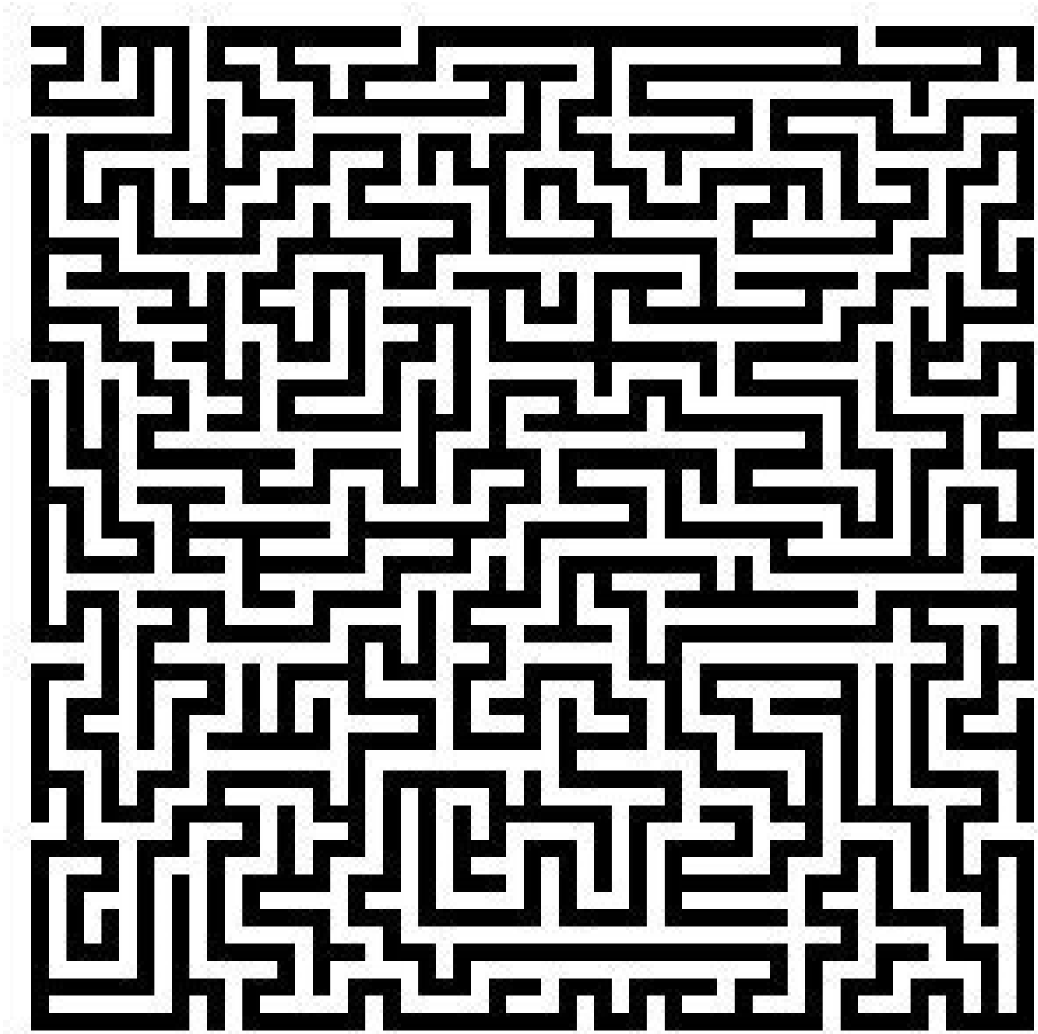
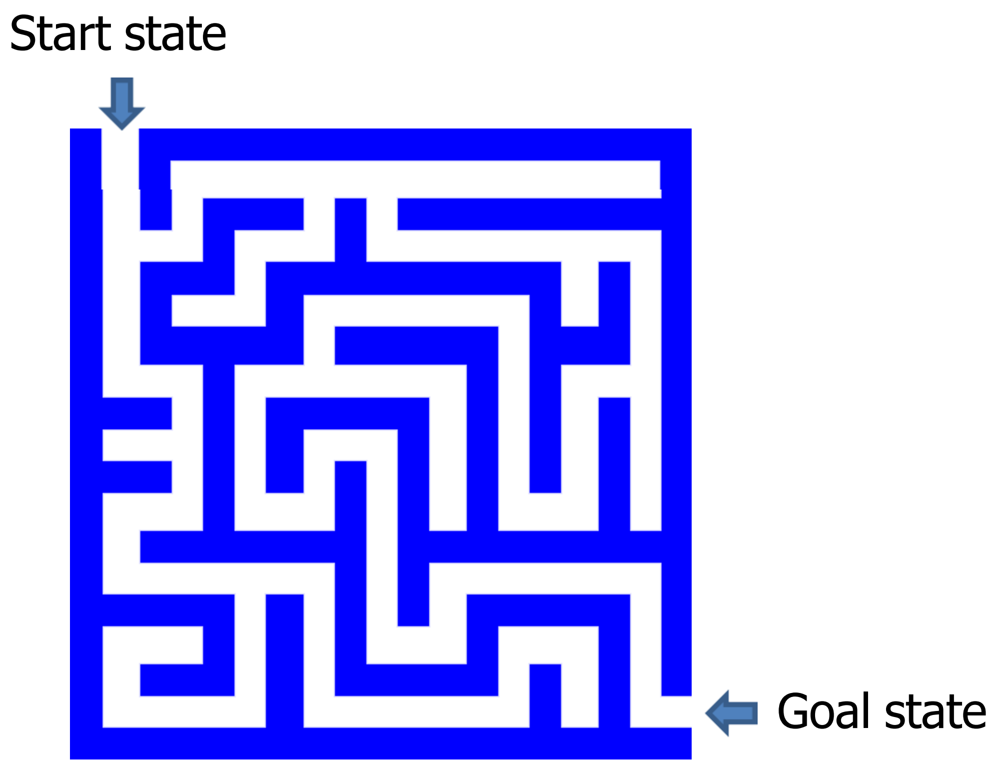
Solving problems by searching

Chapter 3



Search

- Search techniques are universal problem-solving methods
- We will consider the problem of designing **goal-based agents** in **observable, deterministic, discrete, known** environments
- Example:



Search

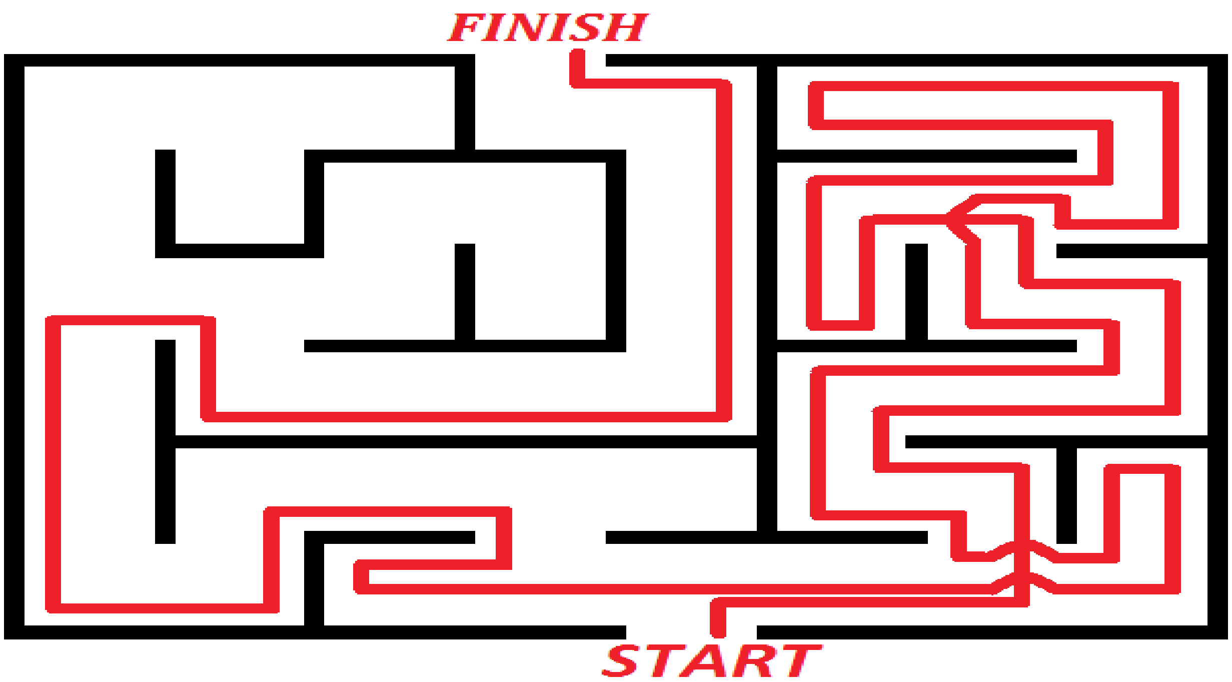
- We will consider the problem of designing **goal-based agents** in **observable**, **deterministic**, **discrete**, **known** environments
  - The solution is a fixed sequence of actions
  - Search is the process of looking for the sequence of actions that reaches the goal
  - Once the agent begins executing the search solution, it can ignore its percepts (**open-loop system**)

To build a system to solve a problem

1. Define the problem precisely
2. Analyze the problem
3. Isolate and represent the task knowledge that is necessary to solve the problem
4. Choose the best problem-solving techniques and apply it to the particular problem.

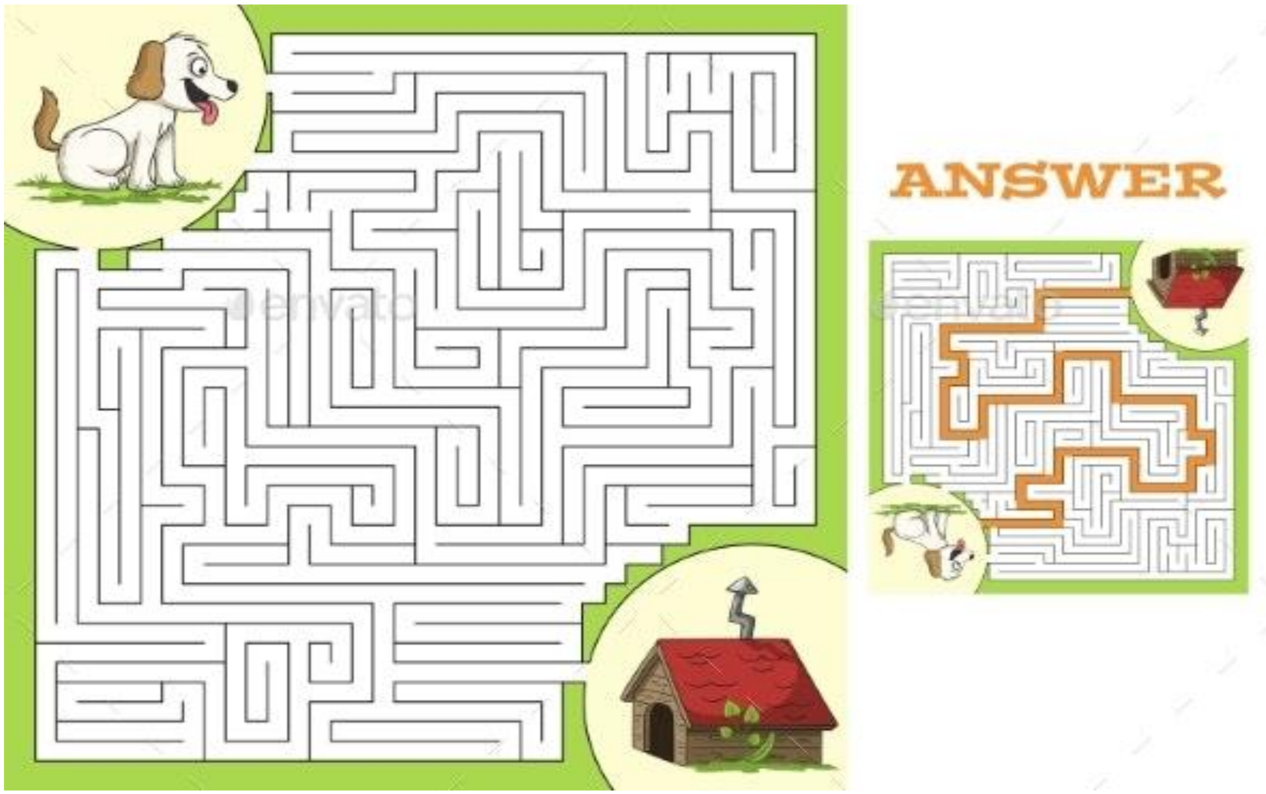
Search Algorithm Terminologies:

- **Search:** Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
  - **Search Space:** Search space represents a set of possible solutions, which a system may have.
  - **Start State:** It is a state from where agent begins **the search**.
  - **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.
- **Search tree:** A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.



Search Algorithm Terminologies:

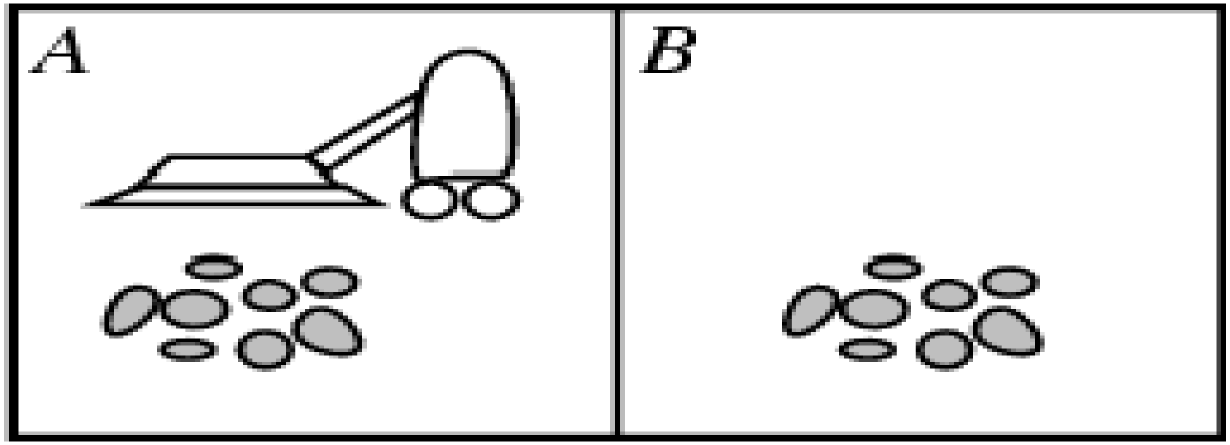
•**Path Cost:** It is a function which assigns a numeric cost to each path.
•**Solution:** It is an action sequence which leads from the start node to the goal node.
•**Optimal Solution:** If a solution has the lowest cost among all solutions.



State space

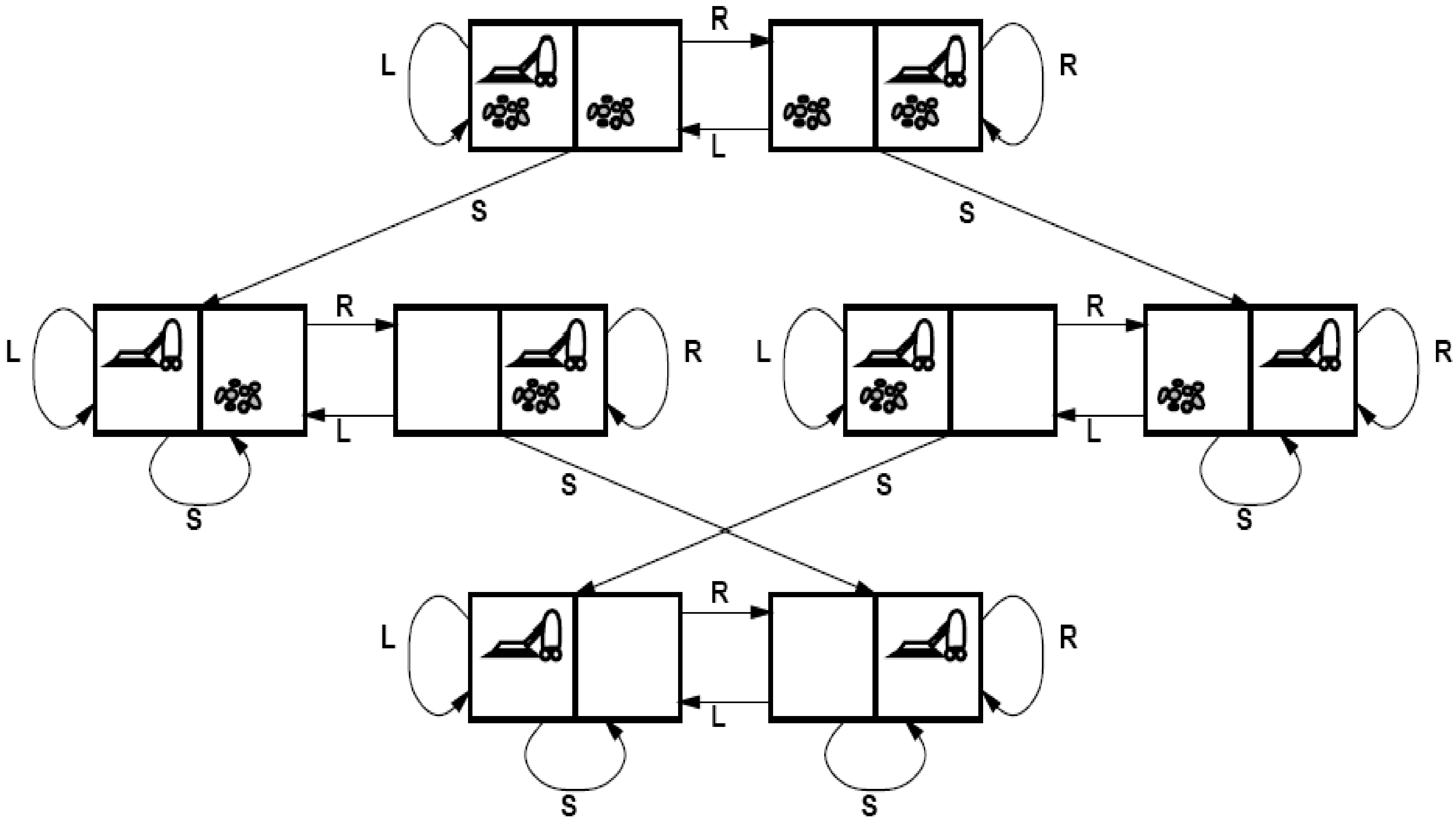
- The initial state, actions, and transition model define the **state space** of the problem
  - The set of all states reachable from initial state by any sequence of actions
  - Can be represented as a **directed graph** where the nodes are states and links between nodes are actions.

Example: Vacuum world



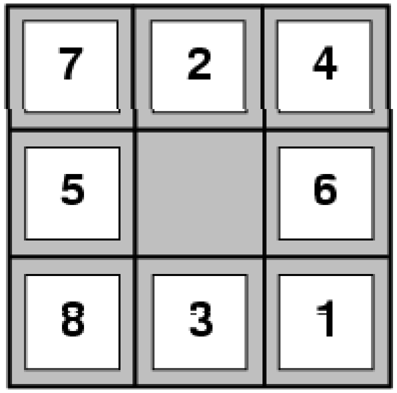
- **States**
  - Agent location and dirt location
  - How many possible states?
  - What if there are *n* possible locations?
- **Actions**
  - Left, right, suck

Vacuum world state space graph

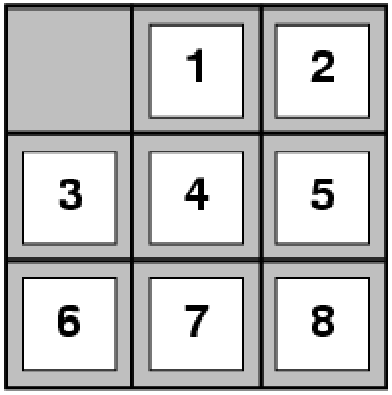


Example: The 8-puzzle

- **States**
  - Locations of tiles
    - 8-puzzle: 181,440 states
    - 15-puzzle: 1.3 trillion states
    - 24-puzzle: 10^{25} states
- **Actions**
  - Move blank left, right, up, down
- **Path cost**
  - 1 per move
- Optimal solution of n-Puzzle is NP-hard

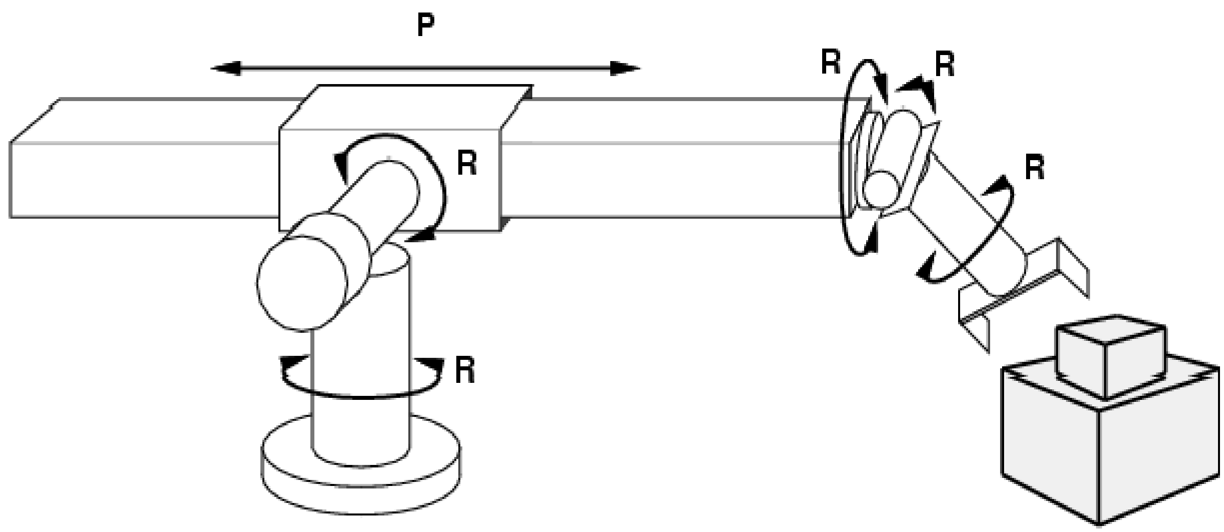


Start State



Goal State

Example: Robot motion planning



- **States**
  - Real-valued coordinates of robot joint angles
- **Actions**
  - Continuous motions of robot joints
- **Goal state**
  - Desired final configuration (e.g., object is grasped)
- **Path cost**
  - Time to execute, smoothness of path, etc.

Search

- Given:
  - **Initial state**
  - **Actions**
  - **Transition model**
  - **Goal state**
  - **Path cost**

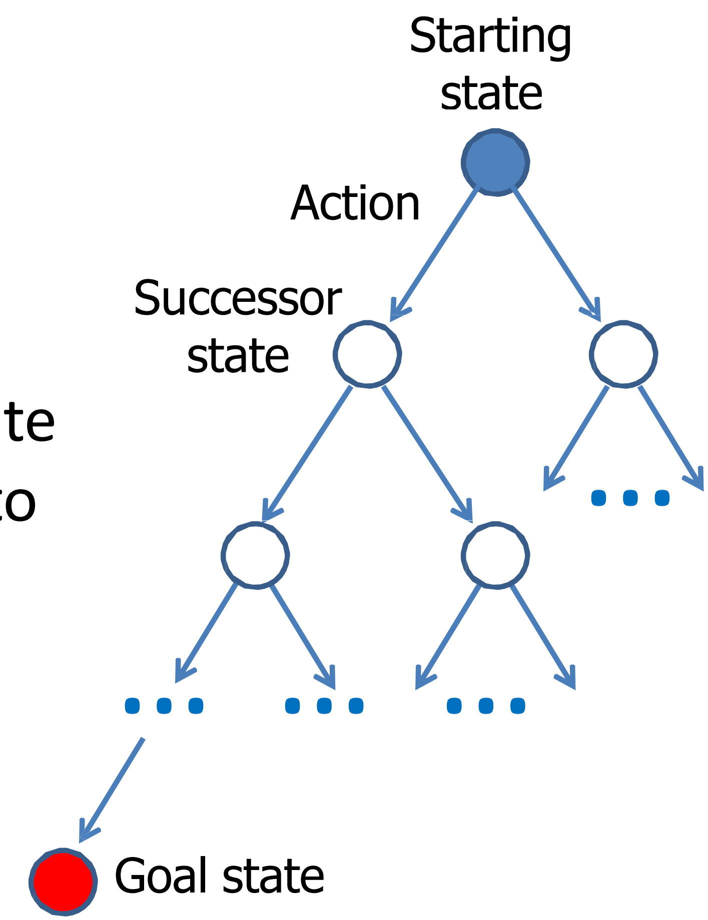
- How do we find the optimal solution?
  - How about building the state space and then using Dijkstra's shortest path algorithm?
    - The state space is huge!
    - Complexity of Dijkstra's is $O(E + V \log V)$, where V is the size of the state space

Tree Search

- Let’s begin at the start node and **expand** it by making a list of all possible successor states
- Maintain a **fringe** or a list of unexpanded states
- At each step, pick a state from the fringe to expand
- Keep going until you reach the goal state
- Try to expand as few states as possible

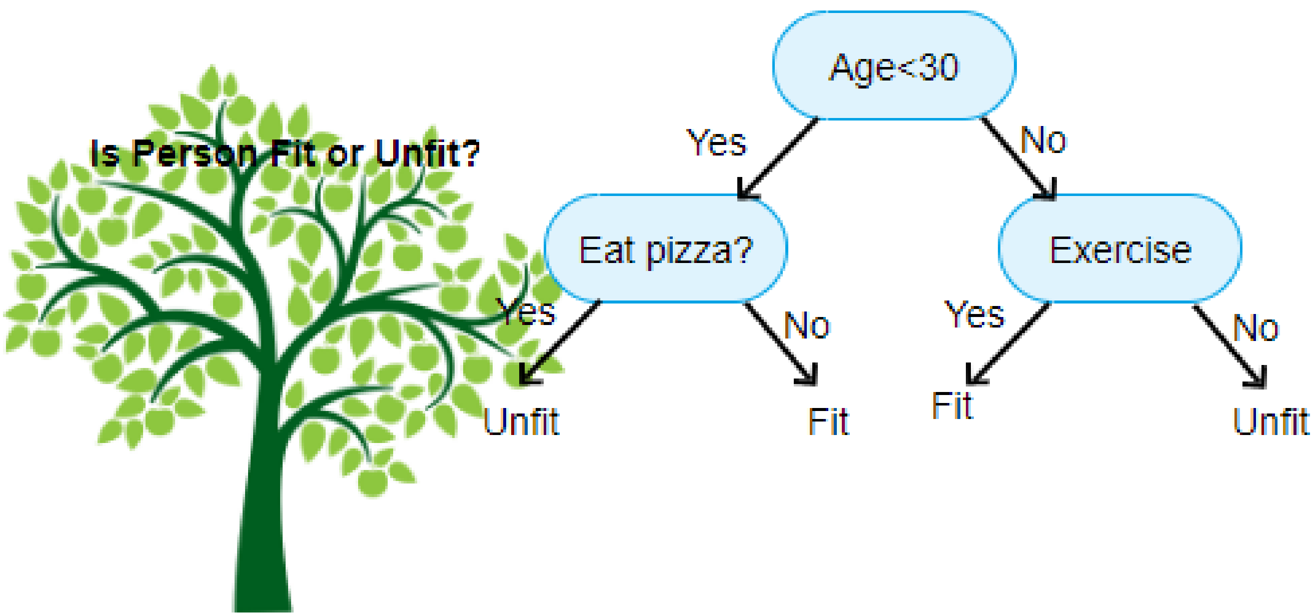
Search tree

- "What if" tree of possible actions and outcomes
- The root node corresponds to the starting state
- The children of a node correspond to the **successor states** of that node's state
- A path through the tree corresponds to a sequence of actions
  - A solution is a path ending in the goal state
- Nodes vs. states
  - A state is a representation of a physical configuration, while a node is a data structure that is part of the search tree

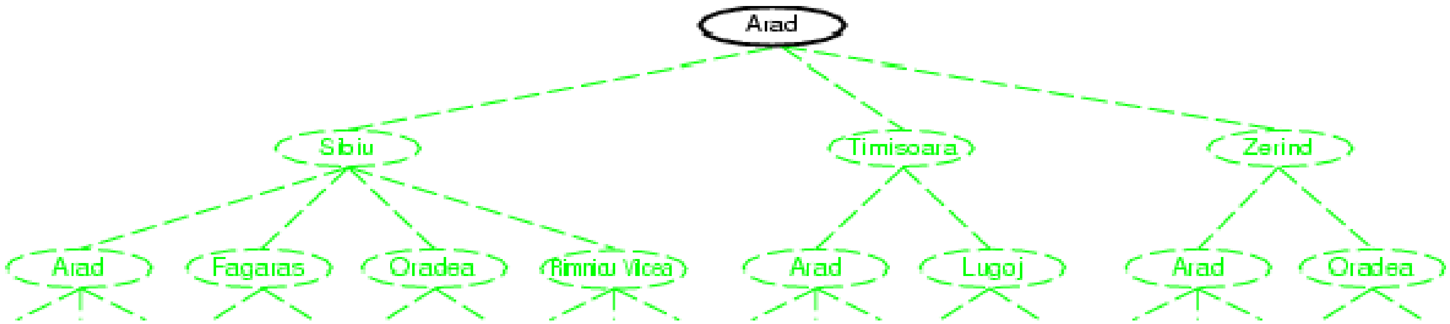


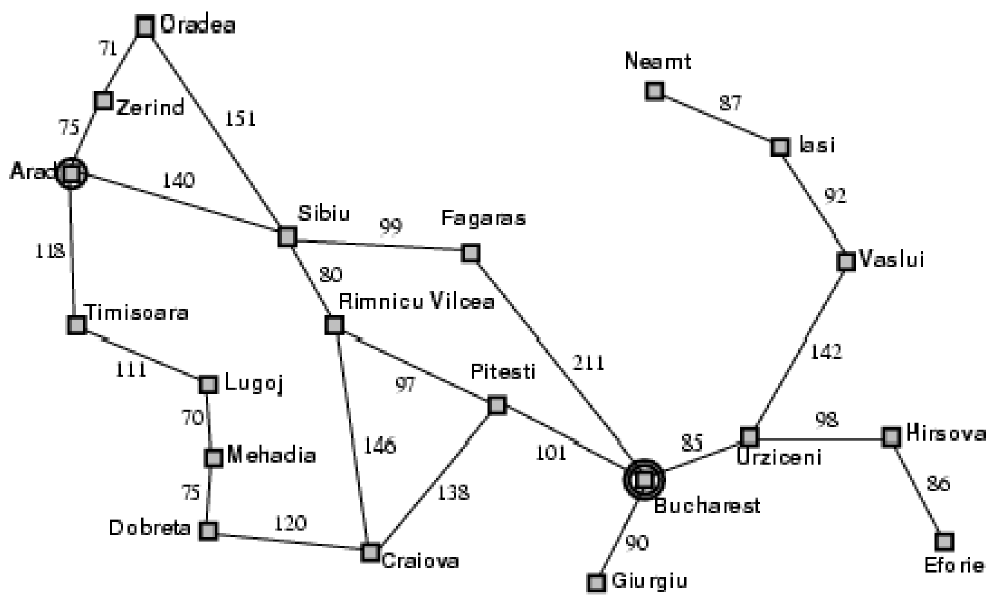
Tree Search Algorithm Outline

- Initialize the **fringe** using the **starting state**
- While the fringe is not empty
  - Choose a fringe node to expand according to **search strategy**
  - If the node contains the **goal state**, return solution
  - Else **expand** the node and add its children to the fringe

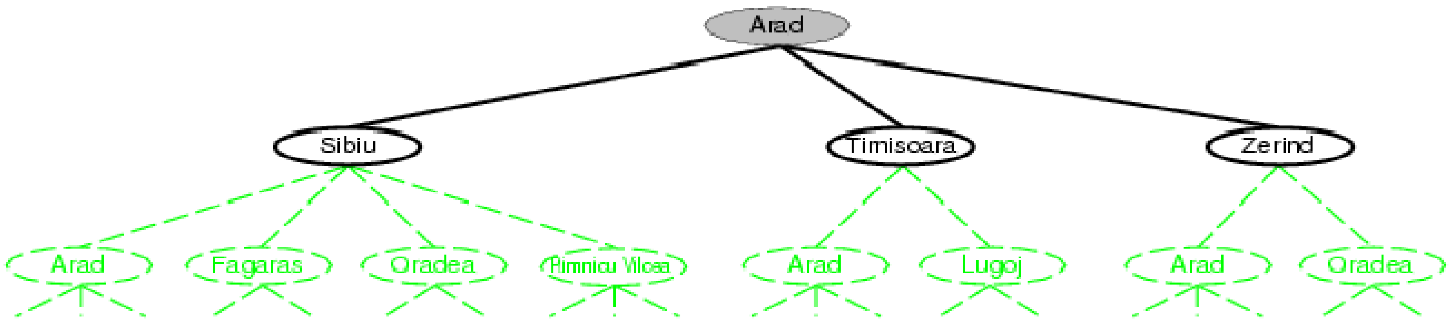


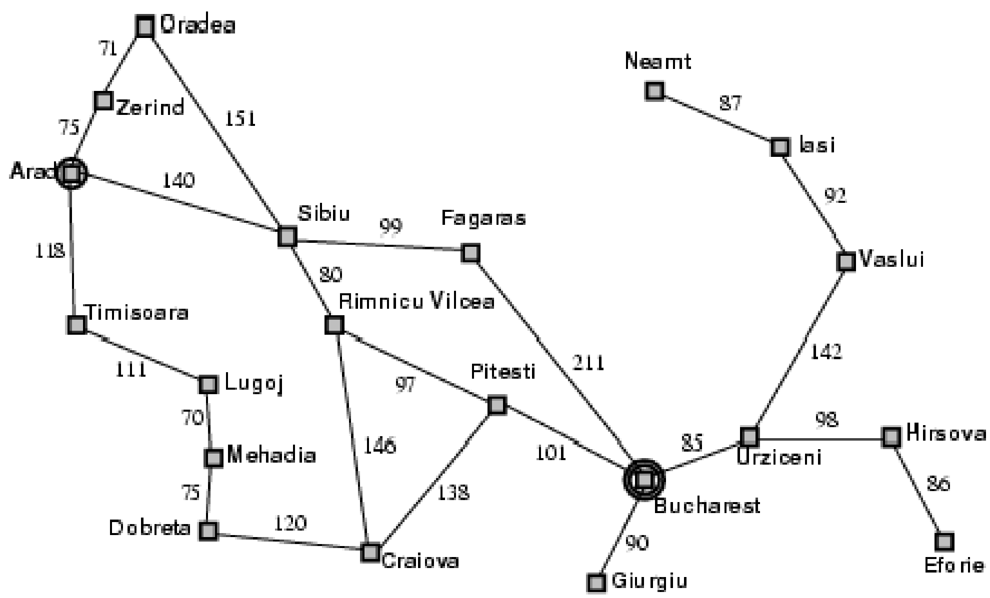
Tree search example



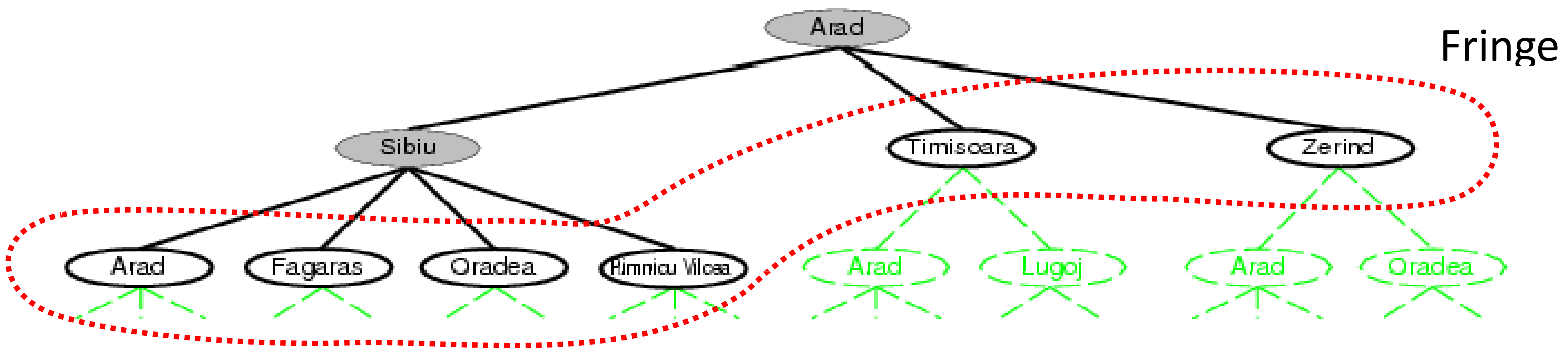


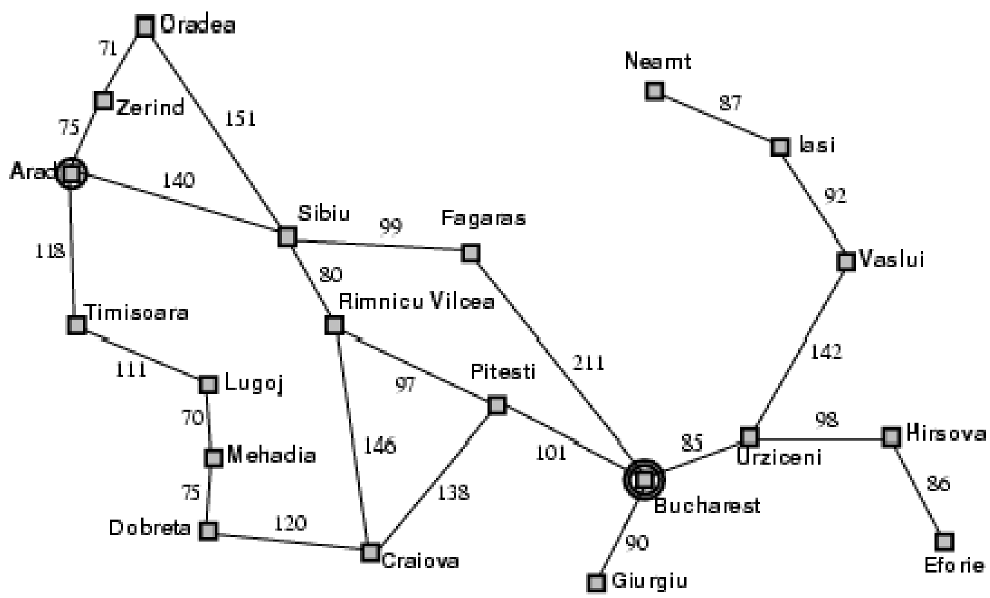
Tree search example





Tree search example





Search strategies

- A **search strategy** is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - **Completeness:** does it always find a solution if one exists?
  - **Optimality:** does it always find a least-cost solution?
  - **Time complexity:** number of nodes generated
  - **Space complexity:** maximum number of nodes in memory
- 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 length of any path in the state space (may be infinite)

Uninformed search strategies

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

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Iterative deepening search