Advanced Artificial Intelligence

Lab 02

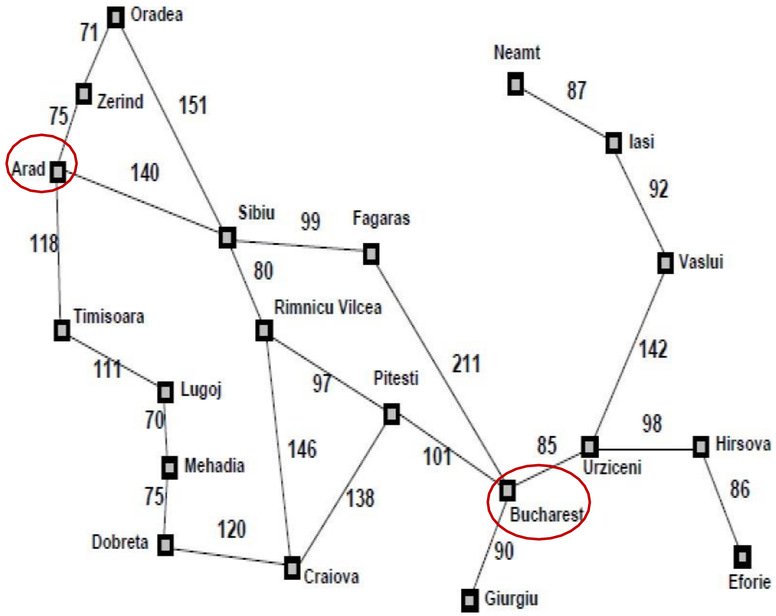
Outline

- A concrete problem
- Implementation of different search algorithms for this problem
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search
  - Bidirectional search
- Exercise

Problem Formulation

Objective: Find the shortest path from *Arad* to *Bucharest*.

- States: {(cur_city,walk_dist)}
- Initial state: $(Arad, 0)$
- Actions: walk to an adjacent city.
- Next state: e.g. $RESULT((Arad,0), go\_to\_Sibiu)$
- Goal test: reach '*Bucharest*'?
- Path cost: accumulated walk distance.



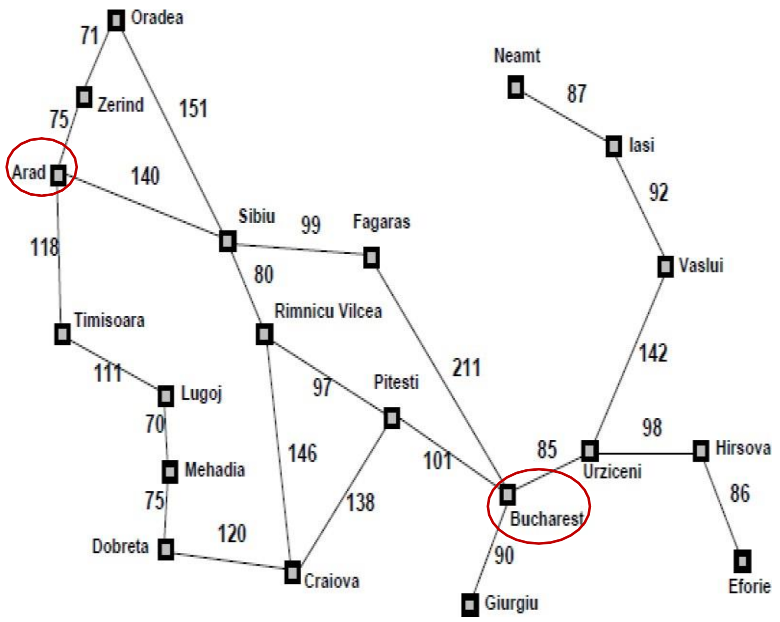
Breadth-related Search Methods

Remarks of BFS Searching

- Expand the shallowest unexpanded node.
- Data structure: a FIFO queue.

PF Metric	Breadth-first Search
Complete?	Yes*, if b is finite.
Optimal?	Yes*, if costs on the edge are non-negative.
Time?	$O(b^d)$
Space?	$O(b^d)$

b – maximum # successors of any node in search tree.
d – depth of the least-cost solution.

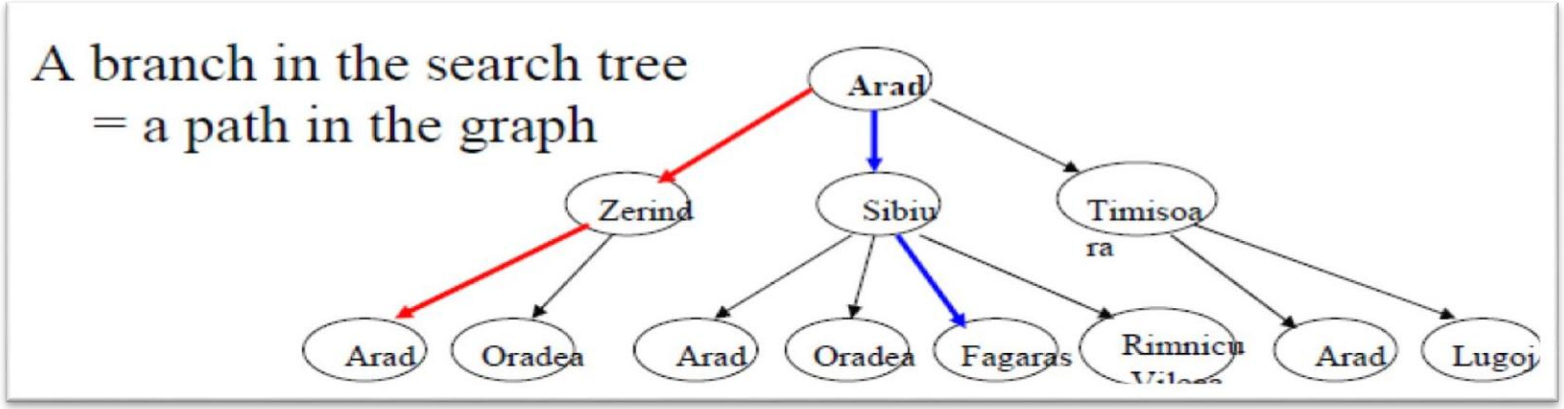


BFS: Pseudo-code

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)   /* chooses the shallowest node in frontier */
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
           frontier ← INSERT(child, frontier)
```

Remarks of Search Trees

- A search tree models the sequence of legal actions.
  - Root: initial state.
  - Nodes: the states resulting from actions.
  - Child nodes: the follow-up states of a previous node.
  - Branch: a sequence of states (and thereby a sequence of actions).
- Expand: create all children nodes for a given node.

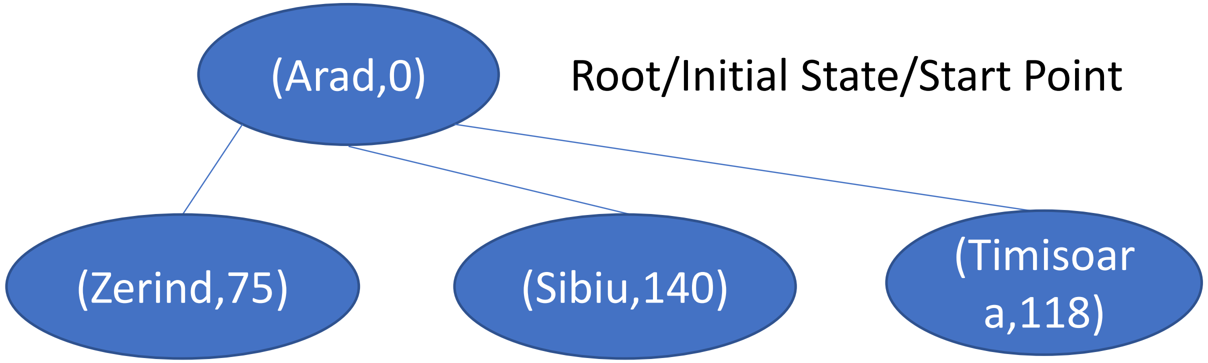


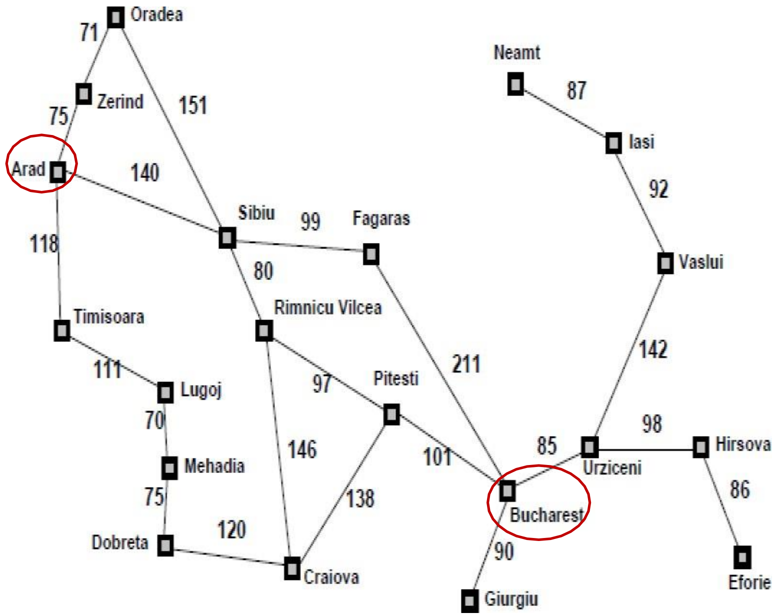
Breadth-first Search Tree

Tree Node:

- 4 components of node n:
  - n.STATE: node n's state(s).
  - n.PARENT: node that generated n.
  - n.ACTION: the action applied to the *parent* to generate node n.
  - n.PATHCOST: the cost of the entire path from the initial state.

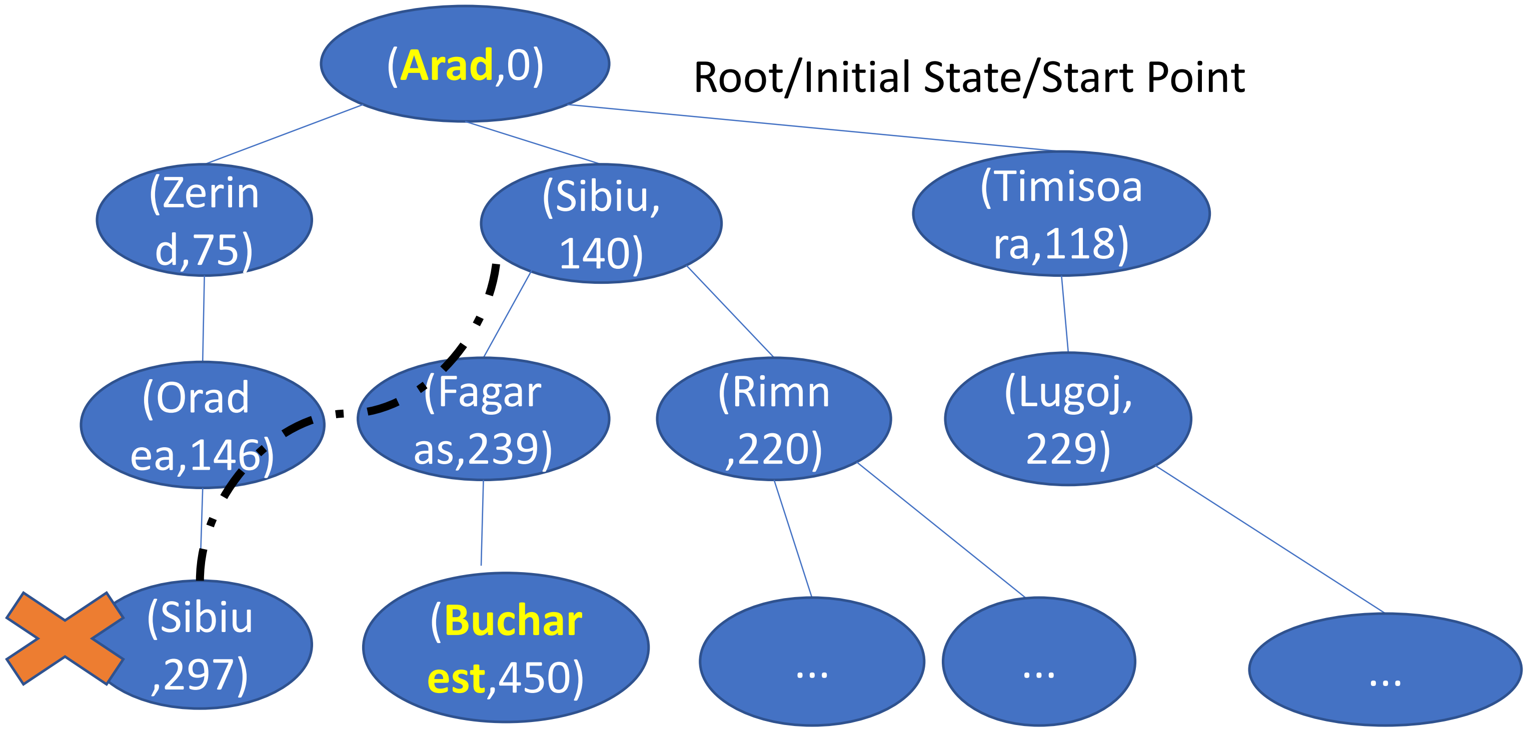
To simplify, only show state (include distance) in nodes here.

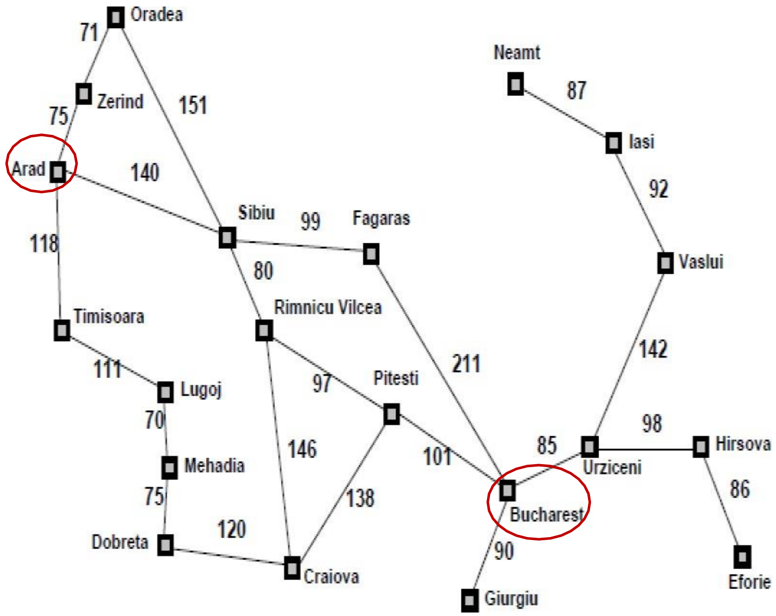




How about next layer of BFS Search Tree?

Breadth-first Search Tree

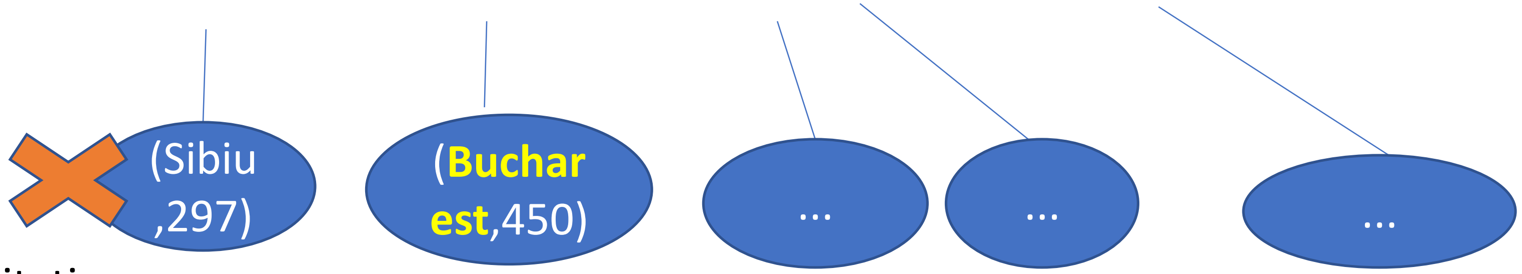




Elimitation:
Non-cyclic State Repetation

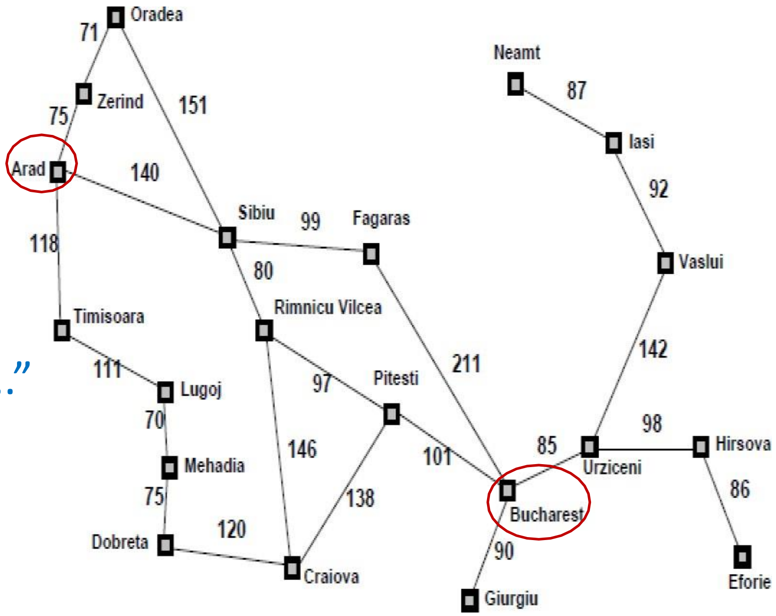
Target Arrived!!

Breadth-first Search Tree



Elimitation:
Non-cyclic State Repetation

**Target
Arrived!!**



Now, An available path has been searched by BFS.

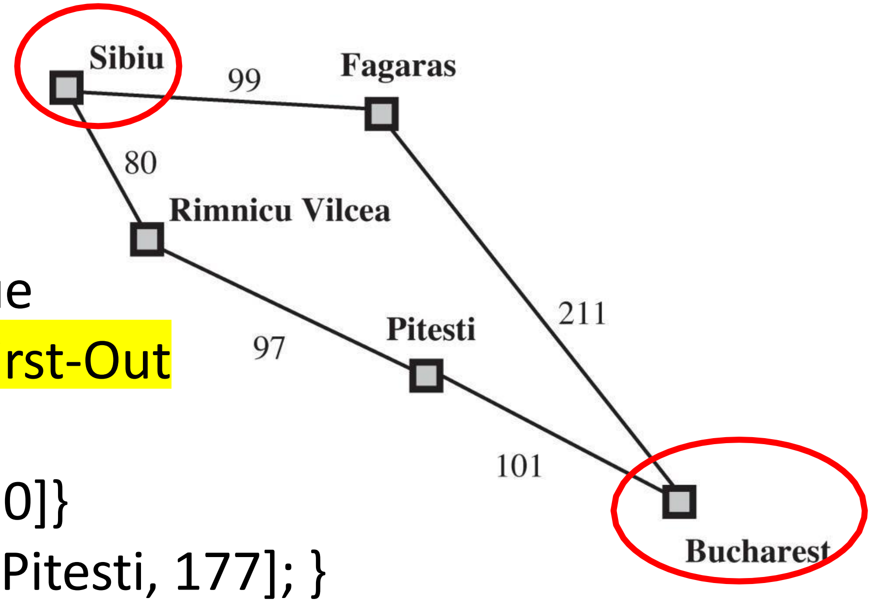
But the process of BFS is not finish! Still three branches to be expanded. i.e. "…"

We can finish the tree and find several different paths from Arad to Bucharest.
And choose the shortest one.

To be wiser, we can also use some strategys, such as eliminate nodes whose distance ≥ 450

Breadth-first Search(BFS)

Consider an easier graph (partial of previous one)
How about the **FIFO** queue?

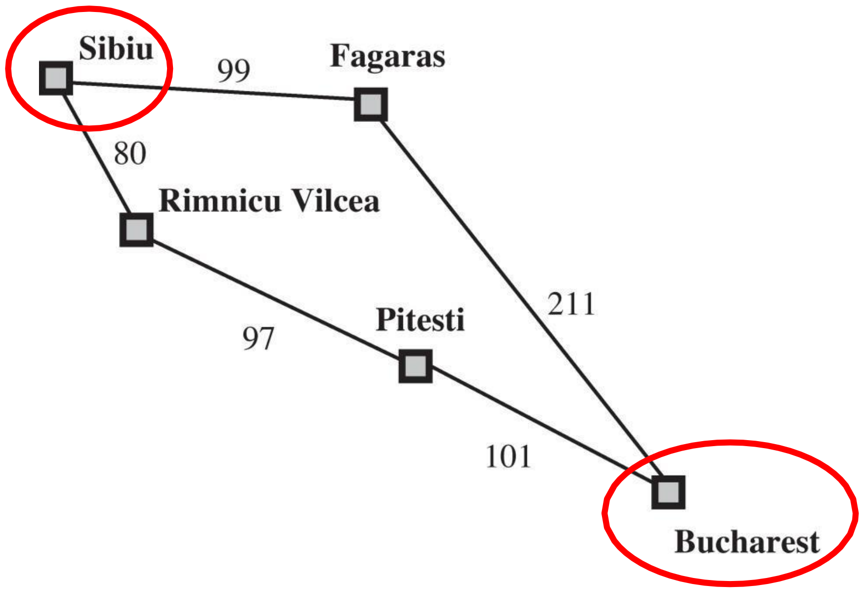


- **Task**: from Sibiu to Bucharest. **Structure**: FIFO queue
- [0] {[Sibiu, 0]} Expand the shallowest node, First-In-First-Out
- [1] {[Sibiu→Fagaras, 99]; [Sibiu→Rimnicu, 80]}
- [2] {[Sibiu→Rimnicu,80]; [Sibiu→Fagaras→Bucharest, 310]}
- [3] {[Sibiu→Fagaras→Bucharest, 310];[Sibiu→Rimnicu→Pitesti, 177]; }
- [4] {[Sibiu→Rimnicu→Pitesti→Bucharest, 278];}
- [5] {}

Uniform-cost Search (UCS)

- The path costs in the search tree may be different.
- Expand the cheapest unexpanded node.
- Data structure: a queue ordered by the path cost, the lowest first.

- **Task**: from Sibiu to Bucharest. **Structure**: priority queue
- [0] {[Sibiu, 0]} Expand the cheapest unexpanded node
- [1] {[Sibiu→Rimnicu, 80]; [Sibiu→Fagaras, 99]}
- [2] {[Sibiu→Rimnicu→Pitesti, 177]; [Sibiu→Fagaras, 99]}
- [3] {[Sibiu→Rimnicu→Pitesti, 177]; [Sibiu→Fagaras→Bucharest, 310]}
- [4] {[Sibiu→Rimnicu→Pitesti→Bucharest, 278]; [Sibiu→Fagaras→Bucharest, 310]}



UCS: Pseudo-code

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)   /* chooses the lowest-cost node in frontier */
      if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              frontier ← INSERT(child, frontier)
          else if child.STATE is in frontier with higher PATH-COST then
              replace that frontier node with child
```

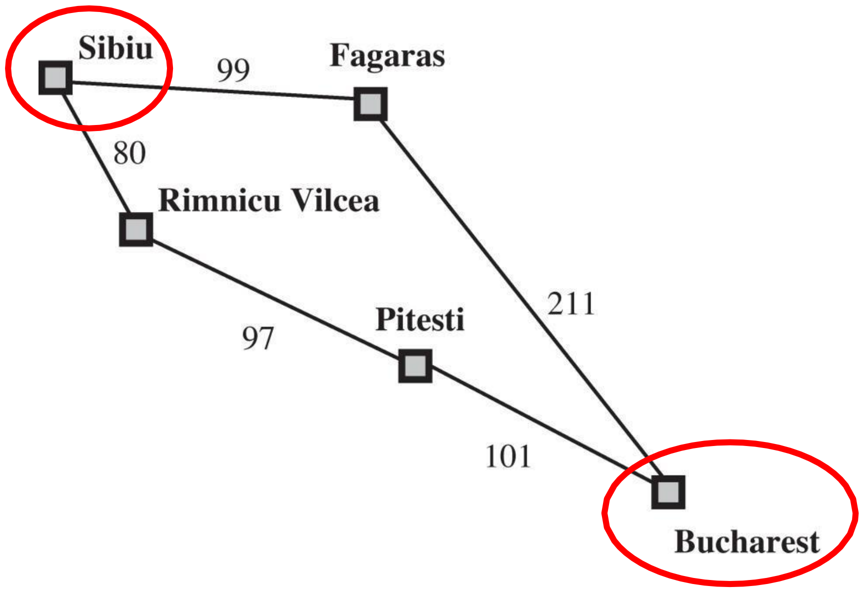
PF Metric	Uniform-cost Search
Complete?	Yes*, if step costs$\geq \epsilon$.
Optimal?	Yes
Time?	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$
Space?	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$

Depth-related Search Methods

Remarks of DFS Searching

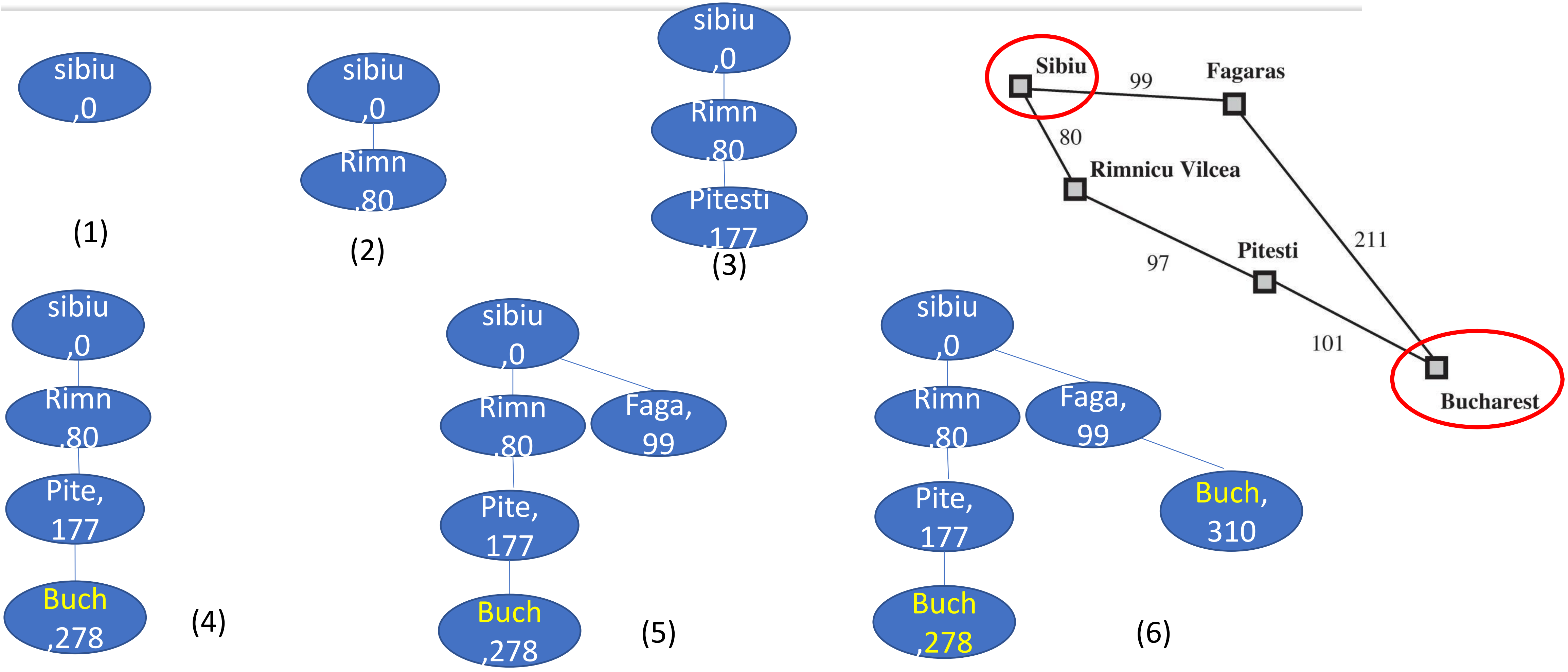
- Expand the deepest unexpanded node.
- Data structure: LIFO stack.

PF Metric	Depth-first Search
Complete?	No, infinite loops can occur.
Optimal?	No
Time?	$O(b^m)$
Space?	$O(bm)$



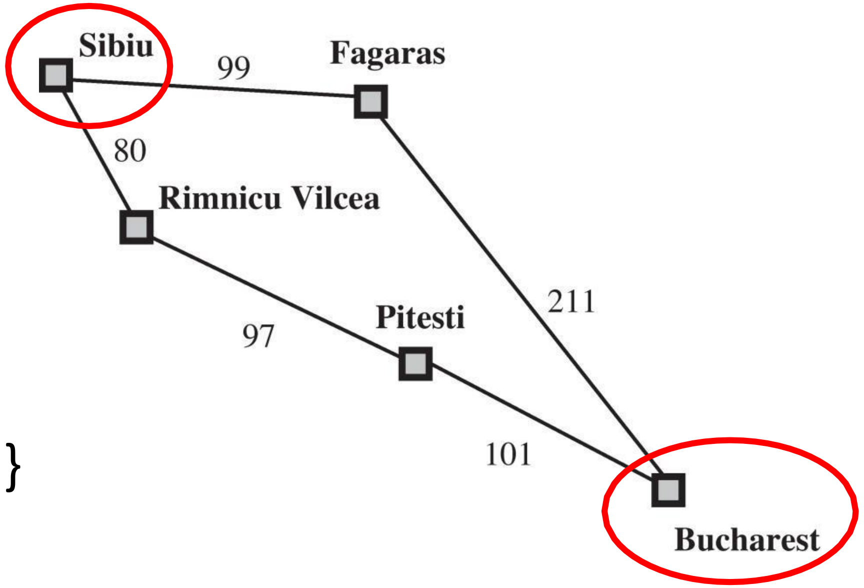
b – maximum # successors of any node in search tree.
d – depth of the least-cost solution.
m – maximum length of any path in the state space.

Depth-first Search Tree



Depth-first Search(DFS)

How about the **LIFO** stack?



- **Task**: from Sibiu to Bucharest. **Structure**: LIFO stack
- [0] {[Sibiu, 0]} Expand the deepest unexpanded node
- [1] {[Sibiu→Rimnicu, 80]; [Sibiu→Fagaras, 99]}
- [2] {[Sibiu→Rimnicu→Pitesti, 177]; [Sibiu→Fagaras, 99]}
- [3] {[Sibiu→Rimnicu→Pitesti→Bucharest, 278]; [Sibiu→Fagaras, 99]} A branch finish!
- [4] {[Sibiu→Rimnicu→Pitesti→Bucharest, 278]; [Sibiu→Fagaras, 99]}
- [5] {[Sibiu→Rimnicu→Pitesti→Bucharest, 278];[Sibiu→Fagaras→Bucharest, 310]}

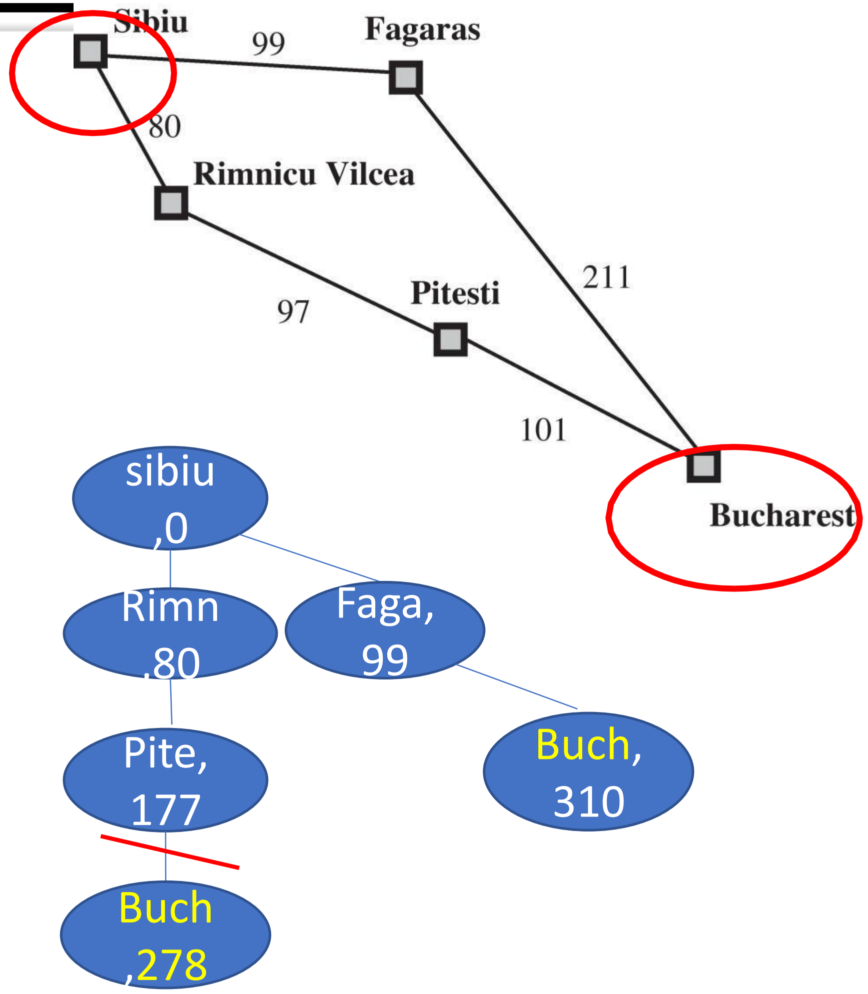
Depth-limited Search(DLS)

- DFS with depth limit ℓ: nodes at depth ℓ have no successors.
  - Limit ℓ is defined based on domain knowledge.
    - e.g. a traveling salesman problem with 20 cities → $\ell < 20$.
  - DLS is a variant of DFS.

- DLS overcomes the failure of DFS in an infinite-depth space.

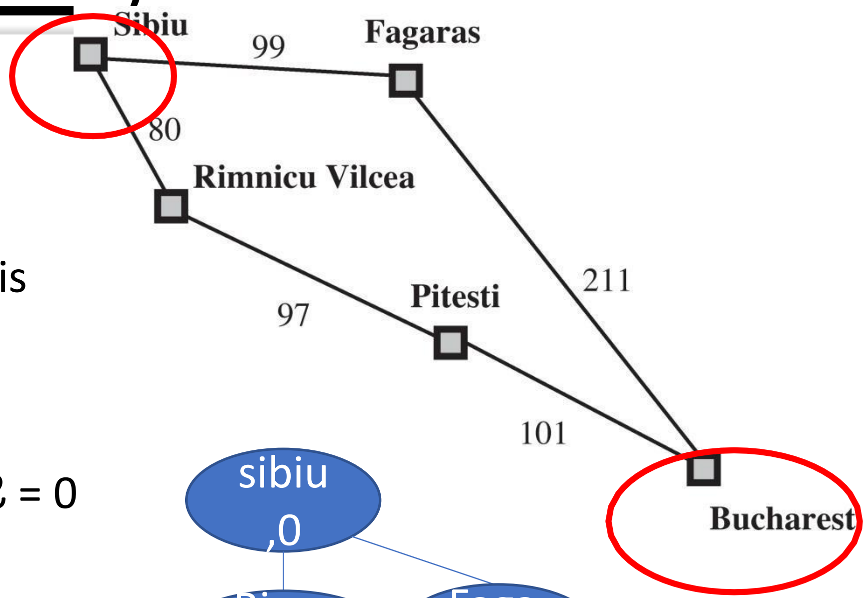
- In this Problem,
  - If $\ell \geq 3$, the process is the same as DFS.
  - If $\ell = 2$, the lower path stop at Pitesti. Not optimal ! !
    [Sibiu→Rimn→Pitesti~~→~~Bucharest]   Depth limited 2

  - And If $\ell \leq 1$, the search is not complete ($\ell < d=2$).
    i.e. No available path is searched by too shallow depth limit



Iterative Deepening Search (IDS)

- Apply DLS with increasing limits.
- Combine the benefits of BFS and DFS.
  - Like BFS, complete when b is finite & optimal when the path cost is non-decreasing regarding the depth of the nodes.
  - Like DFS, time complexity is $O(b^d)$.

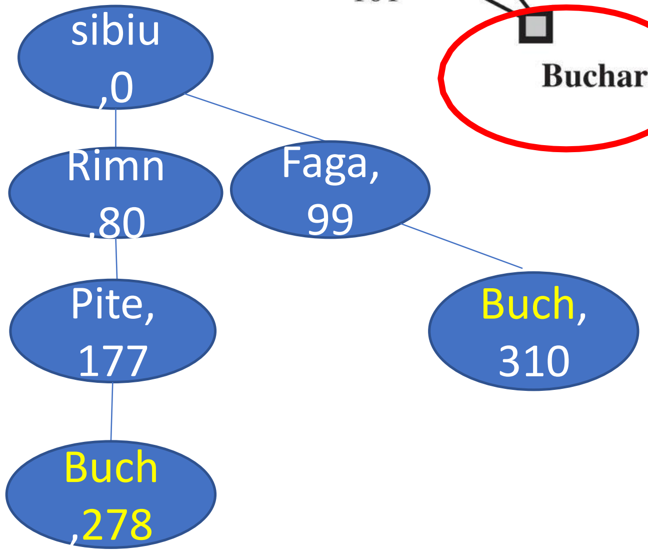


ℓ = 0

ℓ = 1

ℓ = 2

ℓ = 3



- If initial limit ℓ = 0 and increase 1 by time, IDS is similar to BFS

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution, or failure
  for *depth* = 0 **to** ∞ **do**
    result ← DEPTH-LIMITED-SEARCH(*problem*, *depth*)
    if *result* $\neq$ cutoff **then return** *result*

Iterative Deepening Search (IDS)

- Apply DLS with increasing limits.
- Combine the benefits of BFS and DFS.
    - Like BFS, complete when b is finite & optimal when the path cost is non-decreasing regarding the depth of the nodes.
    - Like DFS, time complexity is $O(b^d)$.

- If initial limit ℓ = 0 and increase 1 by time, IDS is similar to BFS

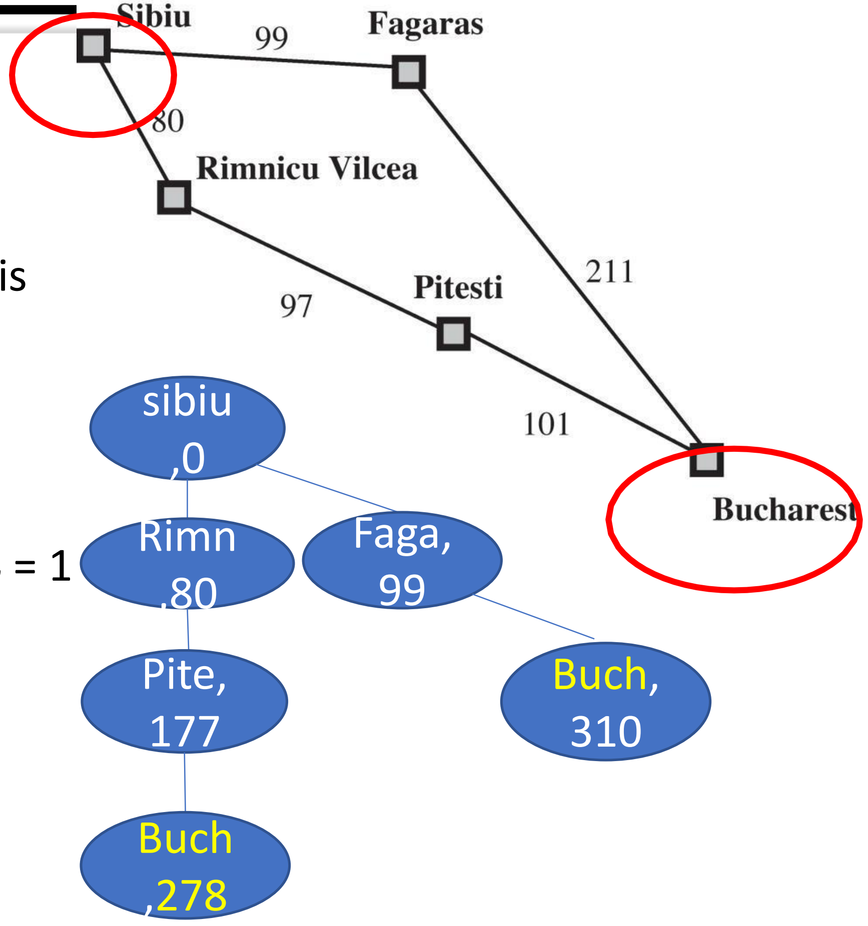
- But we can tune initial ℓ and increase step-length wisely.

    For example, set :

    initial ℓ = 1,

    increase 4 each iteration

initial ℓ = 1

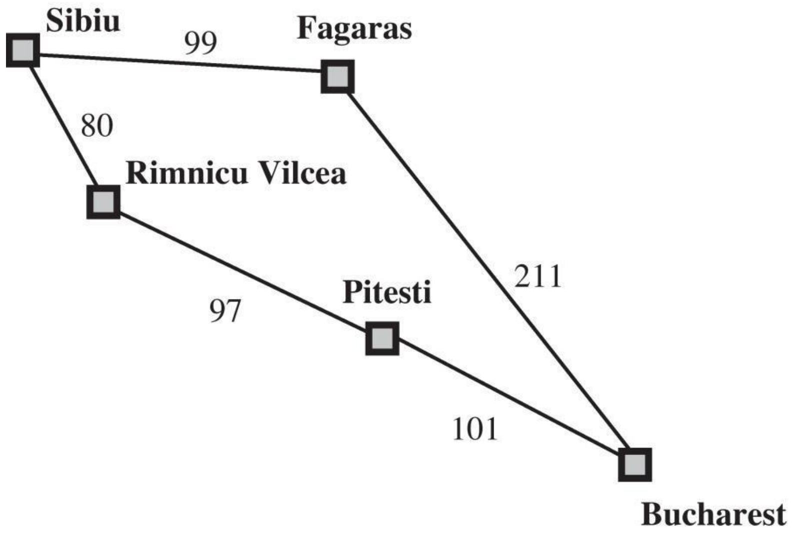


second ℓ = 5

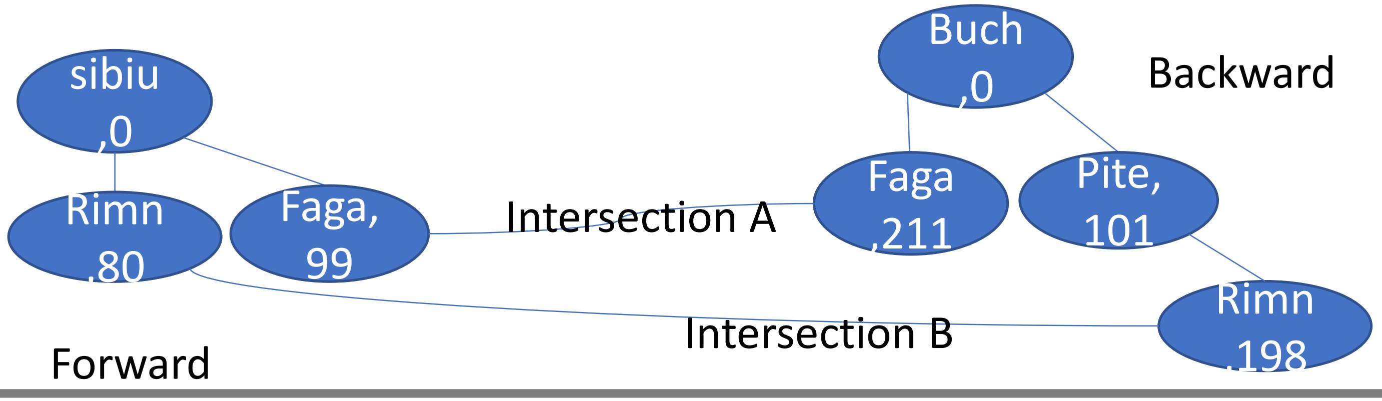
Bidirectional Search Method

Bidirectional Search

- Search from forward & backward directions simultaneously.
- Replace a single search tree with two smaller sub-trees.
  - Forward tree: forward search from source to goal.
  - Backward tree: backward search from goal to source.
- Goal test: two sub-trees intersect.



Bidirectional BFS Search Tree:



Bidirectional Search

- Complete? Yes, if BFS is used in both search.
- Optimal? Yes, if BFS is used & paths have a uniform cost.
- Time? Space? $O(b^{d/2})$

b – maximum # successors of any node in search tree.
d – depth of the least-cost solution.
m – maximum length of any path in the state space.

Intersection A:

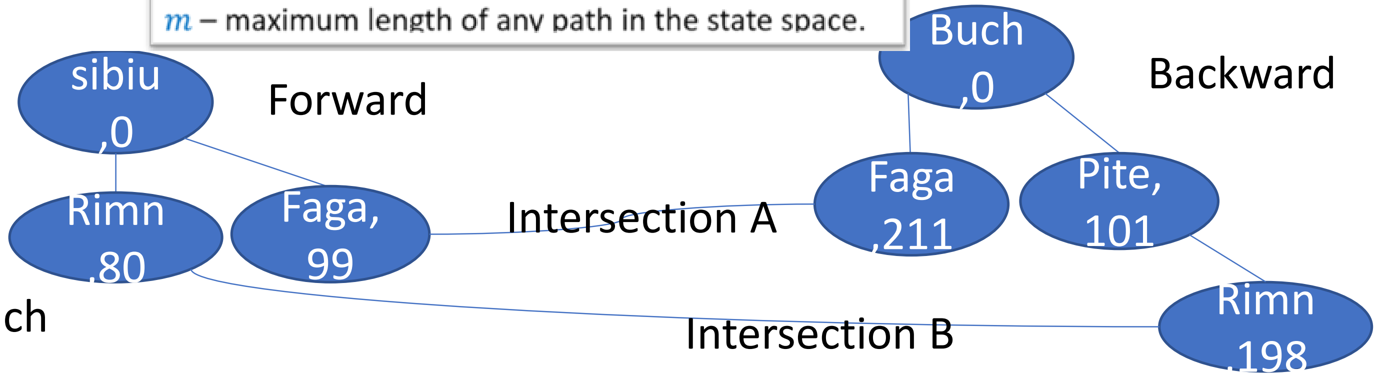
Sibiu → Faga → Buch

99 + 211 = 310

Intersection B:

Sibiu → Rimn → Pite → Buch

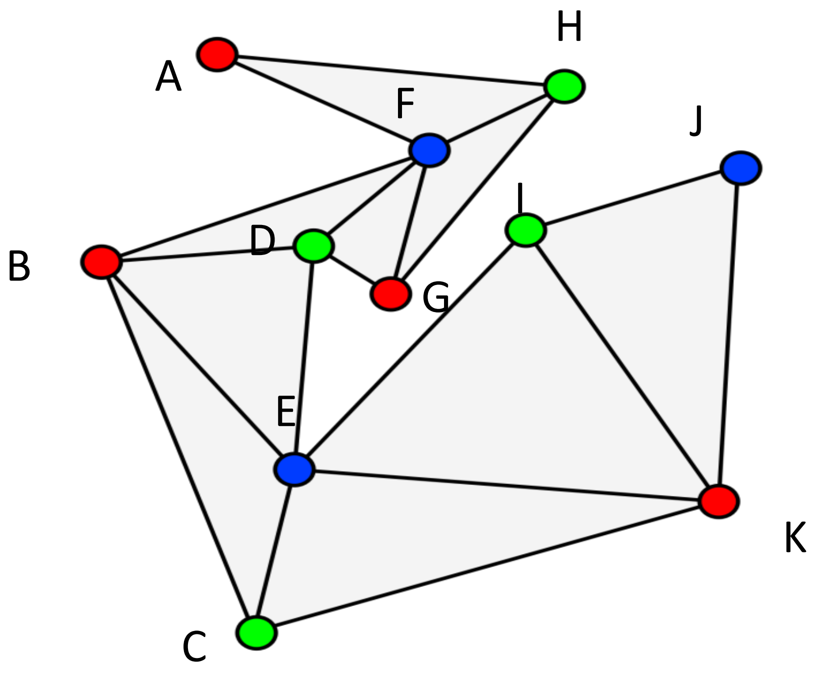
80 + 198 = 278



Bidirectional also adaptive to Depth based search methods,

Think about them after class! Complete? Optimal? And the complexity?

Exercise



1) Use BFS and DFS to find the shortest path from A to K, consider each edge has uniform cost. Draw the search trees separately.

2) Use UCS and IDS to find all paths from H to C. Please show the procedure separately.

3) Find all paths from red points pass green points to blue points, state your search method and show the procedure.