ARTIFICIAL INTELLIGENCE

Russell & Norvig

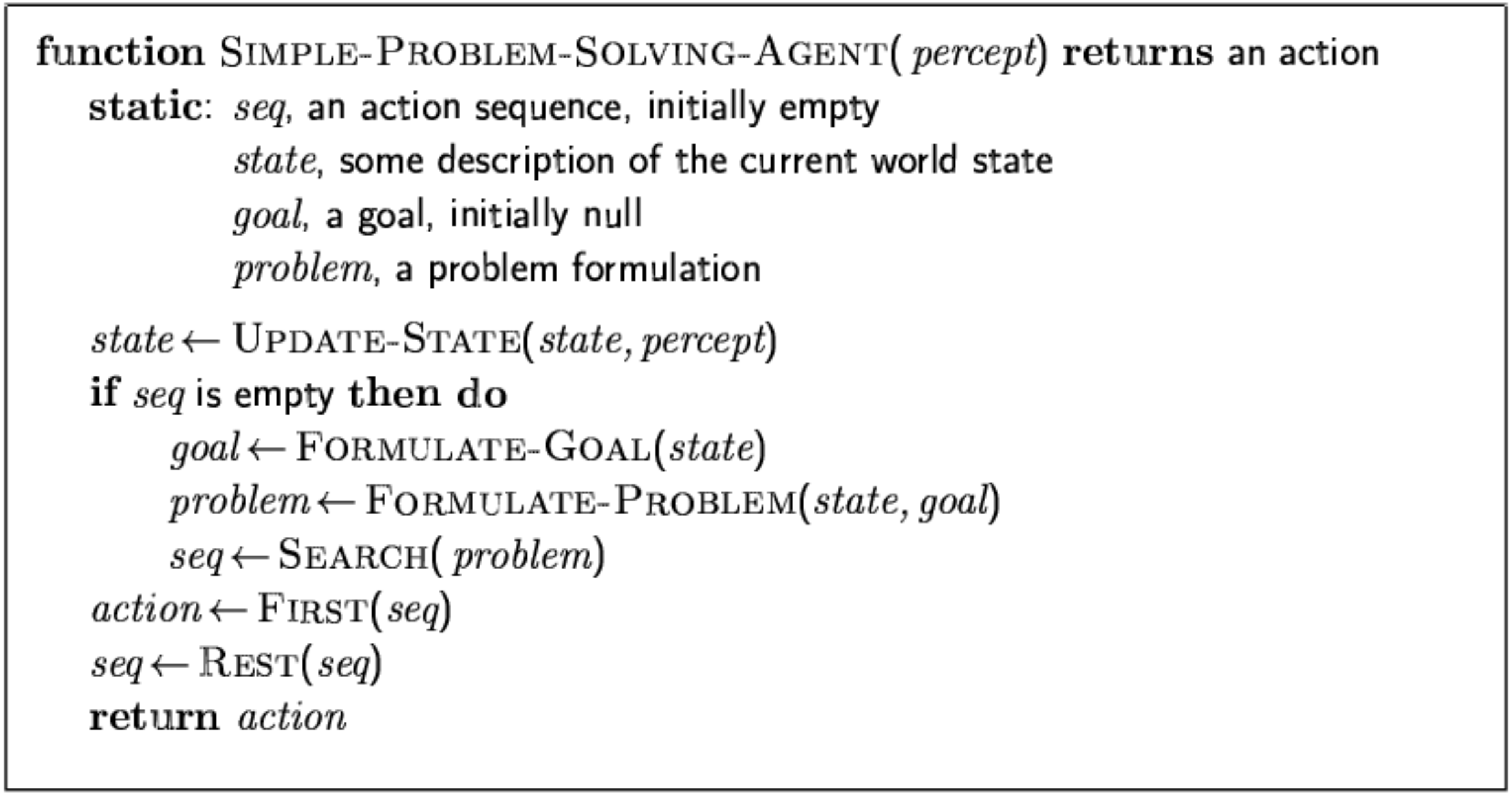
Chapter 3: Solving Problems by Searching

Problem-Solving Agents

- **Goal** is set of states where goal is achieved
- Must consider **level of abstraction** to formulate problem
  - Which actions are important in problem solution?
- Typically consider situation of solving problem "offline" then executing the planned solution
- While executing plan, percepts are ignored

Process: Formulate problem ➔ Search ➔Execute

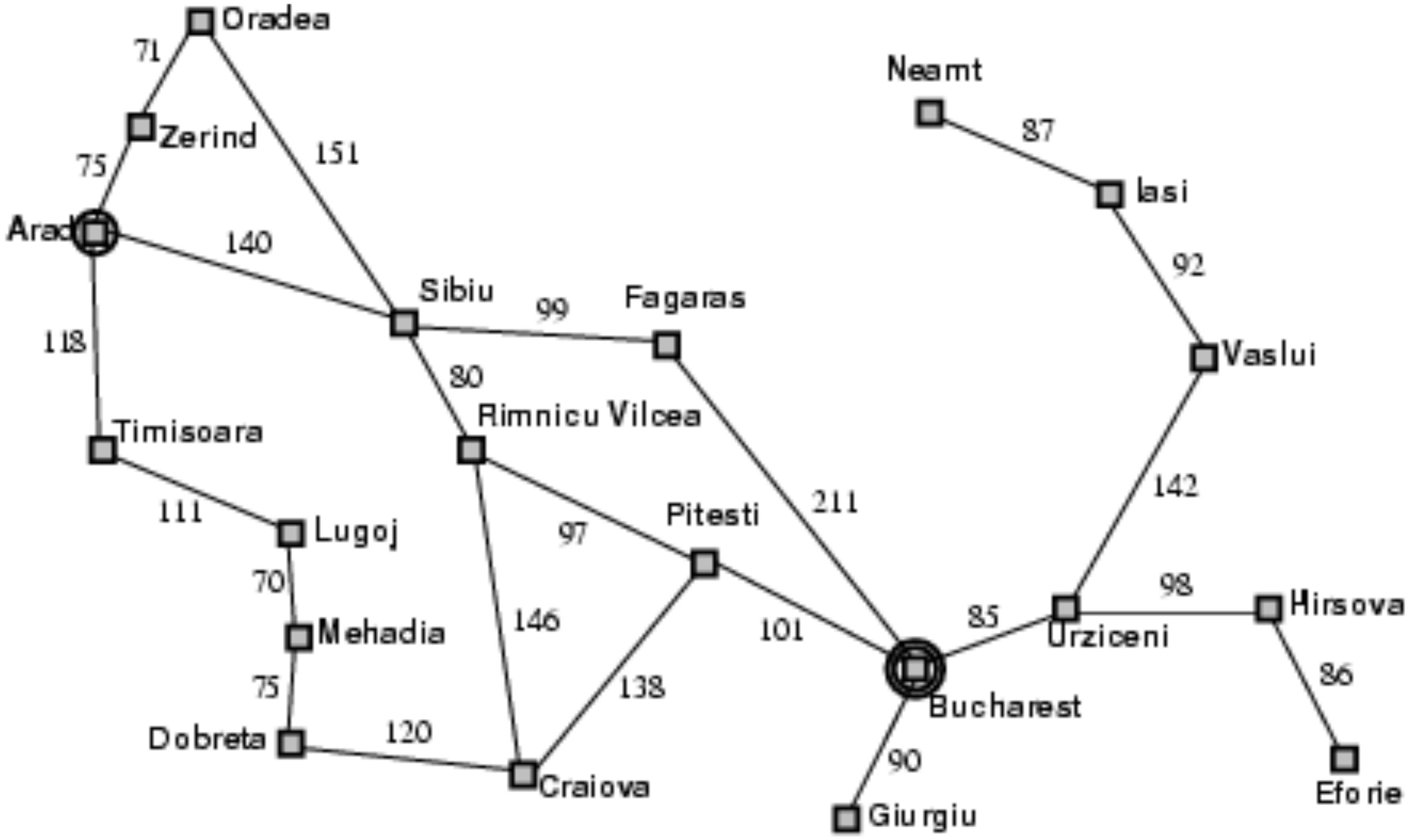
Problem-Solving Agents



```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
    static: 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 do
        goal ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, goal)
        seq ← SEARCH(problem)
    action ← FIRST(seq)
    seq ← REST(seq)
    return action
```

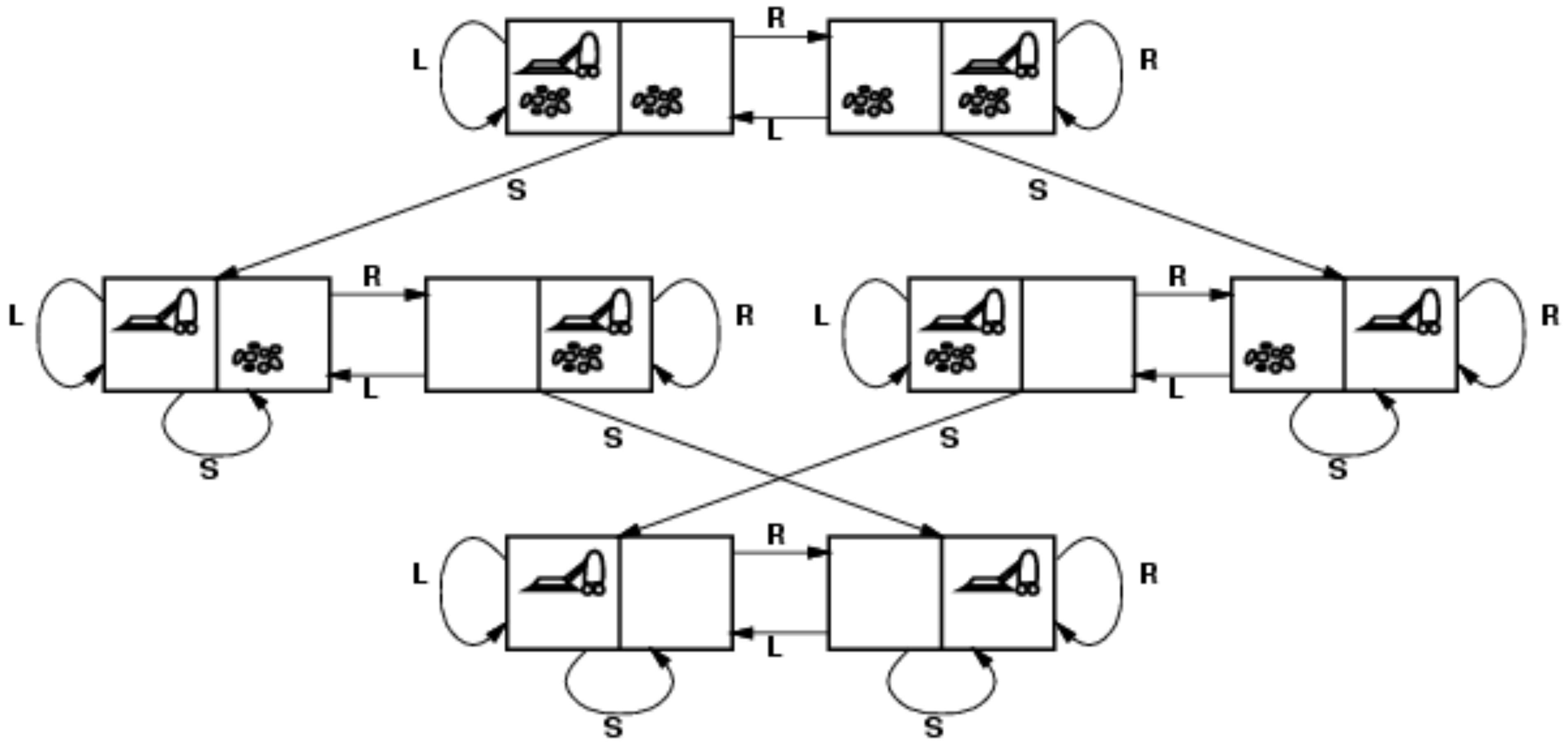
Simple roadmap of Romania



Problem definition components

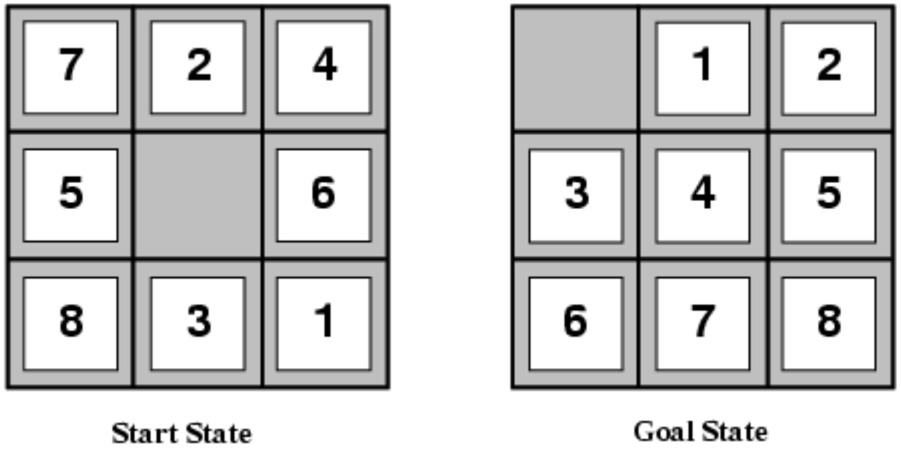
1. Initial State
   - For example, In(Arad)
2. Possible Actions
   - For state s, Action(s) returns actions that can be executed in s
   - Actions(In(Arad)) = {Go(Sibiu), Go(Timisoara), Go(Zerind)}
3. Transition Model
   - Successor function, like delta (δ) transitions in finite state machines
   - Together, initial state, actions and transition model define the **state space**
4. Goal Test
   - Similar to “final state”, e.g. {In(Bucharest)}, or abstract property (checkmate)
5. Path Cost
   - Agent’s cost function used as internal performance measure. Usually sum of cost of actions along path from initial state to goal state

Vacuum cleaner world



8-puzzle (sliding-block puzzle)

- 3x3 board with 8 numbered tiles and a blank
- Any tile adjacent to blank can slide into blank spot
- States: any configuration, e.g.: 7,2,4,5,0,6,8,3,1
- Initial state: any
- Actions: easiest to specify moving of blank space (ULDR)
- Transitions, Goal, Path Cost?



Route-finding problem

- Like the Romania example
- Lots of applications—web sites, in-car systems, airline systems, etc
- For any of these can define problem with respect to:
  - States
  - Initial state
  - Actions
  - Transition model
  - Goal test
  - Path cost
- Other variations: robot navigation, TSP, etc

Formulating Navigation Problem

- **Set of States**
  - individual cities, e.g., Memphis, Oxford, Batesville, Jackson, New Orleans, Biloxi, Mobile, Little Rock
- **Operators**
  - freeway routes from one city to another
  - e.g., Memphis to Jackson, Biloxi to Mobile
- **Start State**
  - current city where we are, Oxford
- **Goal States**
  - City or set of cities that represent a final destination, e.g., New Orleans
- **Solution**
  - a sequence of operators which get us there,
    - e.g., Oxford, Batesville, Jackson, New Orleans

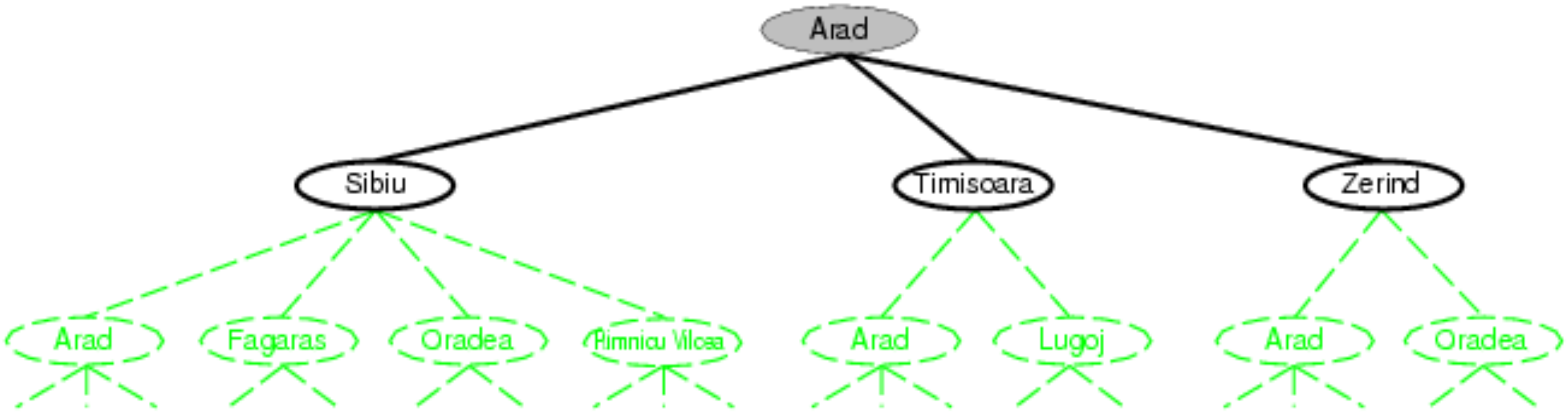
Tree-based Search

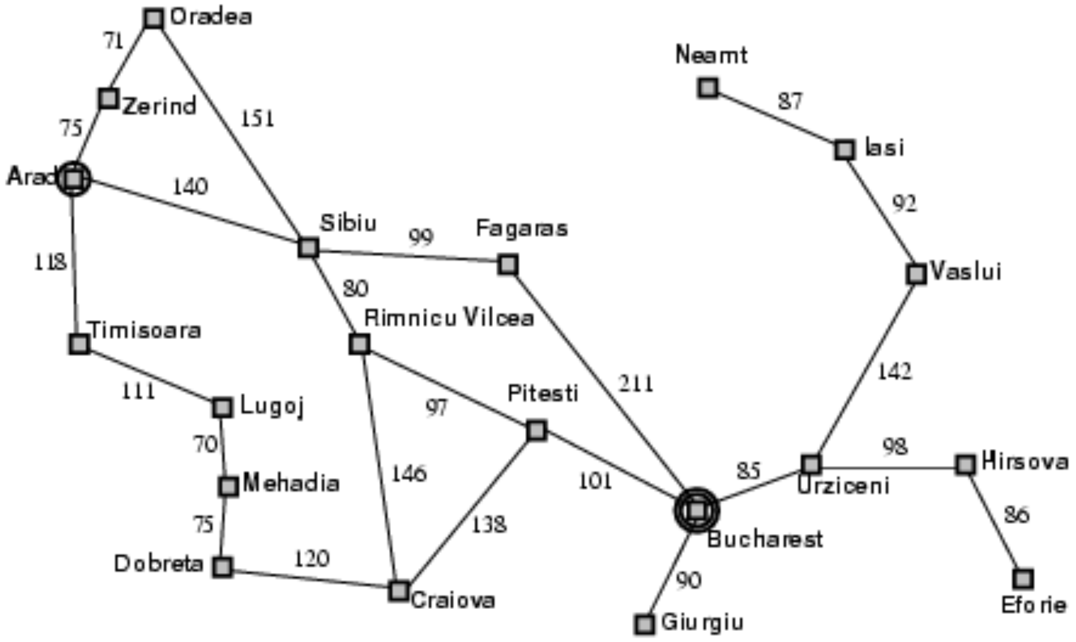
- Basic idea:
  - Exploration of state space by generating successors of already-explored states (a.k.a. expanding states).

  - Every state is evaluated: *is it a goal state*?

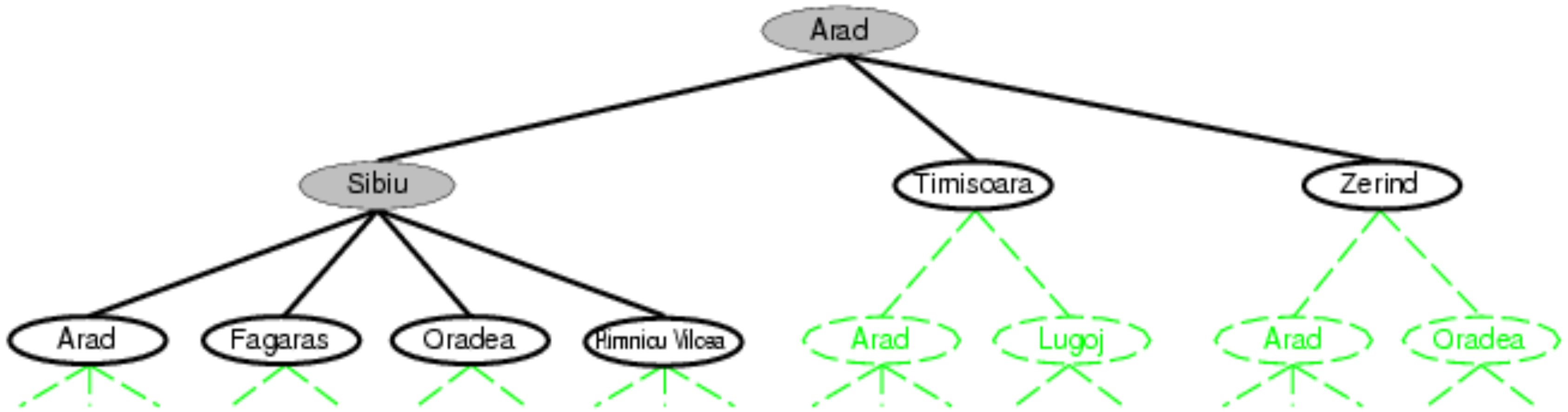
- In practice, the solution space can be a graph, not a tree
  - E.g., 8-puzzle
  - More general approach is graph search
  - Tree search can end up repeatedly visiting the same nodes
    - Unless it keeps track of all nodes visited
    - …but this could take vast amounts of memory

Tree Search Example





Tree Search Example



Tree Search

```
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
```

Graph Search

```
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
```

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 (optimal) 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 depth of the state space (may be ∞)