

Artificial Intelligence-1(CS249)

4th week notes by:

2201AI17

2201AI18

2201AI19

2201AI20

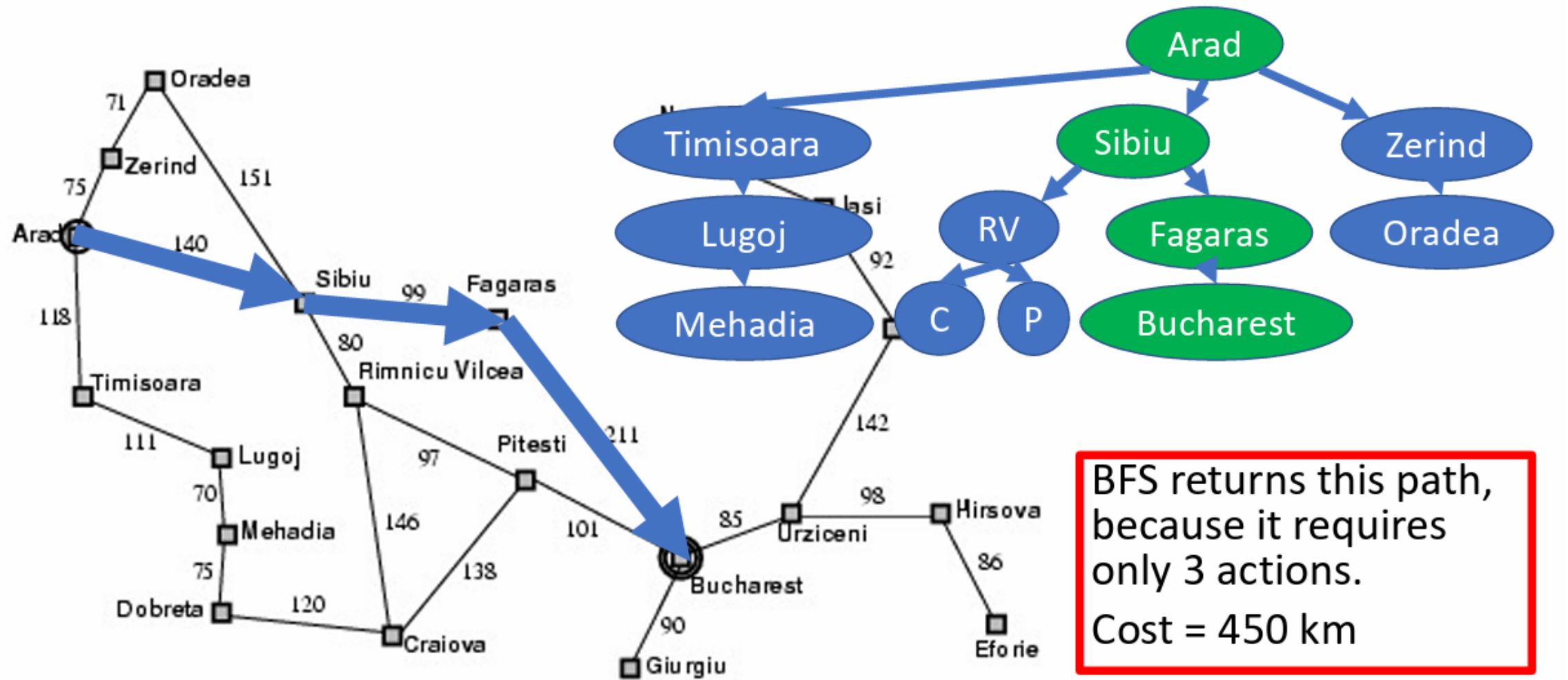
2201AI21

Uniform Cost Search :

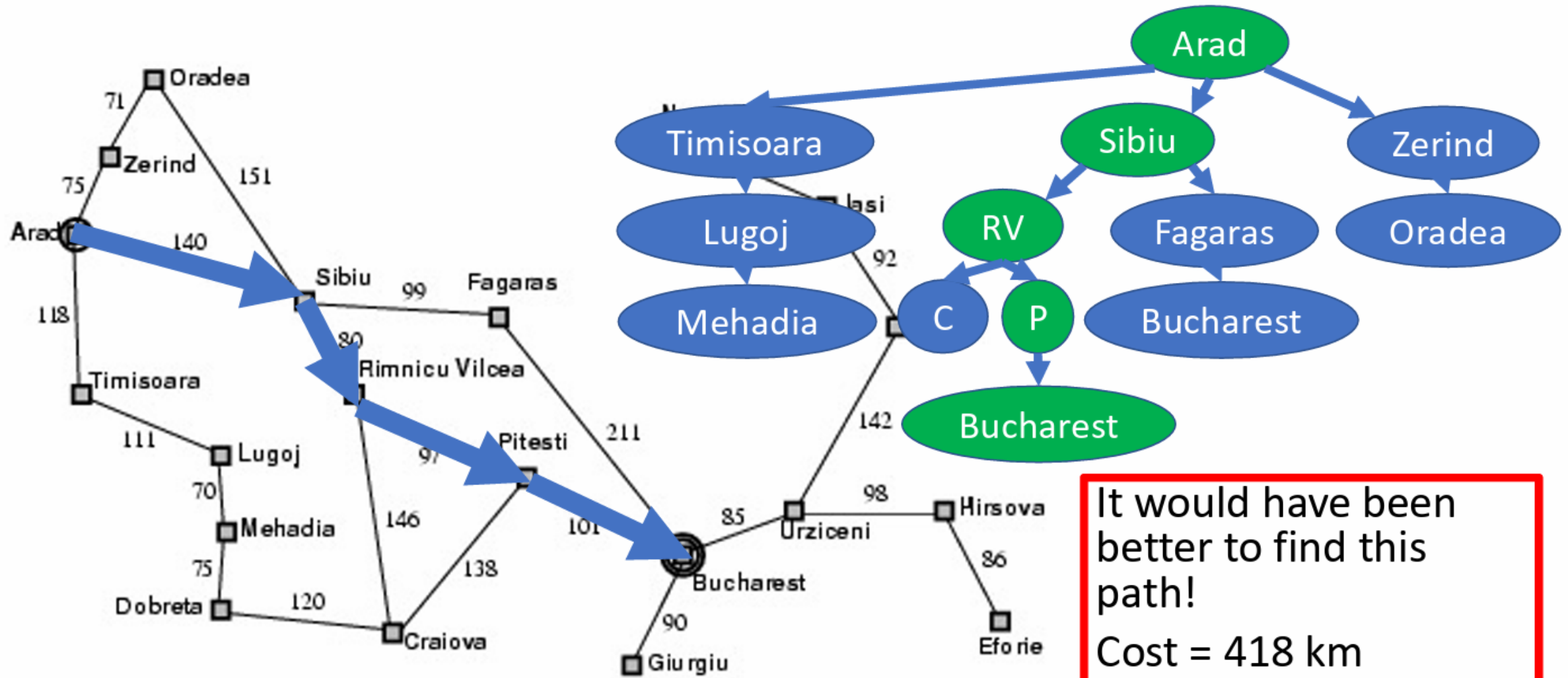
- Uniform cost search is a variant of Dijkstra's algorithm. Here, instead of inserting all vertices into a priority queue, we insert only the source, then one by one insert when needed. In every step, we check if the item is already in the priority queue (using the visited array). If yes, we perform the decrease key, else we insert it.
- This variant of Dijkstra is useful for infinite graphs and that graph which are too large to represent in memory. Uniform Cost Search is mainly used in Artificial Intelligence.

- **UCS assumes all operators have a cost.**
- Initialise : Set $OPEN = \{s\}$ $CLOSED = \{\}$, set $C(s) = 0$
- FAIL : If $OPEN = \{\}$, Terminate & FAIL.
- Select : Select the minimum cost state 'n', from OPEN and save 'n' is closed.
- Terminate : If $n \in G$ terminate with success. **[This Termination condition is used when all operator costs are positive].**
- Expand : Generate the successors of n using O for each successor, 'm'.
 - If $m \notin [OPEN \cup CLOSED]$
 - set $C(m) = \min\{C(m), C(n) + C(n, m)\}$
 - If $m \in [OPEN \cup CLOSED]$
 - set $C(m) = \min\{C(m), C(n) + C(n, m)\}$
 - If $C(m)$ has decreased and $m \in CLOSED$, move it to OPEN.
- Loop

An example for which BFS is not optimal: Romania

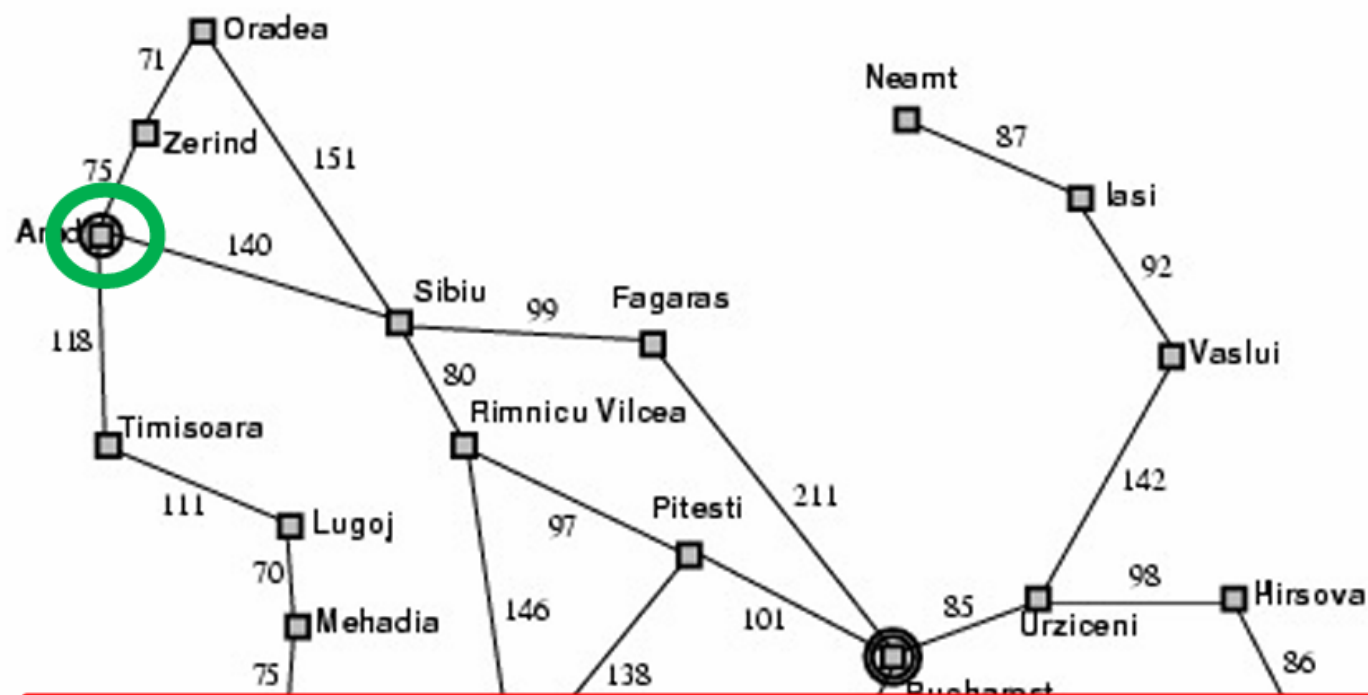


An example for which BFS is not optimal: Romania



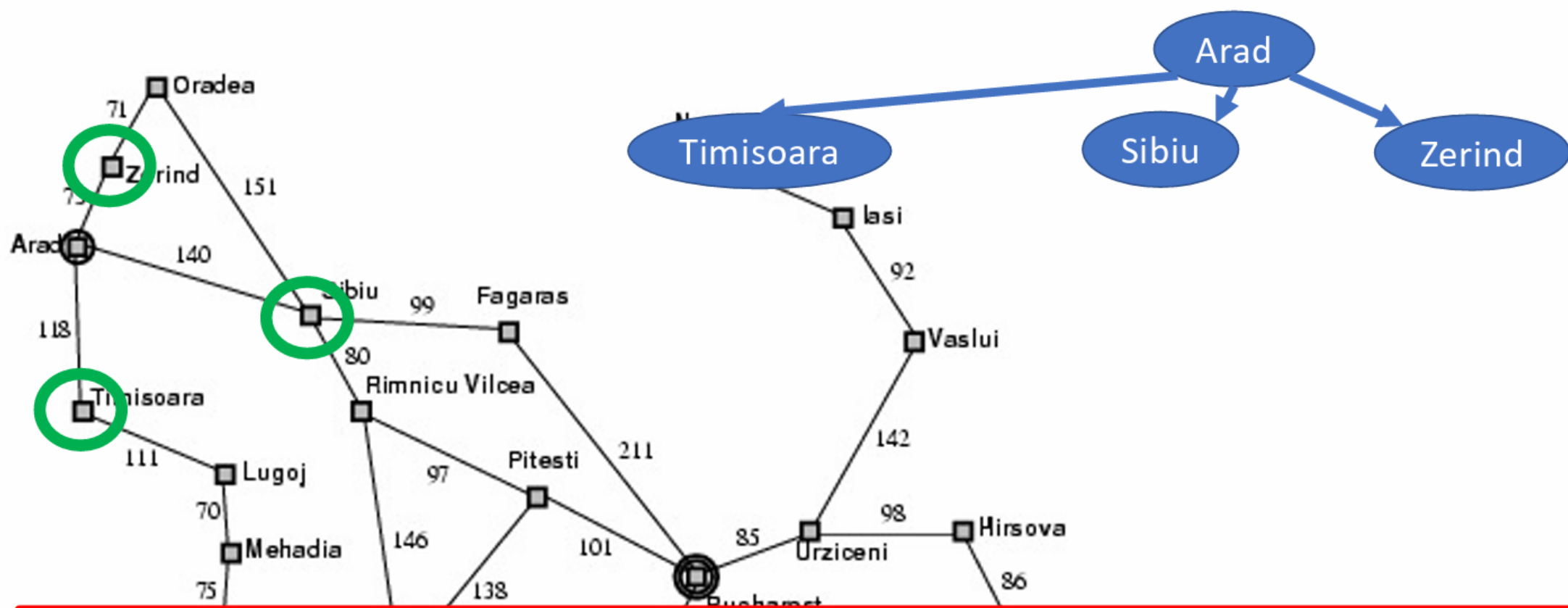
Example of UCS: Romania

Arad



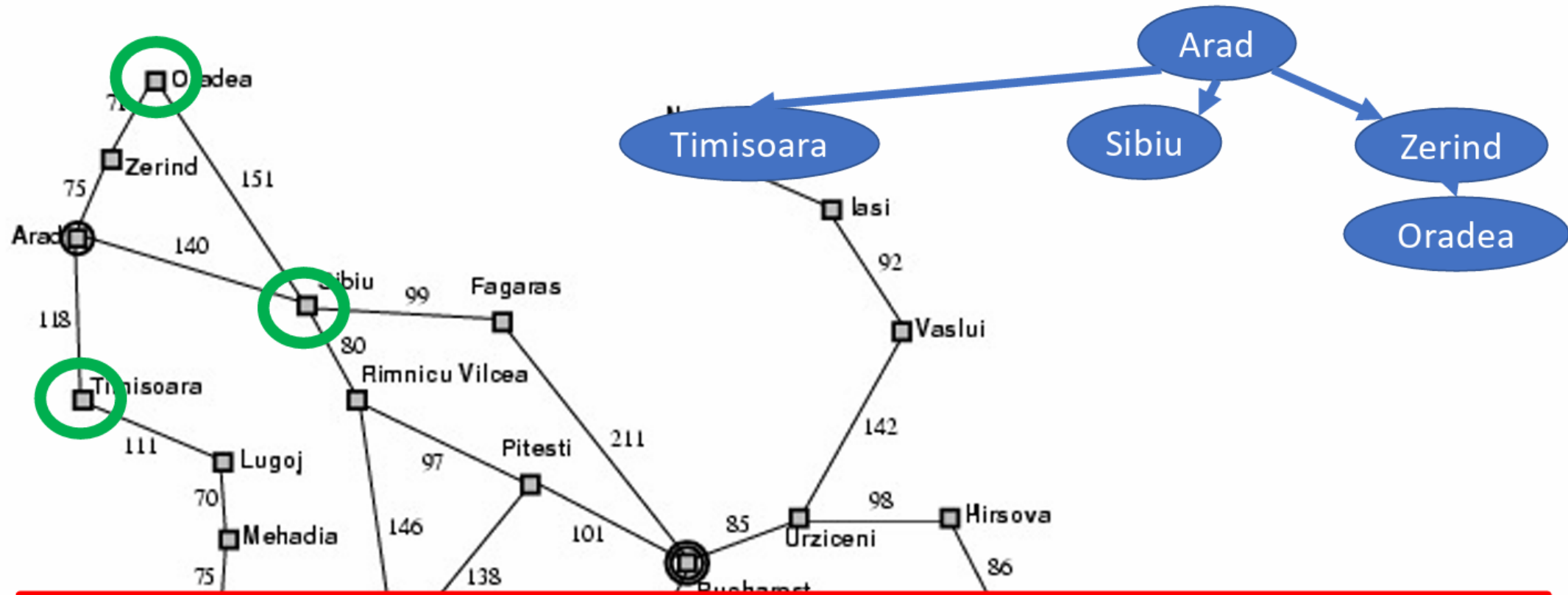
Arad:0

Example of UCS: Romania



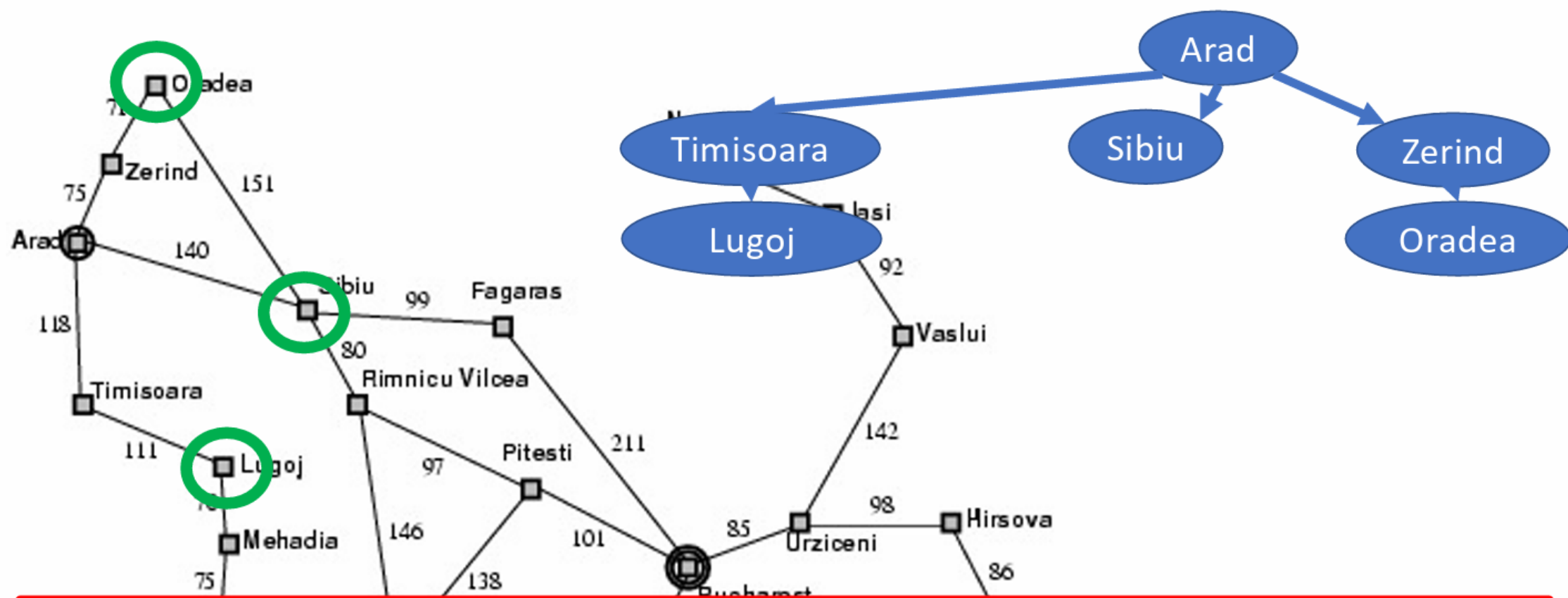
Zerind:75, Timisoara:118, Sibiu:140

Example of UCS: Romania



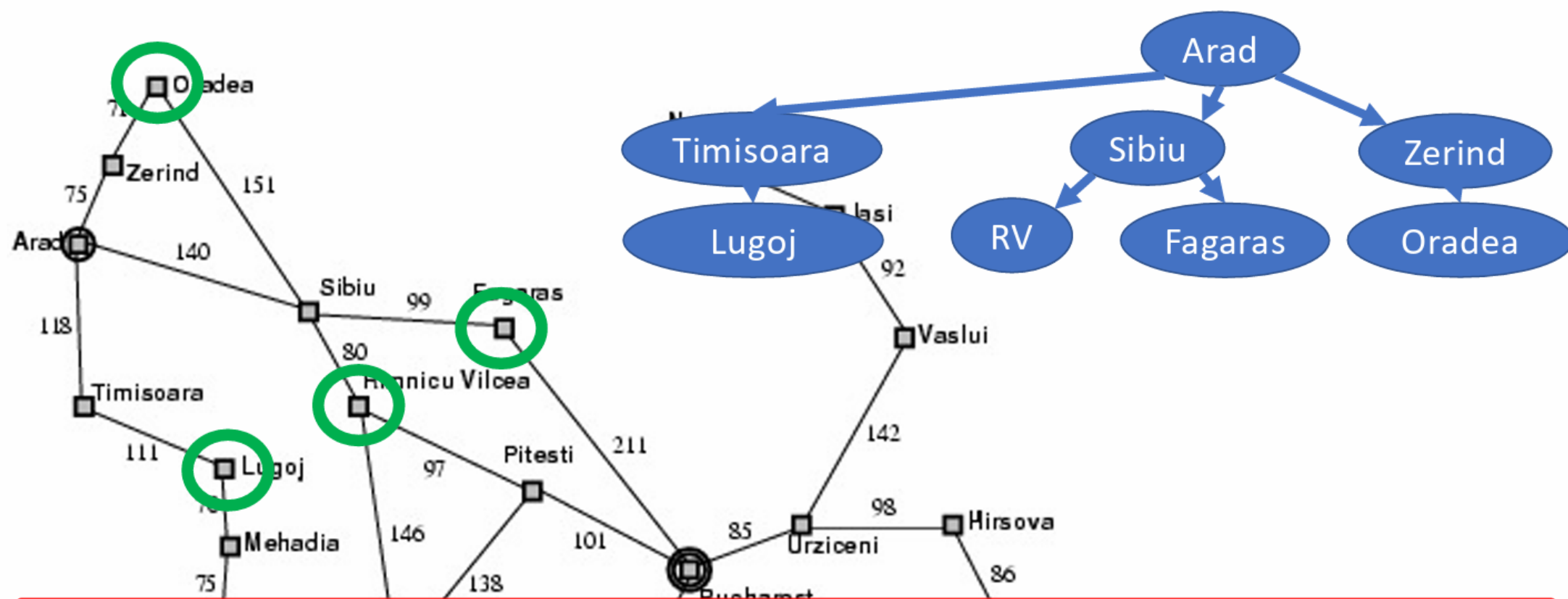
Timisoara:118, Sibiu:140, Oradea:146

Example of UCS: Romania



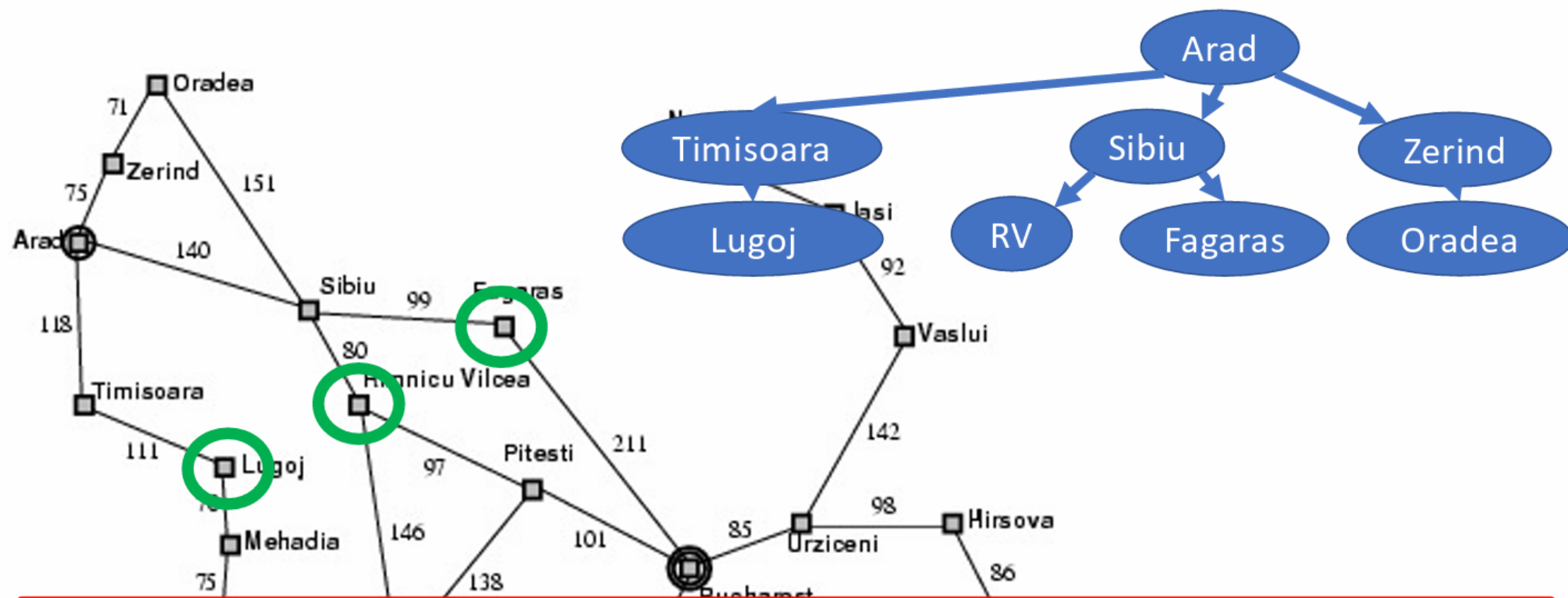
Sibiu:140, Oradea:146, **Lugoj:239**

Example of UCS: Romania



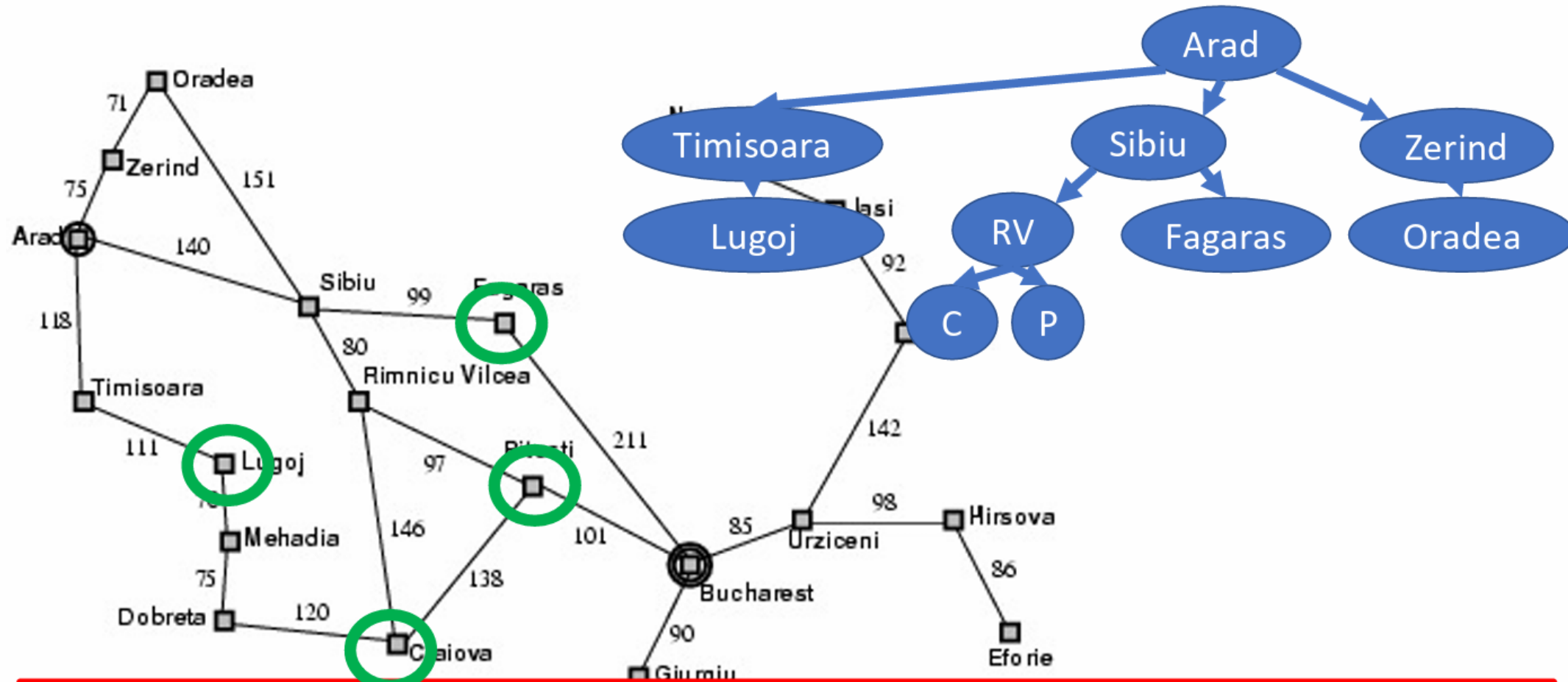
Oradea:146, Ramnicu Valcea:220, Lugoj:239, Fagaras:239

Example of UCS: Romania



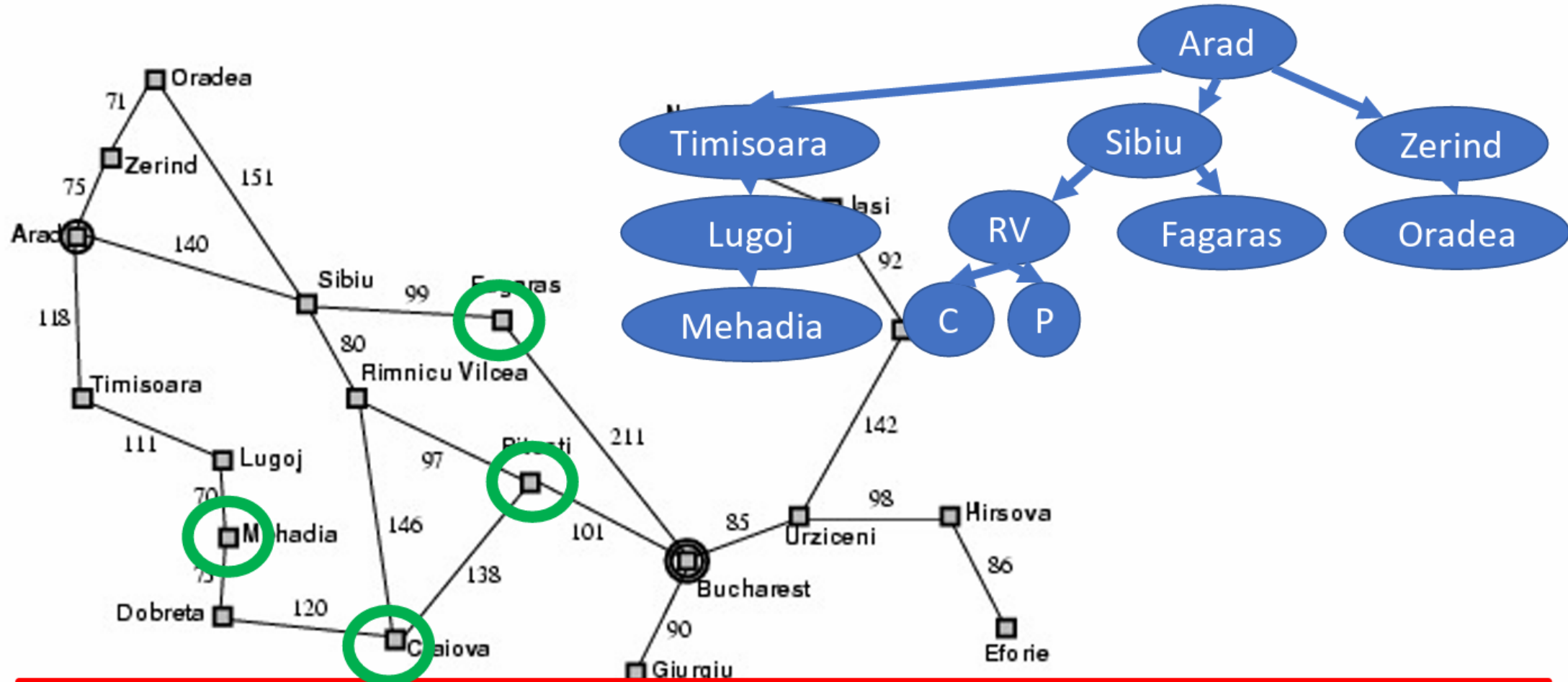
Ramnucu Valcea:220, Lugoj:239, Fagaras:239

Example of UCS: Romania

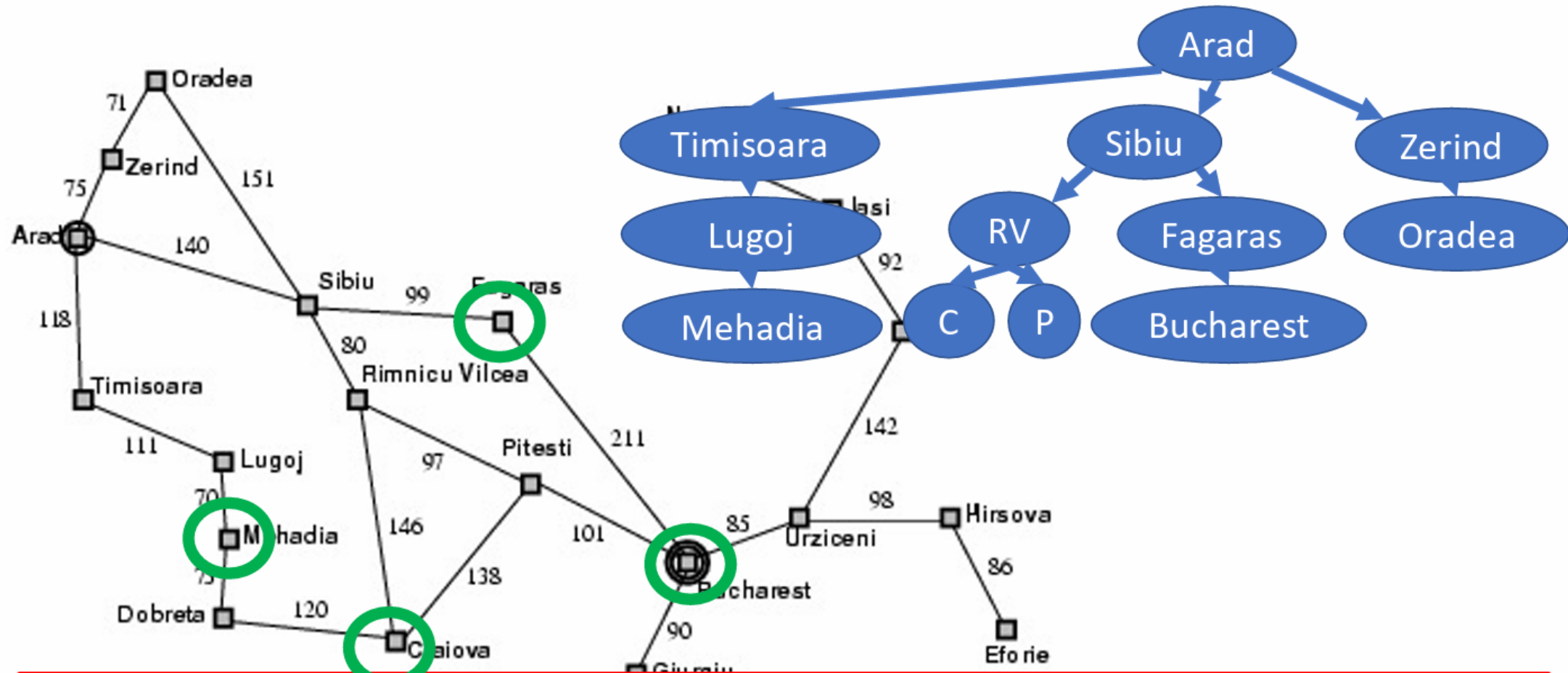


Lugoj:239, Fagaras:239, Pitesti:317, Craiova:366

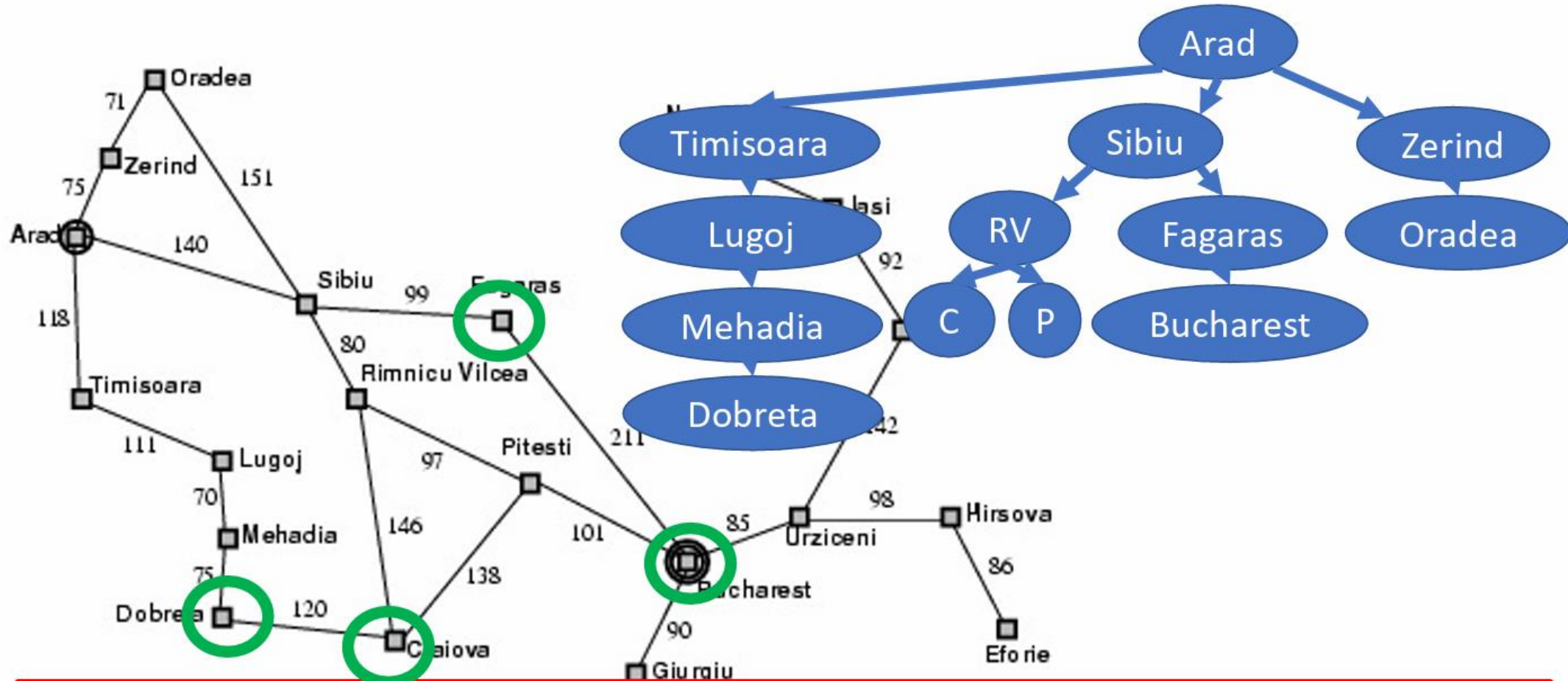
Example of UCS: Romania



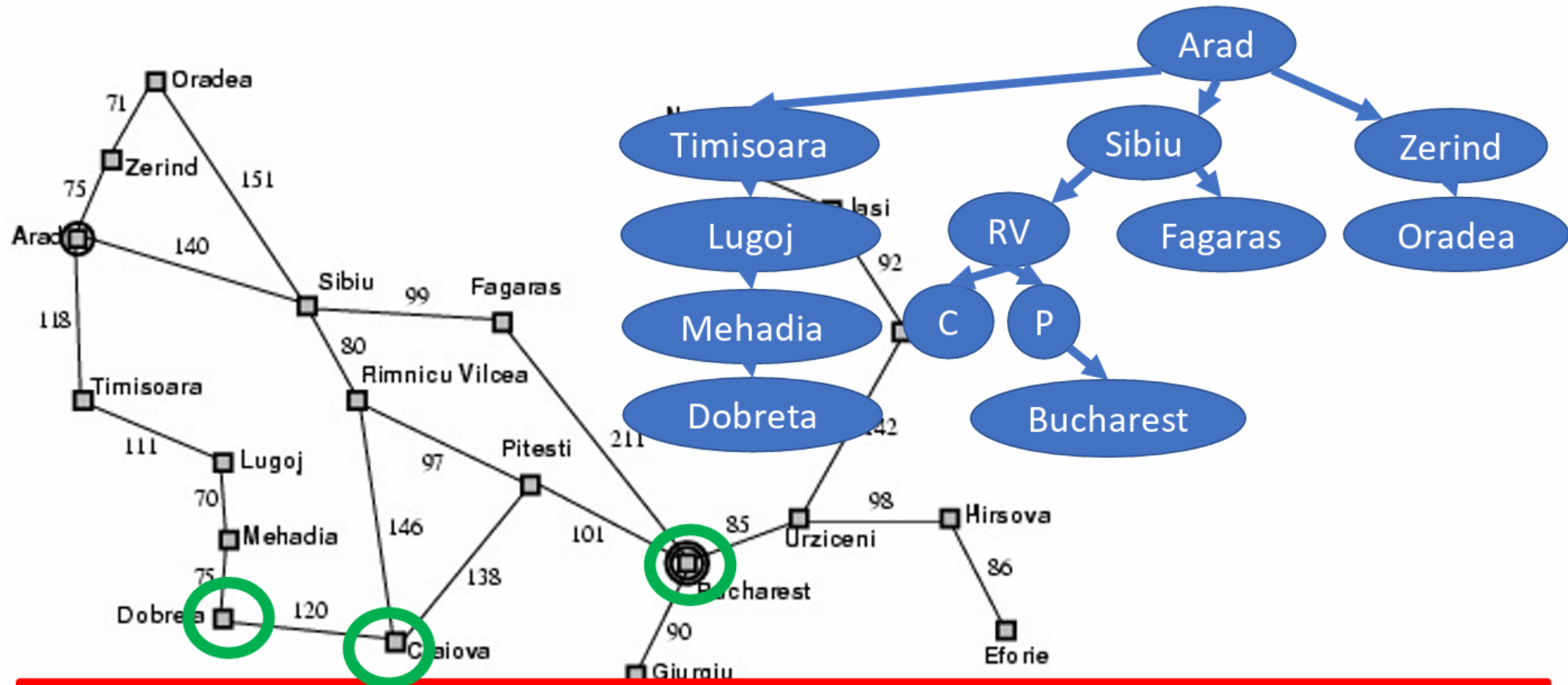
Example of UCS: Romania



Example of UCS: Romania

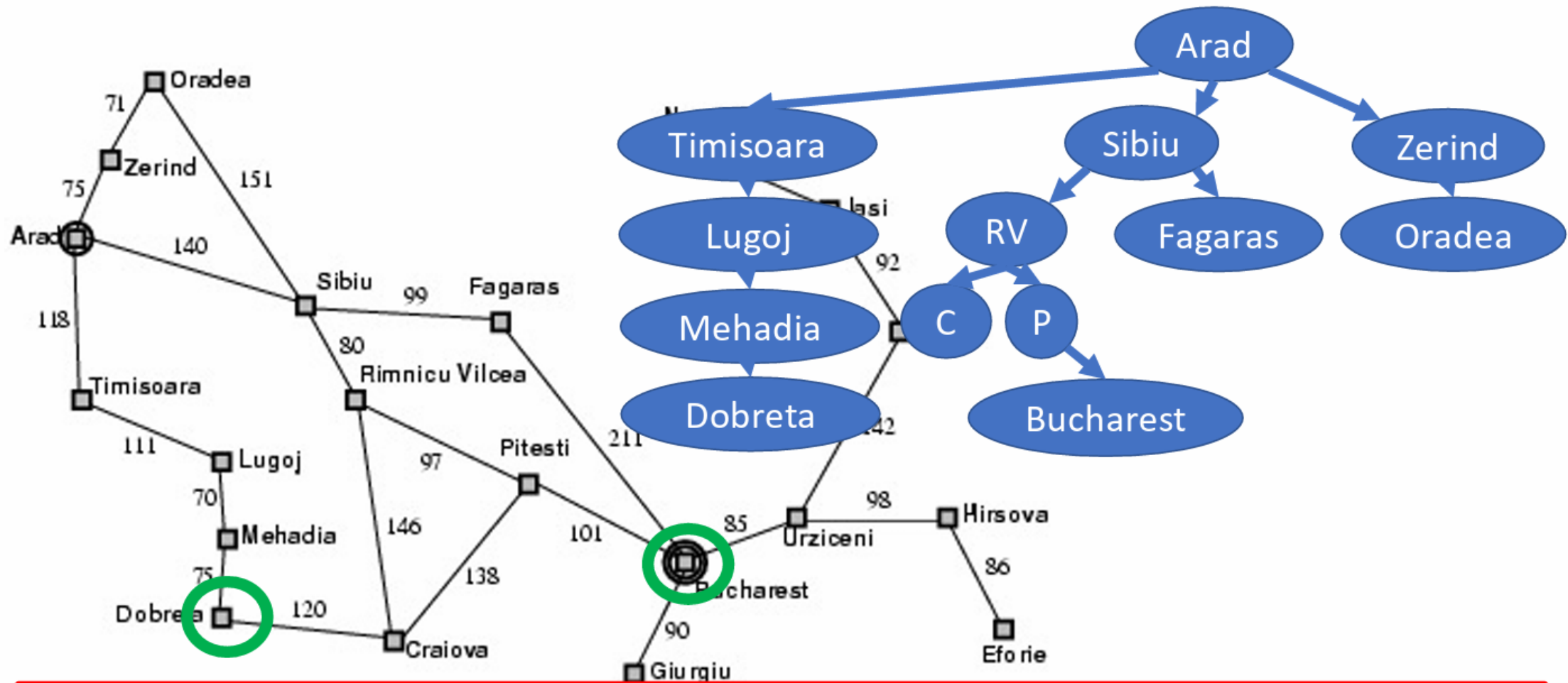


Example of UCS: Romania



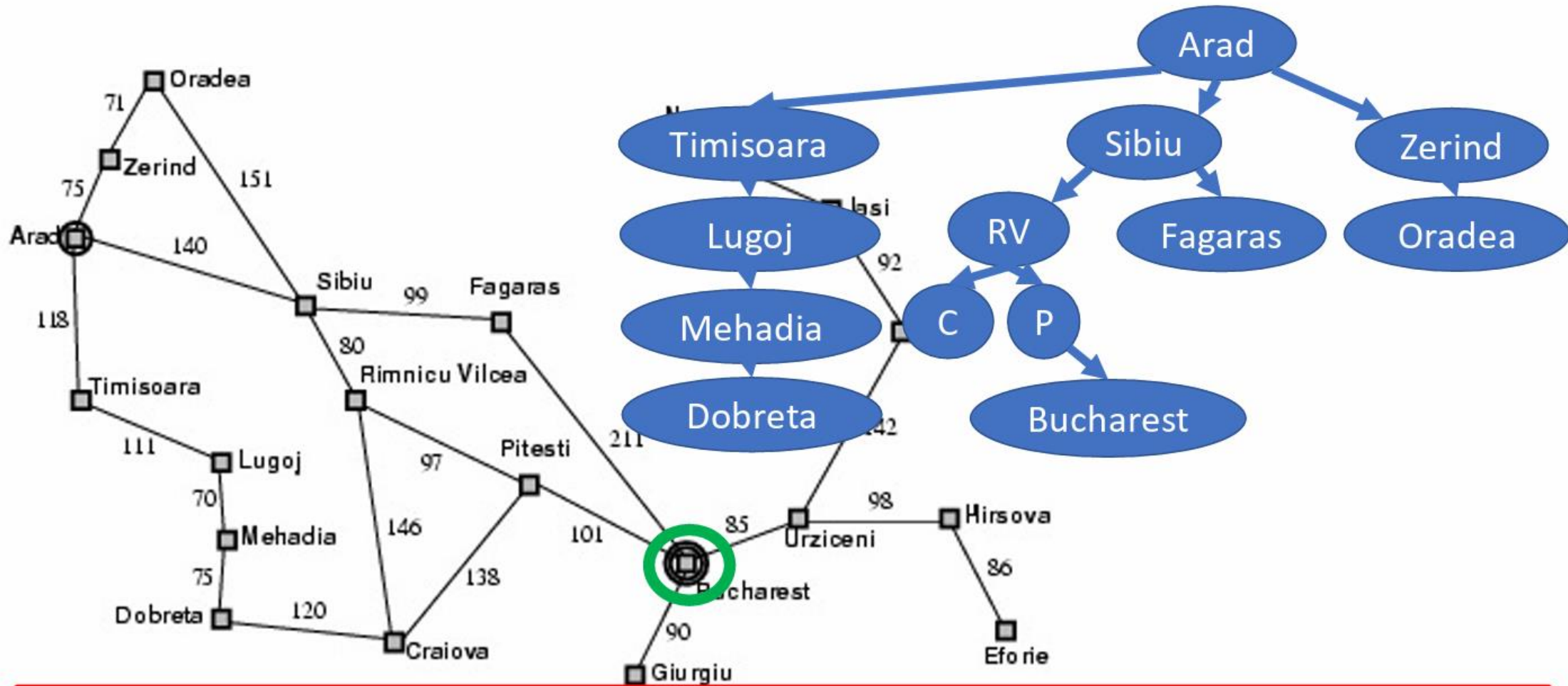
Craiova:366, Dobreta:384, **Bucharest:418**

Example of UCS: Romania



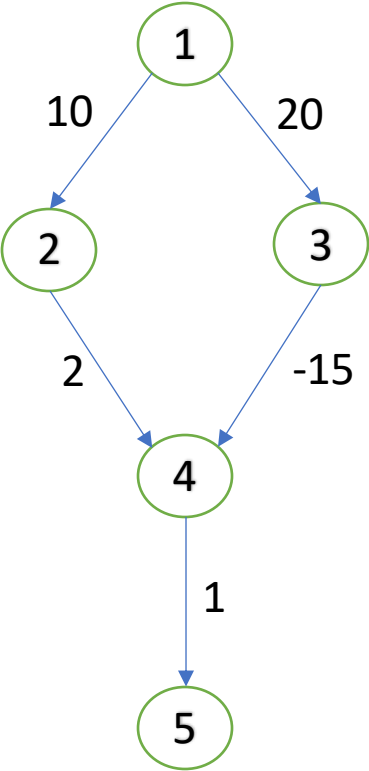
Dobreta:384, Bucharest:418

Example of UCS: Romania



Bucharest:418

Uniform Cost Search for graph with negative edges :



OPEN

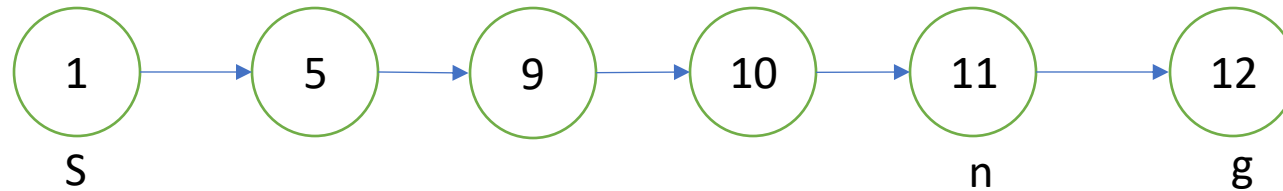
1(0),2(10),3(20)
3(20),4(12)
3(20),5(13)
3(20)
4(5)
4(5)
5(6)

CLOSED

1(0)
1(0),2(10)
1(0),2(10),4(12)
1(0),2(10),4(12),5(13)
1(0),2(10),4(12),5(13),3(20)
1(0),2(10),5(13),3(20)
1(0),2(10),3(20),4(5)

A. Theorem : The UCS algorithm applied on a problem having positive operator cost will return optimal solution

B. Proposition : Atleast one node in the path from s to the $n \in S$ (having optimal cost) should be OPEN before n is being removed from OPEN.



C. There doesn't exist any other path from s to n having lesser cost than $n.cost$.

1. Atleast one node belonging to the path having 'Optimal Cost' from s to n should be in OPEN, always before retrieving n .

Optimal cost from s to $n = C^*(n) = A + B$

In path – 1,

$$A + B = C^*(n)$$

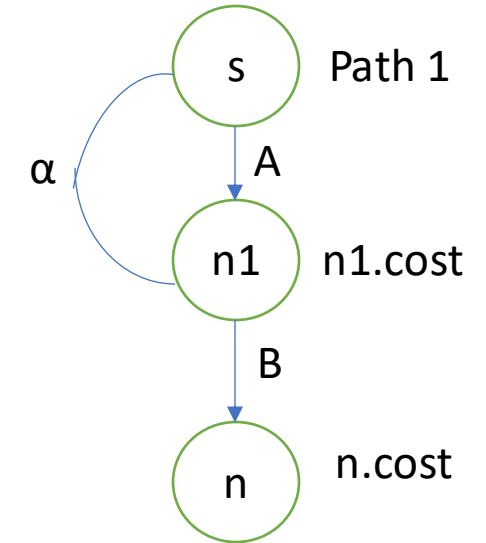
$g \in G$: Closest Optimal goal from s

$C^*(n) \leq C^*(g) \quad \forall n \in s$ will be expanded before g .

Assume : $\alpha < A$

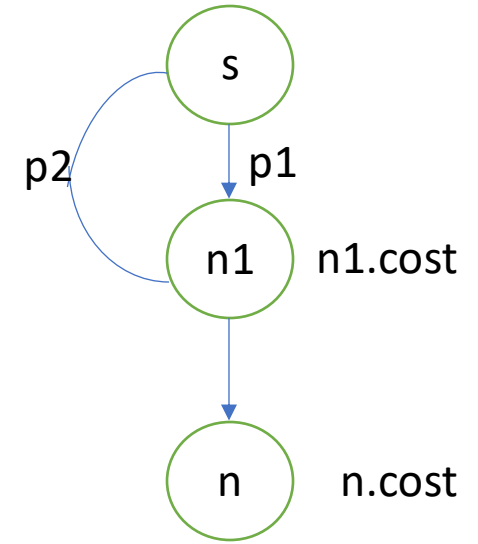
$$\alpha + \beta < C^*(n)$$

This is a contradiction as we assumed $C^*(n)$ to be Optimal.



2. If s to n : Optimal,
then s to $n1$: Optimal

p is optimal path from s to n through $n1$ with cost $C^*(n)$.
 $p1$ is one path of p from s to $n1$.



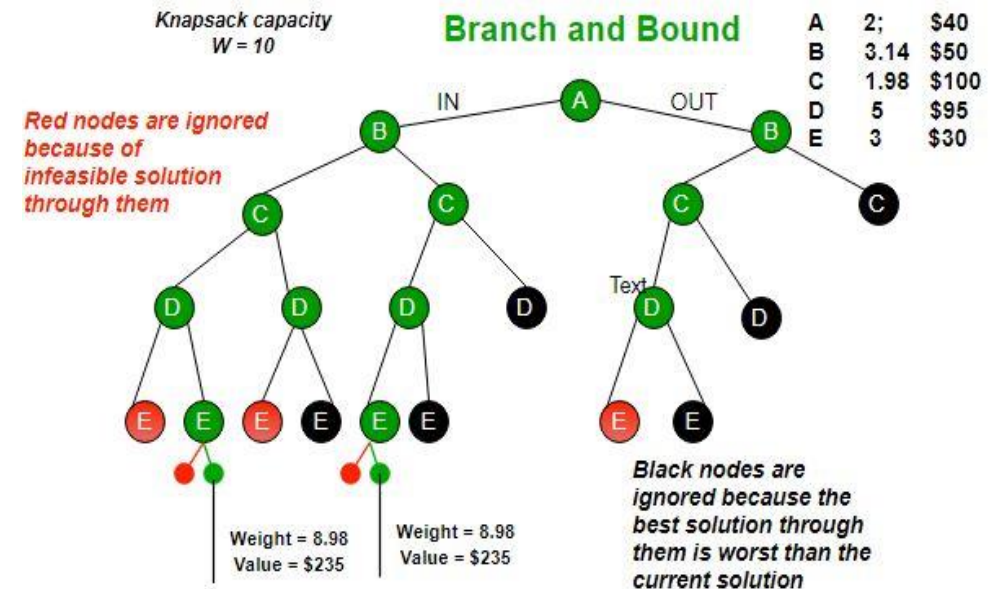
We need to proof $p1$ is optimal subpath from s to $n1$. Assume $p1$ is not optimal path from s to n with cost $C(n1)$
Let $\exists p2$ which is optimal path from s to $n1$.

$\Rightarrow C^*(n1) < C(n1)$.

- **Base Case :** N_p all the nodes that are directly reachable from S are in OPEN in 1st step of Algorithm.
- Let P be the optimal path from s to a node $n \in S$. P consists of atleast one node in N_s .
- Let us assume s to n consists of n_1 and n_2 . n_1 is directly reachable from s , is in OPEN.
- Until n_1 is not being removed, B (statement in slide no 20) is True.
- Where n_1 is expanded, n_2 will be visited,
 1. n_2 is not in (OPEN \cup CLOSED), so n_2 is brought to OPEN.
 2. $n_2 \in$ (OPEN \cup CLOSED), so cost of n_2 will be revisited through n_1 .
- Using induction we can prove B (statement in slide no 20) .

BRANCH AND BOUND ALGORITHM

- A branch-and-bound algorithm consists of a systematic enumeration of candidate solutions by means of state space search: the set of candidate solutions is thought of as forming a rooted tree with the full set at the root. The algorithm explores branches of this tree, which represent subsets of the solution set.



Branch and Bound

- 1. Initialise** : Set $OPEN = [s]$, $CLOSED = []$, set $C(s) = 0$, $C^* = \infty$.
- 2. Terminate** : If $OPEN = []$, then return C^* .
- 3. SELECT** : Select a state n , from $OPEN$ and save in $CLOSED$.
- 4. New Bound Assignment**: If $n \in G$ and $C(n) < C^*$ then set $C^* = C(n)$ and go to step 2.

5. Expand : If $C(n) < C^*$ generate the successor of $n \rightarrow (m)$,

For each successor m ,

- If $m \notin [\text{OPEN} \cup \text{CLOSED}]$
Set $C(m) = C(n) + C(n, m)$ and insert m in OPEN.
- If $m \in (\text{OPEN} \cup \text{CLOSED})$
Set $C(m) = \min(C(m), B)$.

If $C(m)$ has decreased and $m \in \text{CLOSED}$ move to OPEN.

HEURISTIC

- A heuristic is a function that determines how near a state is to the desired state. Heuristics functions vary depending on the problem and must be tailored to match that particular challenge. The majority of AI problems revolve around a large amount of information, data, and constraints, and the task is to find a way to reach the goal state. The heuristics function in this situation informs us of the proximity to the desired condition. The distance formula is an excellent option if one needed a heuristic function to assess how close a location in a two-dimensional space was to the objective point.

$$f(n) = g(n) + h(n)$$

$g(n)$ = Actual Cost of n from s .

$h(n)$ = Estimated cost of n to reach G from n

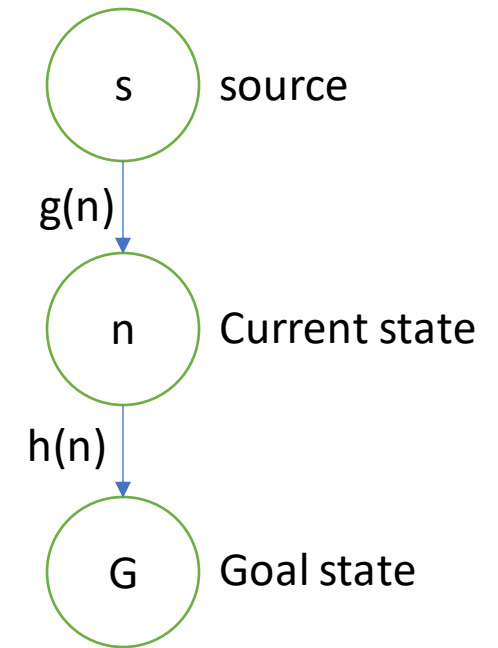
$h^*(n)$ = Optimal cost to reach G from n .

$h(n) \leq h^*(n)$ \rightarrow Underestimated

$h(n) > h^*(n)$ \rightarrow Overestimated

$g(s) = 0$, where s is the start.

$h(G) = 0$, where G is the Goal State.



EXAMPLES OF HEURISTIC FUNCTIONS IN AI

- A variety of issues can be solved using a heuristic function in AI.
- Let's talk about some of the more popular ones.

Traveling Salesman Problem :

- What is the quickest path between each city and its starting point, given a list of cities and the distances between each pair of them?

- This problem could be brute-forced for a small number of cities. But as the number of cities grows, finding a solution becomes more challenging.
- This issue is well-solved by the nearest-neighbor heuristic, which directs the computer to always choose the closest unexplored city as the next stop on the path. While NN only sometimes offers the optimum solution, it is frequently near enough that the variation is insignificant to respond to the salesman's problem. This approach decreases TSP's complexity from $O(n!)$ to $O(n^2)$.

Contributions

2201AI17 - 20%

2201AI18 - 20%

2201AI19 - 20%

2201AI20 - 20%

2201AI21 - 20%