Lecture #11 Part A: Planning and Search Methods

COMP3608: Intelligent Systems

Inzamam Rahaman

The Need to Plan

- Planning is fundamental human activity
- Given some starting “position” want to series of actions (a path) to some goal “position”
  - Sometimes we have numerous paths, but have a notion of some “best” path
  - Example: want to get from your house to UWI in the least amount of time
  - Example: want to go from a particular shuffle of a Rubik’s cube to a solution state of a Rubik’s cube in the smallest number of steps

Planning

- Start at some start state S_0
- Have a set of goal states G
  - Often G contains only one element
- Want to take a finite series of actions (find a path) $A_1, A_2, \ldots A_n$ such that
  - $\text{do\_action}(S_i, A_i) = S_{i+1}$
  - $\text{do\_action}(S_n, A_n) \in G$ (we arrive at a state in G)
- If no such path exists, we also want to report failure

Planning

- Some of the aforementioned terminology should seem familiar to you
- Remember Data Structures
- What sort of techniques from DS can we use here?

Planning

- Some of the aforementioned terminology should seem familiar to you
- Remember Data Structures
- What sort of techniques from DS can we use here?

Planning as Graph Traversal

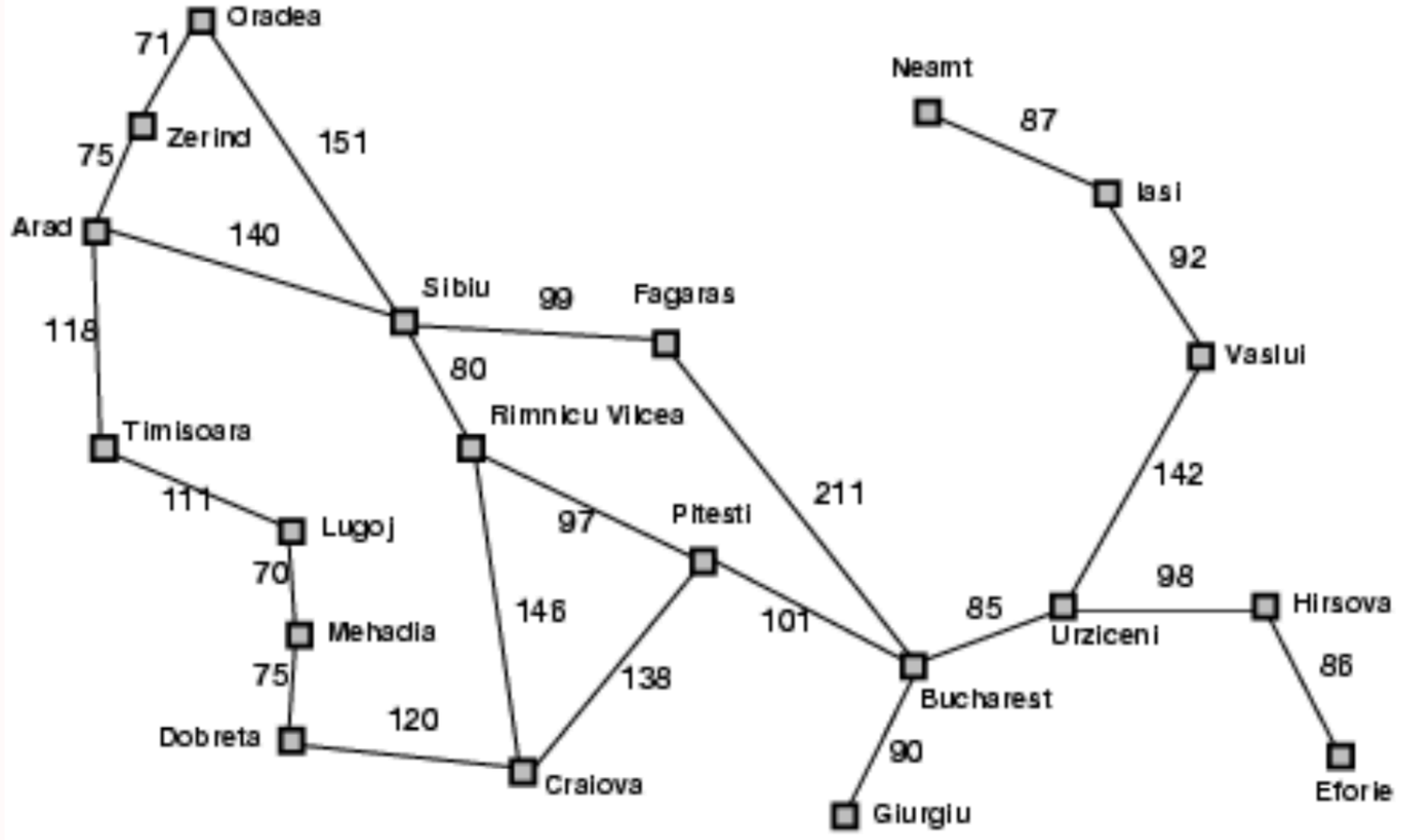
- We can represent the state space as a (potentially infinite) graph
- Available actions from a state to another state are represented by edges
- Travel from state to state through edges in a structured, principled way
- Hoping to reach goal state (goal node)
- Can leverage typical graph traversal algorithms to plan
  - Will even look at a variant of Dijkstra's algorithm

Terminology - Search Algorithm Taxonomy

- Uninformed Search: algorithm only has access to graph topology
  - DFS, BFS, IDS
- Informed Search: algorithm has access to graph topology and side information
  - Best-first, A* heuristic search

Terminology - Search Algorithm Properties

- Action (Edge) cost - the cost of taking a particular action from a particular state
  - Default cost of 1
- Path cost - total cost of an entire path
  - Default cost is the number of actions taken
- Optimality - a search algorithm is optimal if it is guaranteed to find the lowest cost solution if it exists
- Completeness - a search algorithm is complete if it is guaranteed to find a solution if one exists
- Space complexity - the amount of extra memory needed
- Time complexity - number of discrete basic operations needed



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Uninformed Search Methods

- BFS
- DFS
- Depth-limited Search

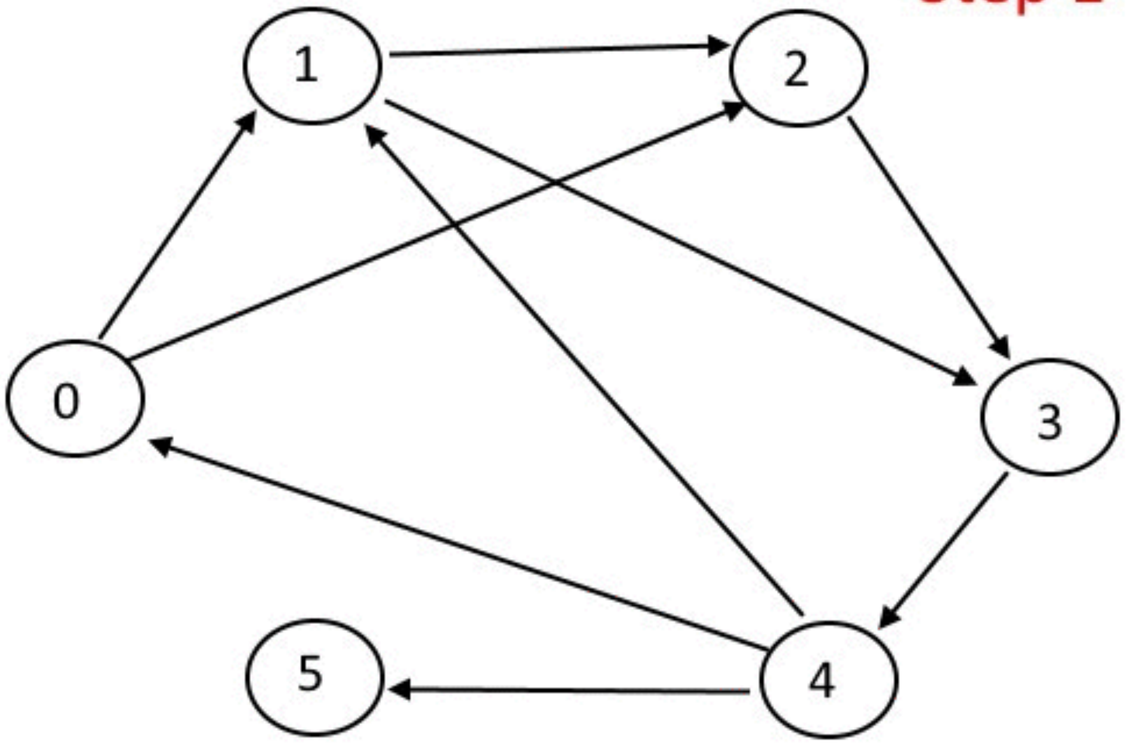
Breadth-first Search

- Children of start state are expanded and explored
- Then their children are expanded and explored
- Then their children are expanded and explored
- ….
- Until we either find the solution or report failure

```
def get_path(parents, goal):
    path = []
    current = goal
    while parents[current] is not None:
        path.push(current)
        current = parents[current]
    path.push(start)
    return path
```

```
def bfs(graph, start, goal):
    parents = {start: None}
    if start == goal:
        return get_path(parents)
    frontier = Queue()
    frontier.enqueue(start)
    explored = []
    while True:
        if frontier.is_empty():
            raise Failure
        current = frontier.dequeue()
        explored.add(current)
        for action in graph.edges_from(current):
            child = do_action(action, current)
            if child not in explored and child not in frontier:
                parents[child] = current
                if child == goal:
                    return get_path(parents)
                frontier.enqueue(child)
    raise Failure
```

Step 1



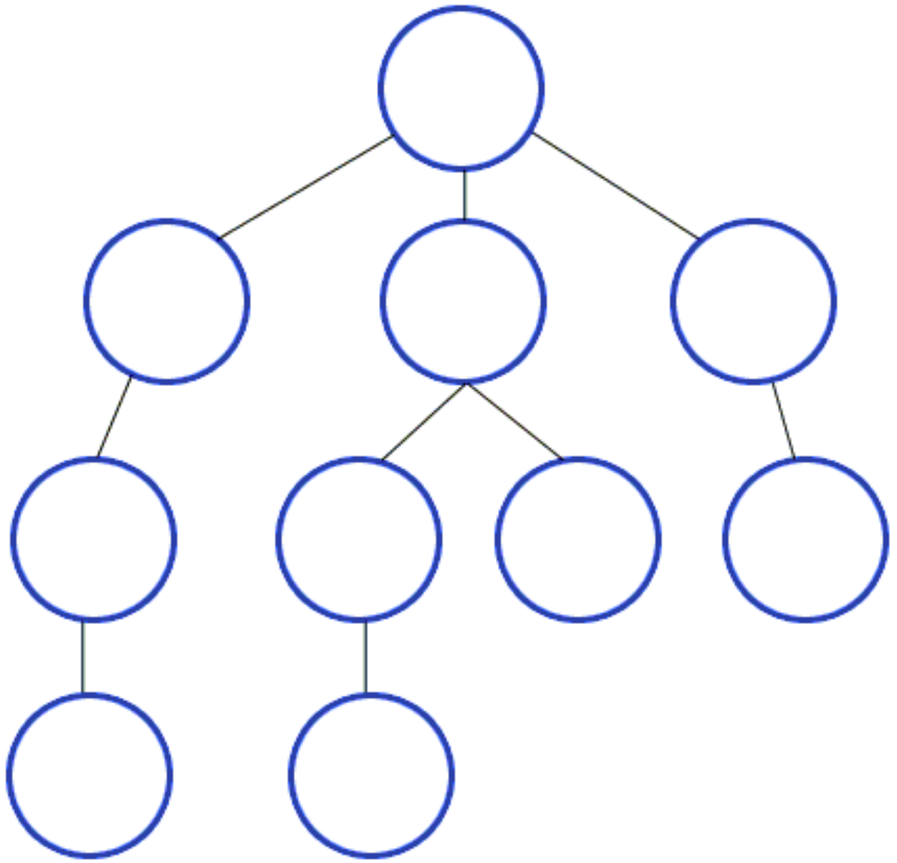
BFS

MakeAGIF.com

Depth-first search

- Instead of going level by level
- Traverse graph by exploring down paths as much as possible
- And then **backtracking** if we don't find a solution
- This is really just another name for a backtracking approach

```
def dfs(graph, start, goal):
    parents = {start: None}
    if start == goal:
        return get_path(parents)
    frontier = Stack()
    frontier.push(start)
    explored = []
    while True:
        if frontier.is_empty():
            raise Failure
        current = frontier.pop()
        explored.add(current)
        for action in graph.edges_from(current):
            child = do_action(action, current)
            if child not in explored and child not in frontier:
                parents[child] = current
                if child == goal:
                    return get_path(parents)
                frontier.push(child)
    raise Failure
```



Depth-Limited Search

- DFS might not converge in reasonable time if graph is infinite or very large
- Sometimes if the solution is very costly, it might as well not exist
- Core idea: traverse a total of d - called the limit - of edges away from the start node

```
def depth_limited_s(graph, start, goal, limit):
    parents = {start: None}
    levels = {start: 0}
    if start == goal:
        return get_path(parents)
    frontier = Stack()
    frontier.push(start)
    explored = []
    while True:
        if frontier.is_empty():
            raise Failure
        current = frontier.pop()
        if levels[current] <= limit:
            explored.add(current)
            for action in graph.edges_from(current):
                child = do_action(action, current)
                if child not in explored and child not in frontier:
                    parents[child] = current
                    if child == goal:
                        return get_path(parents)
                    frontier.push(child)
                    levels[child] = levels[current] + 1
    raise Failure
```

Informed Search Methods

- Suppose that we have some heuristic $h(s) : S \rightarrow \mathbb{R}$ that gives an approximate cost from a state to a goal state
- Obviously $h(s) = 0$ if $s \in G$
- Suppose $h^*(s) : S \rightarrow \mathbb{R}$ is the function that gives the **exact** cost from a state to a goal state
- A heuristic is called admissible if $\forall s \in S, h(s) \leq h^*(s)$
  - An admissible heuristic always underestimates the cost from state to a goal state
  - The informed search method we will consider, A* search is guaranteed to be optimal if you use a heuristic that is admissible

Informed Search Methods

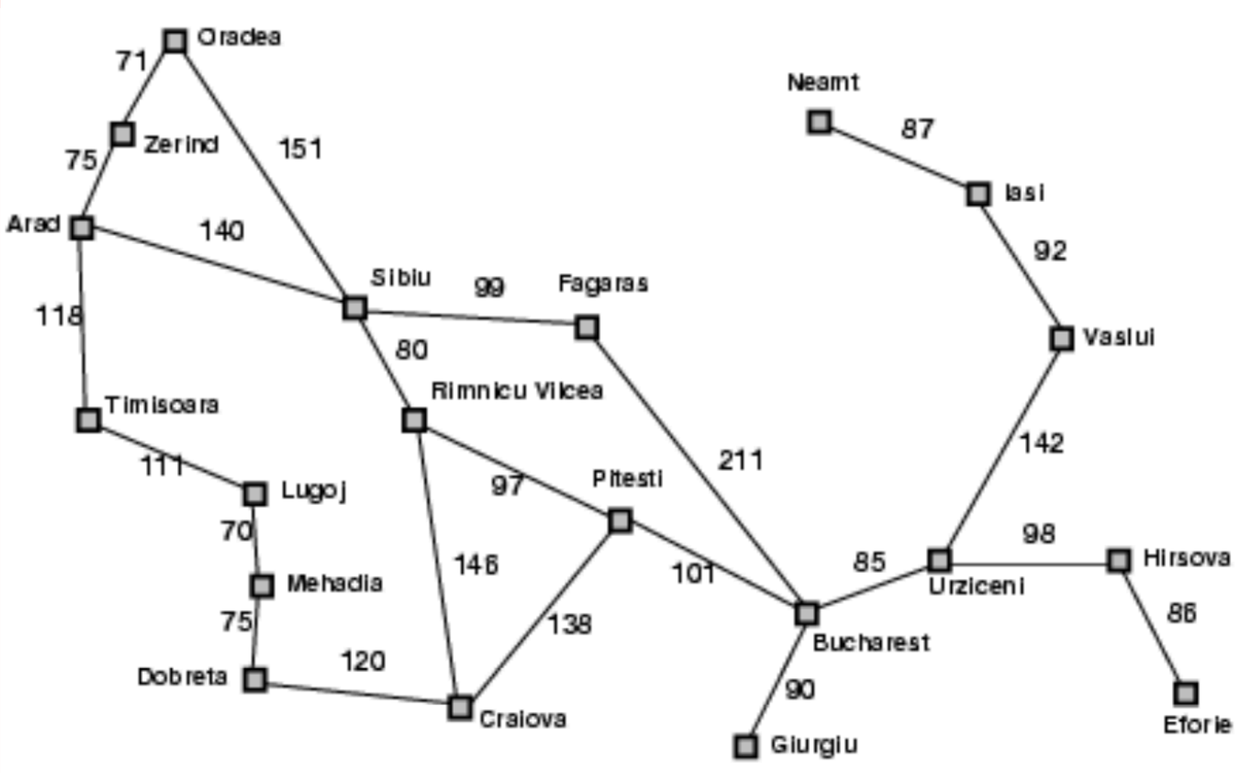
- Suppose that we have a heuristic function h that approximates distance from current state to goal state, and a function f that gives the **exact** cost from start state to current state

- Then $g(s) = f(s) + h(s)$ gives the approximate cost to goal state if we pass through state s

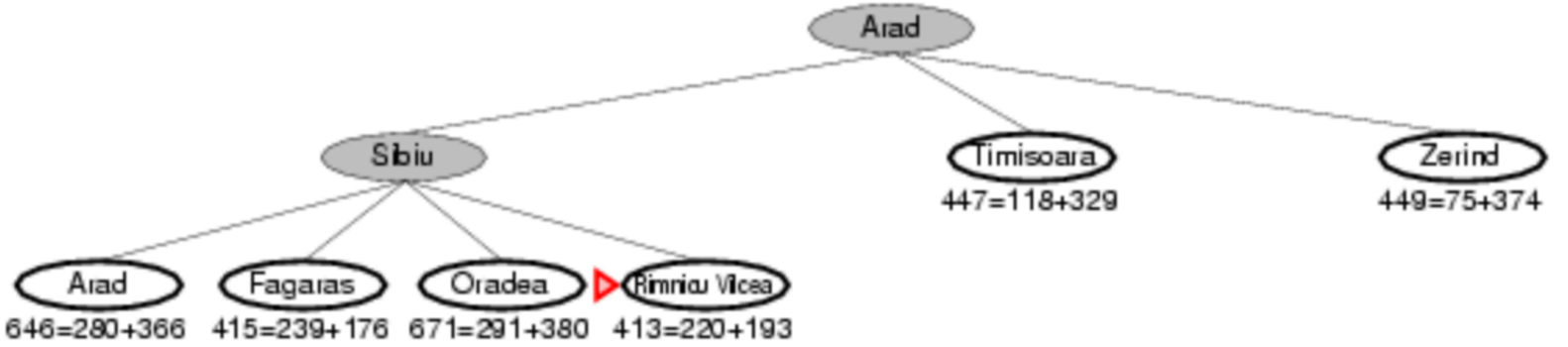
- This observation is the core idea of A* search

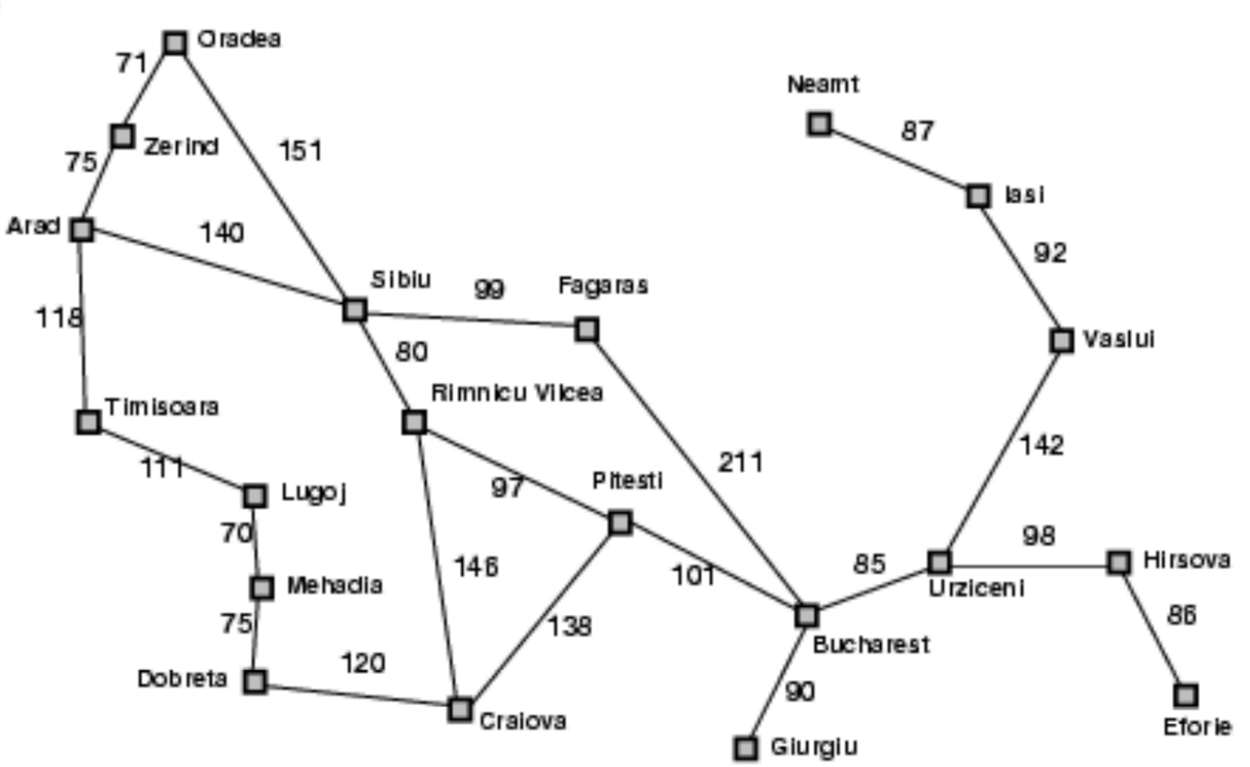
A* Search

- Core idea: use a priority queue (min heap) to manage current node
- Node priority based on g
- As we expand in new directions we might need to update priority value for nodes
  - Update priority if we have not visited not yet
  - Similar to Dijkstra's algorithm



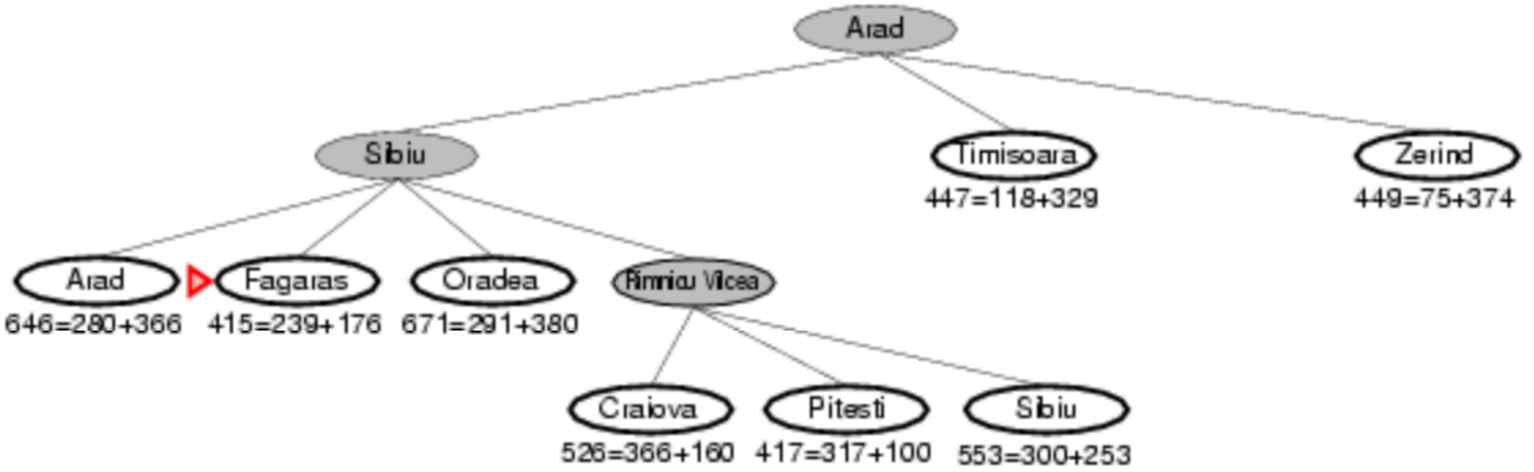
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

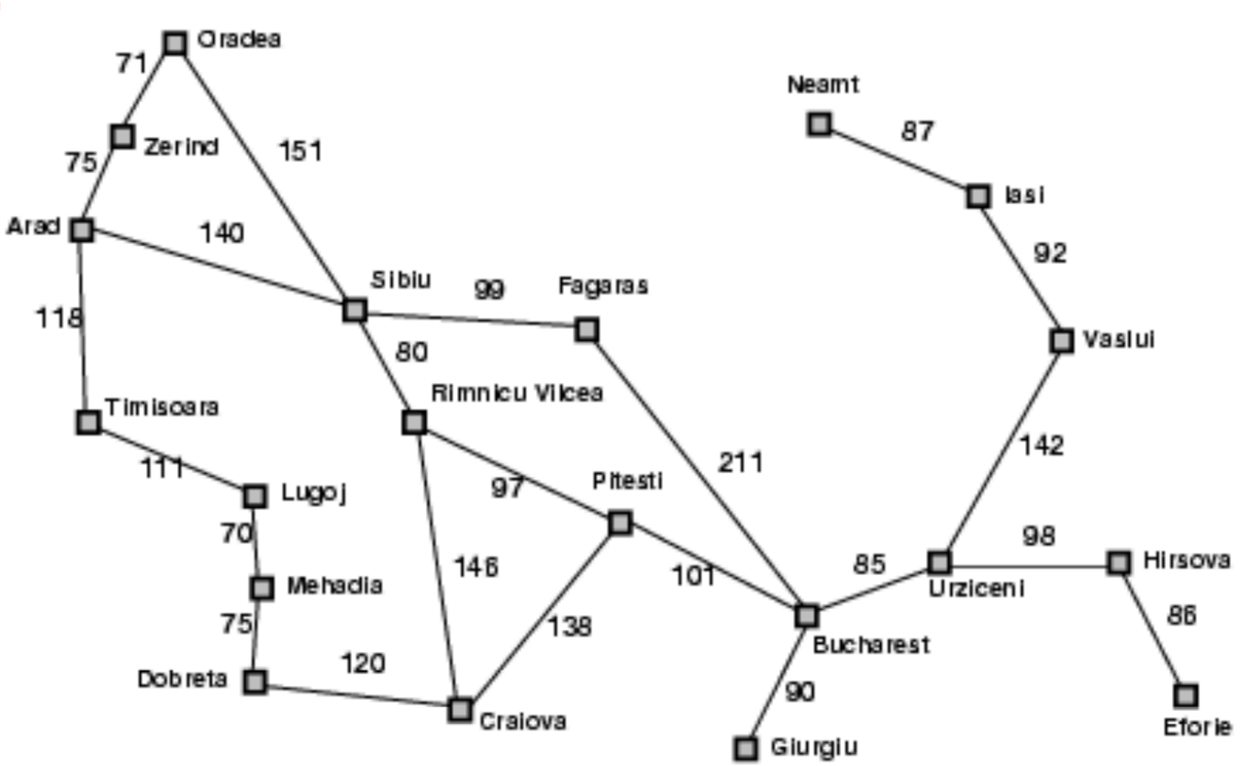




Straight–line distance to Bucharest

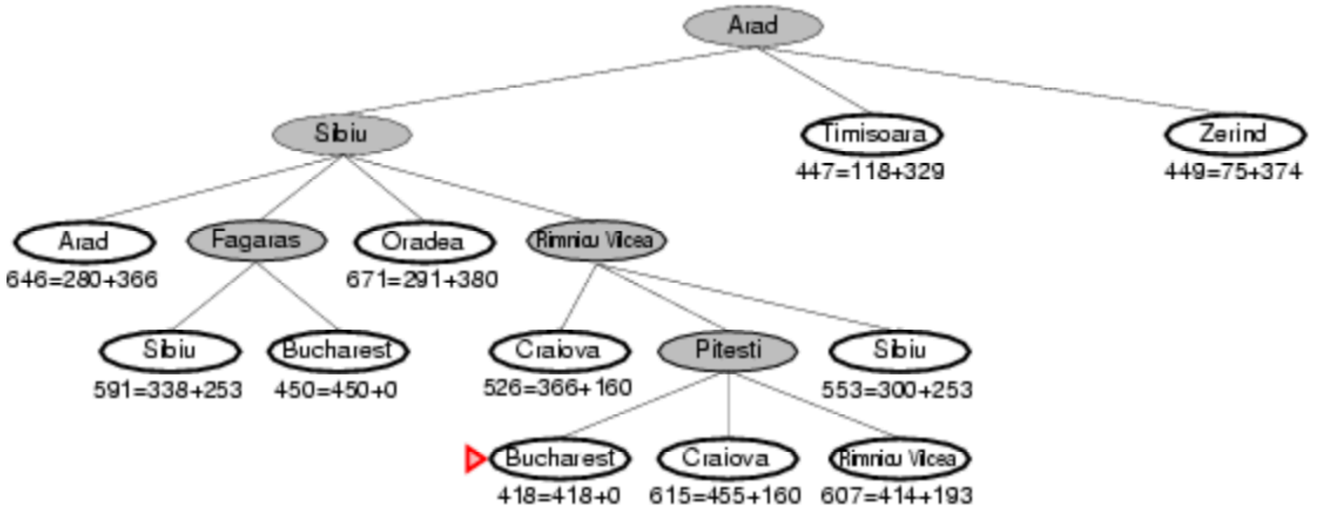
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

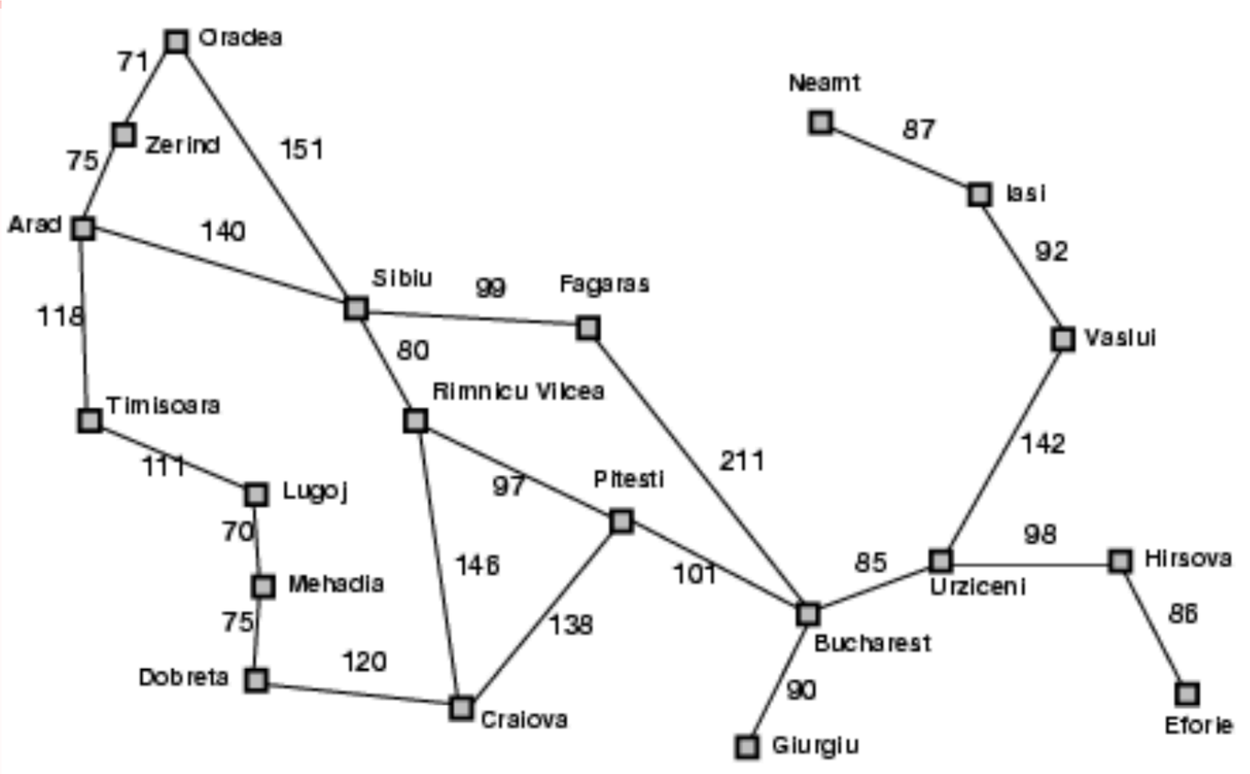




Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374





Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374