Artificial Intelligence
AI 2002

Lecture 3

Mahzaib Younas

Lecture Department of Computer Science

FAST NUCES CFD

Recap

- What is an Intelligent agent?
- Agents & Environments
- Performance measure
- Environment
- Actuators
- Sensors
- Types of intelligent agents

Problem Solving Agents

- A type of goal-based agent decide what to do by

  finding sequences of actions that lead to desirable states

- It based on two things
  - Goal Formulation
  - Problem Formulation

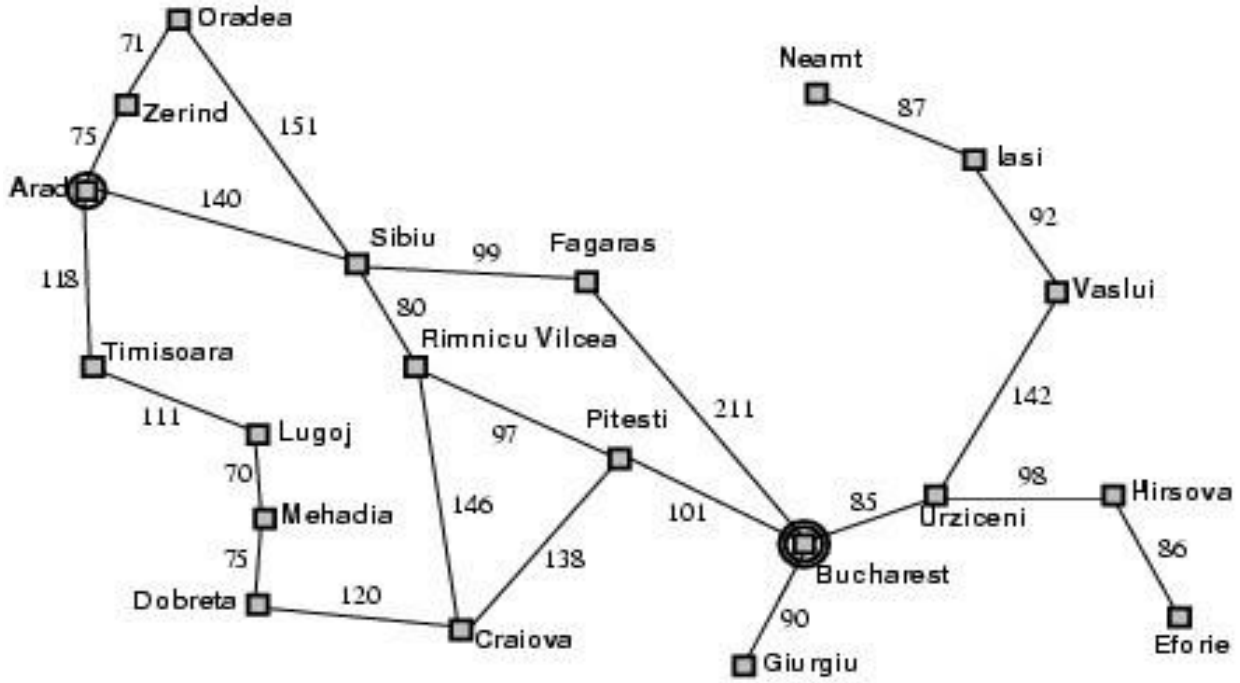
Problem Solving Agents (Cont…)

- Goal Formulation
  - based on current situation and agent's performance measure
- Problem Formulation
  - deciding what actions and states to consider, given a goal

The process of looking for such a sequence of actions is called search

Example: Romania Touring

On holiday the agent is in Romania visiting in Arab. Flight Leaves tomorrow from Bucharest.



Solution

1. Formulate goal (perf. evaluation):

   be in Bucharest before the flight

2. Formulate problem:

   states: various cities

   actions: drive between cities

3. Search:

   sequence of cities

Problem Solving Agent

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
              state, some description of the current world state
              goal, a goal, initially null
              problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

Detail

Step 1. Initialize the variables:

- seq: an action sequence, initially empty
- state: some description of the current world state
- goal: a goal, initially null
- problem: a problem formulation

Step 2. Update the state based on the percept received:

- state ← UPDATE-STATE(state, percept)

Step 3. Check if the action sequence is empty:

- if seq is empty then
- Formulate the goal based on the current state:
- goal ← FORMULATE-GOAL(state)
- Formulate the problem based on the current state and goal:
- problem ← FORMULATE-PROBLEM(state, goal)
- Search for a solution to the problem:
- seq ← SEARCH(problem)
- If no solution is found (seq = failure), return a null action.

Step 4. If a solution is found, execute the first action in the sequence:

- action ← FIRST(seq)
- Remove the first action from the sequence:
- seq ← REST(seq)

Step 5. Return the action to be performed.

Well Defined Problem:

- A problem can be defined formally by five components:
  - Initial state
  - Actions
  - Transition model: description of what each action does (successor)
  - Goal test
  - Path cost

Problem Solving Agent (Cont…)

Actions

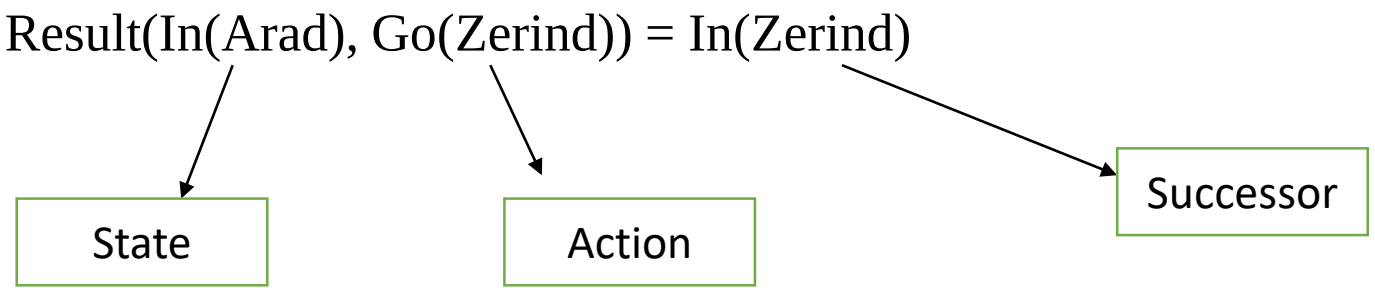
- A description of possible actions available to agent.
- Given particular state s, **Action(s)**
- **Action(s),** is applicable for s.

- Example:
  - From state In(Arad) our actions can be
    1. Go (Sibiu)
    2. Go (Timisoara)
    3. Go (Zerind)

Problem Solving Agent (Cont…)

Transition Model: A description of what each step does; is called the transition model.

Example:



Problem Solving Agent (Cont…)

State Space:

- Combine the initial state, action and transition is called the state space of the problem.
- The set of all states reachable from the initial state by any sequence of action

Graph:

- Directed network in which the nodes are state and links between nodes and action.

- **Path**
  - Path in the state space is the sequence of states connected by sequence of action.

Problem Solving Agent (Cont…)

Goal Test

- It determine whether a given state is goal state.
- Two types of goal test
  - explicit, e.g. In(Bucharest)
    - set of possible goal states, and the test simply checks whether the given state is one of them. The agent's goal in Romania is the singleton set {In(Bucharest)}.
  - implicit, e.g. checkmate
    - in chess, the goal is to reach a state called "checkmate," where the opponent's king is under attack and can't escape.

- **Path Cost**
  - that assigns a numeric cost to each path
  - Example:
    - distance traveled
- **Step Cost:**
  - The step cost of taking action a in state s to reach state s′ is denoted by c(s, a, s′). The step costs for Romania are shown in Figure as route distances.

Problem Solving Agent (Cont…)

Solution:

- A solution to a problem is an action sequence that leads from the initial state to a goal state.

Optimal Solution:

- Solution quality is measured by OPTIMAL SOLUTION the path cost function, and an optimal solution has the lowest path cost among all solutions

Problem Abstraction

- Real world is complex and has more details

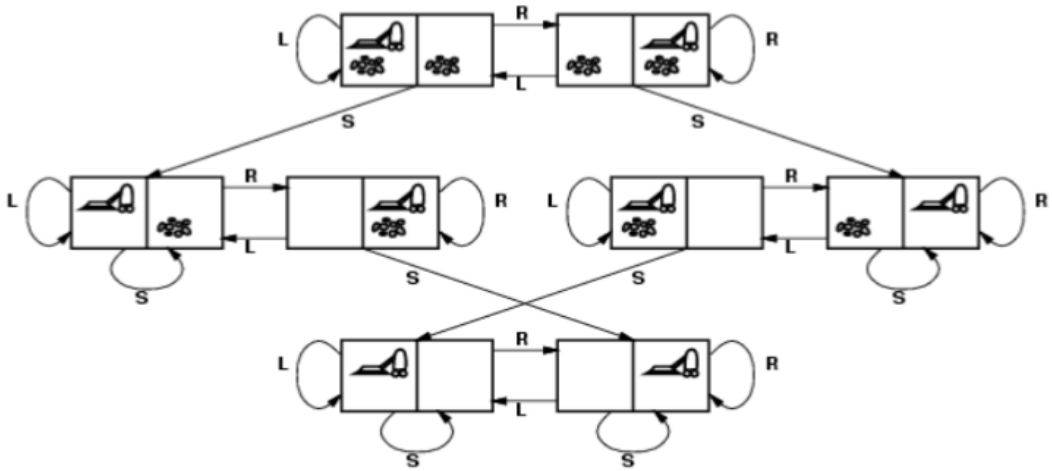
  Irrelevant details should be removed from state space and actions, which is called abstraction

- What's the appropriate level of abstraction?
  - Valid
    - the abstraction is valid, if we can expand it into a solution in the more detailed world
  - Useful
    - the abstraction is useful, if carrying out the actions in the solution is easier than the original problem

  remove as much detail as possible while retaining validity and usefulness

Example: Vacuum Problem

The state space for the vacuum world



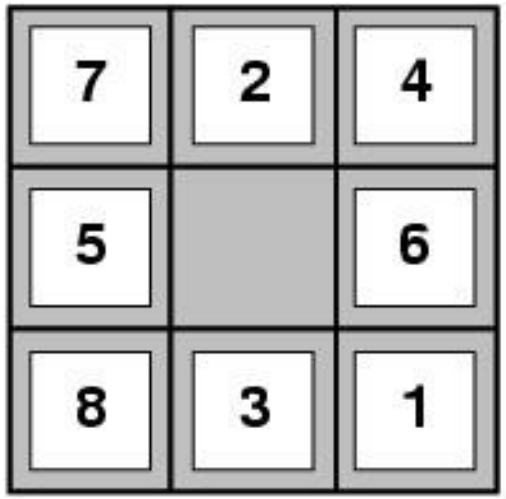
states? integer dirt and robot location
actions? *Left*, *Right*, *Suck*
goal test? no dirt at all locations
path cost? 1 per action

Explanation

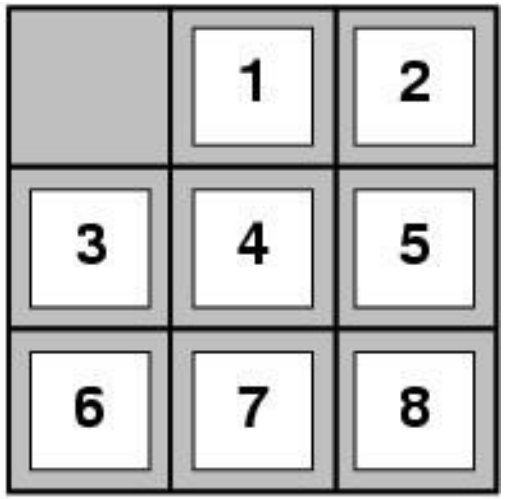
- **States:**
  - The state is determined by both the agent location and the dirt locations. The agent is in one of two locations, each of which might or might not contain dirt. Thus there are $2 \times 2^2 = 8$ possible world states. A larger environment with n locations has $n \cdot 2^n$ states.
- **Initial state:**
  - Any state can be designated as the initial state.
- **Actions:**
  - In this simple environment, each state has just three actions: Left, Right, and Suck. Larger environments might also include Up and Down.
- **Transition model:**
  - The actions have their expected effects, except that moving Left in the leftmost square, moving Right in the rightmost square, and Sucking in a clean square have no effect. The complete state space is shown in Figure
- **Goal test:**
  - This checks whether all the squares are clean.
- **Path cost:**
  - Each step costs 1, so the path cost is the number of steps in the path.

Example 2: 8-Puzzle Problem

1. States
2. Initial state
3. Actions
4. Transition model
5. Goal test
6. Path cost.

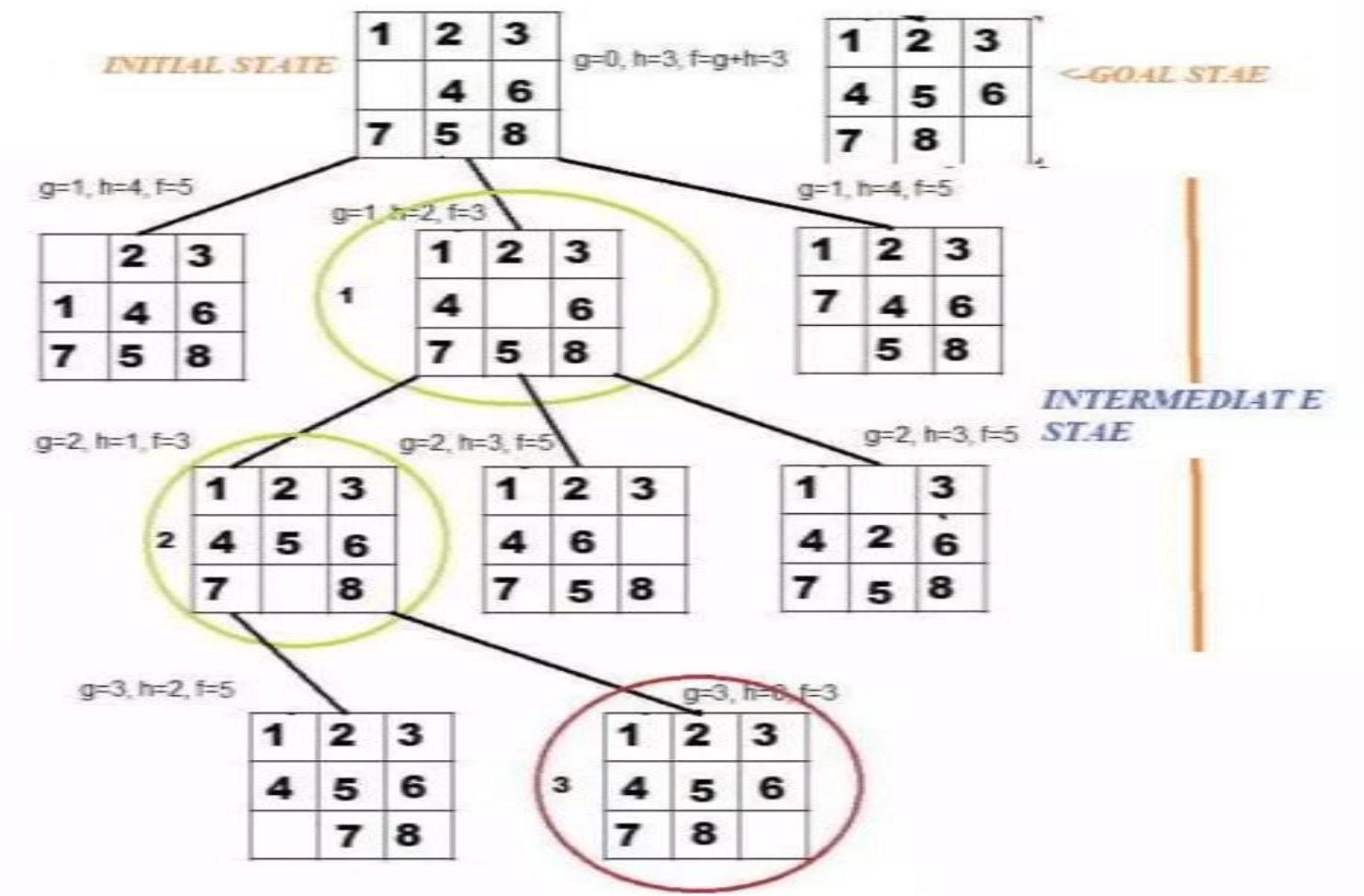


Start State



Goal State

Example 2: 8-Puzzle Problem



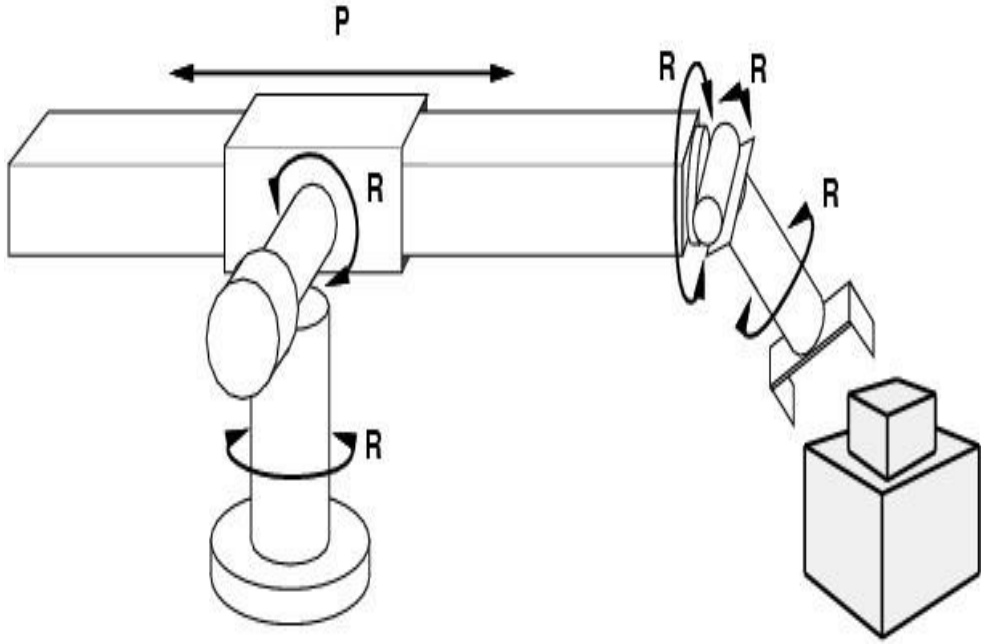
Solution:

The standard formulation is as follows:

- **States**
  - A state description specifies the location of each of the eight tiles and the blank in one of the nine squares. Total states are 9
- **Initial state**
  - Any state can be designated as the initial state. Note that any given goal can be reached from exactly half of the possible initial states (Exercise 3.17).
- **Actions**
  - The simplest formulation defines the actions as movements of the blank space Left, Right, Up, or Down. Different subsets of these are possible depending on where the blank is.
- **Transition model**
  - Given a state and action, this returns the resulting state; for example, if we apply Left to the start state in Figure the resulting state has the 5 and the blank switched.
- **Goal test**
  - This checks whether the state matches the goal
- **Path cost**
  - Each step costs 1, so the path cost is the number of steps in the path.

Example: Robotic Assembly

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



Real World Problem

Airline travel problems that must be solved by a travel planning Web site:

Solution:

States:

Each state obviously includes a location (e.g., an airport) and the current time. Furthermore, because the cost of an action (a flight segment) may depend on previous segments, their fare bases, and whether they were domestic or international, the state must record extra information about these "historical" aspects.

Initial state:

This is specified by the user's query.

Actions

Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if there is a preceding flight segment.

Transition model:

The state resulting from taking a flight will have the flight's destination as the current location and the flight's arrival time as the current time.

Goal test:

Are we at the final destination specified by the user?

Path cost:

This depends on monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards, and so on.

Practice Problem:

1. Touring Problem
2. Travelling salesperson problem
3. VLSI Layout
4. Robot Navigation
5. Automatic Assembly Sequencing

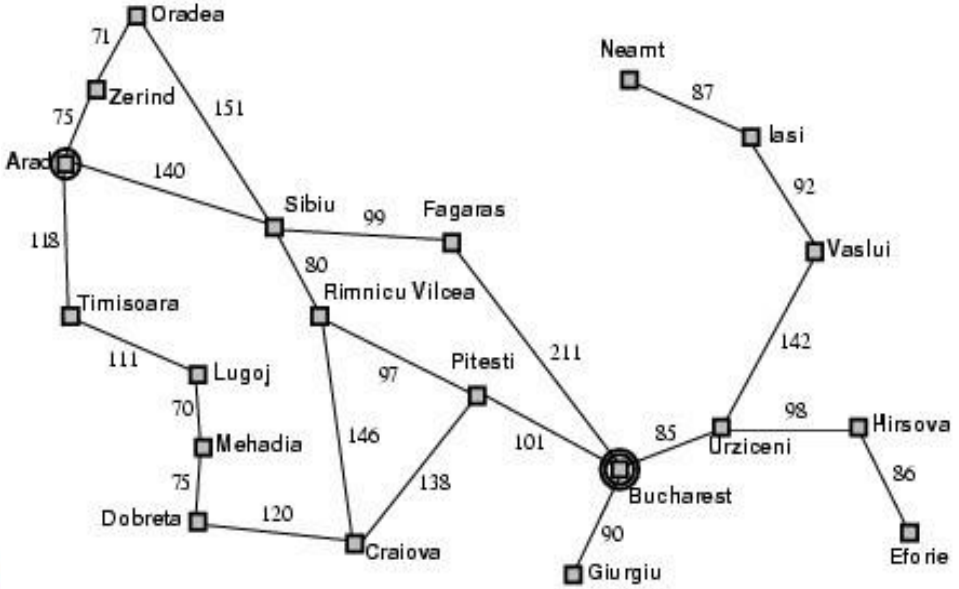
Searching for Solution

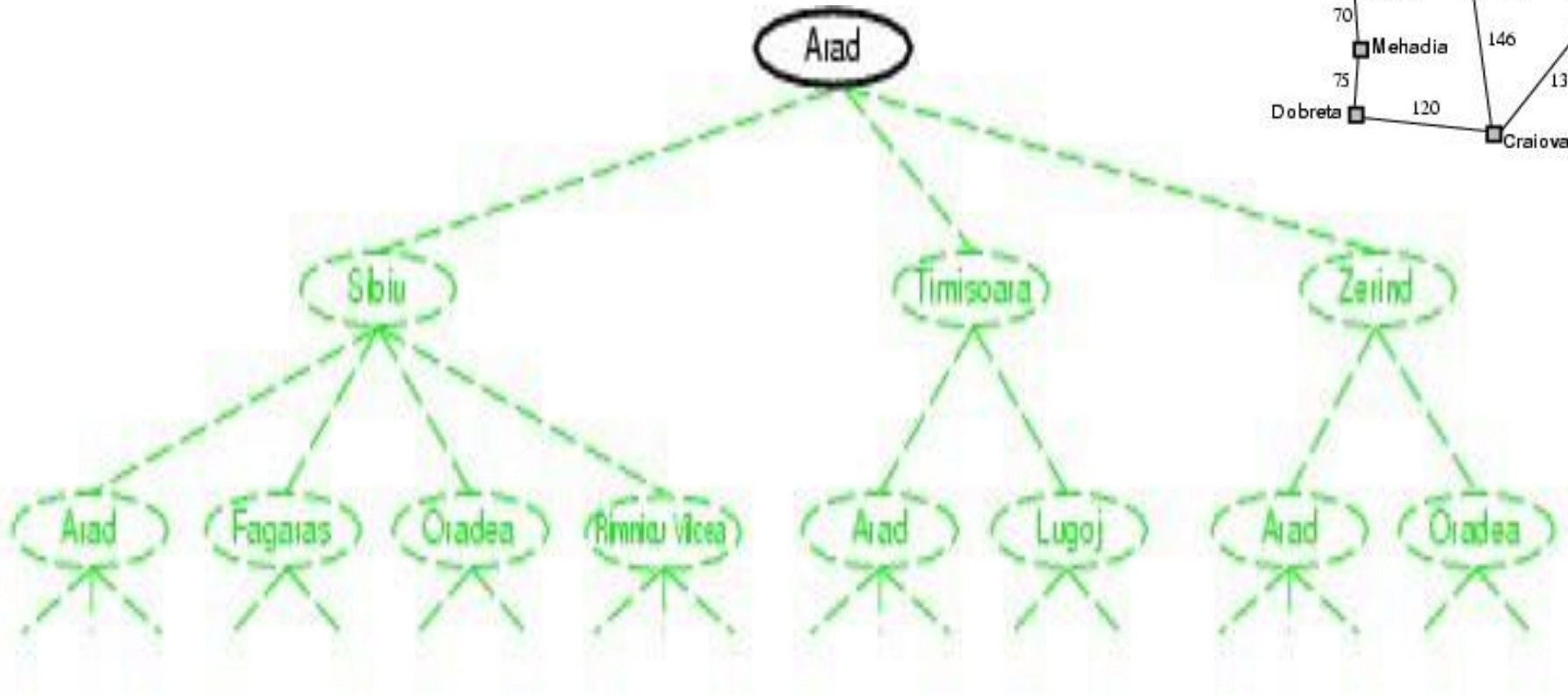
Search Tree

- **Search tree:** generated by initial state and possible actions

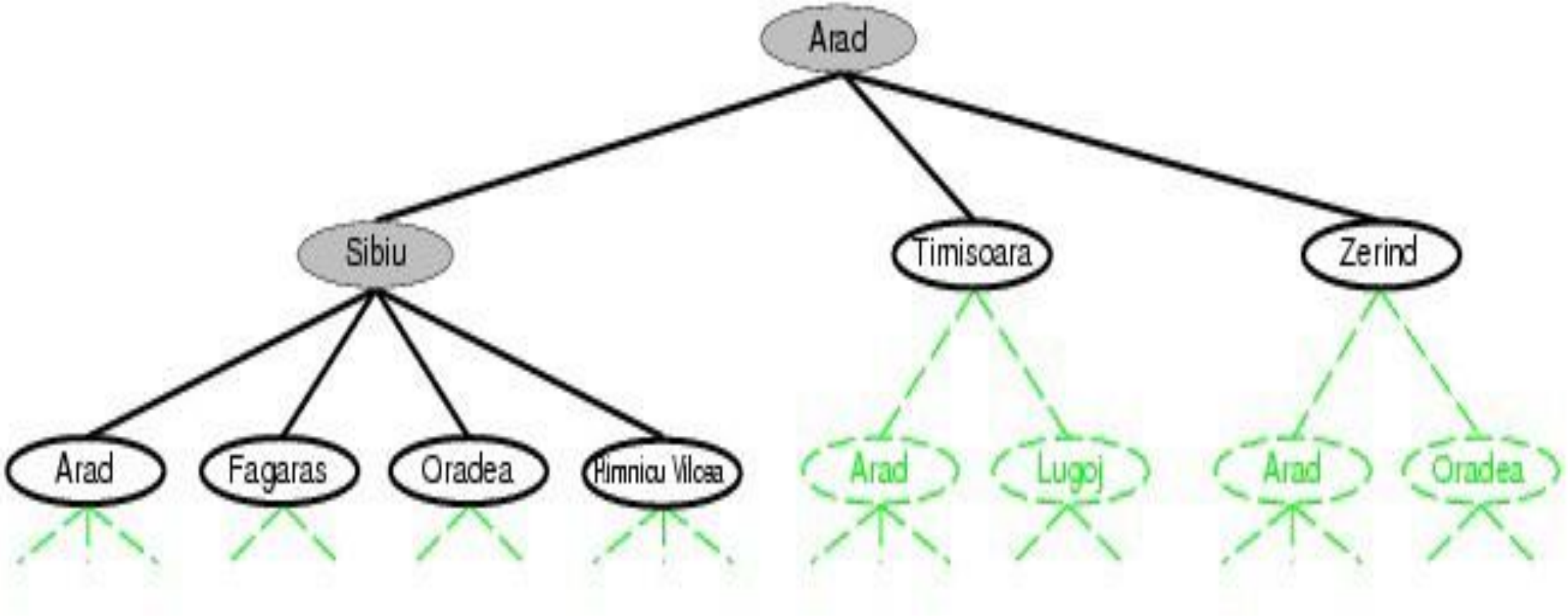
- Basic idea:
  - offline, simulated exploration of state space by generating successors of already-explored states (expanding states)
  - the choice of which state to expand is determined by search strategy

Tree Search Example





Tree Example (Cont…)



Basic Terminologies

- **Frontier**

  Set of all leaf nodes available for expansion at any given point

- **Repeated state**

- **Loopy path**

  Arad to Sibiu to Arad

- **Redundant path**

  More than one way to get from one state to another

General Tree Search Algorithm

```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
```

Explanation:

Step 1: Initialize the frontier using the initial state of the problem

Step 2: Enter a loop:

Check if the frontier is empty:

If the frontier is empty, return failure as no solution is found.

Choose a leaf node from the frontier and remove it:

Select a node from the frontier based on the chosen search strategy

Step 3: Check if the chosen node contains a goal state:

If the chosen node contains a goal state, return the corresponding solution.

Step 4: Expand the chosen node:

Generate new nodes by applying valid actions to the chosen node's state.

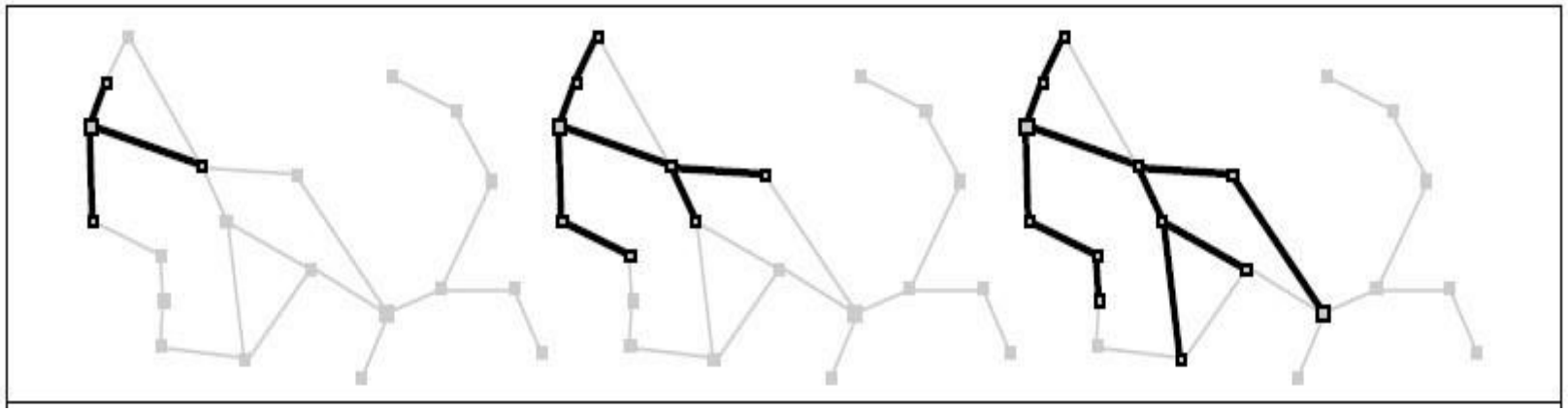
Add the resulting nodes to the frontier.

General Graph Search Algorithm

```
function GRAPH-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    initialize the explored set to be empty
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        add the node to the explored set
        expand the chosen node, adding the resulting nodes to the frontier
            only if not in the frontier or explored set
```

We augment Tree-Search with a explored set, which remembers every expanded node

Graph Search Example

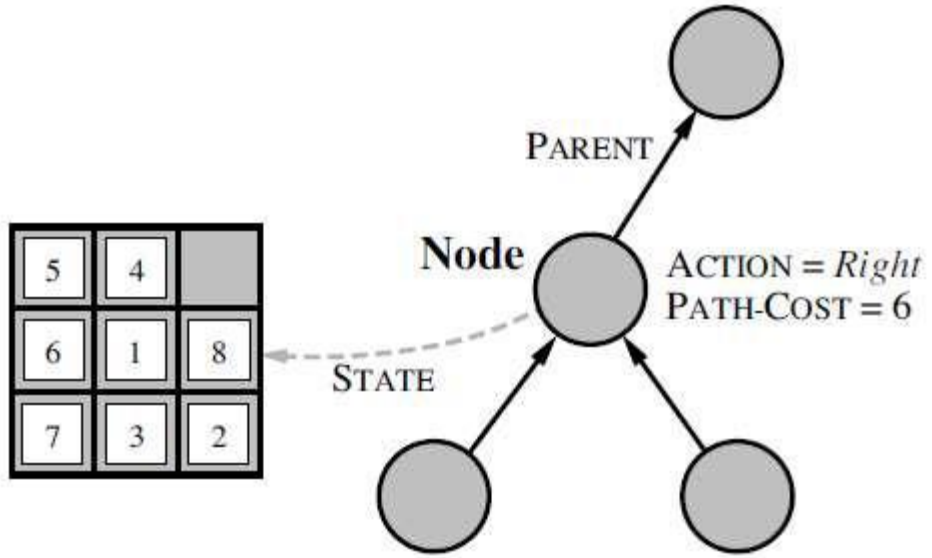


Implementation: State vs Nodes

1. A state is a representation of a physical configuration

2. A node is a data structure constituting part of a search tree includes state, parent node, action, path cost g(n), depth

3. A solution path can be easily extracted



Child Node Function

1. The CHILD-NODE function takes a parent node and an action and returns the resulting child node

```
function CHILD-NODE(problem, parent, action) returns a node
  return a node with
      STATE = problem.RESULT(parent.STATE, action) PARENT = parent
      ACTION = action
      PATH-COST = parent.PATH-COST +
                  problem.STEP-COST(parent.STATE, action)
```