Lecture 16: Uniform Cost Search and A*

Mark Hasegawa-Johnson, February 2022

With some slides by Svetlana Lazebnik, 9/2016

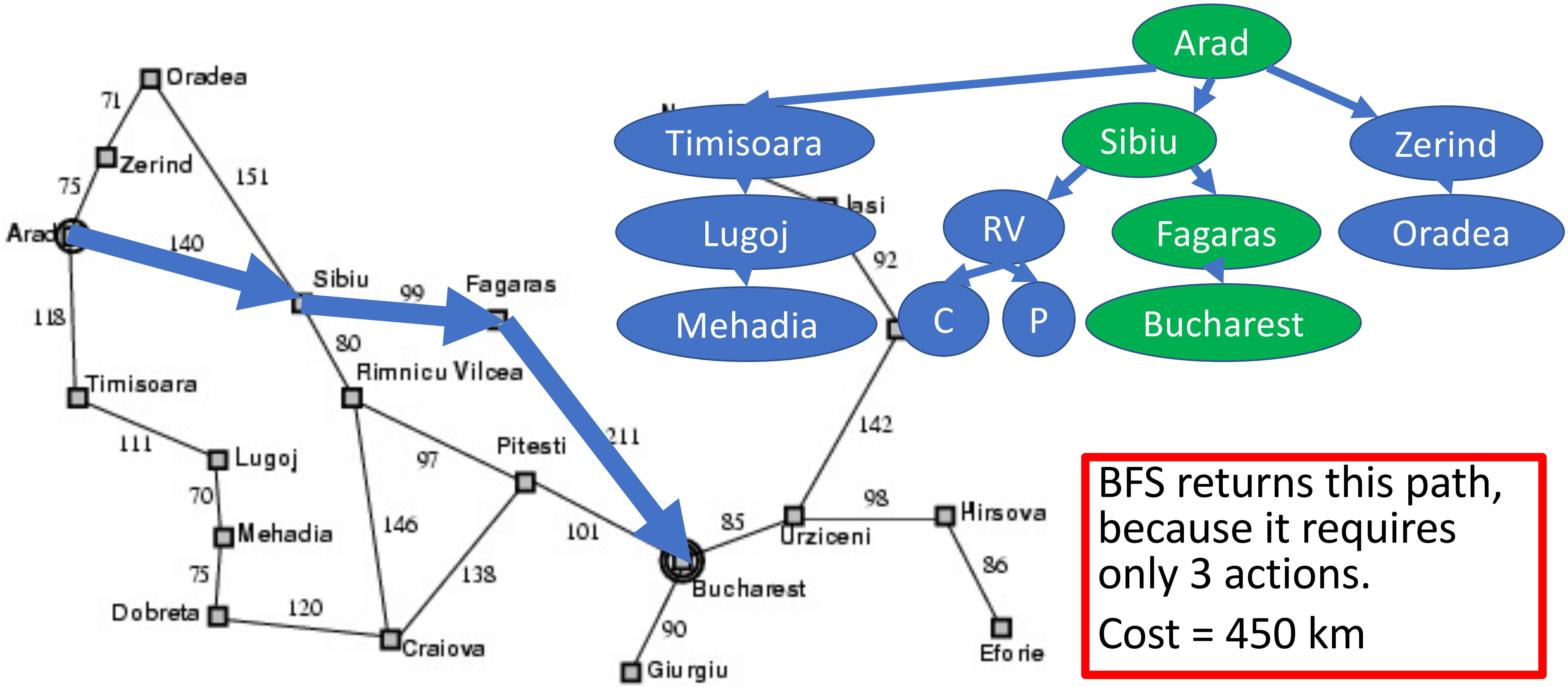
Review: DFS and BFS

- Breadth-first search
  - Frontier is a queue: expand the shallowest node
  - **Complete**: always finds a solution, if one exists
  - **Optimal** (finds the best solution) **if all actions have the same cost**.
  - **Time complexity:** $O\{b^d\}$
  - **Space complexity:** $O\{b^d\}$.
- Depth-first search – utility depends on relationship between *m* and *d*
  - Frontier is a stack: expand the deepest node
  - **Not complete** (might never find a solution, if *m* is infinite)
  - **Not optimal** (returned solution is rarely the best one)
  - **Time complexity:** $O\{b^m\}$
  - **Space complexity:** $O\{bm\}$.

Outline of today's lecture

1. Uniform Cost Search (UCS): like BFS, but for actions that have different costs
   - **Complete**: always finds a solution, if one exists
   - **Optimal:** finds the best solution
   - **Time complexity** = # nodes that have cost < goal
   - **Space complexity** = # nodes that have cost < goal
2. Heuristics, e.g., Manhattan distance
3. Greedy Best-first search
4. A*: Like UCS but adds an estimate of the remaining path length
   - **Complete**: always finds a solution, if one exists
   - **Optimal:** finds the best solution
   - **Time complexity** = # nodes that have cost+heuristic < goal
   - **Space complexity** = # nodes that have cost+heuristic < goal

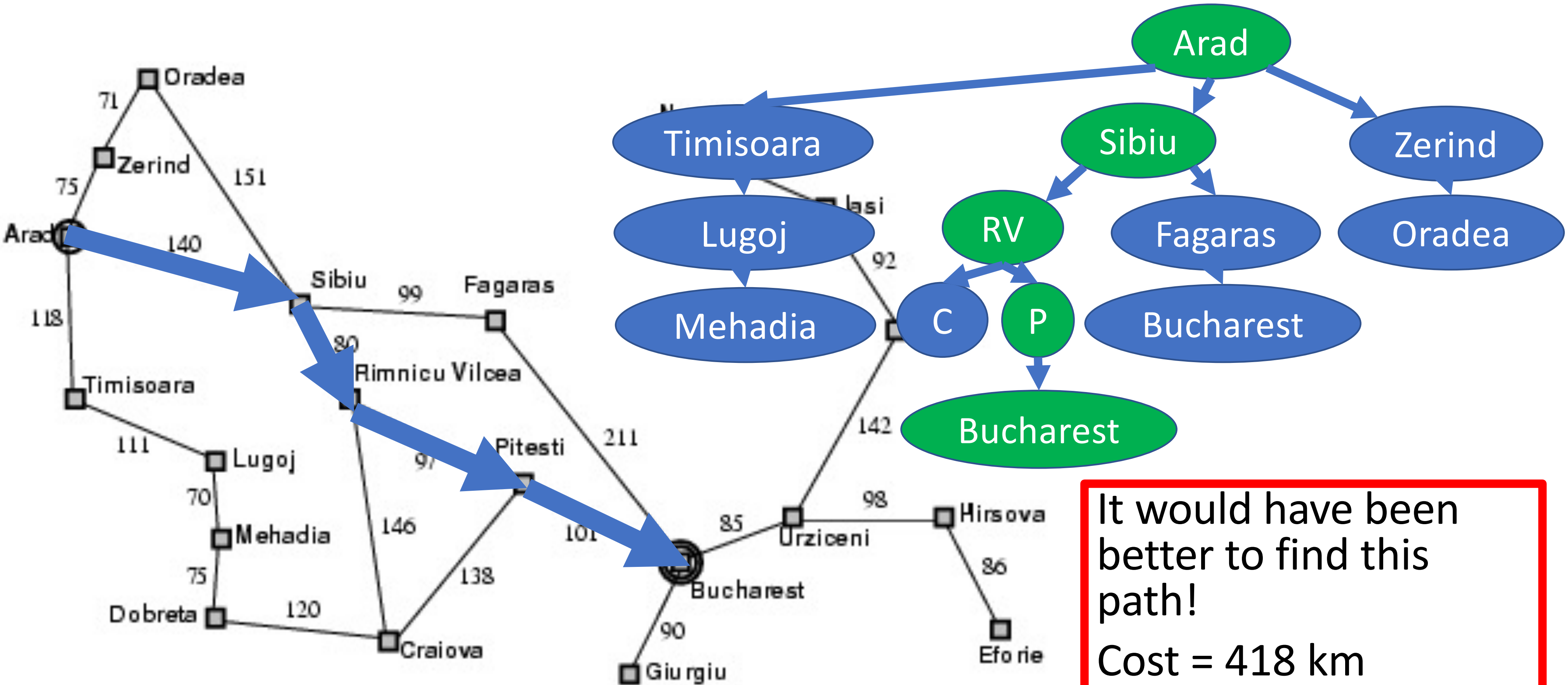
An example for which BFS is not optimal: Romania



BFS returns this path, because it requires only 3 actions.

Cost = 450 km

An example for which BFS is not optimal: Romania



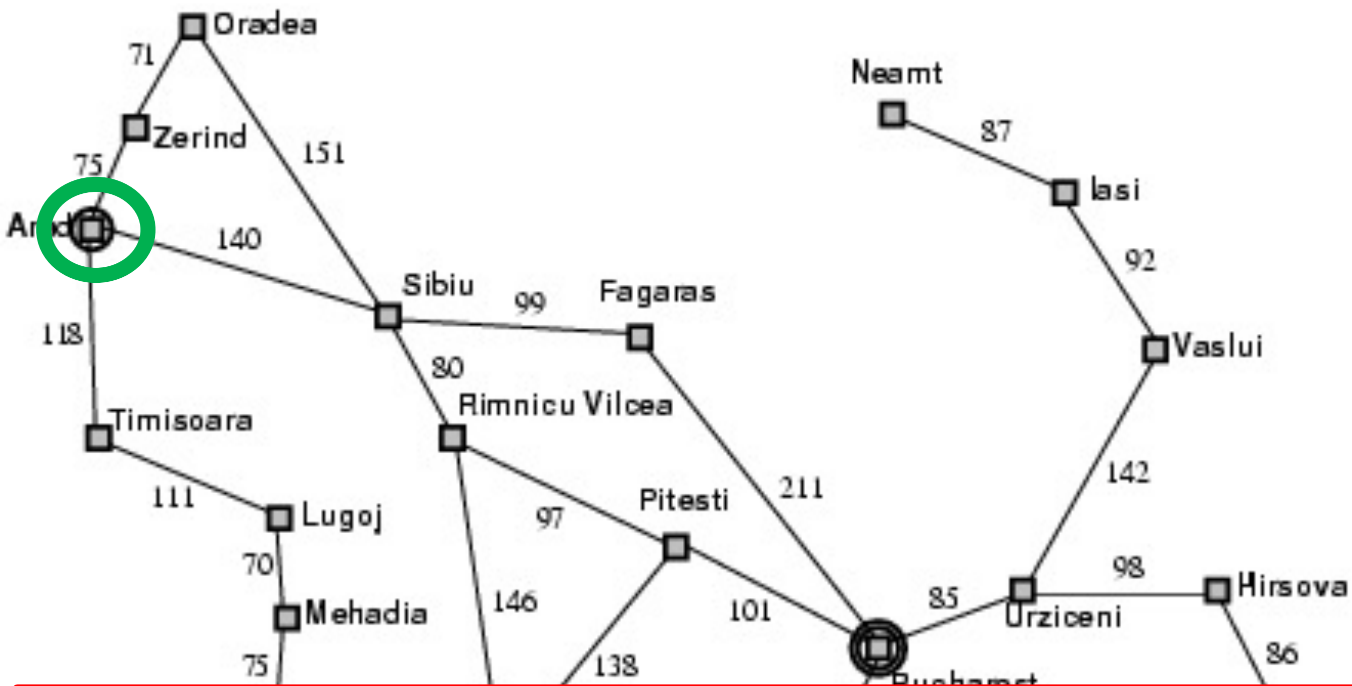
It would have been better to find this path!

Cost = 418 km

The solution: Uniform Cost Search

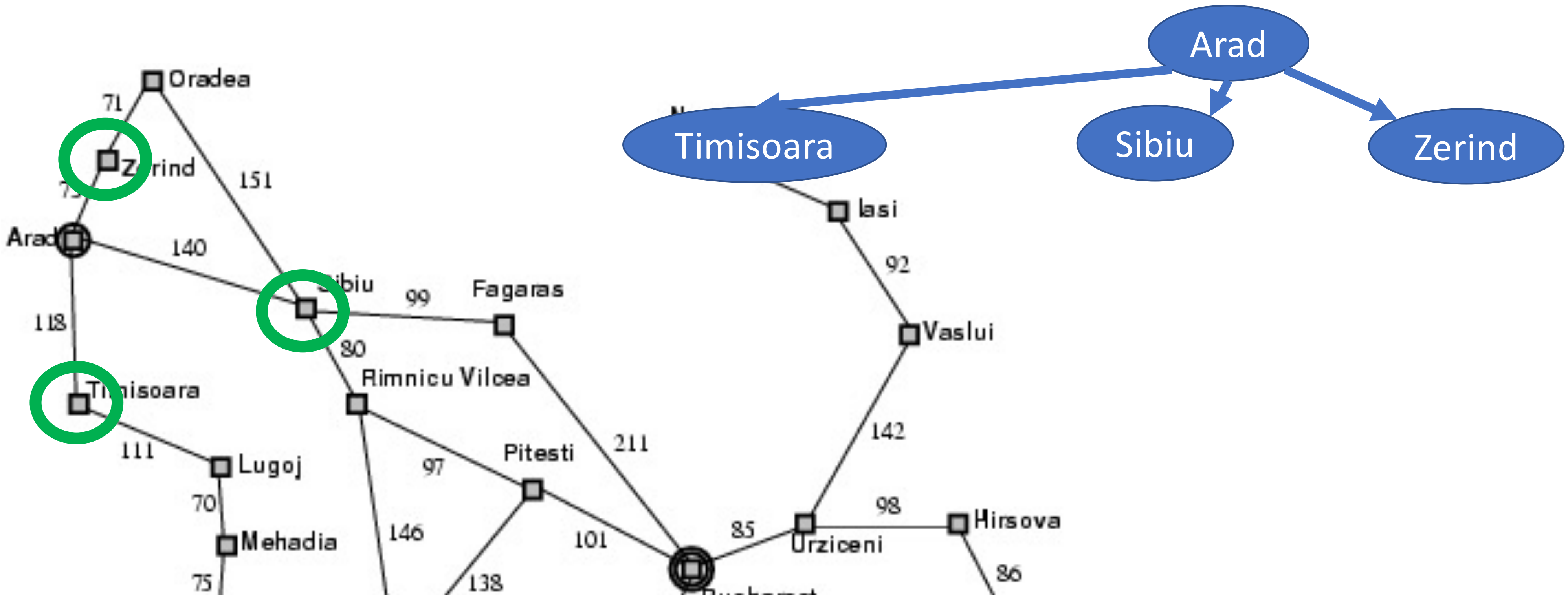
- Breadth-first search (BFS): Next node expanded is the one with the fewest required actions
  - Frontier is a queue
  - First node into the queue is the first one expanded (FIFO)
- Uniform cost search (UCS): Next node expanded is the one with the lowest accumulated path cost
  - Frontier is a ***priority queue***
  - ***Lowest-cost node*** is the first one expanded

Example of UCS: Romania



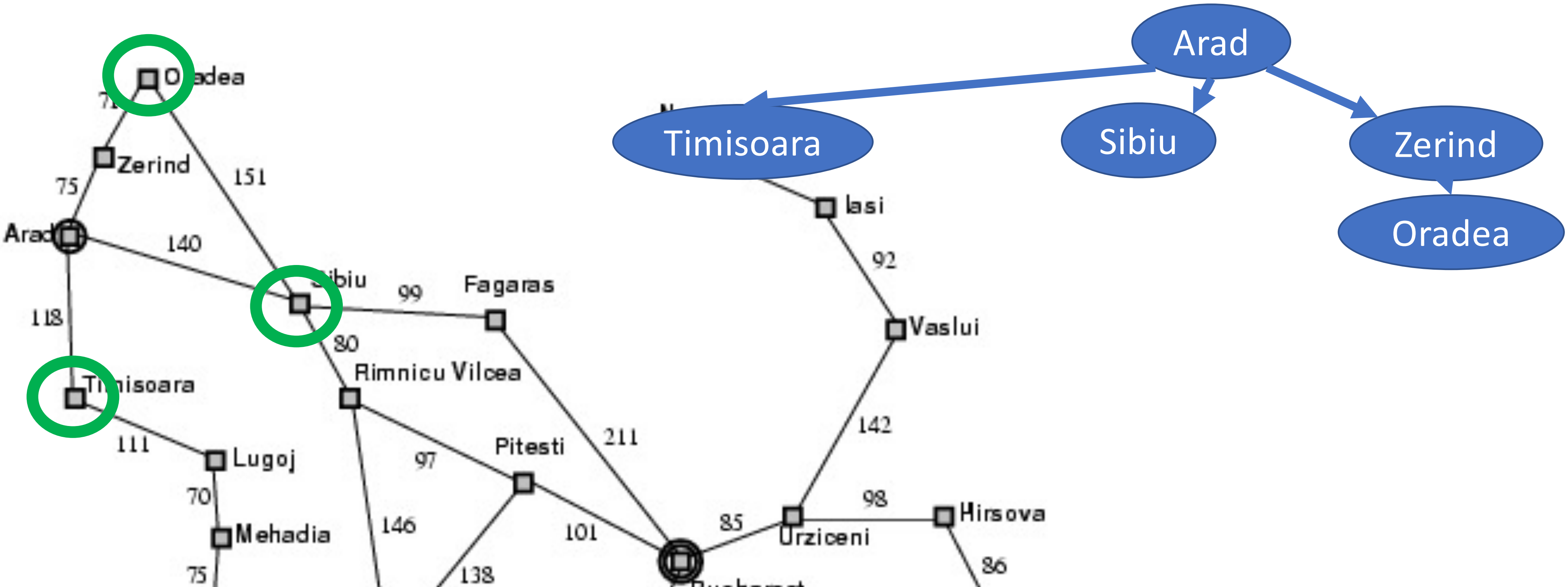
Arad:0

Example of UCS: Romania



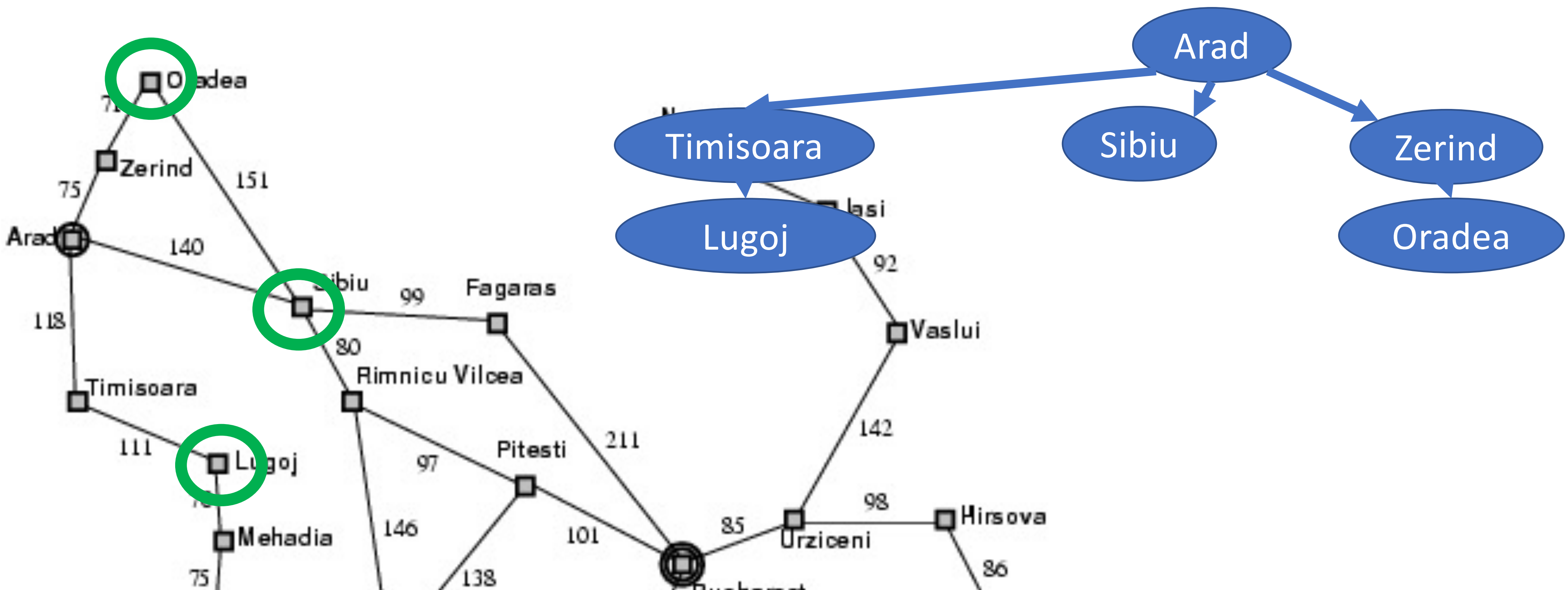
<u>Zerind:75, Timisoara:118, Sibiu:140</u>

Example of UCS: Romania



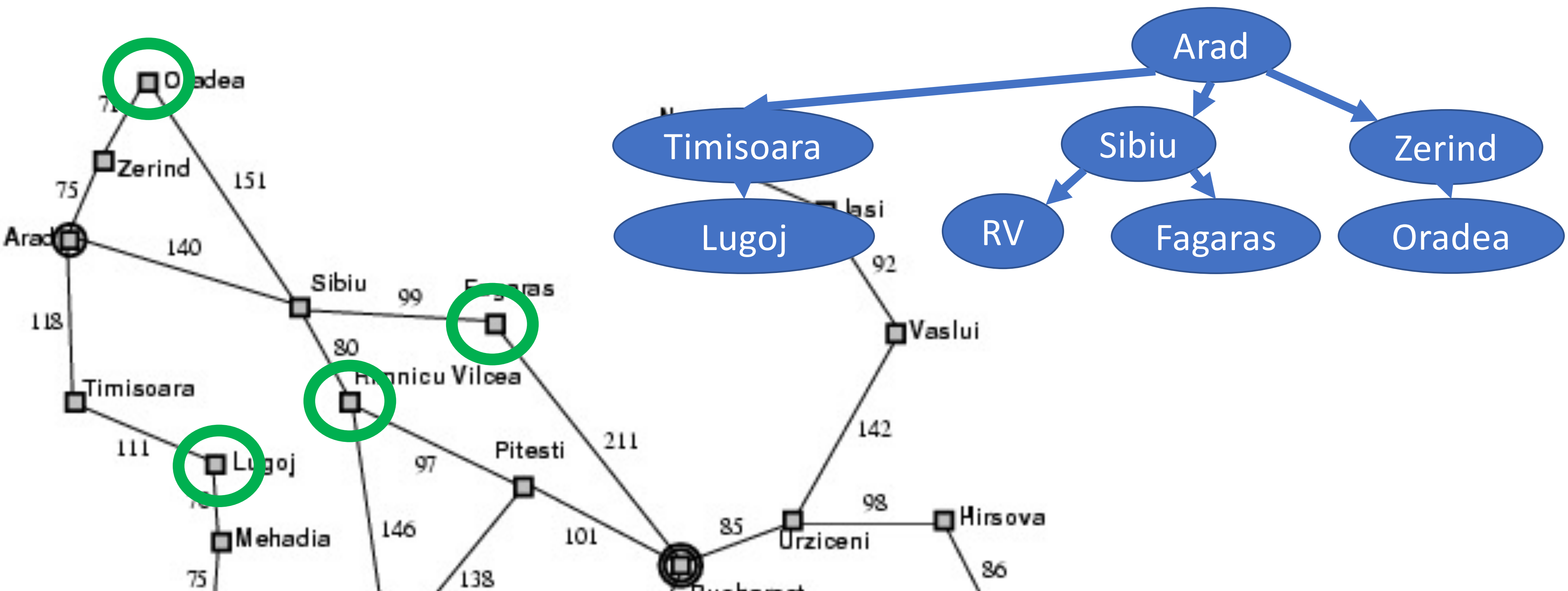
Timisoara:118, Sibiu:140, **Oradea:146**

Example of UCS: Romania



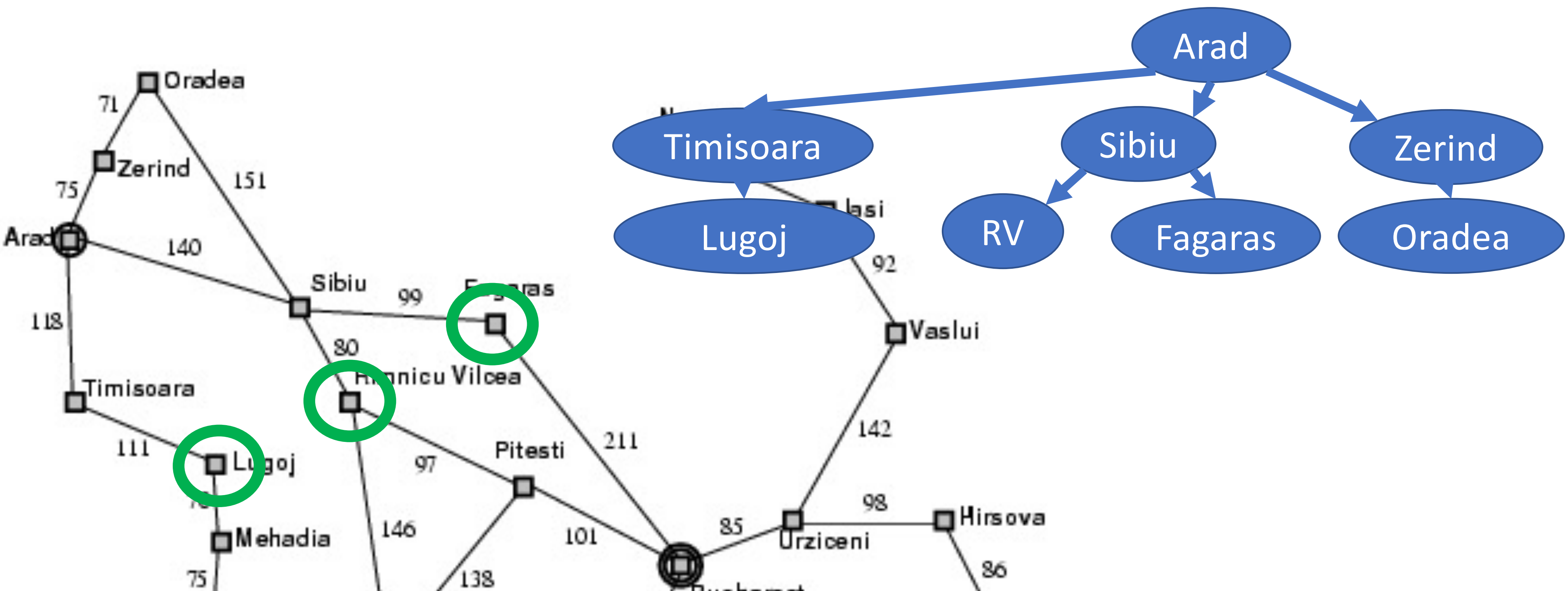
Sibiu:140, Oradea:146, **Lugoj:239**

Example of UCS: Romania



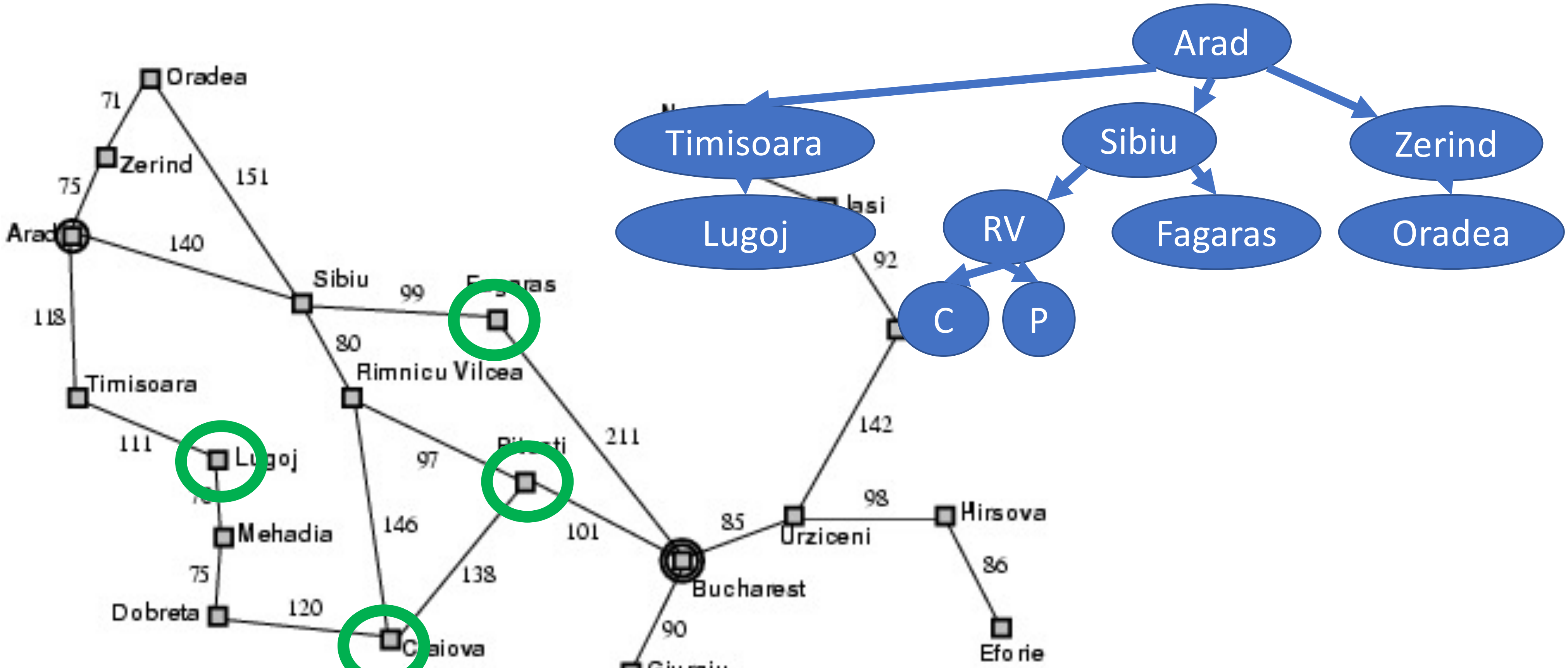
Oradea:146, **Ramnicu Valcea:220**, Lugoj:239, **Fagaras:239**

Example of UCS: Romania



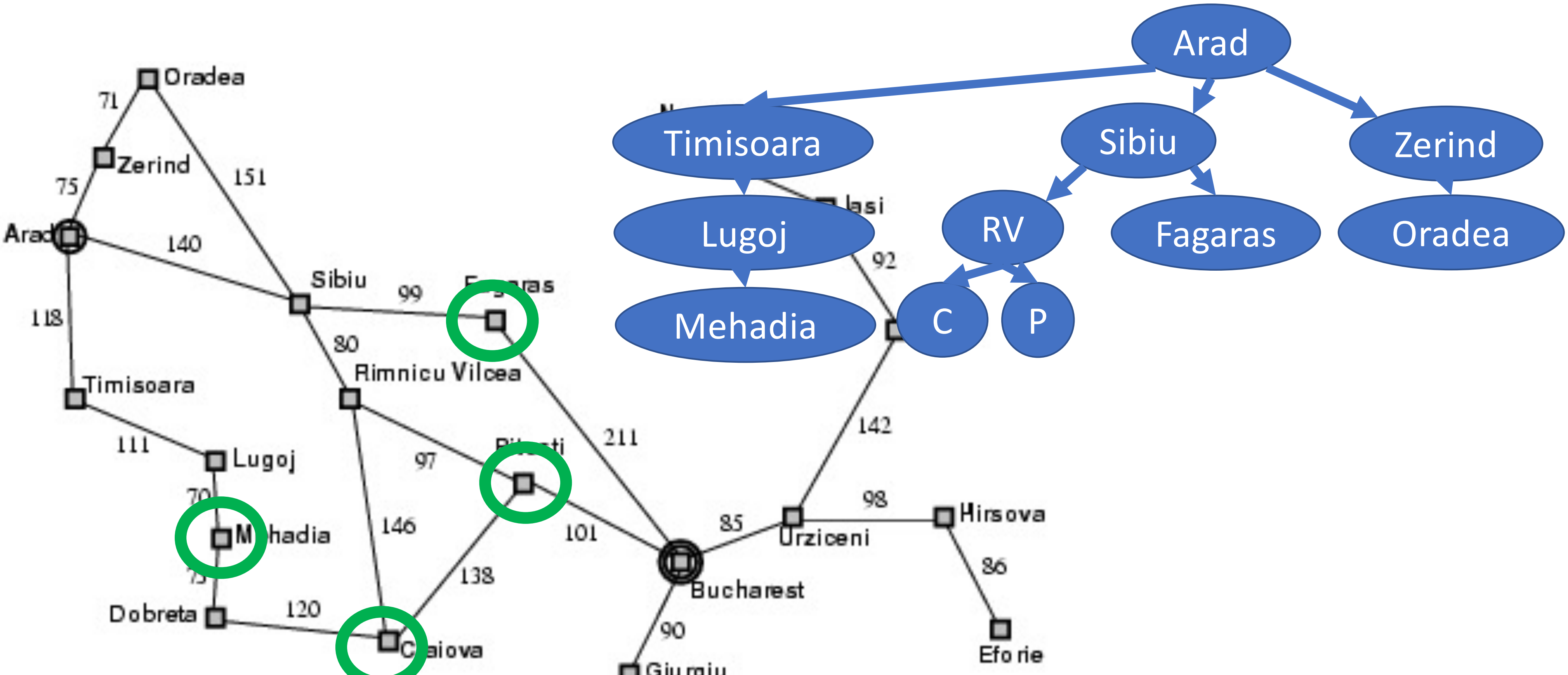
Ramnicu Valcea:220, Lugoj:239, Fagaras:239

Example of UCS: Romania



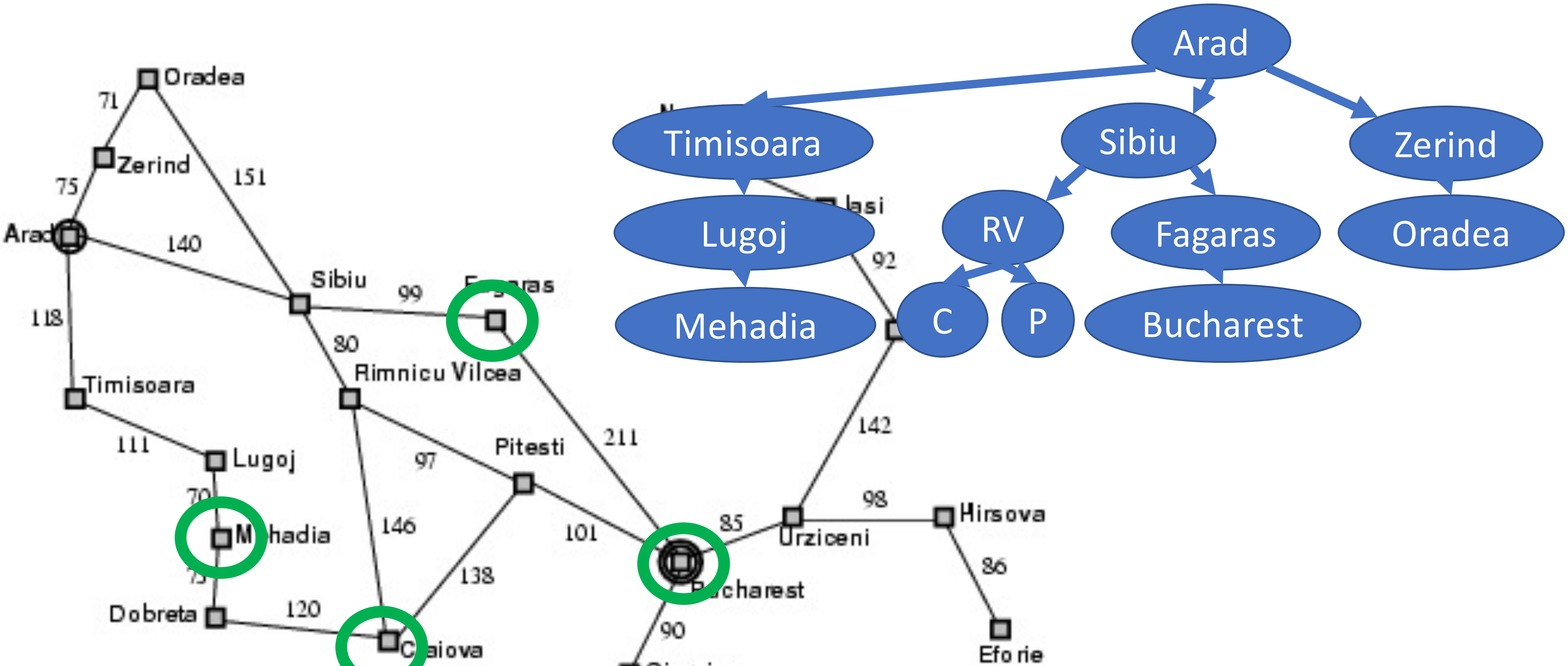
Lugoj:239, Fagaras:239, **<u>Pitesti:317, Craiova:366</u>**

Example of UCS: Romania



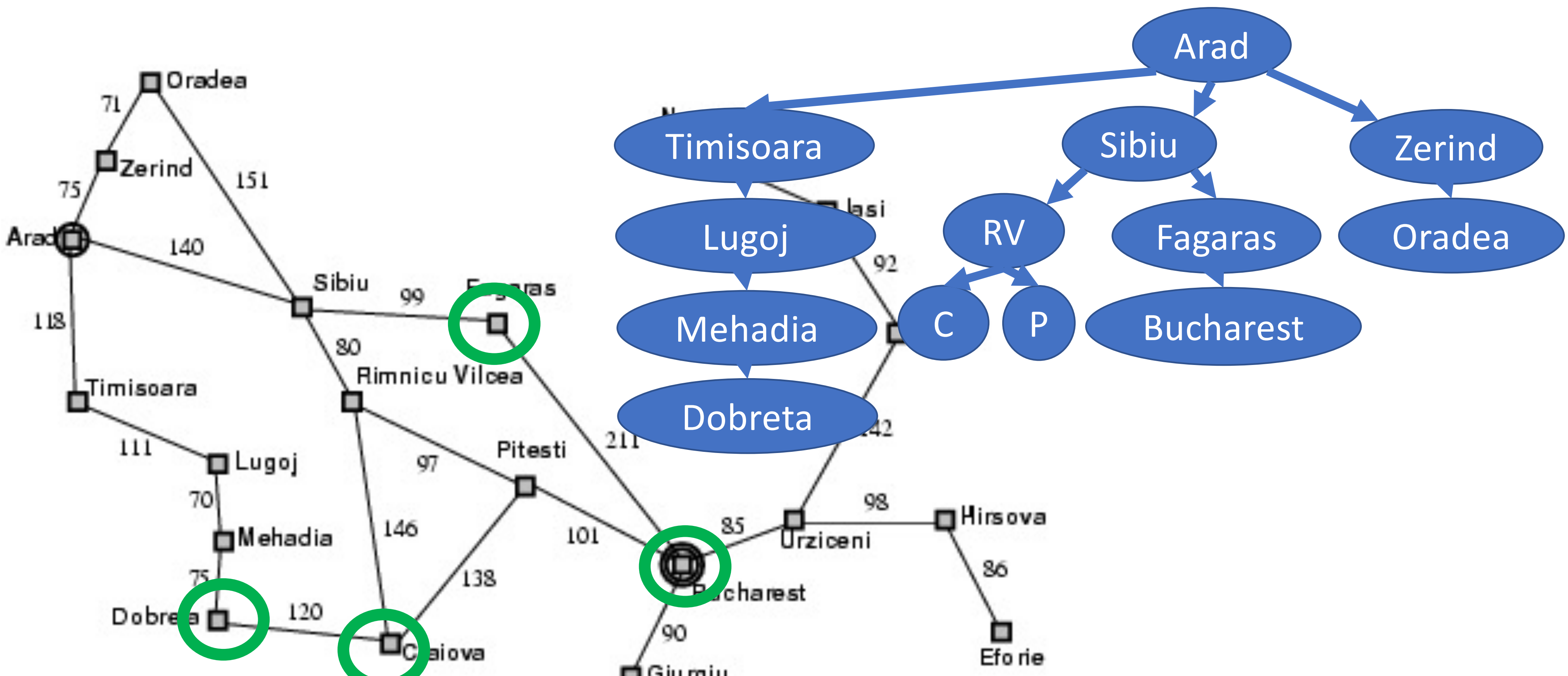
Fagaras:239, **<u>Mehadia:309</u>**, Pitesti:317, Craiova:366

Example of UCS: Romania



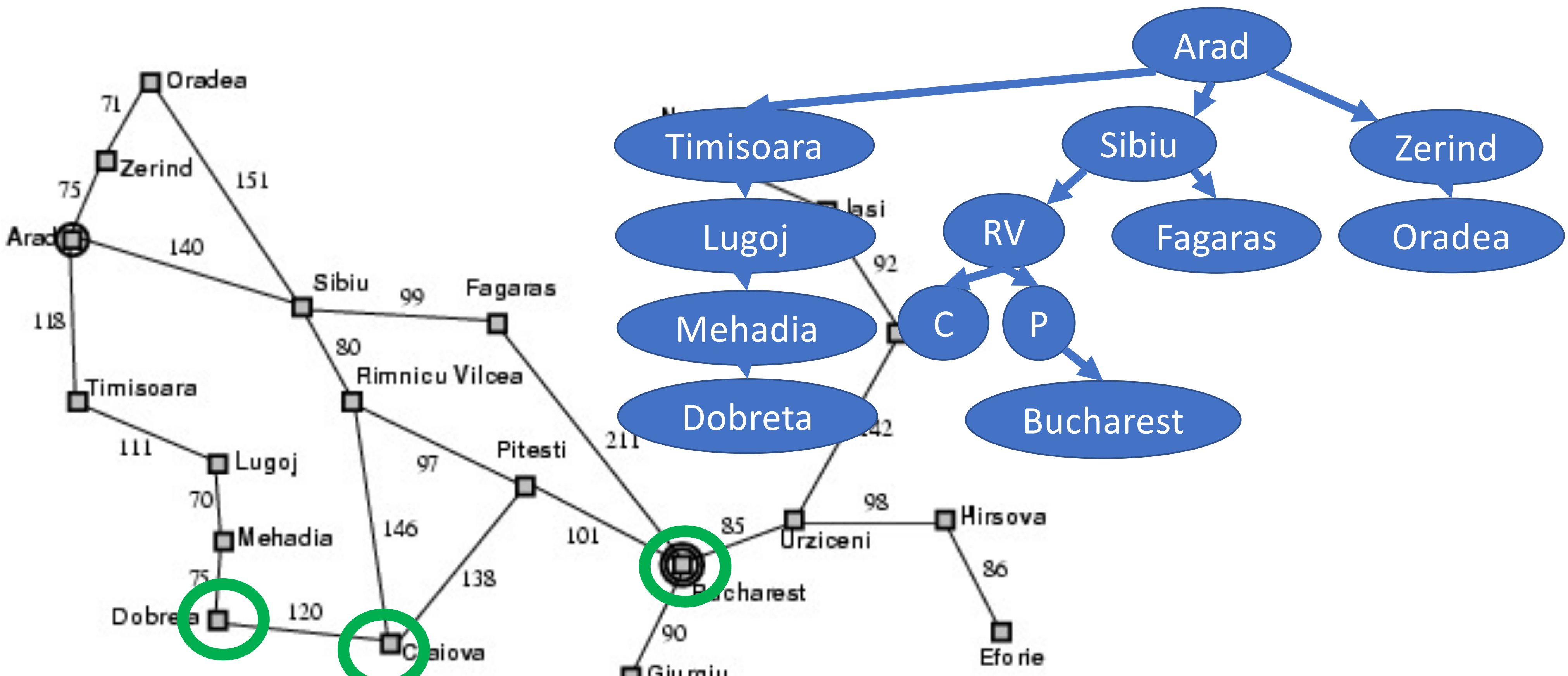
Mehadia:309, Pitesti:317, Craiova:366, **Bucharest:450**

Example of UCS: Romania



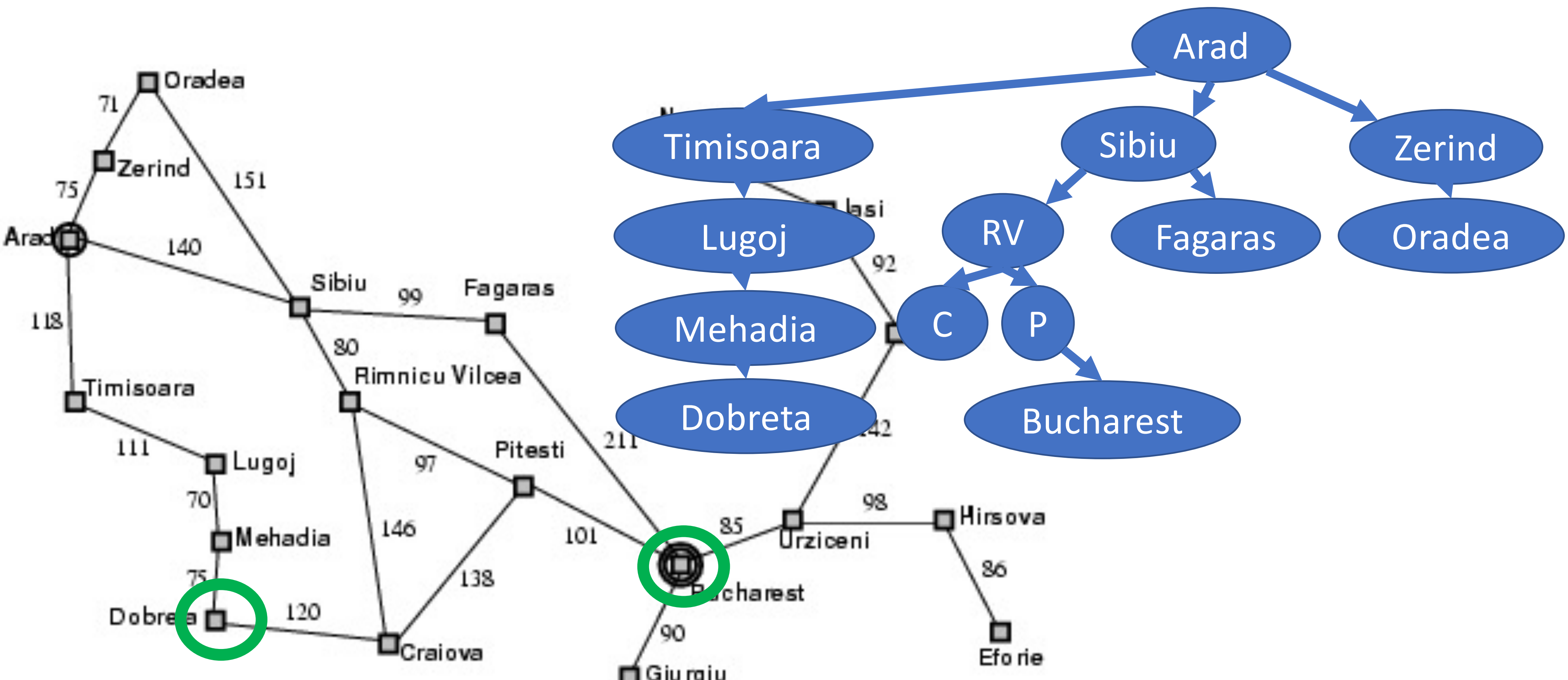
Pitesti:317, Craiova:366, **<u>Dobreta:384</u>**, Bucharest:450

Example of UCS: Romania



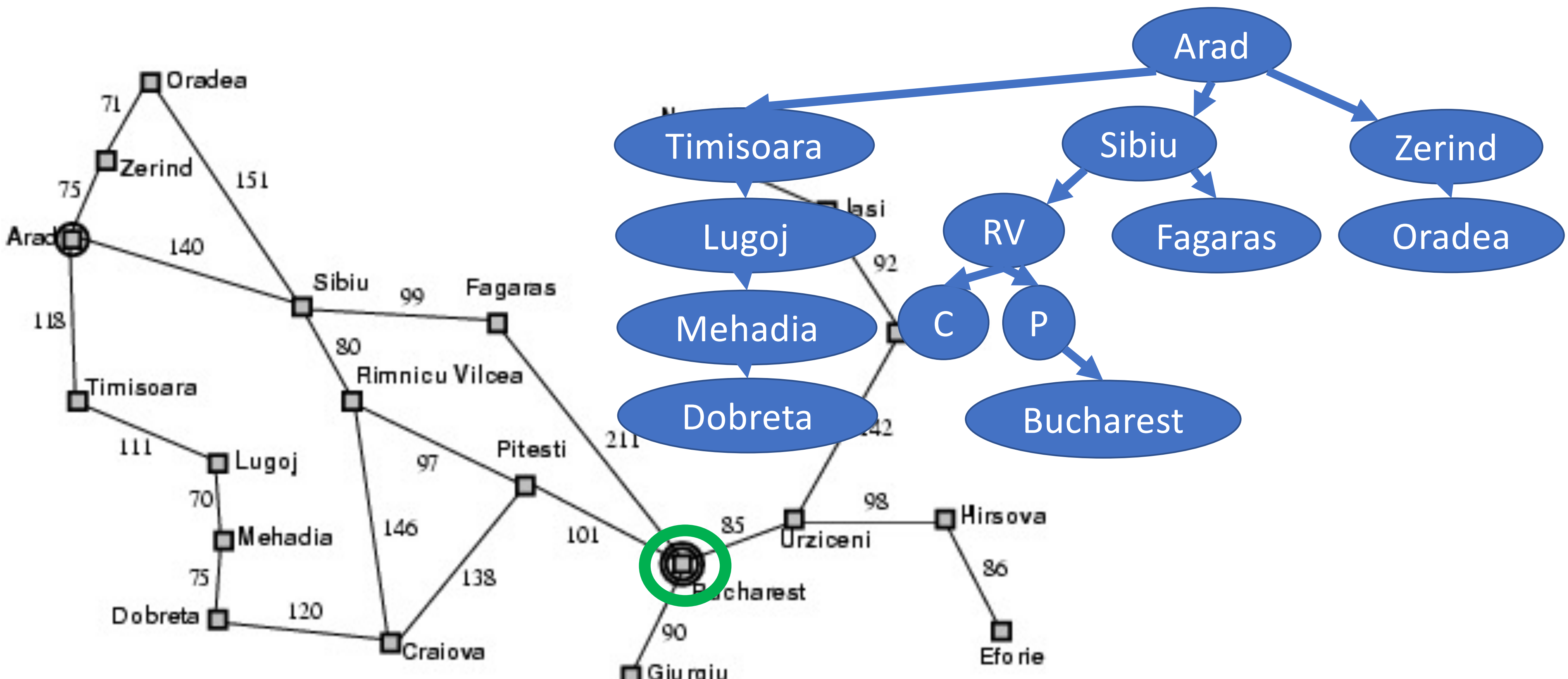
Craiova:366, Dobreta:384, **Bucharest:418**

Example of UCS: Romania



Dobreta:384, Bucharest:418

Example of UCS: Romania



Bucharest:418

GOAL!!!!   GOL!!!!!

Image by Rick Dikeman, GFDL 1996, https://commons.wikimedia.org/wiki/File:Football_iu_1996.jpg

Outline of today's lecture

1. Uniform Cost Search (UCS): like BFS, but for actions that have different costs
   - **Complete**: always finds a solution, if one exists
   - **Optimal:** finds the best solution
   - **Time complexity** = # nodes that have cost < goal
   - **Space complexity** = # nodes that have cost < goal
2. Heuristics, e.g., Manhattan distance
3. Greedy Best-first search
4. A*: Like UCS but adds an estimate of the remaining path length
   - **Complete**: always finds a solution, if one exists
   - **Optimal:** finds the best solution
   - **Time complexity** = # nodes that have cost+heuristic < goal
   - **Space complexity** = # nodes that have cost+heuristic < goal

Heuristics main idea

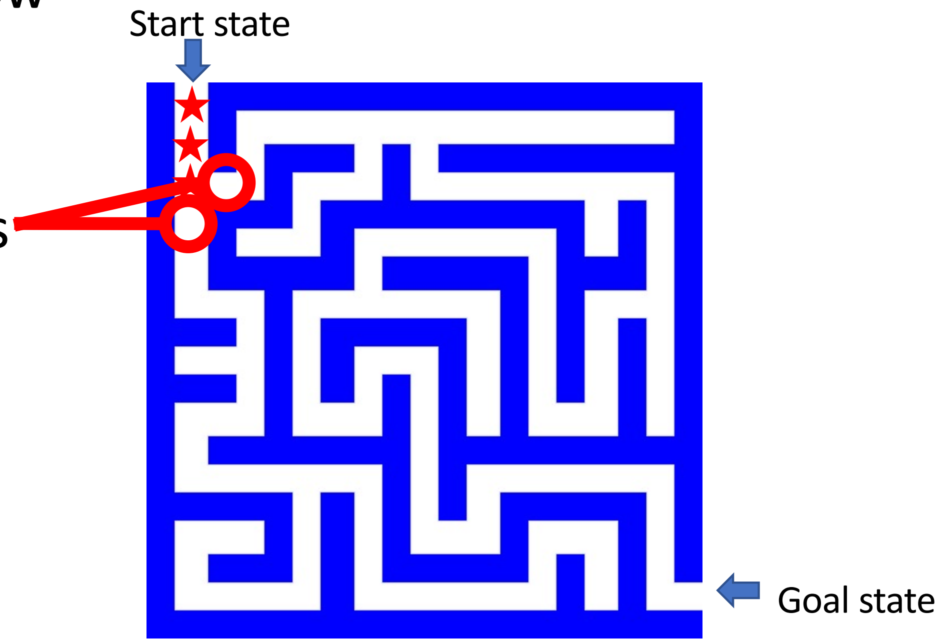
Instead of choosing the node with the smallest total cost so far (UCS),

why not choose the node that's CLOSEST TO GOAL,

and expand that one first?

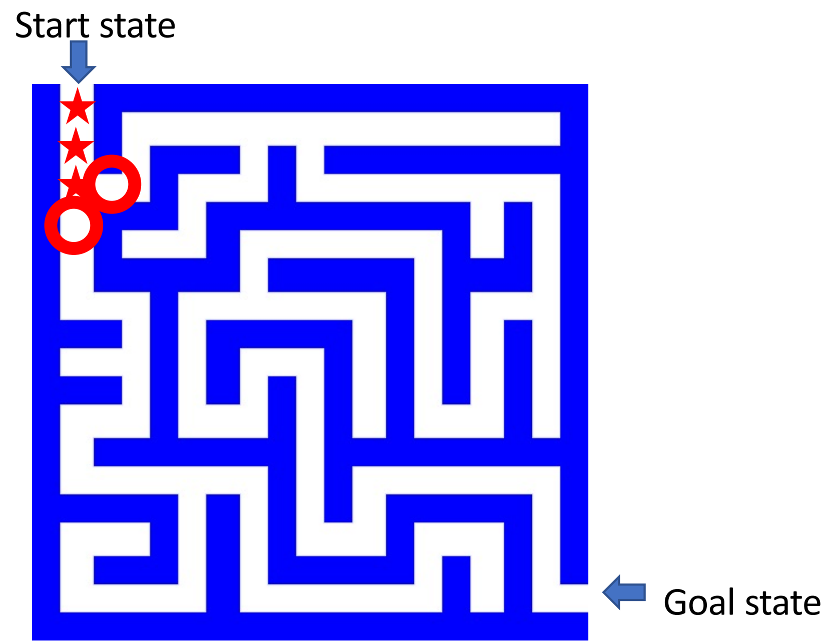
Why not choose the node CLOSEST TO GOAL?

- Answer: because we don't know which node that is!!
- Example: which of these two is closest to goal?



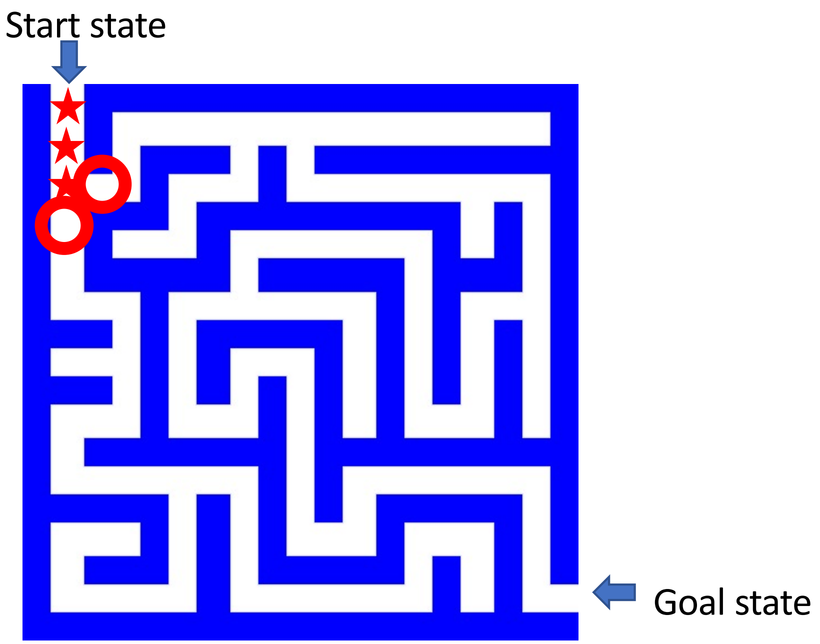
We don’t know which state is closest to goal

- Finding the shortest path is the whole point of the search
- If we already knew which state was closest to goal, there would be no reason to do the search
- Figuring out which one is closest, in general, is a complexity $O\{b^d\}$ problem.



Search heuristics: estimates of distance-to-goal

- Often, even if we don't know the distance to the goal, we can estimate it.
- This estimate is called a heuristic.
- A heuristic is useful if:
  1. **<u>Accurate</u>**: $h(n) \approx d(n)$, where $h(n)$ is the heuristic estimate, and $d(n)$ is the true distance to the goal
  2. **<u>Cheap:</u>** It can be computed in complexity less than $O\{b^d\}$

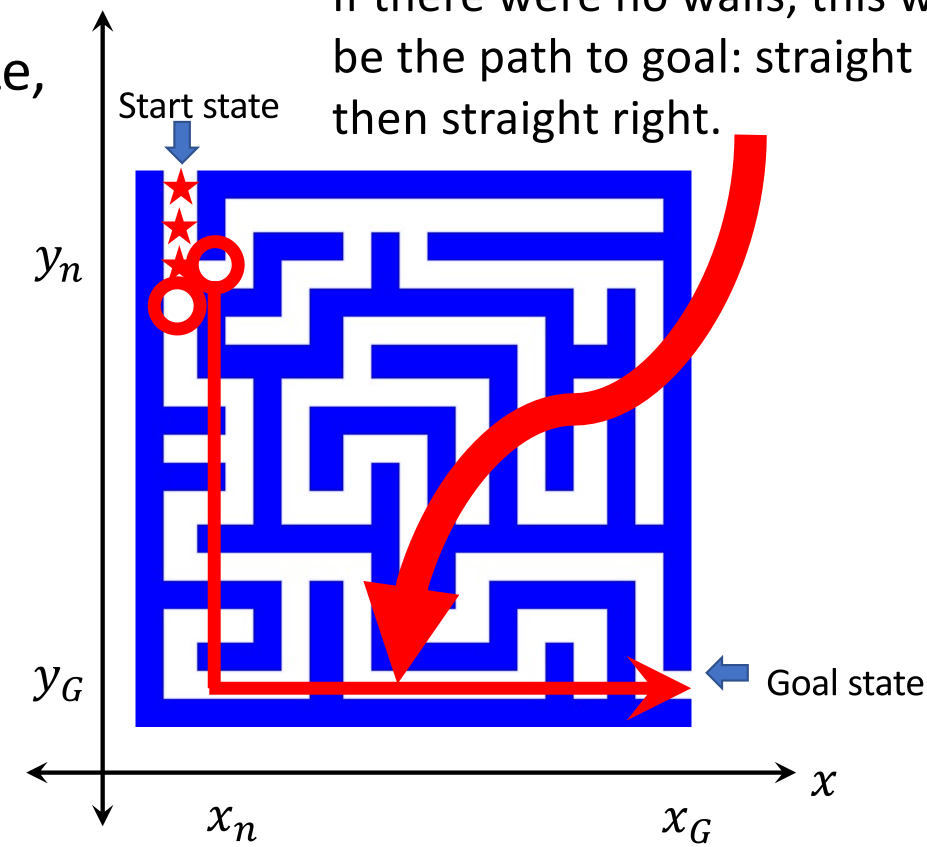


Example heuristic: Manhattan distance

If there were no walls in the maze, then the number of steps from position (x_n, y_n) to the goal position (x_G, y_G) would be

$$h(n) = |x_n - x_G| + |y_n - y_G|$$

If there were no walls, this would be the path to goal: straight down, then straight right.



Outline of today's lecture

1. Uniform Cost Search (UCS): like BFS, but for actions that have different costs
   - **Complete**: always finds a solution, if one exists
   - **Optimal:** finds the best solution
   - **Time complexity** = # nodes that have cost < goal
   - **Space complexity** = # nodes that have cost < goal
2. Heuristics, e.g., Manhattan distance
3. Greedy Best-first search
4. A*: Like UCS but adds an estimate of the remaining path length
   - **Complete**: always finds a solution, if one exists
   - **Optimal:** finds the best solution
   - **Time complexity** = # nodes that have cost+heuristic < goal
   - **Space complexity** = # nodes that have cost+heuristic < goal

Greedy Best-First Search

Instead of choosing the node with the smallest total cost so far (UCS),

why not choose the node whose

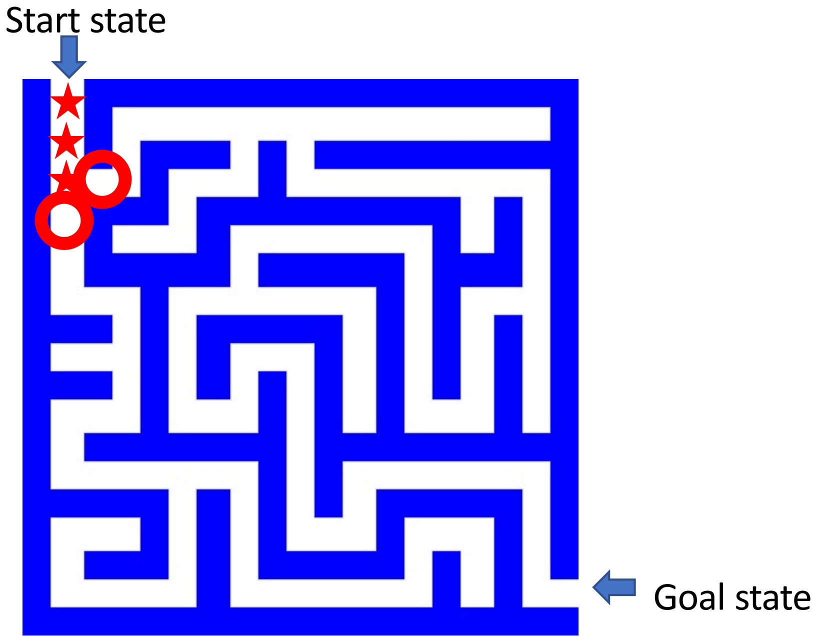
HEURISTIC ESTIMATE

indicates that it might be

CLOSEST TO GOAL?

Greedy Search Example

According to the Manhattan distance heuristic, these two nodes are equally far from the goal, so we have to choose one at random.

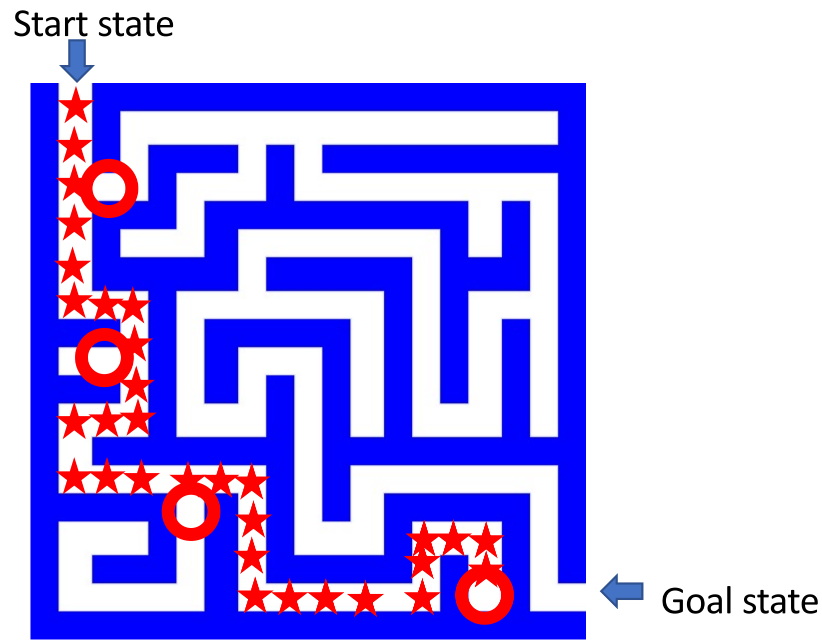


Greedy Search Example

If our random choice goes badly, we might end up very far from the goal.

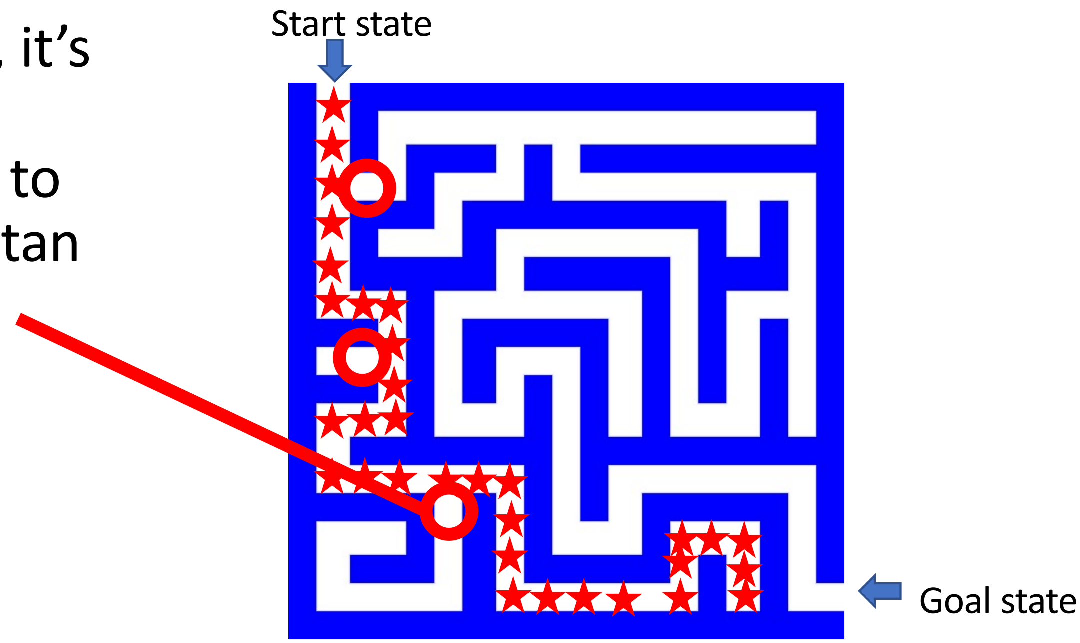
★ = states in the explored set

○ = states on the frontier



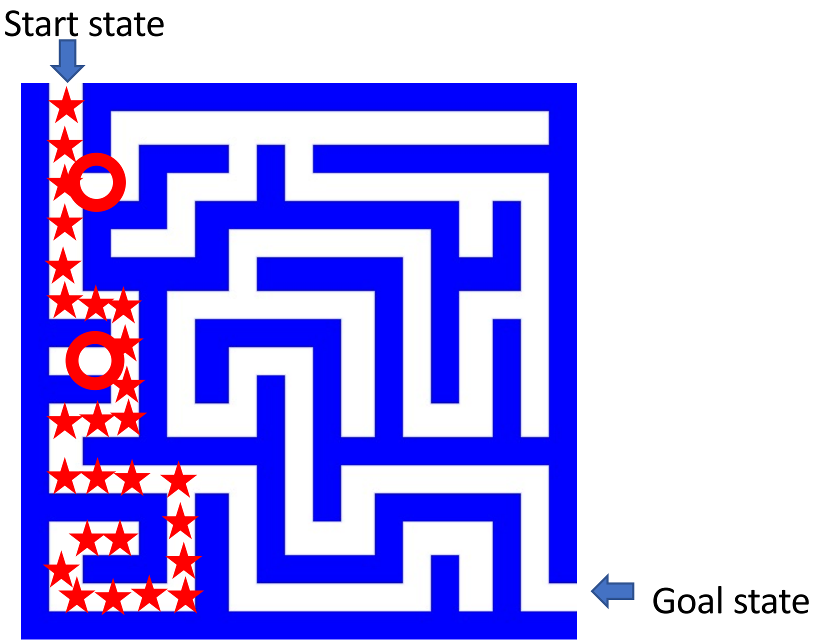
The problem with Greedy Search

Having gone down a bad path, it's very hard to recover, because now, the frontier node closest to goal (according to the Manhattan distance heuristic) is this one:



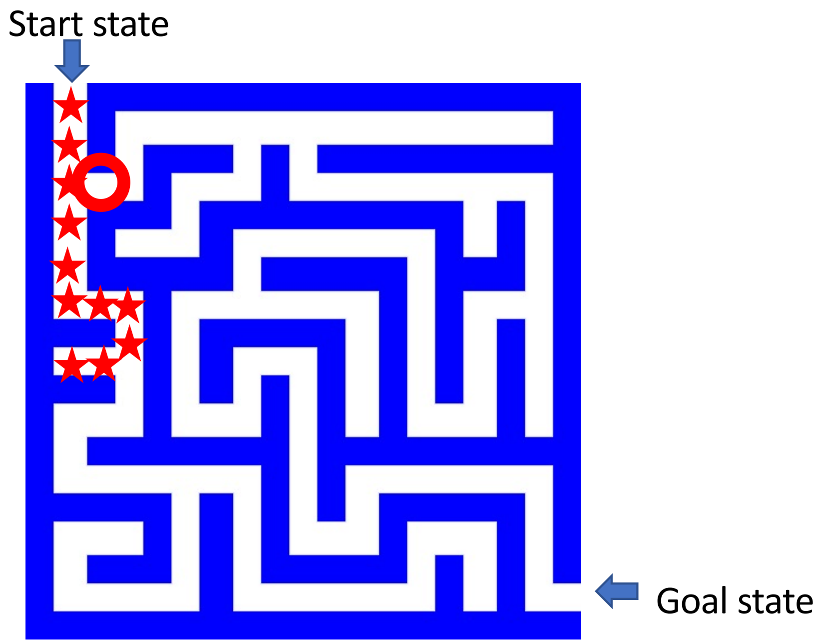
The problem with Greedy Search

That's not a useful path…



The problem with Greedy Search

Neither is that one…



What went wrong?

Outline of today's lecture

1. Uniform Cost Search (UCS): like BFS, but for actions that have different costs
   - **Complete**: always finds a solution, if one exists
   - **Optimal:** finds the best solution
   - **Time complexity** = # nodes that have cost < goal
   - **Space complexity** = # nodes that have cost < goal
2. Heuristics, e.g., Manhattan distance
3. Greedy Best-first search
4. A*: Like UCS but adds an estimate of the remaining path length
   - **Complete**: always finds a solution, if one exists
   - **Optimal:** finds the best solution
   - **Time complexity** = # nodes that have cost+heuristic < goal
   - **Space complexity** = # nodes that have cost+heuristic < goal

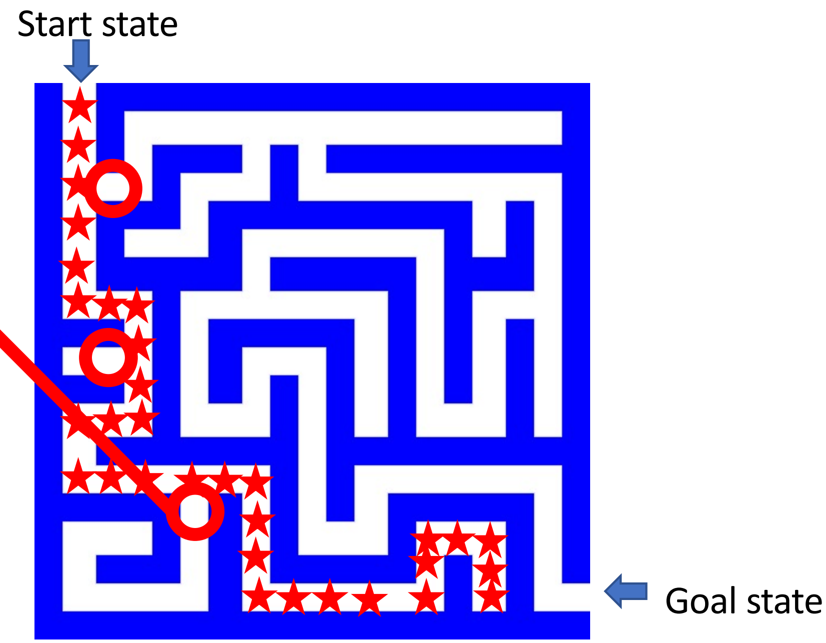
The problem with Greedy Search

Among nodes on the frontier, this one seems closest to goal (smallest $h(n)$, where $h(n) \leq d(n)$).

But it's also farthest from the start. Let's say $g(n)$ = total path cost so far.

So the total distance from start to goal, going through node n, is

$$c(n) = g(n) + d(n) \geq g(n) + h(n)$$



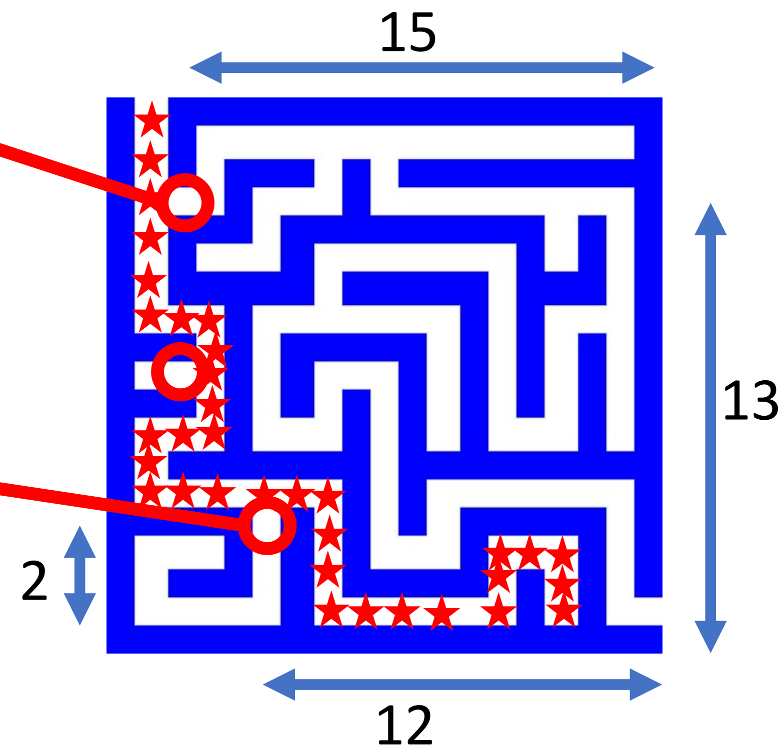
The problem with Greedy Search

Of these three nodes, this one has the smallest $g(n) + h(n)$

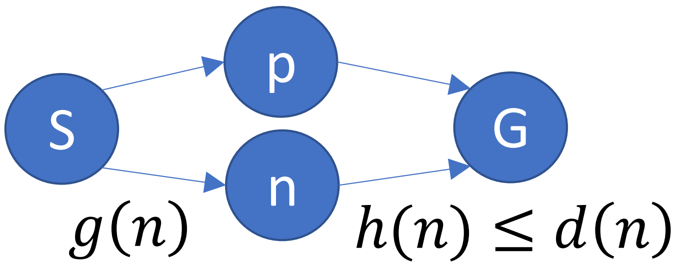
$$(g(n) + h(n) = 4 + 28 = 32)$$

So if we want to find the lowest-cost path, then it would be better to try that node, instead of this one, which has

$$g(n) + h(n) = 21 + 14 = 35$$



A* search



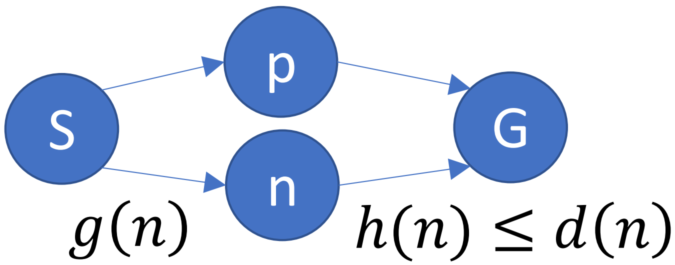
In an A* search, we keep track of TWO things about each path:

1. The cost from START to NODE n. Let's call this $g(n)$.
2. The cost from NODE n to GOAL.
   - The true cost is $d(n)$. But it's unknown.
   - The heuristic estimate is $h(n)$, and $d(n) \ge h(n)$.

The total cost of the best path that goes START→NODE n→GOAL is:

- $c(n) = g(n) + d(n)$. But it's unknown.
- Known to be greater than or equal to $f(n) = g(n) + h(n)$.

A* search



The total cost of the best path that goes START→NODE n→GOAL is:

- $c(n) = g(n) + d(n)$. But it's unknown.
- Known to be greater than or equal to $f(n) = g(n) + h(n)$.

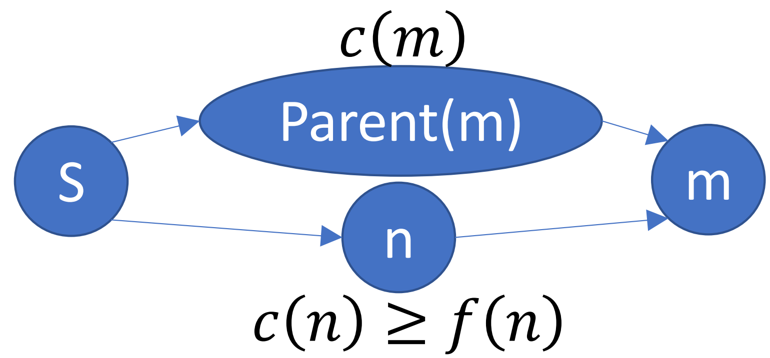
An A* search is a search in which the frontier is a priority queue, sorted in order of increasing f(n):

$$\{(m, f(m)), (n, f(n)), (p, f(p)), (q, f(q)), \dots\}$$

…where "priority queue" means that $f(m) \le f(n) \le f(p) \le f(q) \le \cdots$

Thus, the next node we expand, n, is always the one that seems to be part of the shortest path between START and GOAL.

Optimality of A*

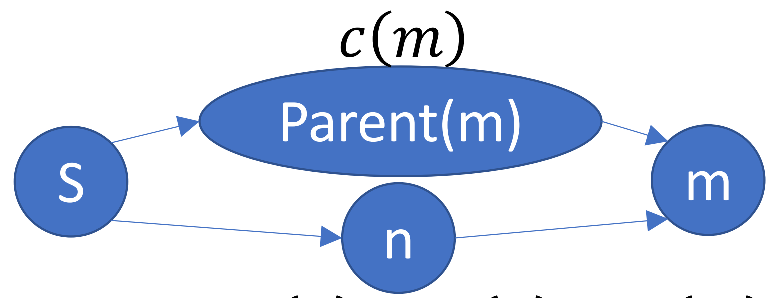


- Suppose that the frontier is a priority queue of tuples:
$$\{(m, f(m)), (n, f(n)), (p, f(p)), (q, f(q)), \ldots\}$$
…where "priority queue" means that $f(m) \leq f(n) \leq f(p) \leq f(q) \leq \cdots$

- Suppose we expand the first node, and discover that it's the goal:
$$\text{State}(m) = \text{GOAL!}$$

- Does that mean that the path from START to GOAL specified by back-tracking $\text{Parent}(m)$ is the SHORTEST path to the goal?

Optimality of A*



$c(n) \geq f(n) \geq c(m)$

Suppose that the frontier is a priority queue of tuples:

$$\{(m, f(m)), (n, f(n)), (p, f(p)), (q, f(q)), \ldots\}$$

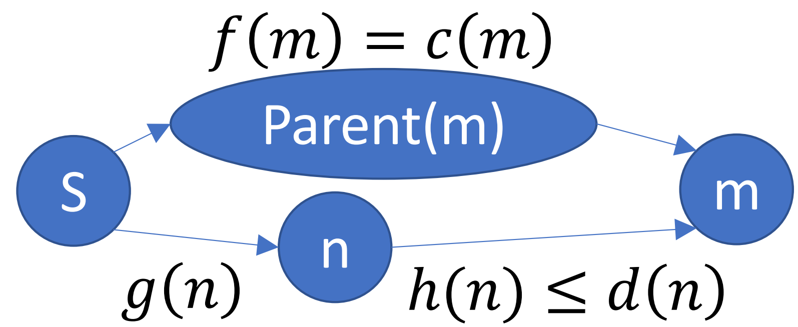
1. $f(m) = g(m) + h(m) \leq g(m) + d(m) = g(m) + 0 = c(m)$

So $f(m)$ is the cost to reach GOAL along the path through $\text{Parent}(m)$.

1. $f(m) \leq f(n) = g(n) + h(n) \leq g(n) + d(n) = c(n)$

So every other node has a higher cost than node m.

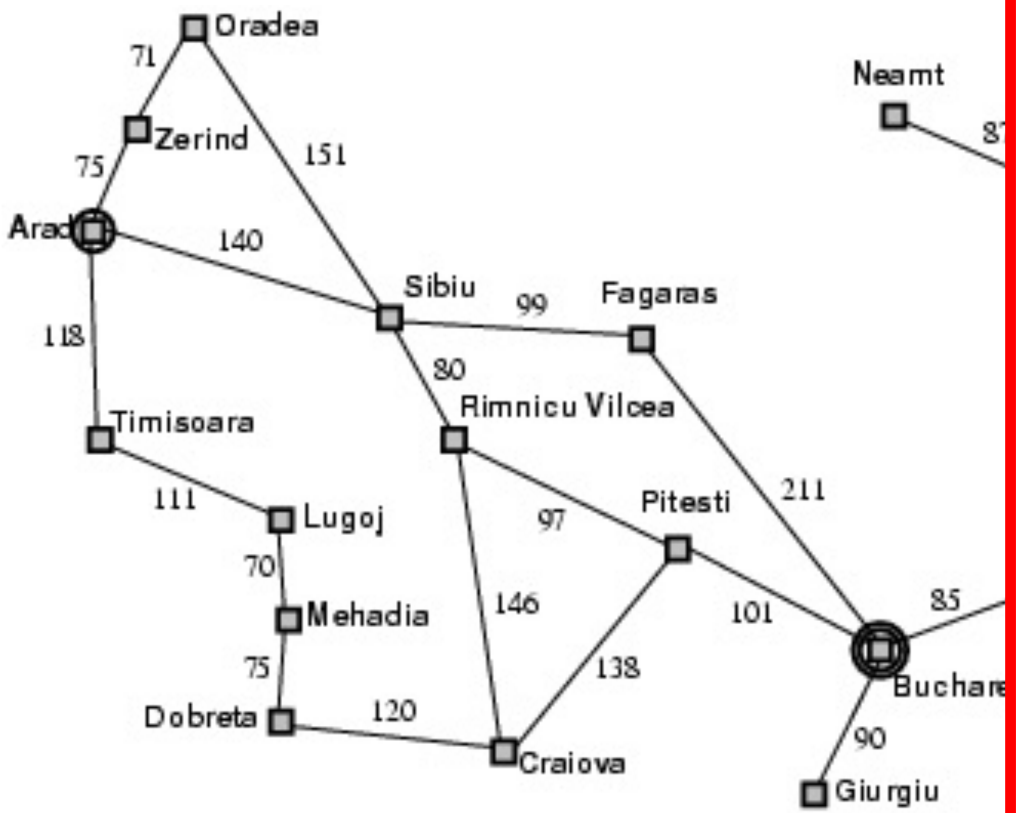
Optimality of A* Search



$$c(n) = g(n) + d(n) \geq g(n) + h(n) = f(n) \geq f(m)$$

- **<u>Definition:</u>** An admissible heuristic is a heuristic that satisfies the condition $d(n) \geq h(n)$.
- If $h(n)$ is admissible, and if the frontier is a priority queue sorted according to $g(n) + h(n)$, then
- the FIRST path to goal discovered by the tree search, path m, is guaranteed to be the SHORTEST path to goal.

Example of A*: Romania

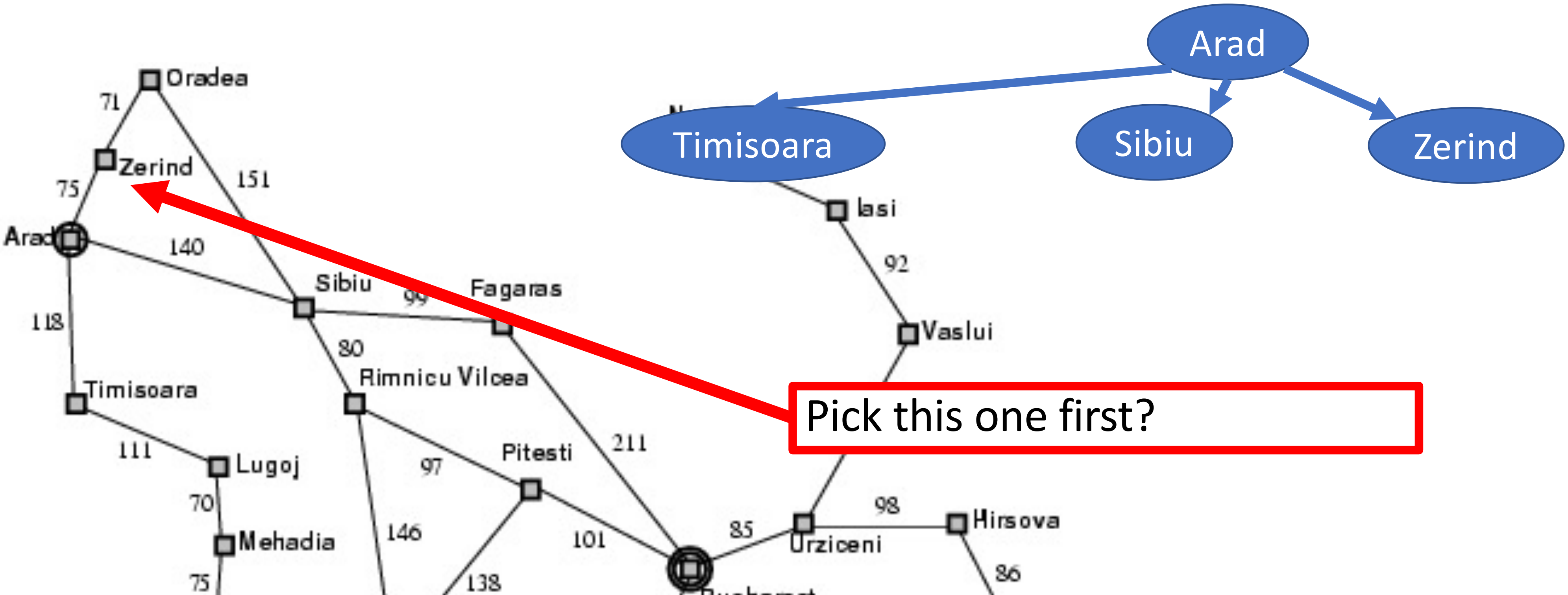


Suppose we don't know the distance from Sibiu to Bucharest on highways, but we DO know the distance "as the crow flies."

$h(n)$ = Euclidean distance (as the crow flies)

- Sibiu: h(n) = 260km
- Timisoara: h(n) = 410km
- Zerind: h(n) = 422km

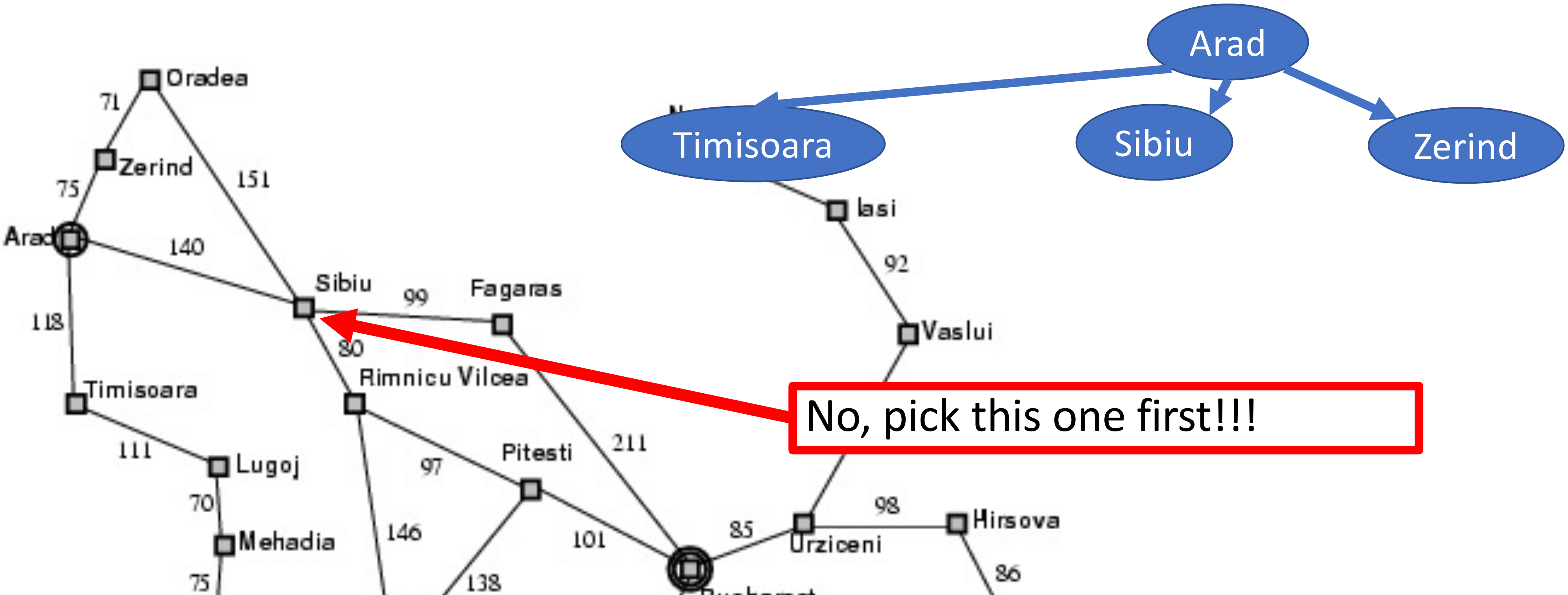
Romania using UCS



Pick this one first?

Zerind:75, Timisoara:118, Sibiu:140

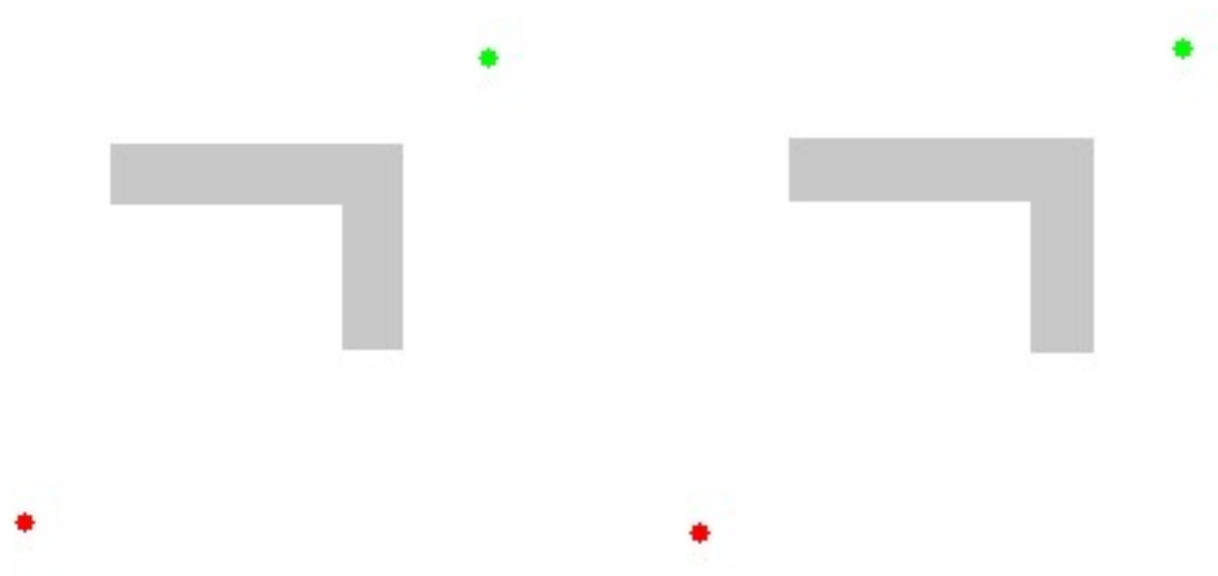
Romania using A*



Sibiu:140+260=400, Zerind:75+422=495, Timisoara:118+410=528

BFS vs. A* Search Example

The heuristic h(n)=Euclidean distance favors nodes on the main diagonal. Those nodes all have the same g(n)+h(n), so A* evaluates them first.



Outline of today's lecture

1. Uniform Cost Search (UCS): like BFS, but for actions that have different costs
   - **Complete**: always finds a solution, if one exists
   - **Optimal:** finds the best solution
   - **Time complexity** = # nodes that have cost < goal
   - **Space complexity** = # nodes that have cost < goal
2. Heuristics, e.g., Manhattan distance
3. Greedy Best-first search
4. A*: Like UCS but adds a **lower bound** of the remaining path length
   - **Complete**: always finds a solution, if one exists
   - **Optimal:** finds the best solution
   - **Time complexity** = # nodes that have cost+heuristic < goal
   - **Space complexity** = # nodes that have cost+heuristic < goal