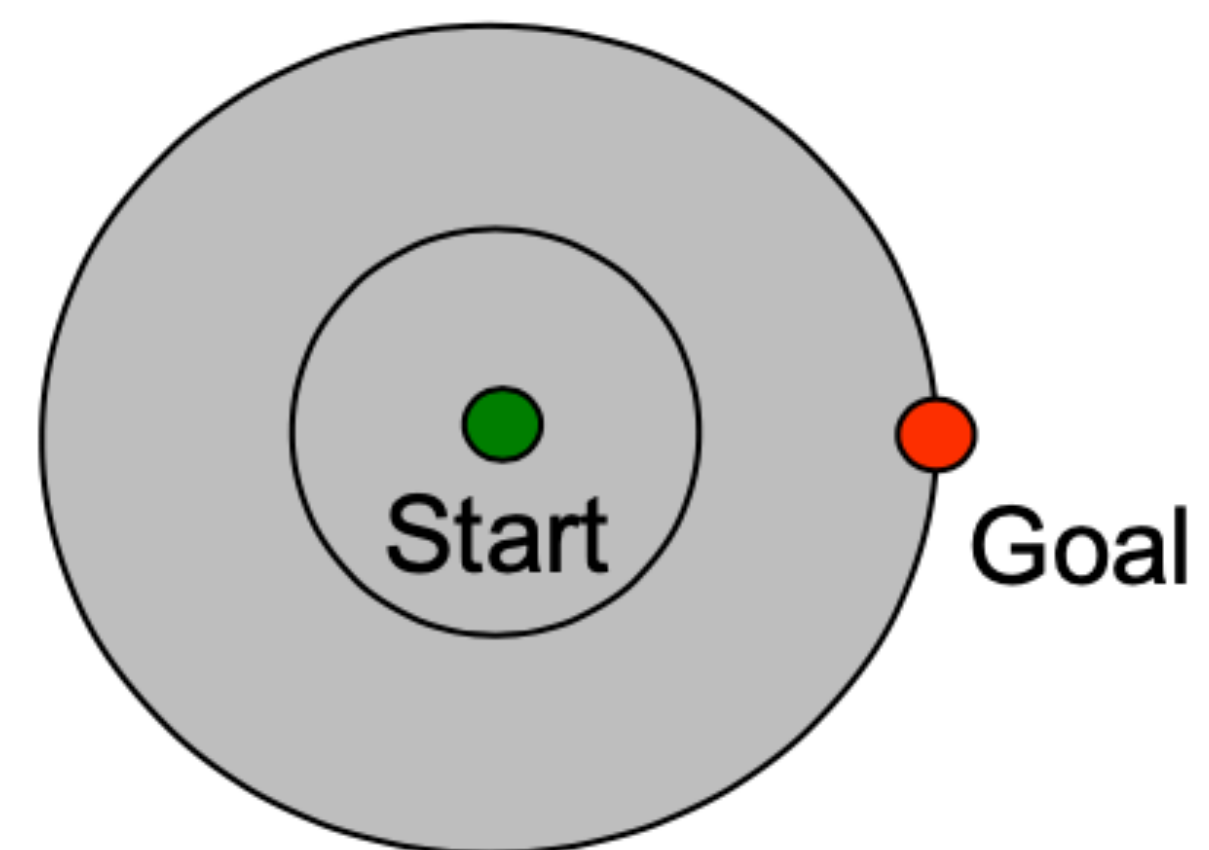
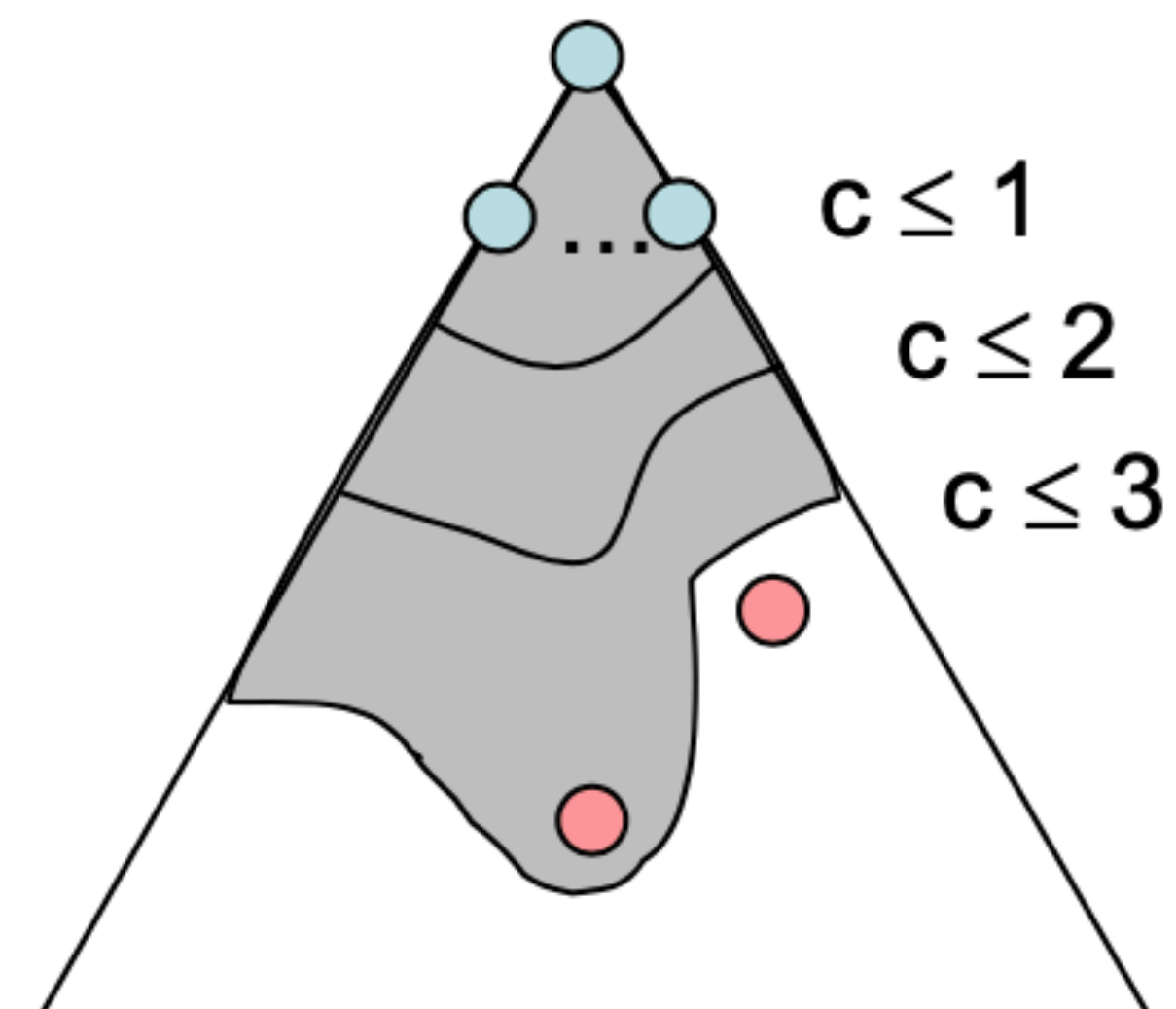
70 Properties of Uniform-Cost Search

- **Complete?** Yes, if step cost $\geq \epsilon > 0$ (ϵ is the lower bound of each action), it will never get caught going down a single infinite path
- **Time?** # of nodes with $g \leq \text{cost of optimal solution}$, $O(b^{\text{ceiling}(C^*/\epsilon)})$ where C^* is the cost of the optimal solution.
- **Space?** # of nodes with $g \leq \text{cost of optimal solution}$, $O(b^{\text{ceiling}(C^*/\epsilon)})$.
- **Optimal?** Yes – nodes expanded in increasing order of $g(n)$.



71 Uniform-Cost Search 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!



Summary of Algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes ¹	Yes ^{1,2}	No	No	Yes ¹
Optimal cost?	Yes ³	Yes	No	No	Yes ³
Time	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(b\ell)$	$O(bd)$

73 Summary

- ◎ **All these search algorithms are the same except for fringe strategies**
 - Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
 - Can even code one implementation that takes a variable queuing object

Thanks! Q&A