Lect 6-8: Solving problems by searching

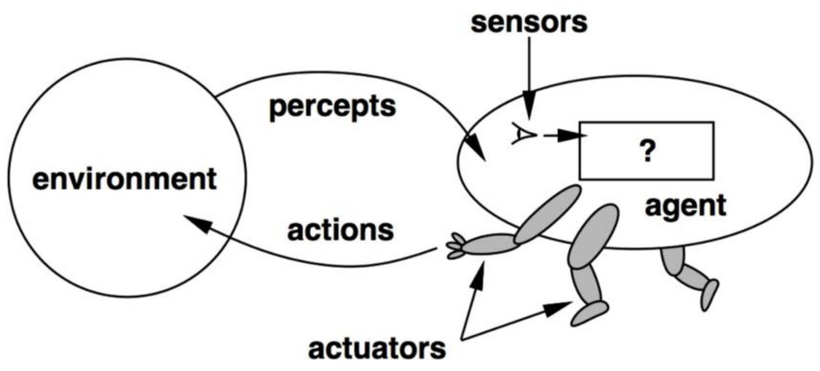
Dr. Debasis Das
Department of Computer Science and Engineering
Indian Institute of Technology Jodhpur

Search Strategies

Dr. Debasis Das
Associate Professor, Dept. of CSE
IIT Jodhpur

Problem Solving

We have rational agents that need to perform sequences of actions in order to achieve goals.



General approach for problem solving:

The agent to have **knowledge of the world** and how its actions affect it and be able to simulate execution of actions in an internal model of the world in order to determine a sequence of actions that will accomplish its goals.

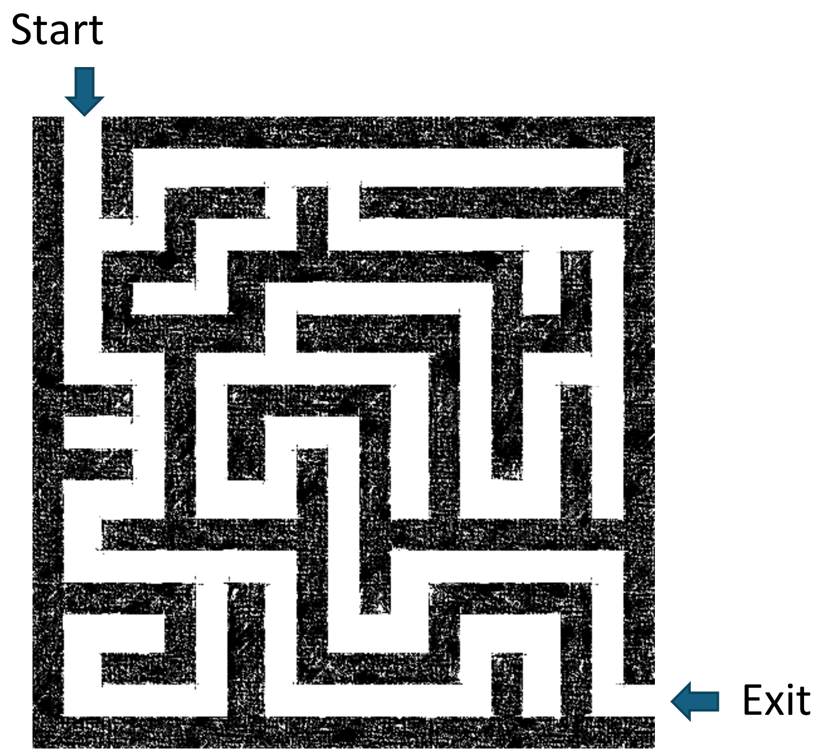
Typically performed by **searching** through an internally modelled space of world states

Contents

Contents

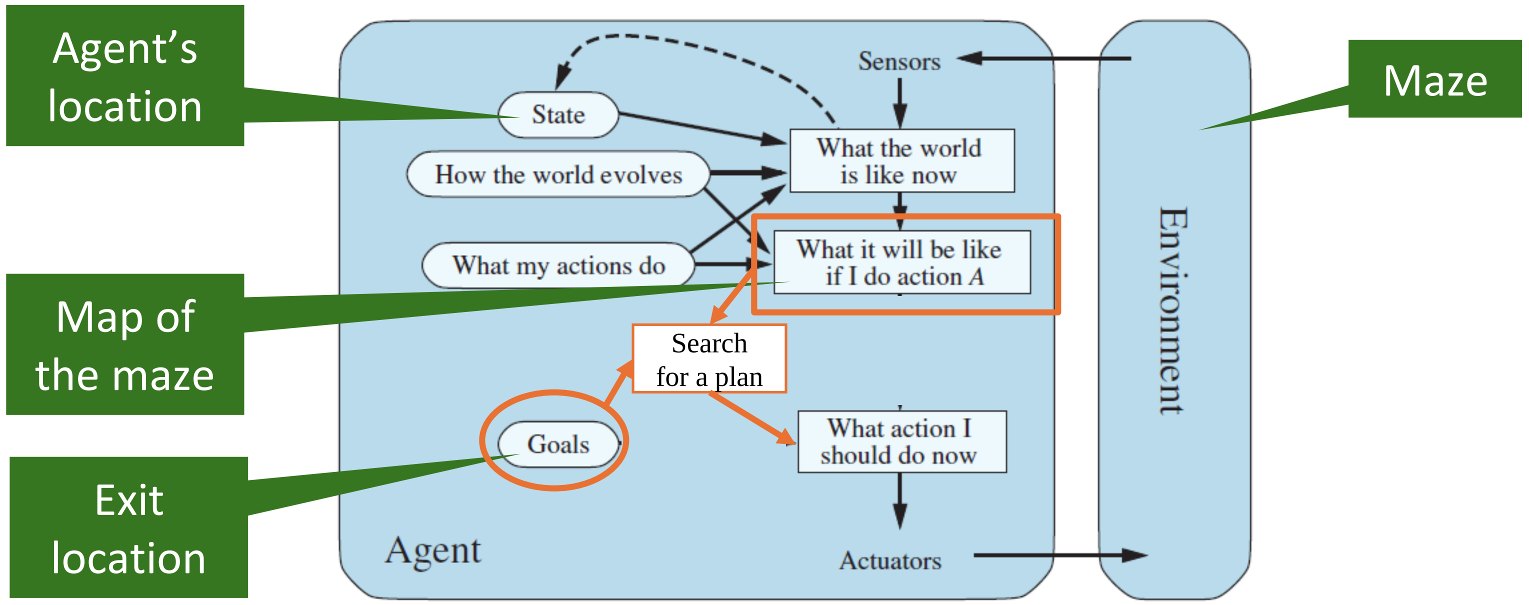
What are Search Problems?

- We will consider the problem of designing **goal-based agents** in **known, fully observable, and deterministic** environments.
- Example environment:



Remember: Goal-based Agent

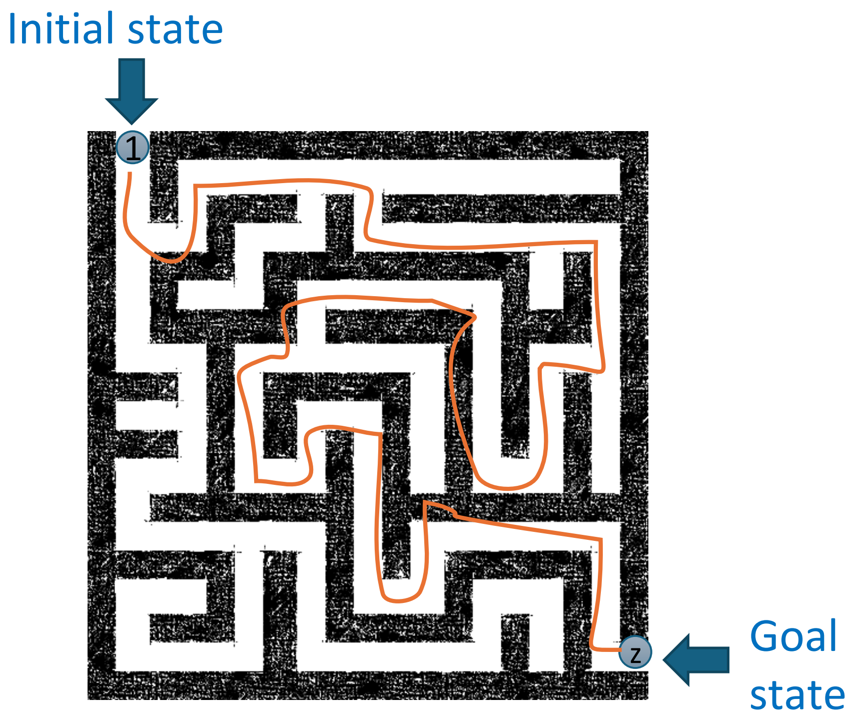
- The agent has the task to reach a defined **goal state**.
- The performance measure is typically the cost to reach the goal.
- We will discuss a special type of goal-based agents called **planning agents** which use **search algorithms** to plan a sequence of actions that lead to the goal.



$$a_s = \text{argmin}_{a \in A}[cost(s, s_1, s_2, \dots, s_n | s_n \in S^{goal})]$$

Planning for Search Problems

- For now, we consider only a discrete environment using an **atomic state representation** (states are just labeled 1, 2, 3, …).
- The **state space** is the set of all possible states of the environment and some states are marked as **goal states**.
- The **optimal solution** is the sequence of actions (or equivalently a sequence of states) that gives the lowest path cost for reaching the goal.

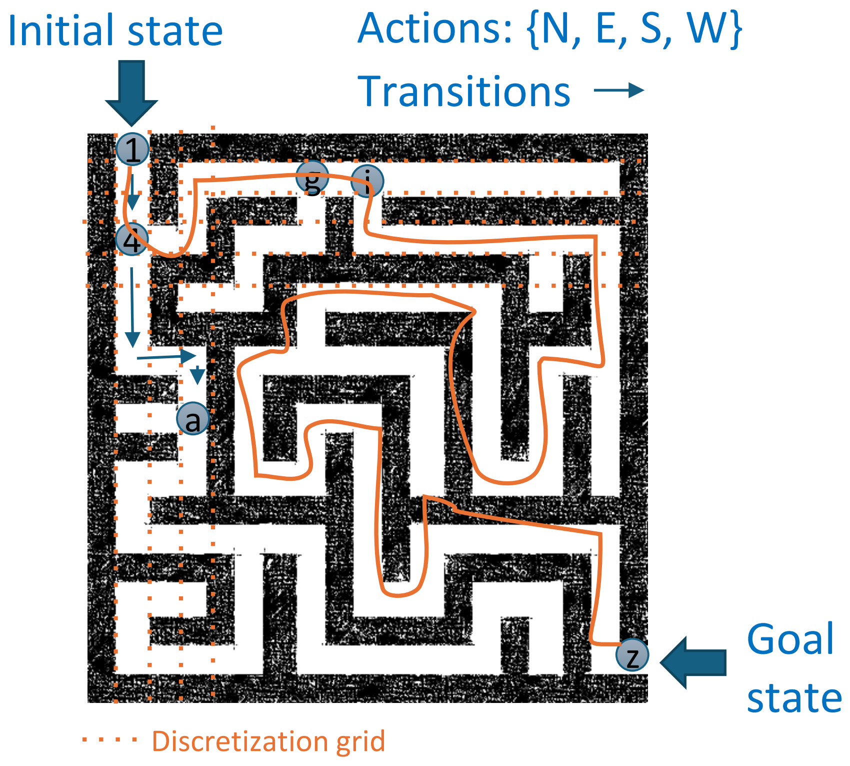


Phases:

1) **Search/Planning**: the process of looking for the **sequence of actions** that reaches a goal state. Requires that the agent knows what happens when it moves!
2) **Execution**: Once the agent begins executing the search solution in a deterministic, known environment, it can ignore its percepts (**open-loop system**).

Definition of a Search Problem

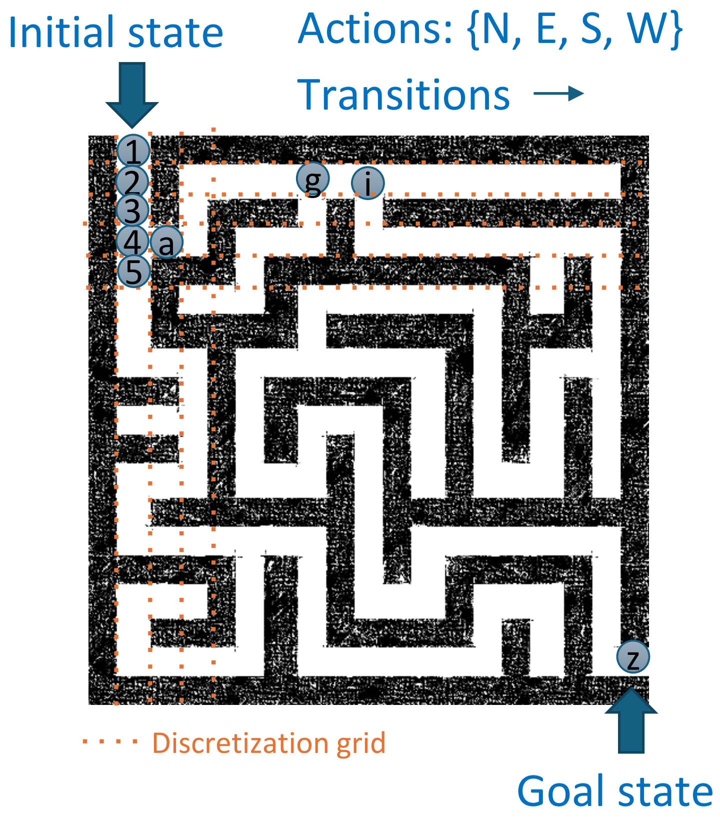
- **Initial state:** state description
- **Actions:** set of possible actions A
- **Transition model:** a function that defines the new state resulting from performing an action in the current state
- **Goal state:** state description
- **Path cost:** the sum of *step costs*



Important: The **state space** is typically too large to be enumerated, or it is continuous. Therefore, the problem is defined by initial state, actions and the transition model and not the set of all possible states.

Transition Function and Available Actions

Original Description



- As an action schema:
  $Action(go(dir))$
  PRECOND: no wall in direction dir
  EFFECT: change the agent's location according to dir
- As a function:
  $f: S \times A \rightarrow S$ or $s' = result(a, s)$
- Function implemented as a table representing the state space as a graph.

1	S	2
2	N	1
2	S	3
…	…	…
4	E	a
4	S	5
4	N	3
…	…	…

- Available actions in a state come from the transition function. E.g.,
  $actions(4) = \{E, S, N\}$

Note: Known and deterministic is a property of the transition function!

Original Description

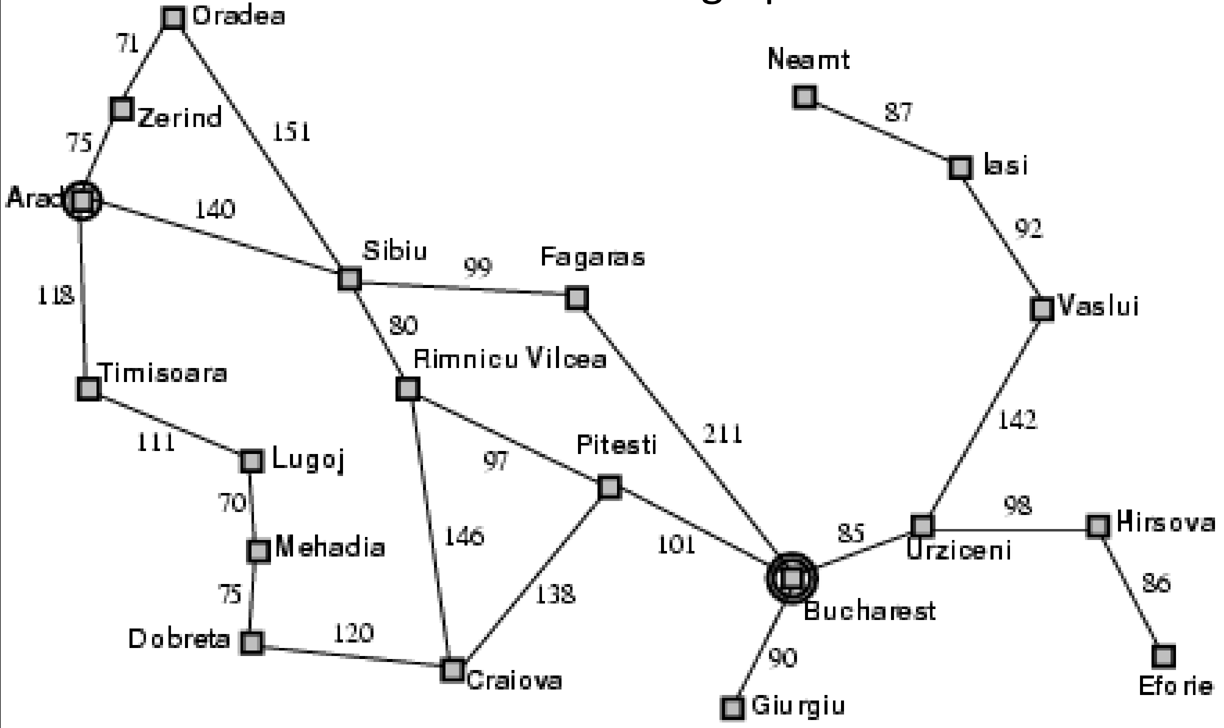
Example: Romania Vacation

- On vacation in Romania; currently in Arad
- Flight leaves tomorrow from Bucharest

- **Initial state:** Arad
- **Actions:** Drive from one city to another.
- **Transition model and states:** If you go from city A to city B, you end up in city B.
- **Goal state:** Bucharest
- **Path cost:** Sum of edge costs.

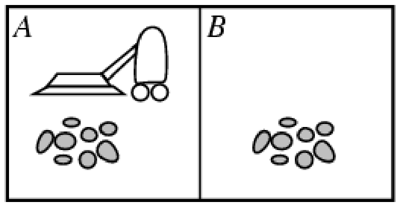
State Space/Transition model

Defined as a graph

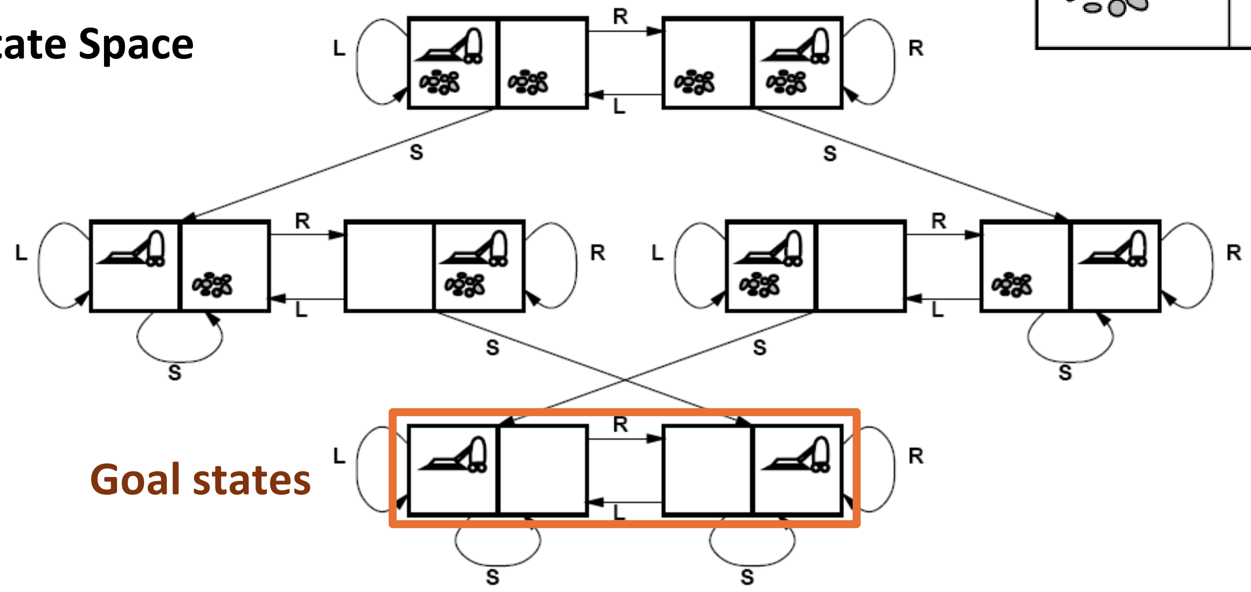


Distance in miles

Example: Vacuum world



State Space



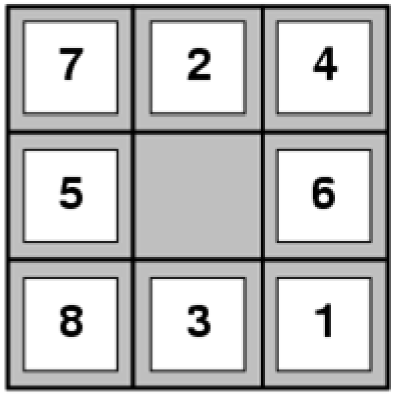
Goal states

- **Initial State:** Defined by agent location and dirt location.
- **Actions:** Left, right, suck
- **Transition model:** Clean a location or move.
- **Goal state:** All locations are clean.
- **Path cost:** E.g., number if actions

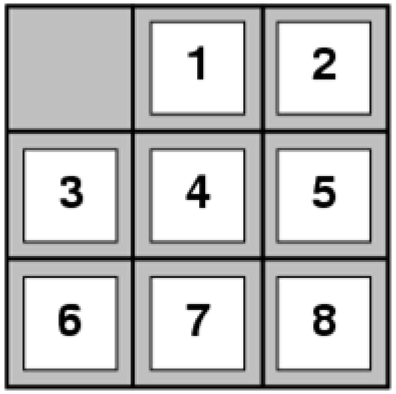
There are 8 possible atomic states of the system.
Why is the number of states for *n* possible locations $n(2^n)$?

Example: Sliding-tile Puzzle

- **Initial State:** A given configuration.
- **Actions:** Move blank left, right, up, down
- **Transition model:** Move a tile
- **Goal state:** Tiles are arranged empty and 1-8 in order
- **Path cost:** 1 per tile move.



Start State



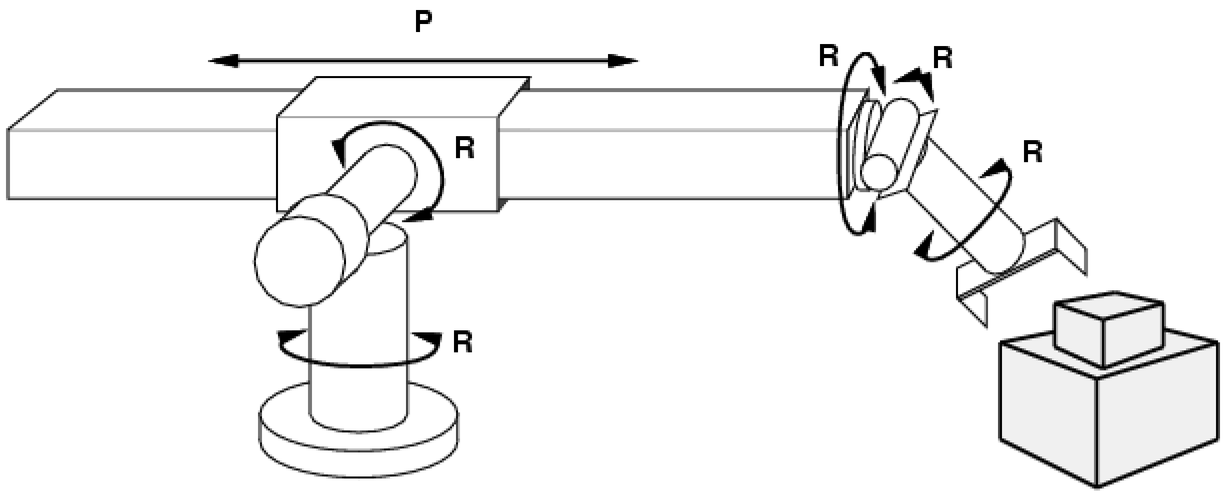
Goal State

State space size

Each state describes the location of each tile (including the empty one). ½ of the permutations are unreachable.

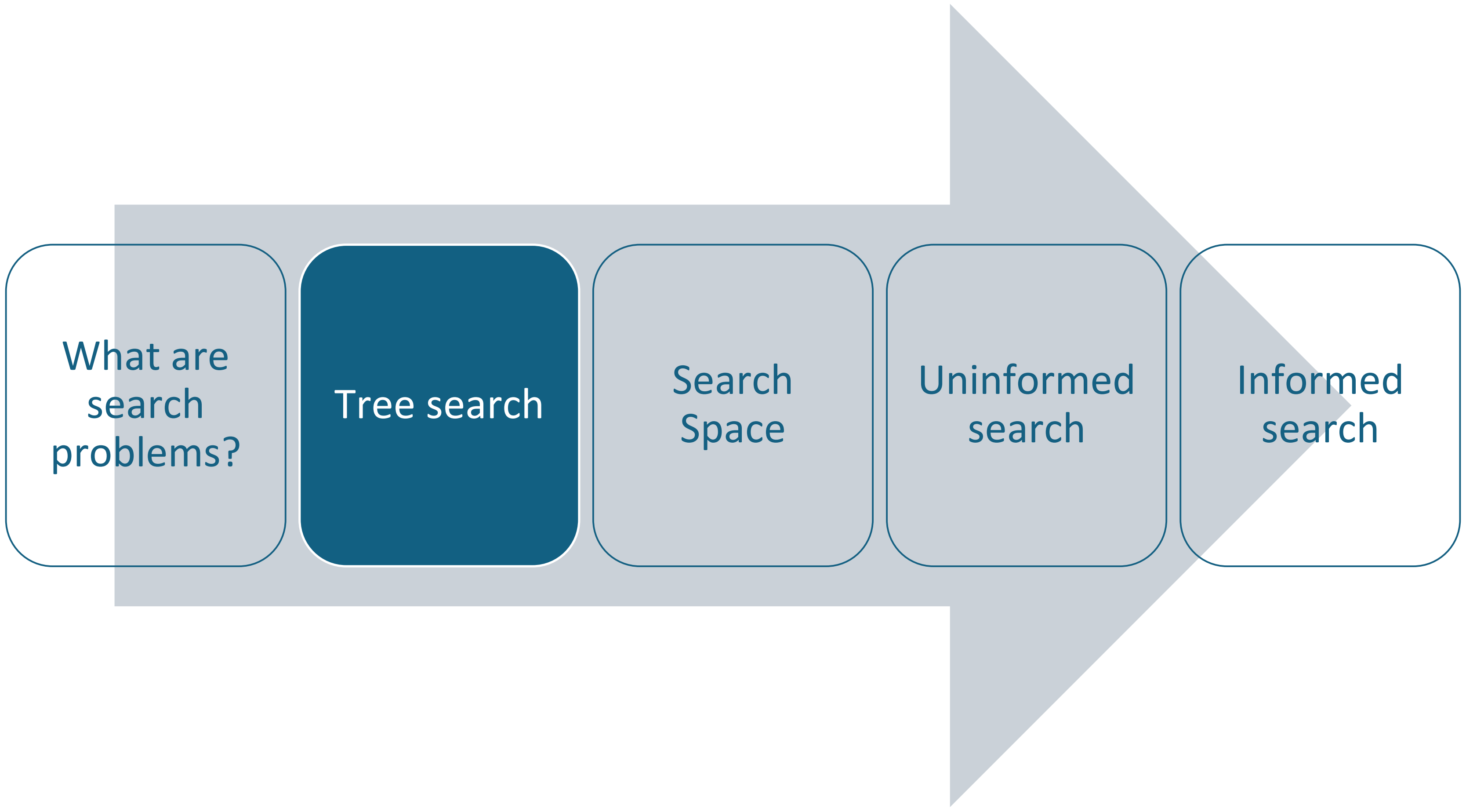
- 8-puzzle: $9!/2 = 181{,}440$ states
- 15-puzzle: $16!/2 \approx 10^{13}$ states
- 24-puzzle: $25!/2 \approx 10^{25}$ states

Example: Robot Motion Planning



- **Initial State:** Current arm position.
- **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.

Contents


What are search problems?
Tree search
Search Space
Uninformed search
Informed search

Solving Search Problems

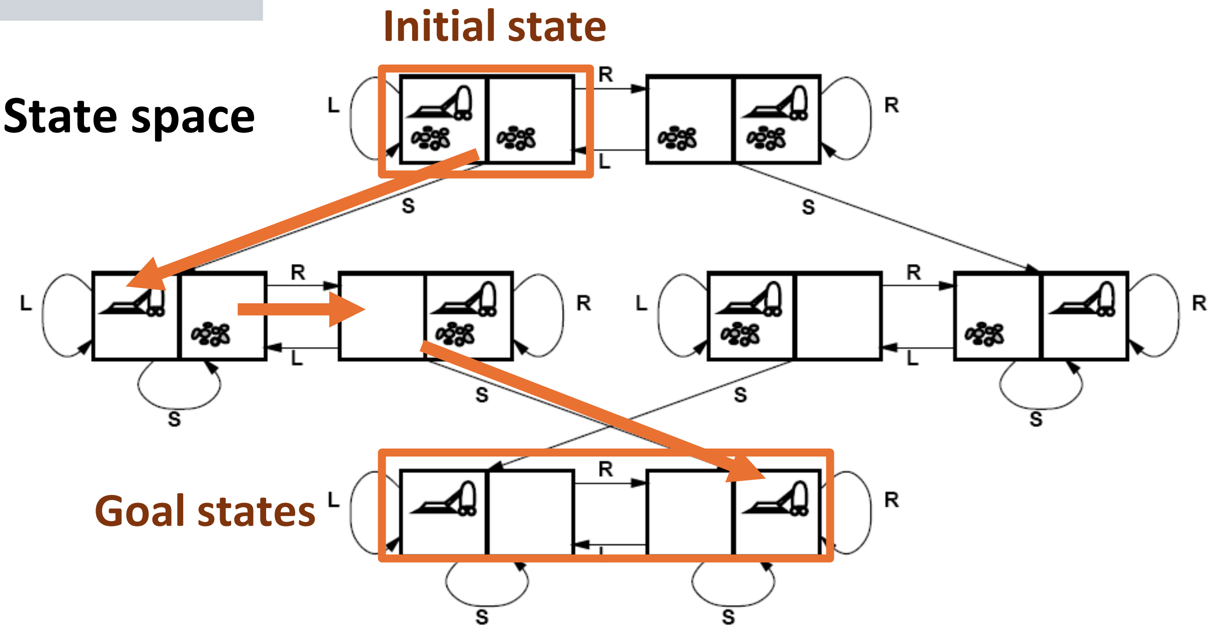
Given a search problem definition

- Initial state
- Actions
- Transition model
- Goal state
- Path cost

How do we find the optimal solution (sequence of actions/states)?

Construct a search tree for the state space graph!

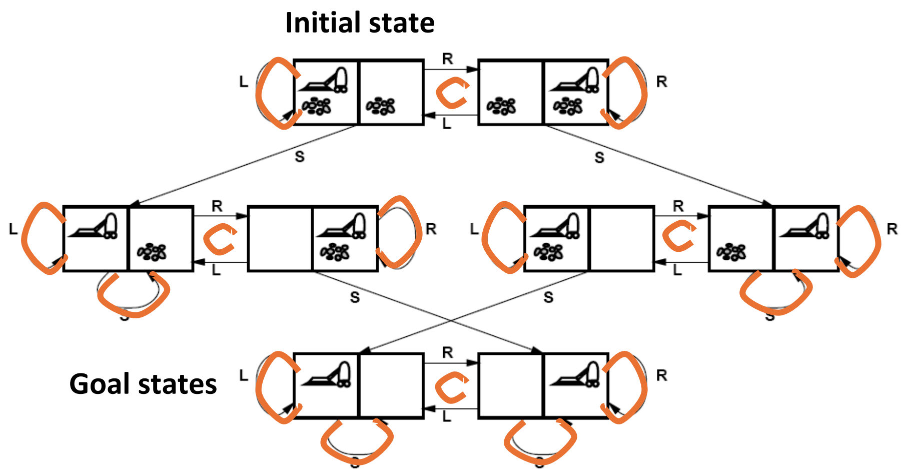
State space



Issue: Transition Model is Not a Tree! It can have Redundant Paths

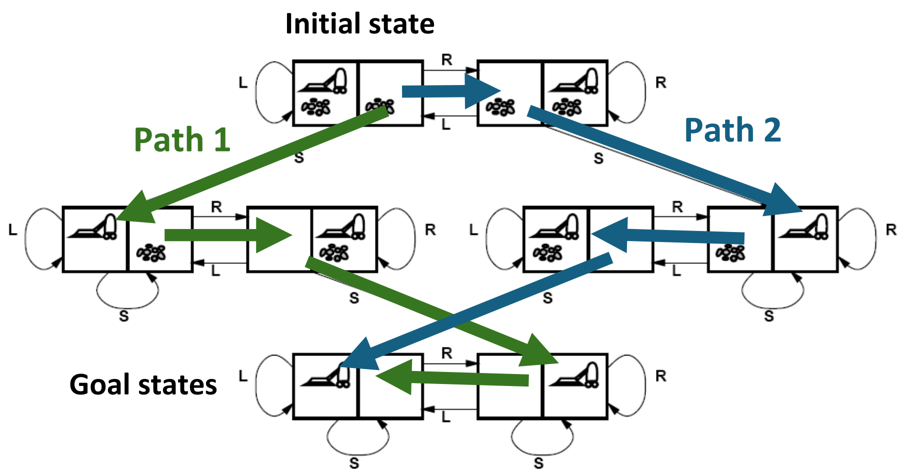
Cycles

Return to the same state. The search tree will create a new node!



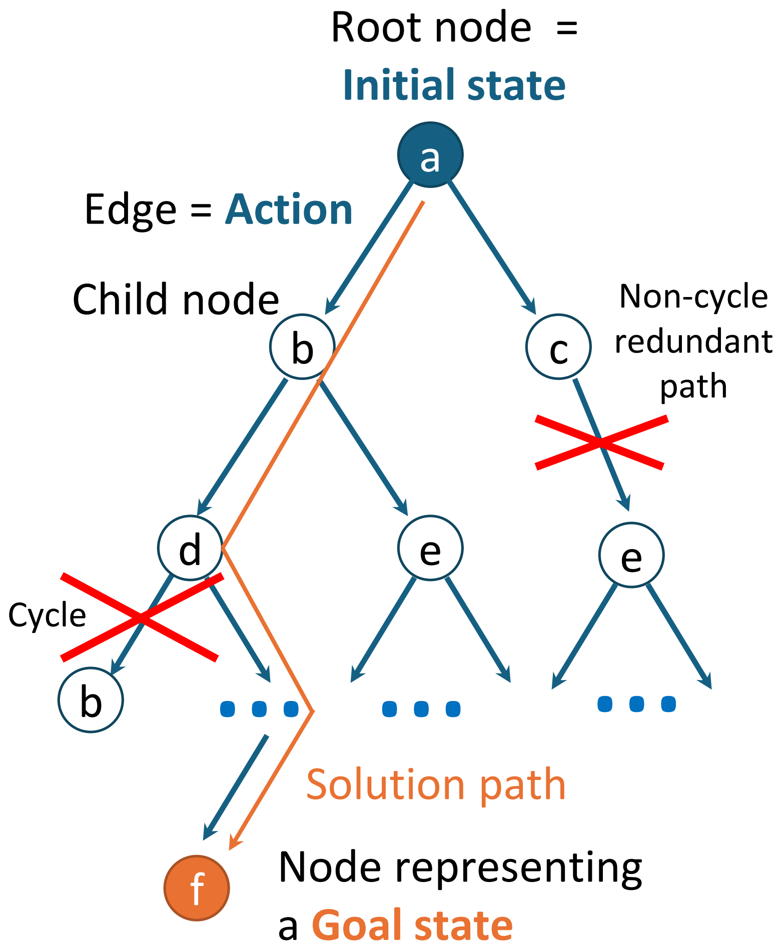
Non-cycle redundant paths

Multiple paths to get to the same state



Search Tree

- Superimpose a “what if” tree of possible actions and outcomes (states) on the state space graph.
- The **Root node** represents the initial stare.
- An action child node is reached by an **edge** representing an action. The corresponding state is defined by the transition model.
- Trees cannot have **cycles (loops)** or **multiple paths to the same state.** These are called redundant paths. Cycles in the search space must be broken to prevent infinite loops. Removing other redundant paths improves search efficiency.
- A **path** through the tree corresponds to a sequence of actions (states).
- A **solution** is a path ending in a node representing a goal state.
- **Nodes vs. states**: Each tree node represents a state of the system. If redundant path cannot be prevented then state can be represented by multiple nodes.



Differences Between Typical Tree Search and AI Search

Typical tree search

- Assumes a given tree that fits in memory.
- Trees have by construction no cycles or redundant paths.

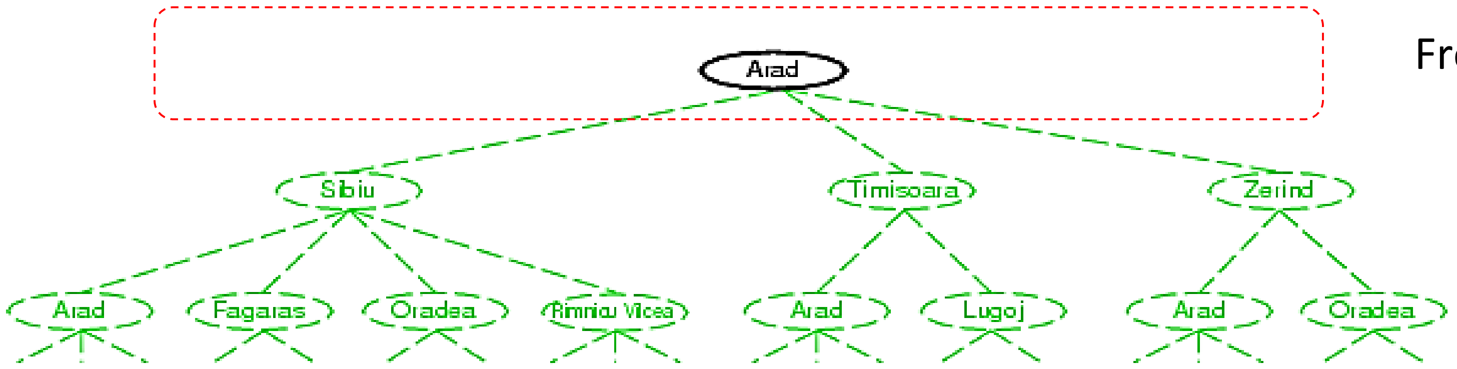
AI tree/graph search

- The search tree is too large to fit into **memory**.
  a. **Builds parts of the tree** from the initial state using the transition function representing the graph.
  b. **Memory management** is very important.
- The search space is typically a very large and complicated graph. Memory-efficient **cycle checking** is very important to avoid infinite loops or minimize searching parts of the search space multiple times.
- Checking redundant paths often requires too much memory and we accept searching the same part multiple times.

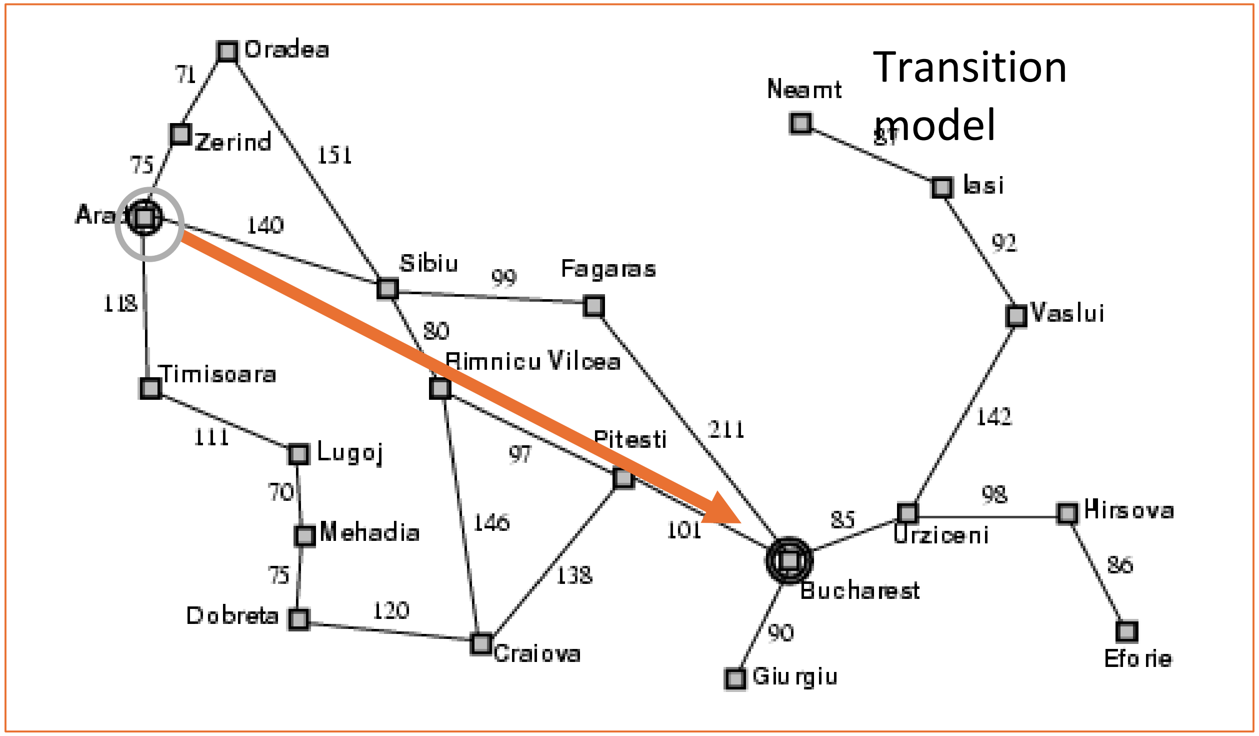
Tree Search Algorithm Outline

1. Initialize the **frontier** (set of unexplored know nodes) using the **starting state/root node.**
2. While the frontier is not empty:
   a) Choose next frontier node to expand according to **search strategy.**
   b) If the node represents a **goal state,** return it as the solution.
   c) Else **expand** the node (i.e., apply all possible actions to the transition model) and add its children nodes representing the newly reached states to the frontier.

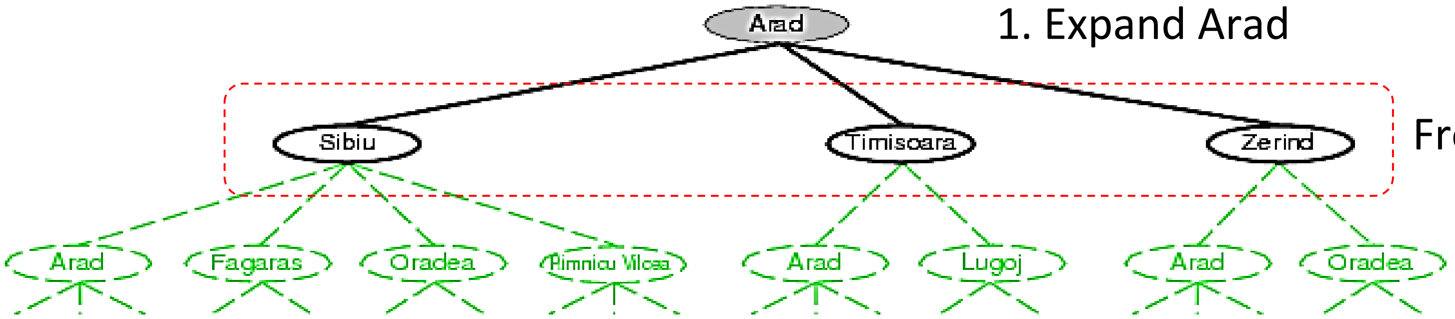
Tree Search Example



Frontier

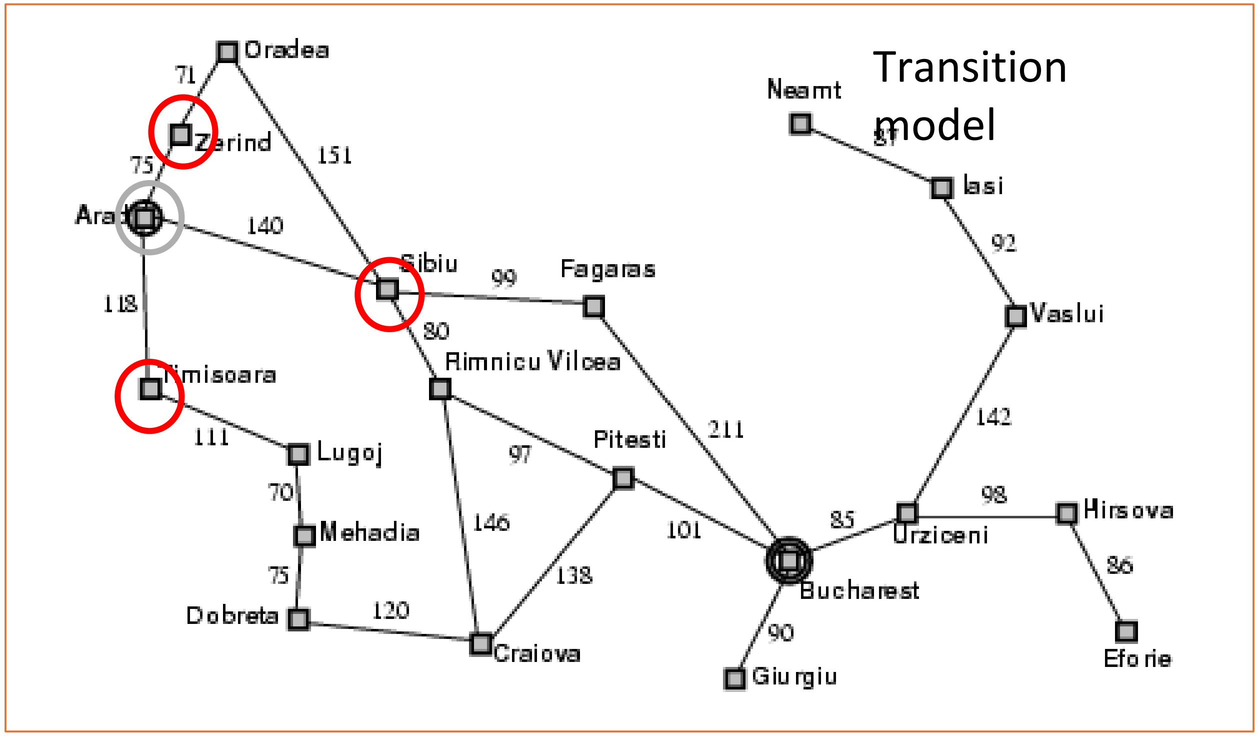


Tree Search Example



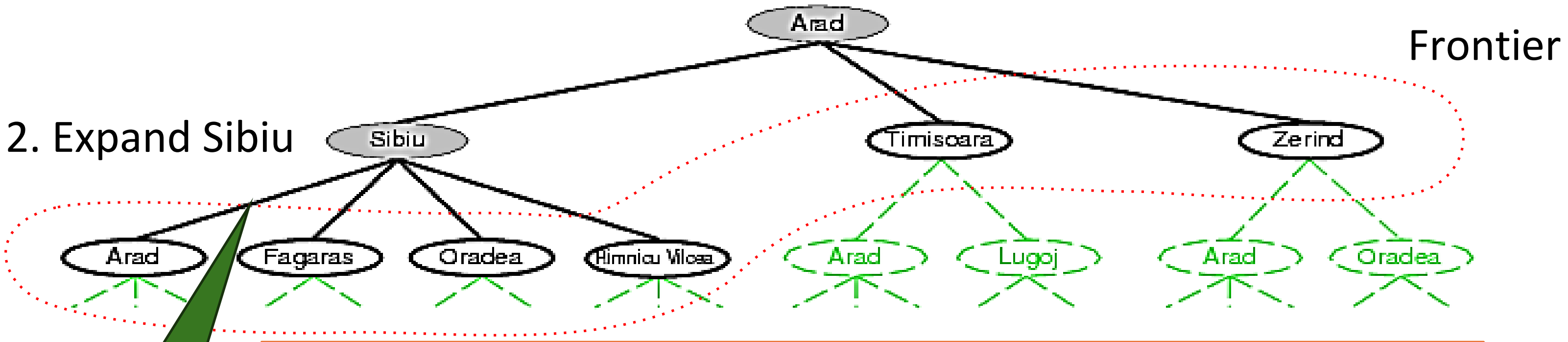
1. Expand Arad

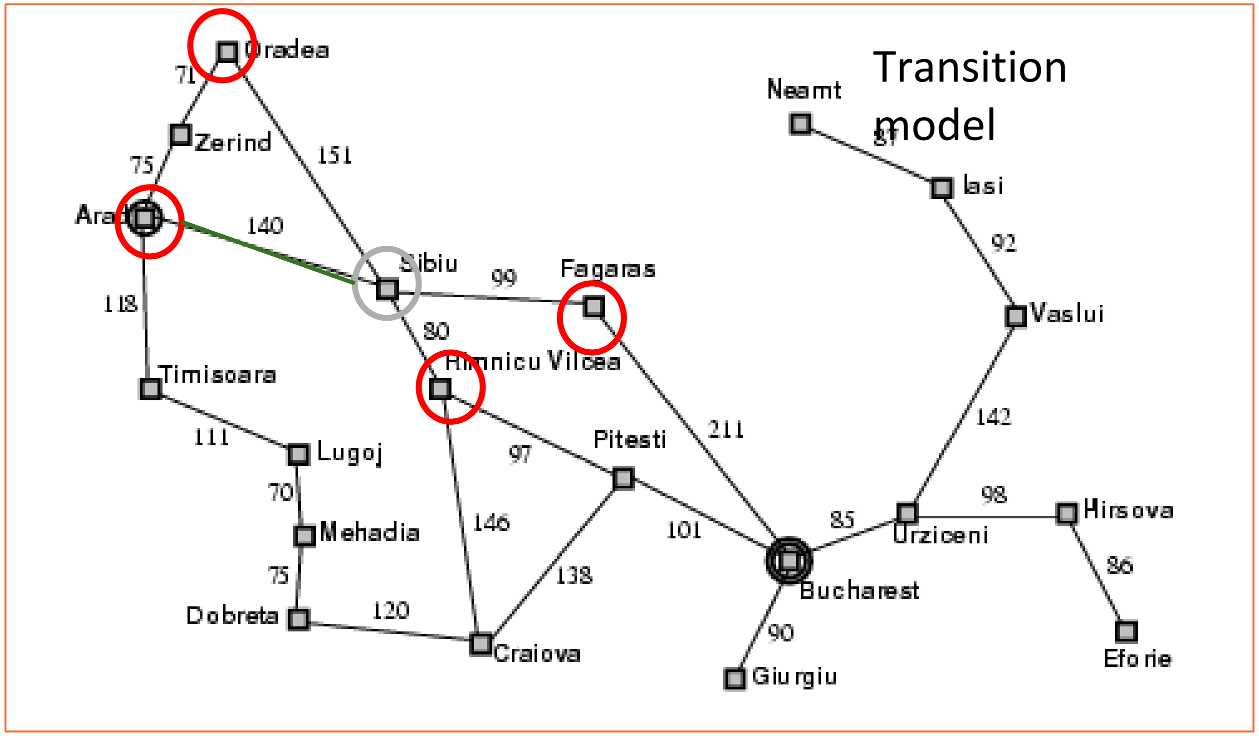
Frontier



Transition model

Tree Search Example





Example of a cycle

We could have also expanded Timisoara or Zerind!

Search Strategies: Properties

- 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:** how long does it take?
  - **Space complexity:** how much memory does it need?

Search Strategies: Time and Space Complexity

- A **search strategy** is defined by picking the **order of node expansion**.

- Worst case time and space complexity are measured in terms of the **size of the state space *n*** (= number of nodes in the search tree).

$$O(n)$$

Conclusion

- Tree search can be used for planning actions for **goal-based agents** in known, fully observable and deterministic environments.

- Issues are:
    - The large search space typically does not fit into memory. We use a transition function as a compact description of the **transition model.**
    - The search tree is built on the fly, and we have to deal with **cycles, redundant paths, and memory management**.