Uninformed search

EMI| Semestre 1 | Pr Mohamed RHAZZAF

AI Agent

An **autonomous** system that **senses** the **environment** through **sensors**, makes **decisions** based on its **internal model** or algorithms, and **acts** through **actuators** to **maximize** its performance according to a **defined objective**.

The Agent–Environment Interaction

- An agent operates inside an **environment**. The loop is:

  - **Perception:** The agent receives information from the environment.
    Examples: camera images, sensor readings, text input, game state.
  - **Decision-making:** The agent processes this information using:
    - rules,
    - logic,
    - machine learning models,
    - optimization algorithms.
  - **Action:** The agent performs an action that changes the environment.
    Examples: moving a robot arm, choosing a game move, answering a question.
  - **Feedback:** The environment responds (new state, reward, etc.).

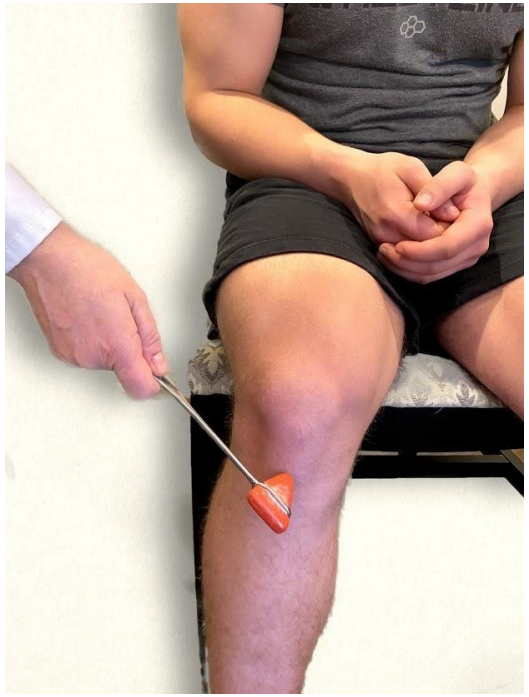
Key Properties of Agents

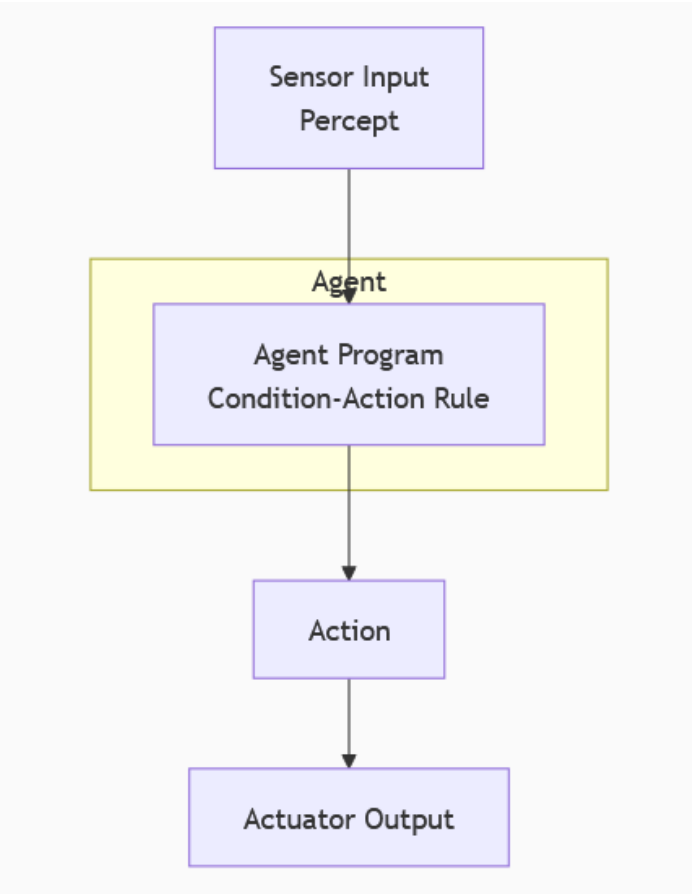
- **Autonomy:** Can operate without human intervention.
- **Reactivity:** Responds to changes in the environment.
- **Proactiveness:** Takes initiative to achieve goals.
- **Social ability:** Can interact with other agents or humans (optional).

Reflex Agent

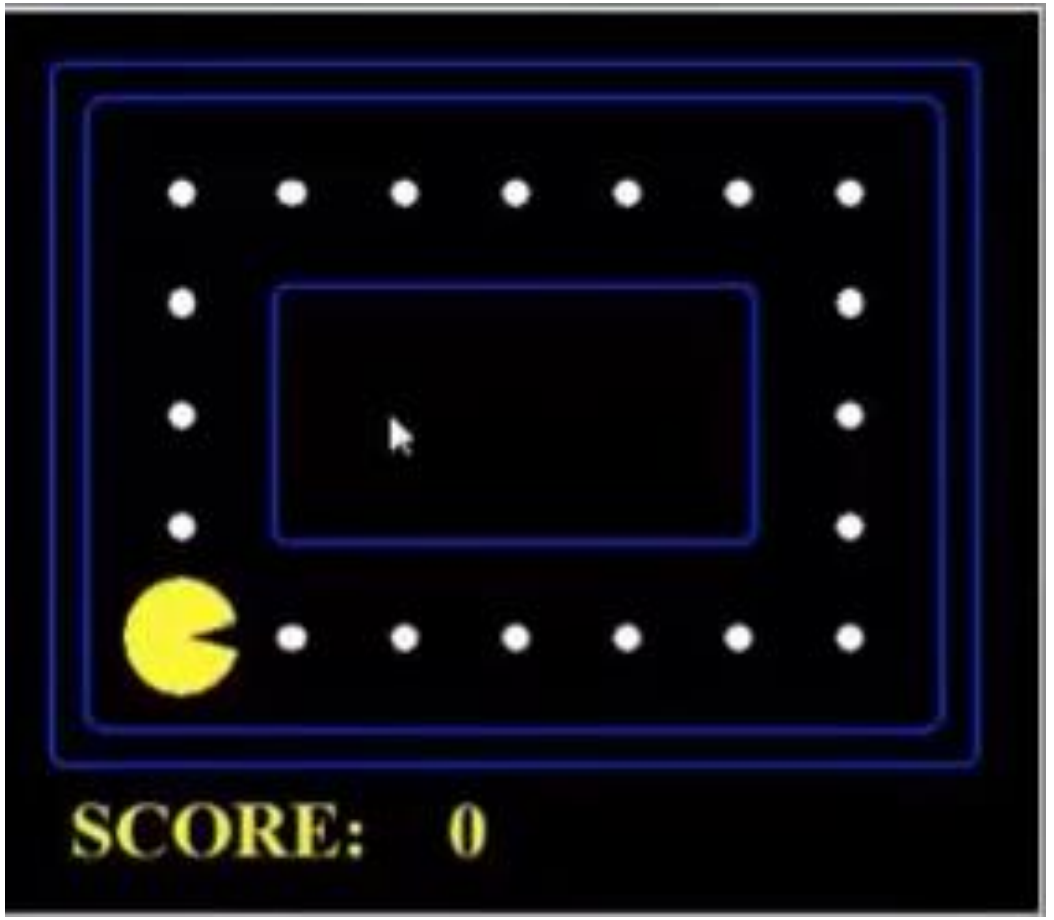
- Reflex agents:
  - Choose action based on current percept
  - Do not consider the future consequences of their actions

“percept something, do something.“

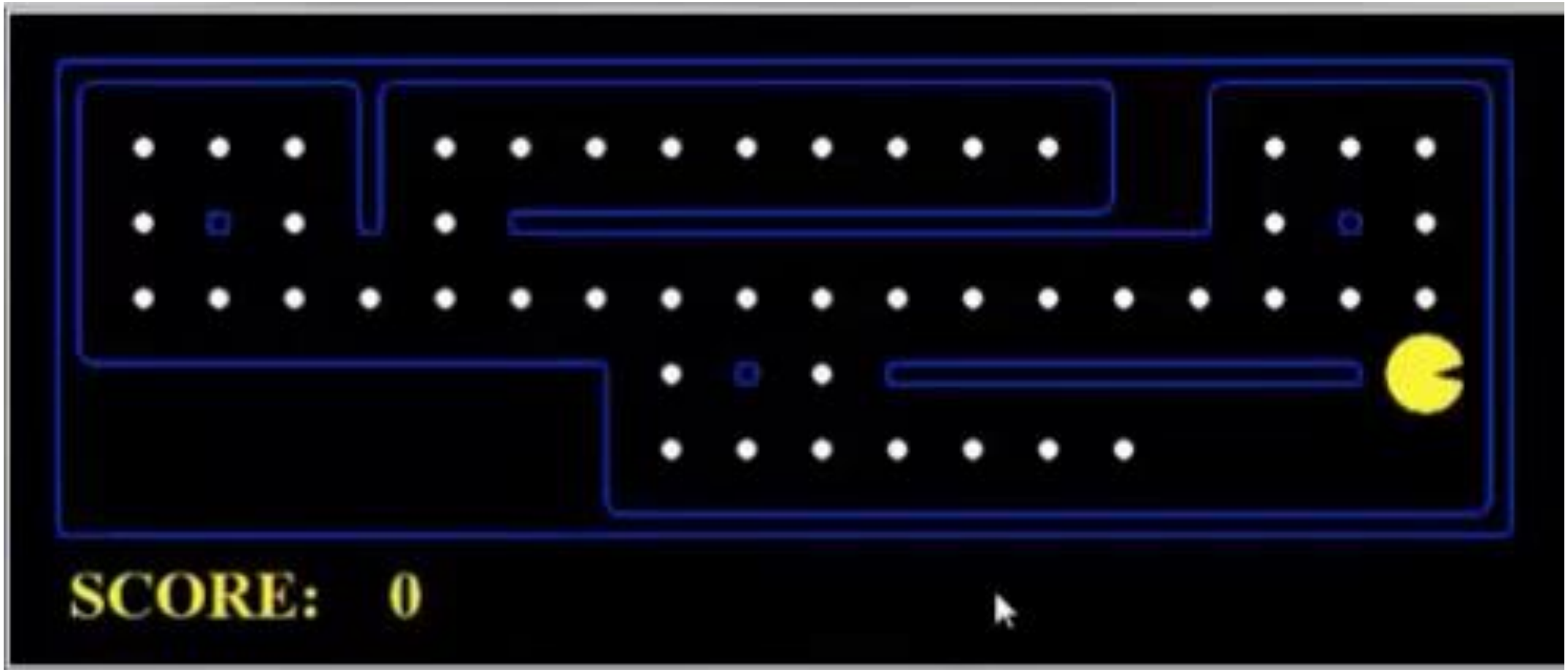




Reflex Agent: Example PacMan Optimal situation



Reflex Agent: Example PacMan not good situation



Reflex Agent

Advantage	Disadvantage
Simple. Very easy to design, program, and understand.	**Limited Intelligence.** Cannot learn from past experiences.
Fast. Very quick to respond, as there's no complex reasoning.	**Inflexible.** Only works in the environments it was designed for. Fails in new or changing environments.
Effective for specific tasks. Perfect for well-defined, repetitive problems.	**Requires Full Observability.** A simple reflex agent fails if it doesn't have all the information.

Planning Agent

- Planning agents:
  - Careful consideration of options / ability to anticipate the future and plan accordingly
  - 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)

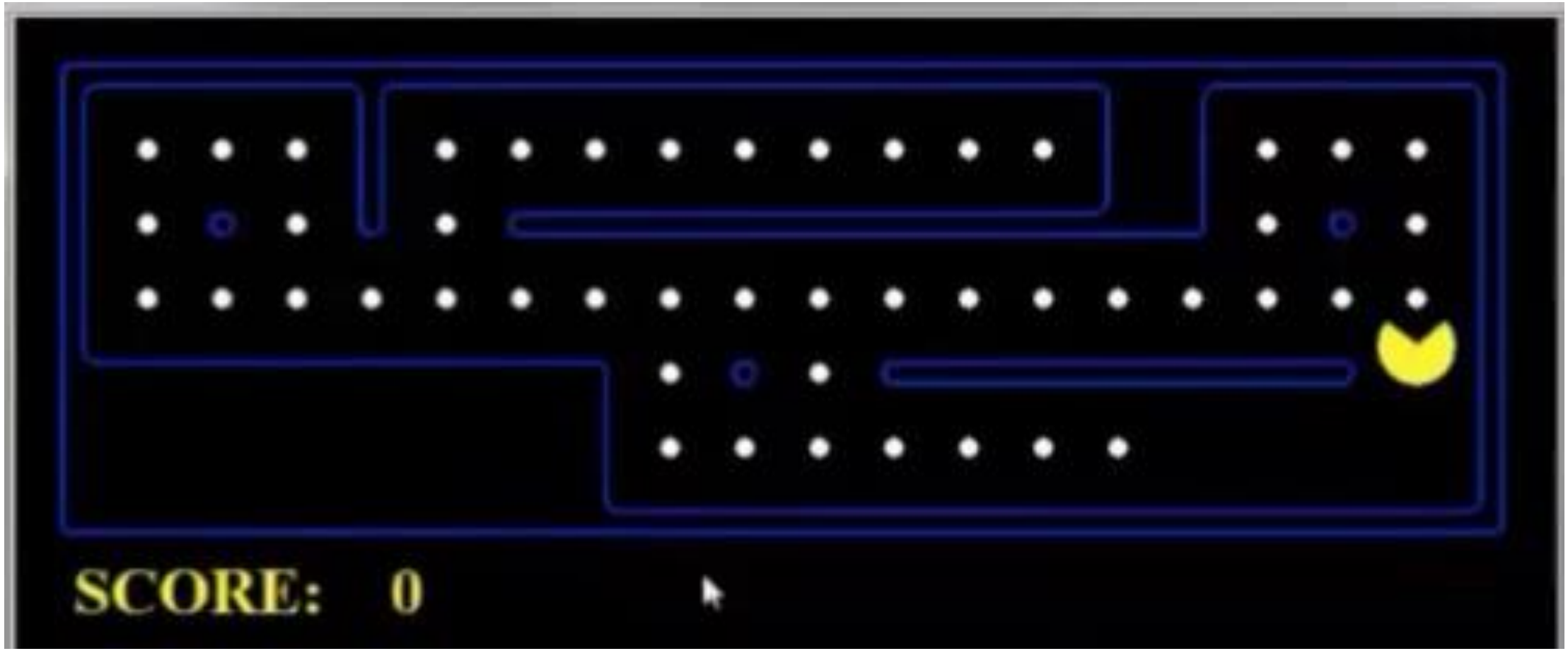
  "Think before you act.“

Optimal vs. complete planning

- Planning a Trip

  A Planning Agent is like a driver who, before even starting the car:
  - Goal: "I need to get to the airport by 5 PM."
  - Plan: open a map (model), considers traffic, construction, and gas stations, and plots a detailed route: "Take Highway 1 North, exit at Main Street, then take the bypass..."
  - Execution: drive that route, checking periodically to make sure you are on track and rerouting if there's an unexpected traffic jam.

Planning Agent: Example PacMan



Planning Agent

Advantage	Disadvantage
Flexible & Intelligent: Can achieve complex, multi-step goals in a structured world.	**Computationally Expensive:** Thinking and searching for a plan takes time and resources. It can be slow.
Robust: Can handle unexpected changes by re-planning.	**Requires a Good Model:** The agent is only as good as the model it's given. If the model is inaccurate, the plan will fail.
Purposeful: Actions are part of a coherent strategy to achieve a goal.	**Struggles with Dynamic Environments:** If the world changes faster than the agent can plan, it will fail.

Search Problems

Search Problems

- **Definition**

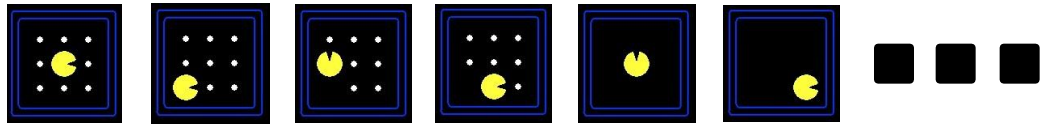
A **search problem** is a formalized problem that involves finding **a sequence of actions** (a path) that leads from an **initial state** to a **goal state**.

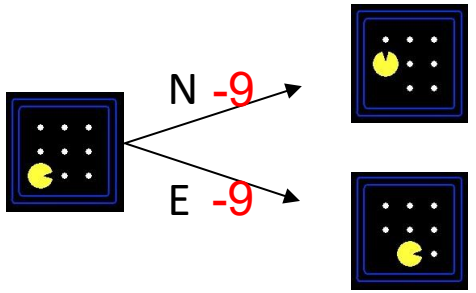
It is defined by a well-defined **state space**, a set of **actions**, a **transition model**, and a **goal test**.

It is the fundamental abstraction behind most problem-solving in AI, where an intelligent agent must consider various possibilities to achieve a desired outcome.

Search Problems

Components of a Search Problem





- Initial State s_0
- State Space (S)
- Actions $(A(s))$
- Transition Model $(Result(s, a))$
- Goal Test
- Path Cost

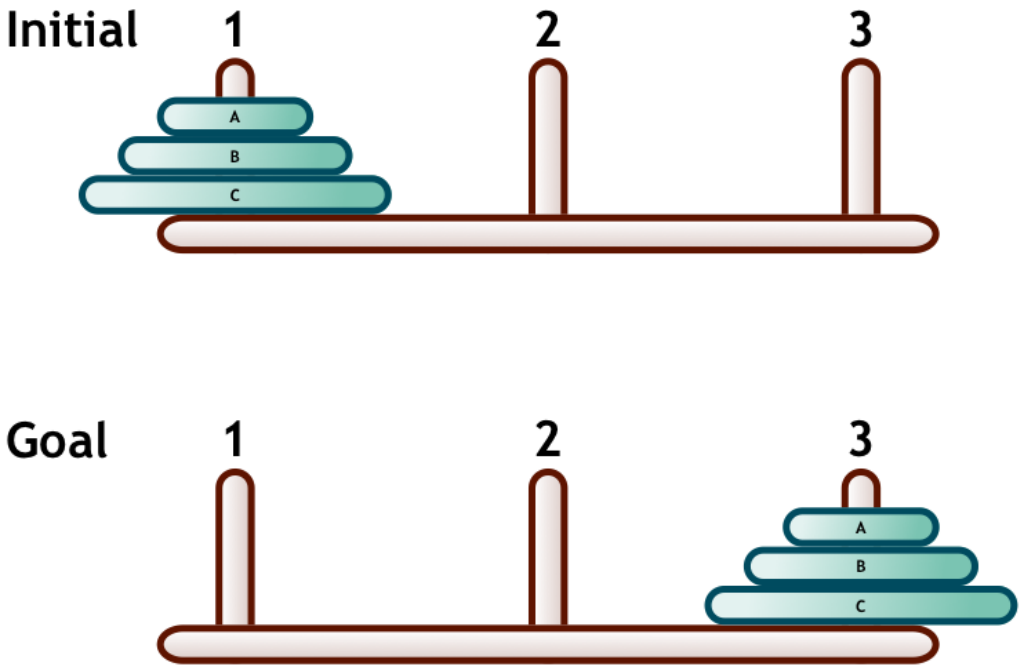
A solution is an action sequence that reaches a goal state
An optimal solution has least cost among all solutions

Examples

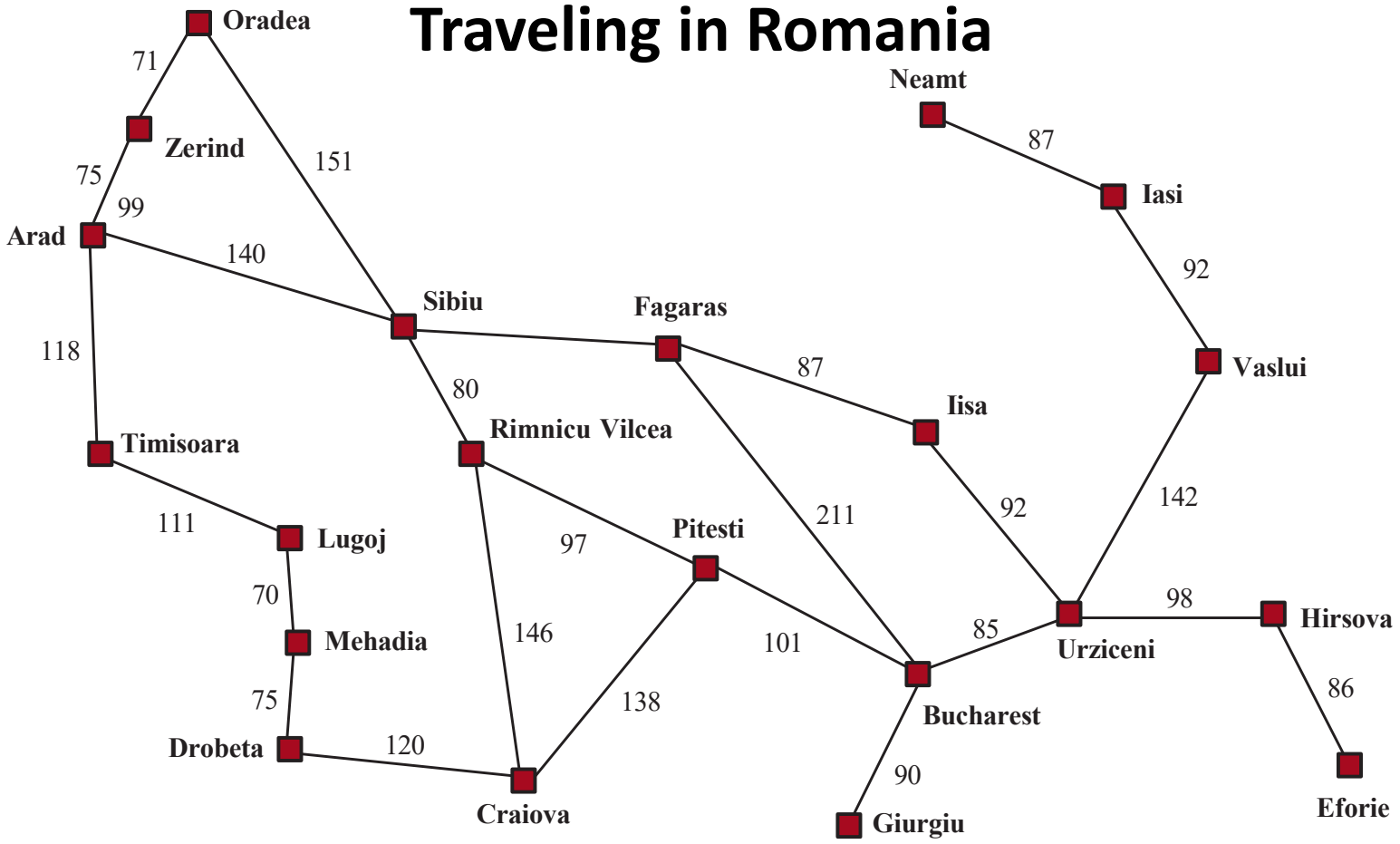
The Wolf, Goat, and Cabbage problem

Examples

Towers of Hanoi

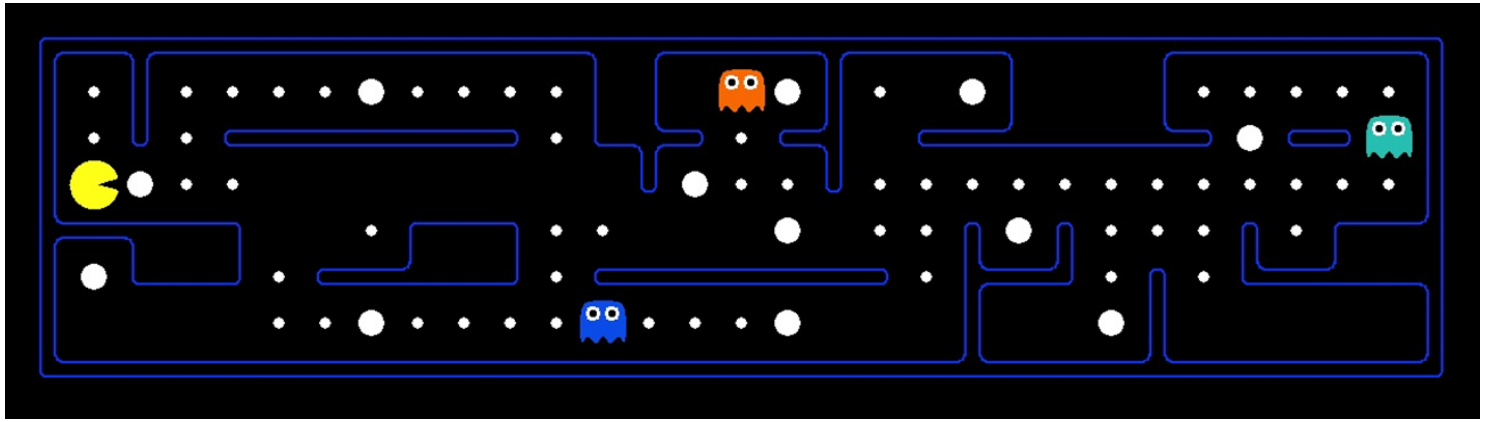


Examples


Traveling in Romania
Oradea
71
Neamt
87
Zerind
151
Iasi
75
99
Arad
140
92
Sibiu
Fagaras
118
87
Vaslui
80
Iisa
Rimnicu Vilcea
Timisoara
142
111
92
211
Lugoj
97
Pitesti
70
98
Hirsova
Mehadia
146
85
101
Urziceni
75
138
86
Bucharest
Drobeta
120
90
Craiova
Giurgiu
Eforie

What's 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), 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?

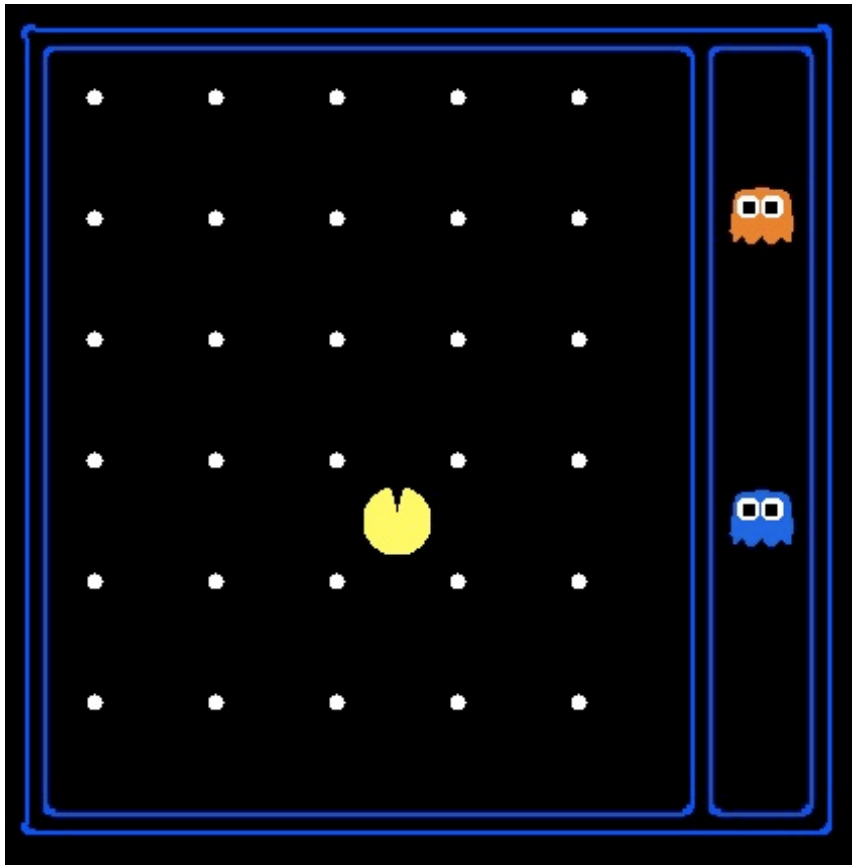
$$4 * 120(2^{30})(12^2)$$

  - States for pathing?

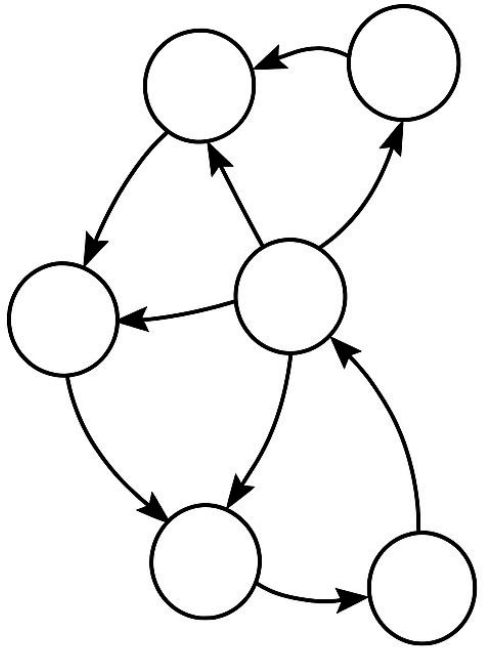
$$120$$

  - States for eat-all-dots?

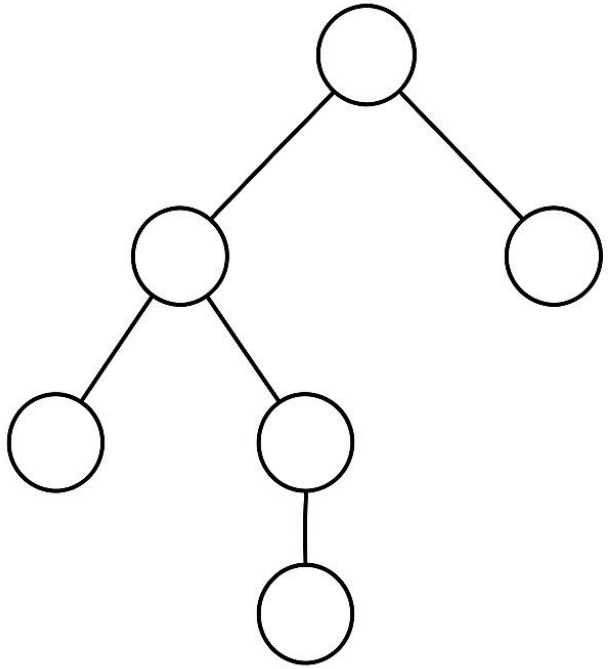
$$120(2^{30})$$



State Space Graphs and Search Trees



State Space Graph

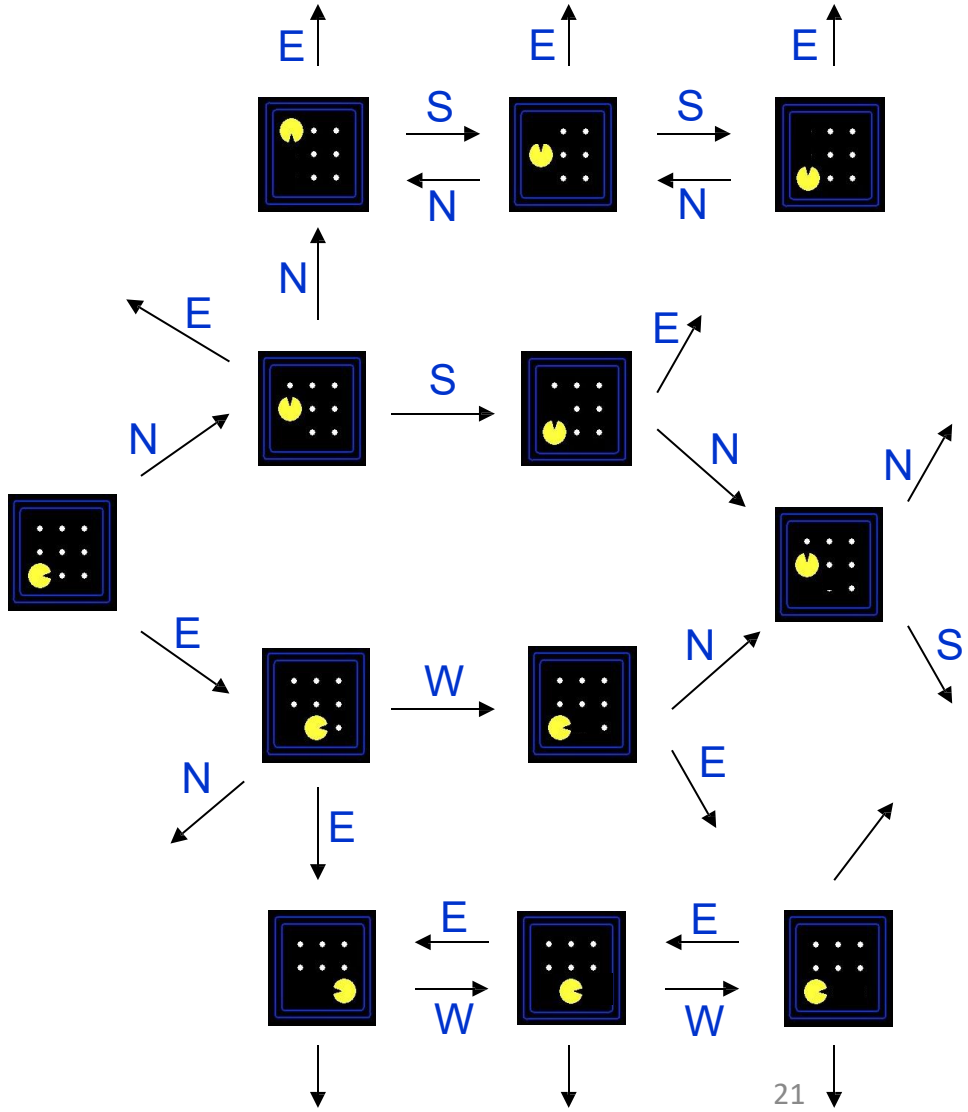


Search Tree

State Space Graphs

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

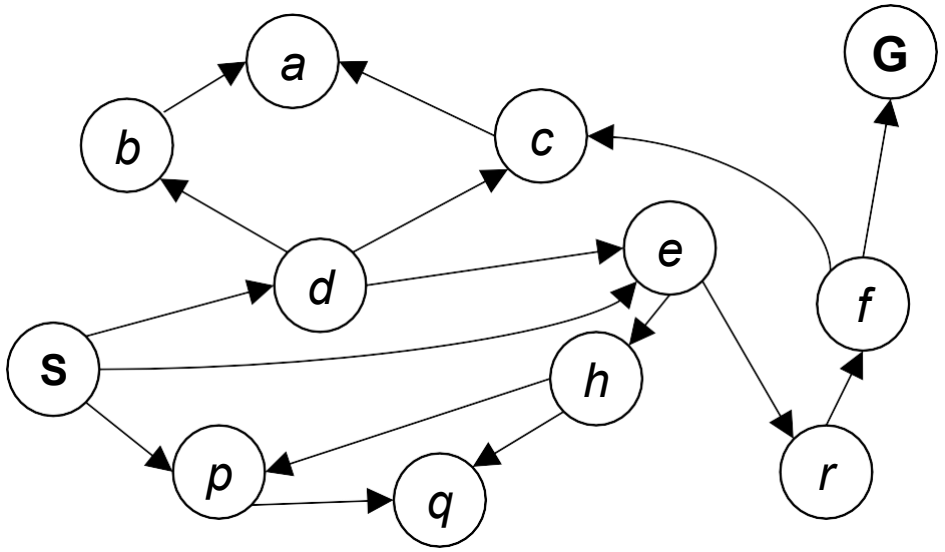
- **In a state space graph, each state occurs only once!**



State Space Graphs

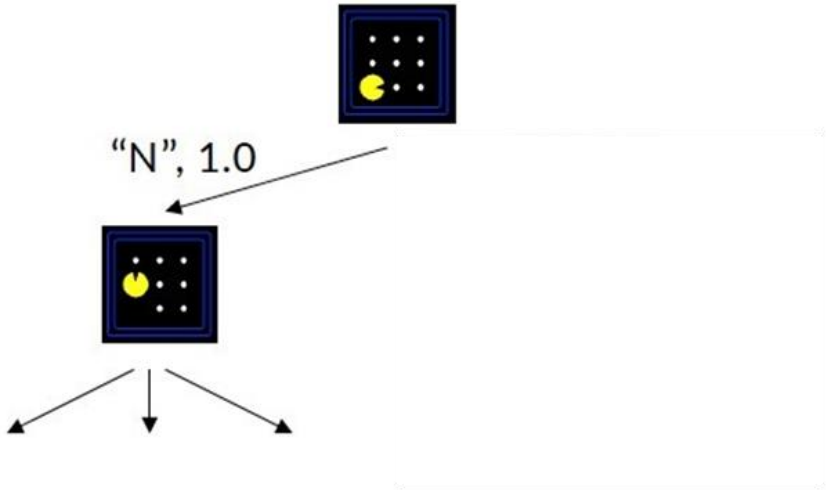
- **State space graph: A mathematical representation of a search problem**
  - Nodes are (abstracted) world configurations
  - Arcs represent transitions (labeled with actions)
  - The goal test is a set of goal nodes (maybe only one)

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



Tiny state space graph for a tiny search problem

Search Trees

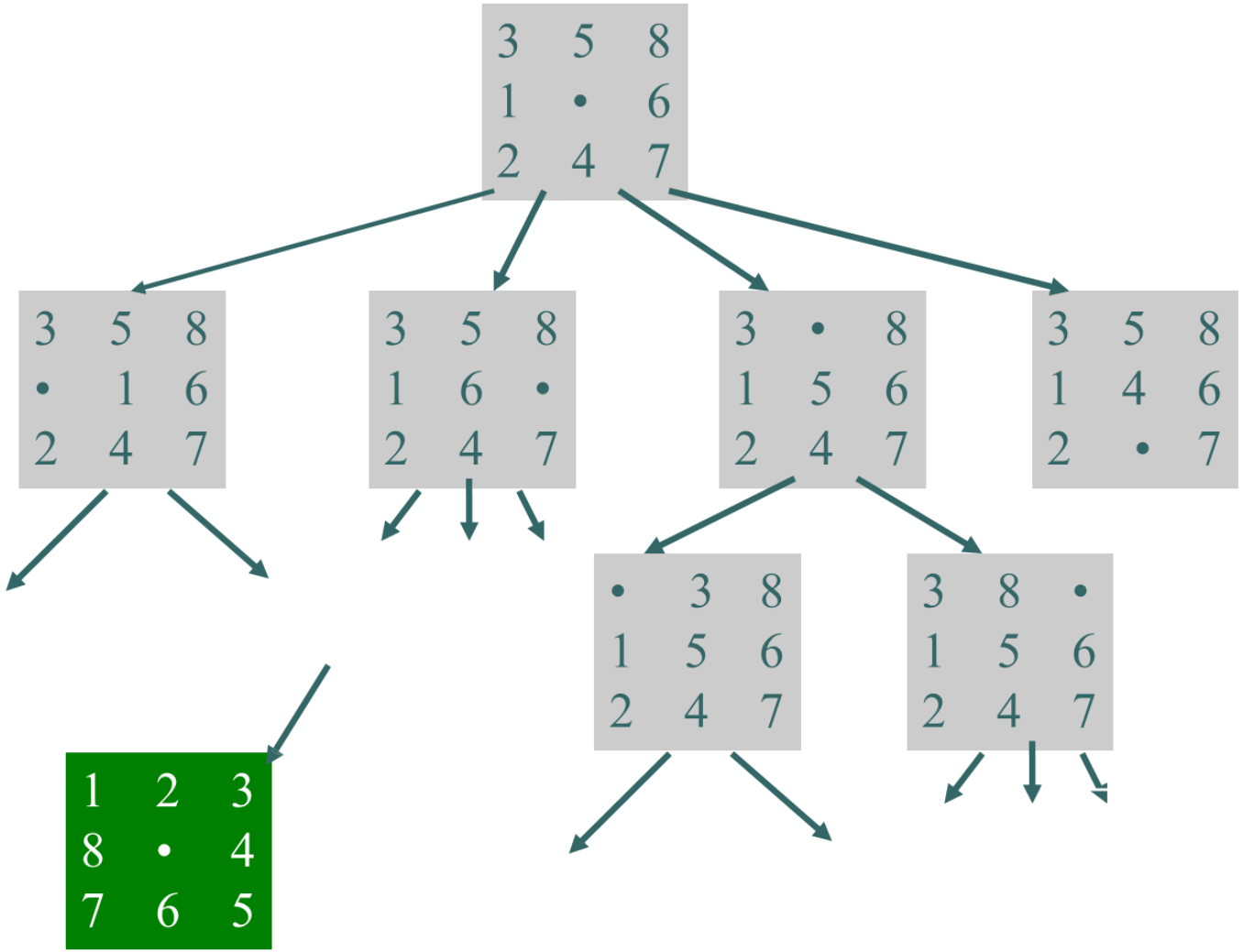


This is now / start

Possible futures

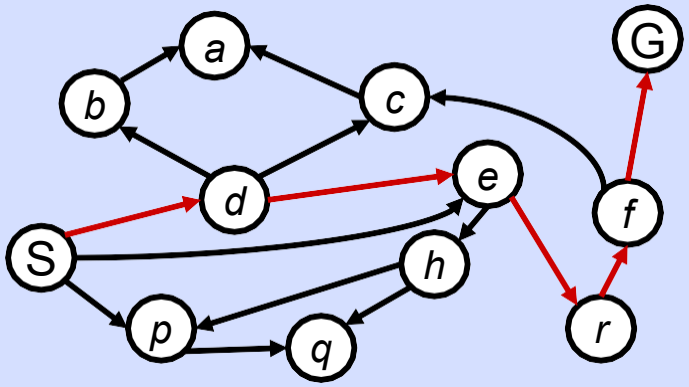
- A search tree:
  - A "what if" tree of plans and their outcomes
  - The start state is the root node
  - Children correspond to successors
  - Nodes show states, but correspond to PLANS that achieve those states
  - For most problems, we can never build the whole tree!

Search Trees



State Space Graphs vs. Search Trees

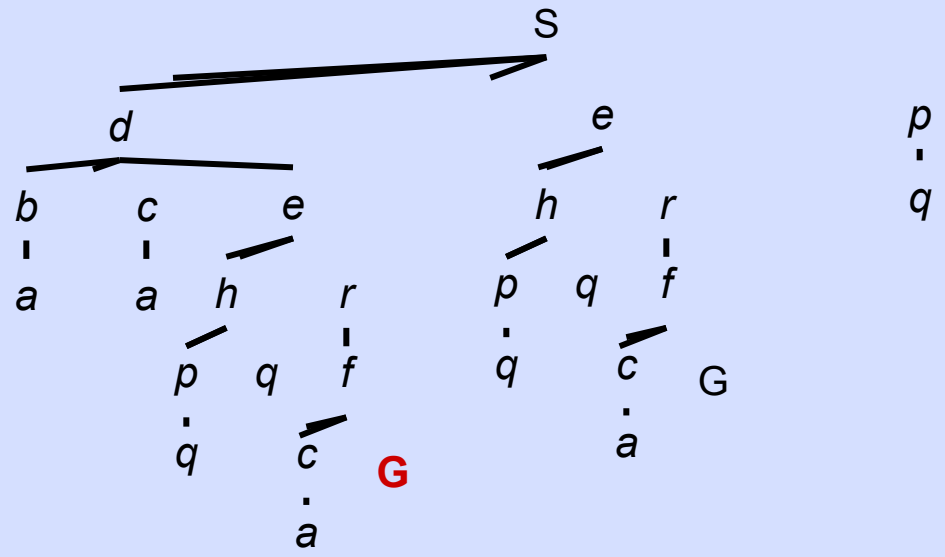
State Space Graph



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

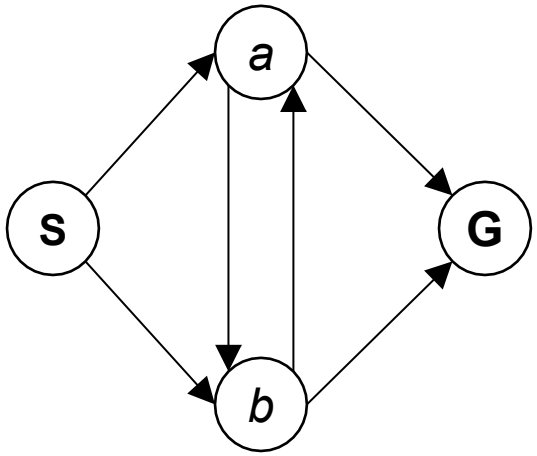
We construct the tree on demand – and we construct as little as possible.

Search Tree

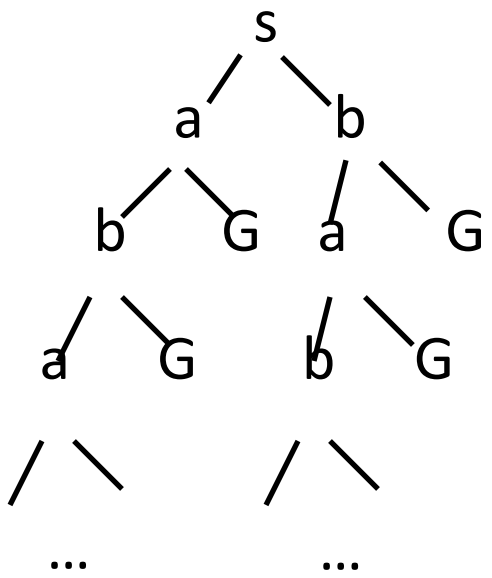


State Space Graphs vs. Search Trees

Consider this 4-state graph:

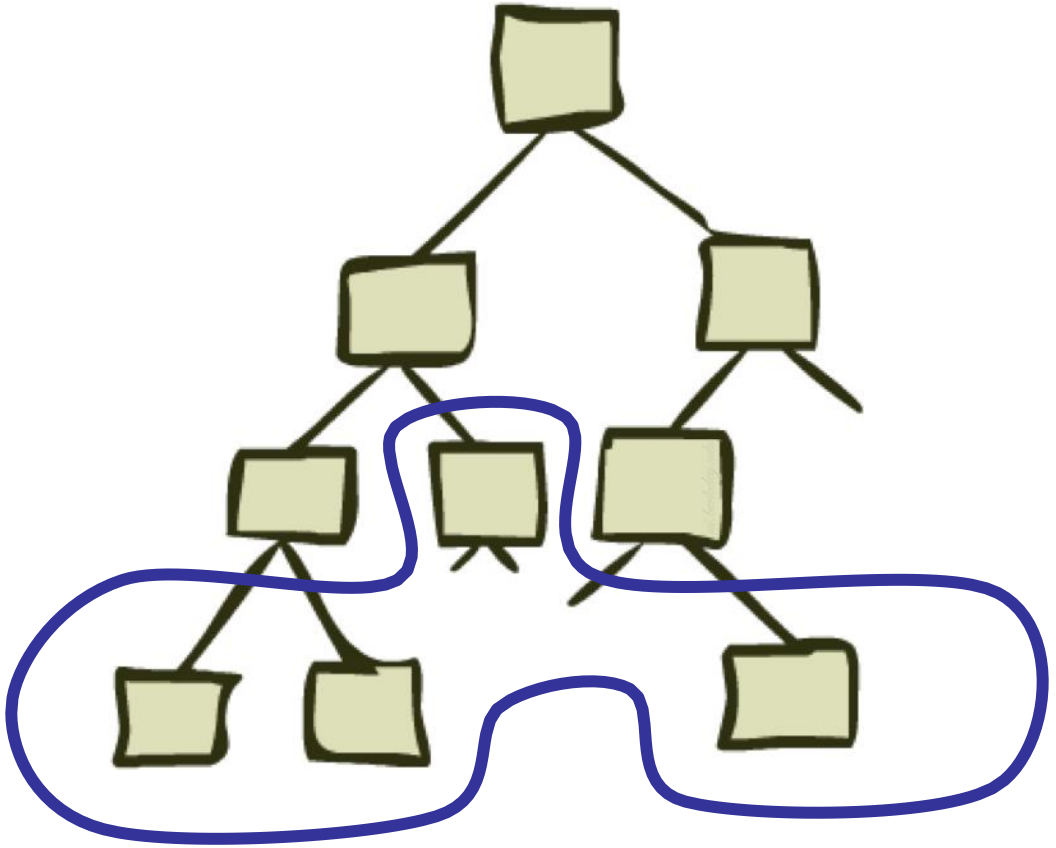


How big is its search tree (from S)?

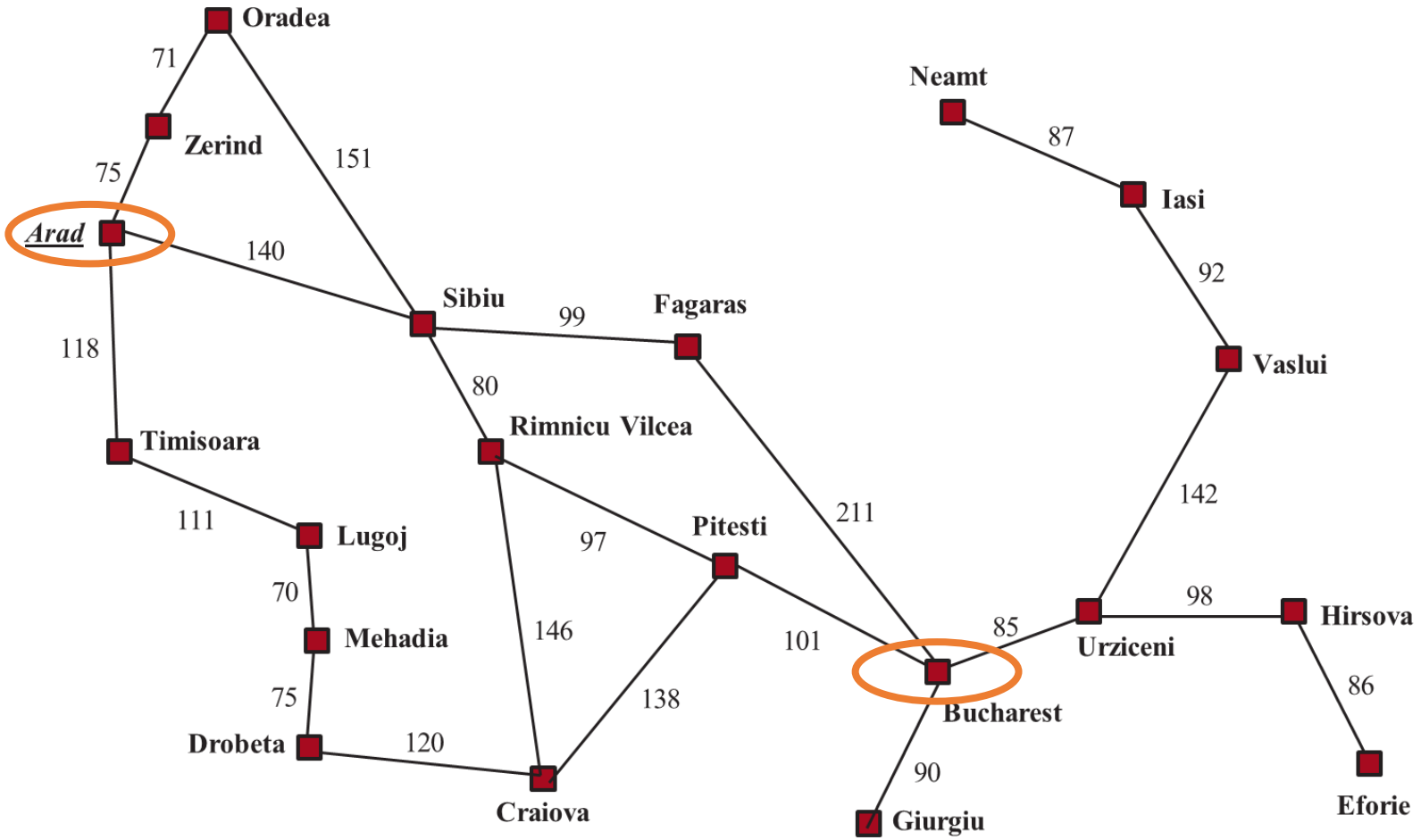


Important: Those who don't know history are doomed to repeat it!

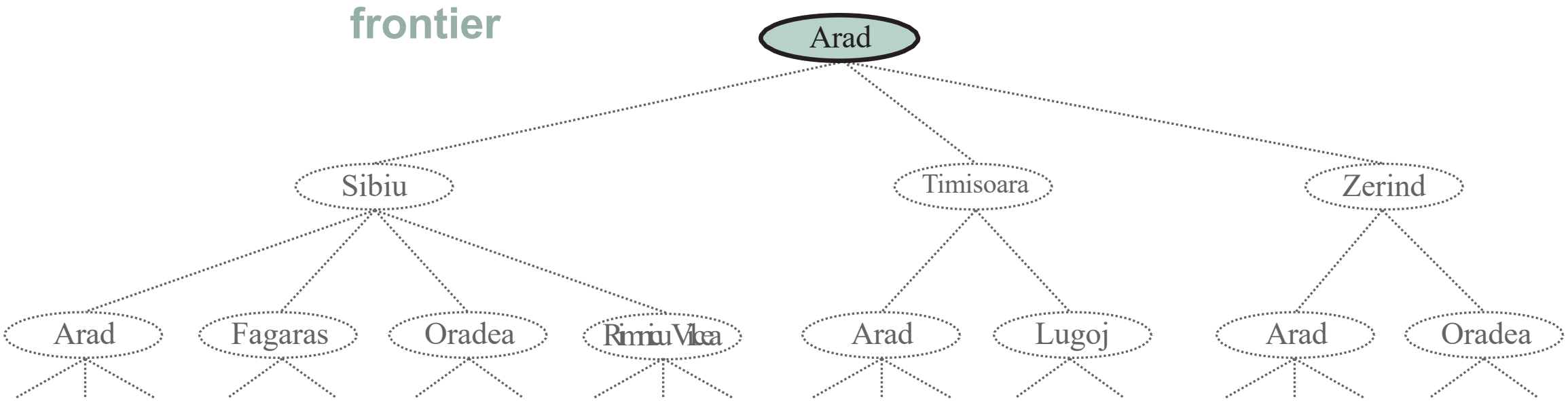
Tree Search



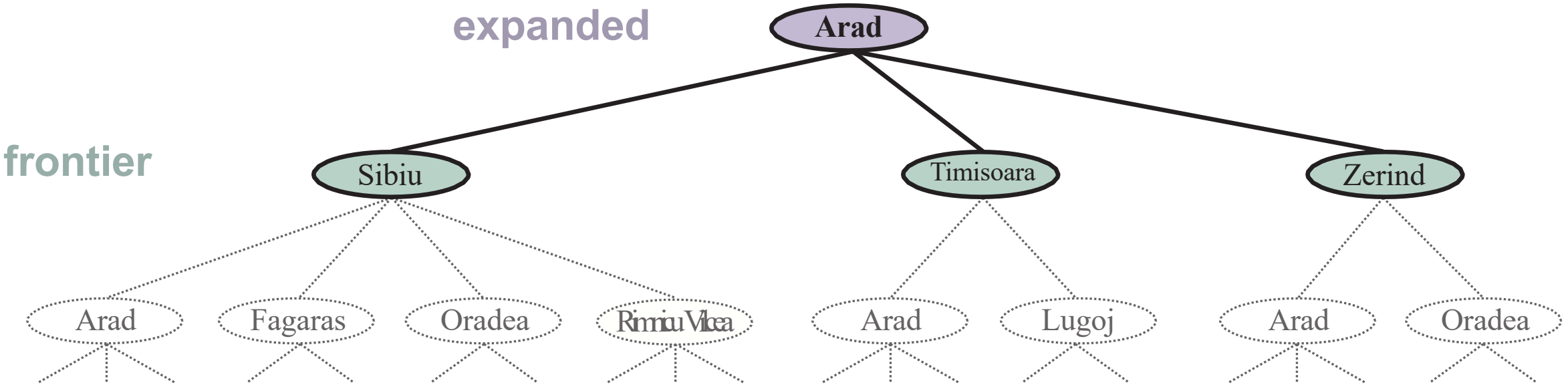
Search Example: Romania



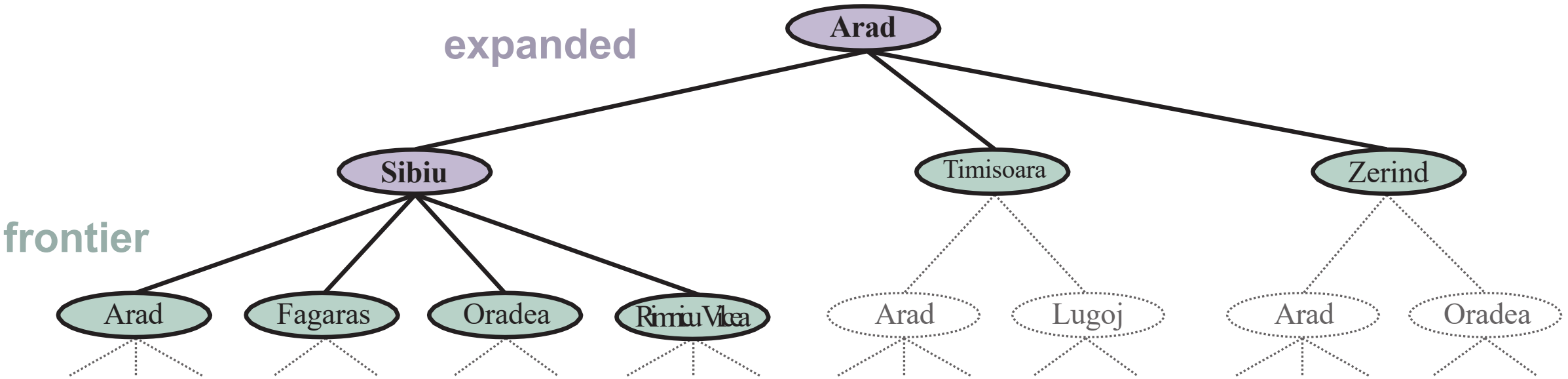
Creating the search tree



Creating the search tree



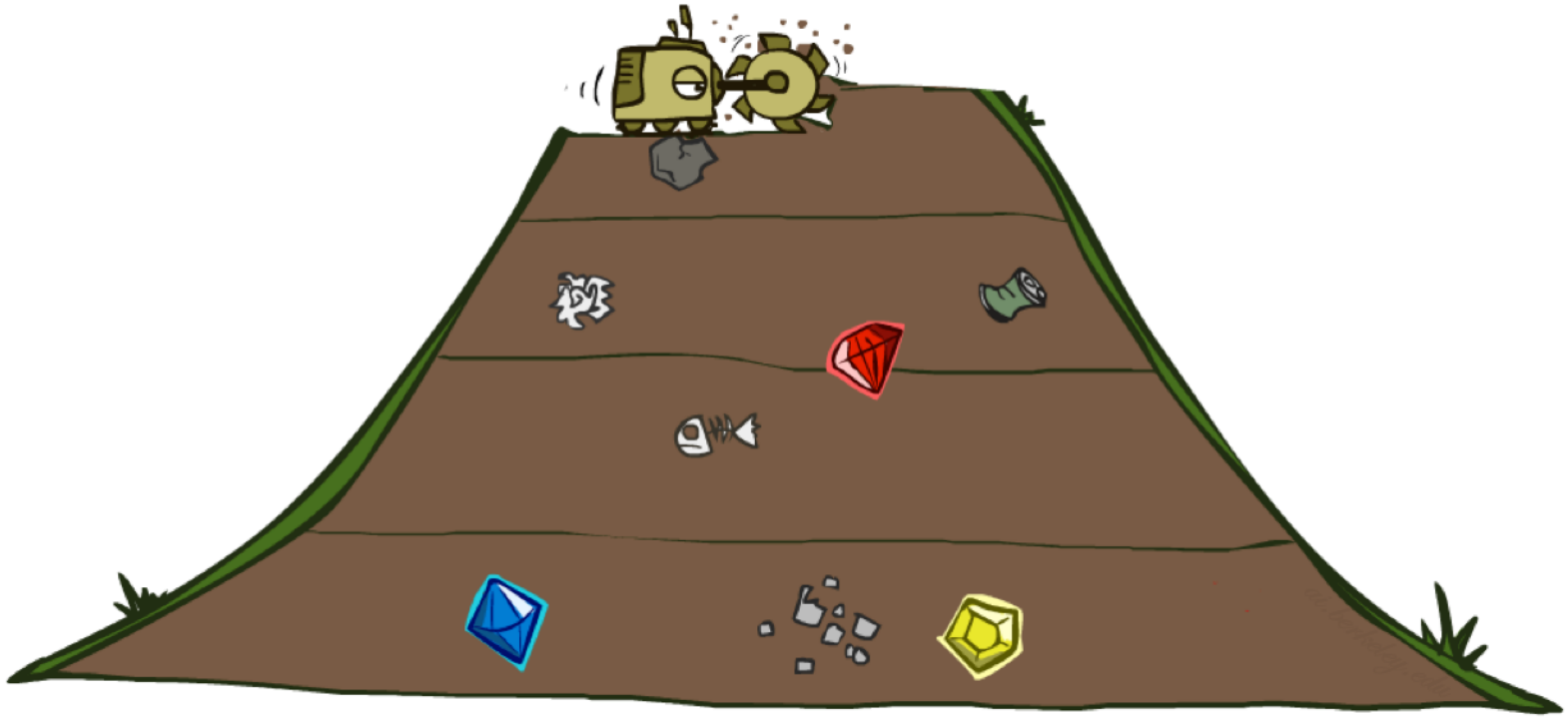
Creating the search tree



Uninformed Search

- **Un-informed Search** (also called **Blind Search**) is a category of algorithms used to traverse or search through a graph or tree structure for a problem's solution **without any additional information about the goal** other than its definition.
- The key characteristic is that these algorithms have **no heuristic**—no way to judge whether one node (state) is closer to the goal than another. They can only distinguish between:
  - **A goal node.**
  - **A non-goal node.**

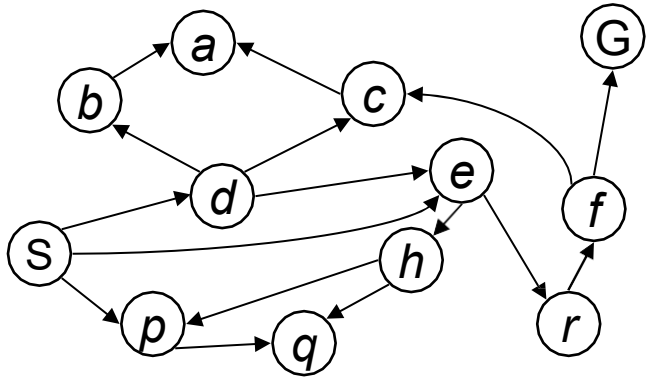
Breadth-First Search

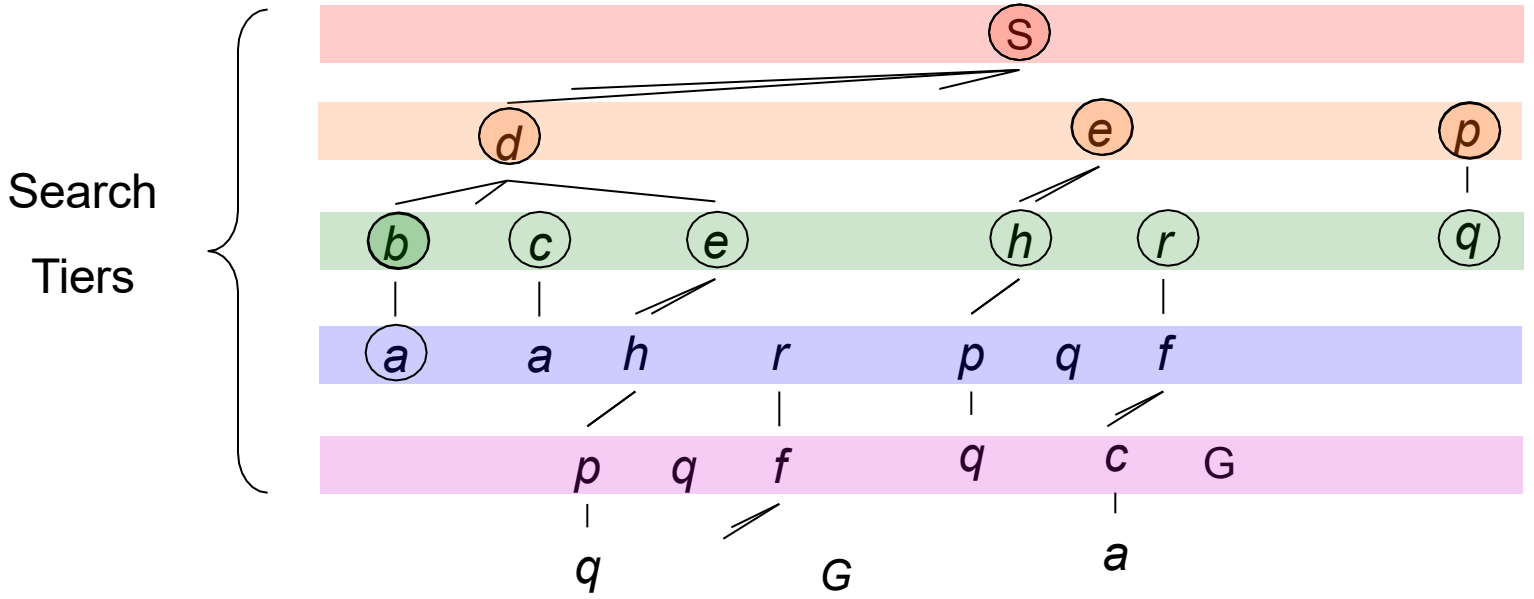


Breadth-First Search

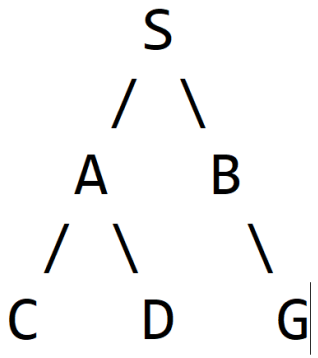
Strategy: expand a shallowest node first

Implementation: Frontier is a FIFO queue





Breadth-First Search

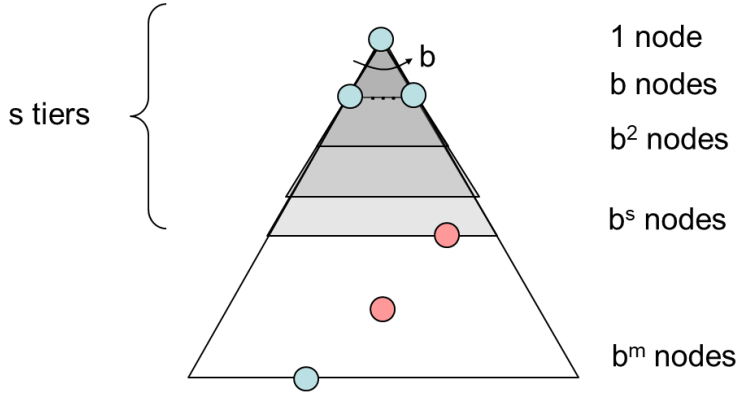


- **Process:**

1. Start with the root node S; add it to the queue. Queue: [S]
2. Dequeue S. Is it the goal? No. Enqueue its neighbors, A and B. Queue: [A, B]
3. Dequeue A. Is it the goal? No. Enqueue its neighbors, C and D. Queue: [B, C, D]
4. Dequeue B. Is it the goal? No. Enqueue its neighbor, G. Queue: [C, D, G]
5. Dequeue C. Not goal. Queue: [D, G]
6. Dequeue D. Not goal. Queue: [G]
7. Dequeue G. This is the goal! Solution found.

- **Path to Goal: S -> B -> G**

- **Completeness:** Yes (if the branching factor b is finite).
- **Optimality:** Yes (if all step costs are identical). It always finds the shallowest (shortest) solution.
- **Time Complexity:** $O(b^d)$ — very bad for deep solutions. (b is branching factor, d is solution depth)
- **Space Complexity:** $O(b^d)$ — keeps every expanded node in memory; the main problem with BFS.



BFS Algo

- **BFS Algorithm with Goal**
  - **Input: Graph G(V, E) with vertices V and edges E, starting node s, goal node g.**
  - **Output: Path from s to g if found, otherwise failure.**

  Create a queue Q.

  Enqueue the path [s] into Q.

  While Q is not empty:

  a. Dequeue the first path from Q, call it path.

  b. Let current = last node in path.

  c. If current = g:

  – Return path (goal found).

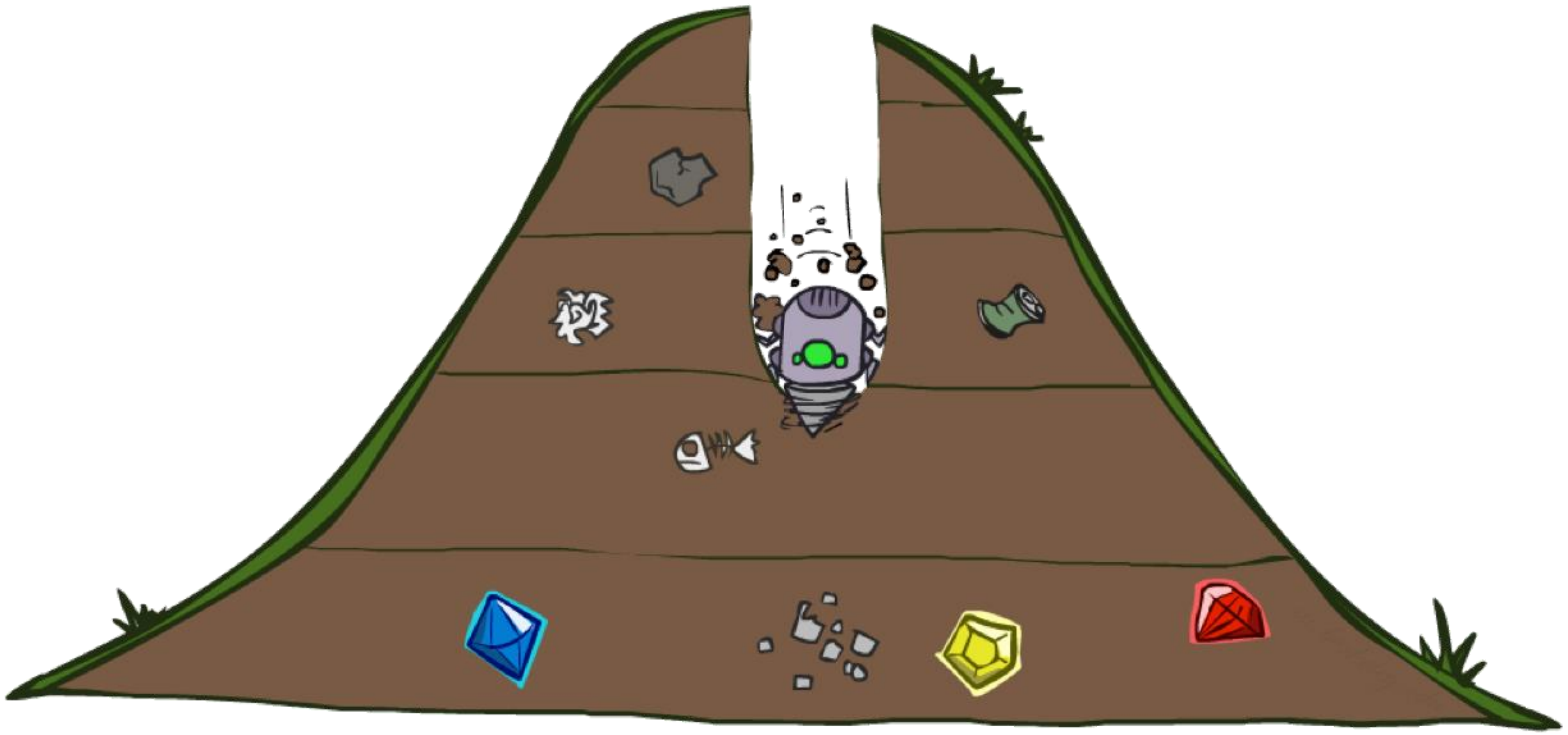
  d. For each neighbor n of current:

  – Create new_path = path + [n].

  – Enqueue new_path into Q.

  If the queue becomes empty, return "Failure: goal not reachable".

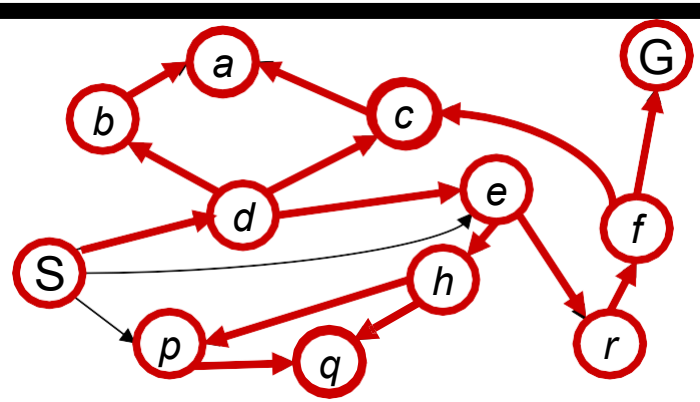
Depth-First Search

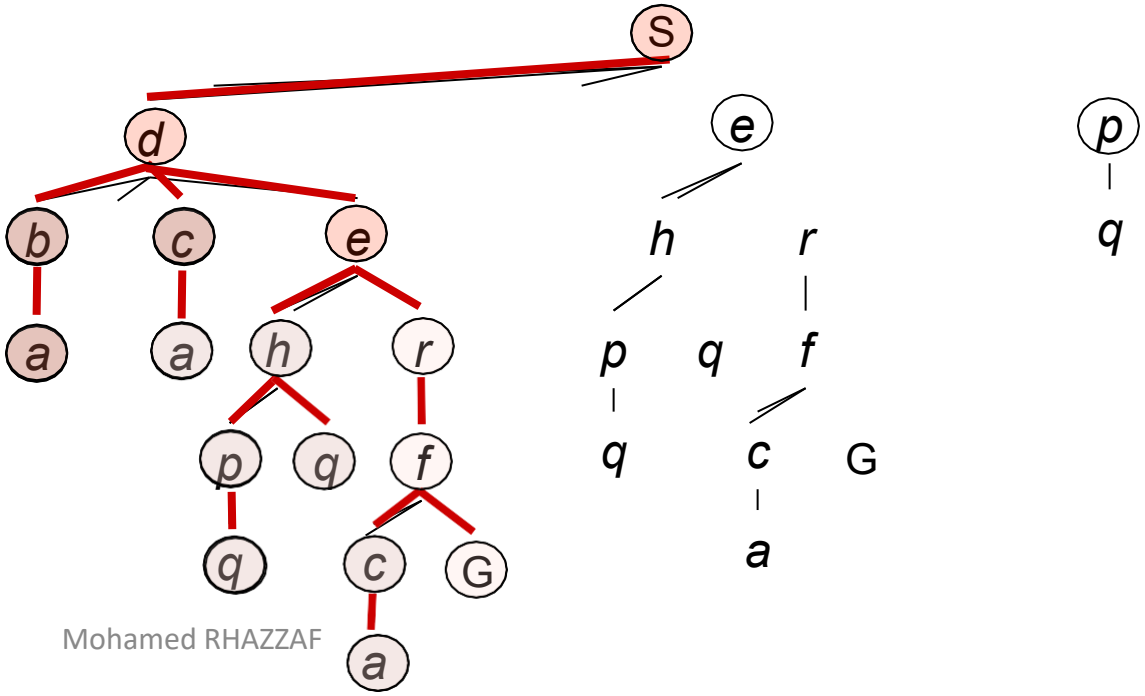


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

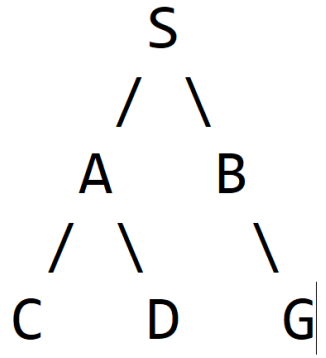




Depth-First Search

- **Process:**

1. Start with the root node S; push it to the stack. Stack: [S]
2. Pop S. Not goal. Push its neighbors, A and B. (The order matters). Stack: [A, B]
3. Pop A. Not goal. Push its neighbors, C and D. (Push D, then C). Stack: [C, D, B]
4. Pop C. Not goal. It has no children. Stack: [D, B]
5. Pop D. Not goal. It has no children. Stack: [B]
6. Pop B. Not goal. Push its neighbor, G. Stack: [G]
7. Pop G. This is the goal! Solution found.



Path to Goal: S -> B -> G

- **Completeness:**

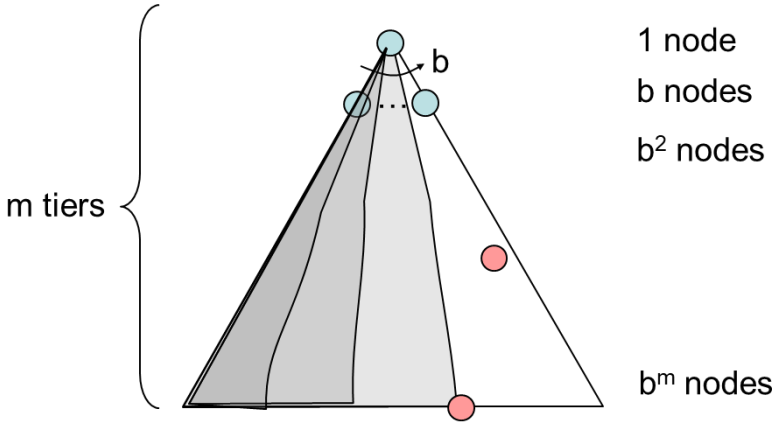
No (can get stuck in infinite loops in infinite state spaces).
Yes, if the search space is finite and cycles are avoided.

- **Optimality:** No (it may find a deeper, more expensive solution before a shallow one).
- **Time Complexity:** $O(b^m)$ — where m is the maximum depth.

Can be better than BFS if the solution is deep.

- **Space Complexity:** $O(bm)$

only needs to store the current path. This is a major advantage over BFS.



DFS Algo

- **DFS Algorithm (with Goal)**
  - **Input: Graph G(V, E) with vertices V and edges E, starting node s, goal node g.**
  - **Output: Path from s to g if found, otherwise failure.**

  Create a stack S.

  Push the path [s] onto S.

  While S is not empty:

  a. Pop the top path from S, call it path.

  b. Let current = last node in path.

  c. If current = g:

  – Return path (goal found).

  d. For each neighbor n of current (in order):

  – Create new_path = path + [n].

  – Push new_path onto S.

  If the stack becomes empty, return "Failure: goal not reachable".

Quiz: DFS vs BFS

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

Quiz: DFS vs BFS

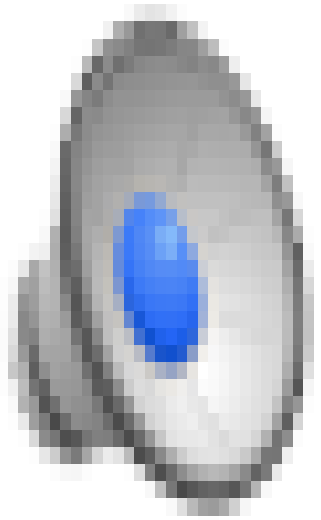
- **When will BFS outperform DFS?**

- The target node (the item you're searching for) is located closer to the starting point in the graph or tree.
- Finding the Shortest Path (Unweighted Graphs)
- When the Graph is Infinitely Deep or Very Deep

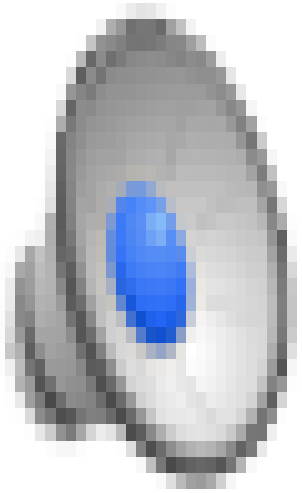
- **When will DFS outperform BFS?**

- the target node is located deep in the graph or tree, or when the graph is extremely wide, making BFS prohibitively expensive in terms of memory.

Demo Maze Water DFS/BFS (part 1)

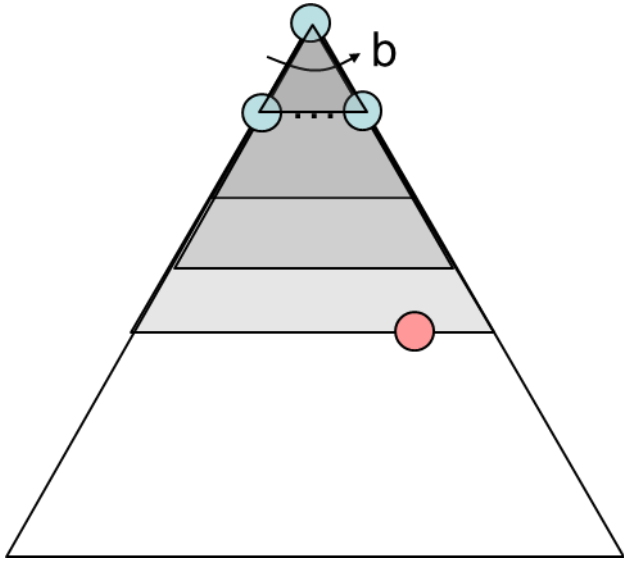


Demo Maze Water DFS/BFS (part 2)

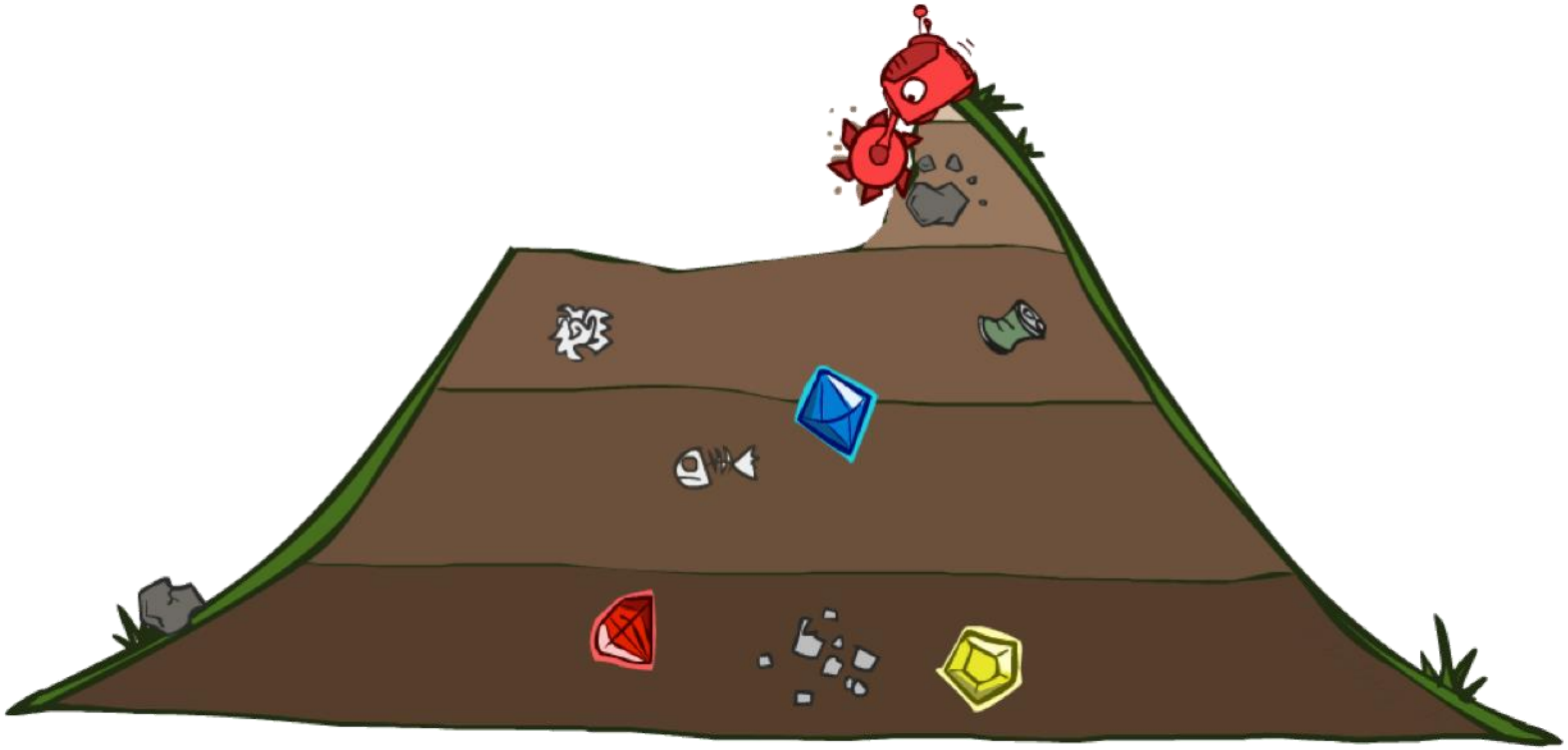


Iterative Deepening

- **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. …..



Uniform Cost Search

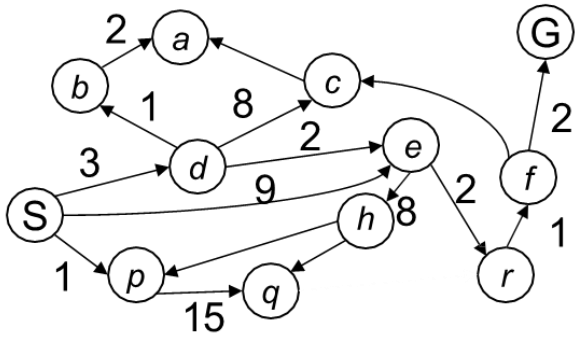


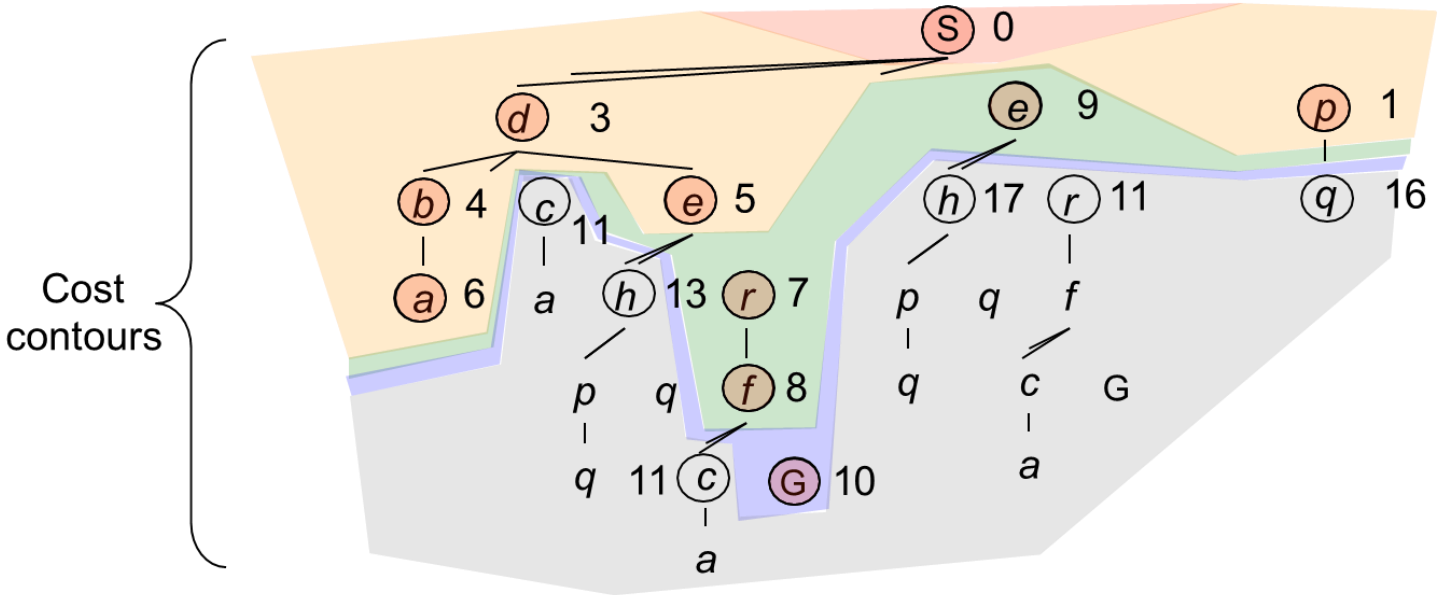
Uniform Cost Search

g(n) = *cost from root to n*

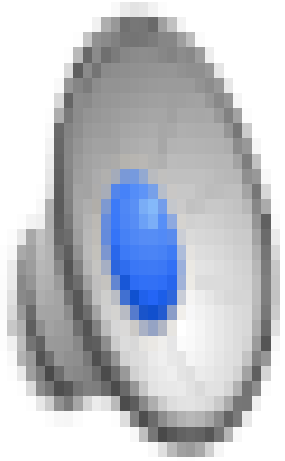
Strategy: expand lowest g(n)

Frontier is a priority queue sorted by g(n)

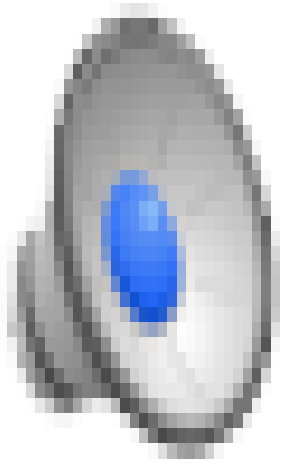




Video of Demo Maze using UCS?



Video of Demo Maze using BFS?



Video of Demo Maze using DFS?