Imperial College London

Uninformed Search
(Blind Search)

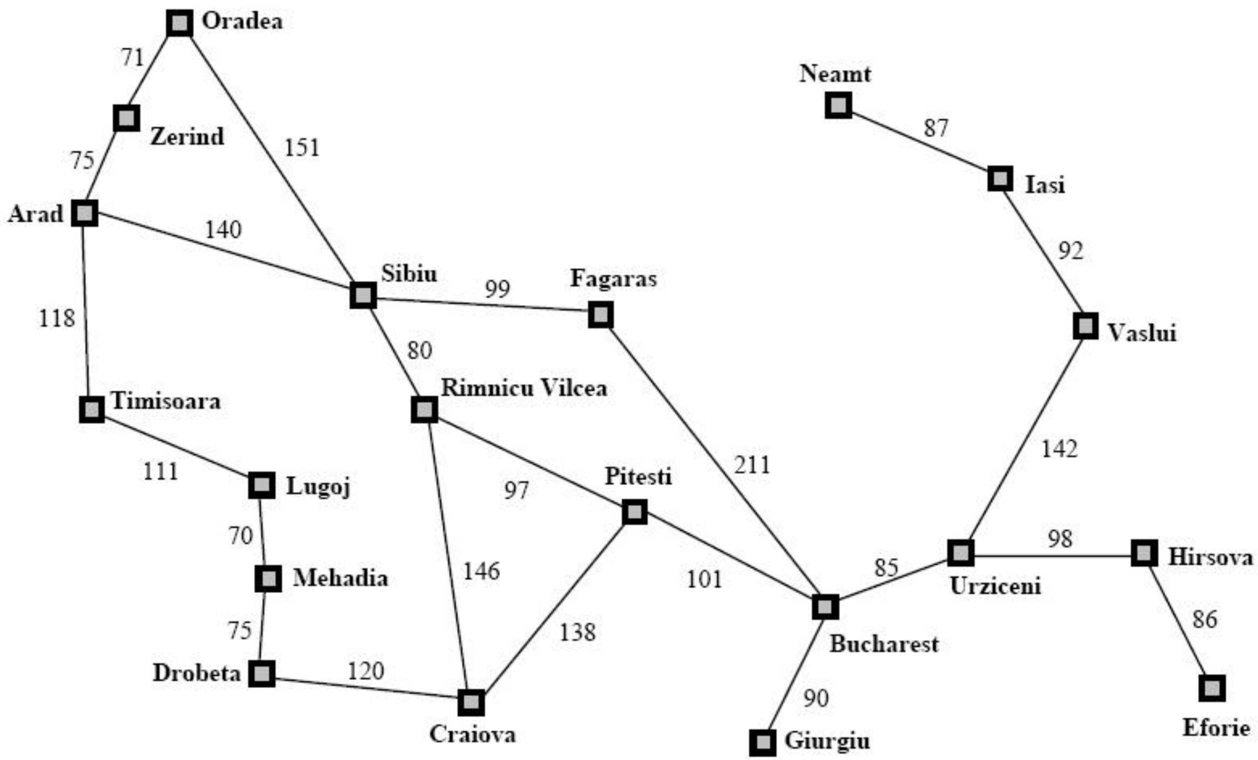
Murray Shanahan

Overview

- Depth-first search
- Breadth-first search
- Iterative deepening
- Uniform cost search

An Example Search Problem

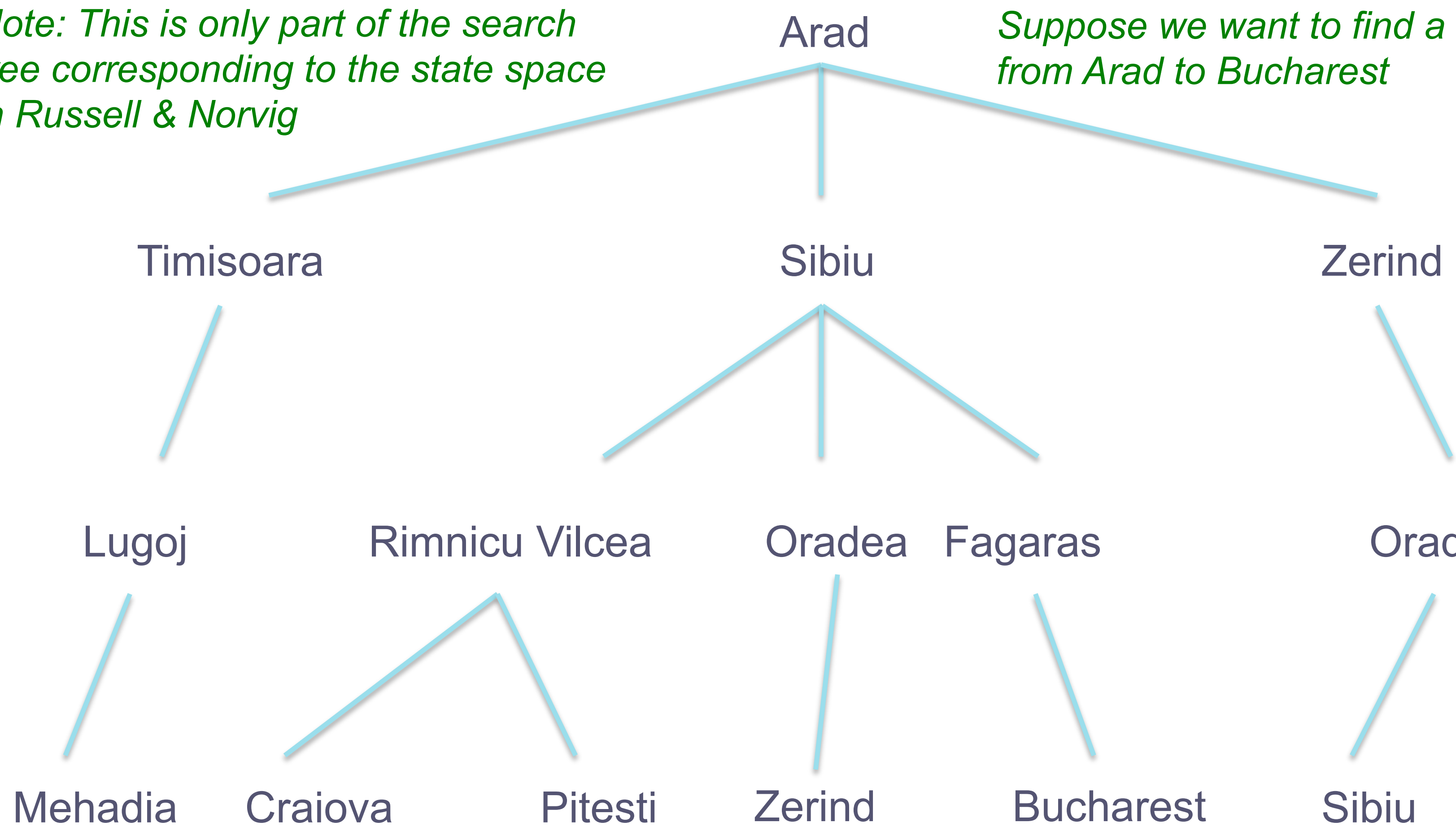
- Here is a *state-space* diagram of railway connections in Romania (taken from Russell & Norvig)
- Suppose we want to find a route from one city (the *initial state*) to another (the *goal state*)



Example Search Tree

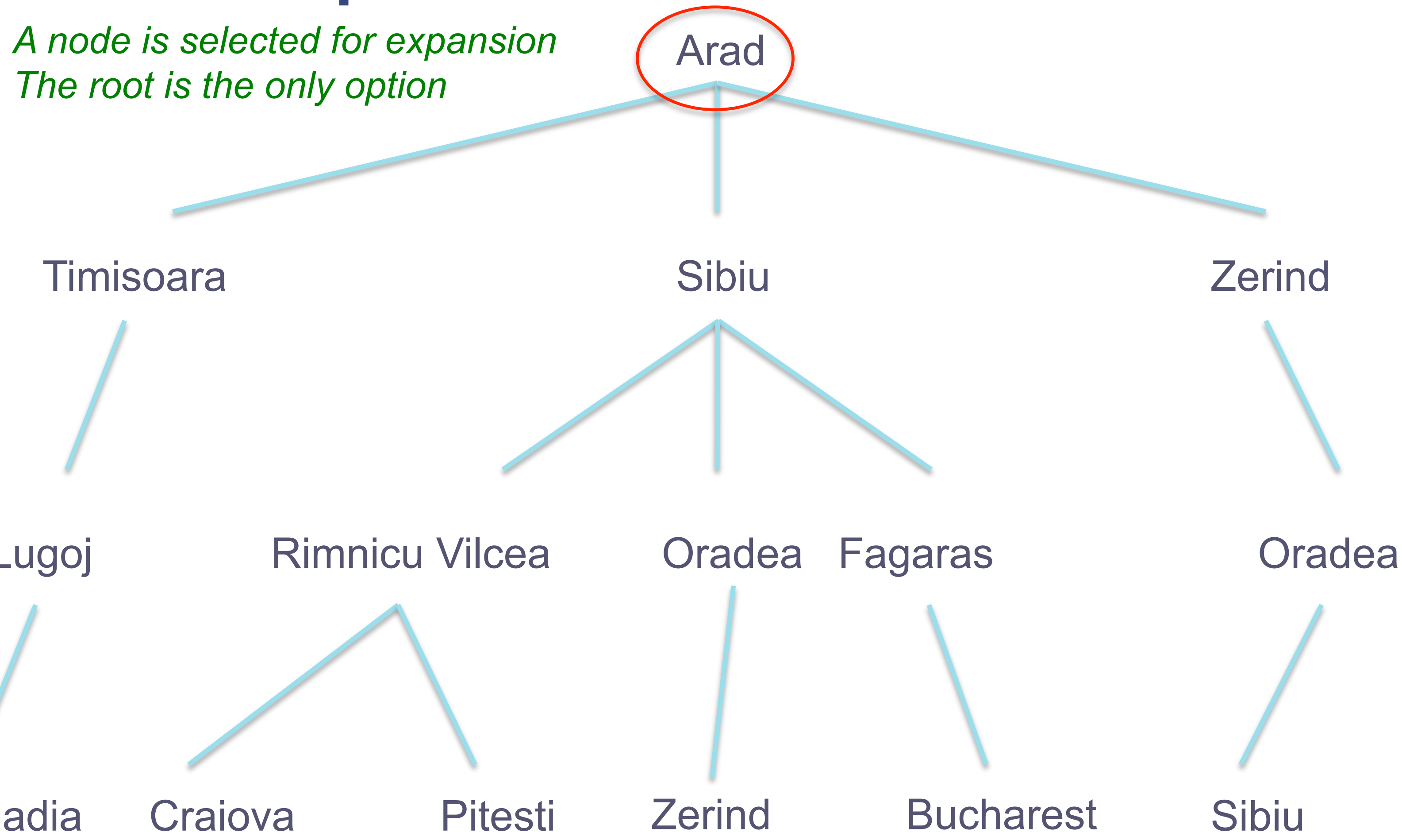
Note: This is only part of the search tree corresponding to the state space in Russell & Norvig

Suppose we want to find a route from Arad to Bucharest



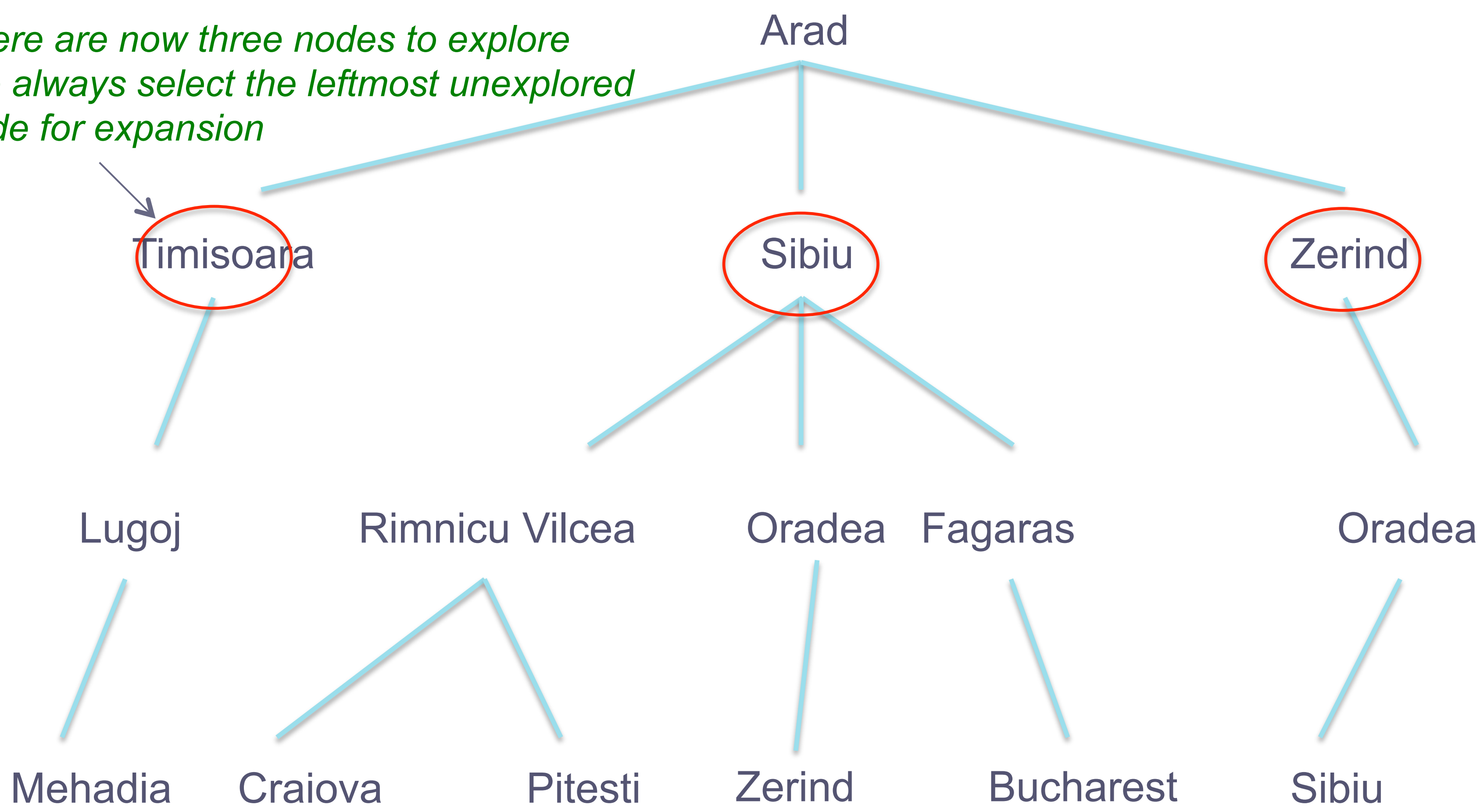
Depth-first Search

Depth-first Search 1



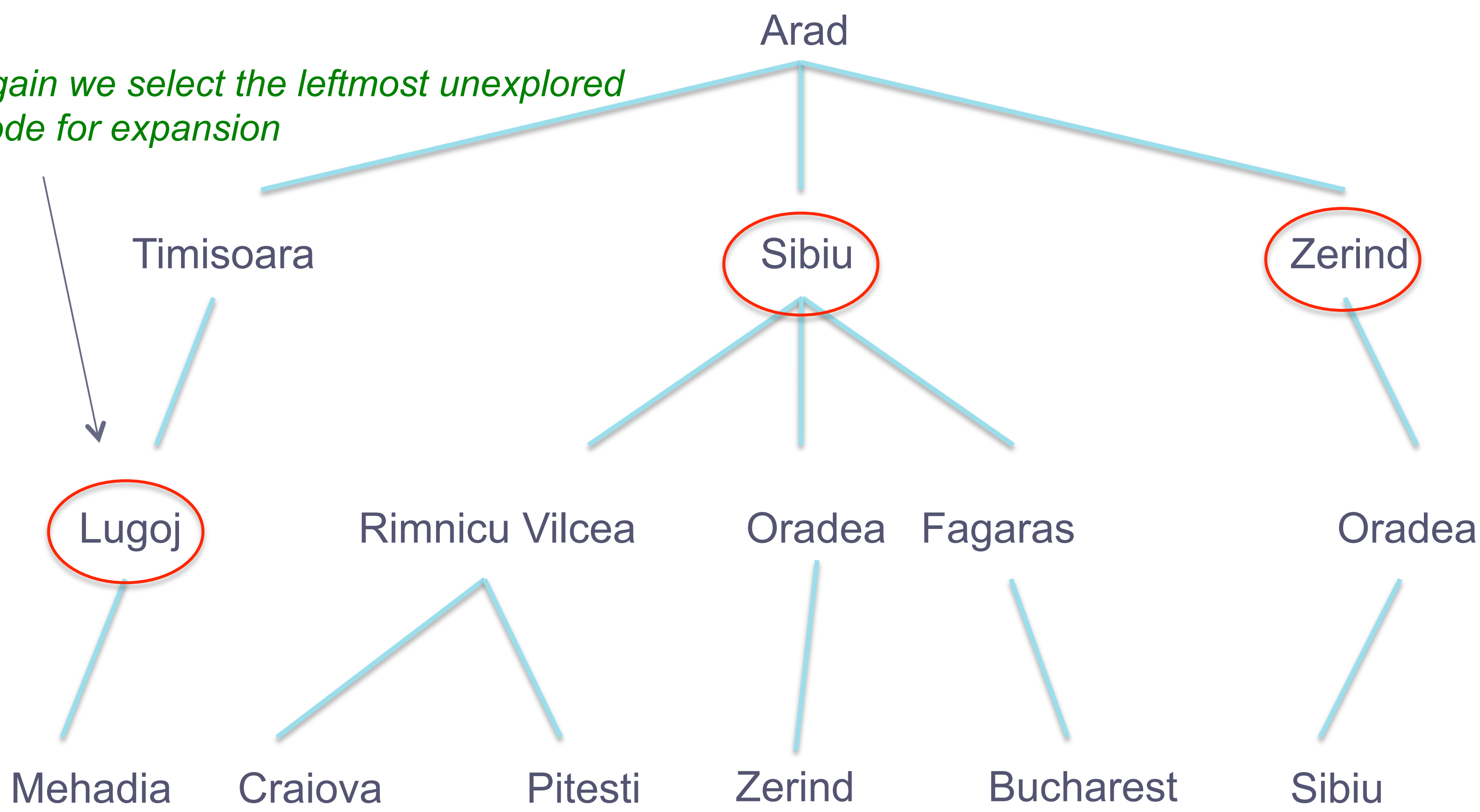
Depth-first Search 2

There are now three nodes to explore
We always select the leftmost unexplored node for expansion

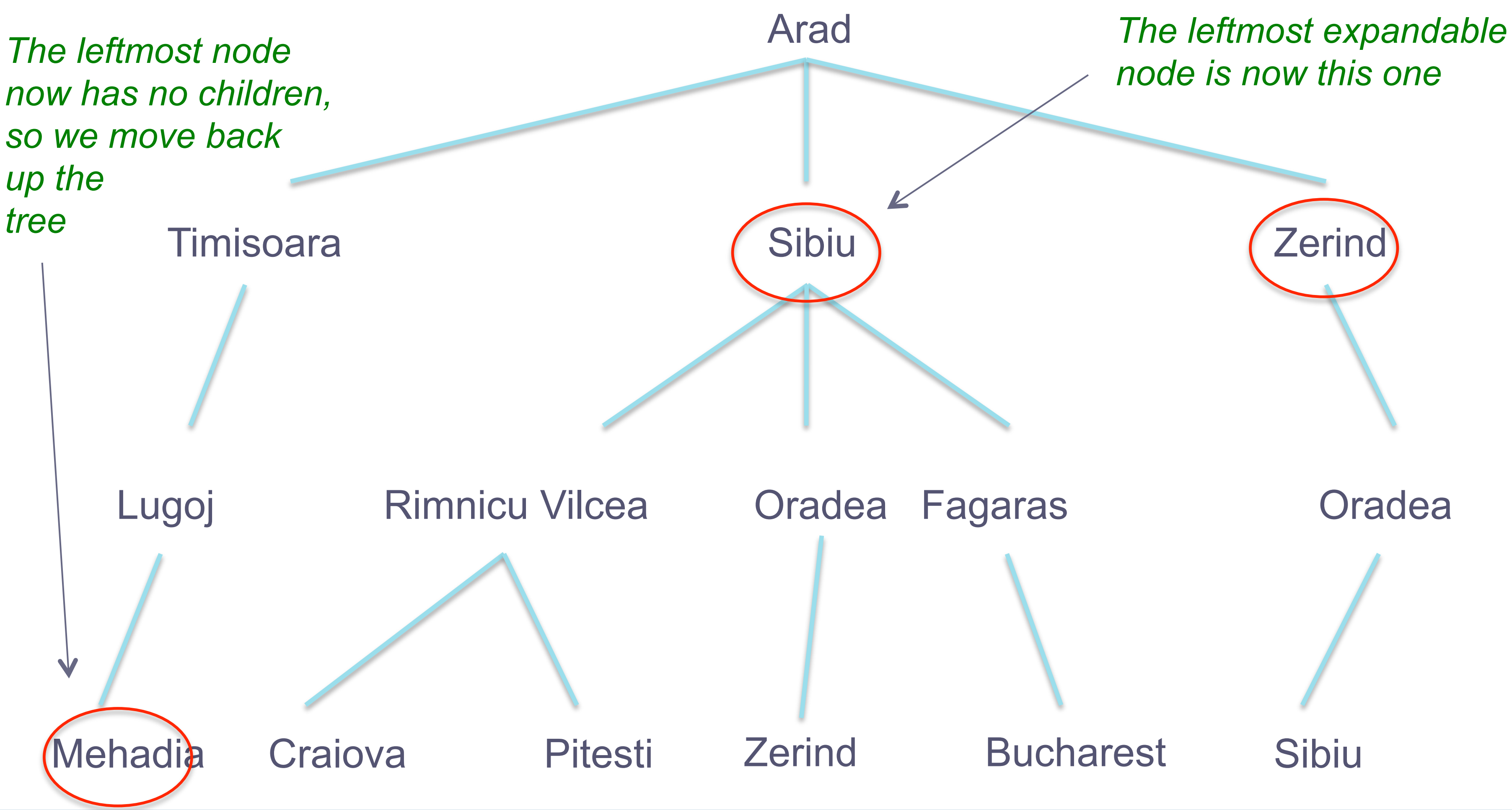


Depth-first Search 3

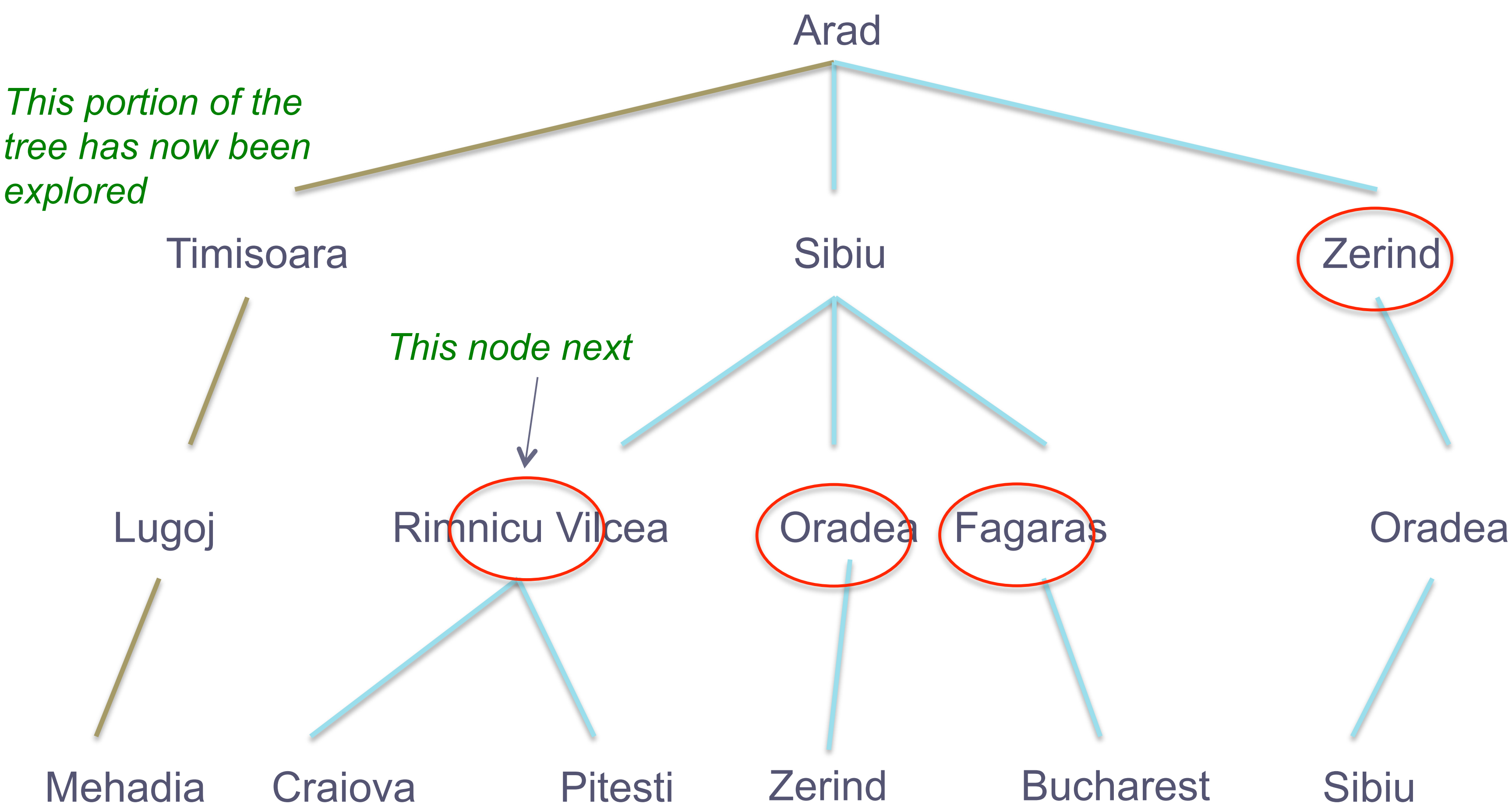
Again we select the leftmost unexplored node for expansion



Depth-first Search 4

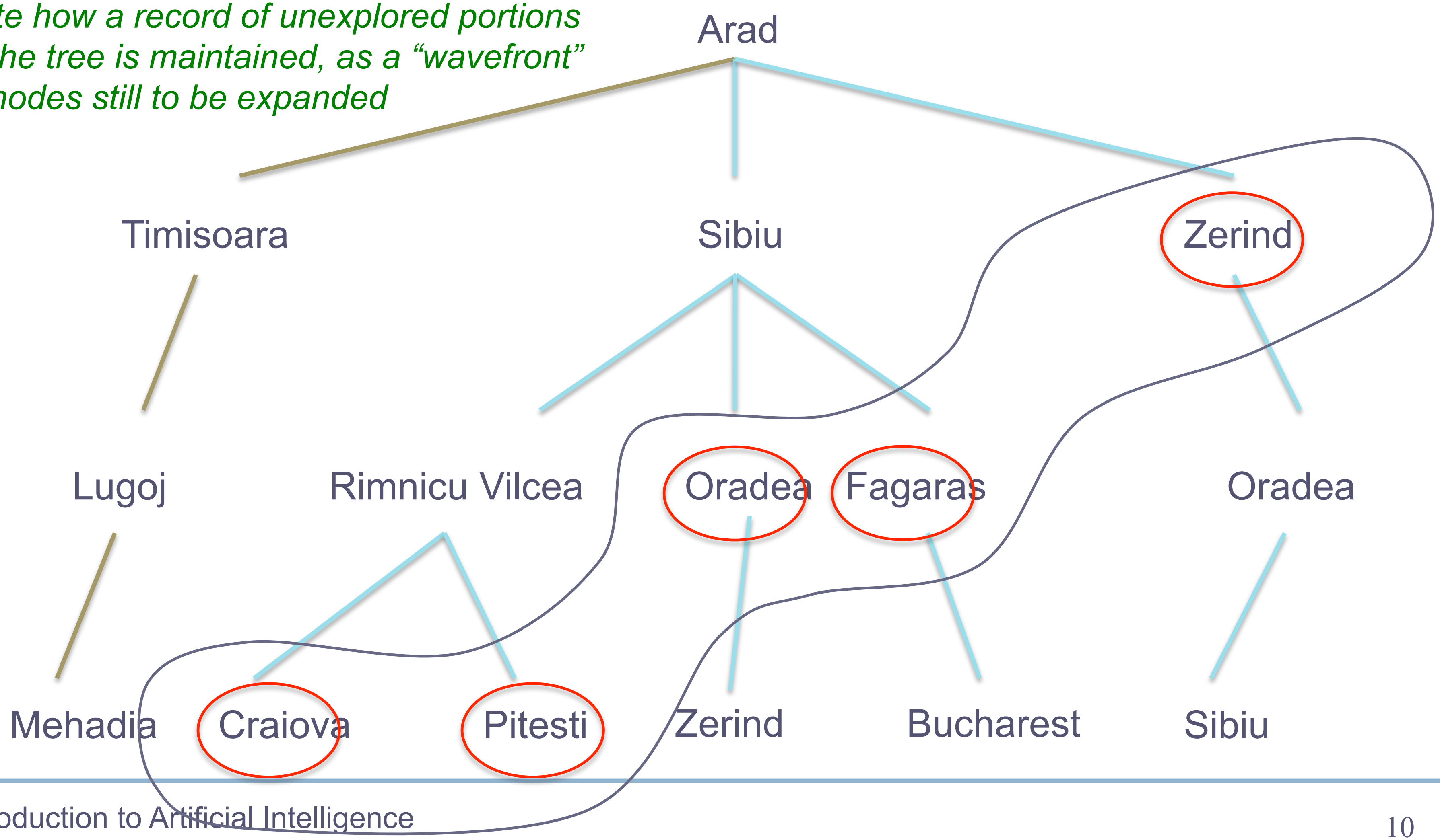


Depth-first Search 5

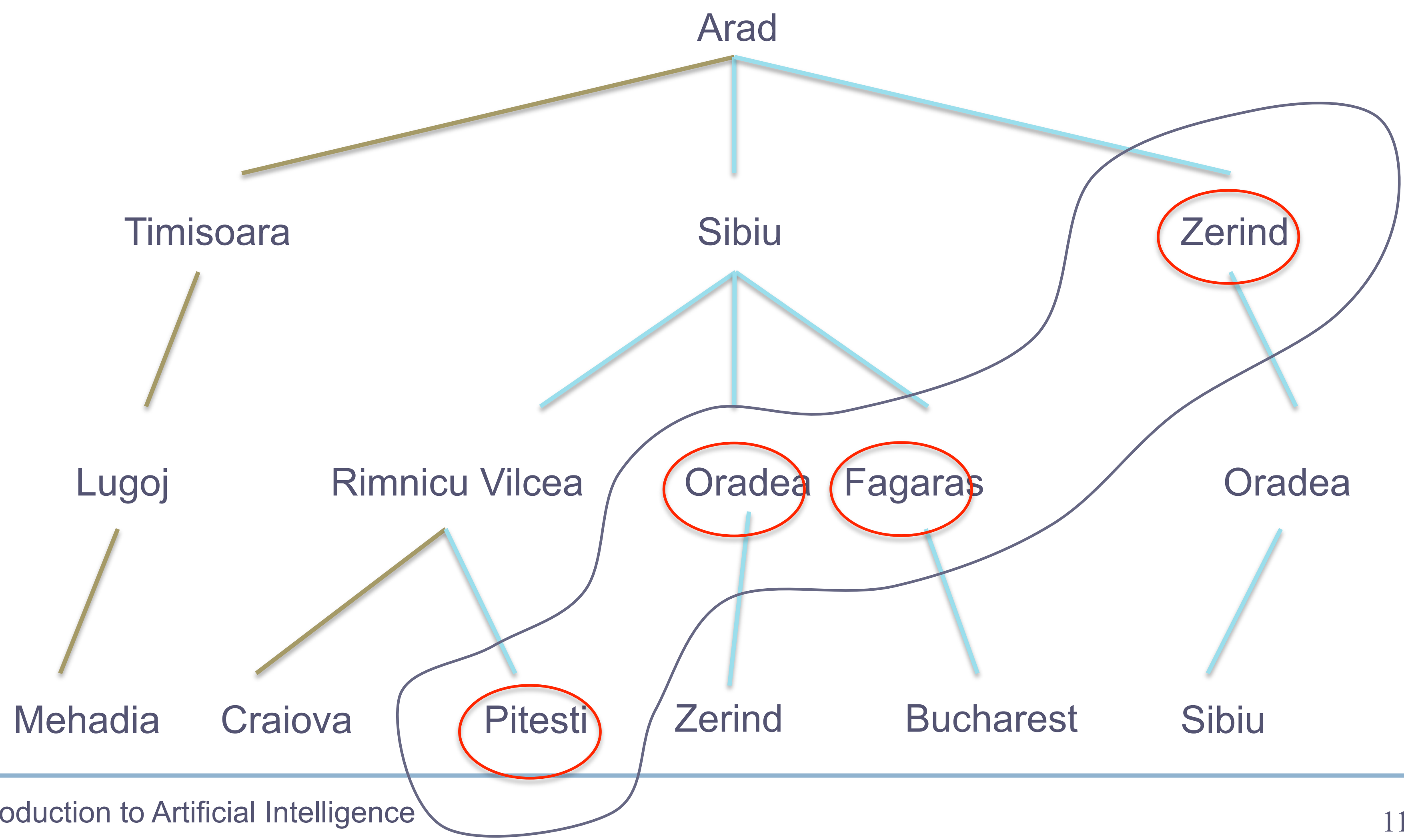


Depth-first Search 6

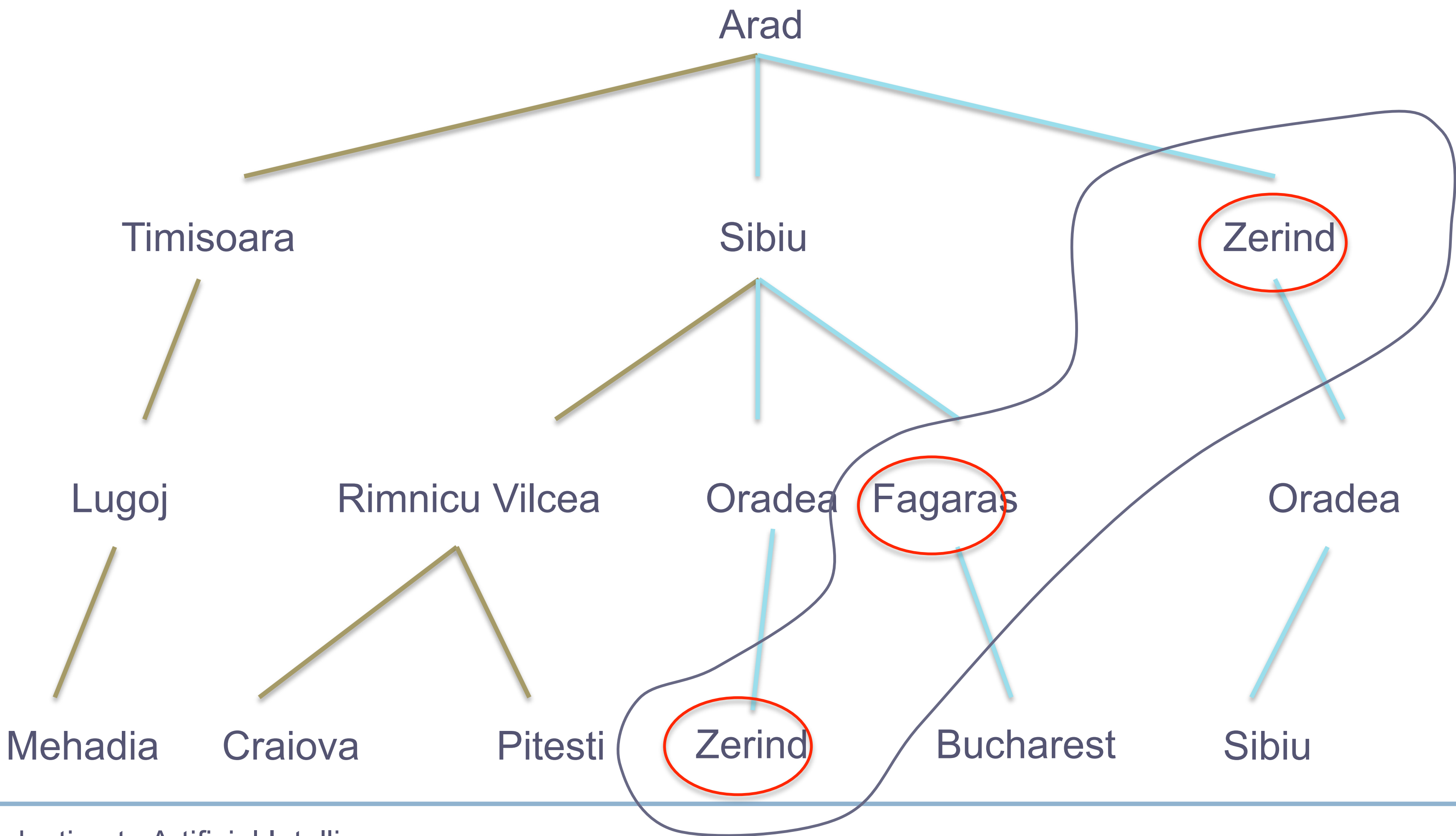
Note how a record of unexplored portions of the tree is maintained, as a "wavefront" of nodes still to be expanded



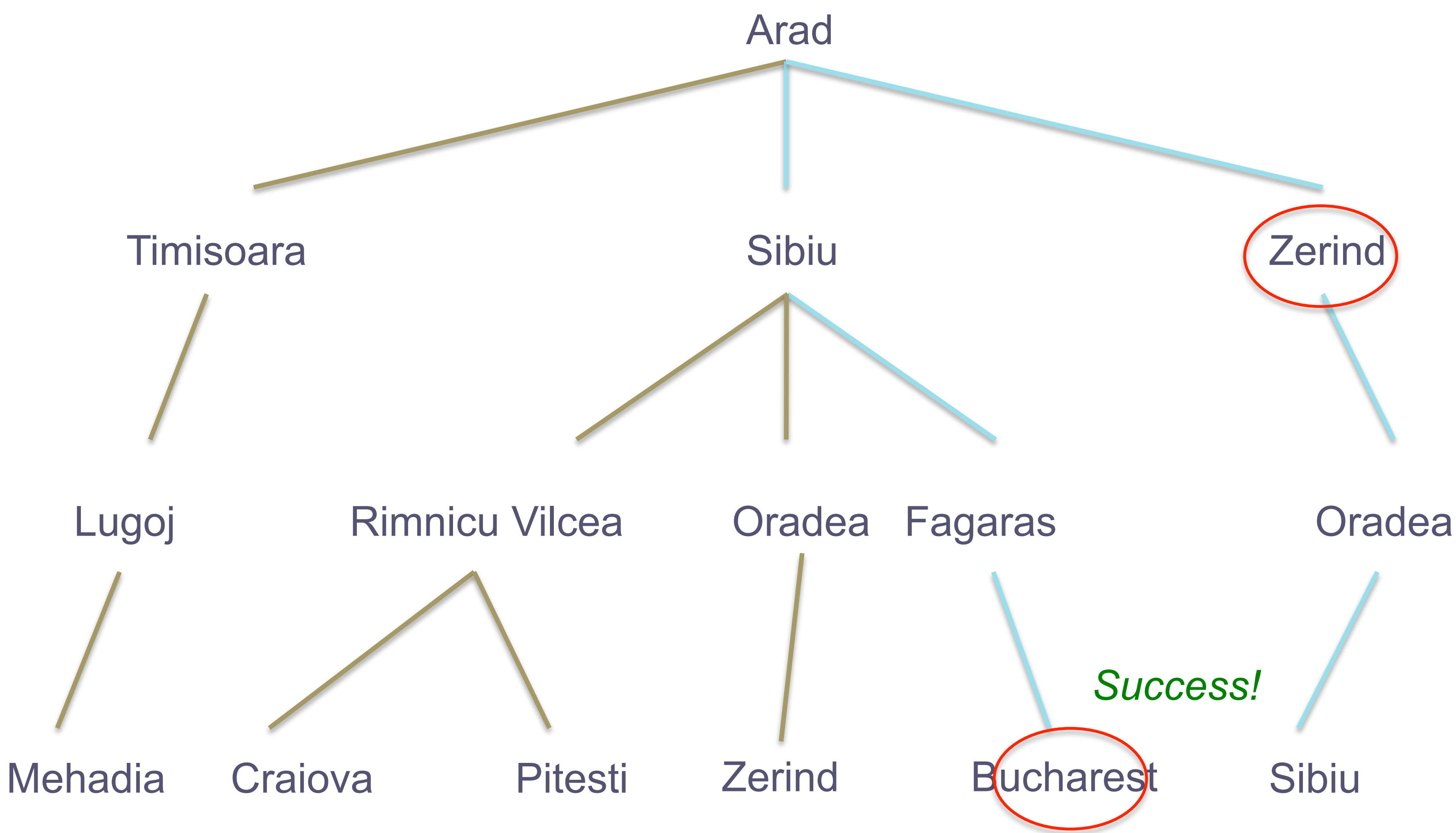
Depth-first Search 7



Depth-first Search 8



Depth-first Search 8



Properties of Depth-first

- Not guaranteed to find a solution (not complete), because it can get lost in infinite branches of the tree
- Not guaranteed to find the shortest path to a solution
- Efficient use of memory

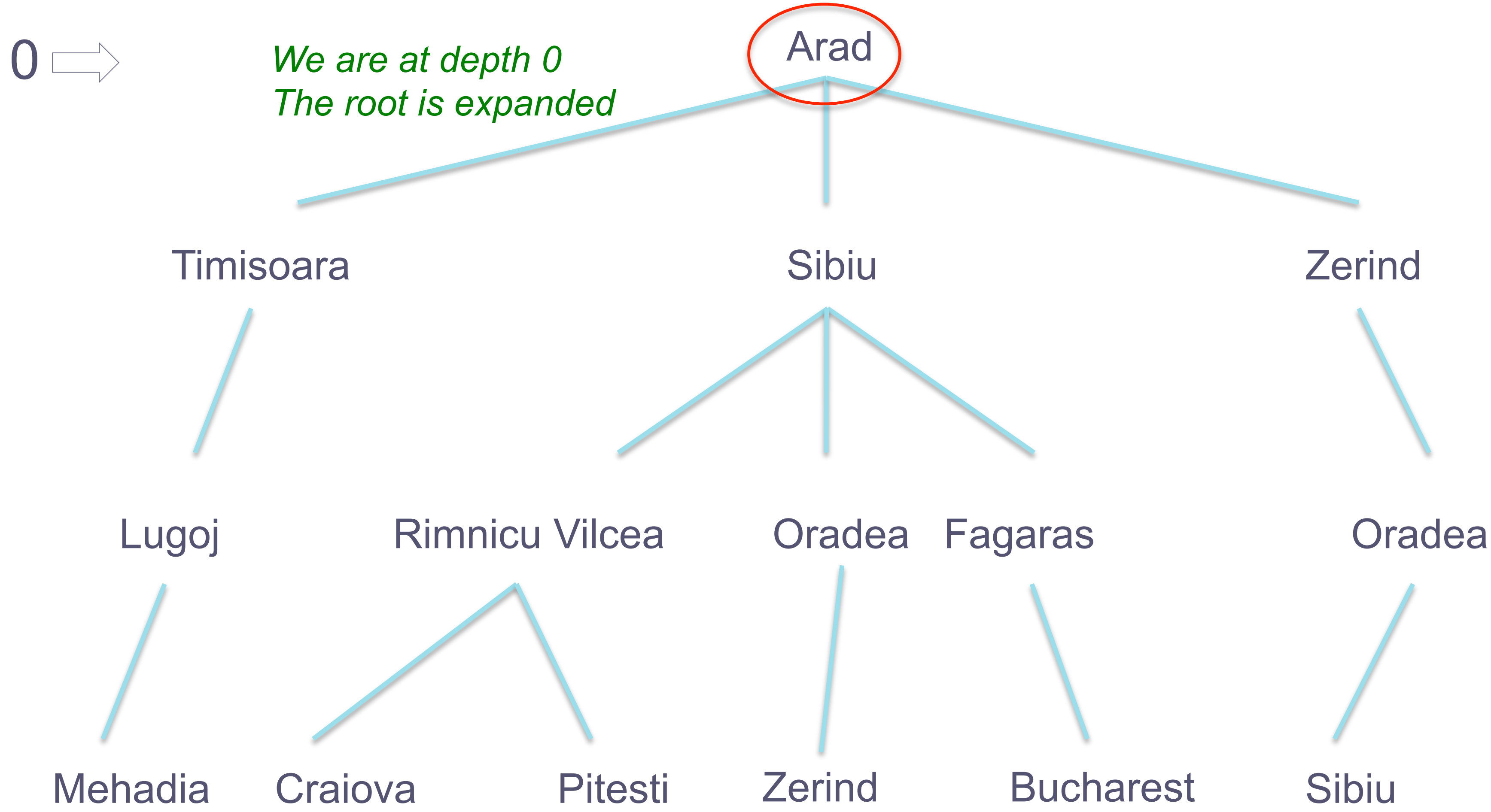
Prolog Code

```
search(Paths,X):-
        choose([Node|Path],Paths,_),
        goal(Node),
        reverse([Node|Path],X).

search(Paths,Path):-
        choose(P,Paths,RestofPaths),
        findall([S|P],S expands P,Exps),
        combine(Exps,RestofPaths,NewPaths),
        search(NewPaths,Path).

NewState expands [State|_]:-
        arc(State,NewState).
```

- Call `search([[s0]],X)` where `s0` is the initial state
- `Paths` is a list of lists of nodes
- Each list of nodes in `Paths` represents a partial branch of the tree
- The head of each list of nodes in `Paths` is the next node in that branch to be expanded

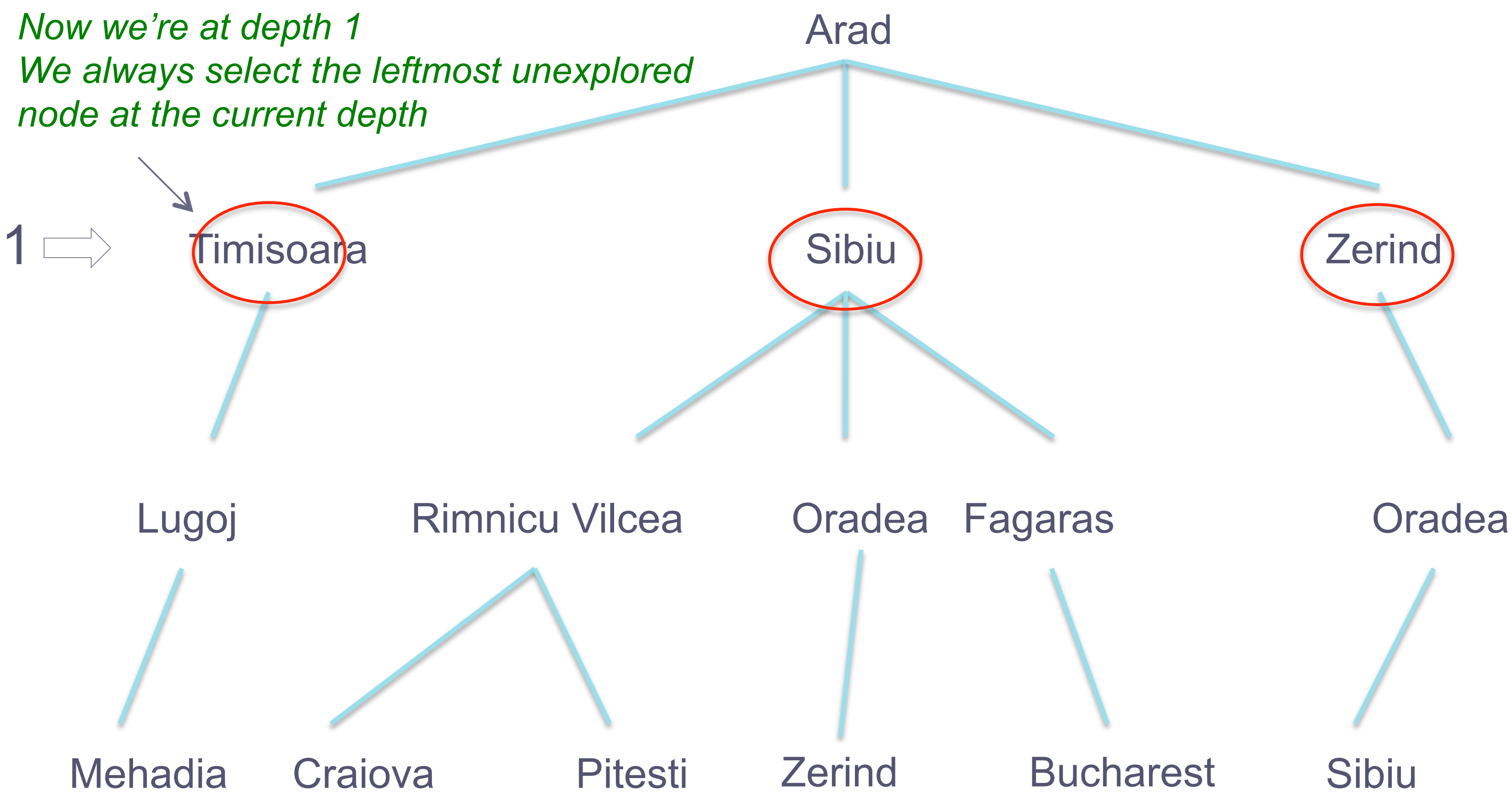
Breadth-first Search

Breadth-first Search 1



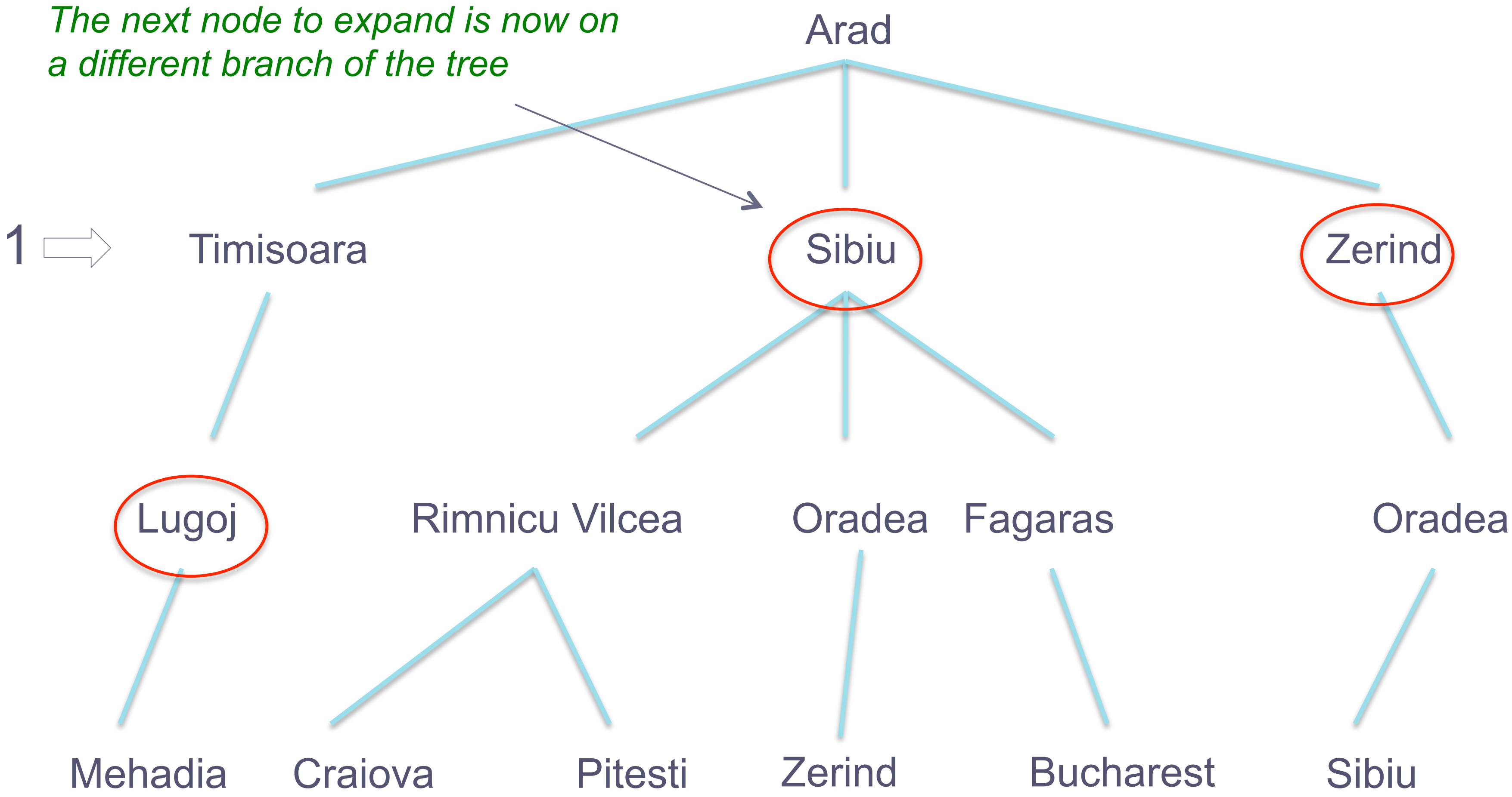
Breadth-first Search 2

Now we're at depth 1
We always select the leftmost unexplored node at the current depth

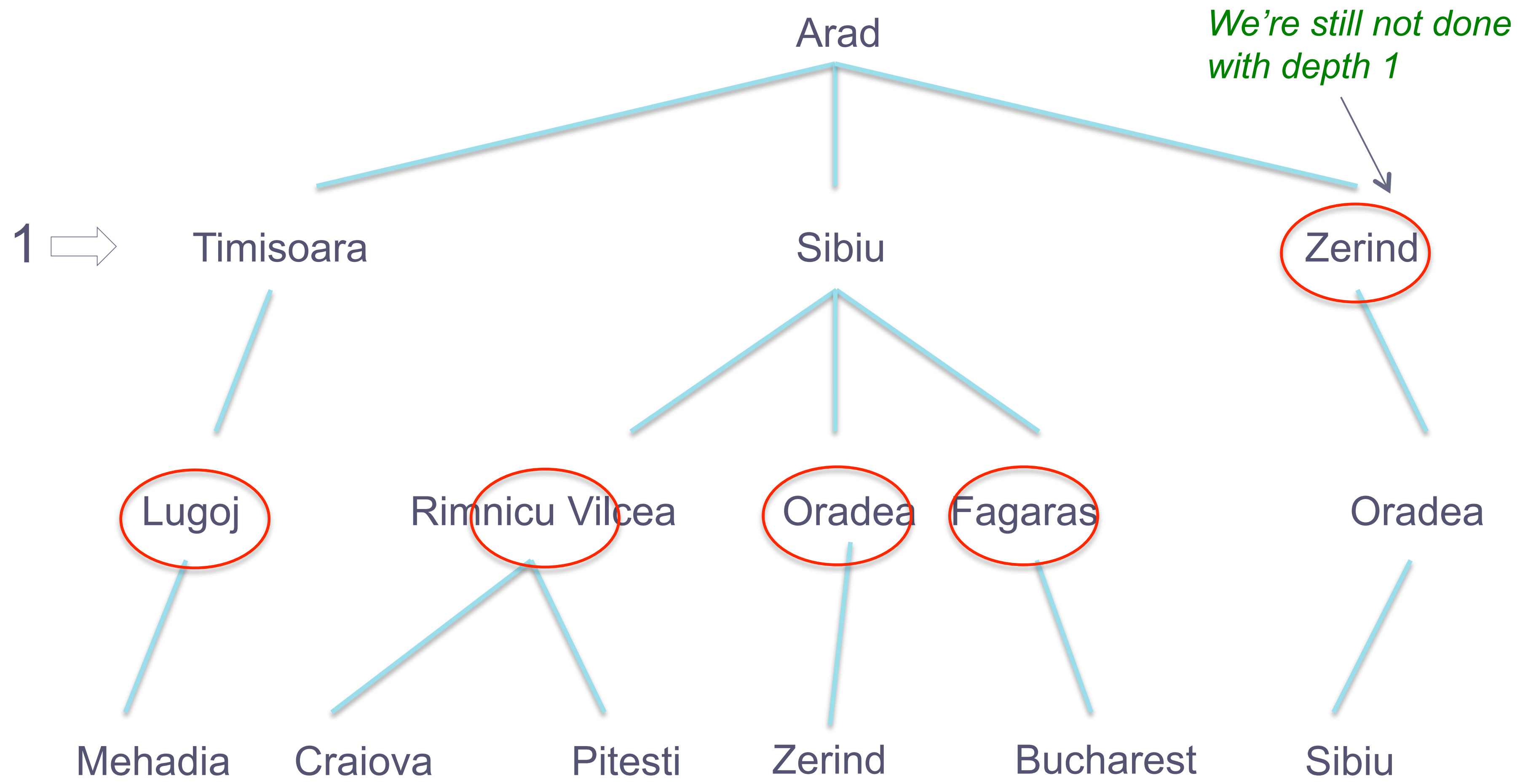


Breadth-first Search 3

The next node to expand is now on a different branch of the tree

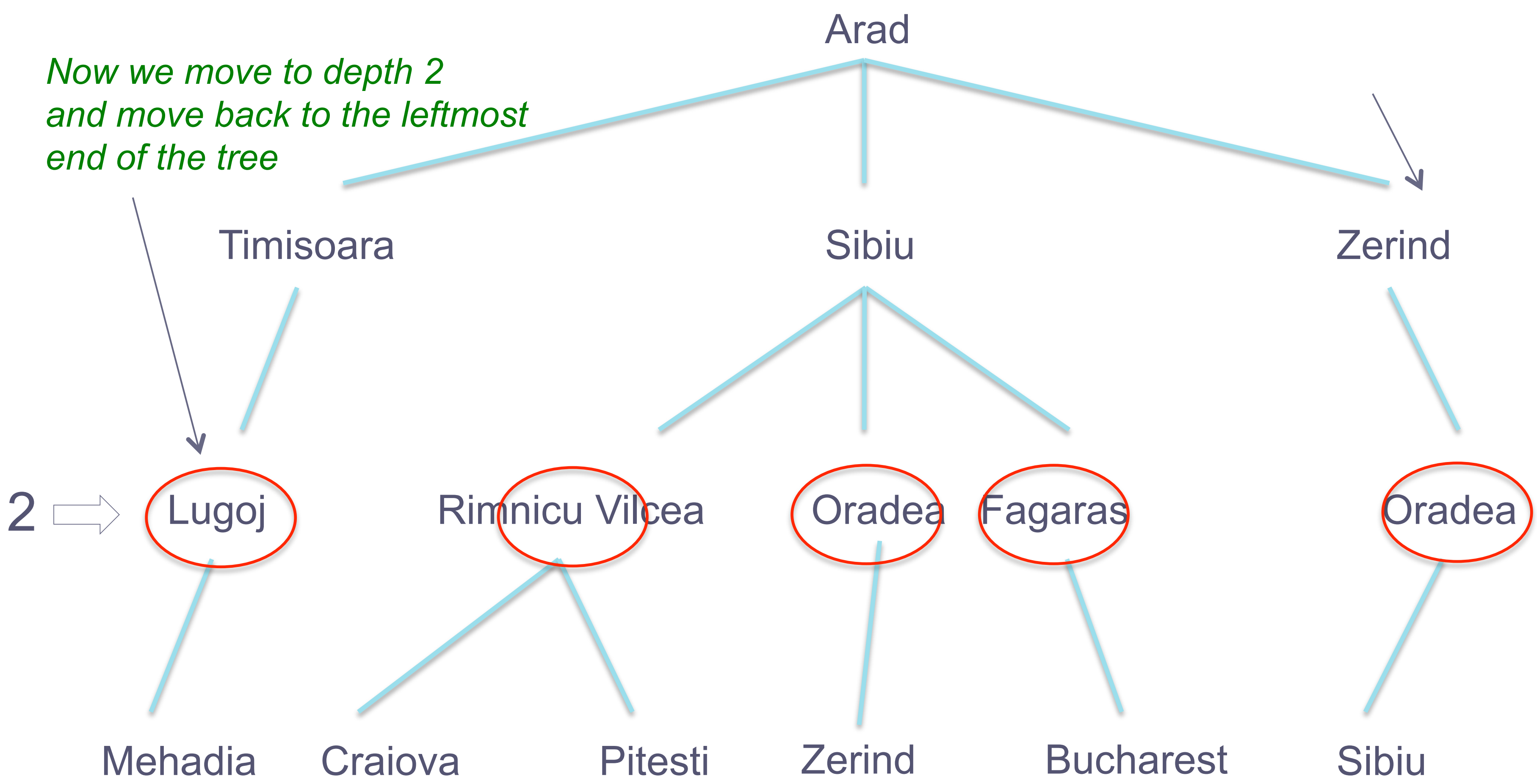


Breadth-first Search 4

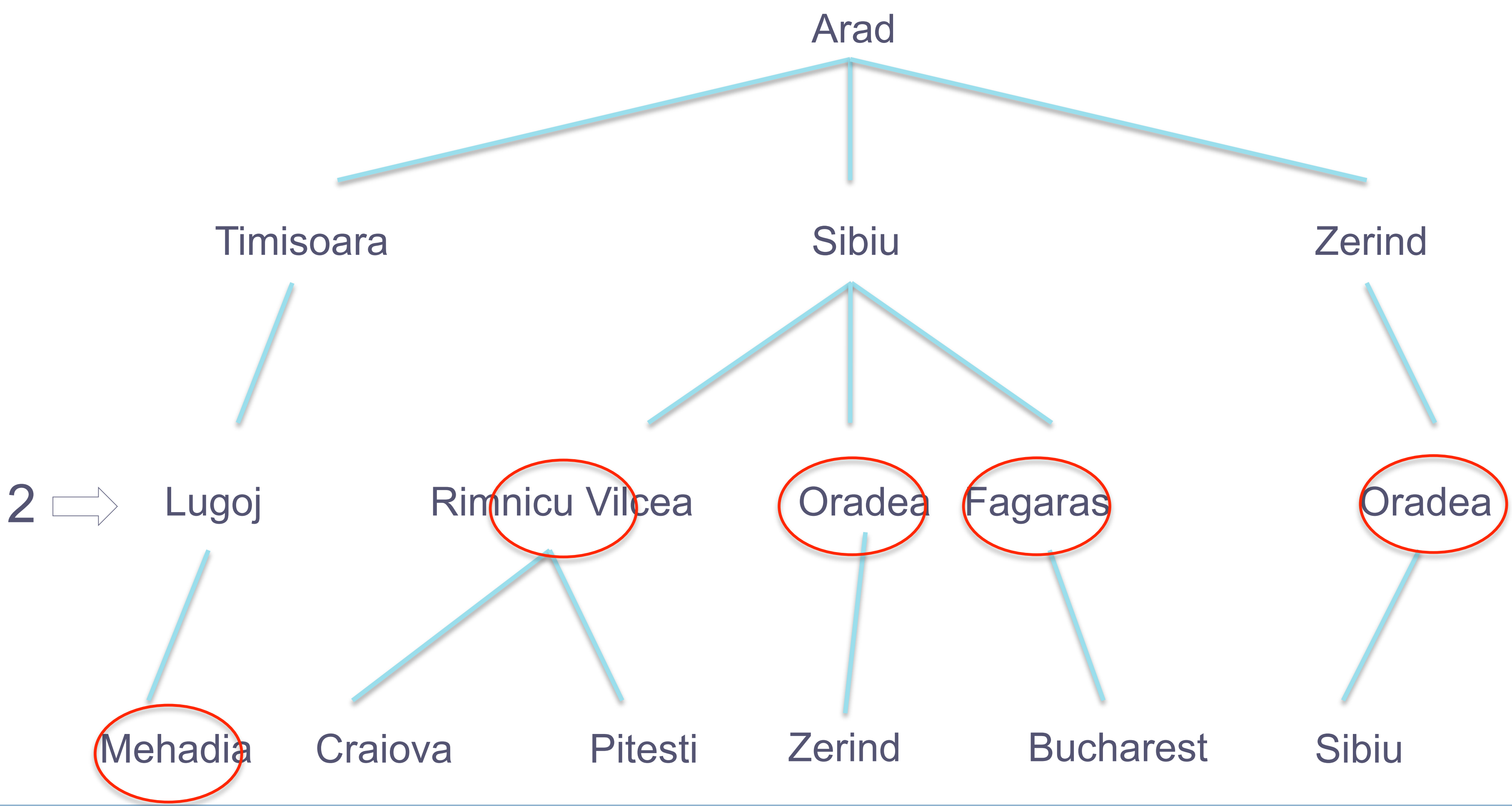


Breadth-first Search 5

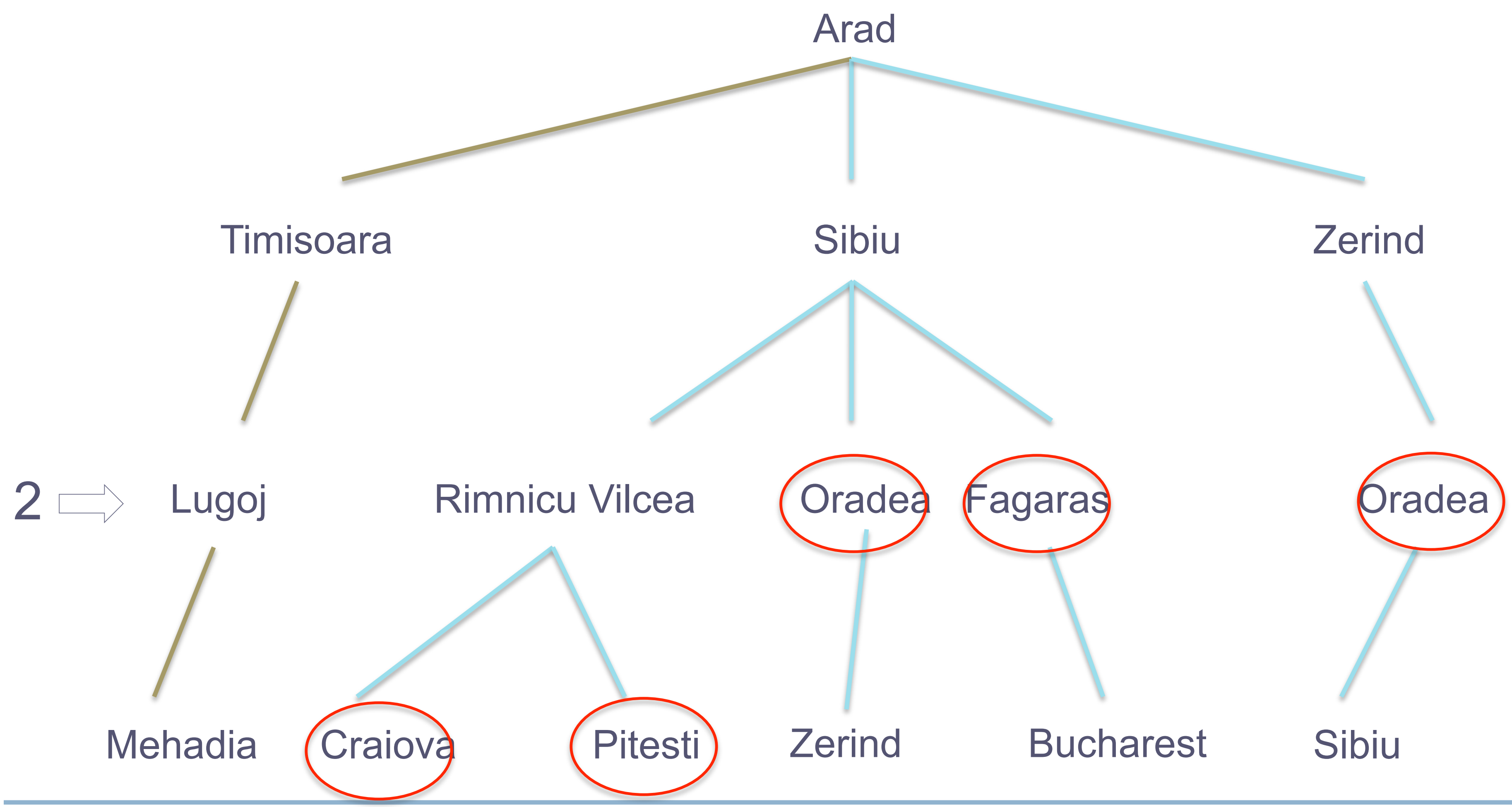
Now we move to depth 2
and move back to the leftmost
end of the tree



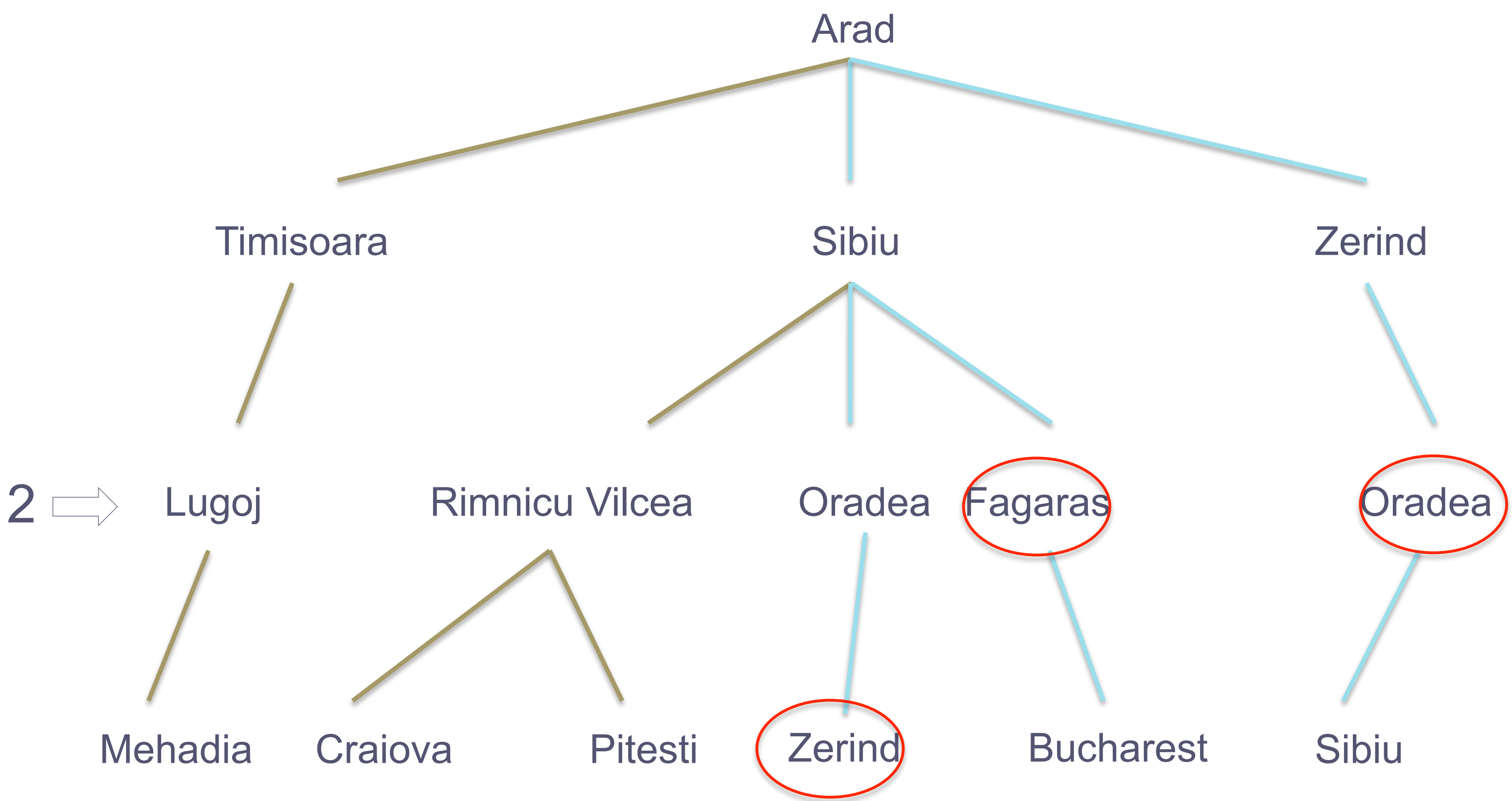
Breadth-first Search 6



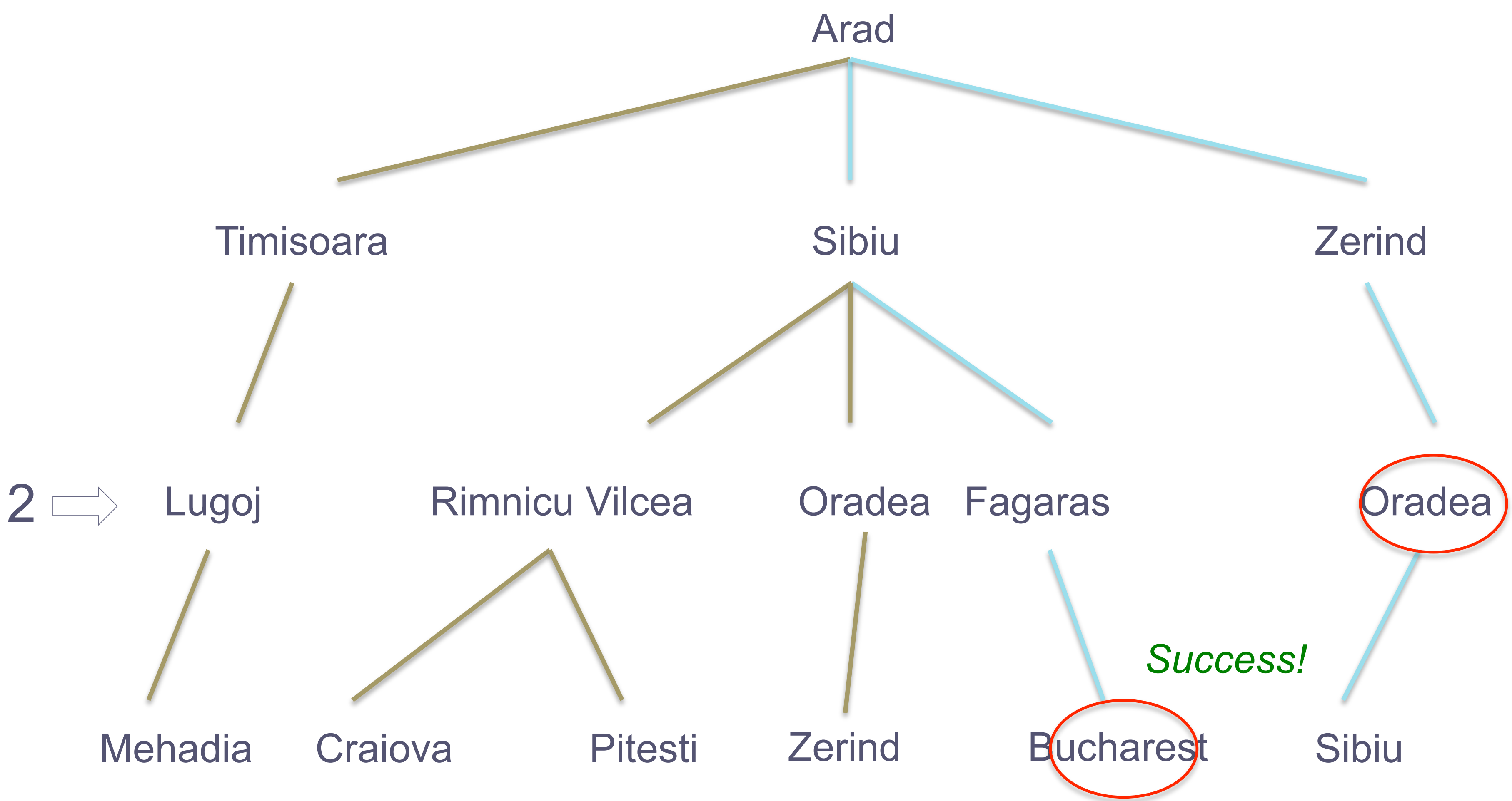
Breadth-first Search 7



Breadth-first Search 8



Breadth-first Search 9



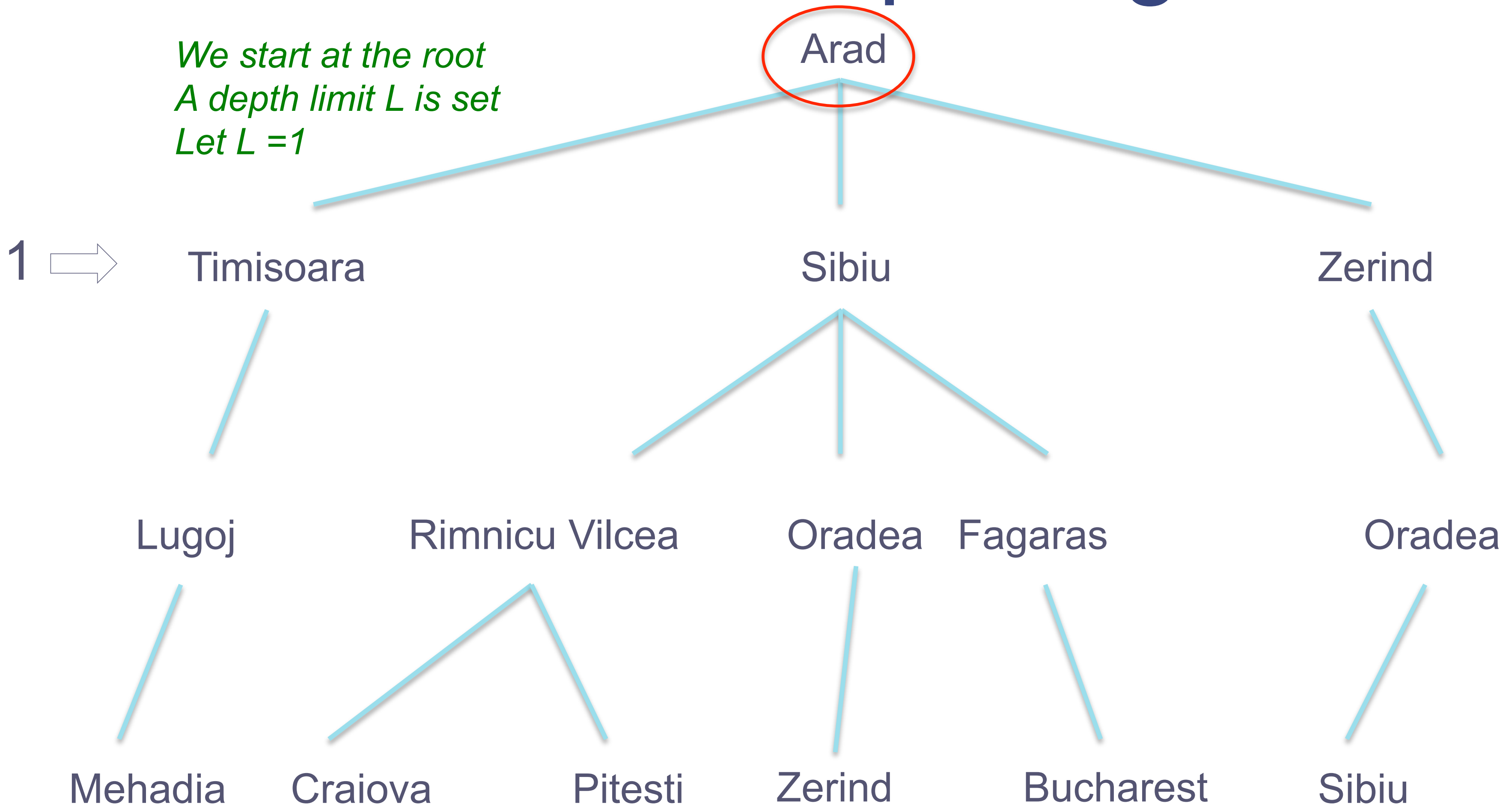
Properties of Breadth-first

- Guaranteed to find a solution if one exists, because every node in the tree is visited eventually
- Guaranteed to find the shortest path to a solution
- Very poor use of memory: exponential in mean branching factor

Iterative Deepening

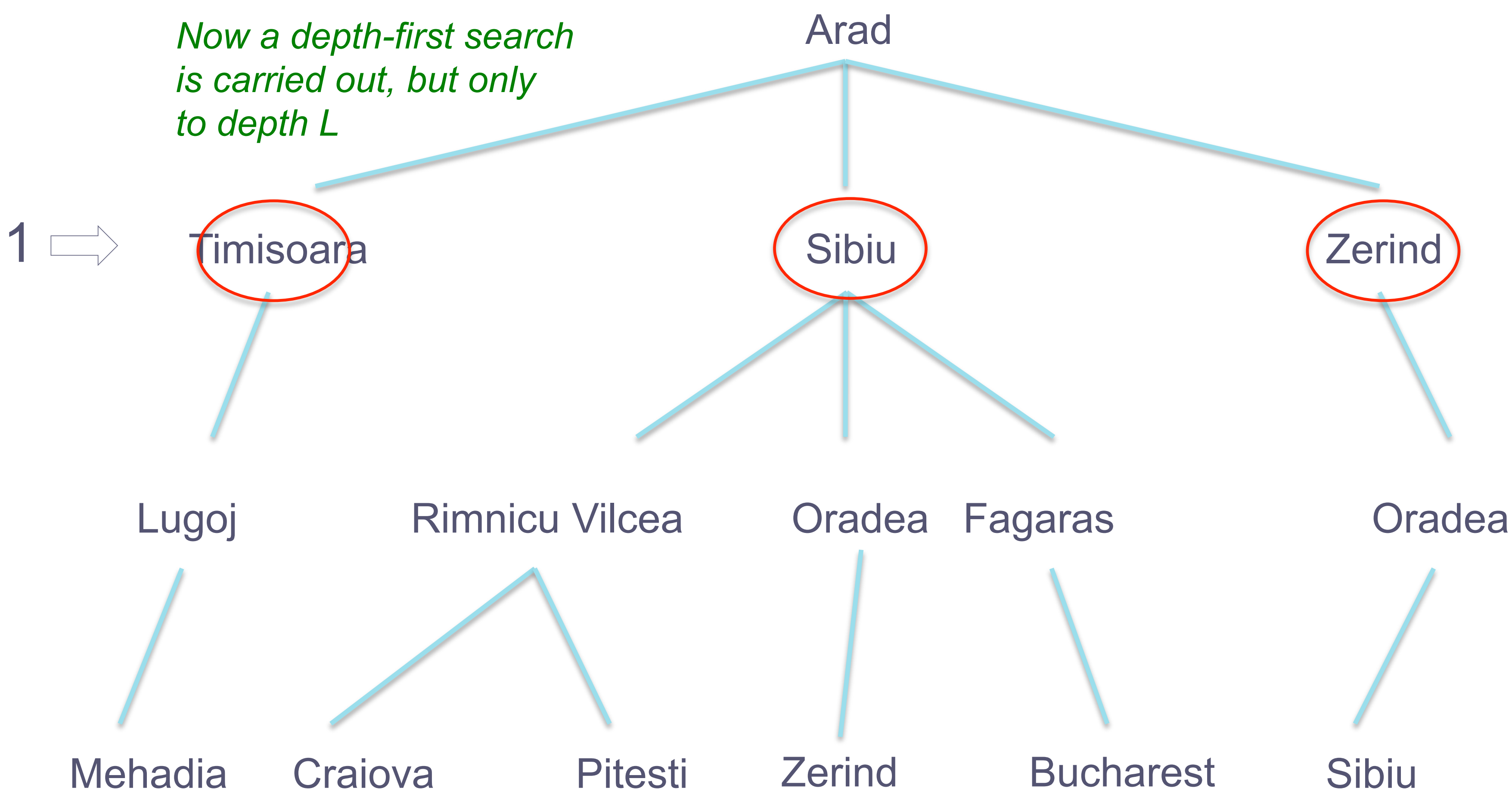
Iterative Deepening 1

We start at the root
A depth limit L is set
Let L =1

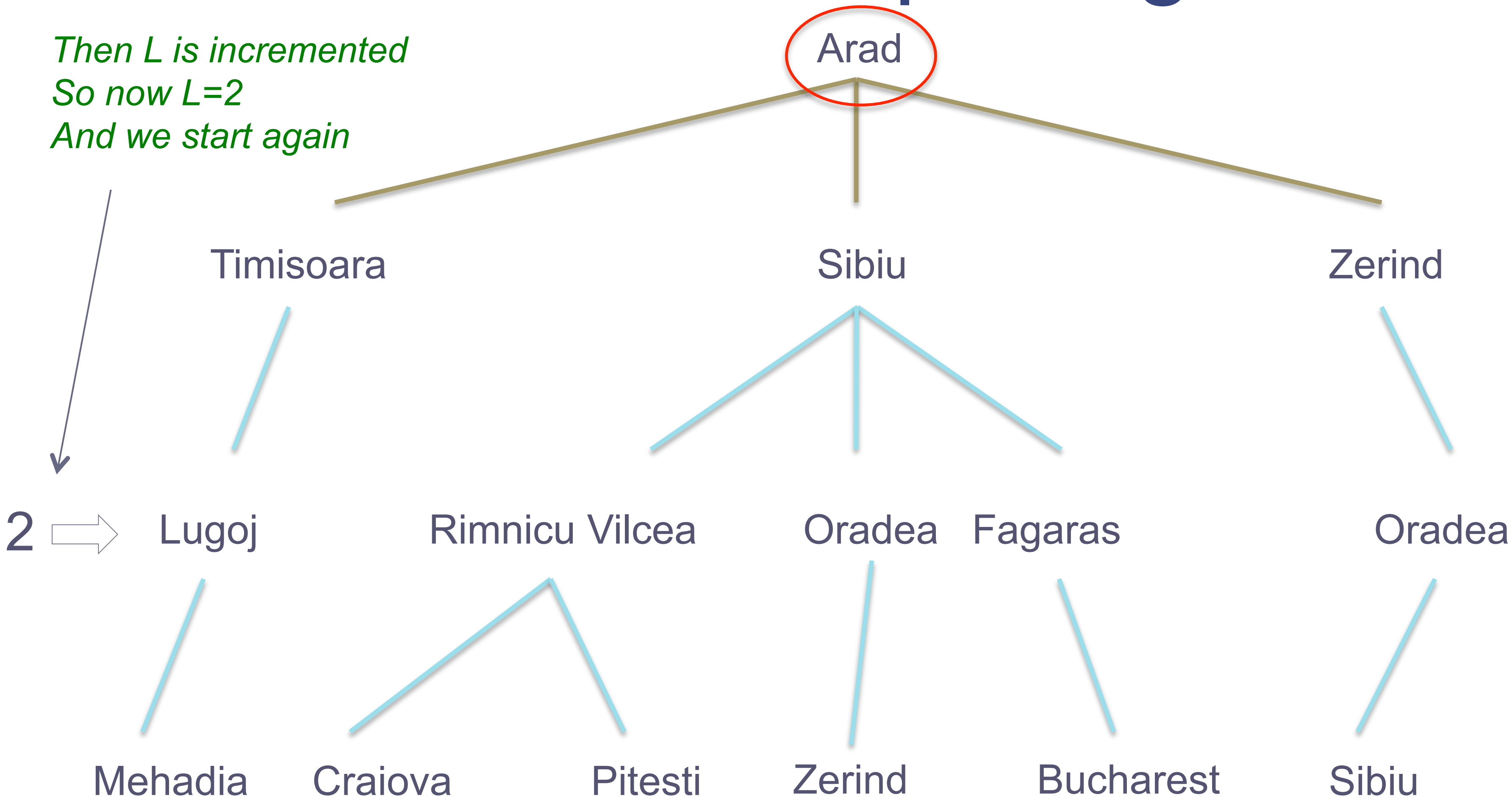


Iterative Deepening 2

Now a depth-first search is carried out, but only to depth L

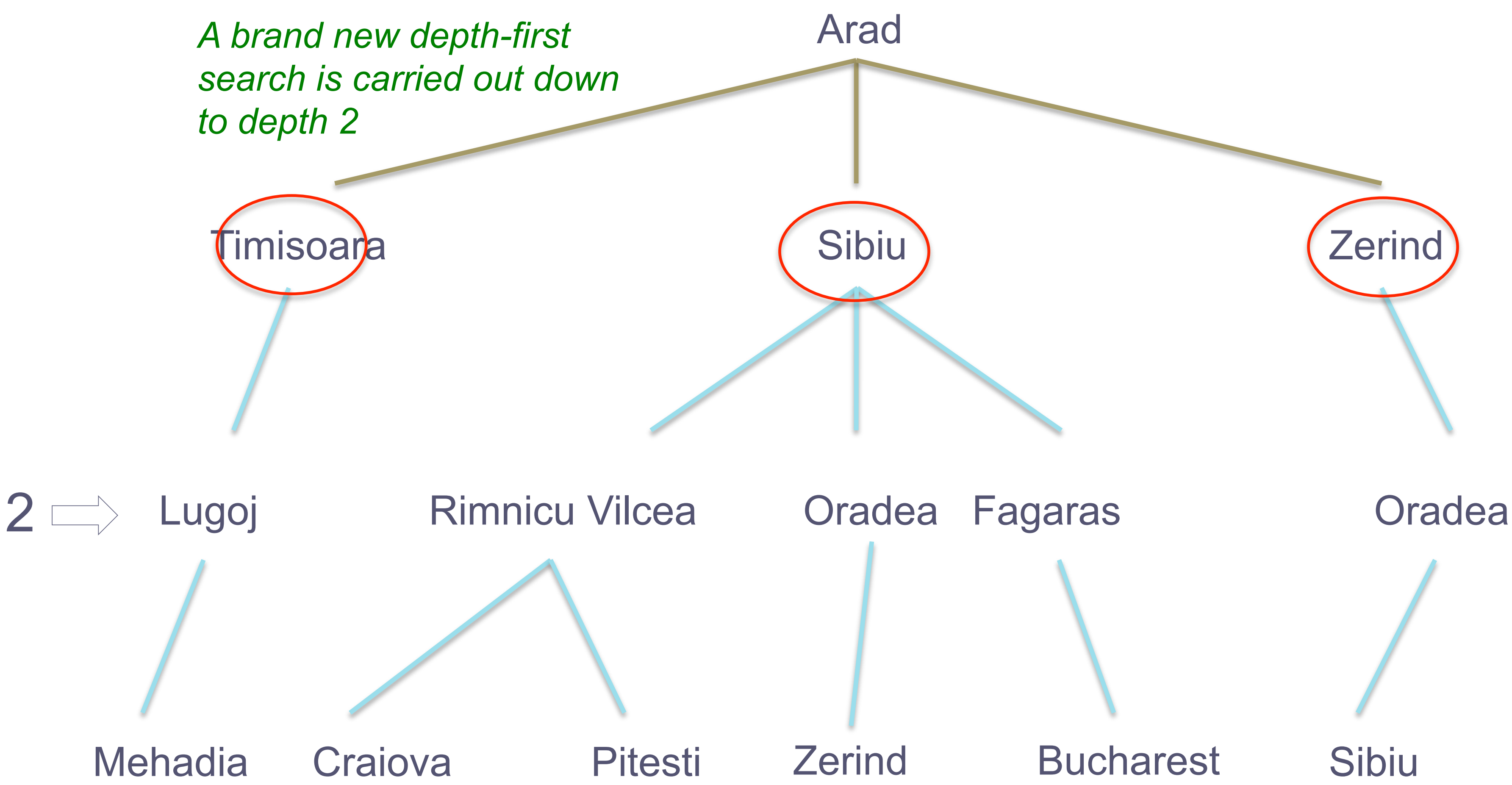


Iterative Deepening 3

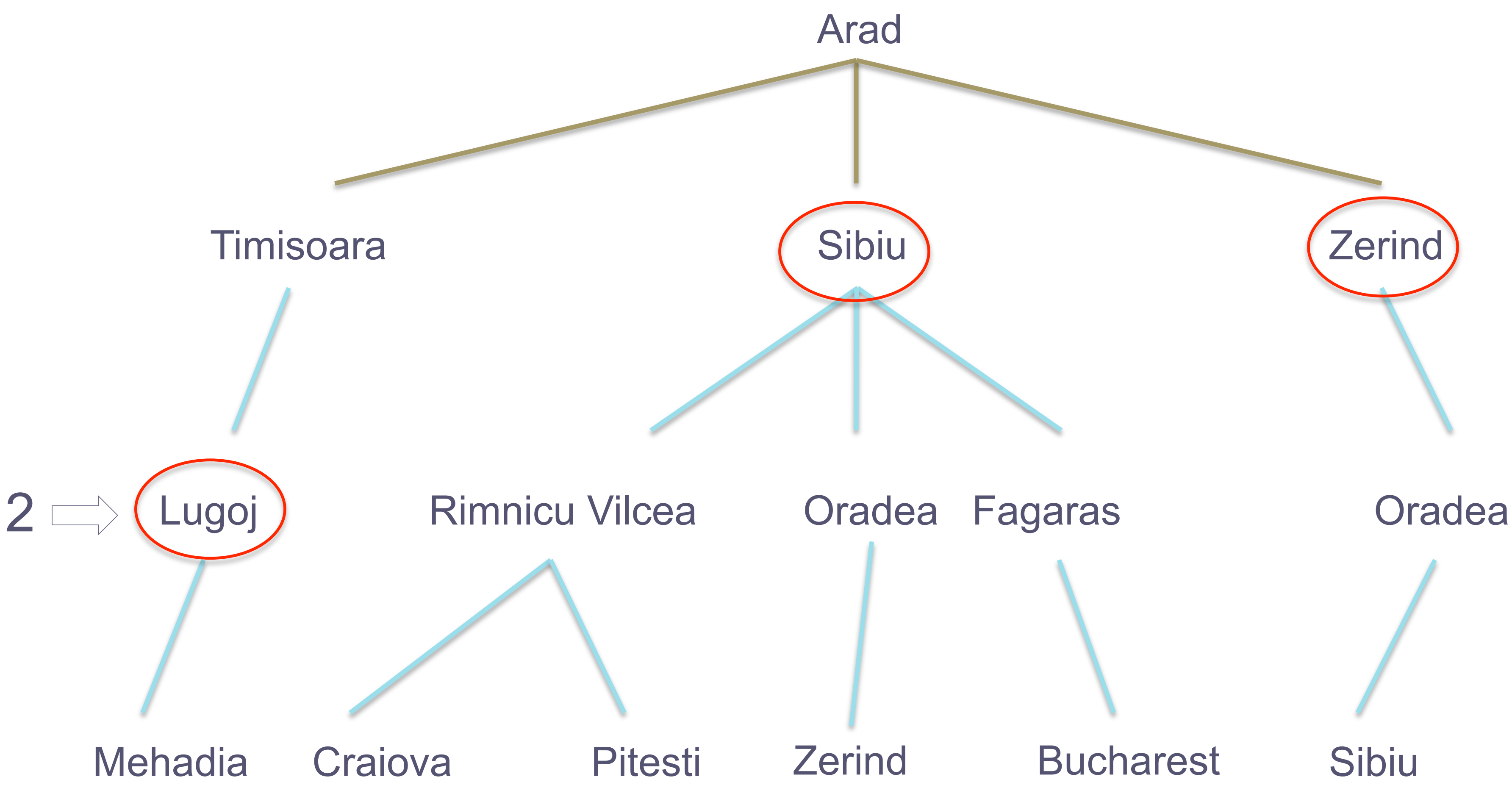


Iterative Deepening 4

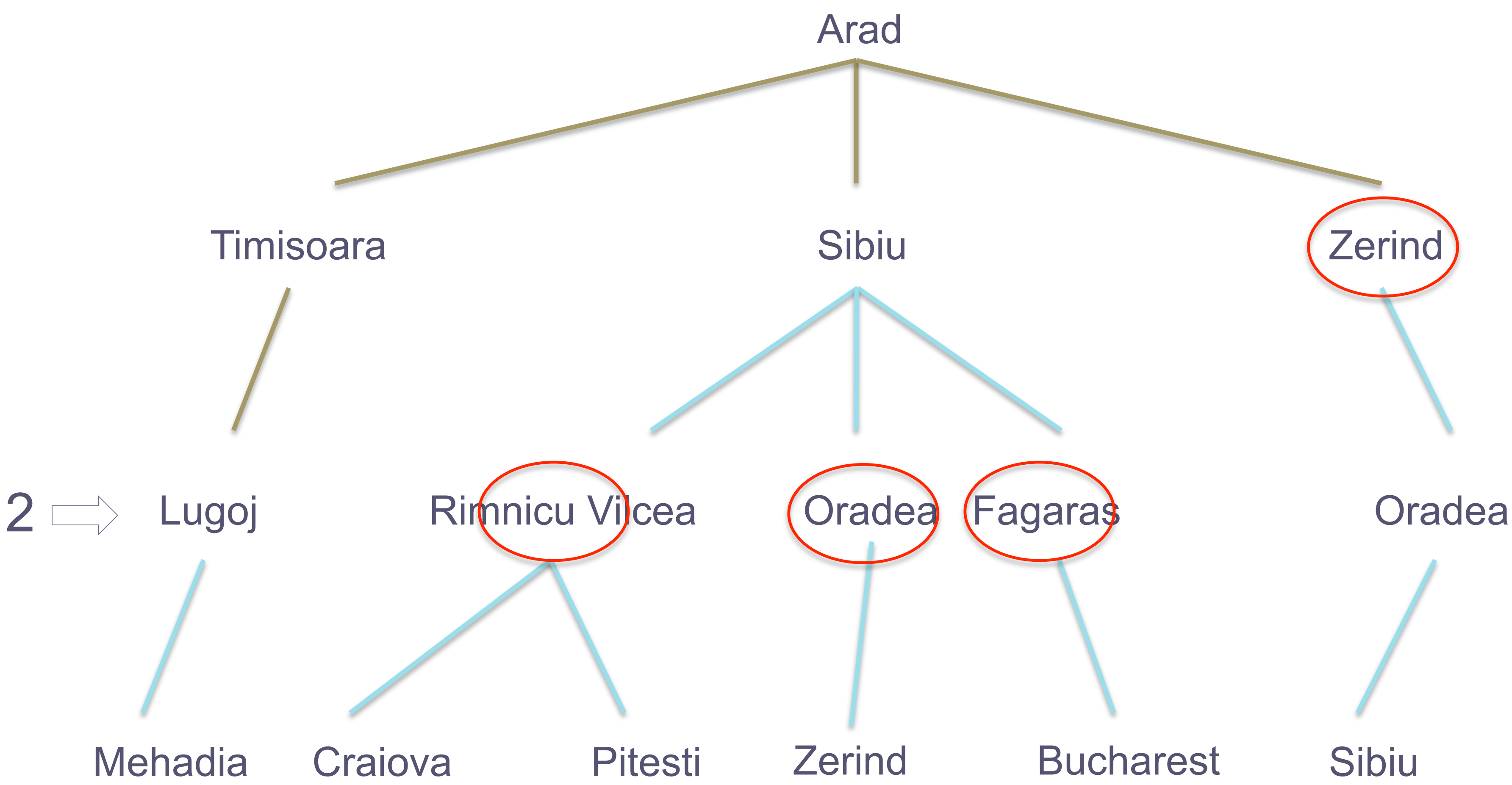
A brand new depth-first search is carried out down to depth 2



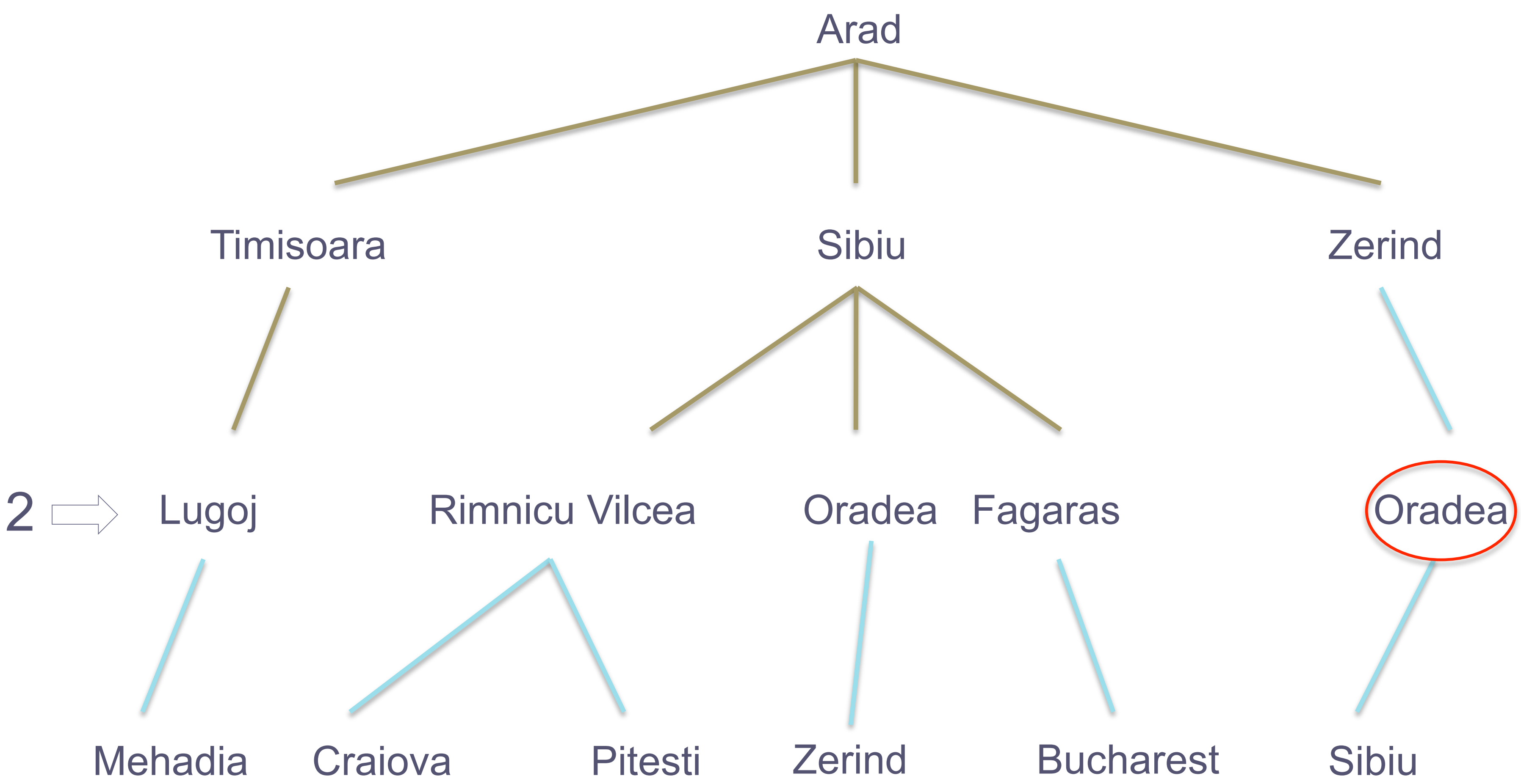
Iterative Deepening 5



Iterative Deepening 6

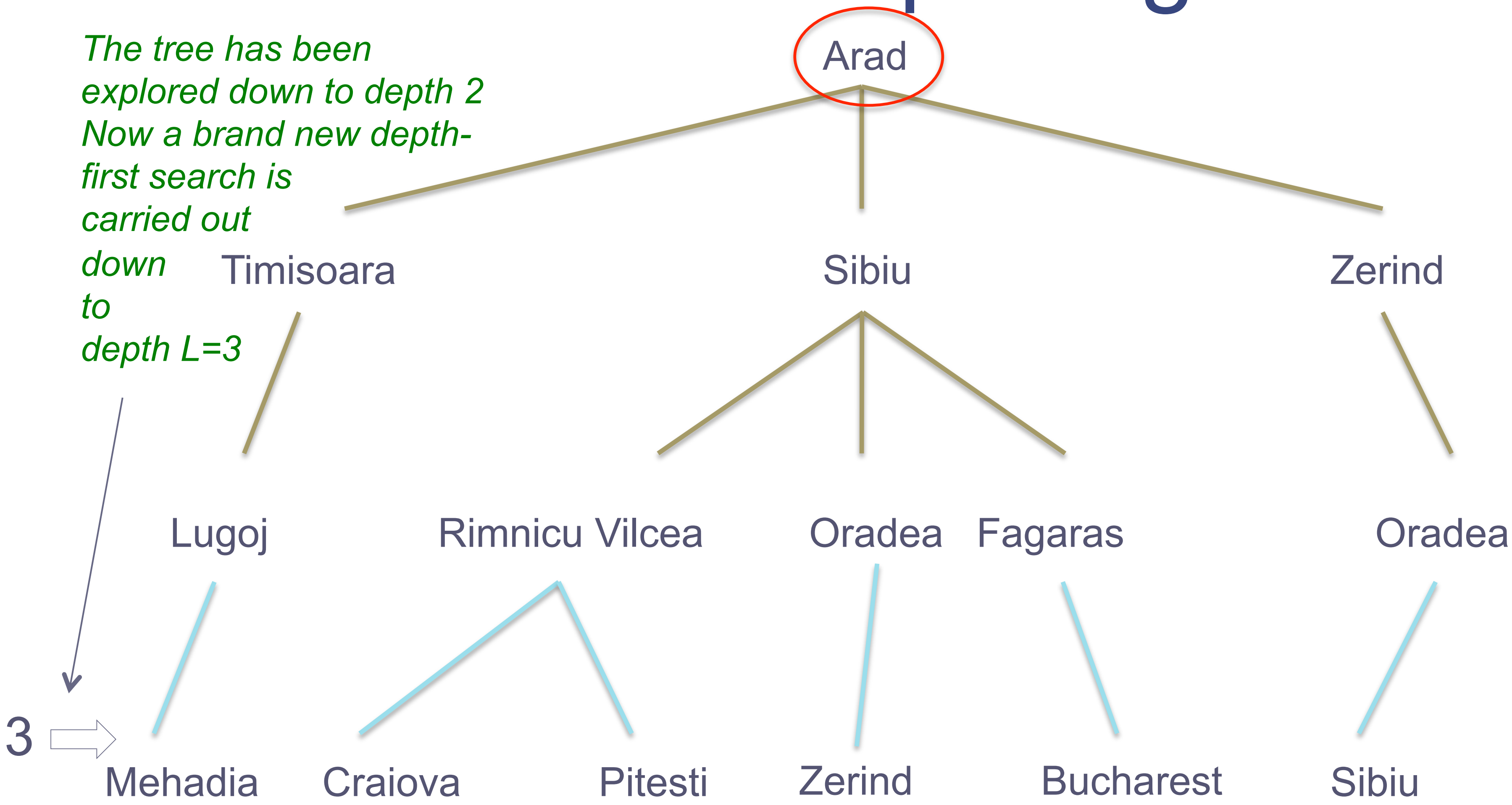


Iterative Deepening 7



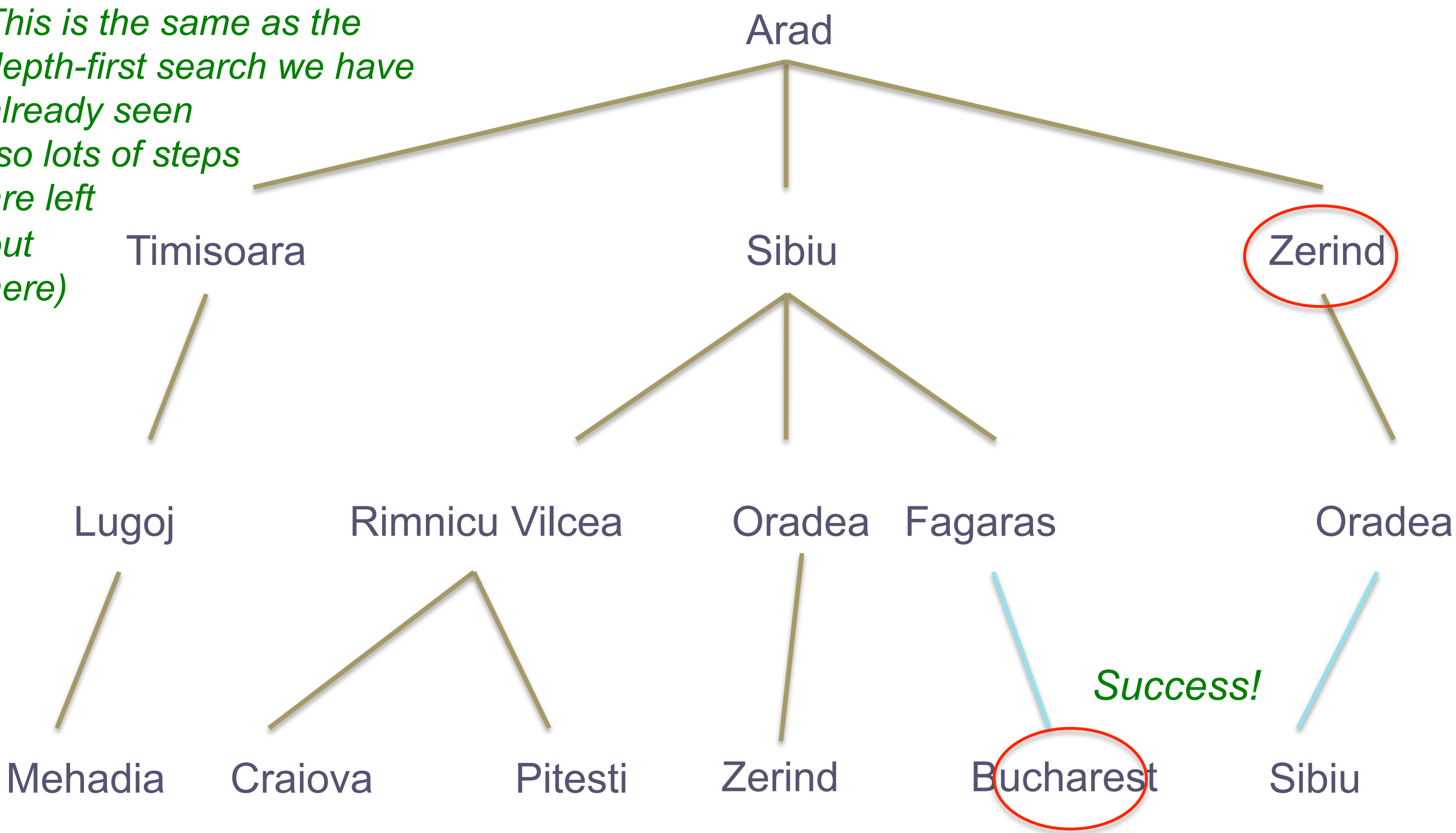
Iterative Deepening 8

The tree has been explored down to depth 2 Now a brand new depth-first search is carried out down to depth L=3



Iterative Deepening 9

This is the same as the depth-first search we have already seen (so lots of steps are left out here)

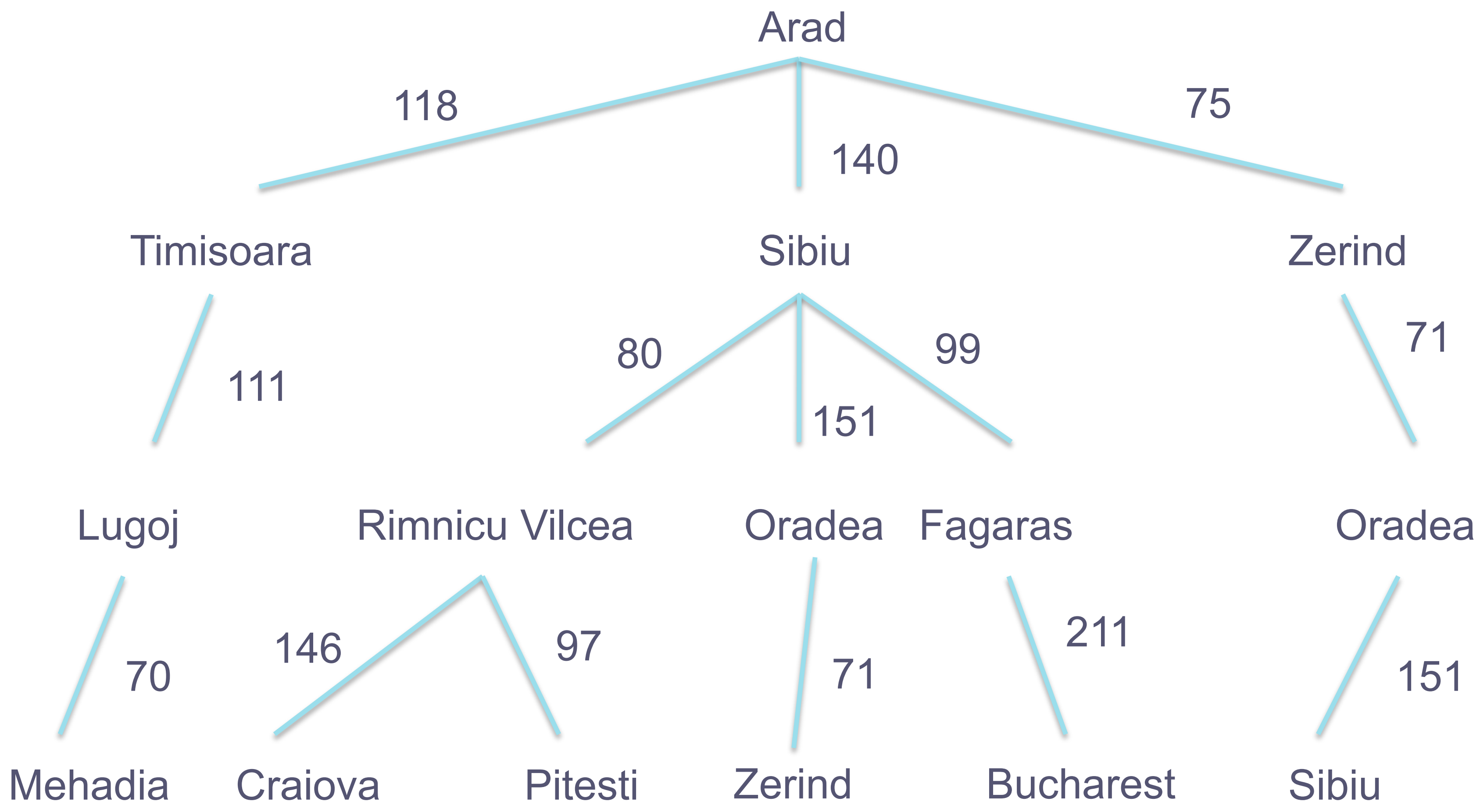


Properties of Iterative Deepening

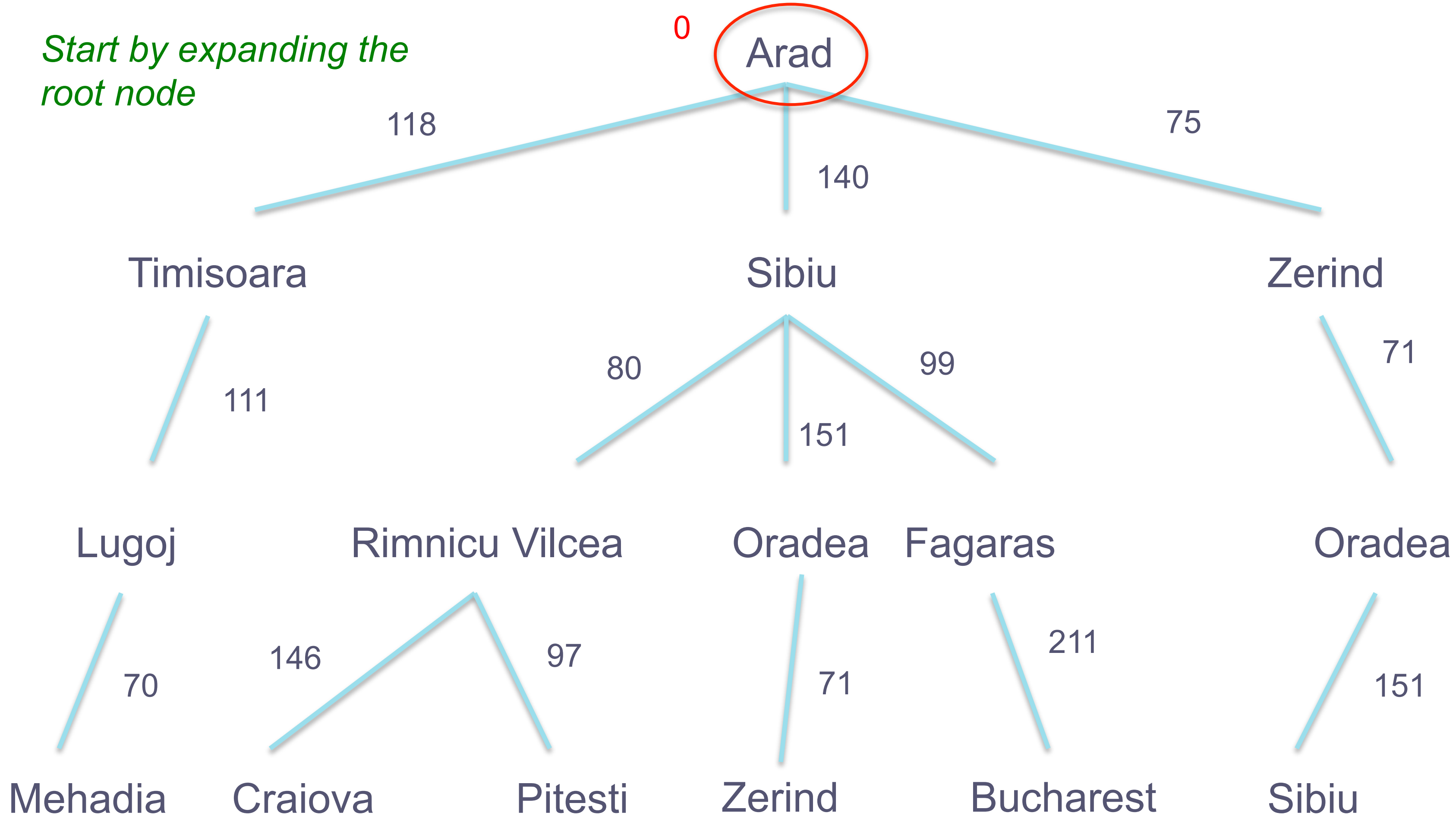
- Combines completeness of breadth-first search with memory efficiency of depth-first search
- Guaranteed to find a solution if one exists
- Slower than both breadth-first and depth-first
- Efficient use of memory
- Guaranteed to find the shortest path to a solution

Uniform Cost Search

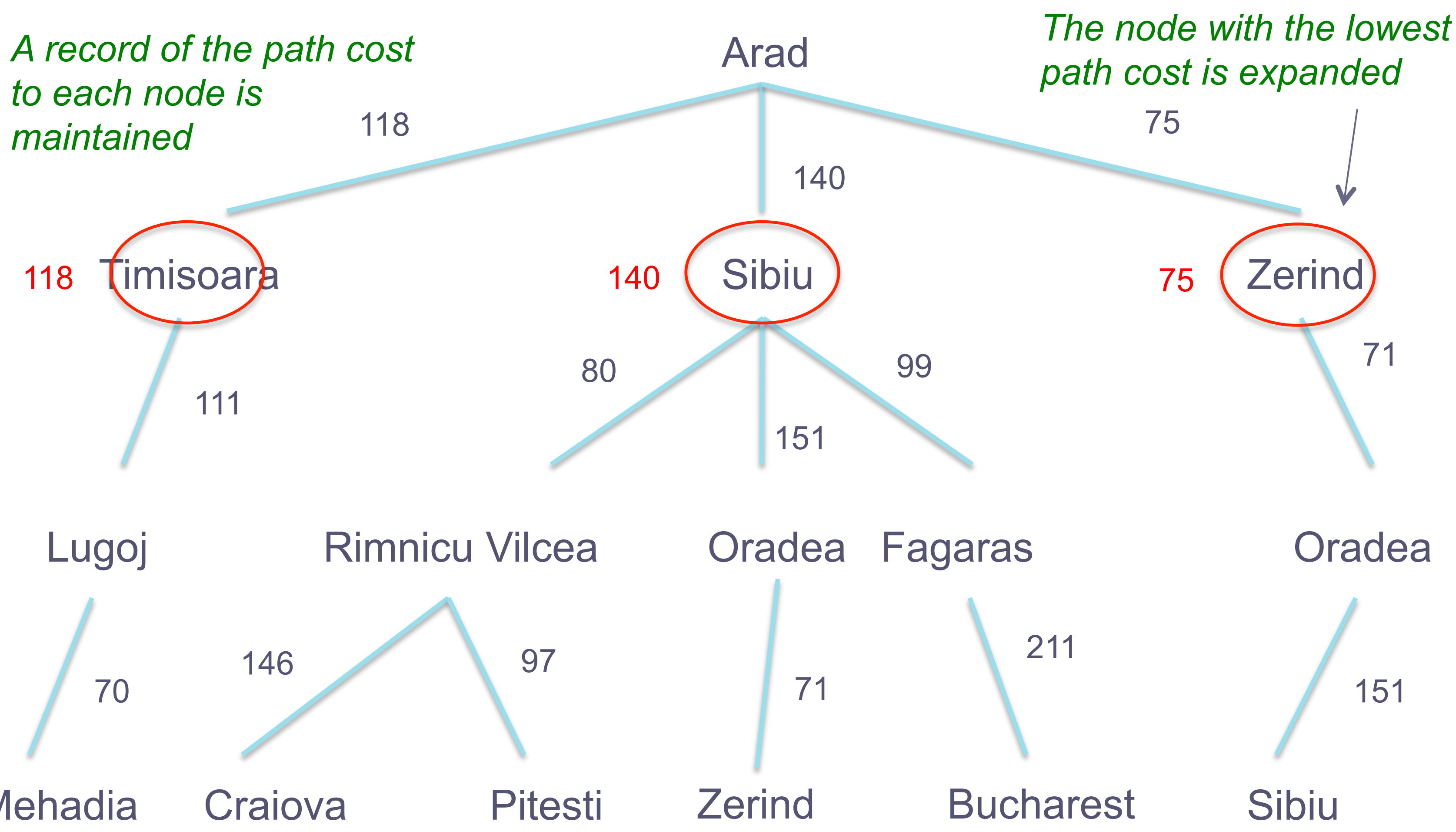
Search Tree with Costs



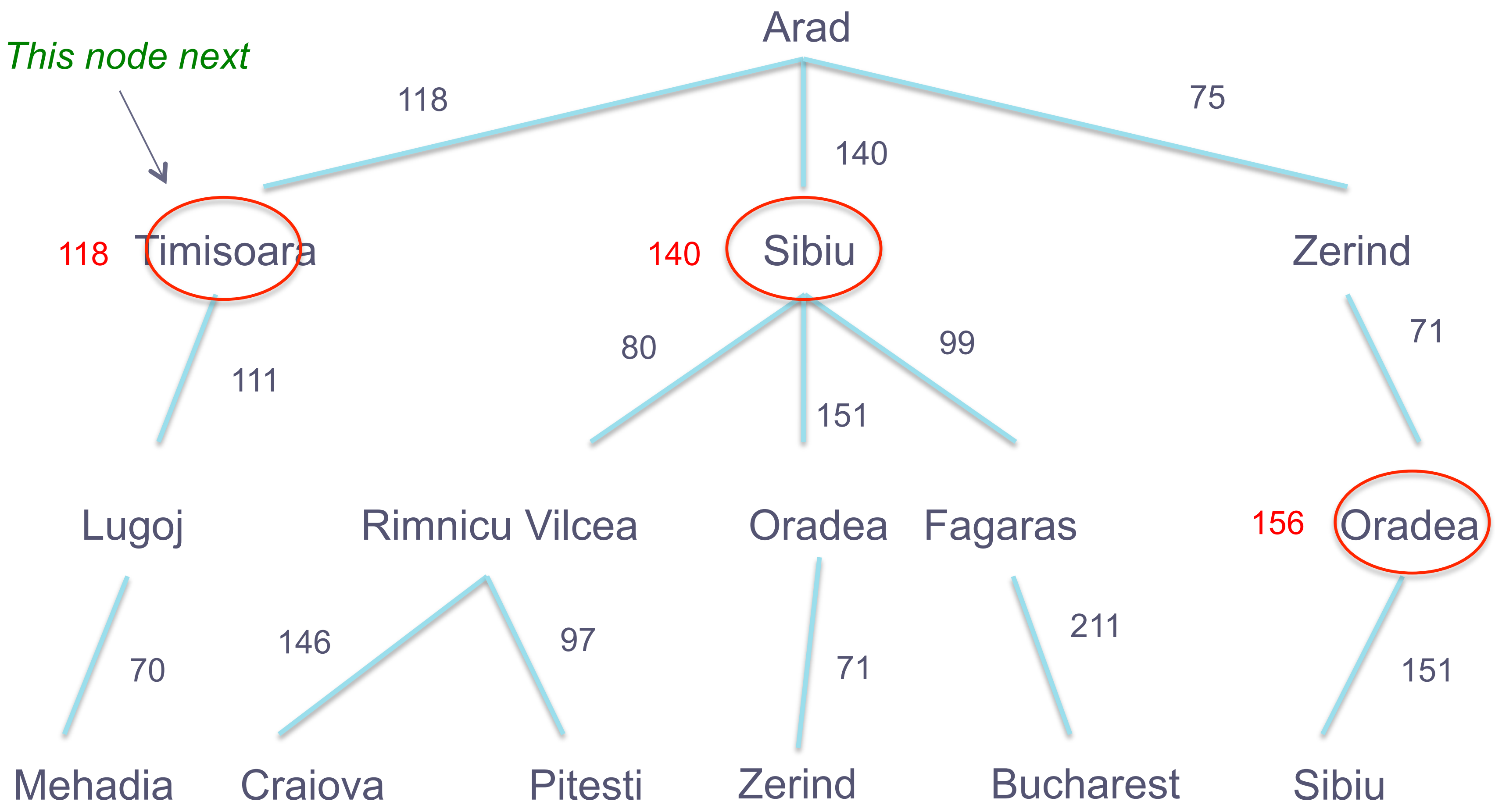
Uniform Cost Search 1



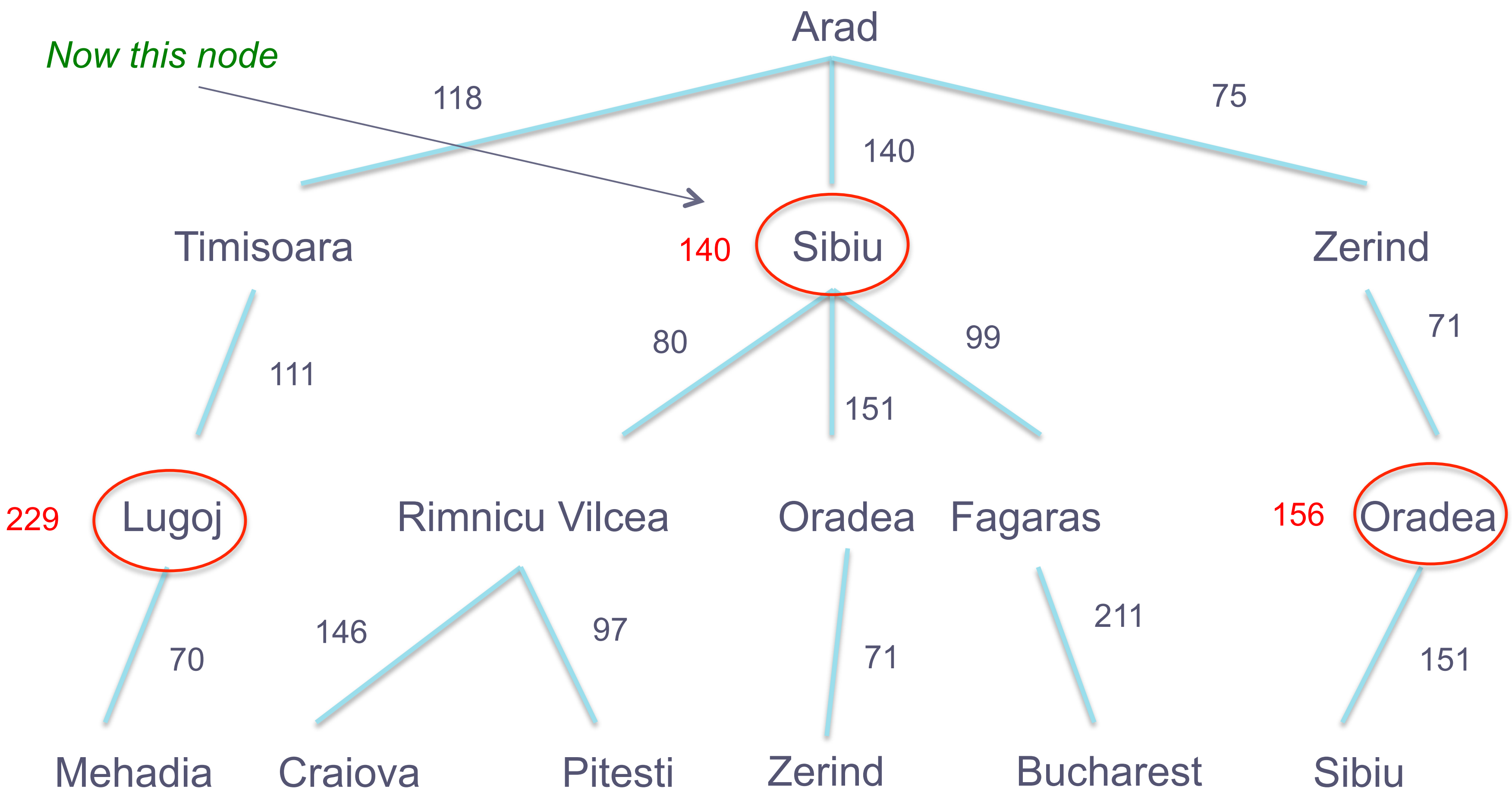
Uniform Cost Search 2



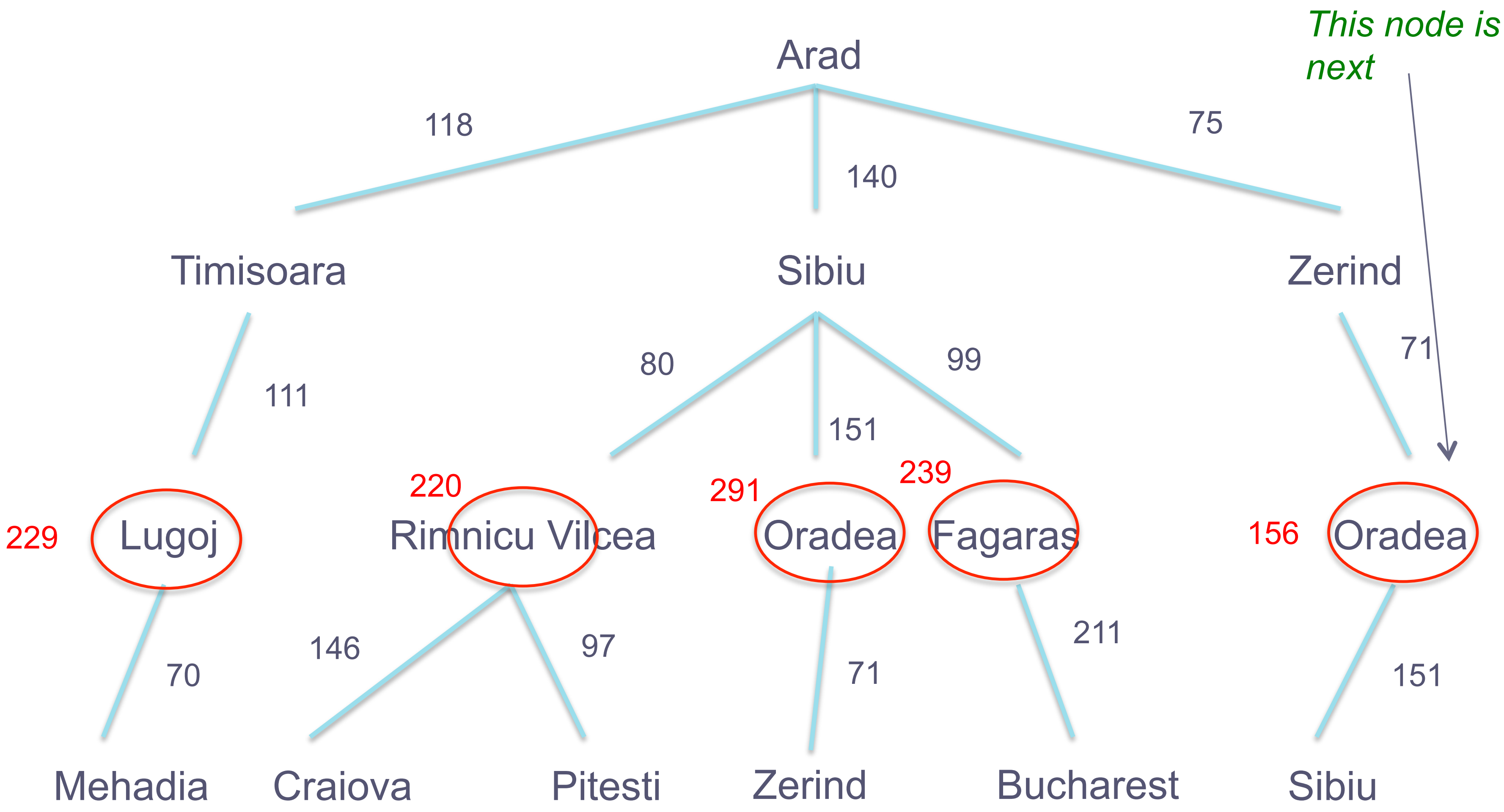
Uniform Cost Search 3



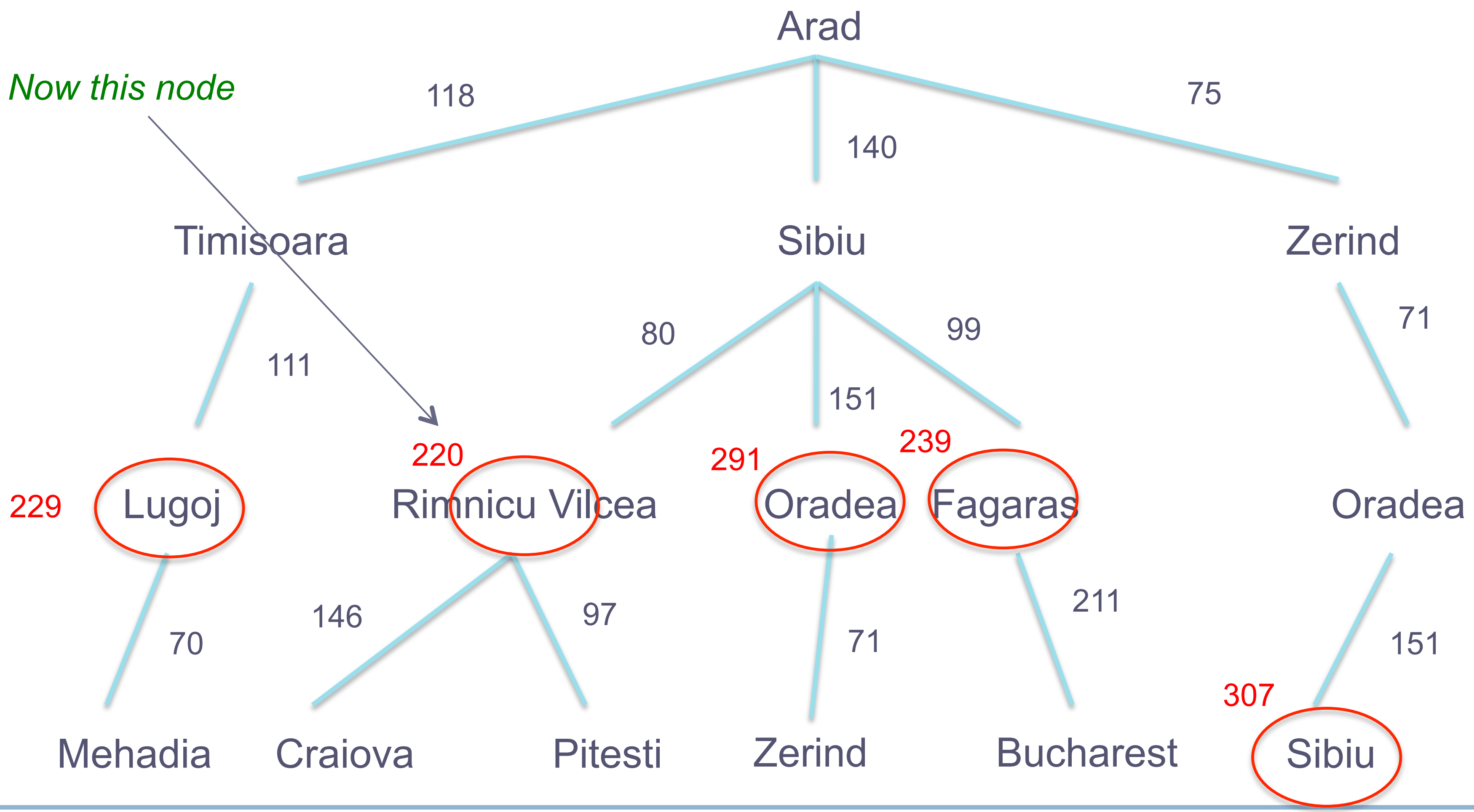
Uniform Cost Search 4



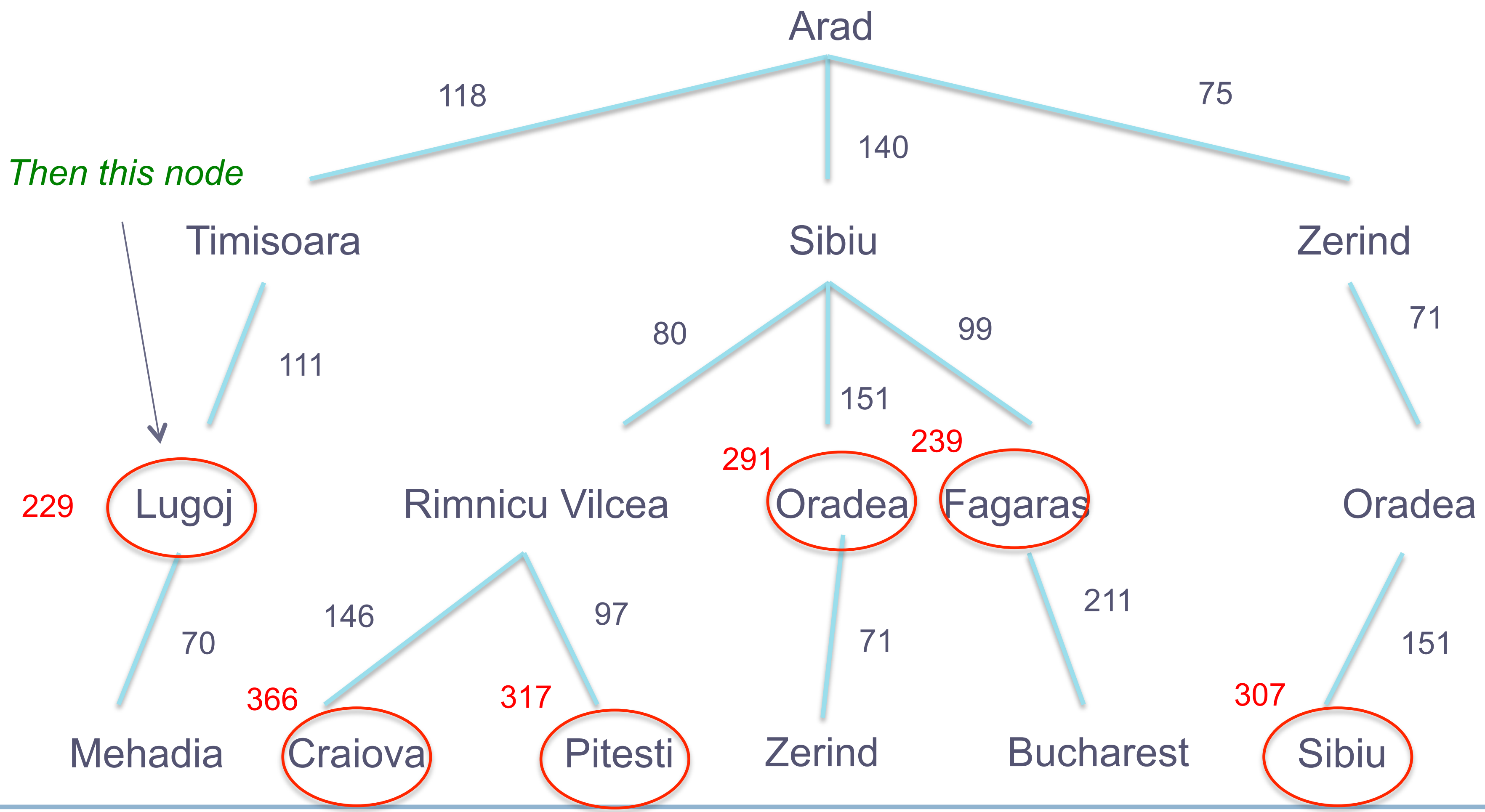
Uniform Cost Search 5



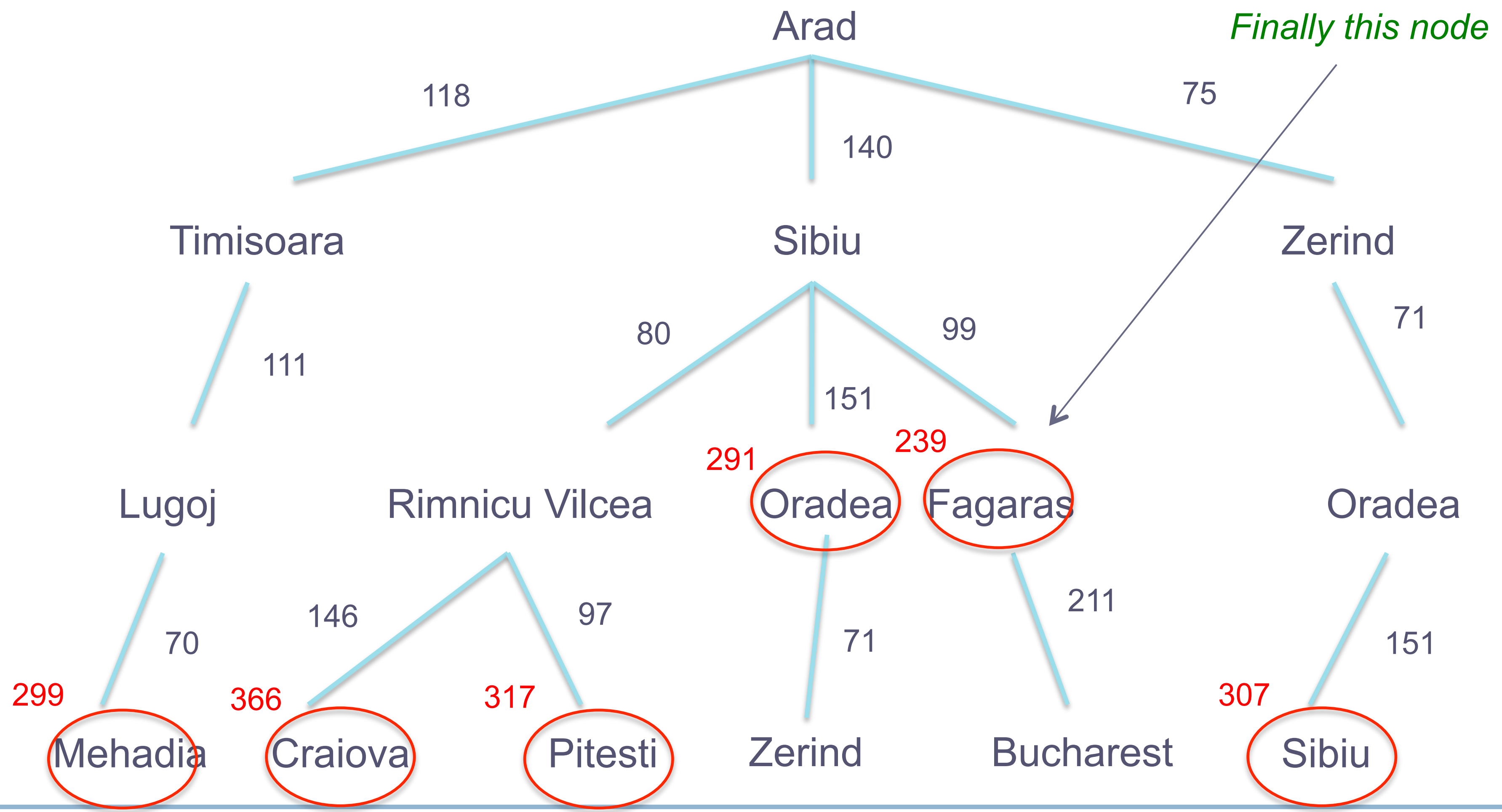
Uniform Cost Search 6



Uniform Cost Search 7

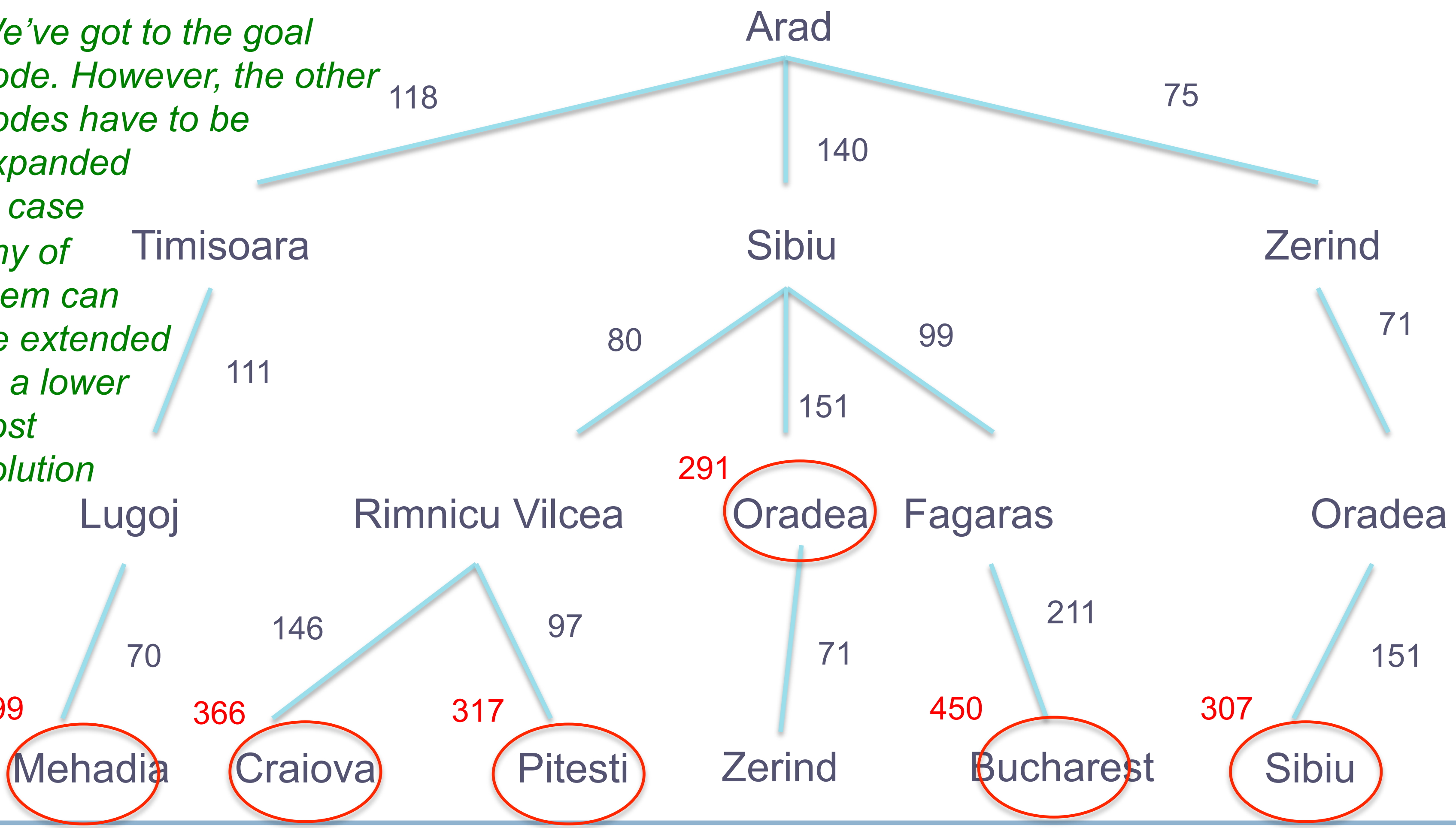


Uniform Cost Search 8



Uniform Cost Search 9

We've got to the goal node. However, the other nodes have to be expanded in case any of them can be extended to a lower cost solution



Properties of Uniform Cost Search

- Guaranteed to find a solution if one exists, as long as costs are all above some ε where $\varepsilon > 0$ (to avoid getting stuck in infinite branches)
  - Note: it's not enough for all costs to be above zero
  - Consider an infinite branch with successive costs $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$
- Time and memory use proportional to number of nodes with cost less than that of optimal solution
- Guaranteed to find optimal solution