BITS Pilani
Pilani Campus

Artificial and Computational Intelligence

AIMLCZG557

Contributors & Designers of document content : Cluster Course Faculty Team

M2 : : Problem Solving Agent using Search

Presented by
Faculty Name
BITS Email ID

Artificial and Computational Intelligence

Disclaimer and Acknowledgement

- Few content for these slides may have been obtained from prescribed books and various other source on the Internet
- I hereby acknowledge all the contributors for their material and inputs and gratefully acknowledge people others who made their course materials freely available online.
- .I have provided source information wherever necessary
- This is not a full fledged reading materials. Students are requested to refer to the textbook w.r.t detailed content of the presentation deck that is expected to be shared over e-learning portal - taxilla.
- I have added and modified the content to suit the requirements of the class dynamics & live session's lecture delivery flow for presentation

- **Slide Source / Preparation / Review:**
- From BITS Pilani WILP**:** Prof.Raja vadhana, Prof. Indumathi, Prof.Sangeetha
- From BITS Oncampus & External : Mr.Santosh GSK

Course Plan

M1	Introduction to AI
M2	Problem Solving Agent using Search
M3	Game Playing
M4	Knowledge Representation using Logics
M5	Probabilistic Representation and Reasoning
M6	Reasoning over time
M7	Ethics in AI

Learning Objective

At the end of this class , students Should be able to:

1. Design problem solving agents
2. Create search tree for given problem
3. Apply uninformed search algorithms to the given problem
4. Compare performance of given algorithms in terms of completeness, optimality, time and space complexity
5. Differentiate for which scenario appropriate uninformed search technique is suitable and justify

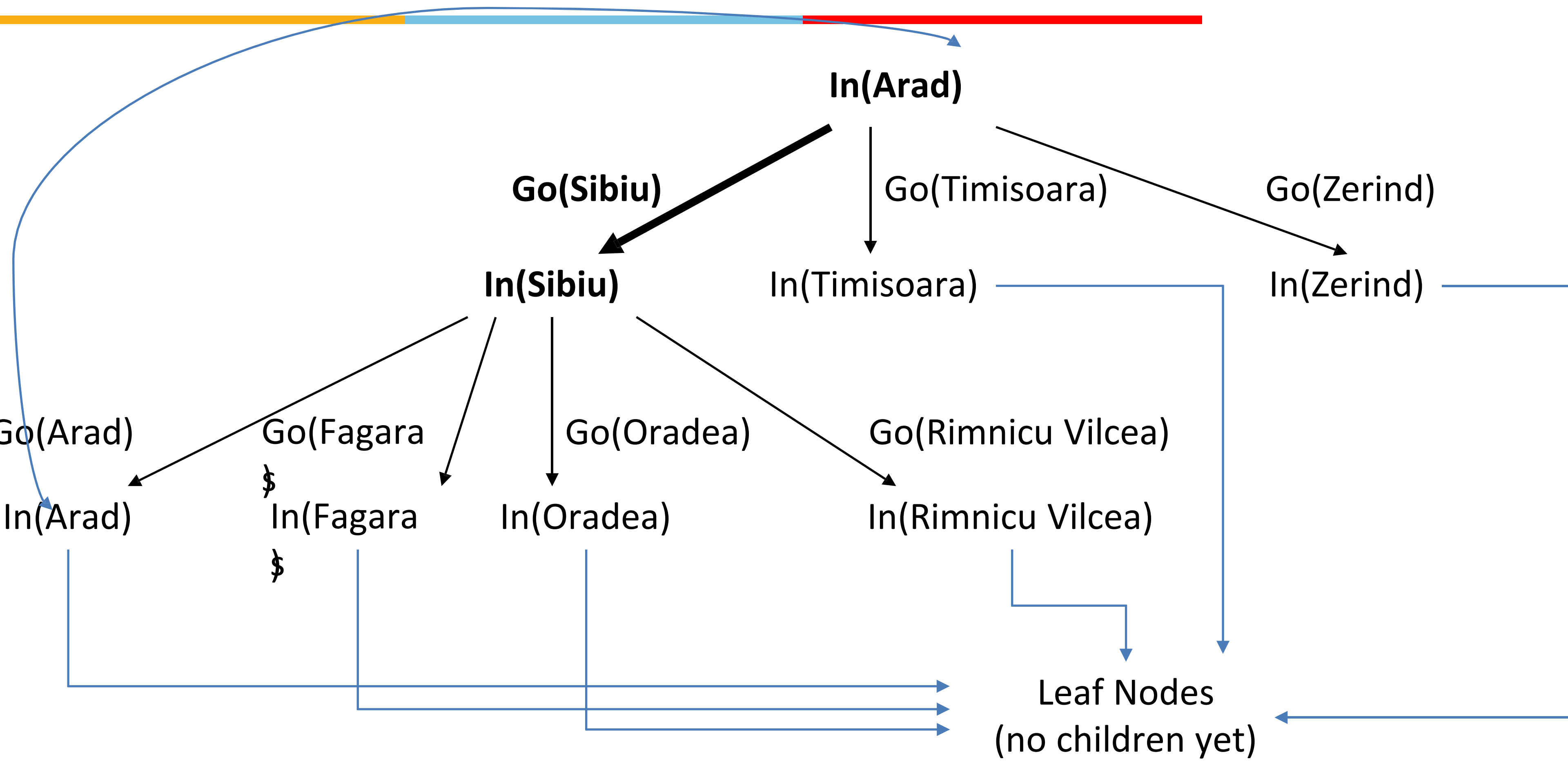
Problem Formulation

Searching for Solutions

Choosing the current state, testing possible successor function, expanding current state to generate new state is called Traversal. Choice of which state to expand – Search Strategy

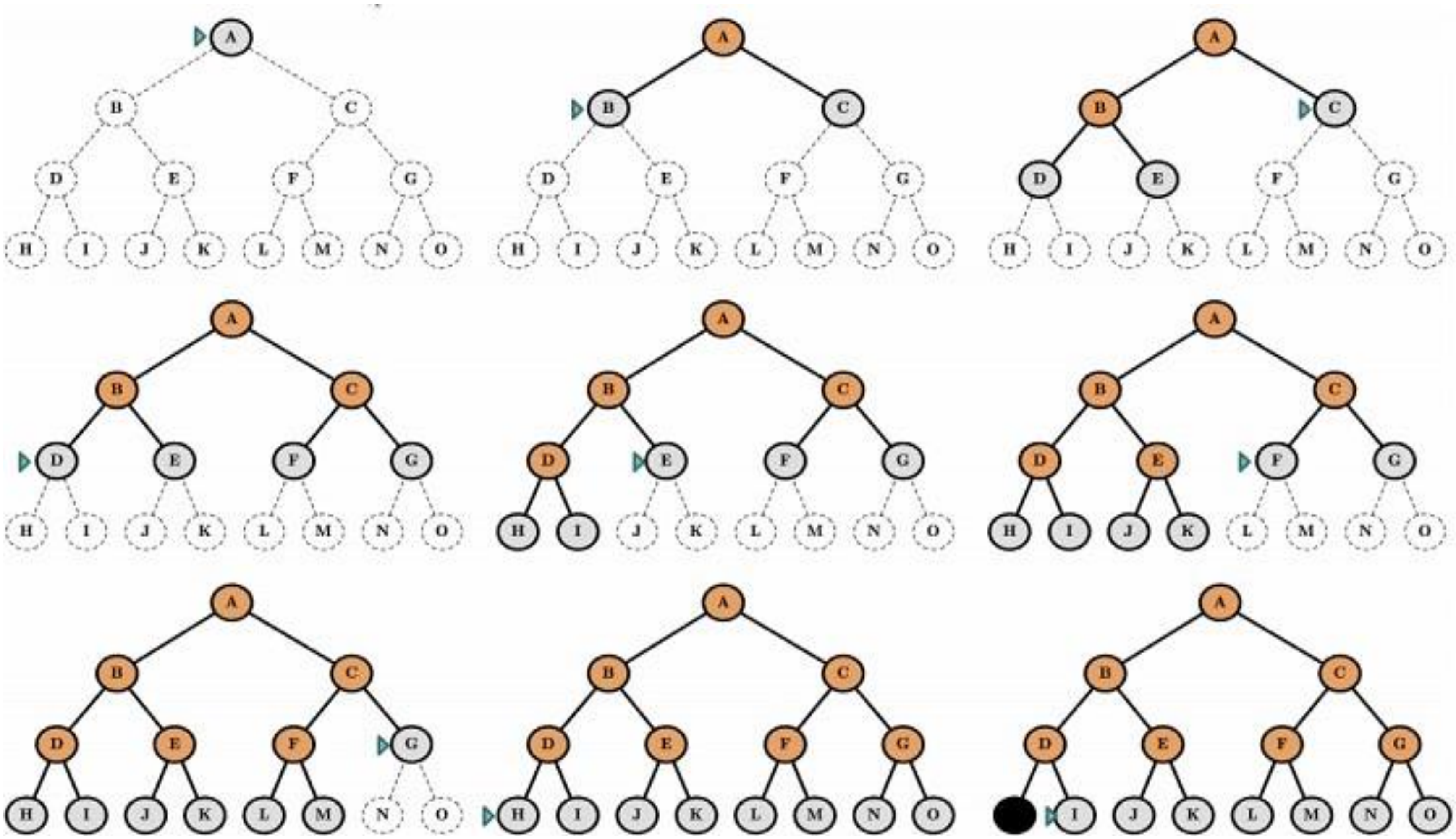
Search Strategy (under certainty)

- **Uninformed**
  - **BFS, DFS, UCS**
  - **IDS**, DLS
  - Bi-Directional
- **Informed**
  - Greedy Best First Search
  - A*
  - AO*

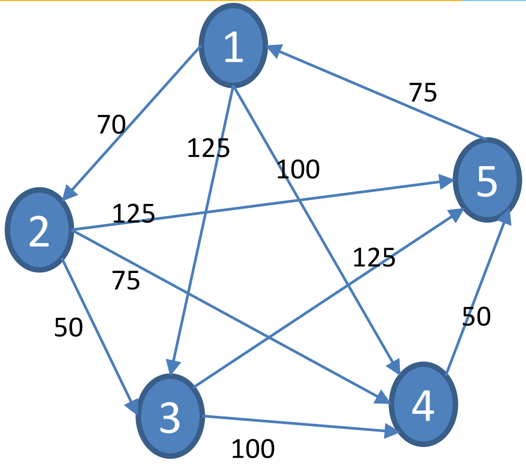

In(Arad)
Go(Sibiu)
Go(Timisoara)
Go(Zerind)
In(Sibiu)
In(Timisoara)
In(Zerind)
Go(Arad)
Go(Fagaras)
Go(Oradea)
Go(Rimnicu Vilcea)
In(Arad)
In(Fagaras)
In(Oradea)
In(Rimnicu Vilcea)
Leaf Nodes
(no children yet)

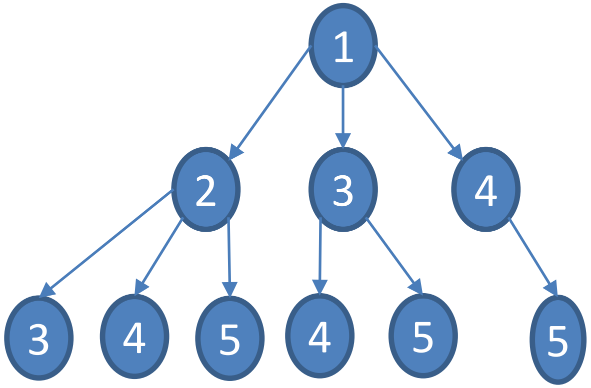
Uninformed Search – BFS & its Variant

Breadth First Search (BFS)



BFS – Uninformed





(1)
(1 2) (1 3) (1 4)
TEST FAILED
:
:
(1 3) (1 4) (1 2 3) (1 2 4) (1 2 5)
(1 2 3) (1 2 4) (1 2 5) (1 3 4) (1 3 5) (1 4 5)
TEST PASSED

C(1-2-5) = 70 + 125 =195
Expanded : 4
Generated : 10
Max Queue Length : 6

Breadth First Search – Evaluation

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

Why is Space Complexity a big problem? Imagine a problem with

– branching factor b = 10

– generates 1 million nodes/sec

– Each node requires 1KB

Breadth First Search – Evaluation

Complete – If the shallowest goal node is at a depth d, BFS will eventually find it by generating all shallower nodes

Optimal – Not necessarily. Optimal if path cost is non-decreasing function of depth of node. E.g., all actions have same cost

Time Complexity – $\mathcal{O}(b^d)$ b - branching factor, d – depth

- Nodes expanded at depth 1 = b
- Nodes expanded at depth 2 = b^2
- Nodes expanded at depth d = b^d
- Goal test is applied during generation, time complexity would be $\mathcal{O}(b^{d+1})$

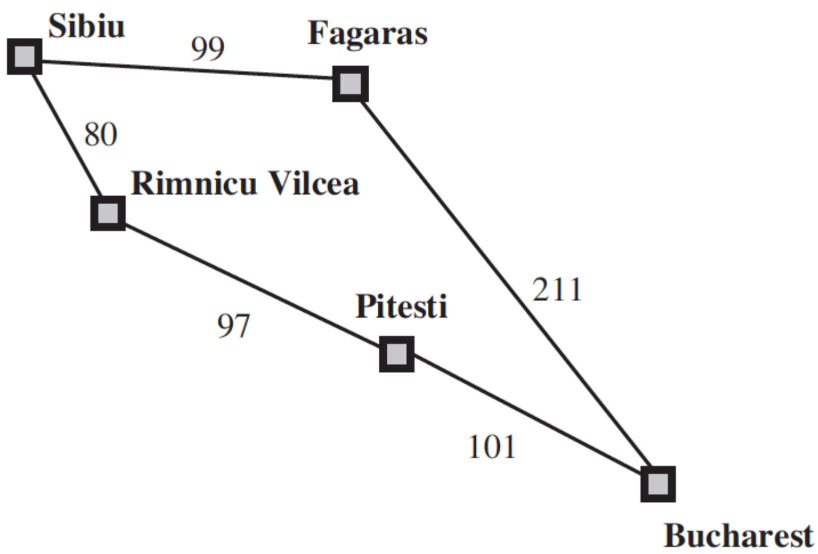
Space Complexity – $\mathcal{O}(b^d)$

- $\mathcal{O}(b^{d-1})$ in explored set
- $\mathcal{O}(b^d)$ in frontier set

Uniform Cost Search

- Instead of expanding the shallowest node, Uniform-Cost search expands the node n with the lowest path cost g(n)
- Sorting the Frontier as a priority queue ordered by g(n)
- Goal test is applied during expansion
  - The goal node if generated may not be on the optimal path
  - Find a better path to a node on the Frontier

Uniform Cost Search



Current State: Sibiu

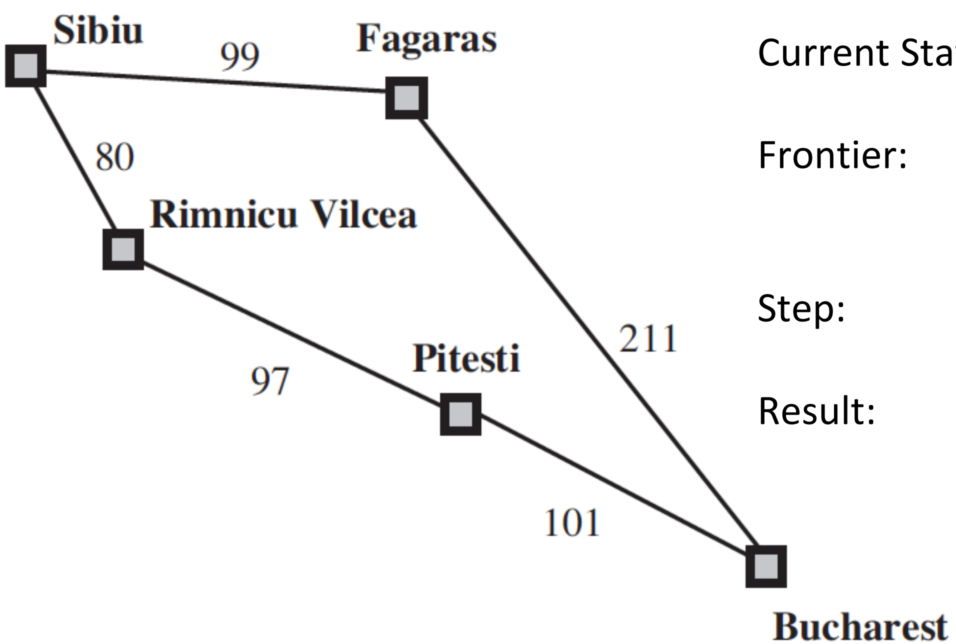
Frontier: []

Step: Expand Sibiu

Result: Generates (“Rimnicu Vilcea” 80)
(“Fagaras”, 99)
Add to Frontier

Initial State: Sibiu
Goal State: Bucharest

Uniform Cost Search



Current State: Sibiu

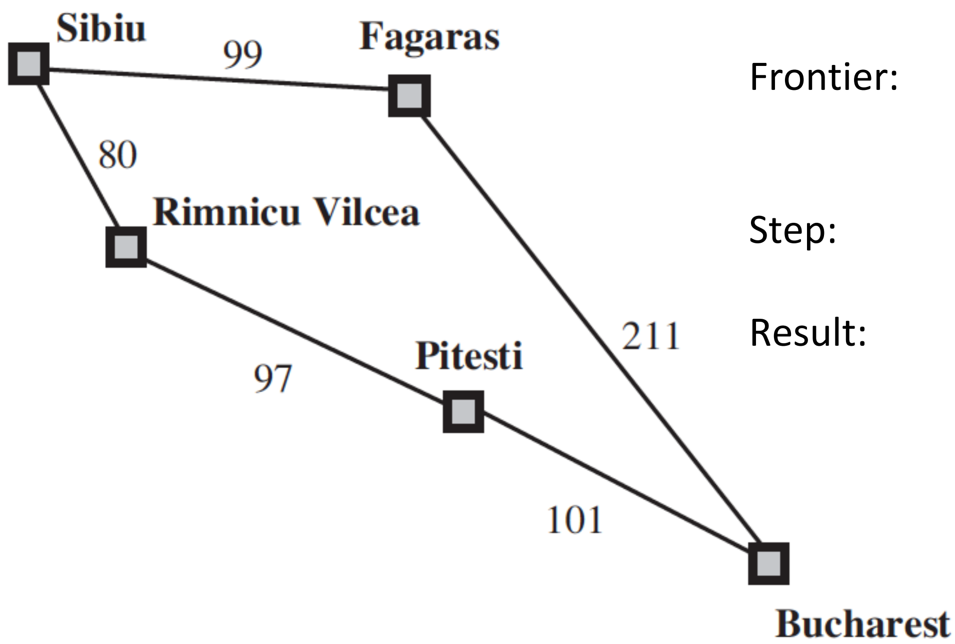
Frontier: [("Rimnicu Vilcea" 80) ("Fagaras", 99)]

Step: Expand "Rimnicu Vilcea" (least cost)

Result: Generates ("Pitesti", 177)
Add to Frontier

Initial State: Sibiu
Goal State: Bucharest

Uniform Cost Search



Current State: Rimnicu Vilcea (not a Goal state)

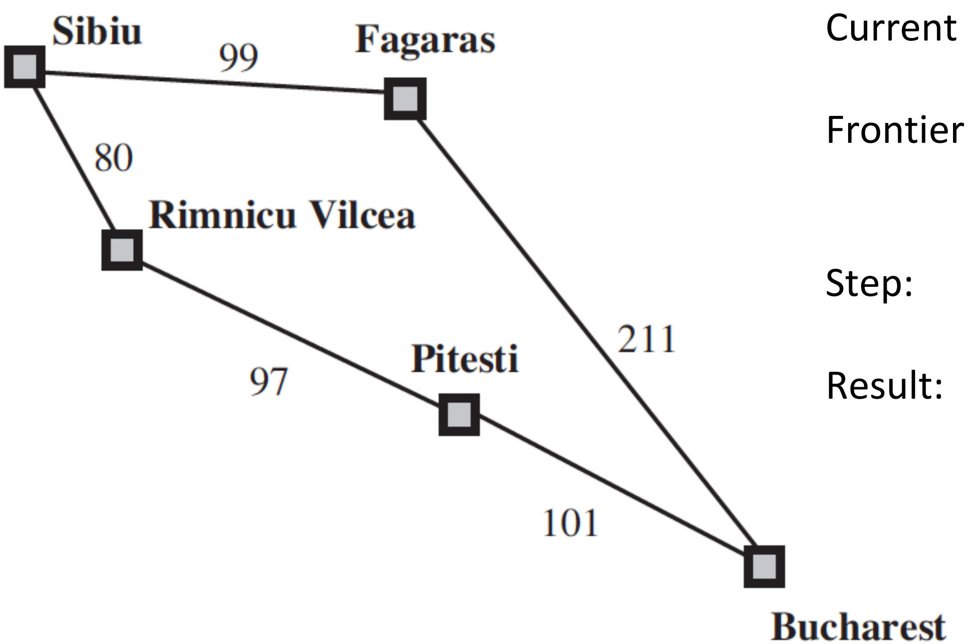
Frontier: [("Fagaras", 99)
("Pitesti", 177)]

Step: Expand "Fagaras" (least cost)

Result: Generates ("Bucharest", 310)
Add to Frontier
(It's a Goal State but we won't test during generation)

Initial State: Sibiu
Goal State: Bucharest

Uniform Cost Search



Current State: Fagaras (not a goal state)

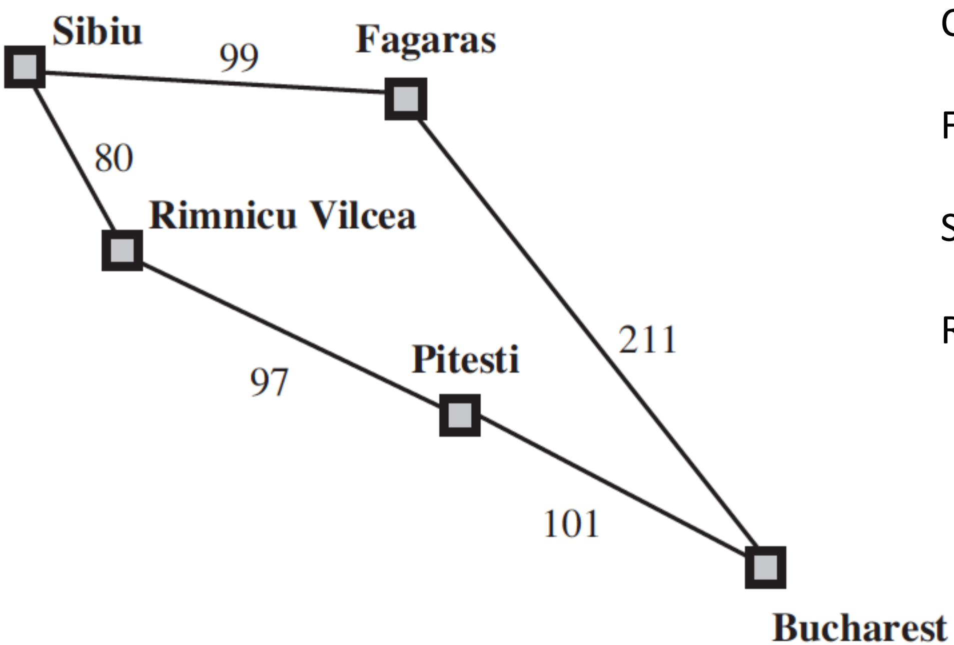
Frontier: [(“Pitesti”, 177)
(“Bucharest”, 310)]

Step: Expand “Pitesti” (least cost)

Result: Generates (“Bucharest”, 278)
Replace in Frontier
(It’s a Goal State but we won’t test during generation)

Initial State: Sibiu
Goal State: Bucharest

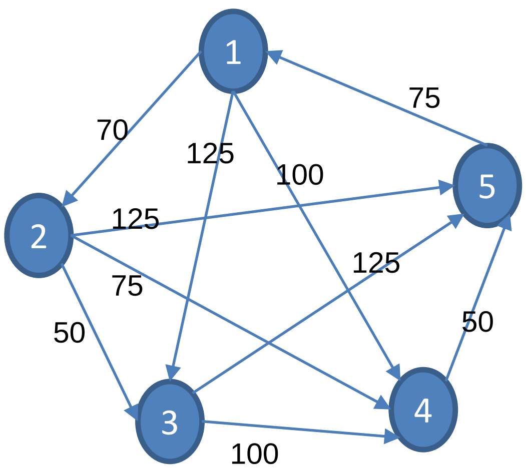
Uniform Cost Search

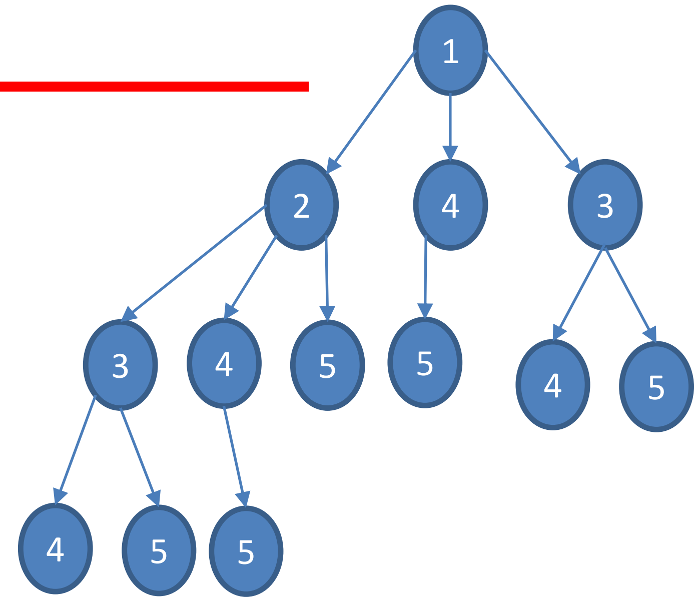


Current State:	Pitesti (not a goal state)
Frontier:	[(“Bucharest”, 278)]
Step:	Expand “Bucharest”
Result:	No further generation as Goal Test satisfied Return Solution

Initial State: Sibiu
Goal State: Bucharest

UCS





(1)
(1 2 : 70) (1 4 : 100) (1 3 : 125)
TEST-F
(1 4 : 100) (1 2 3 :120)(1 3 : 125) (1 2 4 : 145) (1 2 5 : 195)
TEST-F
(1 2 3 :120)(1 3 : 125) (1 2 4 : 145) (1 4 5: 150)(1 2 5 : 195)
TEST-F
(1 3 : 125) (1 2 4 : 145) (1 4 5: 150) (1 2 3 4:170) (1 2 5 : 195) (1 2 3 5 : 245)
TEST-F
(1 2 4 : 145) (1 4 5: 150) (1 2 3 4:170) (1 2 5 : 195) (1 3 4 : 225) (1 2 3 5 : 245) (1 3 5 : 250)
TEST-F
(1 4 5: 150) (1 2 3 4:170) (1 2 4 5 : 195) (1 2 5 : 195) (1 3 4 : 225) (1 2 3 5 : 245) (1 3 5 : 250)
TEST - P

Uniform Cost Search – Evaluation

Completeness – It is complete if the cost of every step > small +ve constant ∈

- It will stuck in infinite loop if there is a path with infinite sequence of zero cost actions

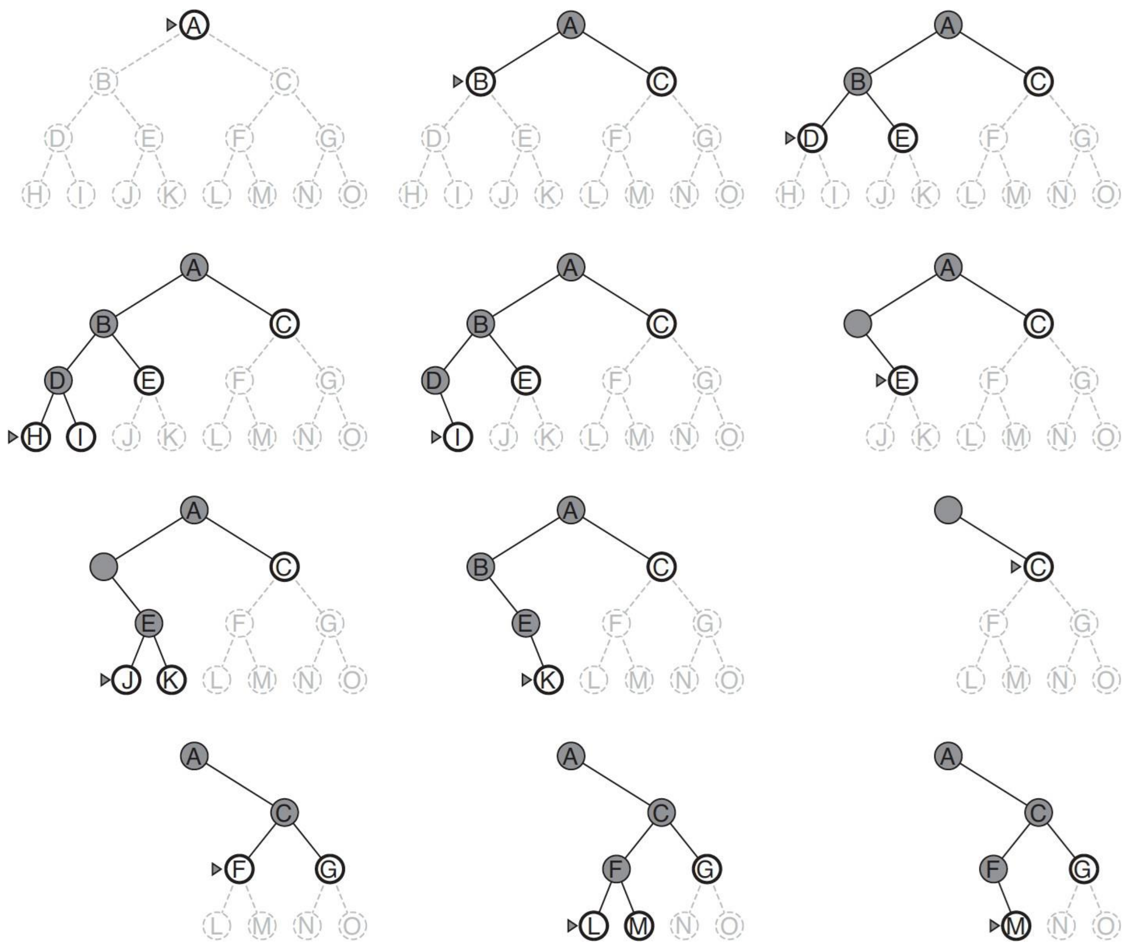
Optimal – It is Optimal. Whenever it selects a node, it is an optimal path to that node.

Time and Space complexity – Uniform cost search is guided by path costs not depth or branching factor.

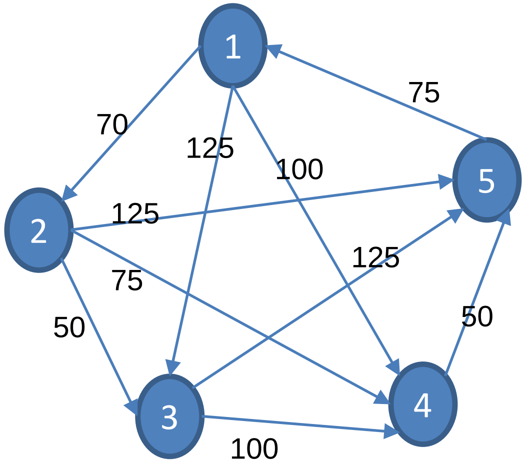
- If C* is the cost of optimal solution and ∈ is the min. action cost
- Worst case complexity = $\mathcal{O}(b^{1+\frac{C^*}{\epsilon}})$,
- When all action costs are equal → $\mathcal{O}(b^{d+1})$, the BFS would perform better
    - As Goal test is applied during expansion, Uniform Cost search would do extra

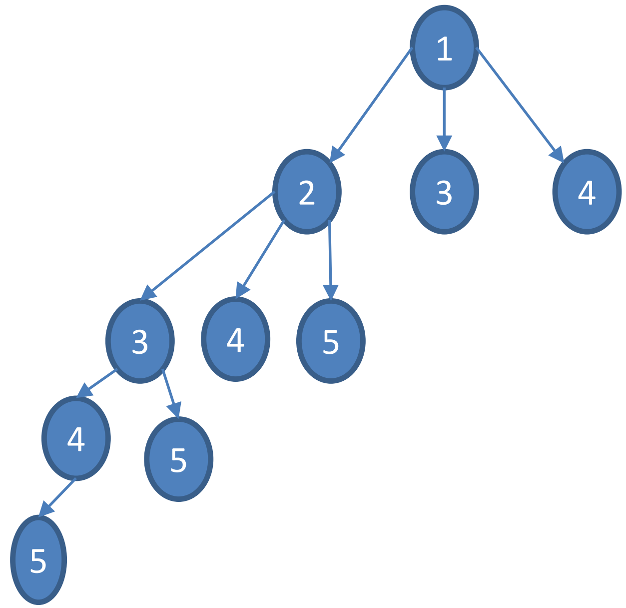
Uninformed Search – DFS & its Variant

Depth First Search (DFS)



DFS – Uninformed





(1)
(1 2) (1 3) (1 4)
(1 2 3) (1 2 4) (1 2 5) (1 3) (1 4)
(1 2 3 4) (1 2 3 5) (1 2 4) (1 2 5) (1 3) (1 4)
(1 2 3 4 5) (1 2 3 5) (1 2 4) (1 2 5) (1 3) (1 4)

C(1-2-3-4-5) = 70 + 50 + 100 + 50 =270
Expanded : 4
Generated : 10
Max Queue Length : 6

Depth First Search (DFS)

Completeness – Complete in finite state spaces because it will eventually expand every node

Optimal – Not Optimal as it would stop when the goal node is reached without evaluating if there is a better path

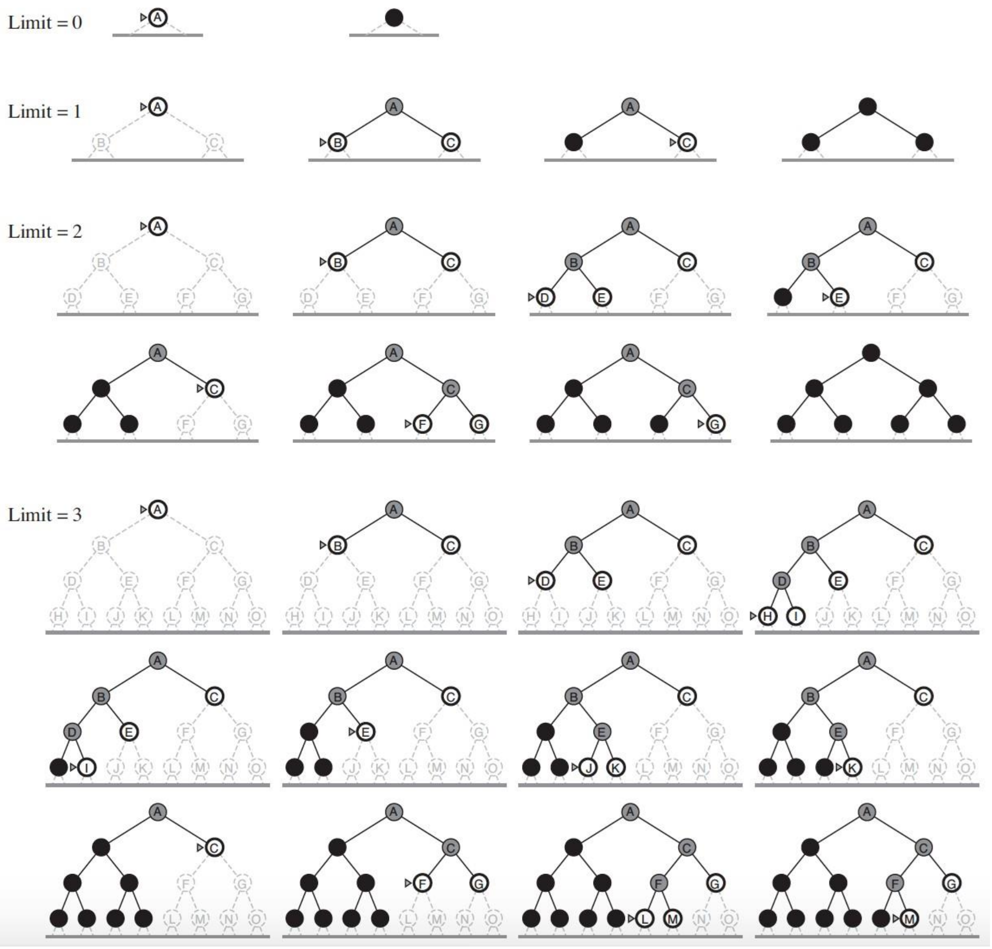
Time Complexity - $\mathcal{O}(b^m)$ where m = maximum depth of any node

- Can be much larger than the size of state space
- m can be much larger than d (shallowest goal)

Space Complexity – Needs to store only one path and unexpanded siblings.

- Any node expanded with all its children can be removed from memory
- Requires storage of only $\mathcal{O}(bm)$, b – branching factor, m - max depth

Iterative Deepening Depth First Search (IDS)



Iterative Deepening Depth First Search (IDS)

Run Depth Limited Search (DLS) by gradually increasing the limit l

- First with l=1, then l=2, l=3 and so on – until goal is found

Its is a combination of Depth First Search + Breadth First Search

Like DFS, memory requirement is a modest $\mathcal{O}(bd)$ where d is the depth of shallowest goal

Like BFS

- Complete when branching factor is finite
- Optimal when path cost is non decreasing function of depth

Iterative Deepening Depth First Search (IDS)

Time Complexity:

- Appears that IDS is generating a lot of nodes multiple times
- However, most of nodes are present in the lower levels which are not repeated often
- Generation of nodes
  - At level 1 - b nodes generated d times – (d)b
  - At level 2 – b^2 nodes generated d-1 times – $(d-1)b^2$
  - At level d – b^d nodes generated once – (1) b^d

- Time Complexity N(IDS) = $\mathcal{O}(b^d)$ same as BFS

IDS is the preferred uninformed search method when search space is large and depth is unknown

Application

Breadth First Search

- Finding path in a graph (many solutions)
- Finding the Bipartitions in a graph

Depth First Search

- Find the Connectedness in a graph
- Topological Sorting

Algorithm Tracing

Students must follow this in the exams for all the search algorithms in addition to the search tree constructions. The ordering of the Open Lists must be in consistent with the algorithm with a note on the justification of the order expected!

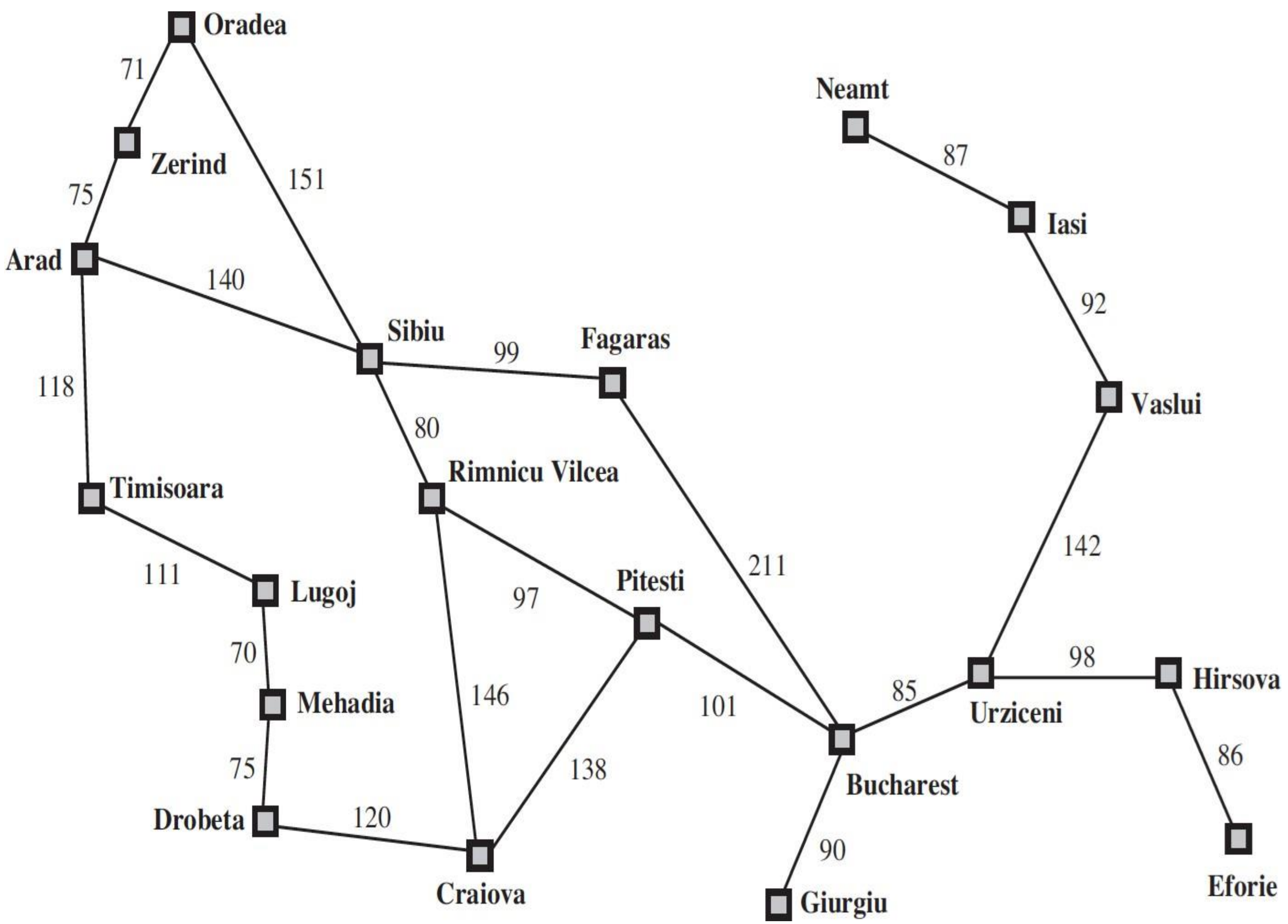
Iter	Open List / Frontiers / Fringes	Closed List	Goal Test
1.	(1)		Fail on (1)
2.	(1 3), (1 4), (1 2)	(1)	Fail on (1 3)

Terminologies – Learnt Today

- Nodes
- States
- Frontier | Fringes
- Search Strategy : LIFO | FIFO | Priority Queue
- Performance Metrics
  - Completeness
  - Optimality
  - Time Complexity
  - Space Complexity
- Algorithm Terminology

- d Depth of a node	- m – maximum
- b Branching factor	- C* - Optimal Cost
- n – nodes	- E – least Cost
- l – level of a node	- N –total node generated

Tree Search Vs Graph Search

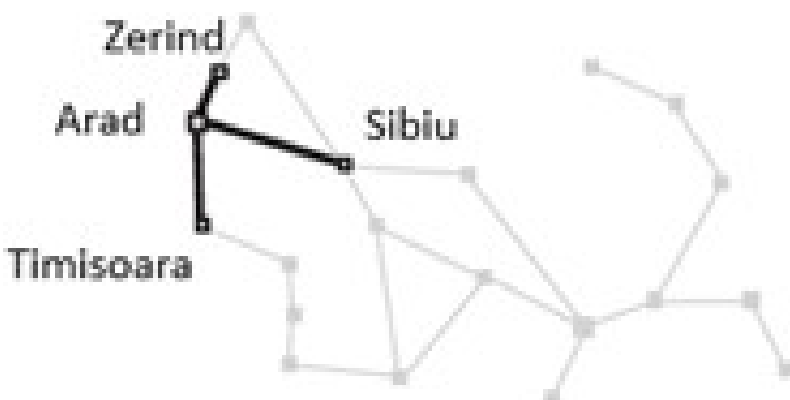


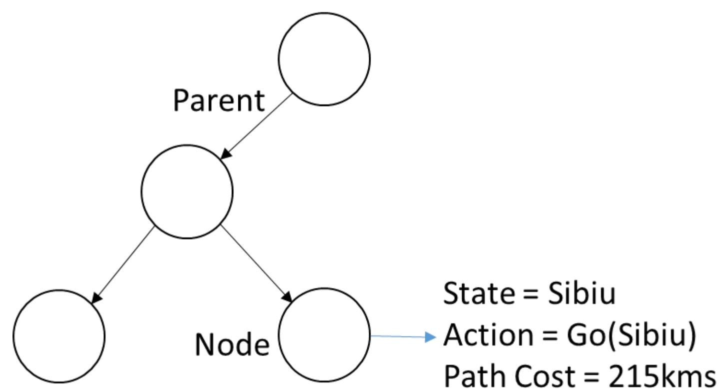
Oradea
71
Zerind
151
75
Arad
140
118
Sibiu
99
Fagaras
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
138
Drobeta
120
Craiova
Neamt
87
Iasi
92
Vaslui
142
Urziceni
98
Hirsova
85
86
Bucharest
90
Giurgiu
Eforie

Coding Aspects

For each node n of the tree,

- **n.STATE** : the state in the state space to which node corresponds
- **n.PARENT** : the node in the search tree that generated this node
- **n.ACTION** : the action that was applied to parent to generate the node
- **n.PATH-COST** : the cost, denoted by g(n), of the path from initial state to node





Algorithm Tracing

Students must follow this in the exams for all the search algorithms in addition to the search tree constructions. The ordering of the Open Lists must be in consistent with the algorithm with a note on the justification of the order expected!

Iter	Open List / Frontiers / Fringes	Goal Test
1.	(1)	Fail on (1)
2.	(1 3), (1 4), (1 2)	Fail on (1 3)

Tree Search Algorithms

```
function Tree-Search (problem, strategy) returns a solution, or failure
        initialize the search tree using the initial state of problems
        loop do
                if there are no candidate for expansion
                        then return failure
                choose: leaf node for expansion according to strategy
                if the node contains a goal state
                        then return the corresponding solution
                else
                        Expand the node
                        Add the resulting nodes to the search tree
        end
```

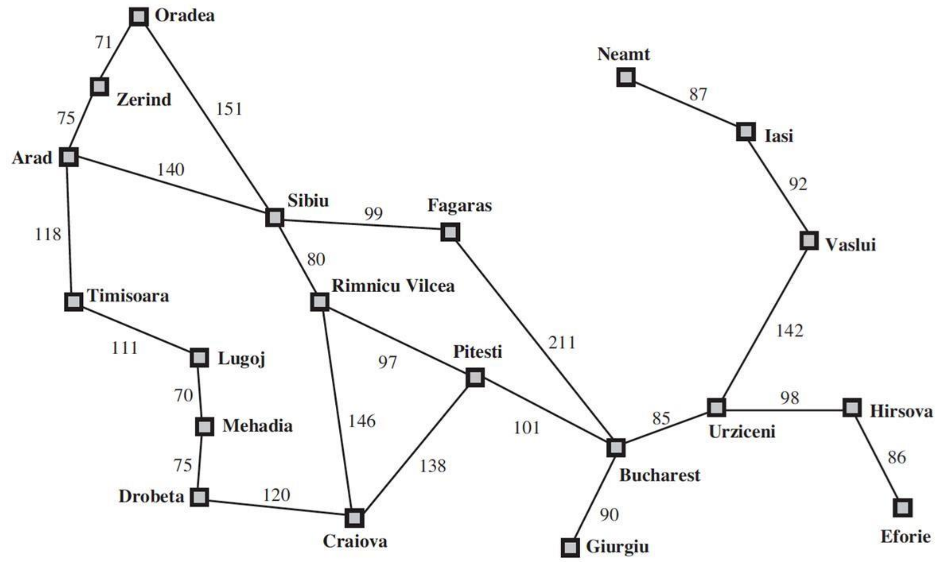
Tree Search Vs Graph Search Algorithms

Coding Aspects

Need:

Redundant Path Problem : More than one way to reach a state from another.

Infinite Loop Path Problem



Start : Arad

Goal : Craiova

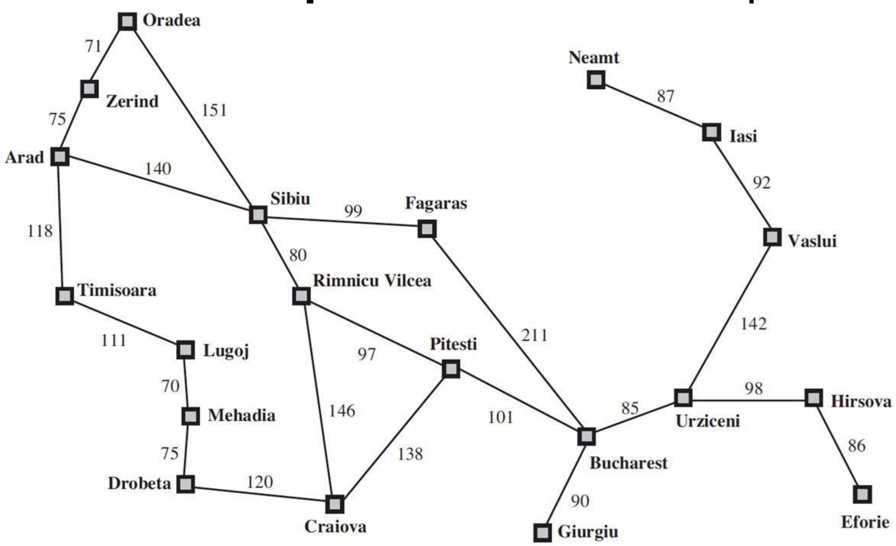
Tree Search Vs Graph Search Algorithms

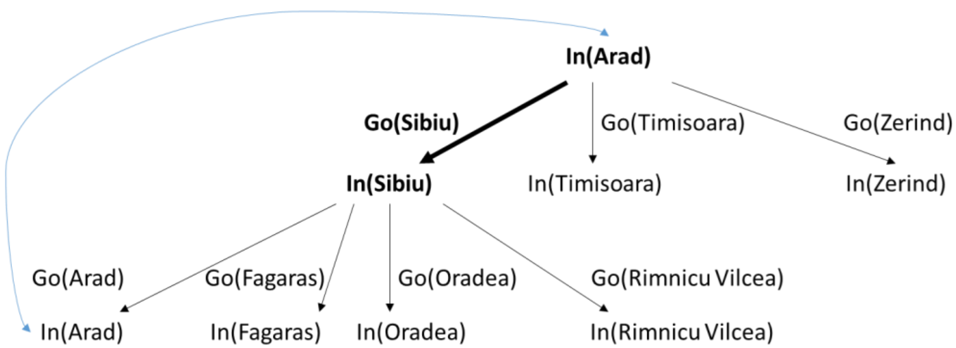
Coding Aspects

Need:

Redundant Path Problem

Infinite Loop Path Problem: Repeated State generated by looped path existence.





Start : Arad

Goal : Craiova

Algorithm Tracing

Students must follow this in the exams for all the search algorithms in addition to the search tree constructions. The ordering of the Open Lists must be in consistent with the algorithm with a note on the justification of the order expected!

Iter	Open List / Frontiers / Fringes	**Closed List**	Goal Test
1.	(1)		Fail on (1)
2.	(1 3), (1 4), (1 2)	(1)	Fail on (1 3)

Search

Coding Aspects

For each node n of the tree,

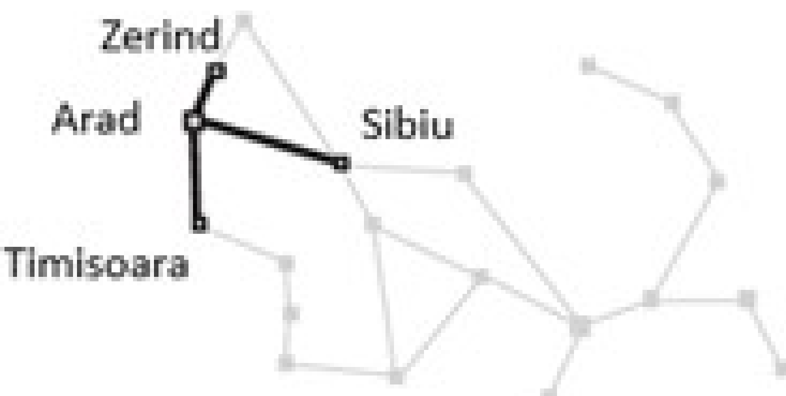
n.STATE : the state in the state space to which node corresponds

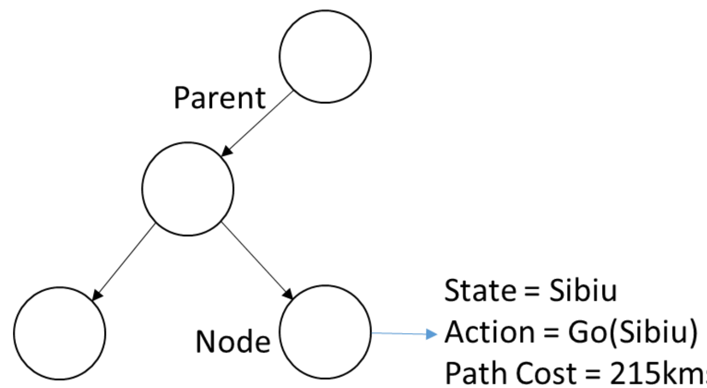
n.PARENT : the node in the search tree that generated this node

n.ACTION : the action that was applied to parent to generate the node

n.PATH-COST : the cost, denoted by g(n), of the path from initial state to node

n.VISITED **:** the boolean indicating if the node is already visited and tested **(or)** a global SET of visited nodes



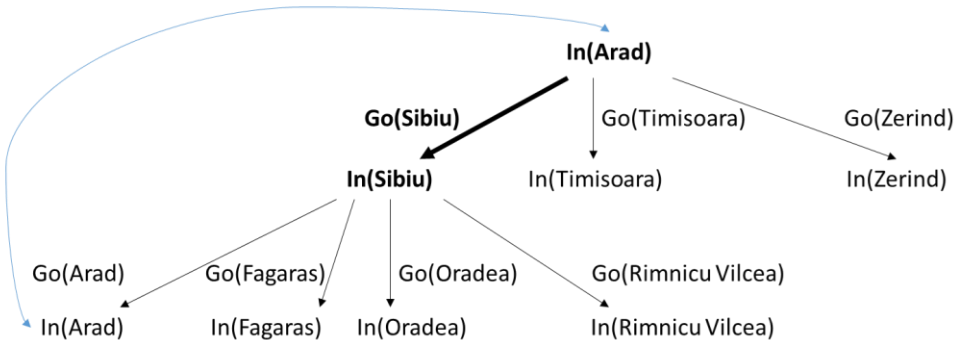


Tree Search Vs Graph Search Algorithms

Coding Aspects

Graph-Search Algorithm

Augments the Tree-Search algorithm to solve redundancy by keeping track of states that are already visited called as **Explored Set**. **Only one copy of each state is maintained/stored.**



Graph Search Algorithms

function **Graph-Search** (problem, fringe) returns a solution, or failure

initialize the search space using the initial state of problems memory to store the visited fringe

closed ← an empty set

fringe ← Insert(Make-Node(Initial-State[problem]), fringe)

loop do

if fringe is empty

then return failure

node ← Remove-Front(fringe)

if the node contains a goal state

then return the corresponding solution

else

if the node is not in closed ie., not visited yet

Add the node to the closed set

Expand all the fringe of the node

Add all expanded sorted successors into the fringe

end

Required Reading: AIMA - Chapter #3: 3.1, 3.2, 3.3, 3.4

Next Class Plan :
Informed Search : GFBS & A*
Heuristic Design

Thank You for all your Attention

Note : Some of the slides are adopted from AIMA TB materials