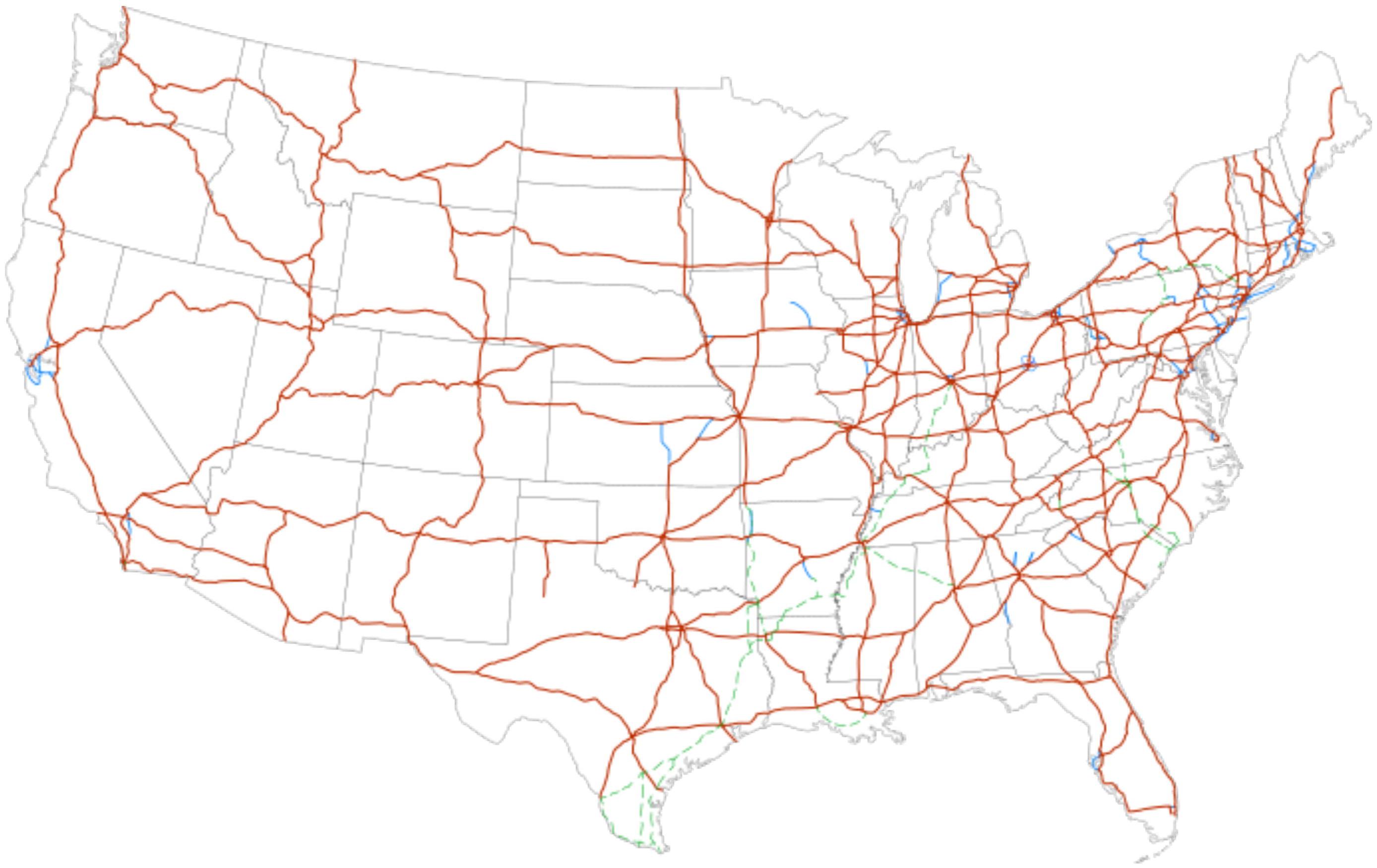
CS440/ECE 448 Lecture 2: Search Intro

Slides by Svetlana Lazebnik, 9/2016

Modified by Mark Hasegawa-Johnson, 1/2020

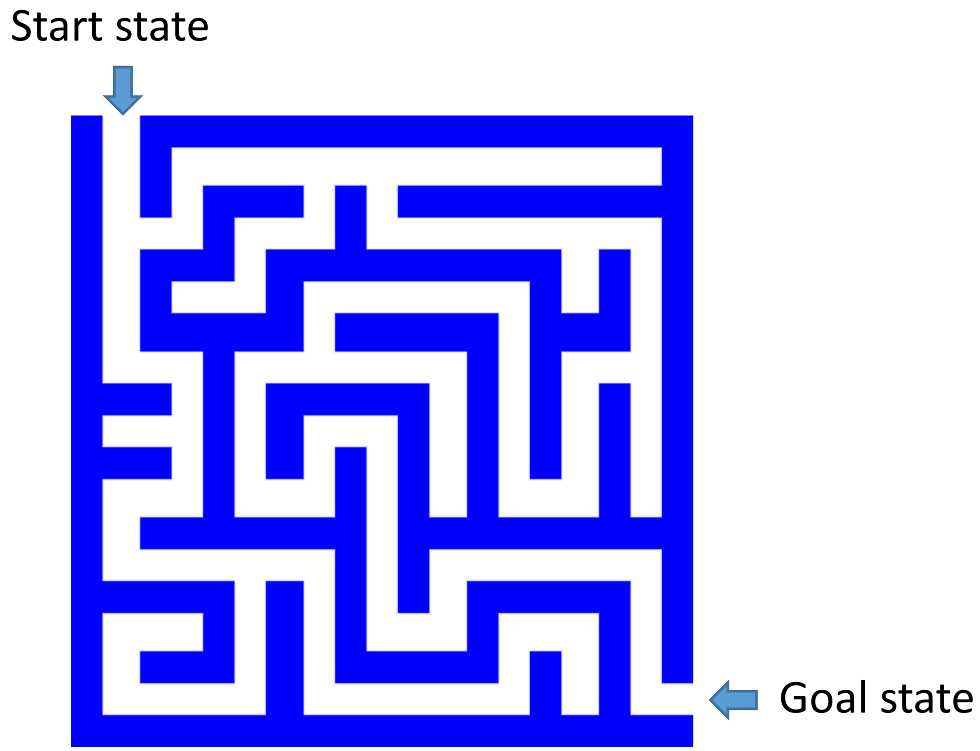


Outline of today’s lecture

1. How to turn ANY problem into a SEARCH problem:
    1. Initial state, goal state, transition model
    2. Actions, path cost
2. General algorithm for solving search problems
    1. First data structure: a frontier list
    2. Second data structure: a search tree
    3. Third data structure: a “visited states” list

Search

- We will consider the problem of designing **goal-based agents** in **fully observable, deterministic, discrete, static, known** environments



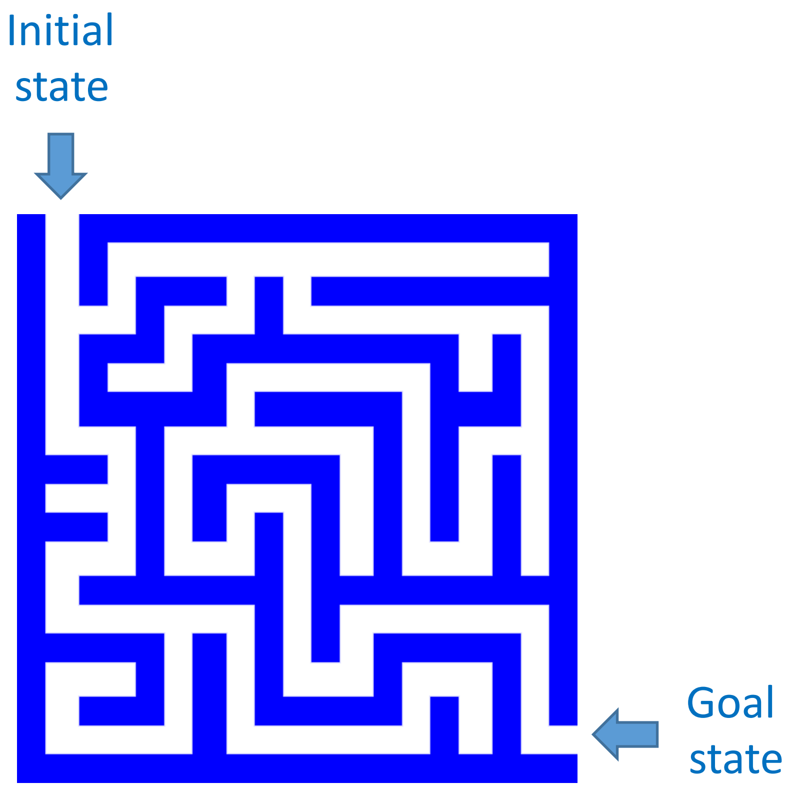
Search

We will consider the problem of designing **goal-based agents** in **fully observable, deterministic, discrete, known** environments

- The agent must find a *sequence of actions* that reaches the goal
- The **performance measure** is defined by (a) reaching the goal and (b) how "expensive" the path to the goal is
- We are focused on the process of finding the solution; while executing the solution, we assume that the agent can safely ignore its percepts (**static environment, open-loop system**)

Search problem components

- **Initial state**
- **Actions**
- **Transition model**
  - What state results from performing a given action in a given state?
- **Goal state**
- **Path cost**
  - Assume that it is a sum of nonnegative *step costs*



- The **optimal solution** is the sequence of actions that gives the *lowest* path cost for reaching the goal

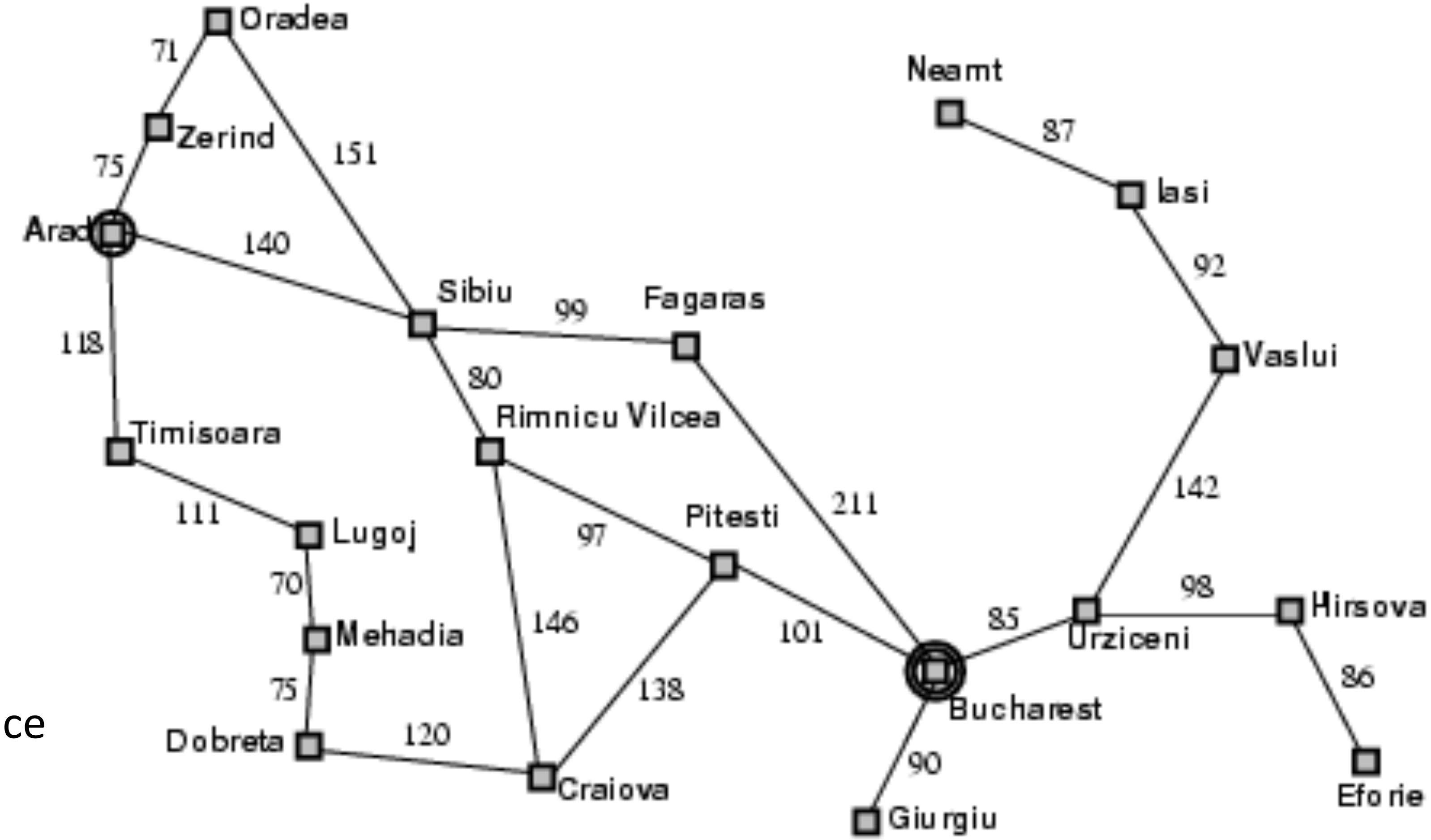
Knowledge Representation: State

- State = description of the world
  - Must have enough detail to decide whether or not you’re currently in the **initial state**
  - Must have enough detail to decide whether or not you’ve reached the **goal state**
  - Often but not always: “defining the state” and “defining the transition model” are the same thing

Example: Romania

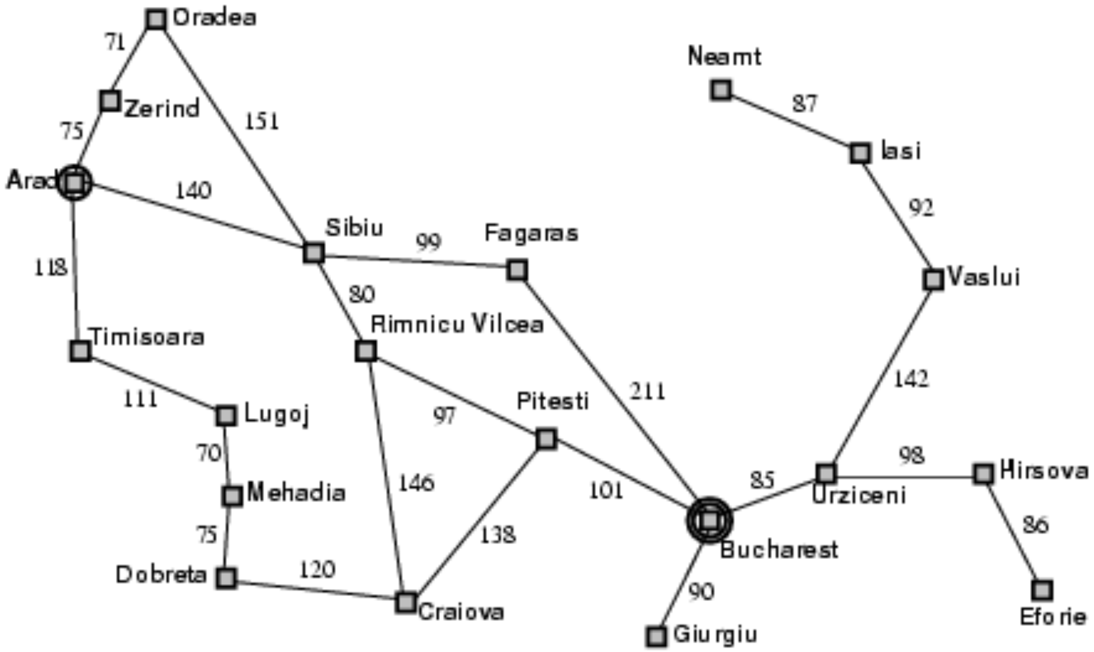
- On vacation in Romania; currently in Arad
- Flight leaves tomorrow from Bucharest

- **Initial state**
  - Arad
- **Actions**
  - Go from one city to another
- **Transition model**
  - If you go from city A to city B, you end up in city B
- **Goal state**
  - Bucharest
- **Path cost**
  - Sum of edge costs (total distance traveled)



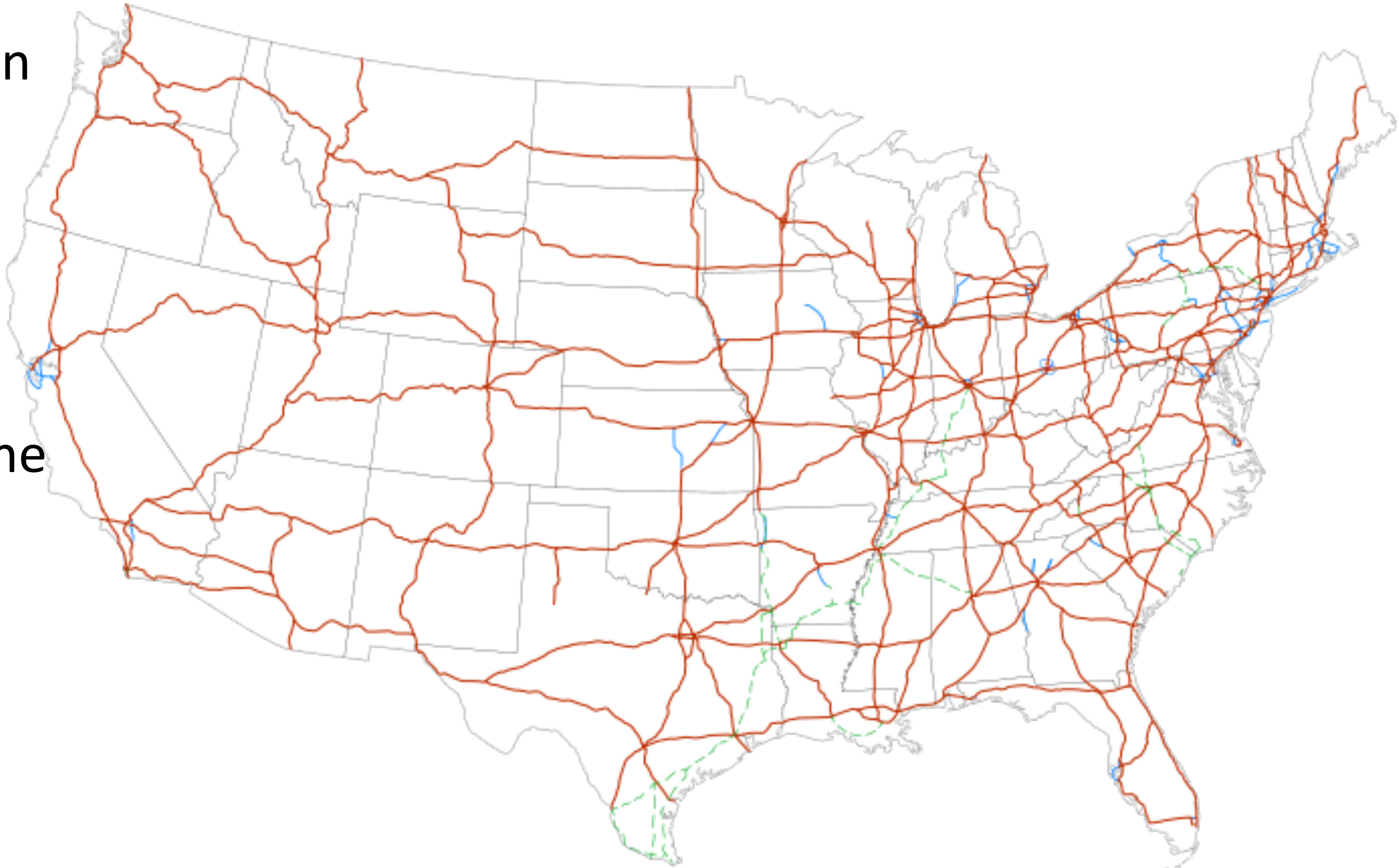
State space

- 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
- What is the state space for the Romania problem?



Traveling Salesman Problem

- Goal: visit every city in the United States
- Path cost: total miles traveled
- Initial state: Champaign, IL
- Action: travel from one city to another
- Transition model: when you visit a city, mark it as “visited.”



Complexity of the State Space

- State Space of Romania problem: size = # cities
  - State space is linear in the size of the world
  - A search algorithm that examines every possible state is reasonable
- State Space of Traveling Salesman problem: size = 2^(#cities)
  - State space is exponential in the size of the world
  - A search algorithm that examines every possible state is unreasonable

Outline of today's lecture

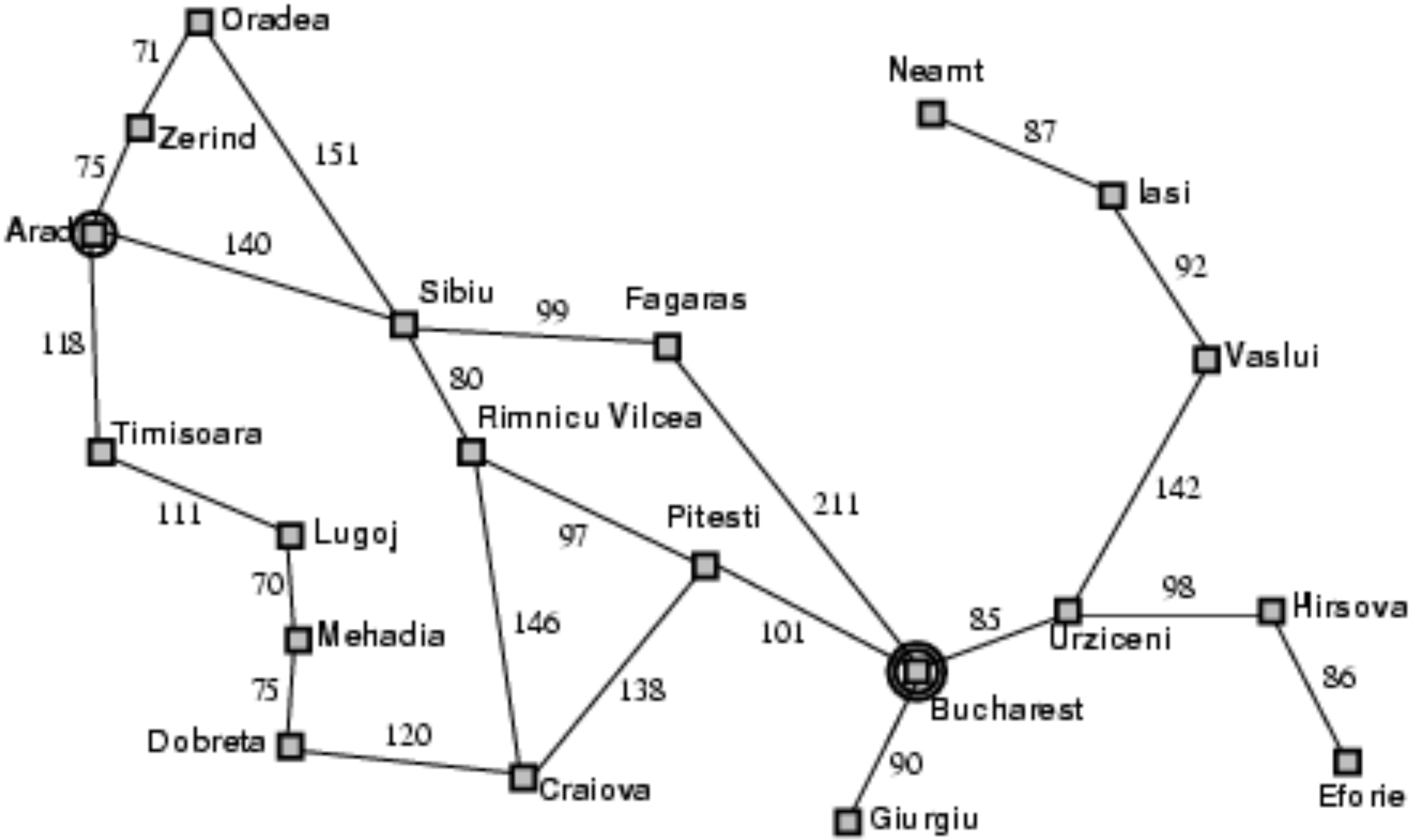
1. How to turn ANY problem into a SEARCH problem:
    1. Initial state, goal state, transition model
    2. Actions, path cost
2. General algorithm for solving search problems
    1. First data structure: frontier (a set)
    2. Second data structure: a search tree (a directed graph)
    3. Third data structure: explored (a dictionary)

First data structure: Frontier Set

- Frontier set = set of states that you know how to reach, but you haven’t yet tested to see what comes next after those states
- Initially: FRONTIER = { initial_state }
- First step in the search: figure out which states you can reach from the initial_state, add them to the FRONTIER

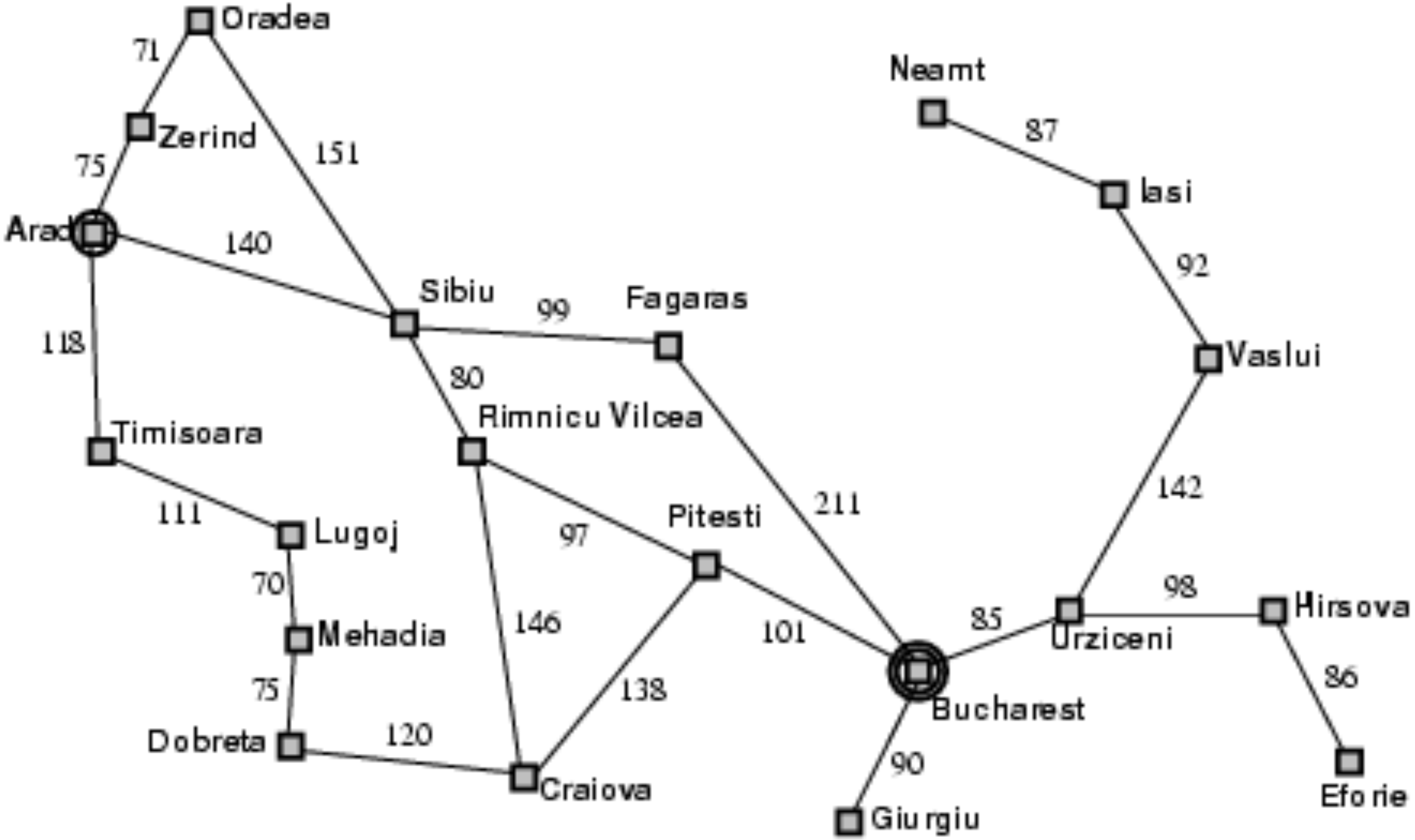
Search step 0

Frontier = { Arad }



Search step 1

Frontier = { Sibiu, Timisoara, Zerind }



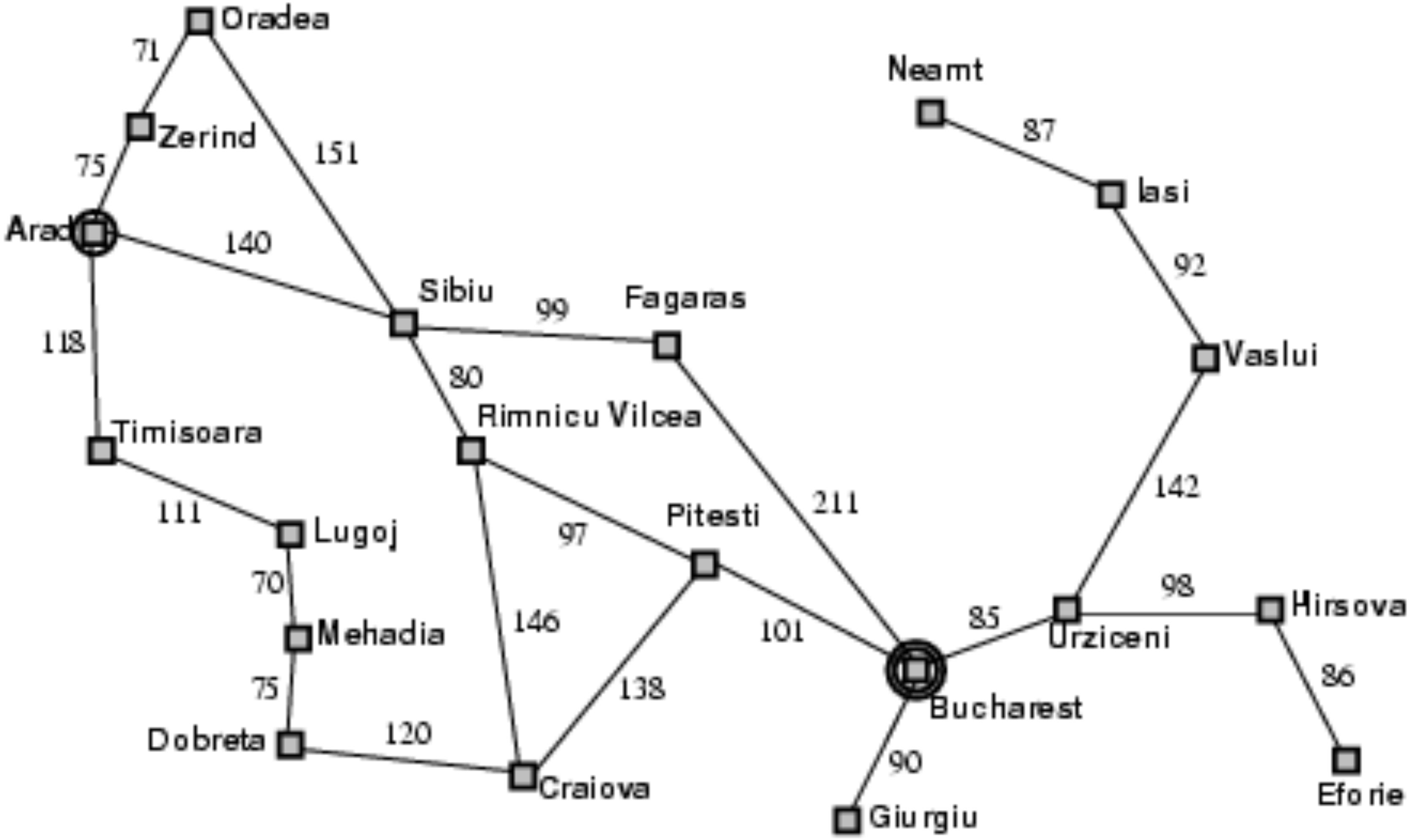
Second data structure: Search Tree

- Tree = directed graph of nodes
- Node = (world_state, parent_node, path_cost)

Search step 0

Frontier: { Arad }

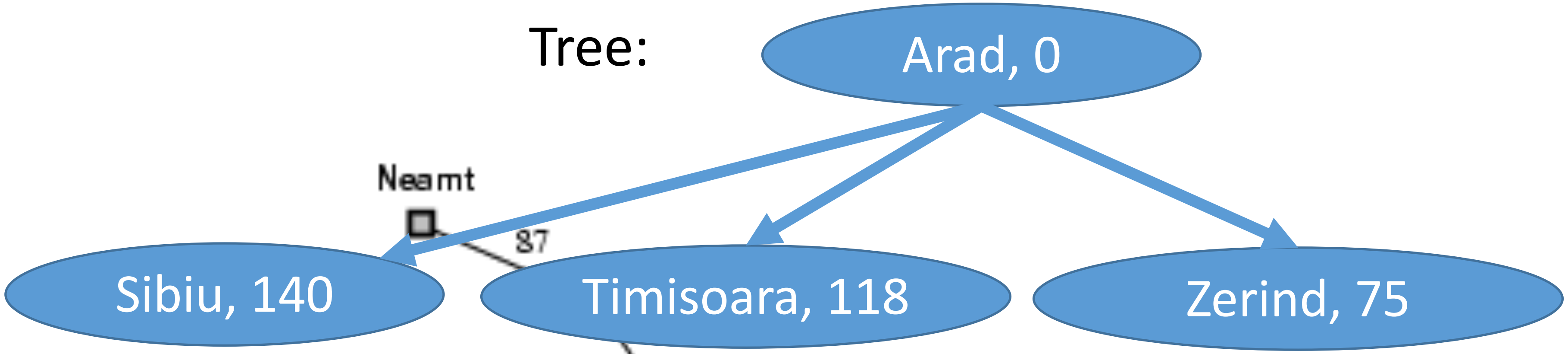
Tree:

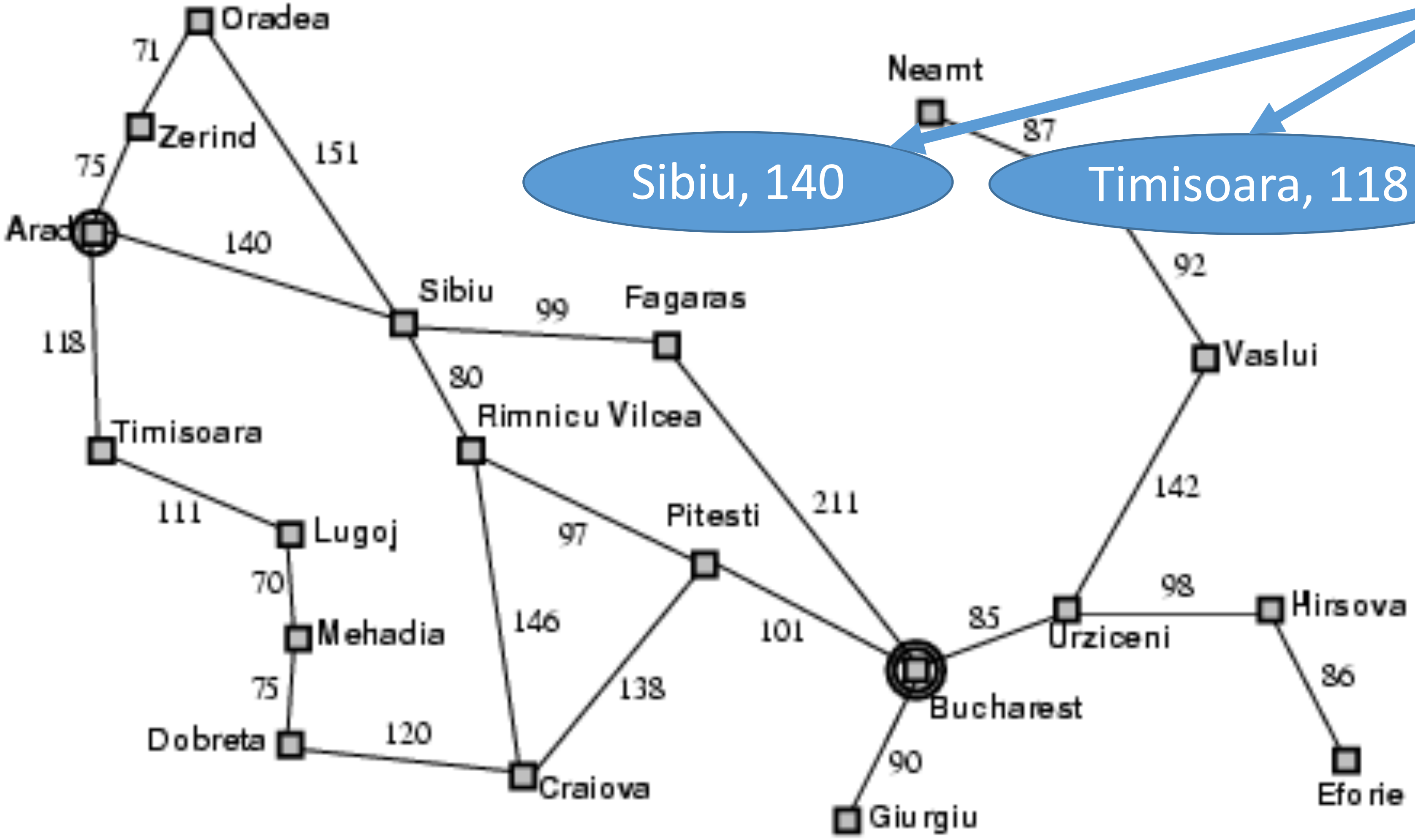


Search step 1

Frontier: { Sibiu, Zerind, Timisoara }

Tree:



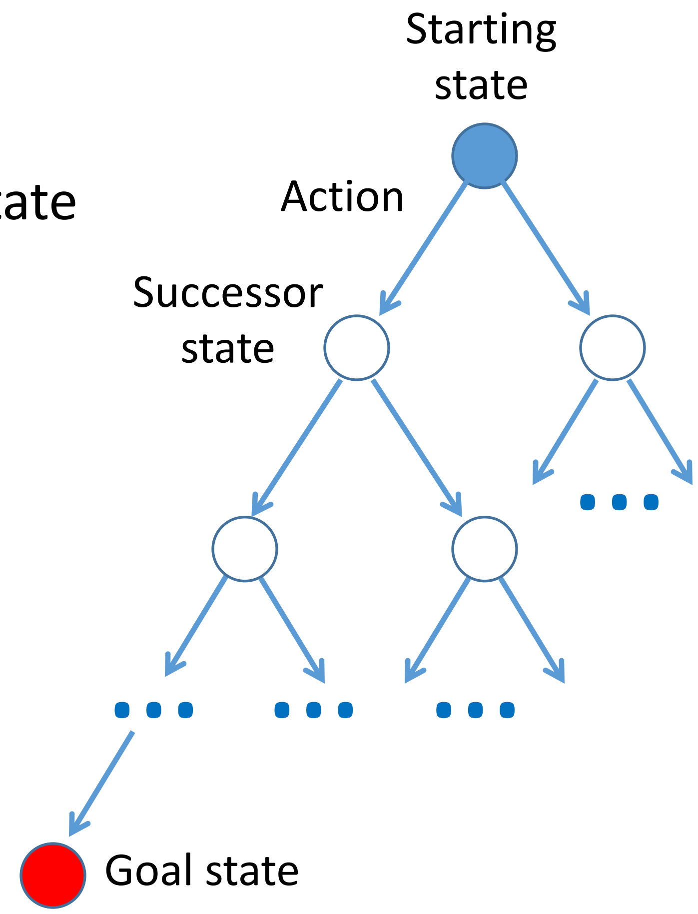


Tree Search: Basic idea

1. SEARCH for an optimal solution
   - Maintain a **frontier** of unexpanded states, and a **tree** showing all known paths
   - At each step, pick a state from the frontier to **expand:**
     - Check to see whether or not this state is the goal state. If so, DONE!
     - If not, then list all of the states you can reach from this state, add them to the frontier, and add them to the tree
2. BACK-TRACE: go back up the tree; list, in reverse order, all of the actions you need to perform in order to reach the goal state.
3. ACT: the agent reads off the sequence of necessary actions, in order, and does them.

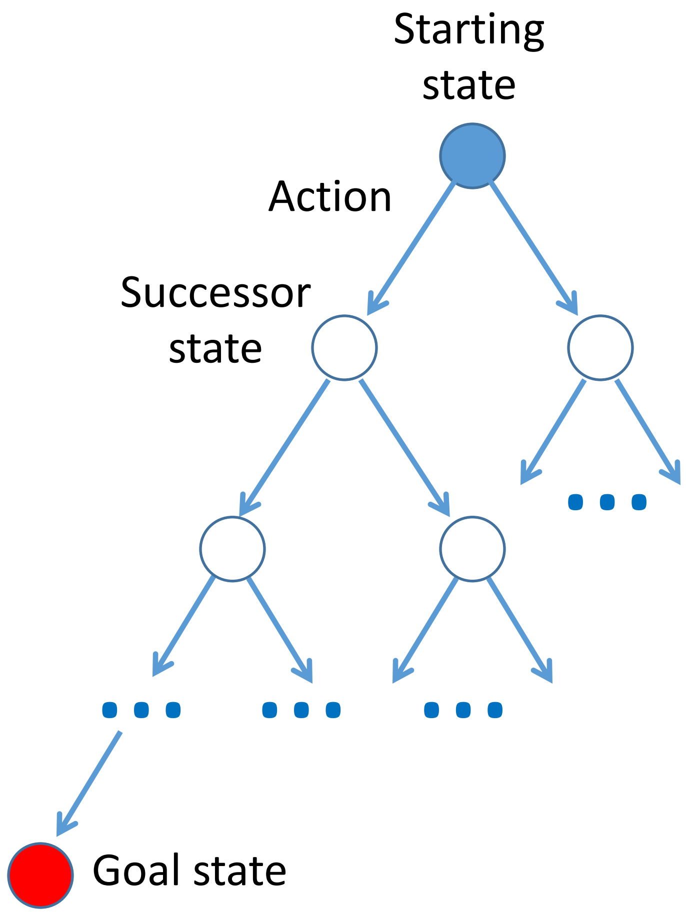
Search Tree

- "What if" tree of sequences of 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 the world, while a node is a data structure that is part of the search tree
    - Node has to keep pointer to parent, path cost, possibly other info



Nodes vs. States

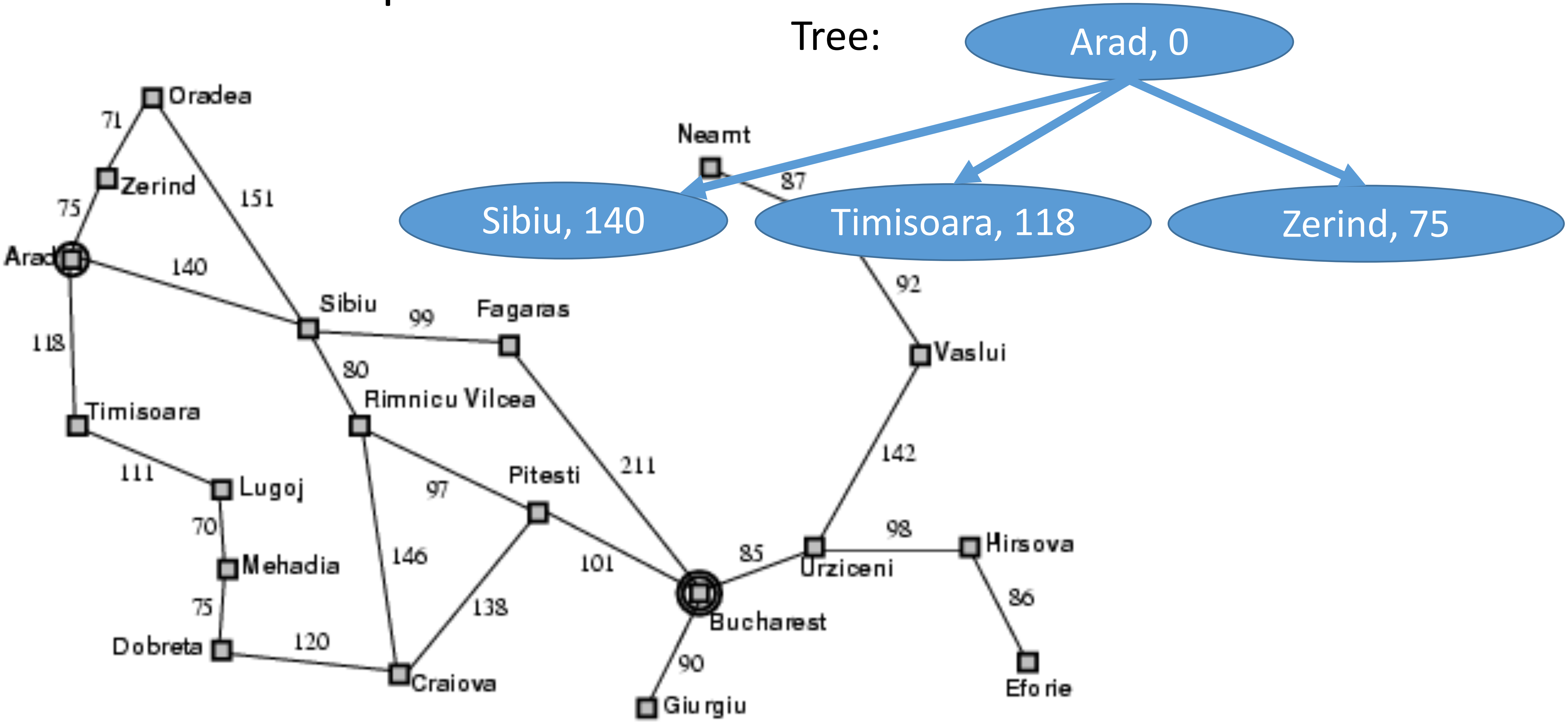
- State = description of the world
  - Must have enough detail to decide whether or not you're currently in the initial state
  - Must have enough detail to decide whether or not you've reached the goal state
  - Often but not always: "defining the state" and "defining the transition model" are the same thing
- Node = a point in the search tree
  - Knows the ID of its STATE
  - Knows the ID of its PARENT NODE
  - Knows the COST of the path



Search step 1

Frontier: { Sibiu, Zerind, Timisoara }

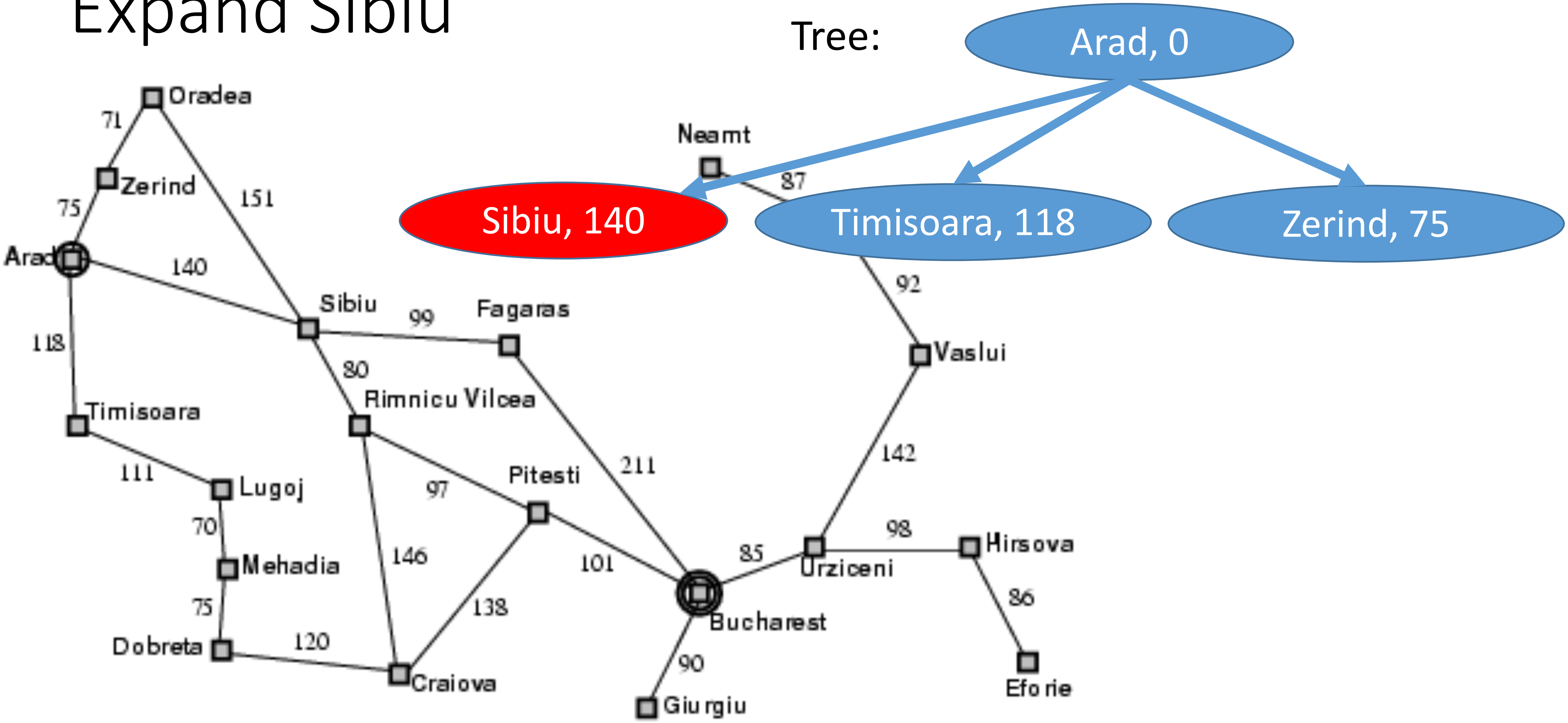
Tree:



Search step 2
Expand Sibiu

Frontier: { Sibiu, Zerind, Timisoara }

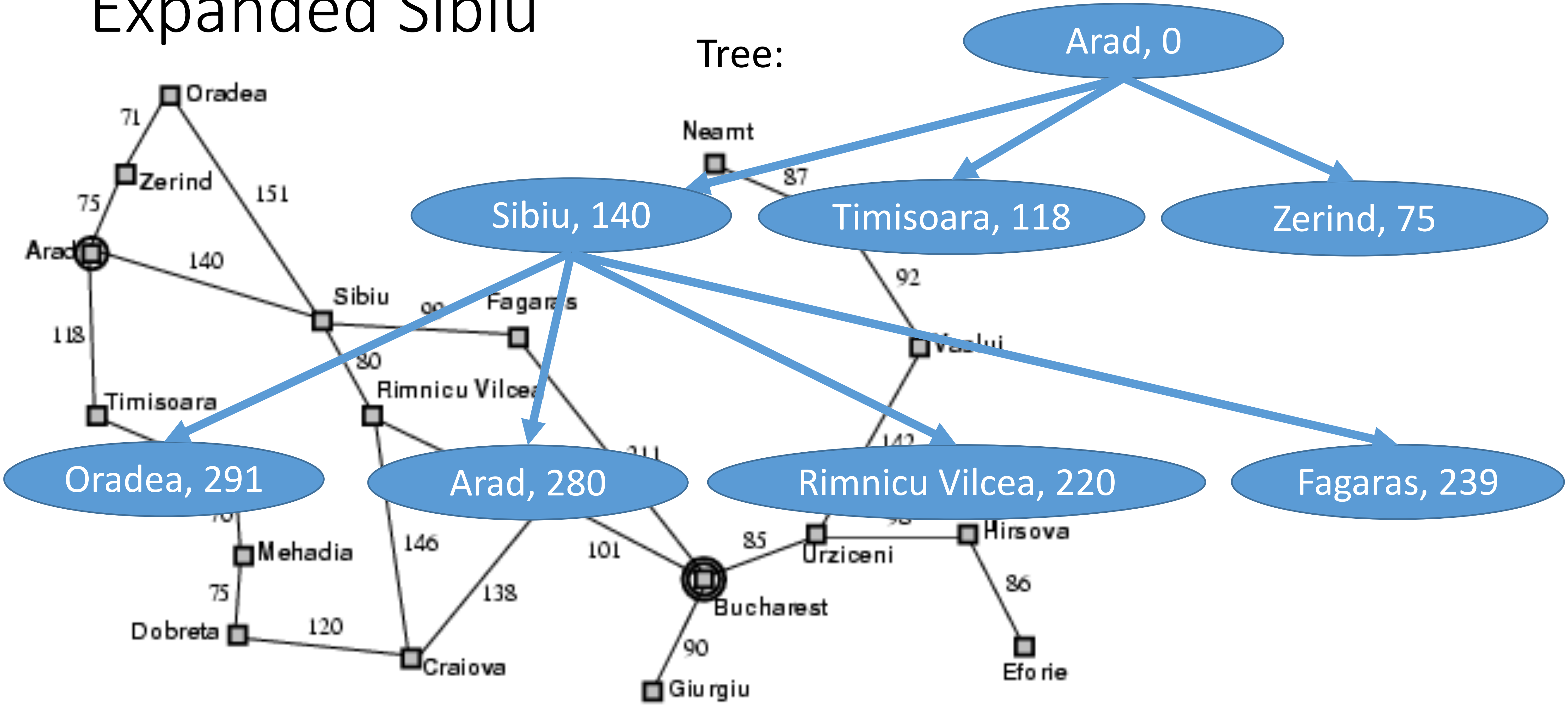
Tree:



Search step 2
Expanded Sibiu

Frontier: { Zerind, Timisoara, Oradea, Arad, Rimnicu Vilcea, Fagaras }

Tree:



Tree Search: Computational Complexity

Without an EXPLORED set

- b = “branching factor” = max # states you can reach from any given state
- d = “depth” = # layers in the tree (# moves that you have made)
- Without an explored set: complexity = O{b^d}

Solution: keep track of the states you have explored

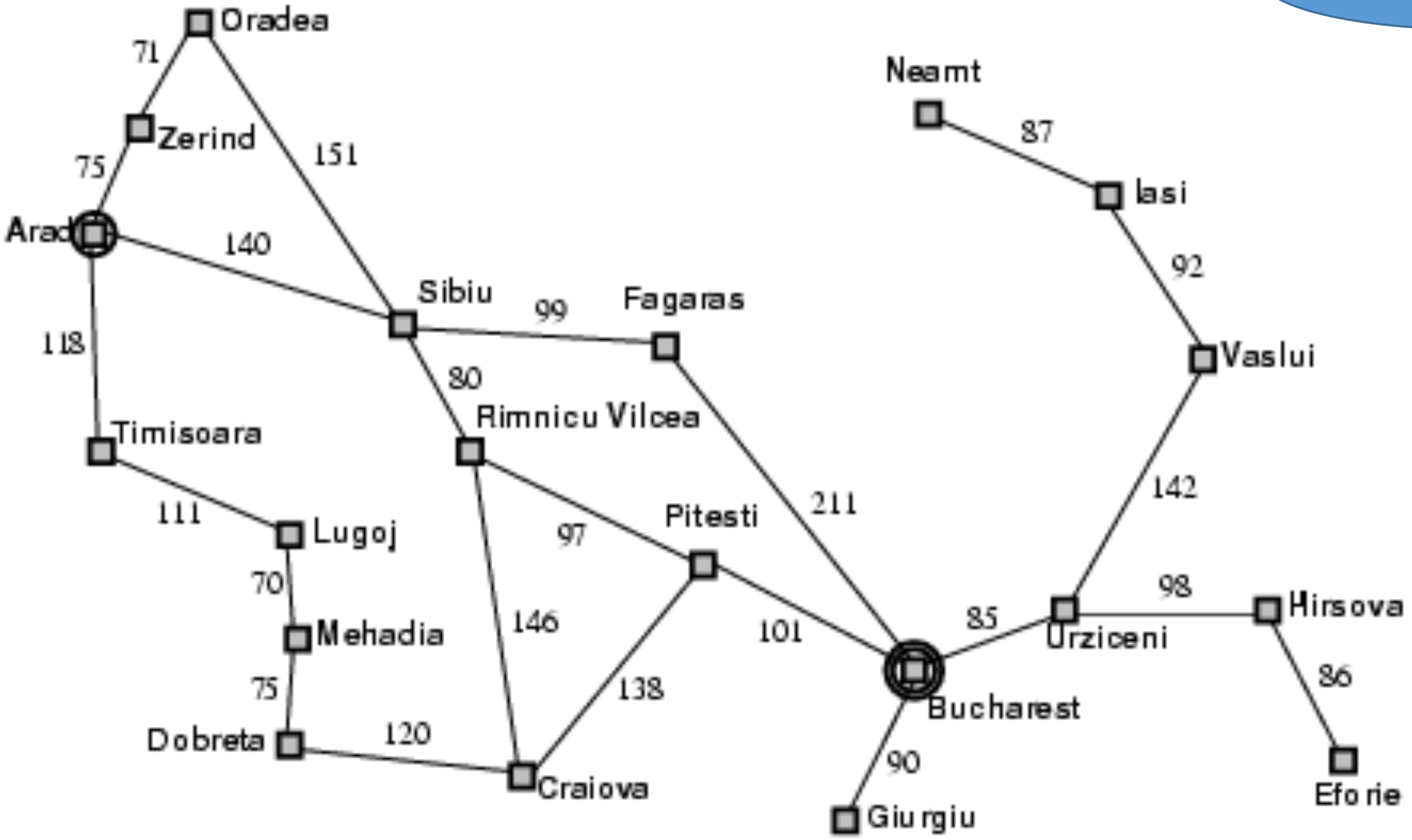
- When you expand a state, you get the list of its possible child states
- ONLY IF a child state is not already explored, put it on the frontier, and put it on the explored set.
- Result: complexity = min(O{b^d}, O{# possible world states})

Search step 0

Frontier: { Arad }

Explored: { Arad }

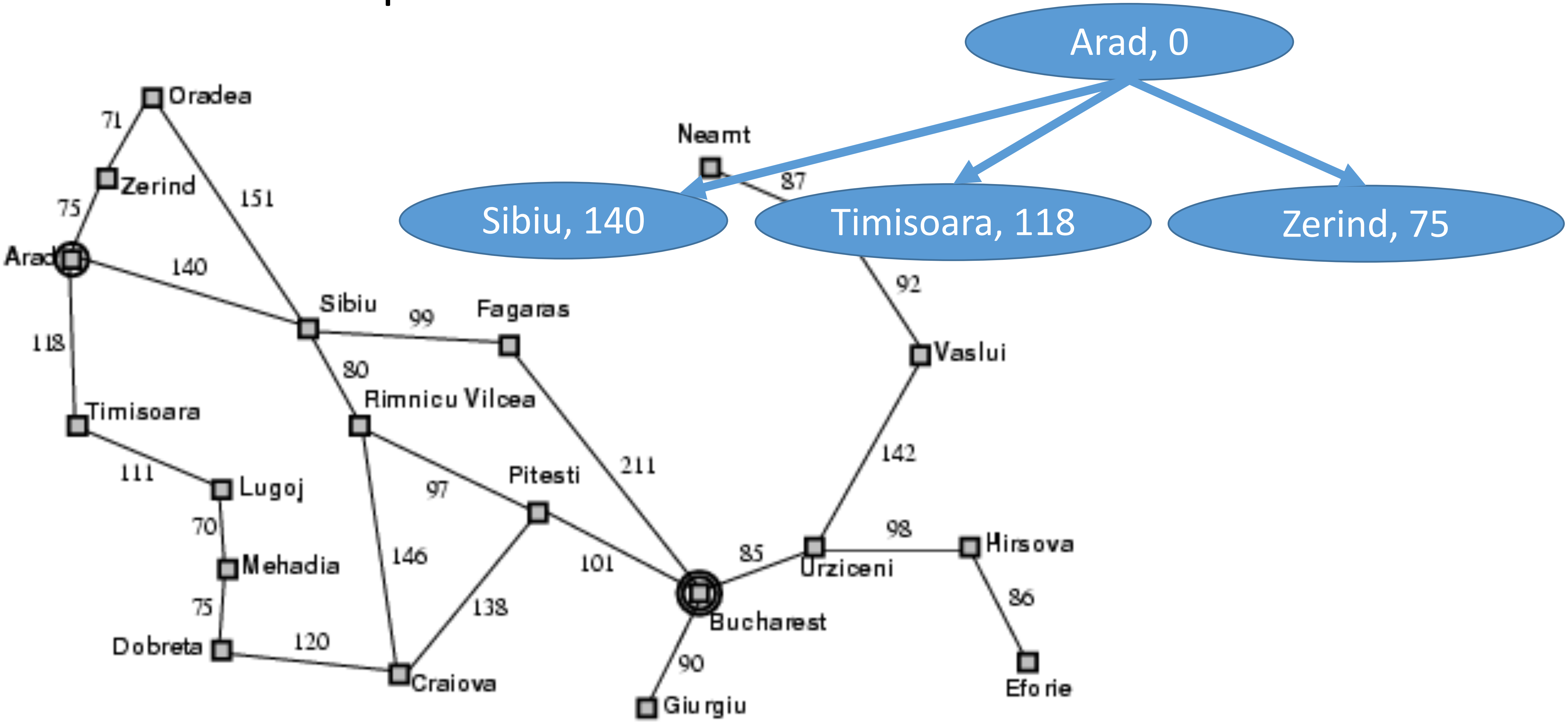
Tree:



Frontier: { Sibiu, Zerind, Timisoara }

Explored: { Arad, Sibiu, Zerind, Timisoara }

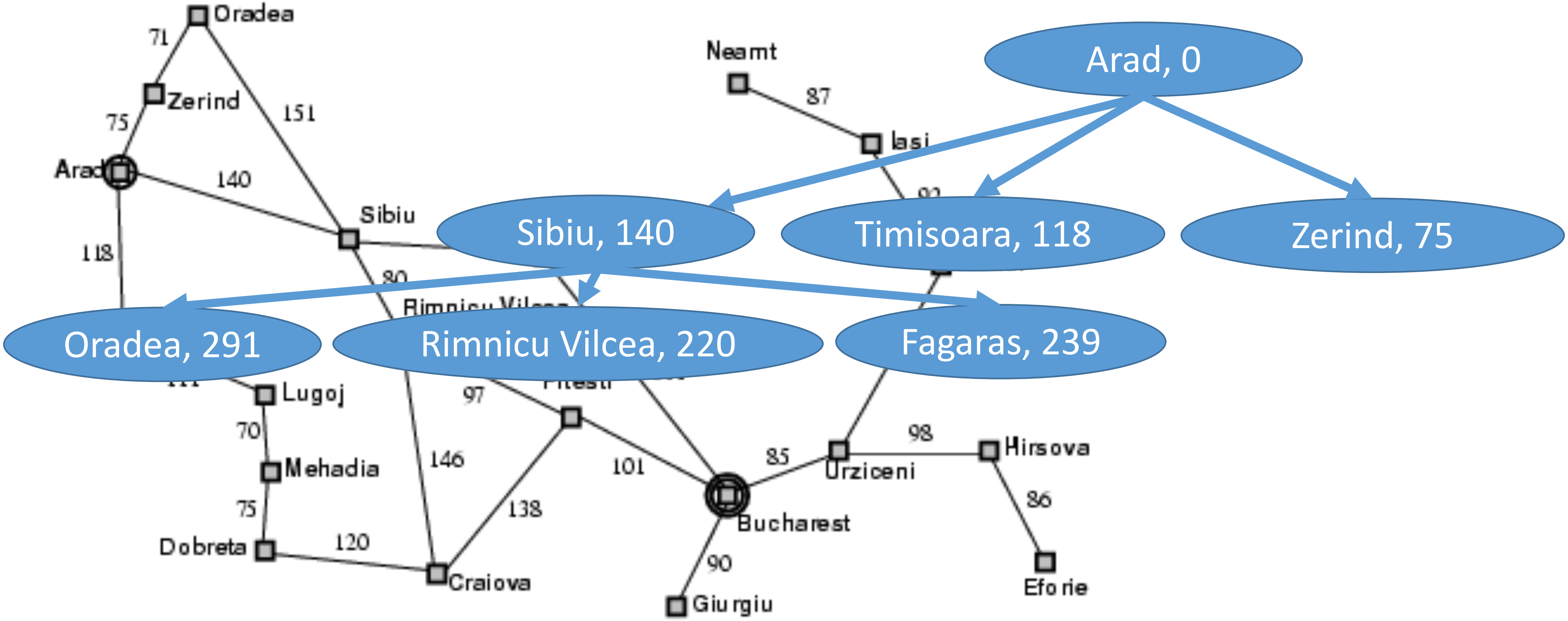
Search step 1



Search step 2

Frontier: { Zerind, Timisoara, Oradea, Rimnicu Vilcea, Fagaras }

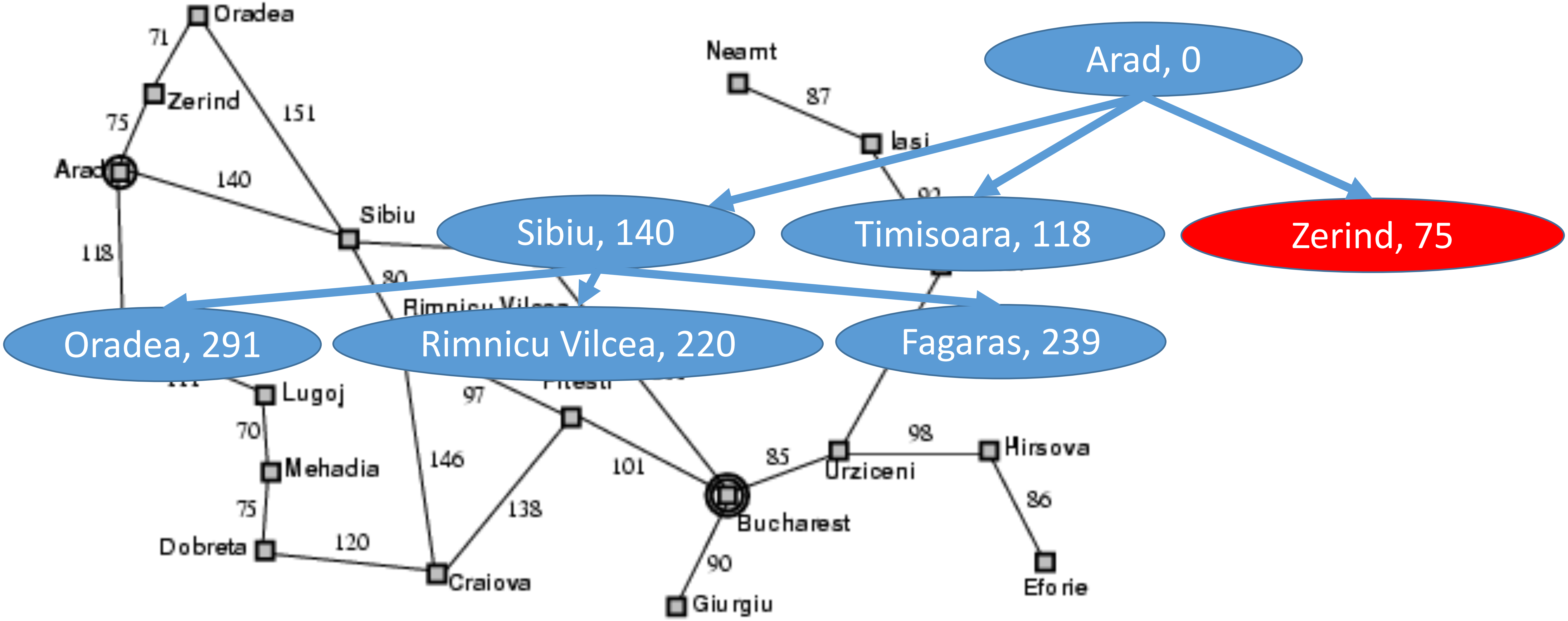
Explored: { Arad, Sibiu, Zerind, Timisoara, Oradea, Rimnicu Vilcea, Fagaras }



Search step 3: expand Zerind

Frontier: { **Zerind**, Timisoara, Oradea, Rimnicu Vilcea, Fagaras }

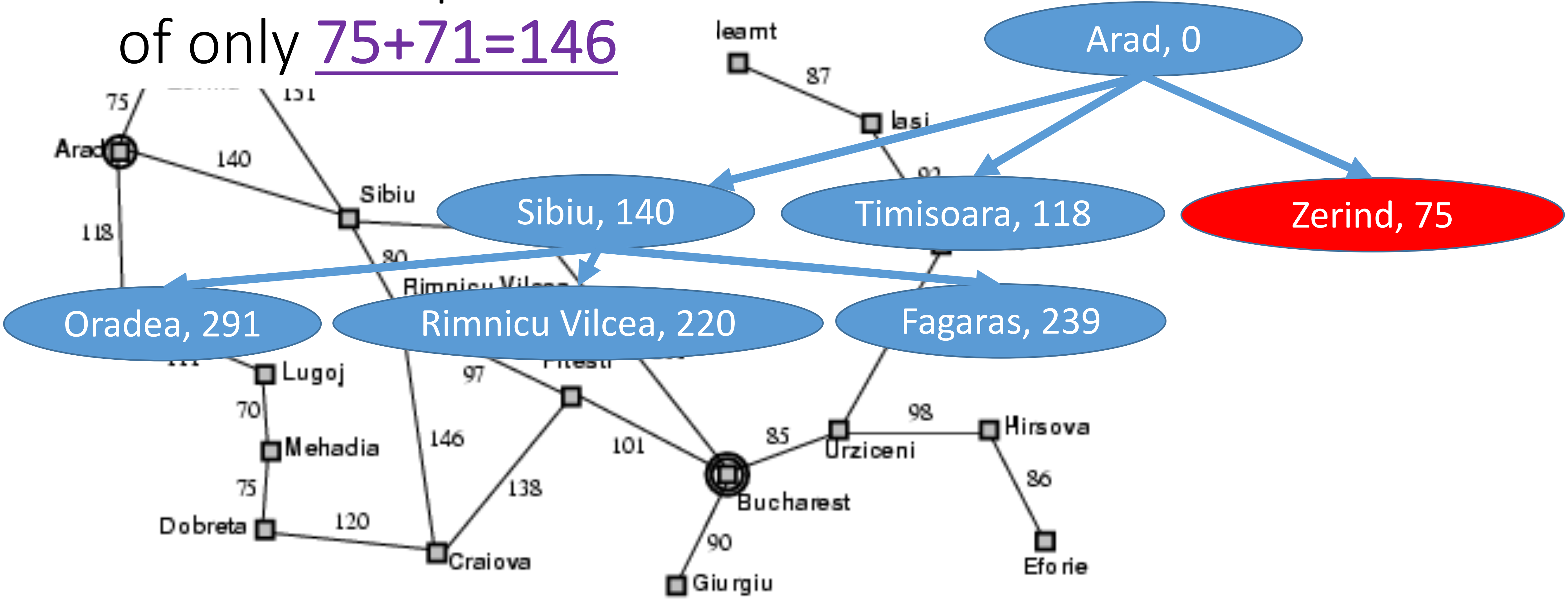
Explored: { Arad, Sibiu, Zerind, Timisoara, Oradea, Rimnicu Vilcea, Fagaras }



Search step 3:
we can reach Oradea with a total path cost of only 75+71=146

Frontier: { **Zerind**, Timisoara, Oradea, Rimnicu Vilcea, Fagaras }

Explored: { Arad, Sibiu, Zerind, Timisoara, Oradea, Rimnicu Vilcea, Fagaras }



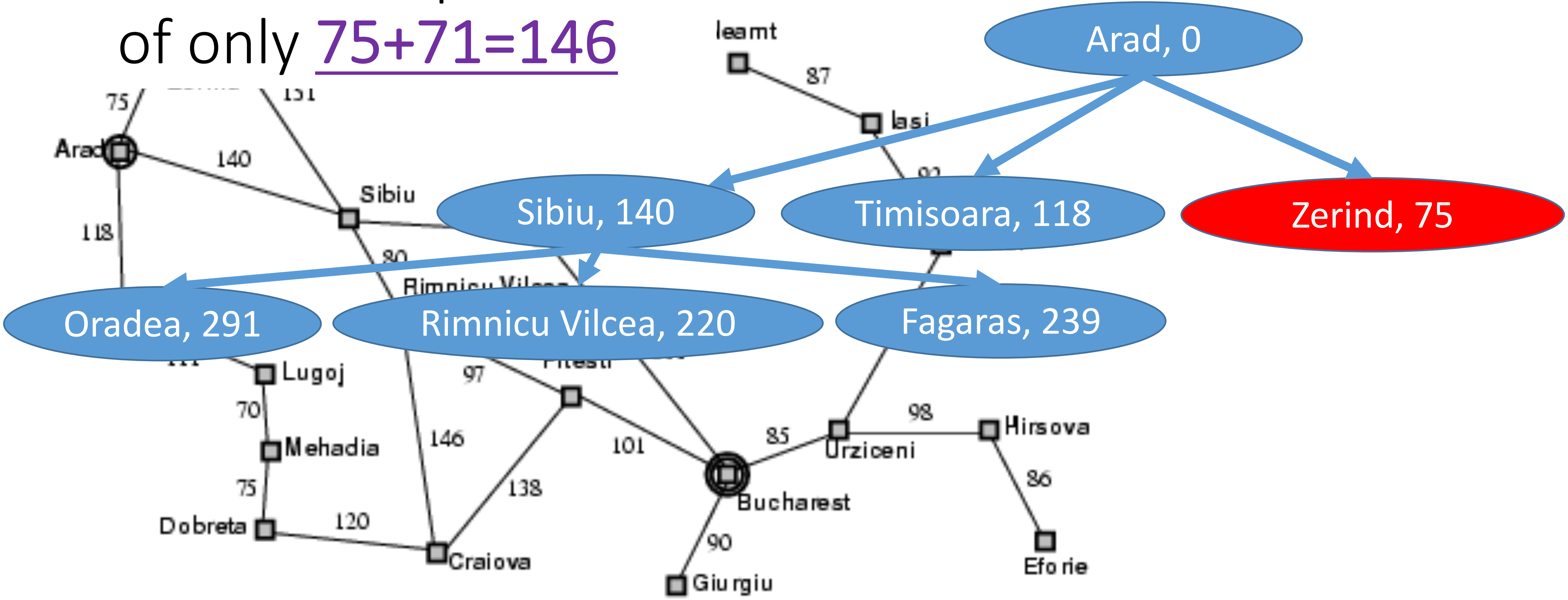
Third data structure: Explored Dictionary

- Explored = dictionary mapping from state ID to path cost
- If we find a new path to the same state, with HIGHER COST, then we ignore it
- If we find a new path to the same state, with LOWER COST, then we expand the new path

Search step 3:
we can reach Oradea with a total path cost of only <u>75+71=146</u>

Frontier: { **<u>Zerind:75</u>**, Timisoara:118, Oradea:291, Rimnicu Vilcea:220, Fagaras:239 }

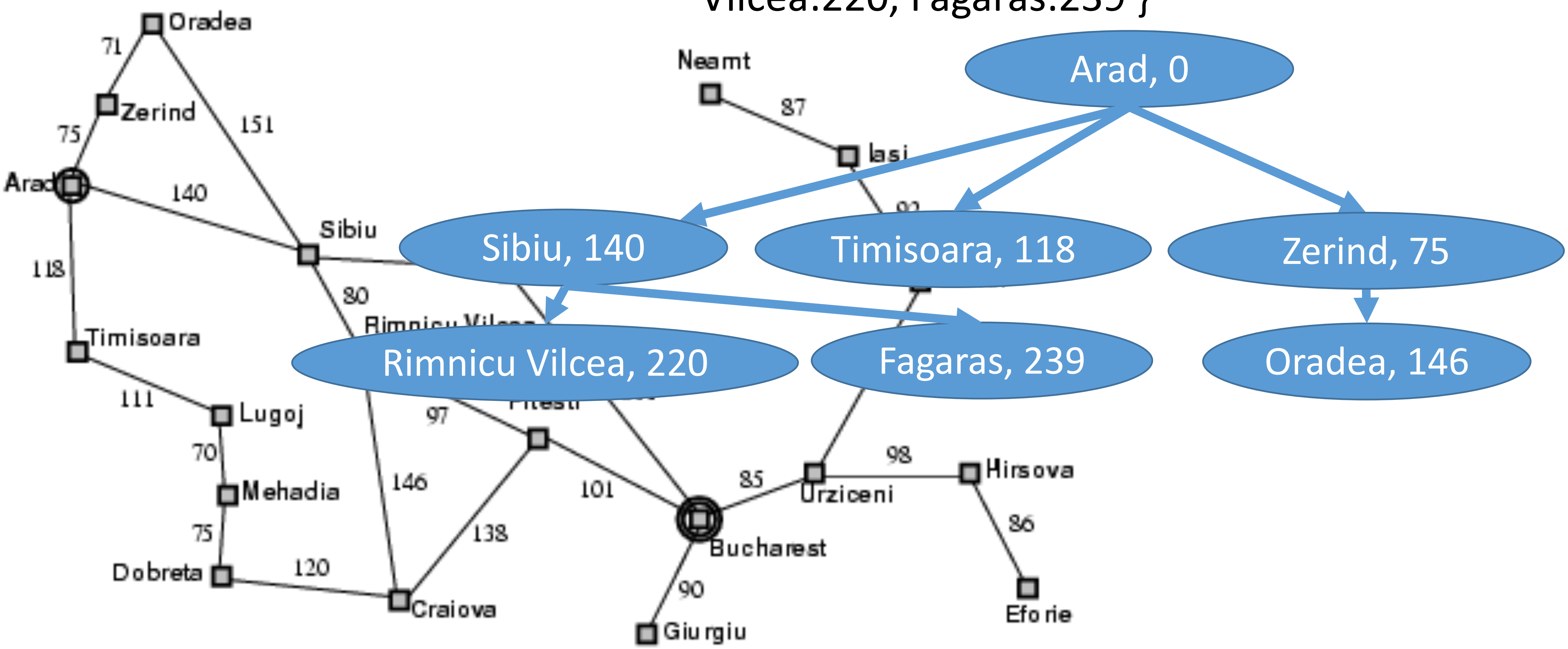
Explored: { Arad:0, Sibiu:140, Zerind:75, Timisoara:118, Oradea:291, Rimnicu Vilcea:220, Fagaras:239 }



Search step 3: expanded Zerind

Frontier: { Timisoara:118, Oradea:**146**, Rimnicu Vilcea:220, Fagaras:239 }

Explored: { Arad:0, Sibiu:140, Zerind:75, Timisoara:118, Oradea:**146**, Rimnicu Vilcea:220, Fagaras:239 }



Tree Search: Basic idea

At each step, pick a state from the frontier to **expand:**

1. Check to see whether or not this state is the goal state. If so, DONE! If not, then for each child:
2. Check to see whether this child is already in the explored set with a LOWER COST. If so, ignore it. If not:
3. Add it to the frontier, to the tree, and to the explored dict.

Complexity = min(O{b^d}, O{# possible world states}).

Next time: how can we limit d?