Artificial & Computational Intelligence

AIML CZG557

M2 : Problem Solving Agent using Search

Dr. Sudheer Reddy

BITS Pilani
Pilani Campus

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

Problem Solving Agents

Goal based decision making agents finds sequence of actions that leads to the desirable state.

Phases of Solution Search by PSA

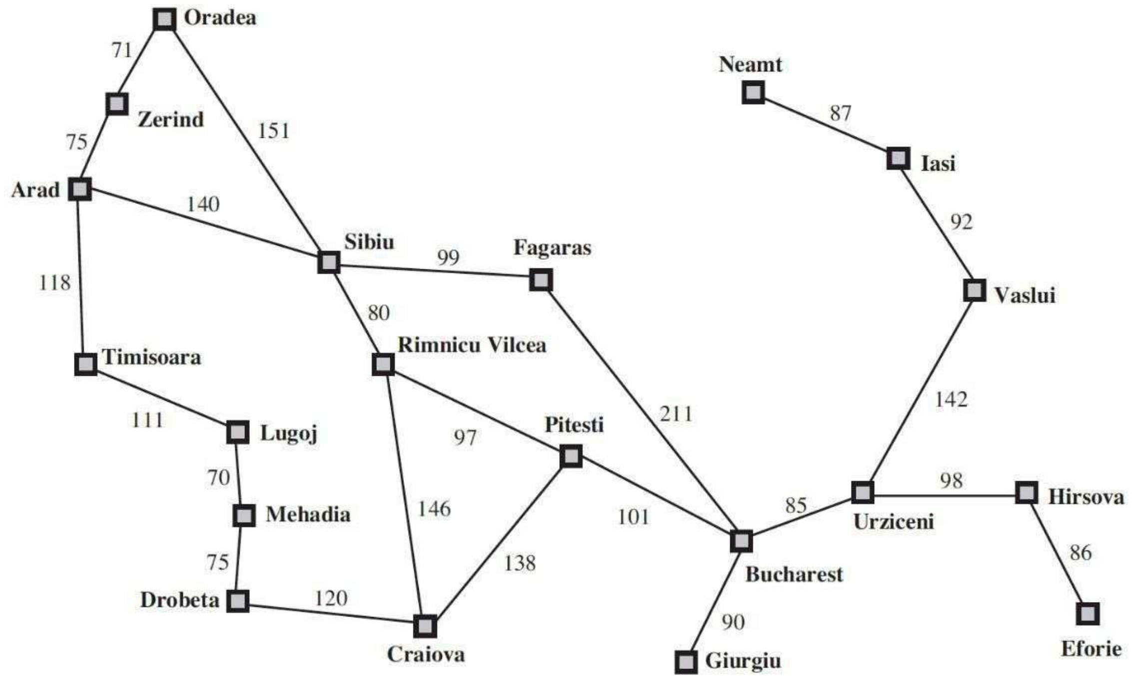
Goal Formulation

Problem Formulation

Search Phase

Execution Phase

Optimizes the Objective (Local | Global) Limits the Actions



Problem Solving Agents

Phases of Solution Search by PSA

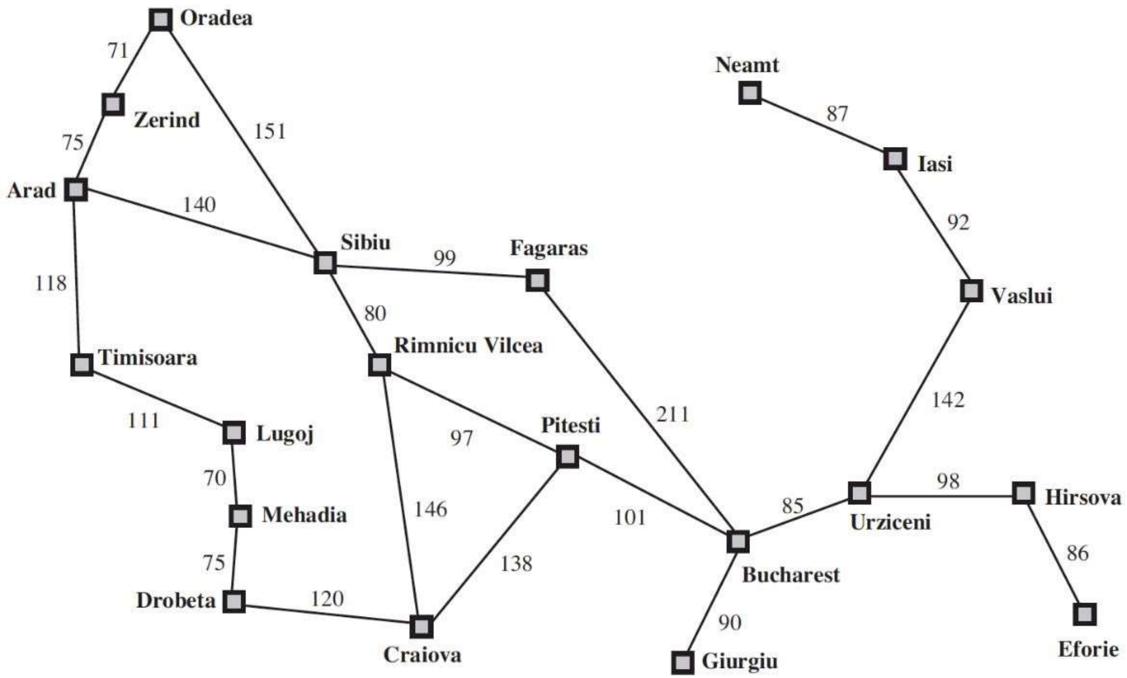
Goal Formulation

Problem Formulation

Search Phase

Execution Phase

State Space Creations [in the path of Goal]
Lists the Actions



Problem Solving Agents

Goal Formulation

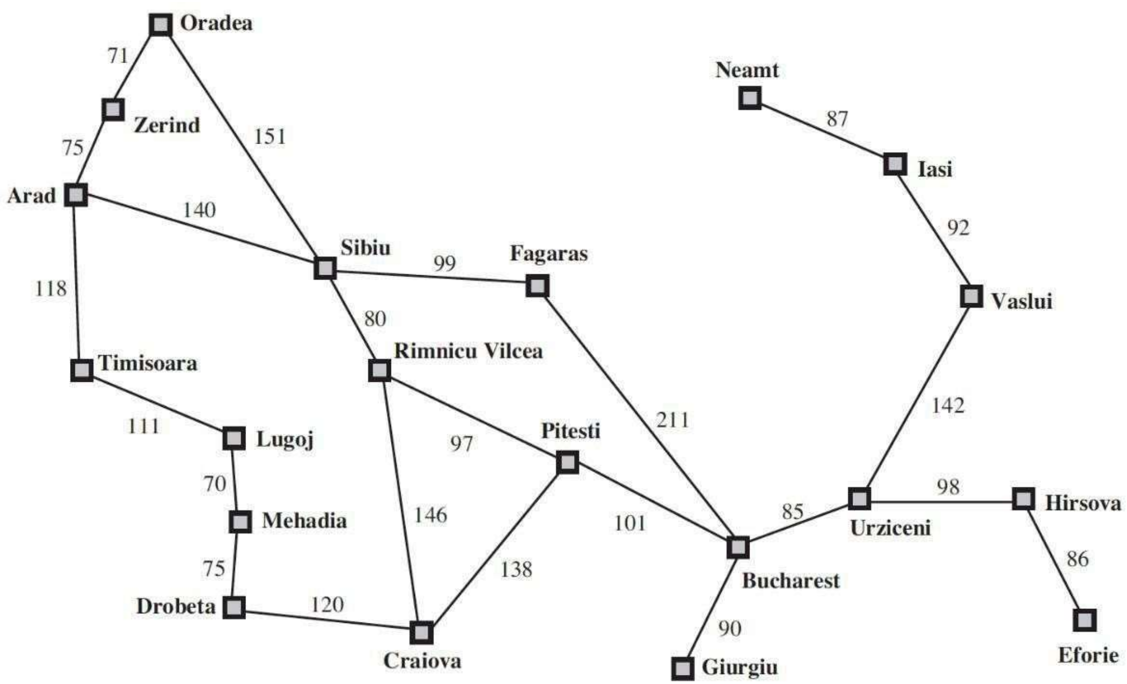
Problem Formulation

Search Phase

Execution Phase

Phases of Solution Search by PSA

Assumptions – Environment :
Static
Observable Discrete
Deterministic



Problem Solving Agents

Phases of Solution Search

Goal Formulation

↓

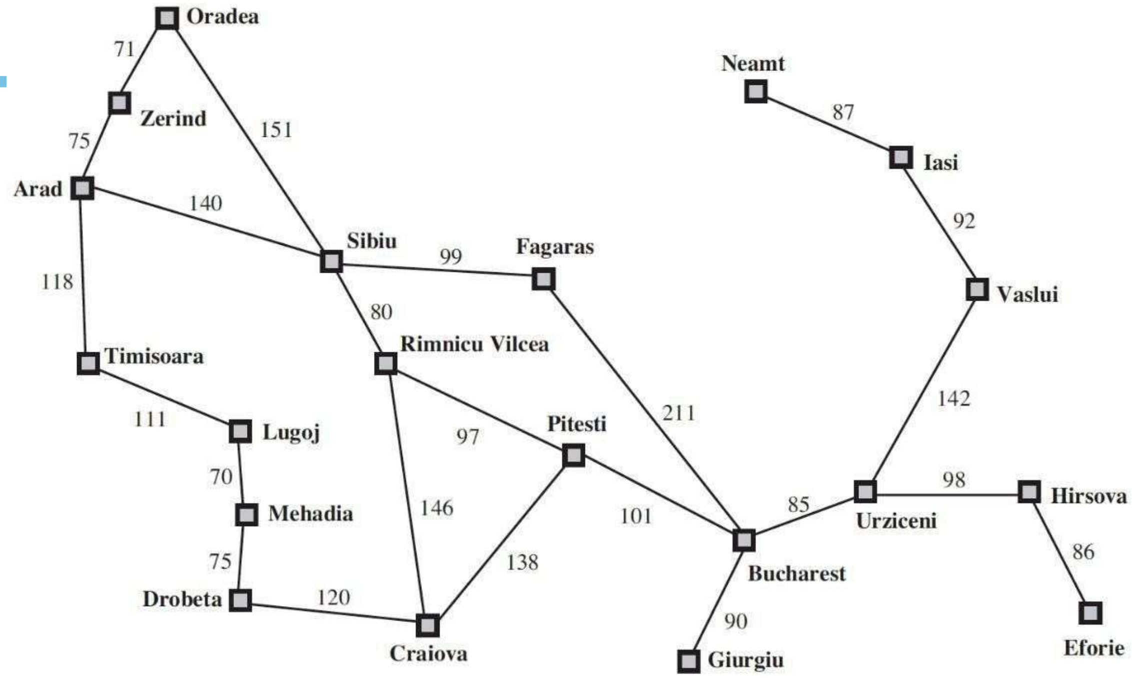
Problem Formulation

↓

Search Phase

↓

Execution Phase



Examine all sequence
Choose best |
Optimal

Problem Solving Agents – Problem Formulation

Abstraction Representation

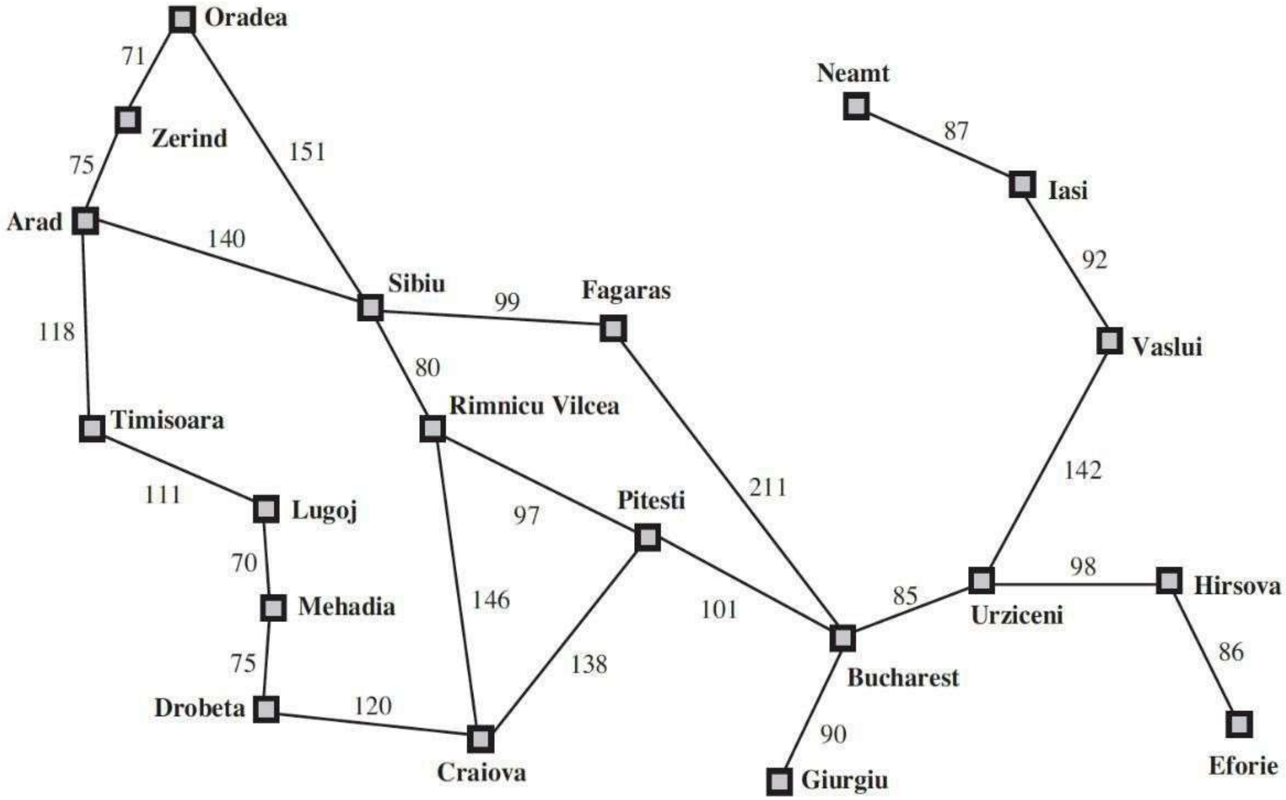
Decide what actions under states to take to achieve a goal

- 5 Components
  - Initial State
  - Possible Action | Operators
  - Successor Function |Transition Model
  - Goal Test
  - Path Cost

A function that assigns a numeric cost to each path. A path is a series of actions. Each action is given a cost depending on the problem.

Solution = Path Cost Function + Optimal Solution

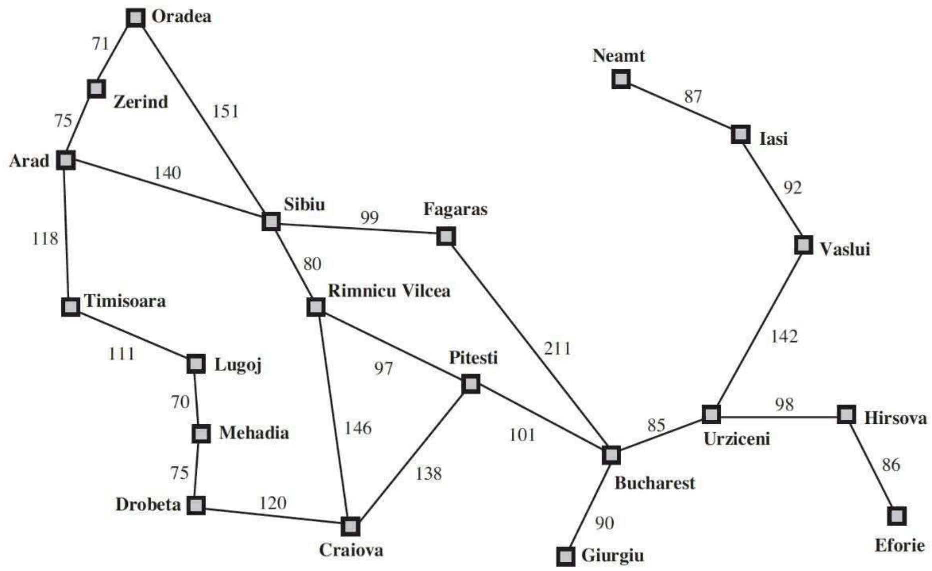
Problem Solving Agents – Problem Formulation: BookExample



Initial State –E.g., *In(Arad)*
Possible Actions – *ACTIONS(s)* □ *{Go(Sibiu), Go(Timisoara), Go(Zerind)}*
Transition Model – *RESULT(In(Arad), Go(Sibiu)) = In(Sibiu)*
Goal Test – *IsGoal(In(Bucharest)) = Yes*
Path Cost – *cost(In(Arad), go(Sibiu)) = 140 kms*

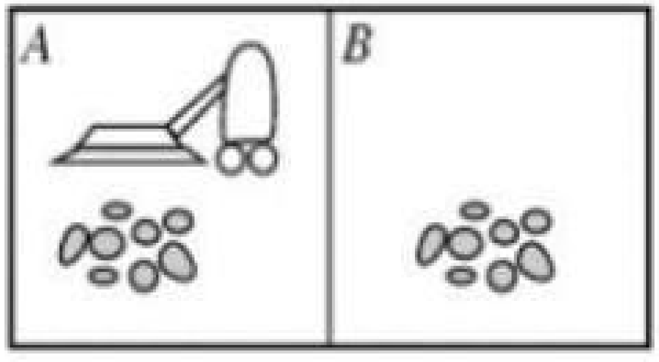
Example Problem Formulation

	Travelling Problem
Initial State	Based on the problem
Possible Actions	Take a flight | Train | Shop
Transition Model/ Successor Function	[A, Go(A->S)] = [S]
Goal Test	Is current = B (destination)
Path Cost	Cost + Time + Quality



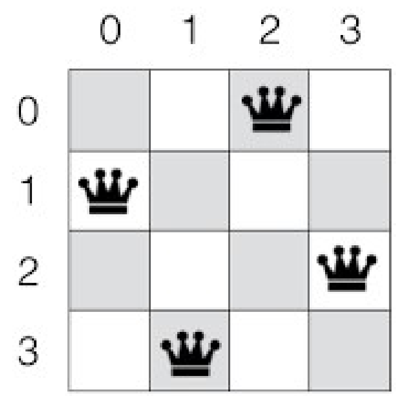
Example Problem Formulation

	Vacuum World
Initial State	Any
Possible Actions	[Move Left, Move Right, Suck, NoOps]
Transition Model/ Successor Function	[A, ML] = [B , Dirty] [A, ML] = [B, Clean]
Goal Test	Is all room clean? [A, Clean] [B, Clean]
Path Cost	No of steps in path



Example Problem Formulation

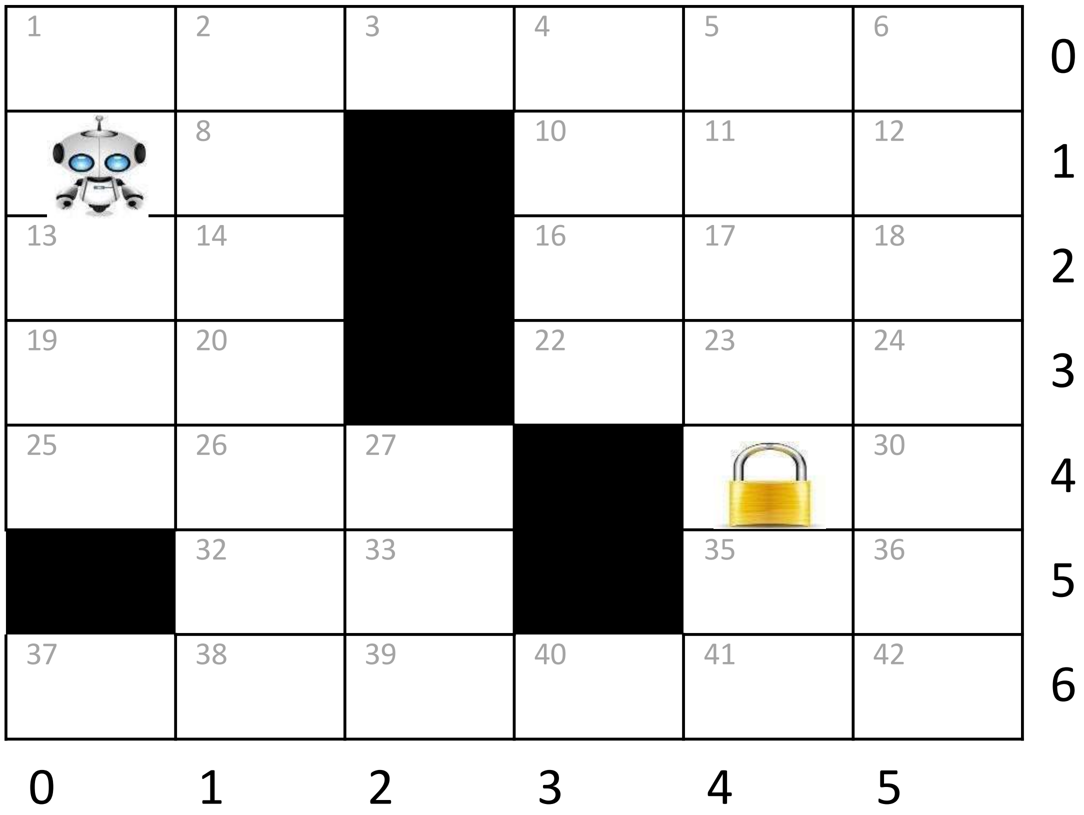
	N-Queen
Initial State	Empty | Partial | Full
Possible Actions	
Transition Model/ Successor Function	
Goal Test	
Path Cost	



board[r][c]

Path finding Robot

Successor Function Design

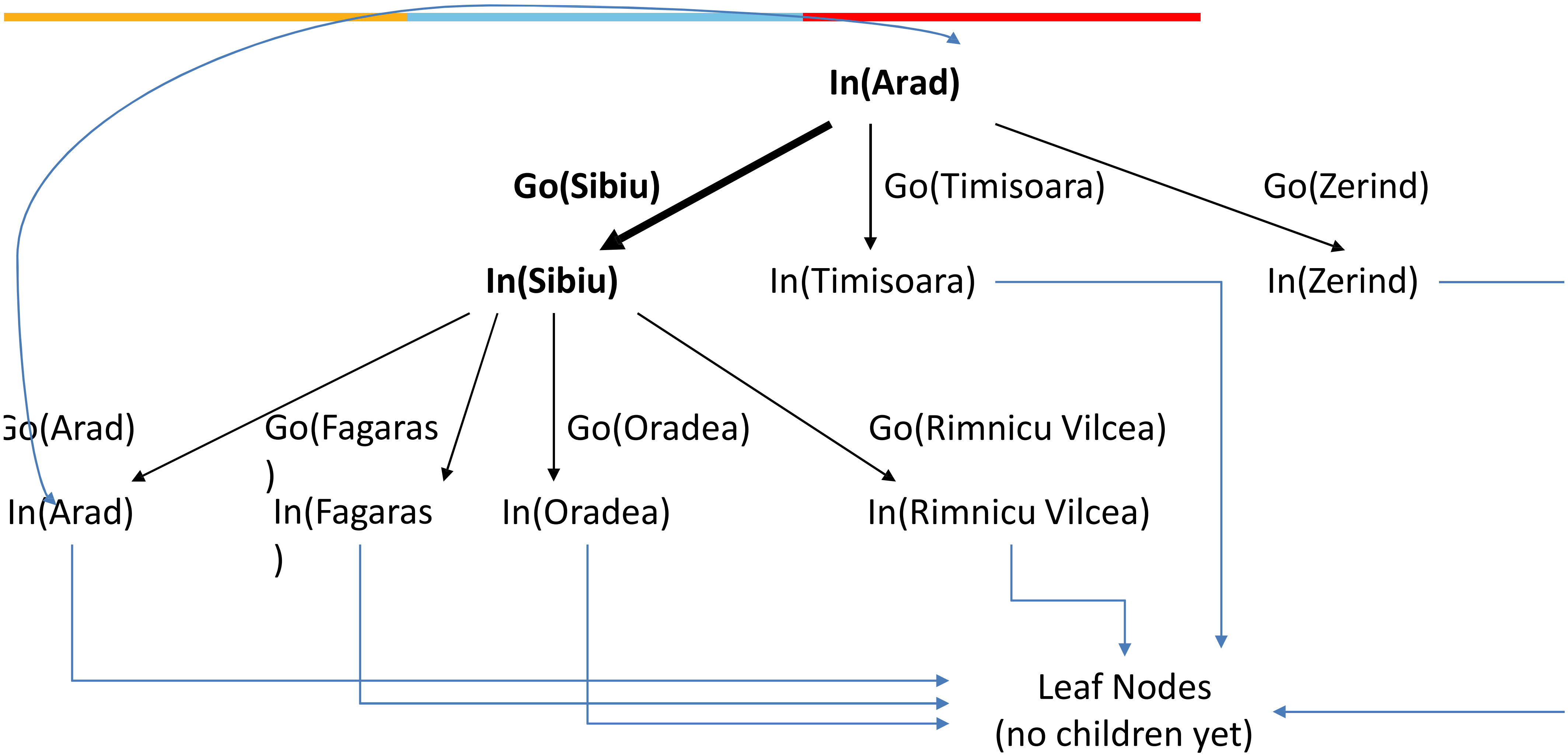


N-W-E-S

Searching for Solutions

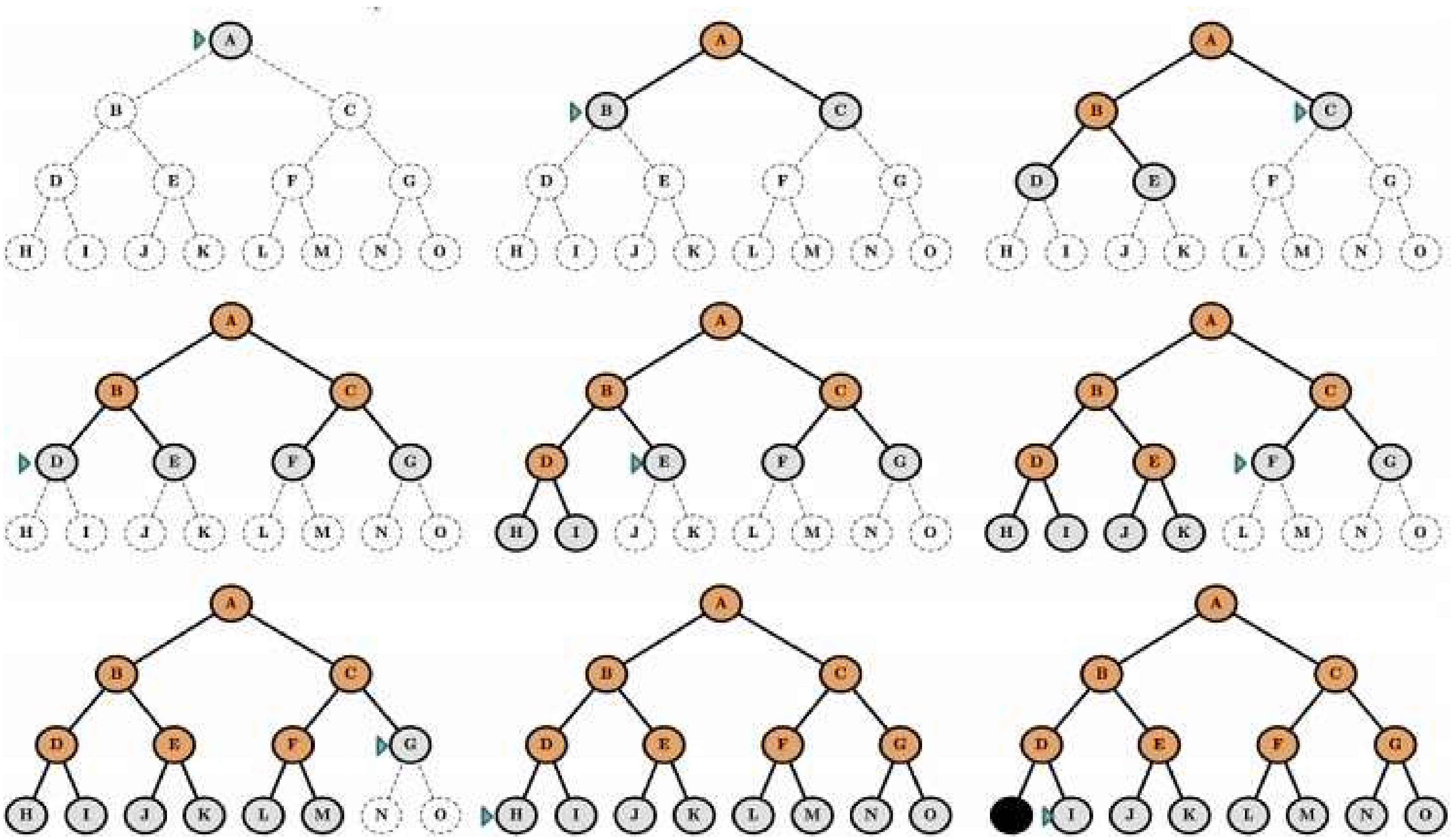
Choosing the current state, testing possible successor function, expanding current state to generate new state is called Traversal. Choice of which state to expand – Search Strategy

- Search Strategy (under certainty)
  - **Uninformed**
    - BFS, DFS, UCS
    - IDS, DLS
    - Bi-Directional
  - **Informed**
    - Best First Search
    - A*
    - AO*

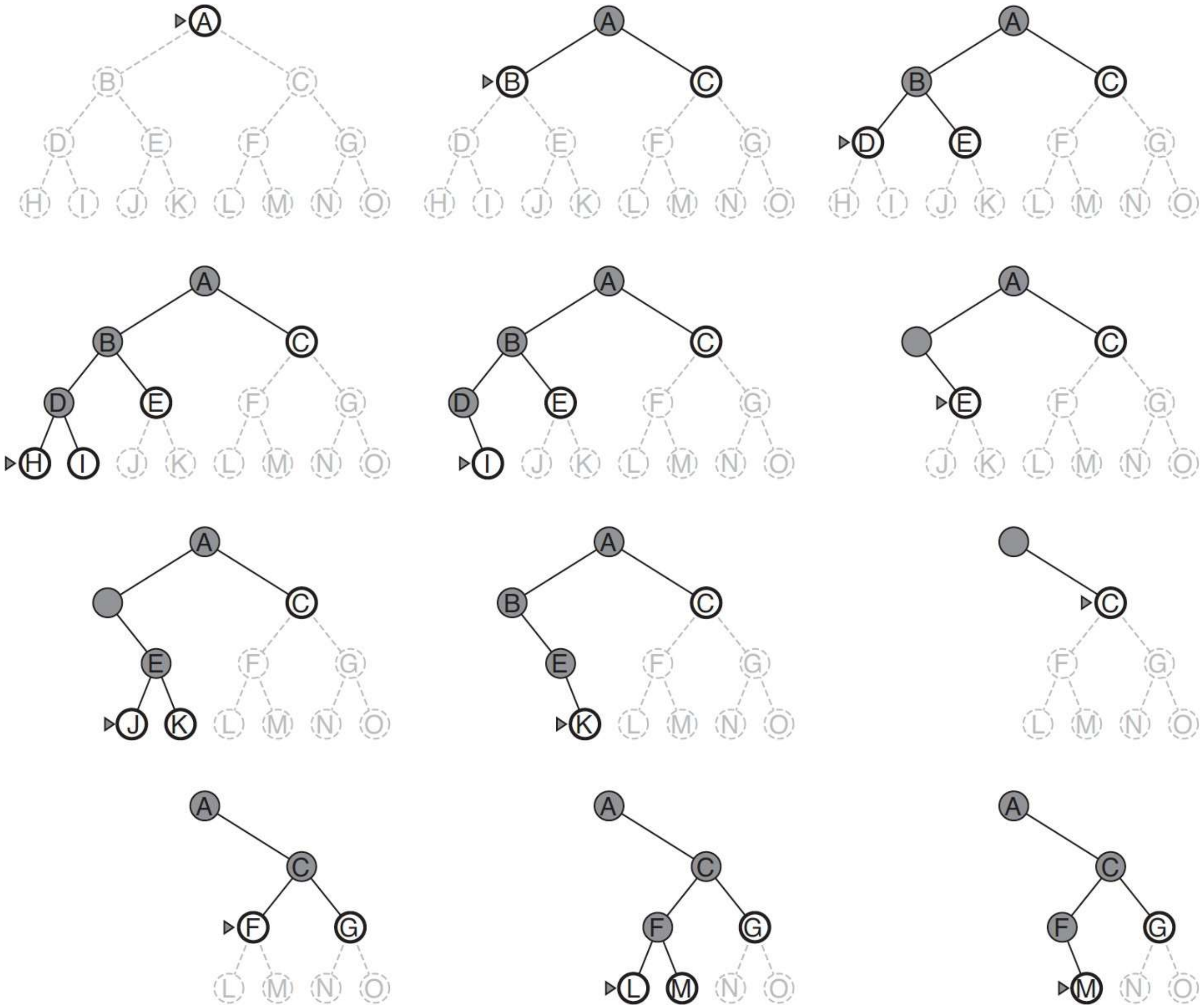

In(Arad)
Go(Sibiu)
Go(Timisoara)
Go(Zerind)
In(Sibiu)
In(Timisoara)
In(Zerind)
Go(Arad)
Go(Fagaras)
Go(Oradea)
Go(Rimnicu Vilcea)
In(Arad)
In(Fagaras)
In(Oradea)
In(Rimnicu Vilcea)
Leaf Nodes
(no children yet)

Uninformed Search
Overview

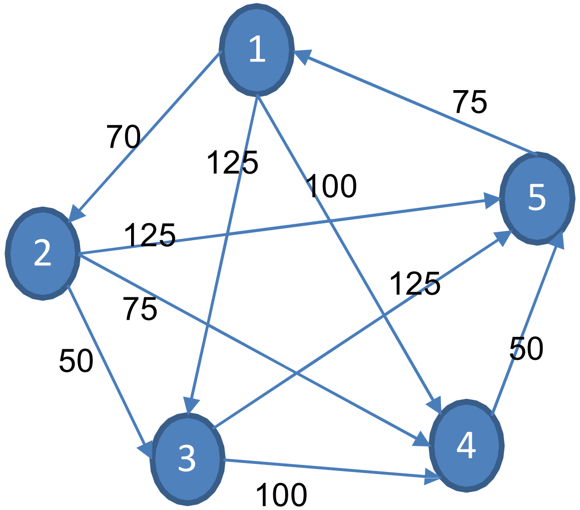
Breadth First Search (BFS)

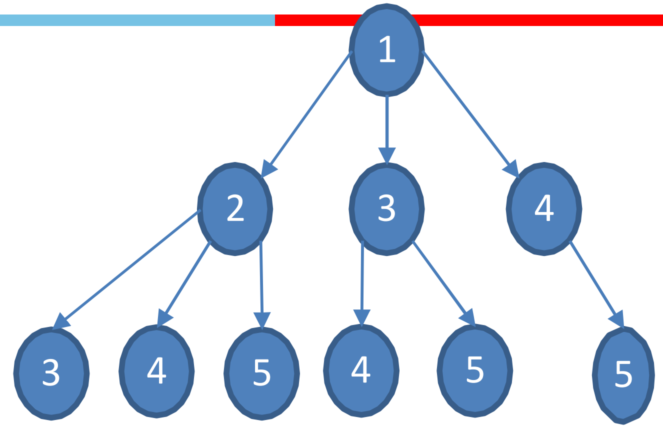


Depth First Search (DFS)



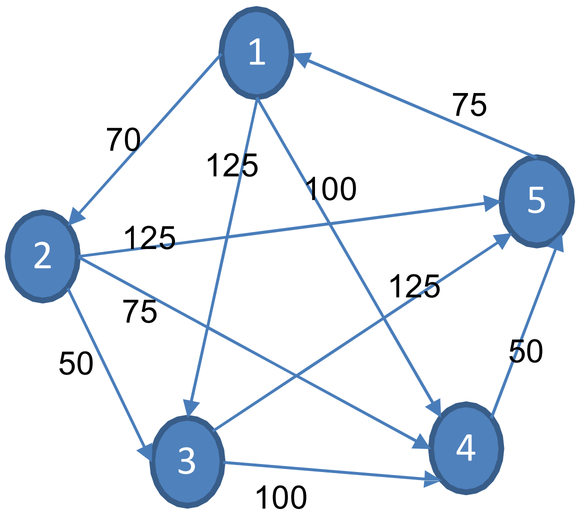
Search Tree – Sample Generation

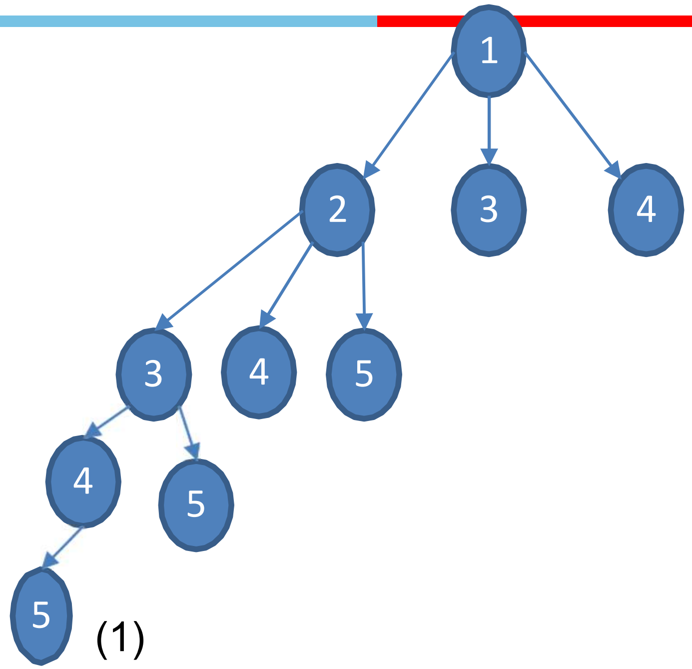




Each NODE in in the search tree denotes an entire PATH through the state space graph.

Search Algorithm – Uninformed Example - 1

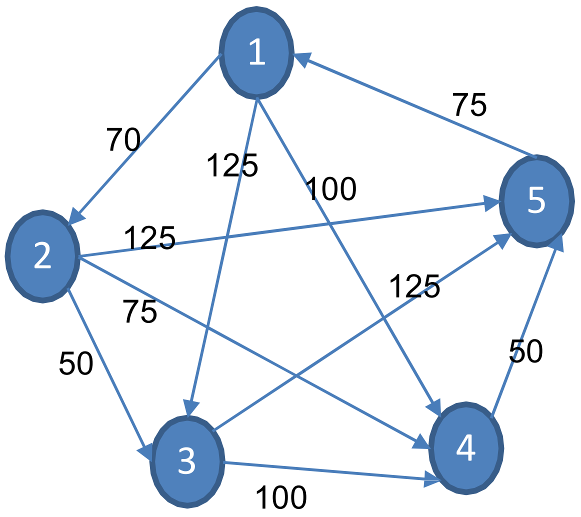


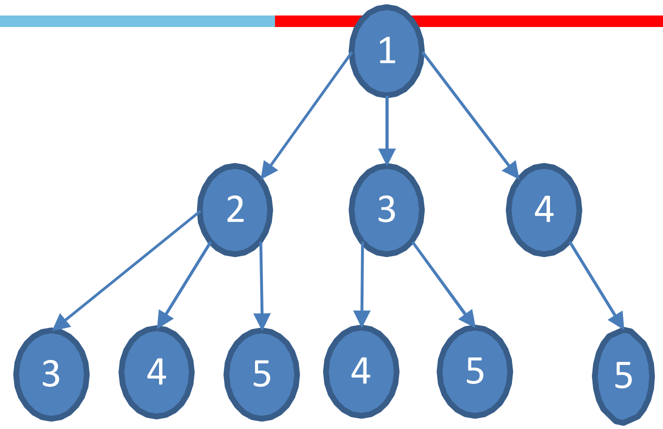


(1)
(1 2) (1 3) (1 4)
(1 2 3) (1 2 4) (1 2 5) (1 3) (1 4)
(1 2 3 4) (1 2 3 5) (1 2 4) (1 2 5) (1 3) (1 4)
(1 2 3 4 5) (1 2 3 5) (1 2 4) (1 2 5) (1 3) (1 4)

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

Search Algorithm – Uninformed Example - 2



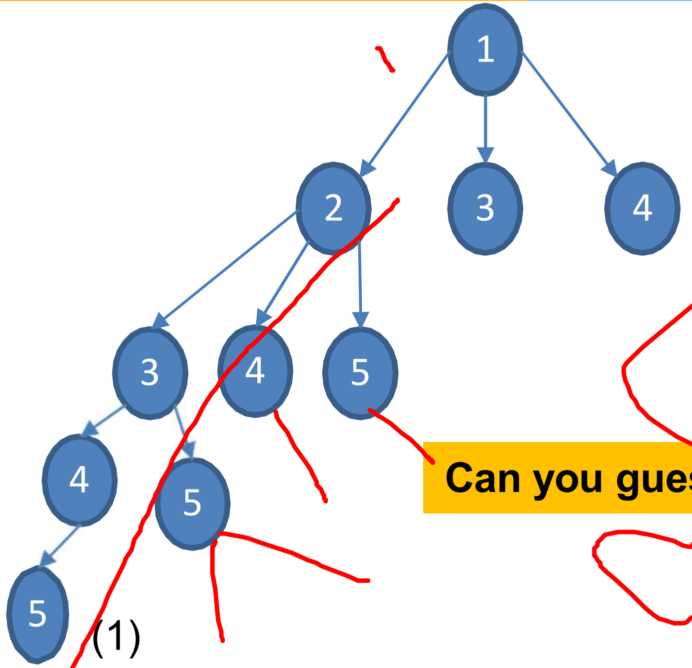


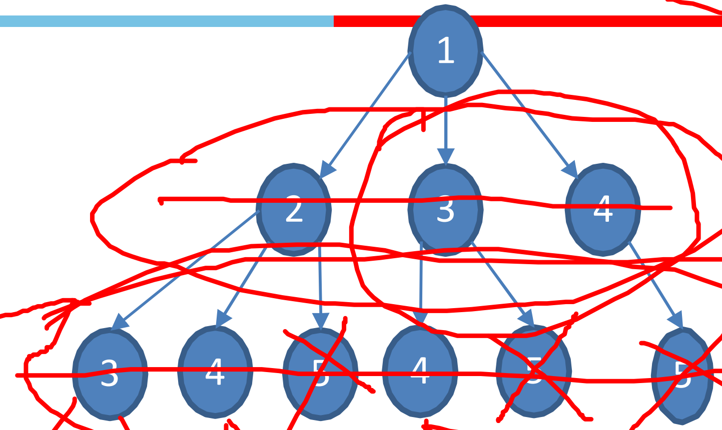
(1)
(1 2) (1 3) (1 4)
TEST FAILED

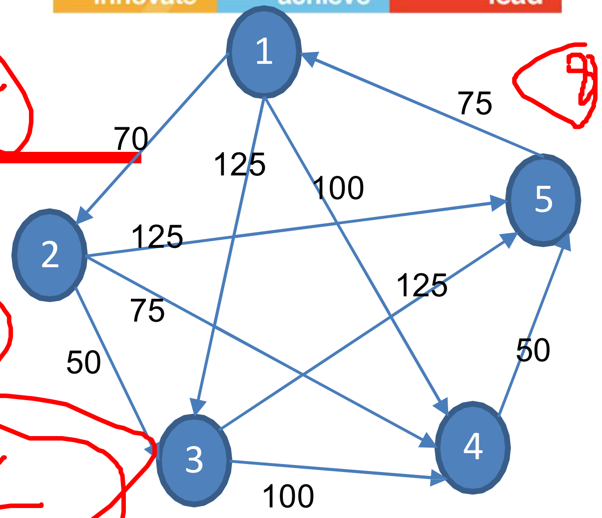
(1 3) (1 4) (1 2 3) (1 2 4) (1 2 5)
(1 2 3) (1 2 4) (1 2 5) (1 3 4) (1 3 5) (1 4 5)
TEST PASSED

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

Search Algorithm – Uninformed Example - 2







Can you guess which algorithm are these ?

(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)

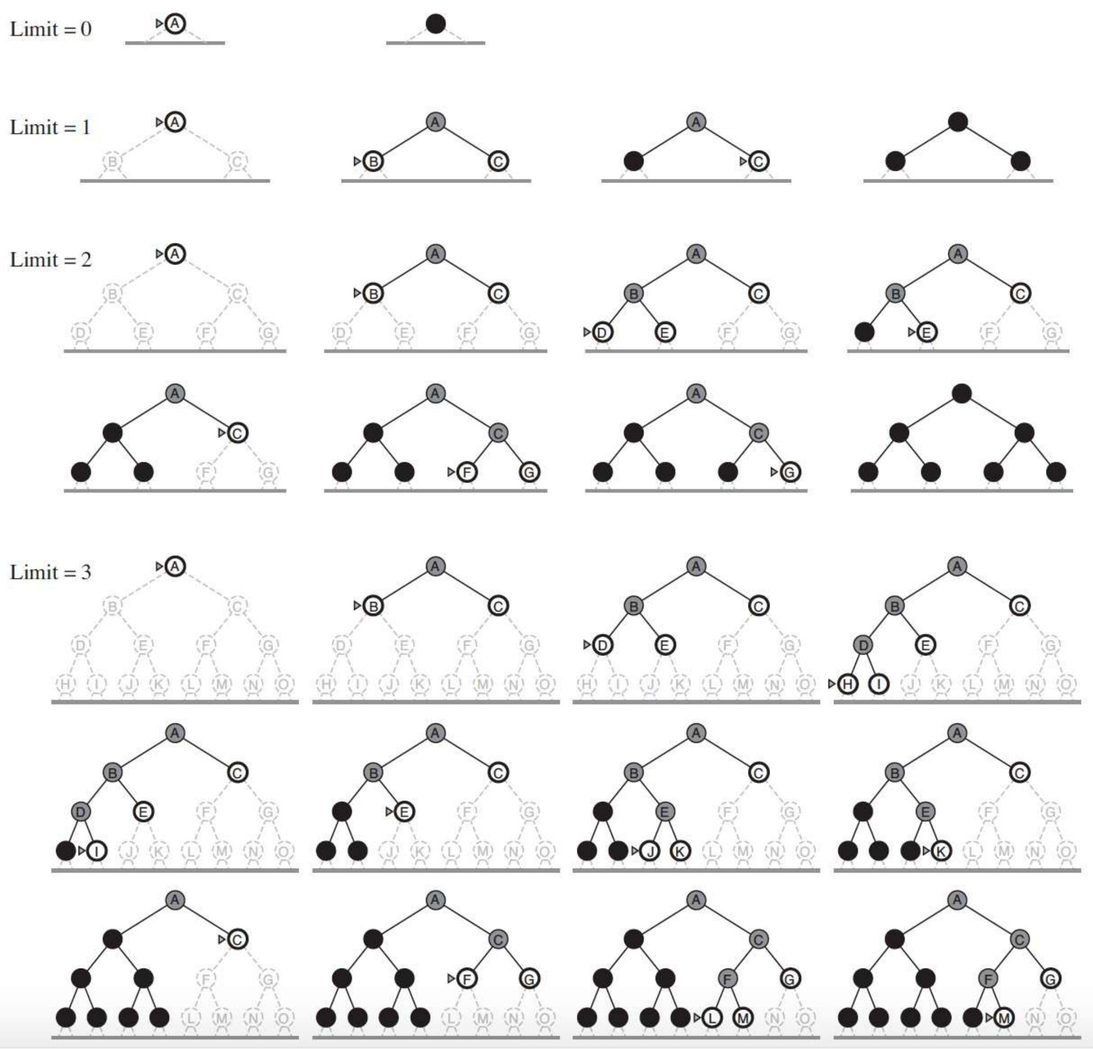
C(1-2-3-4-5) = 70 + 50 + 100 + 50 =270
Expanded : 4
Generated : 10
Max Queue Length : 6

(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

Iterative Deepening Depth First Search (IDS)

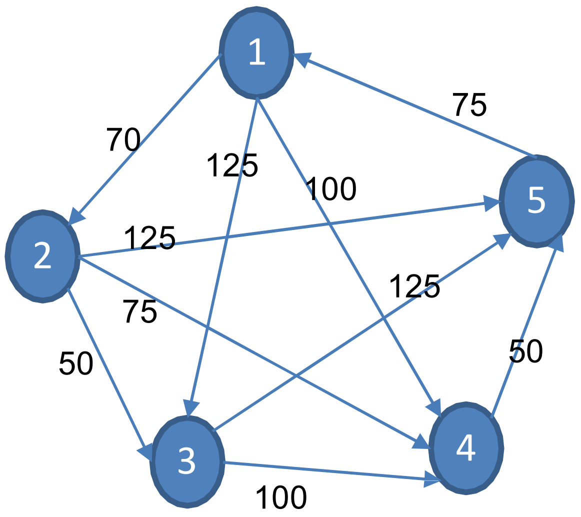


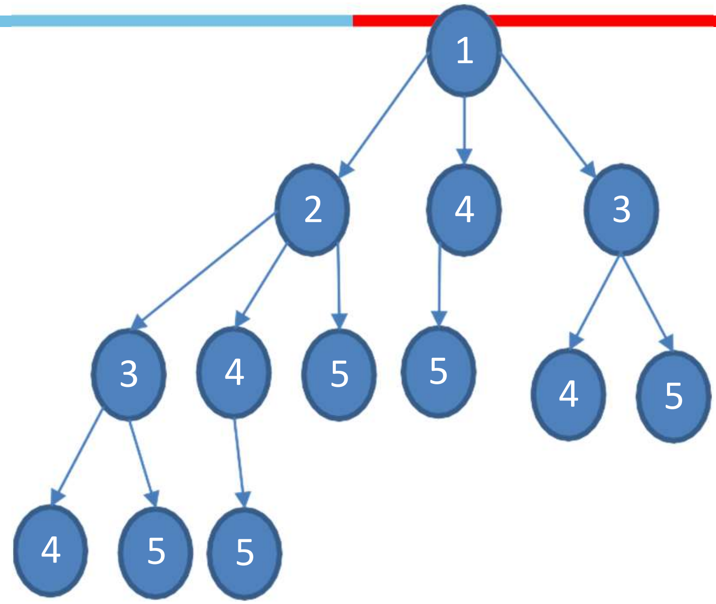
Algorithm Tracing

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)

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

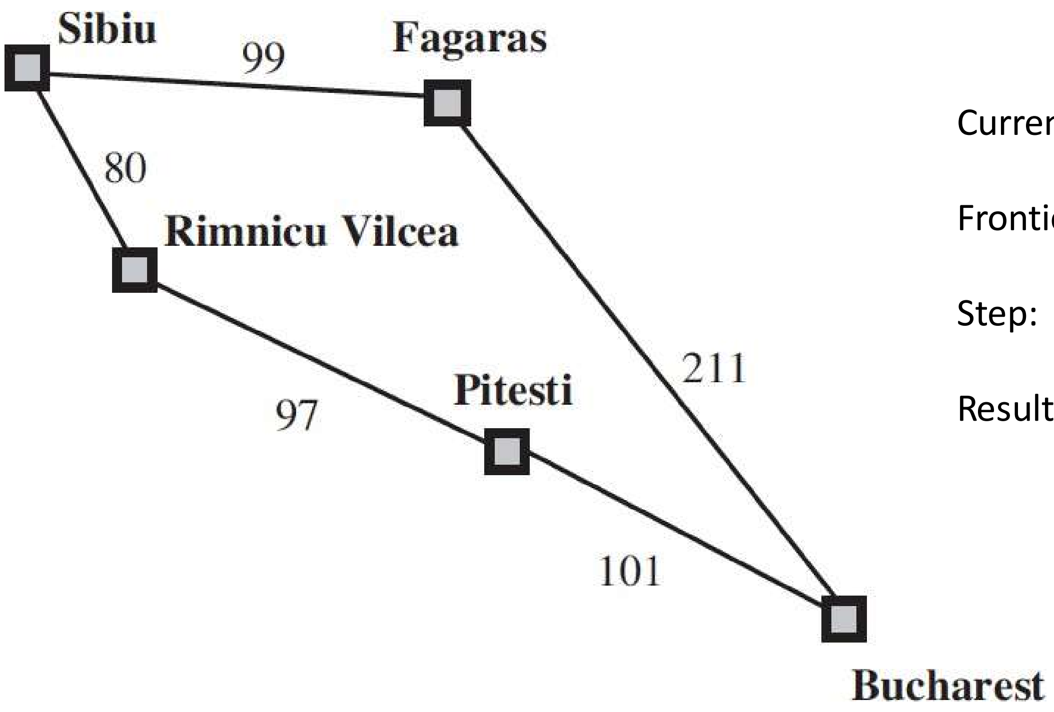
Instead of expanding the shallowest node, Uniform-Cost search expands the node n with the lowest path cost g(n)

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

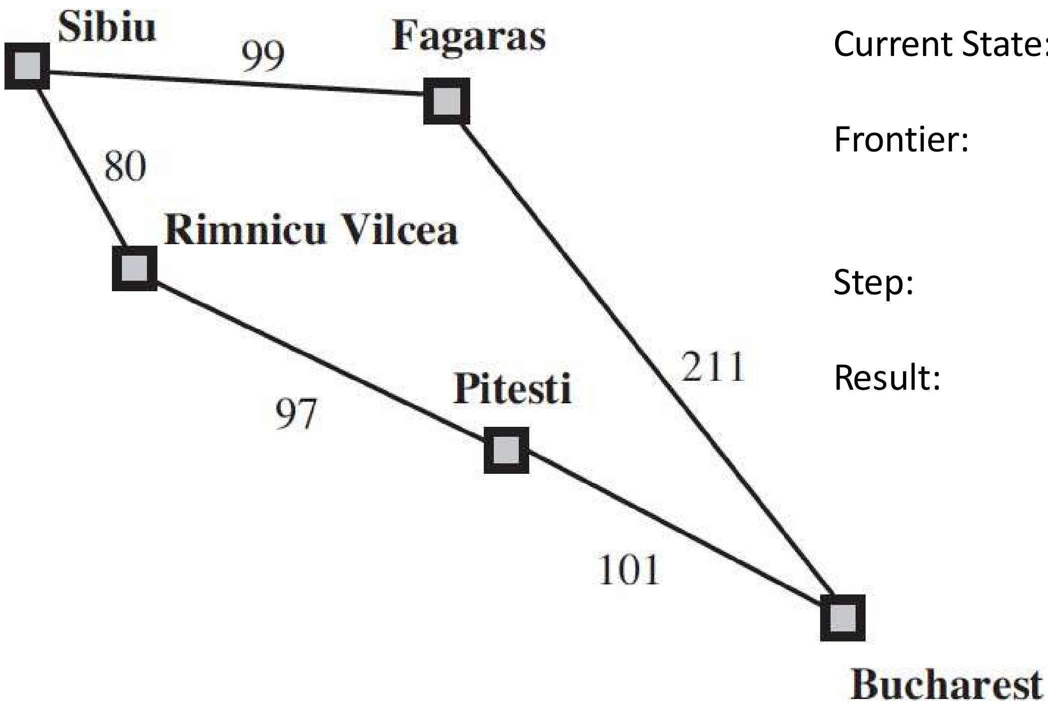
Frontier: []

Step: Expand Sibiu

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

Initial State: Sibiu
Goal State: Bucharest

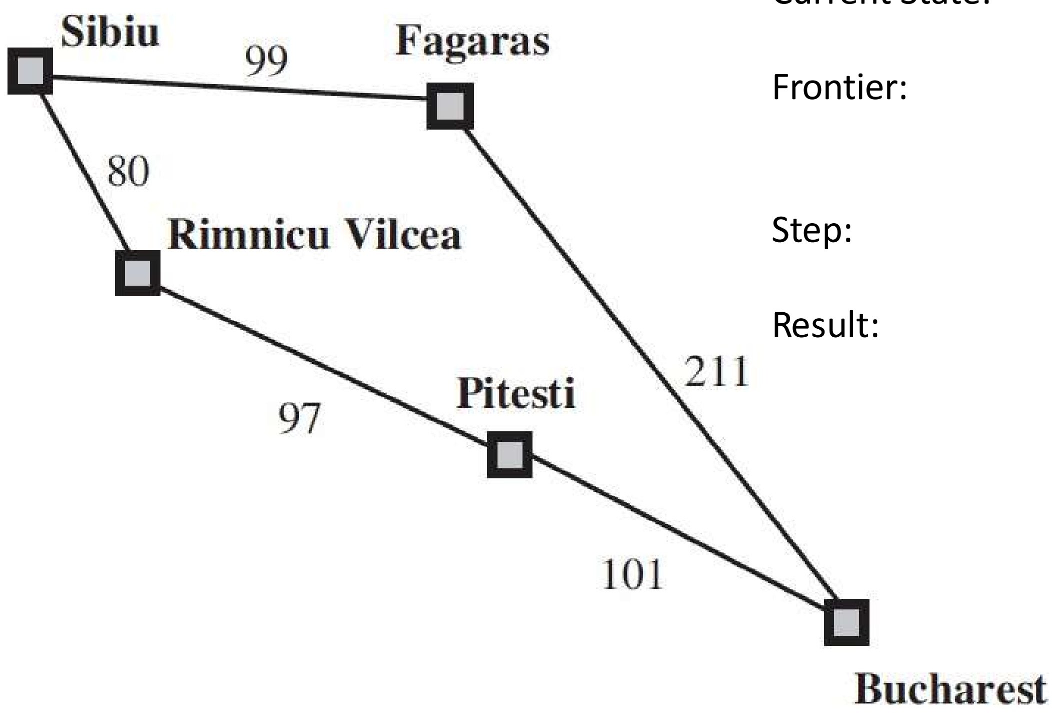
Uniform Cost Search



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)

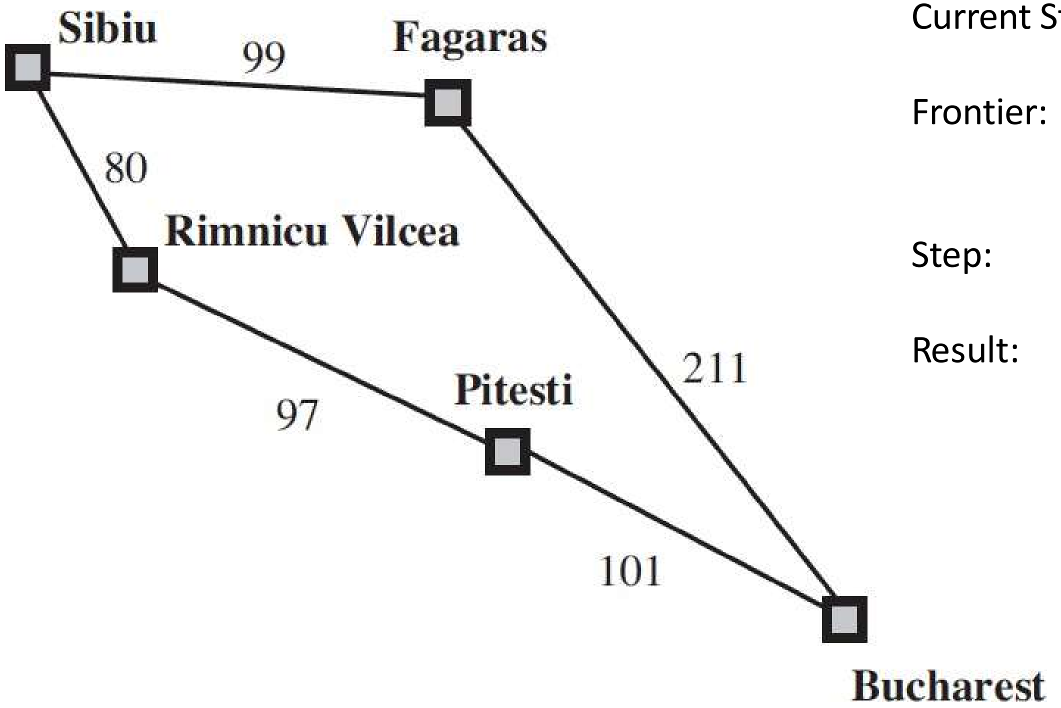
Frontier: [(“Fagaras”, 99)
(“Pitesti”, 177)]

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)

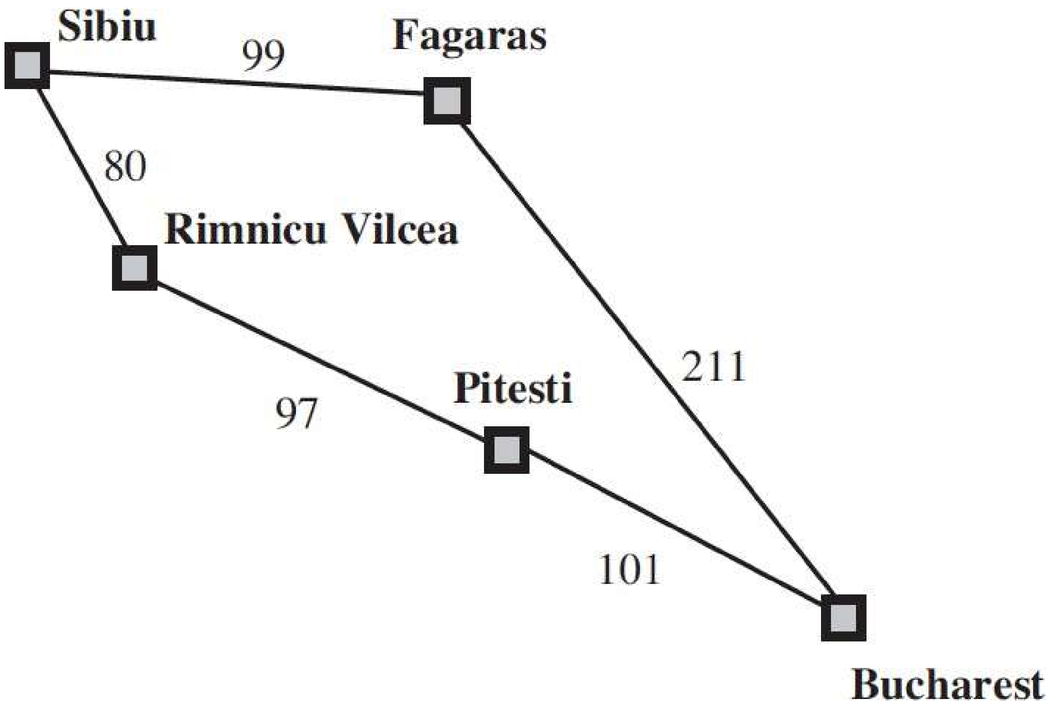
Frontier: [(“Pitesti”, 177)
(“Bucharest”, 310)]

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

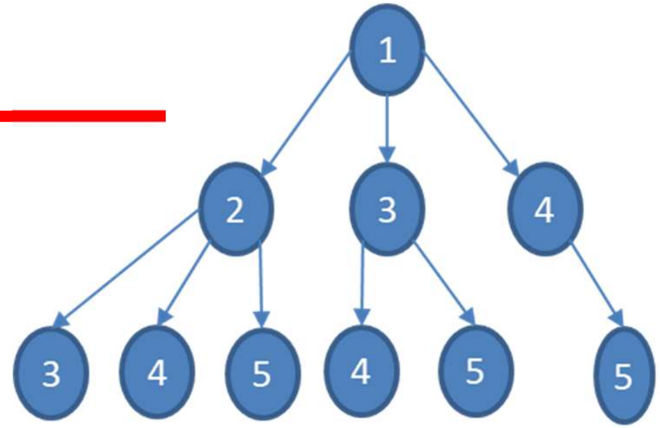
Uniform Cost Search



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

Sample Evaluation of the Algorithm



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

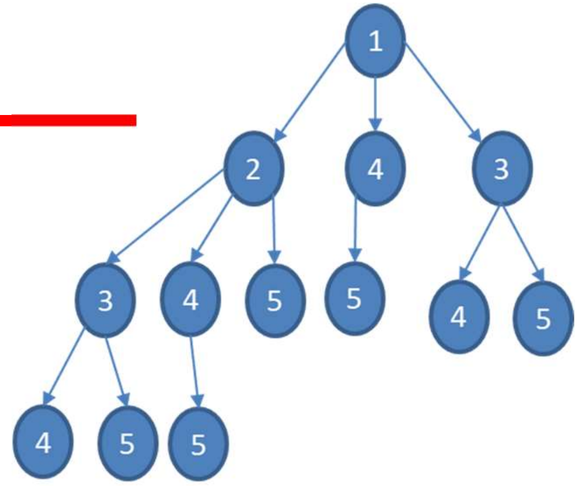
Time Complexity – $G(b^d)$ b - branching factor, d – depth

- 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 $G(b^{d+1})$

Space Complexity – $G(b^d)$

- $G(b^{d-1})$ in explored set
- $G(b^d)$ in frontier set

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 = $G(b^{1+\frac{C^*}{\epsilon}})$,

– When all action costs are equal → $G(b^{d+1})$, the BFS would perform better

  - As Goal test is applied during expansion, Uniform Cost search would do extra work

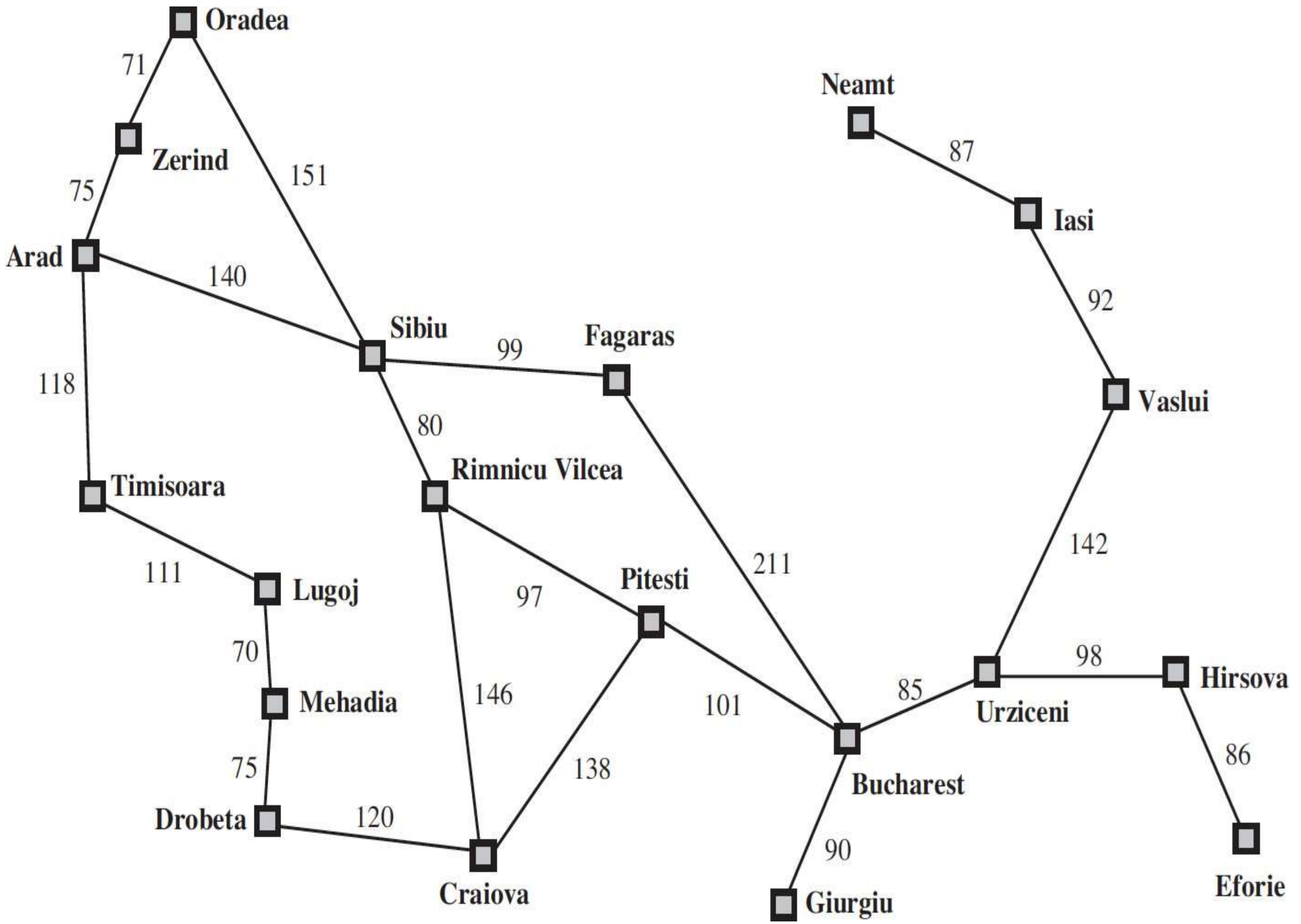
Terminologies – Learnt Today

- 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

innovate achieve lead

Tree Search Vs Graph Search


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

Coding Aspects

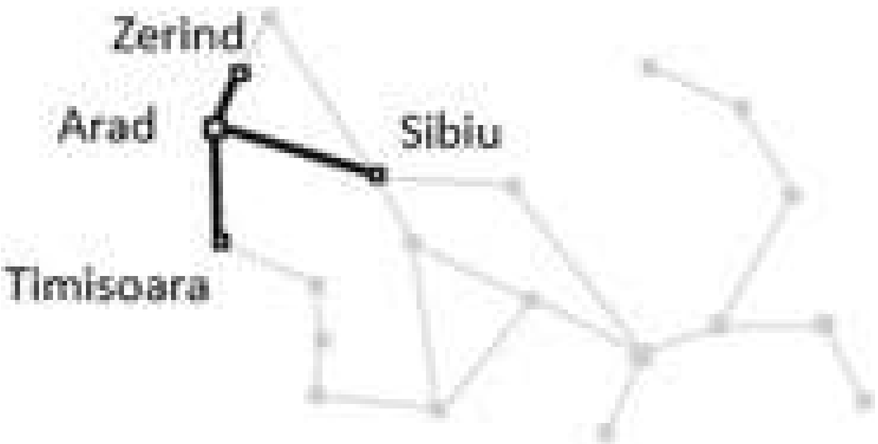
For each node n of the tree,

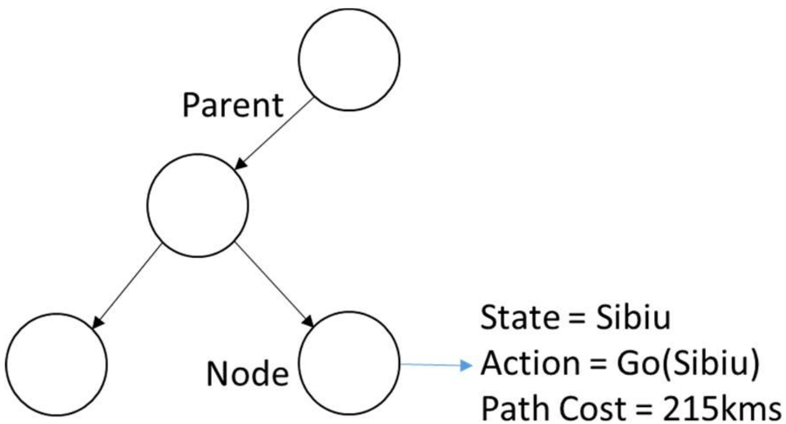
n.STATE : the state in the state space to which node corresponds

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
```

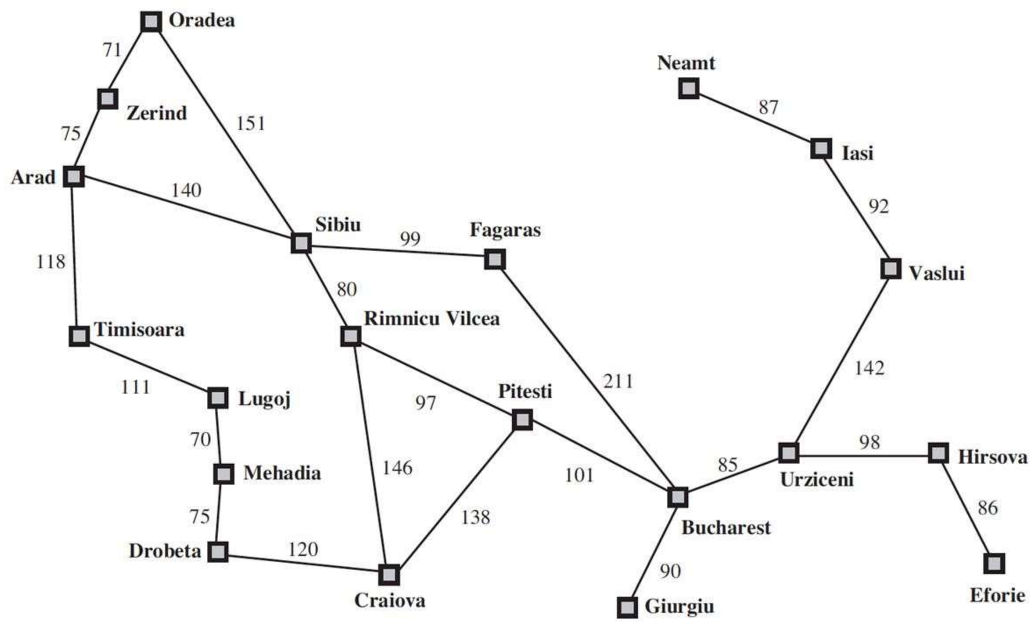
Tree Search Vs Graph Search Algorithms

Coding Aspects

Need:

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

Infinite Loop Path Problem



Start : Arad

Goal : Craiova

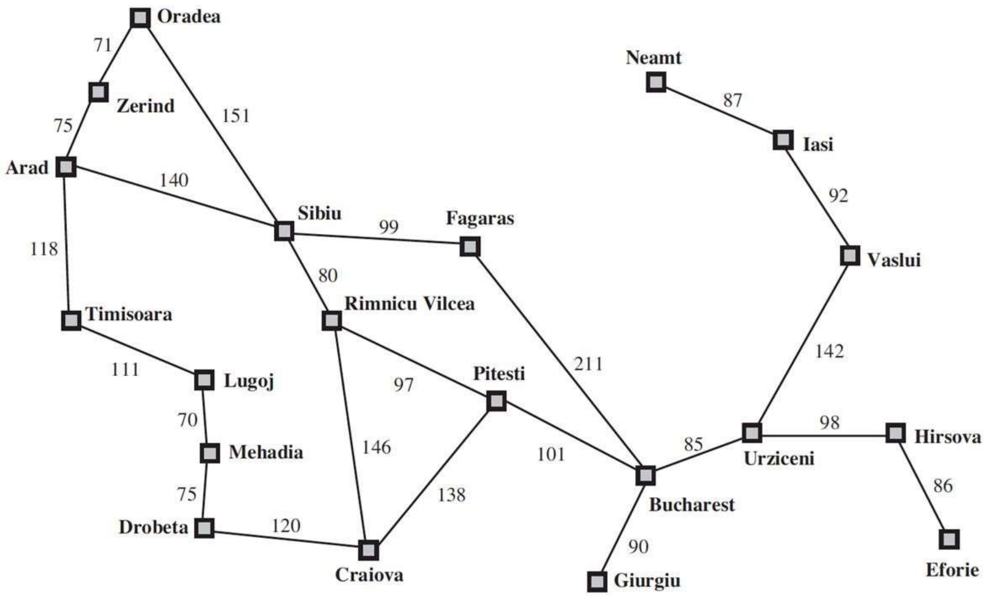
Tree Search Vs Graph Search Algorithms

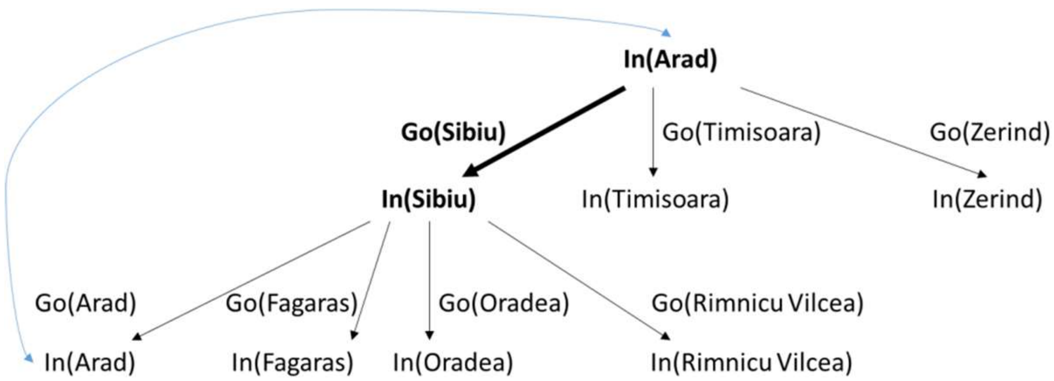
Coding Aspects

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)

Coding Aspects

For each node n of the tree,

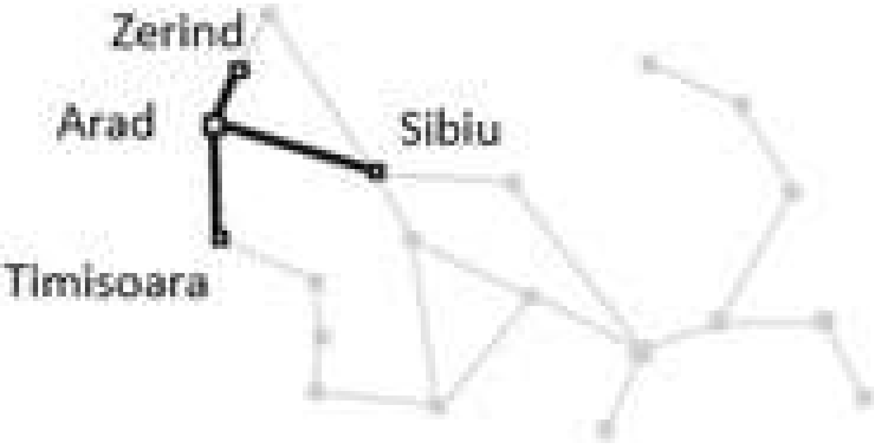
n.STATE : the state in the state space to which node corresponds

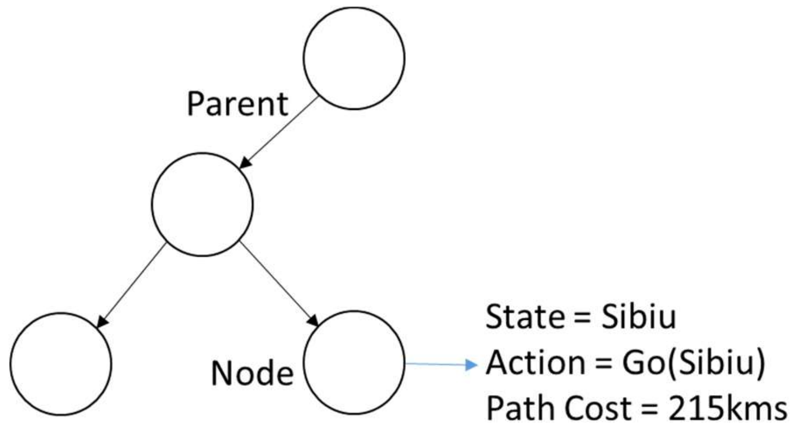
n.PARENT : the node in the search tree that generated this node

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



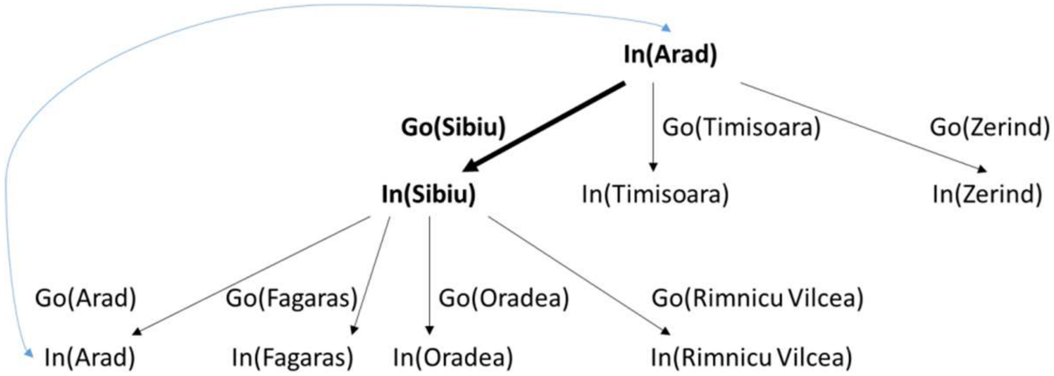


Tree Search Vs Graph Search Algorithms

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
```

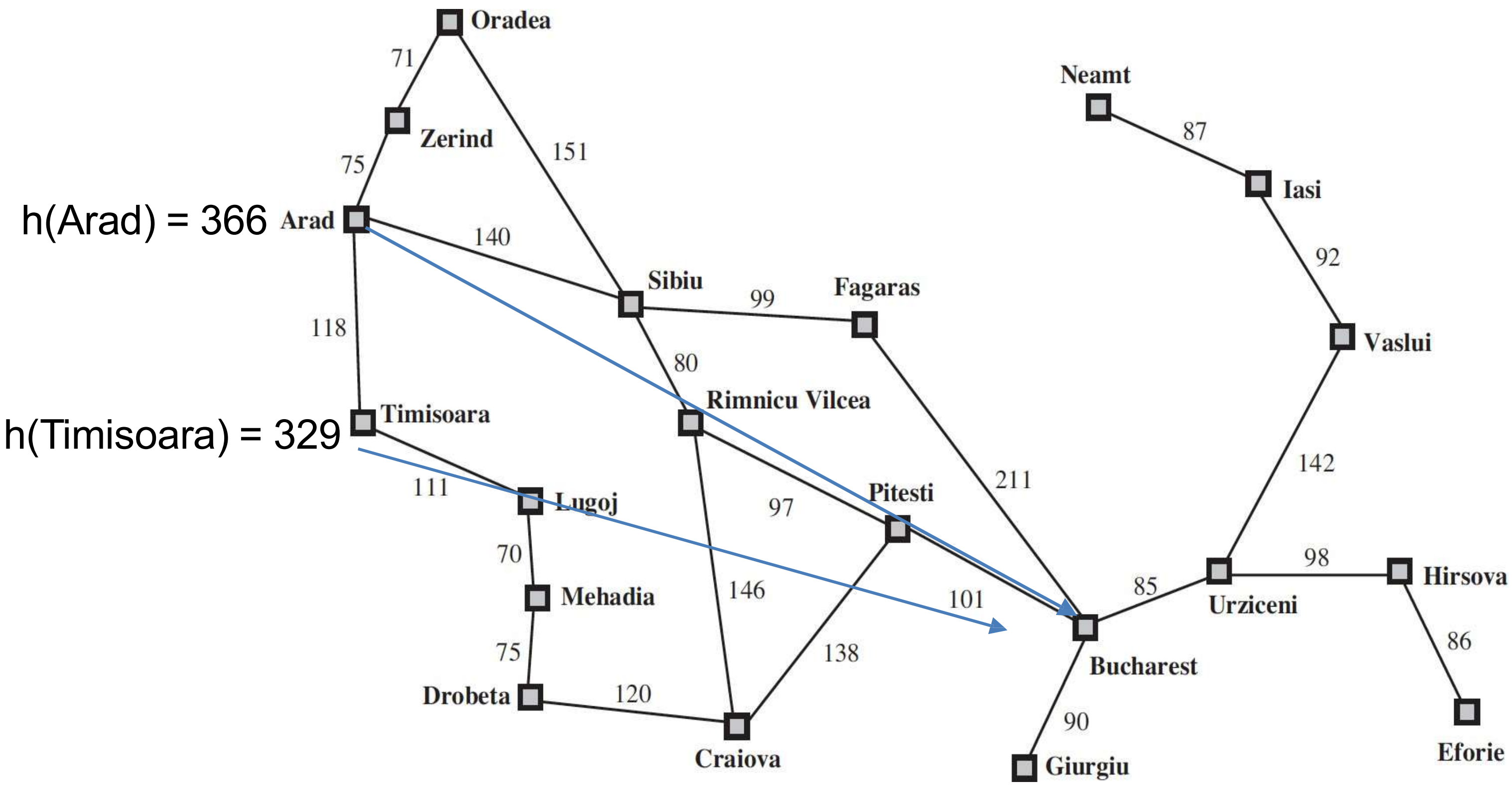
Informed Search

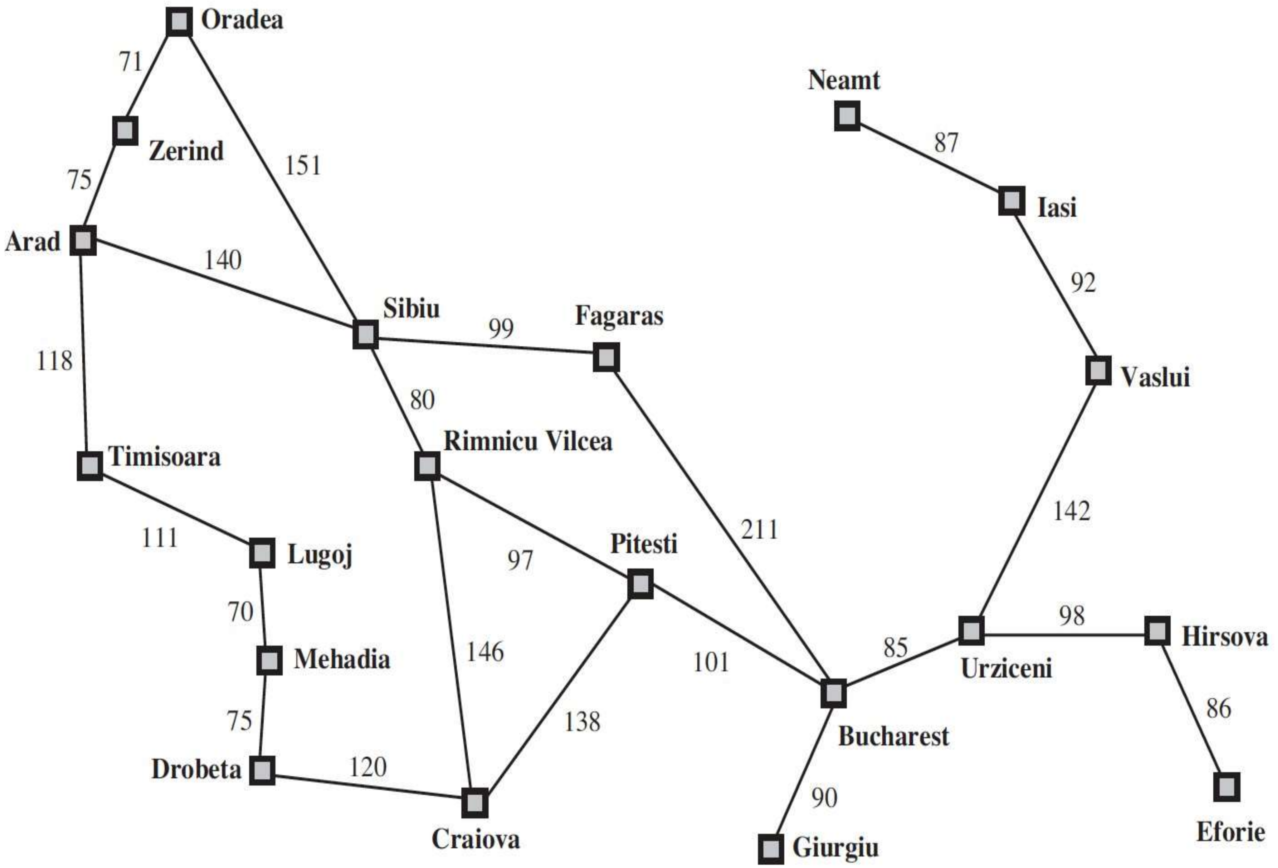
Greedy Best First

A*

Informed /Heuristic Search

Strategies that know if one non-goal state is more promising than another non-goal state





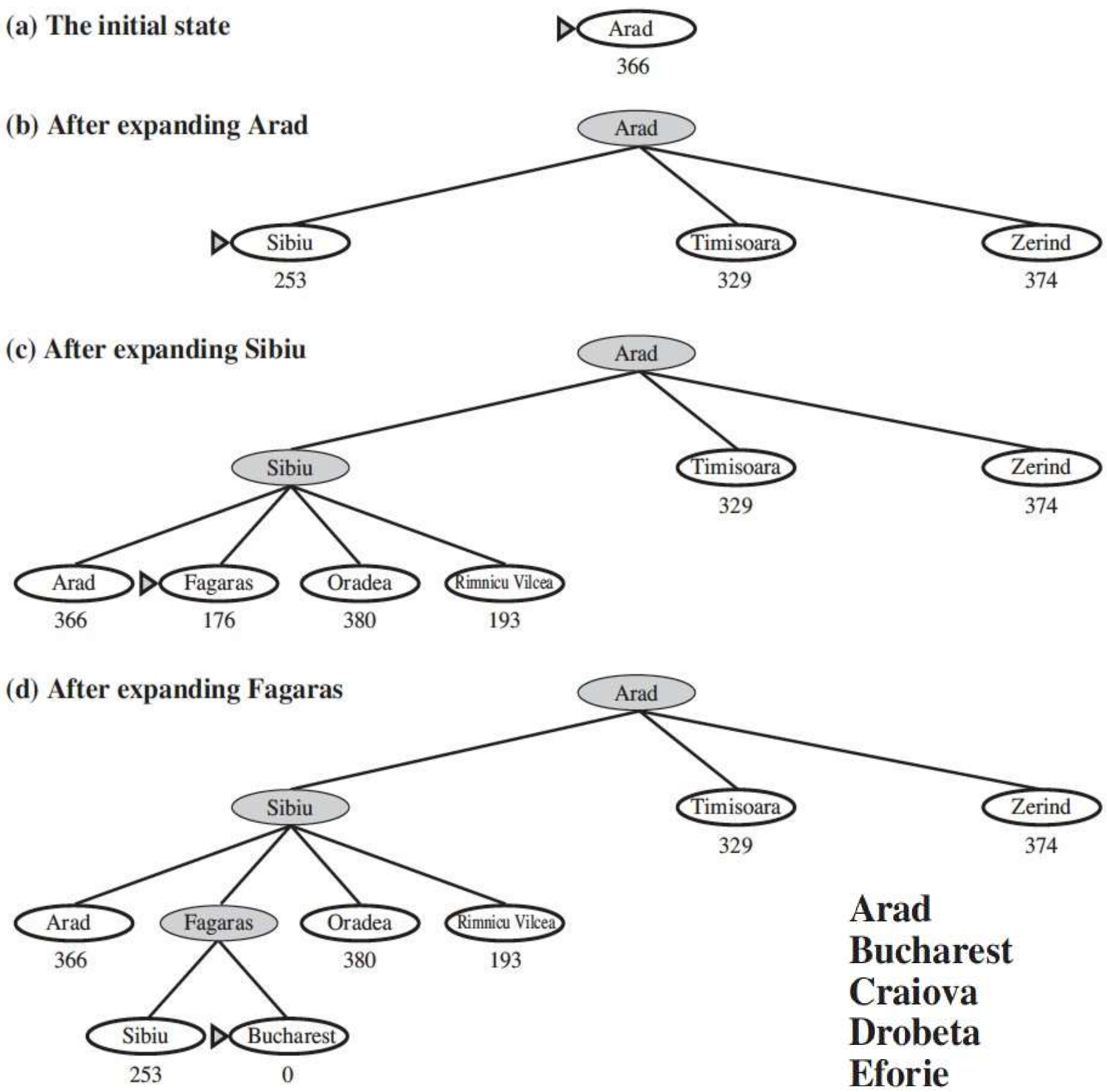
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Greedy Best First Search

Expands the node that is closest to the goal

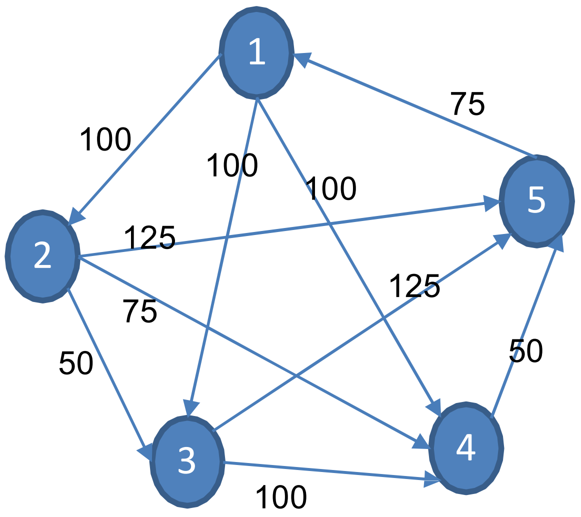
Thus, **f(n) = h(n)**

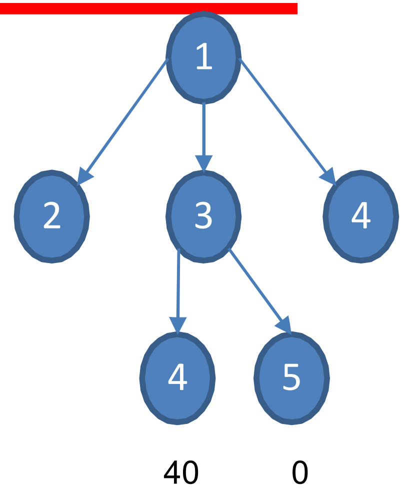
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Greedy Best First Search



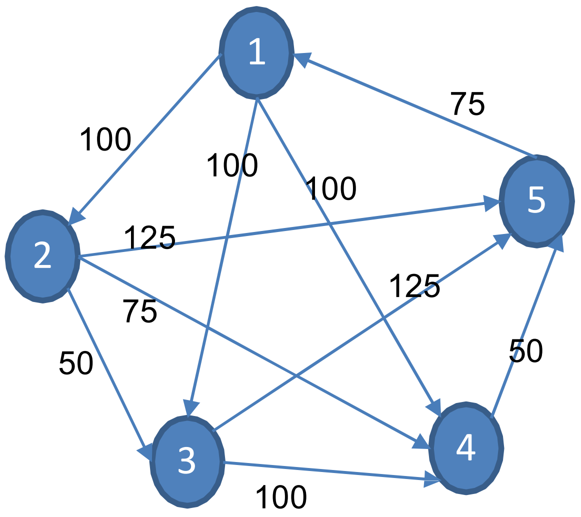


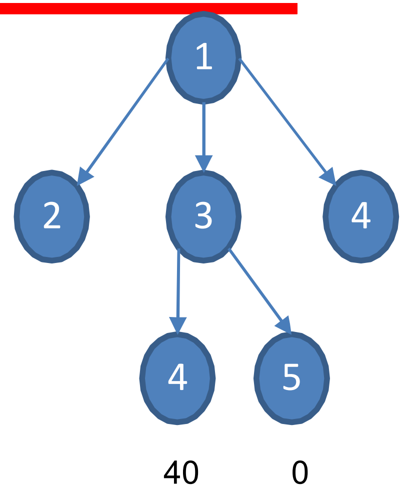
n	h(n)
1	60
2	120
3	30
4	40
5	0

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

C(1-3-5) = 100 + 125 =225
Expanded : 2
Generated : 6
Max Queue Length : 3

Greedy Best First Search





n	h(n)
1	60
2	120
3	30
4	40
5	0

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

C(1-3-5) = 100 + 125 =225
Expanded : 2
Generated : 6
Max Queue Length : 3

A* Search

Expands the node which lies in the closest path (estimated cheapest path) to the goal

Evaluation function **f(n) = g(n) + h(n)**

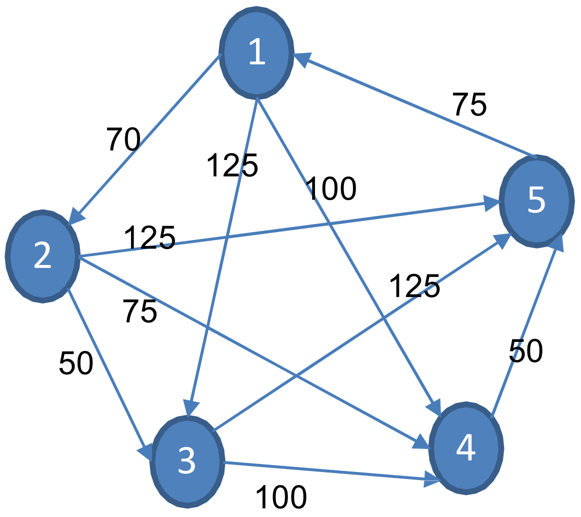
g(n) – the cost to reach the node

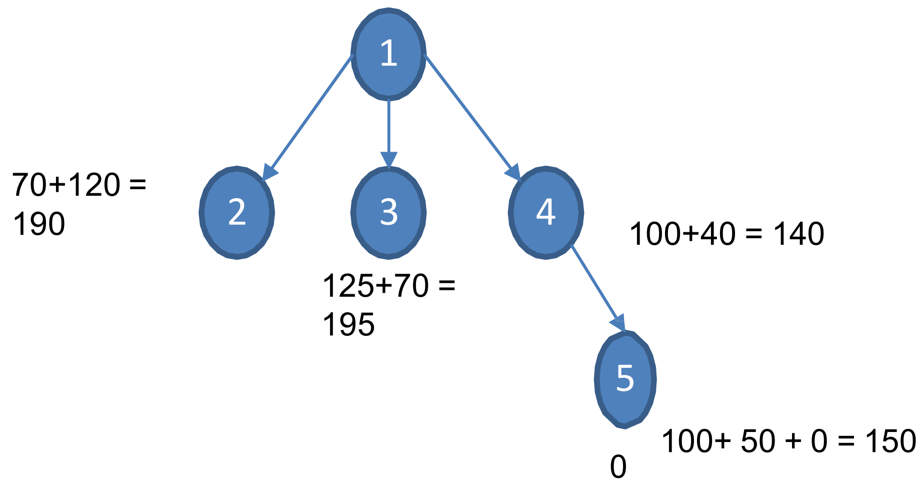
h(n) – the expected cost to go from node to goal

f(n) – estimated cost of cheapest path through node n

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

A* Search





n	h(n)
1	60
2	120
3	70
4	40
5	0

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

C(1-4-5) = 100 + 150 =150
Expanded : 2
Generated : 5
Max Queue Length : 3

Required Reading: AIMA - Chapter #3: 3.1, 3.2, 3.3, 3.4, 3.5

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