Artificial and Computational Intelligence

AIMLCZG557

Contributors & Designers of document content : Cluster Course Faculty Team

M2 : : Problem Solving Agent using Search

BITS Pilani

Pilani Campus

Presented by
V Indumathi –Guest Faculty –BITS-W
indumathi.p@wilp.bits-pilani.ac.in

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

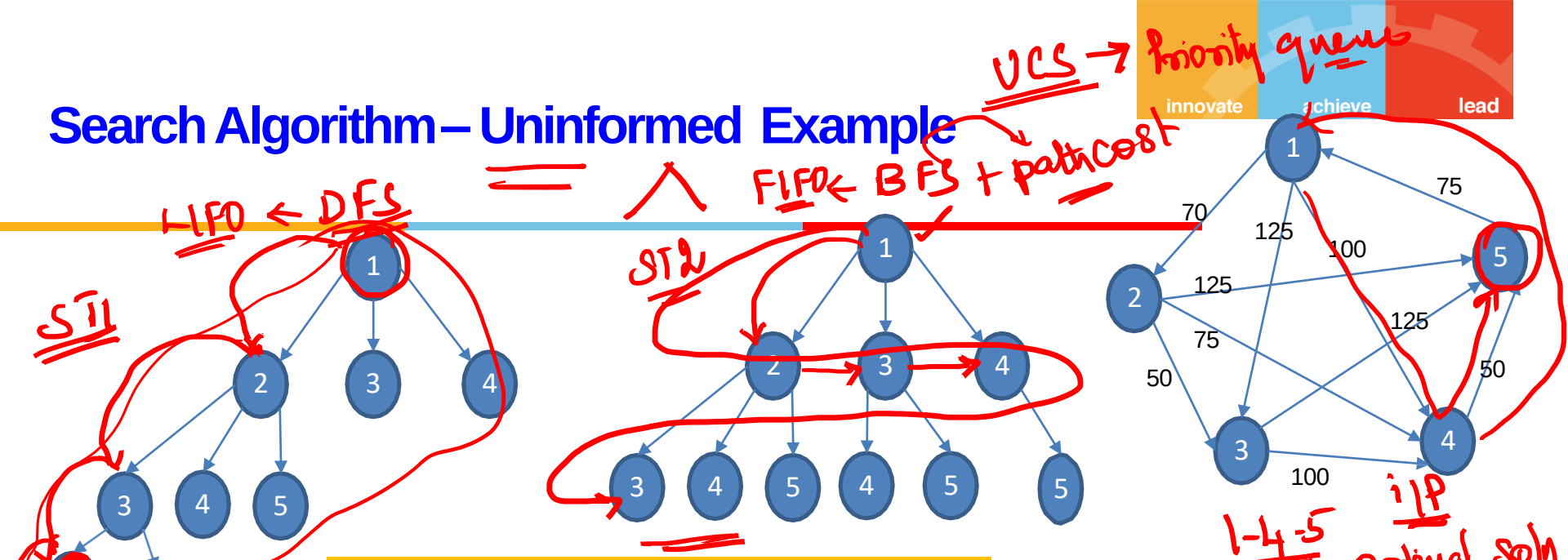
M1	Introduction to AI
M2	Problem Solving Agent using Search
M3	Game Playing
M4	Knowledge Representation using Logics
M5	Probabilistic Representation and Reasoning
M6	Reasoning over time
M7	Ethics in AI

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.**
6. **Differentiate between Tree and Graph search**

Search Algorithm – Uninformed Example



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)

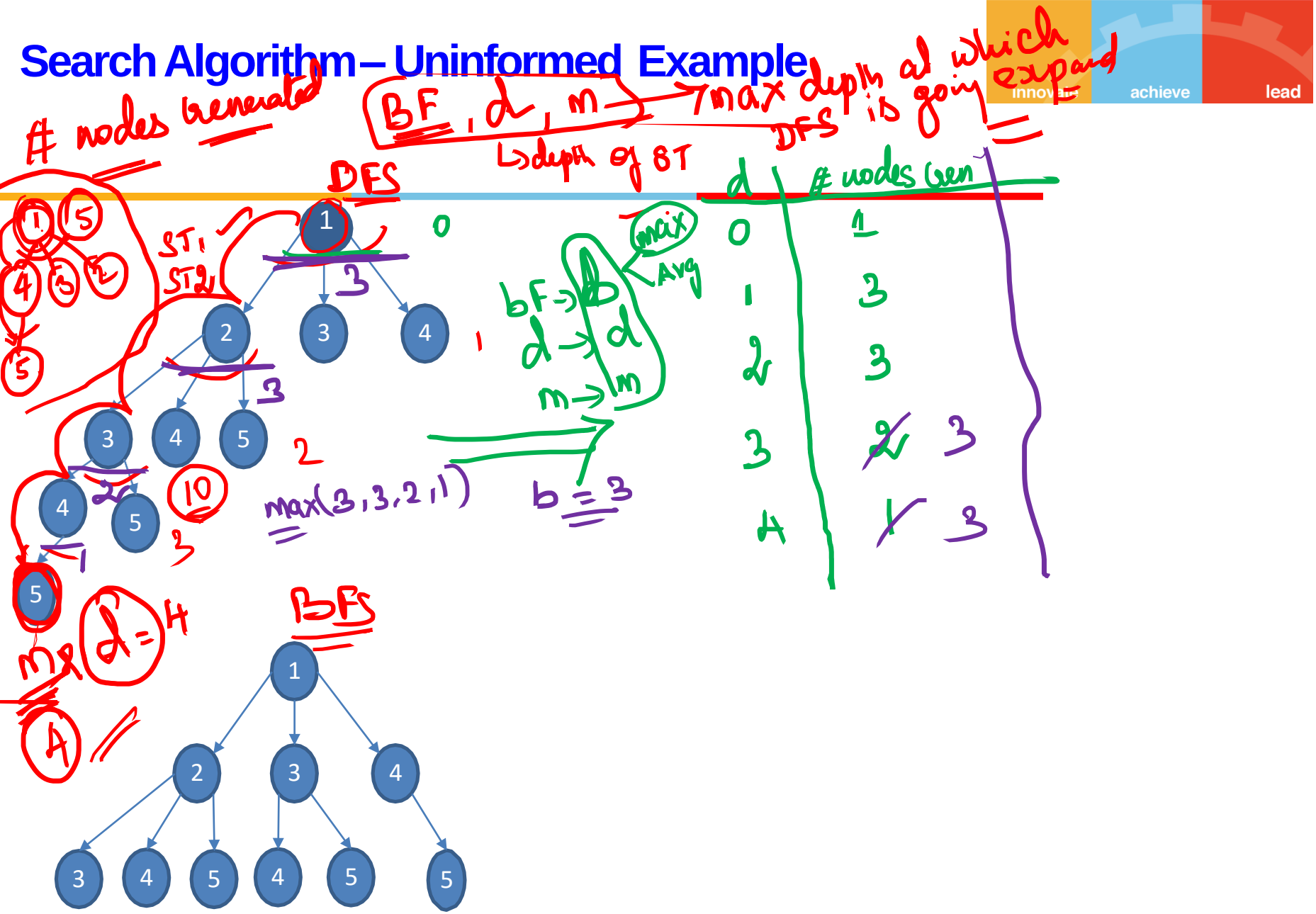
C(1-2-3-4-5) = 70 + 50 + 100 + 50 =270
Expanded : 4
Generated : 10
Max Queue Length : 6

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

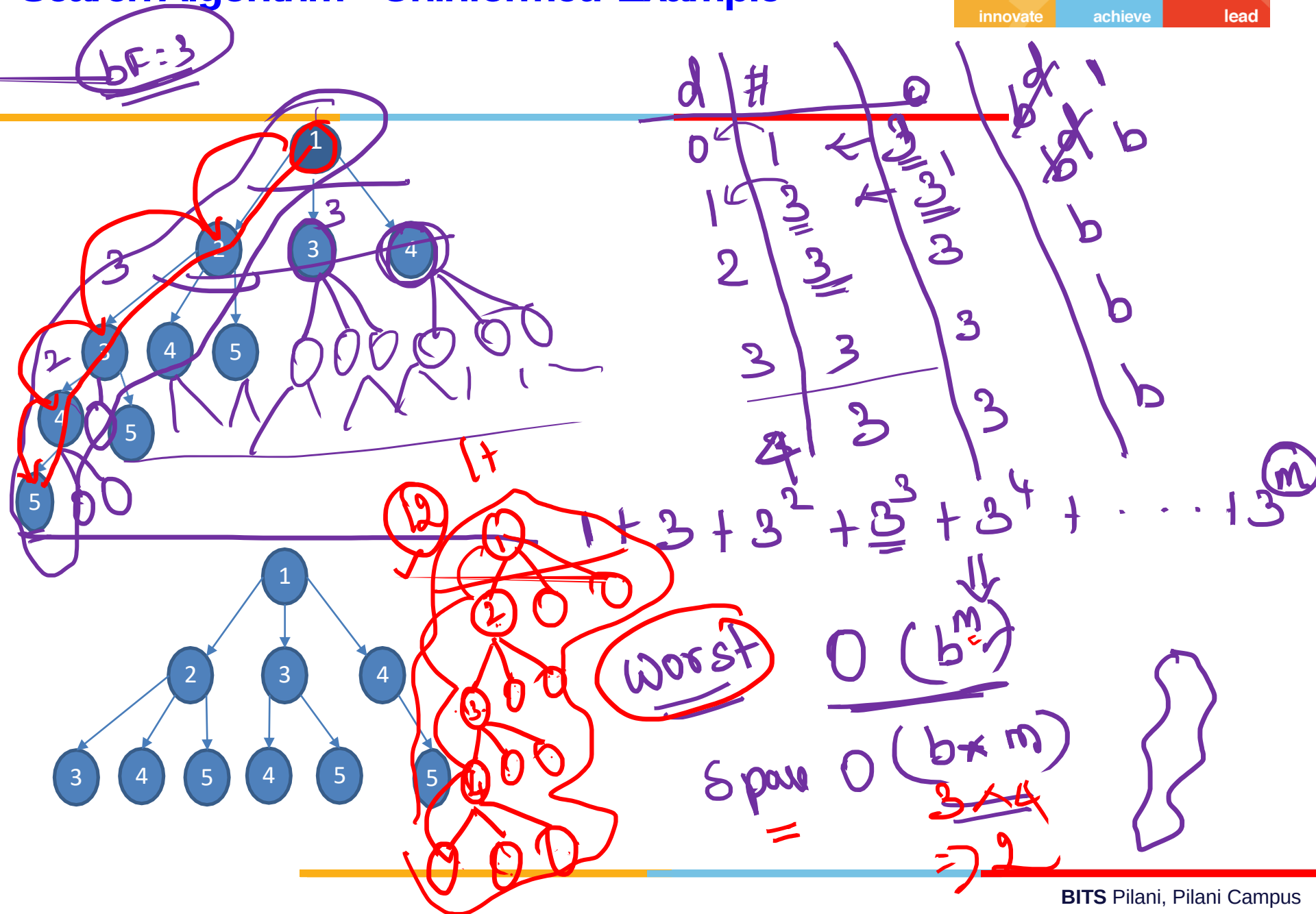
(1 3) (1 4) (1 2 3) (1 2 4) (1 2 5)
(1 2 3) (1 2 4) (1 2 5) (1 3 4) (1 3 5) (1 4 5)
TEST PASSED

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

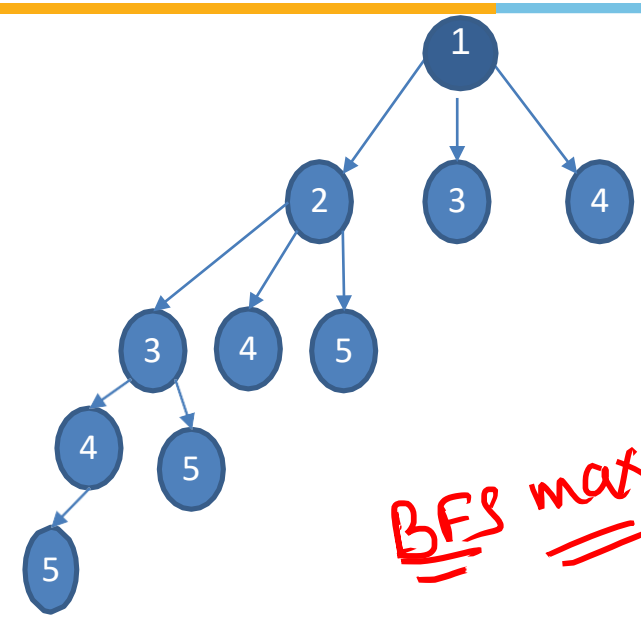
Search Algorithm – Uninformed Example



Search Algorithm – Uninformed Example



Search Algorithm – Uninformed Example

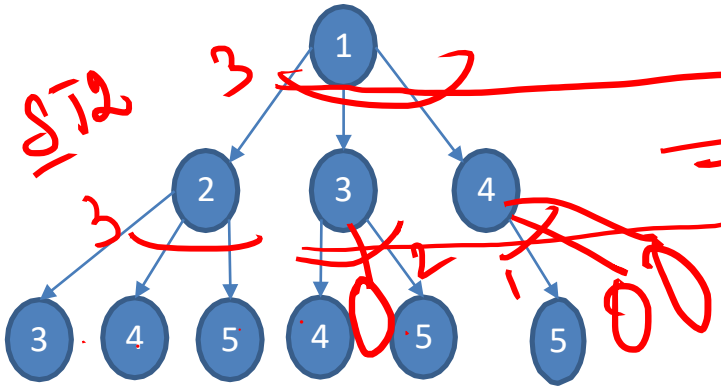


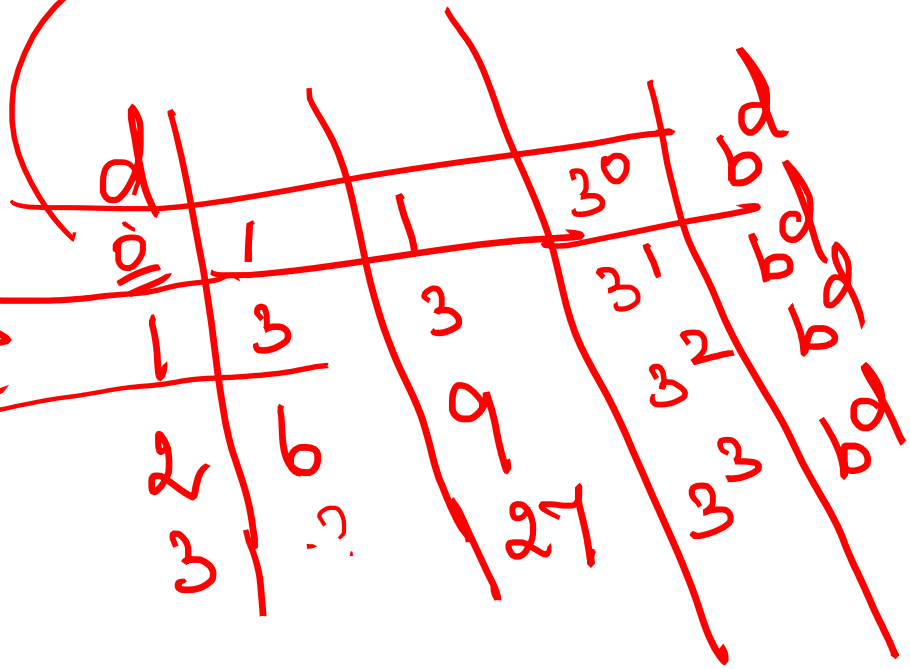
$O(b^d)$

Space ($O(b^{d+1})$

max bF

BFS max bf = 3





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

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

Uniform Cost Search – Evaluation

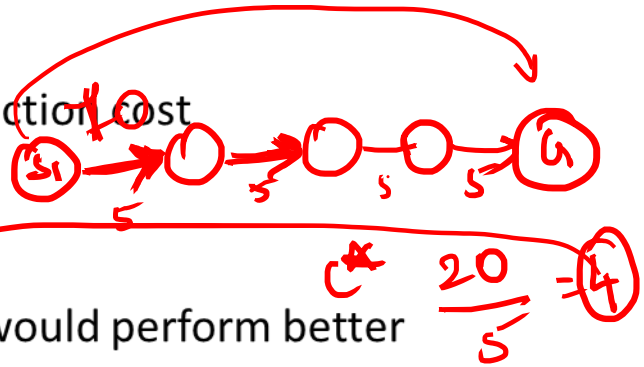
Completeness – It is complete if the cost of every step > small +ve constant ϵ

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



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)

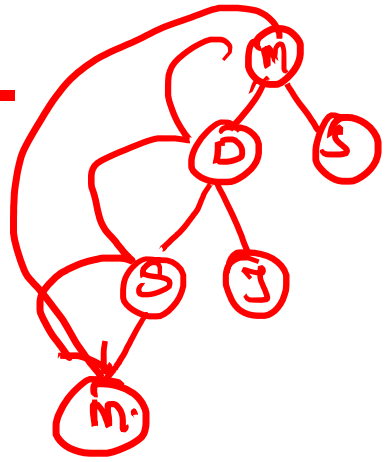
innovate achieve lead

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

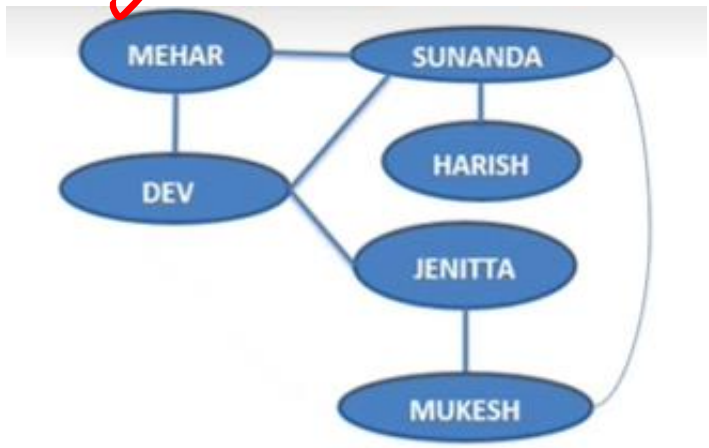


Breadth First Search

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

Depth First Search

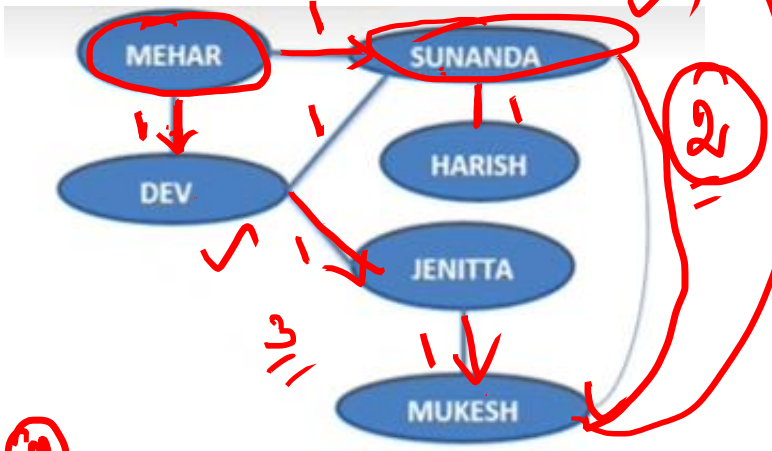
- Find the Connectedness in a graph
- Topological Sorting



Application

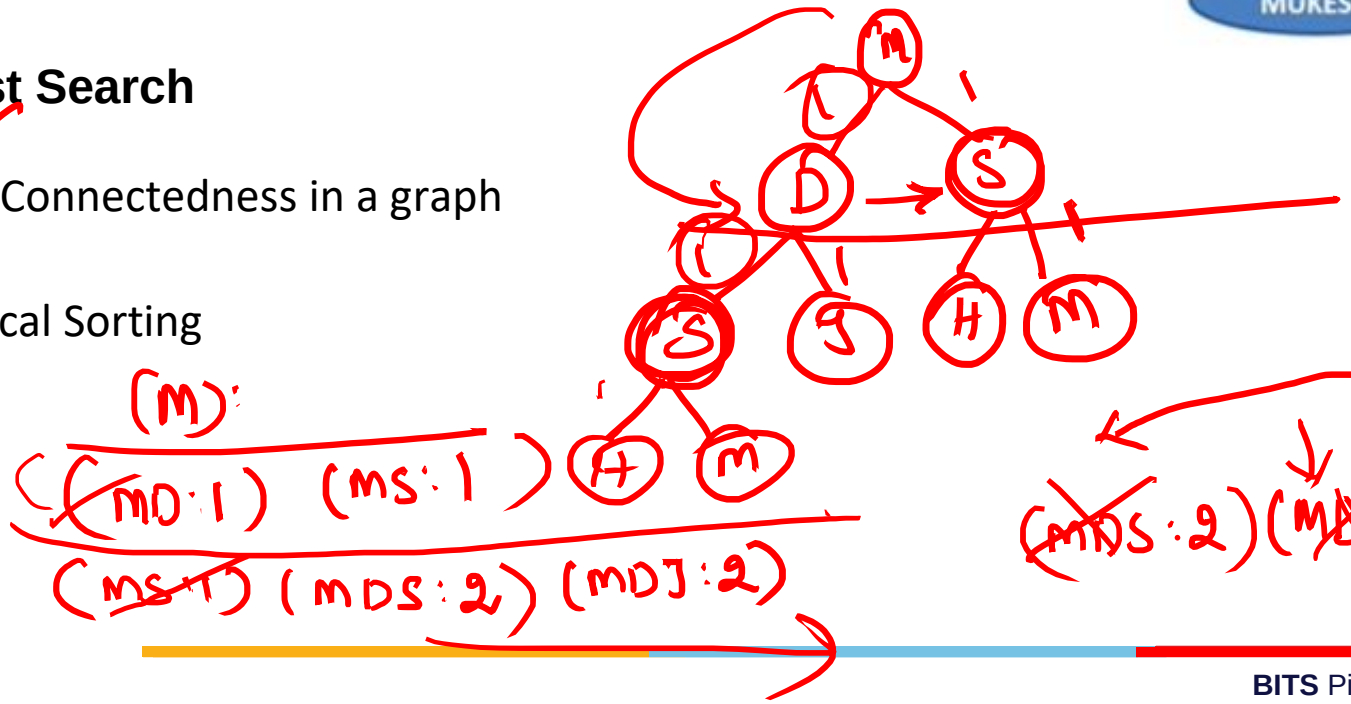
Breadth First Search

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



Depth First Search

- Find the Connectedness in a graph
- Topological Sorting

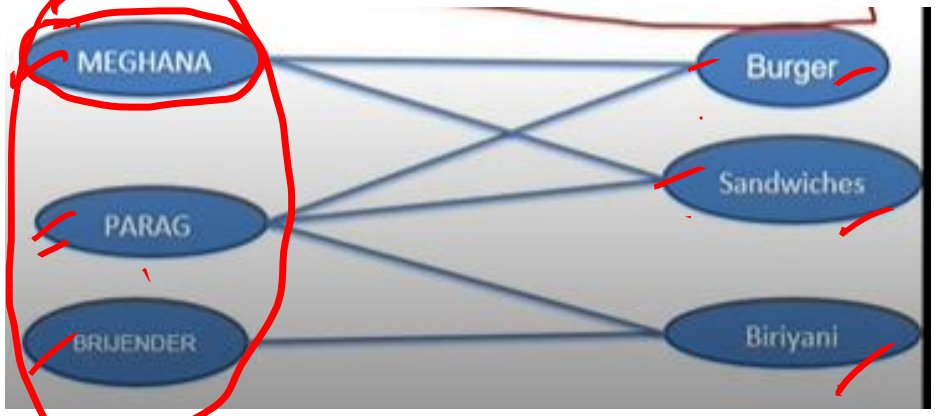


Application

innovate achieve lead

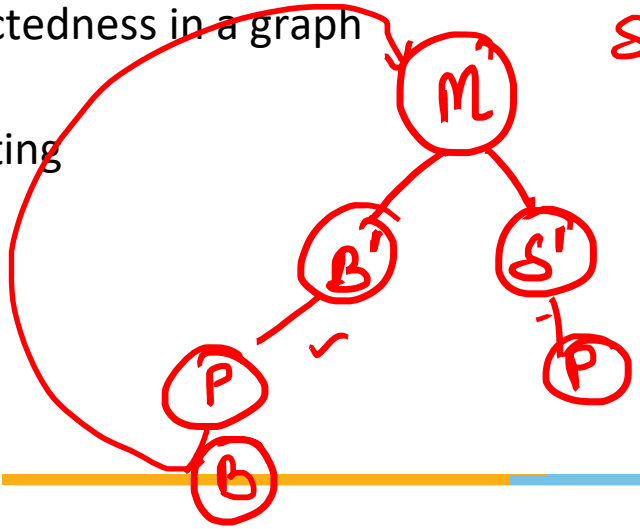
Breadth First Search

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

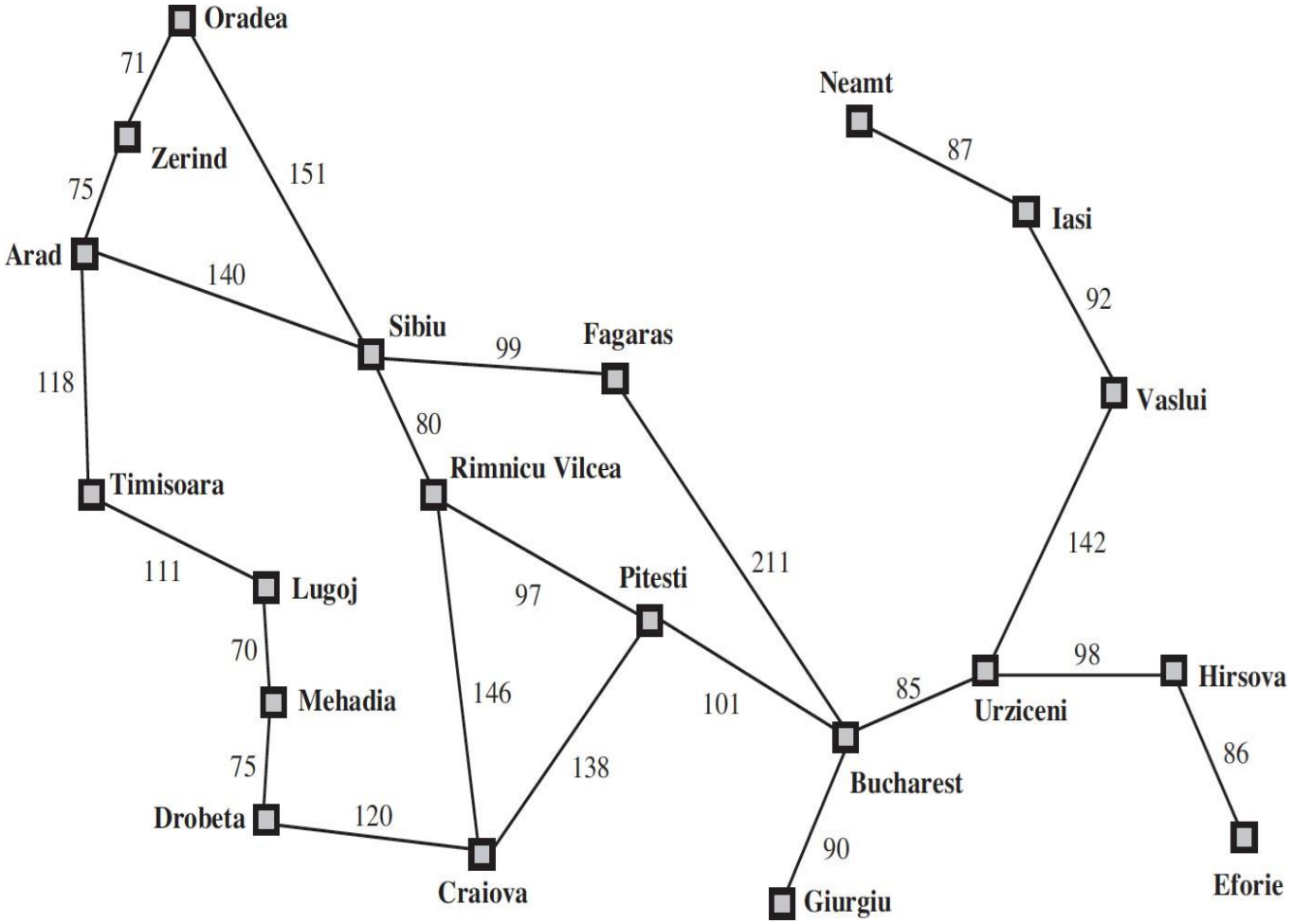


Depth First Search

- Find the Connectedness in a graph
- Topological Sorting



Tree Search Vs Graph Search


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

Search

Coding Aspects

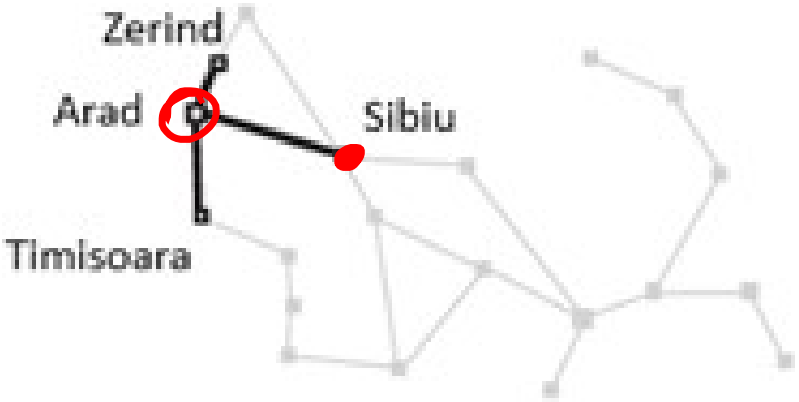
For each node n of the tree,

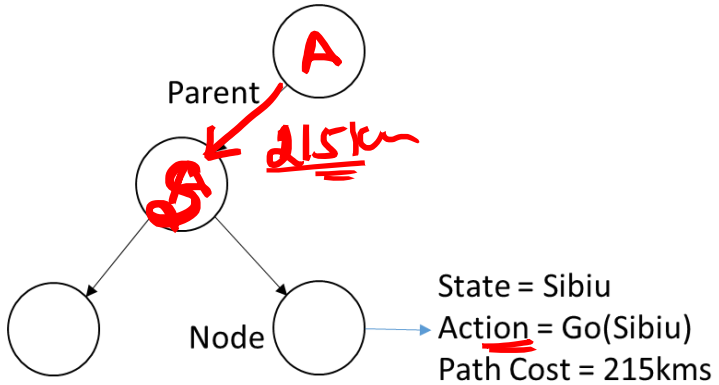
n.STATE : the state in the state space to which node corresponds

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





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

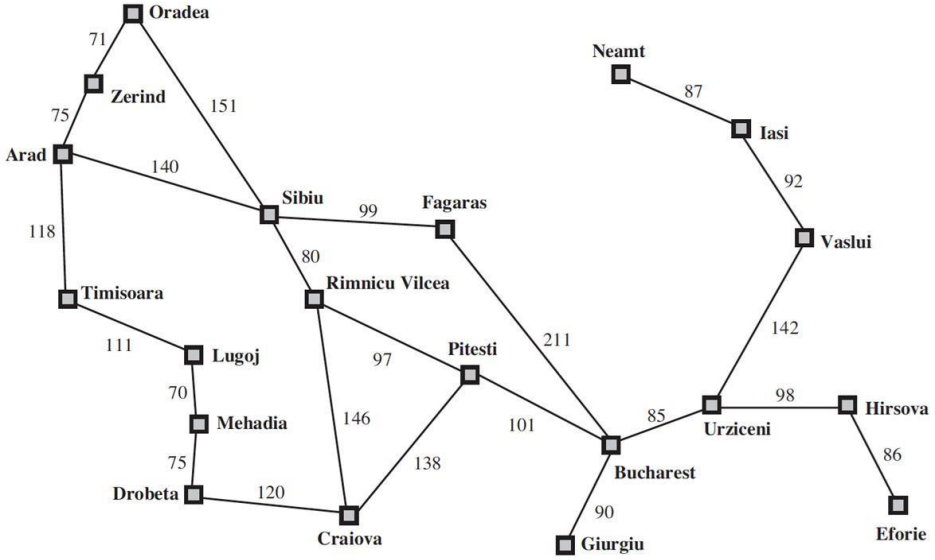
innovate achieve lead

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

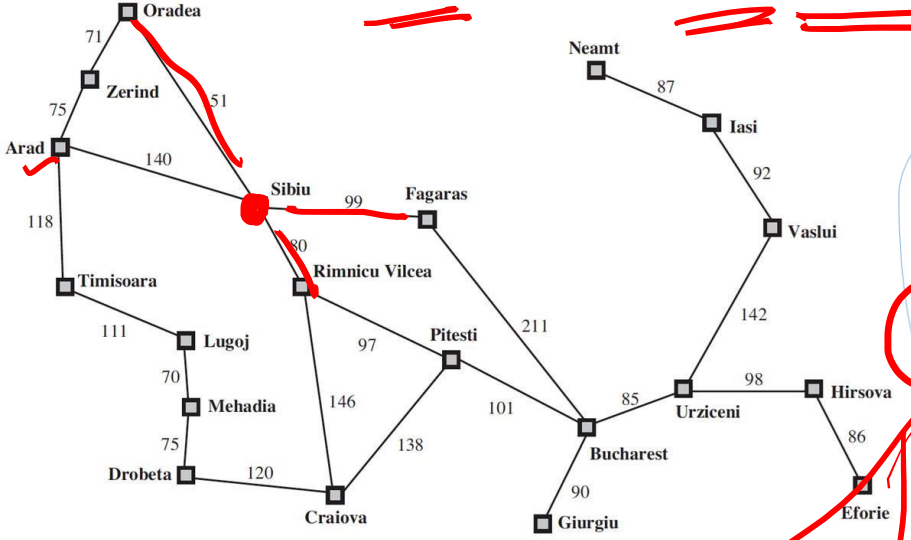
innovate achieve lead

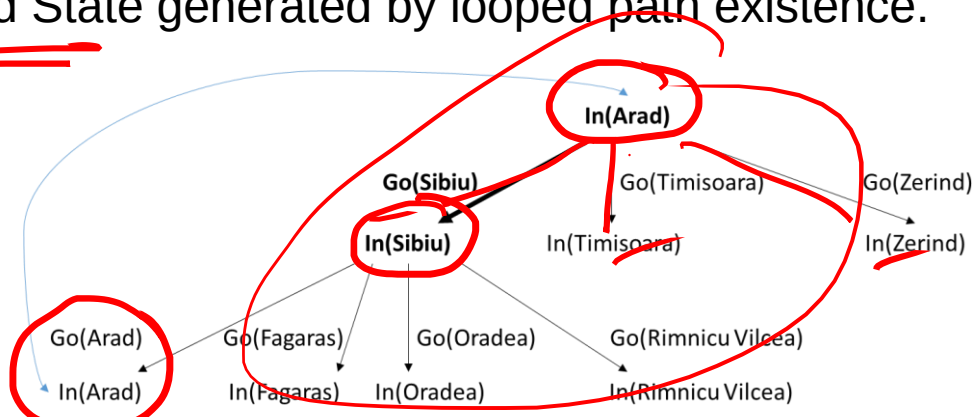
Coding Aspects

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,

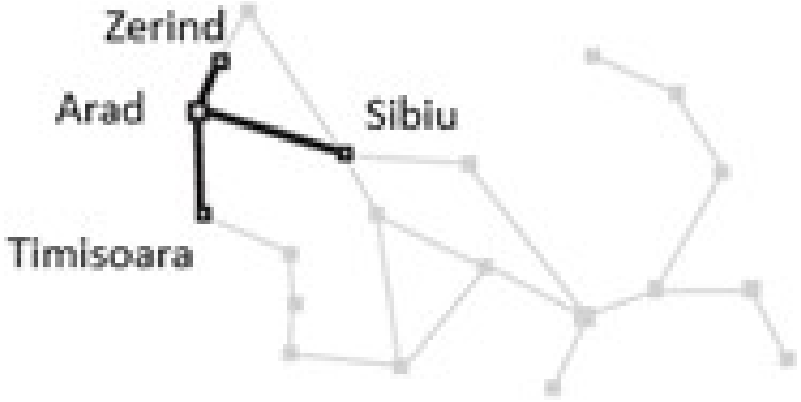
n.STATE : the state in the state space to which node corresponds

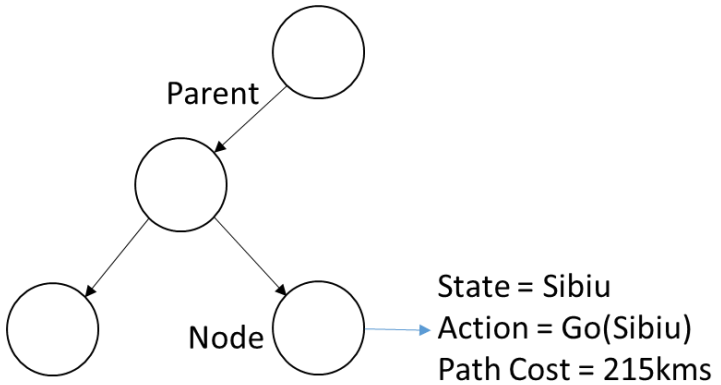
n.PARENT : the node in the search tree that generated this node

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





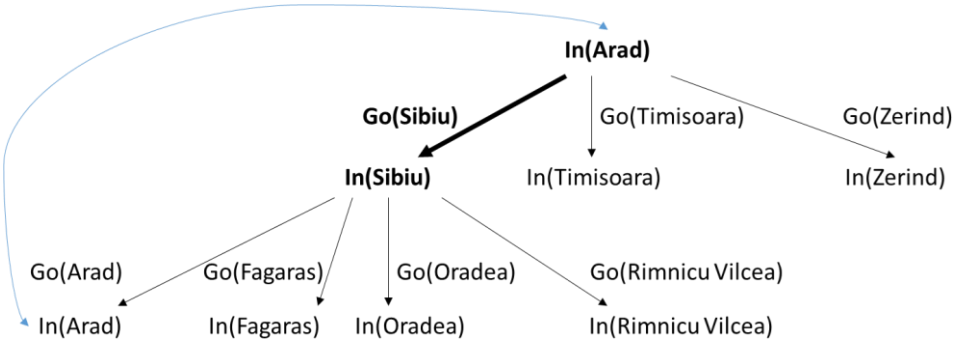
Tree Search Vs Graph Search Algorithms

innovate achieve lead

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

Learning Objective

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

1. Create Search tree for given problem
2. Differentiate between uninformed and informed search requirements
3. Apply GBFS & A* algorithms to the given problem
4. Prove if the given heuristics are admissible and consistent
5. Apply A* variations algorithms to the given problem

Module 2 : Problem Solving Agent using Search

A. Uninformed Search

B. Informed Search

C. Heuristic Functions

D. Local Search Algorithms & Optimization Problems

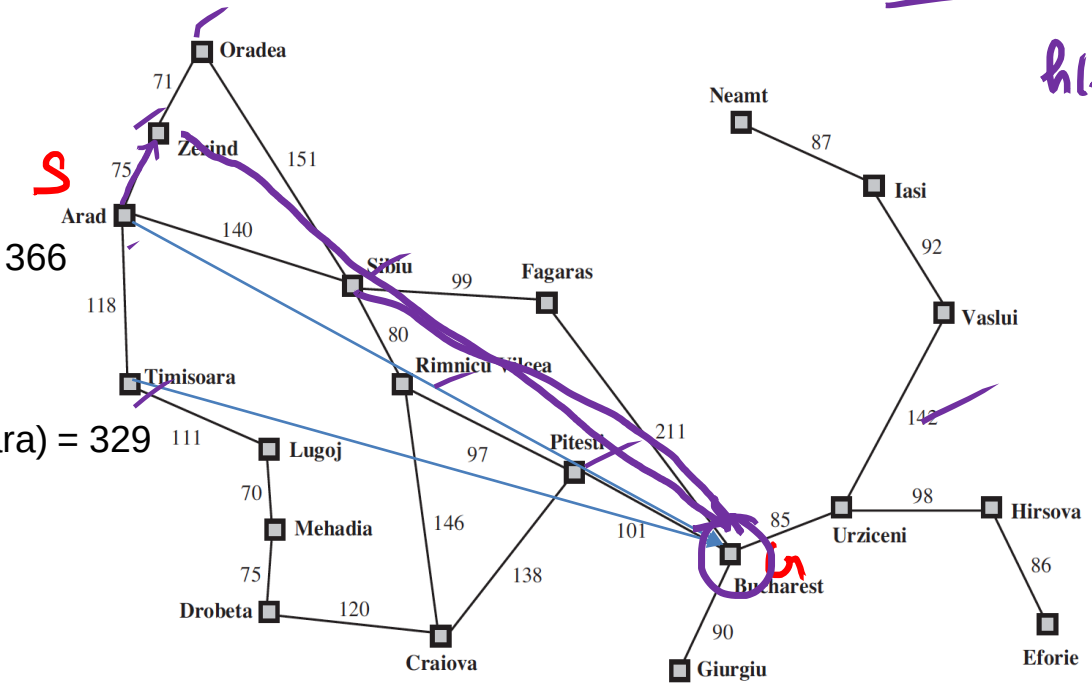
Informed Search
Greedy Best First
A*

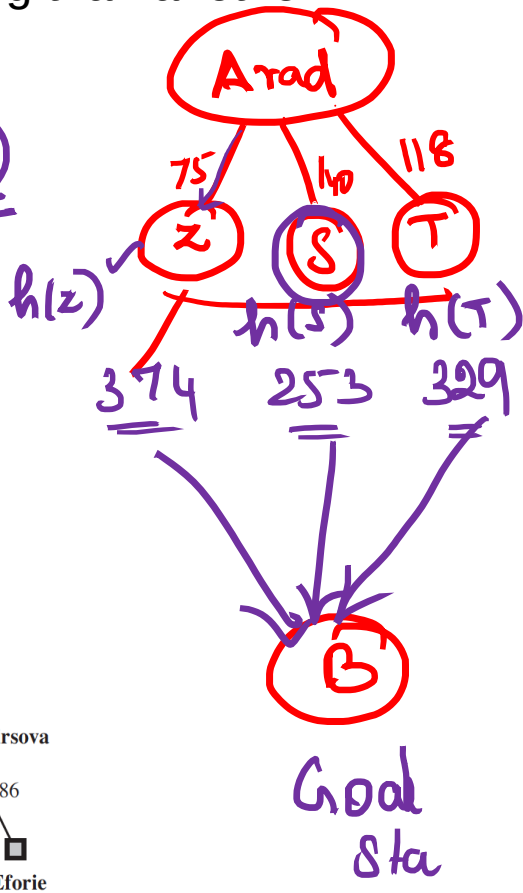
Informed /Heuristic Search

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

h(Arad) = 366

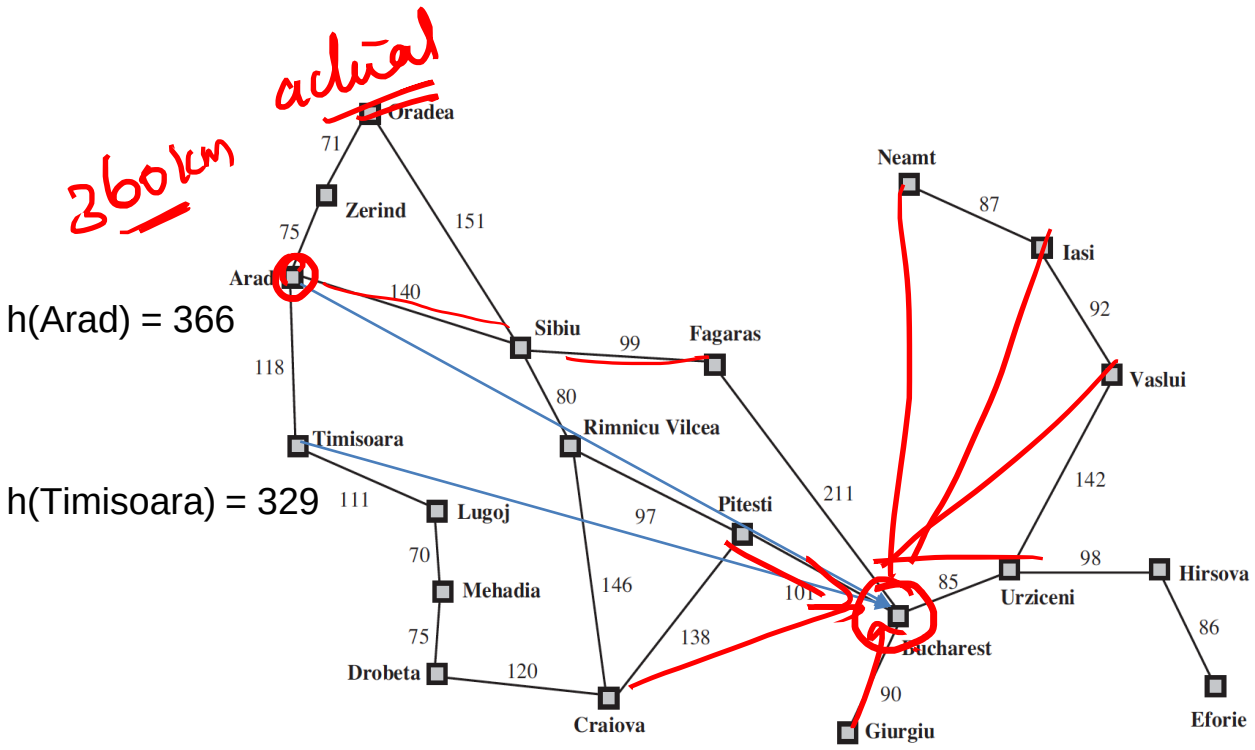
h(Timisoara) = 329





Informed /Heuristic Search

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

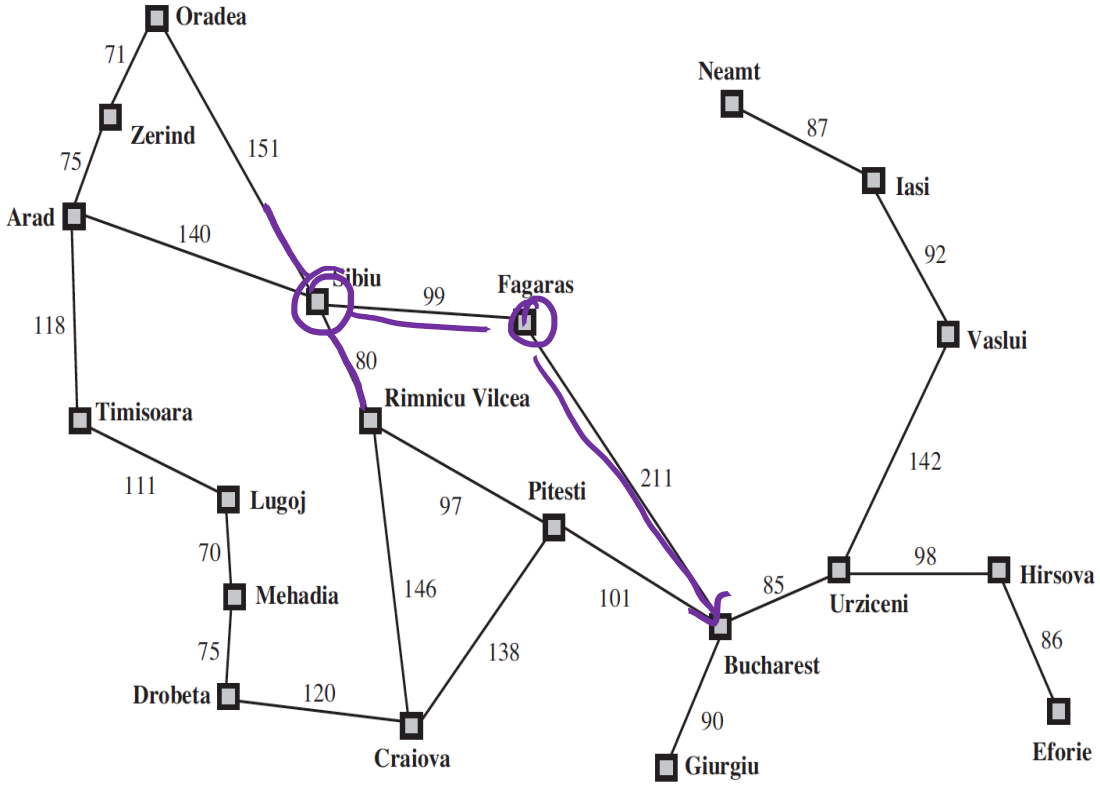


Greedy Best First Search

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

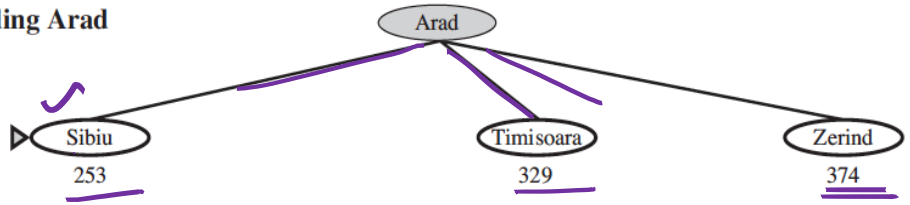


Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

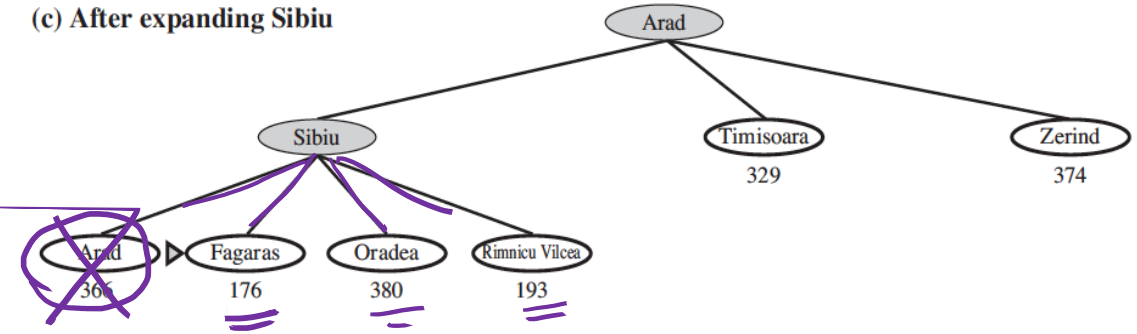
GBFS

(a) The initial state

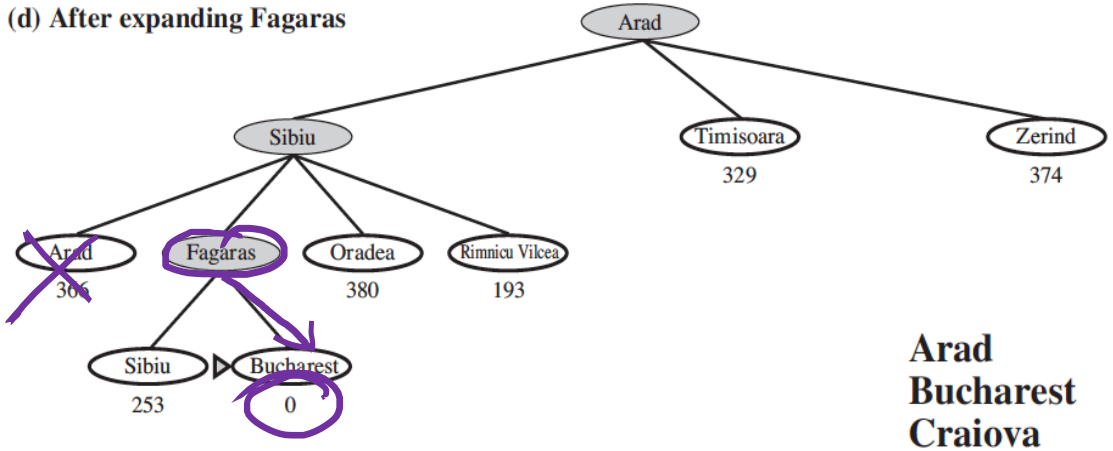
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



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

Not Optimal

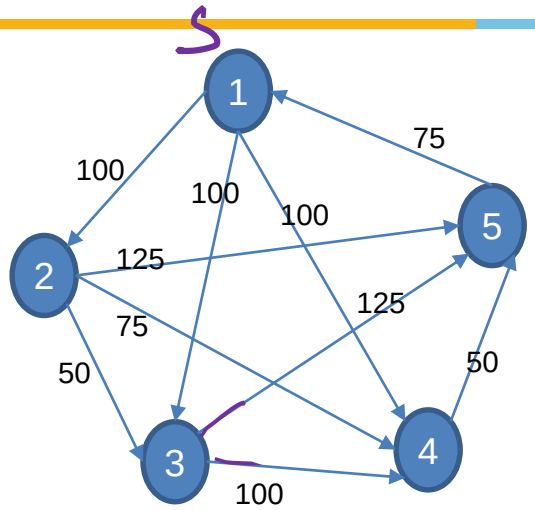
- Because the algorithm is greedy
- It only optimizes for the current action

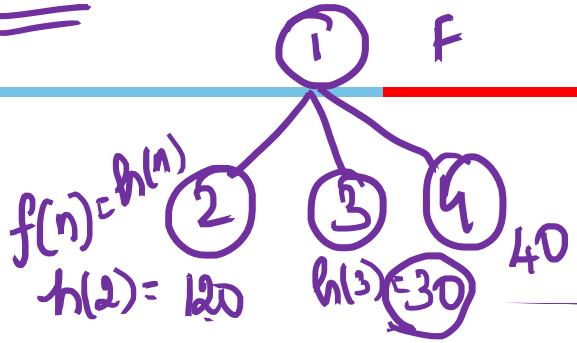
Not Complete

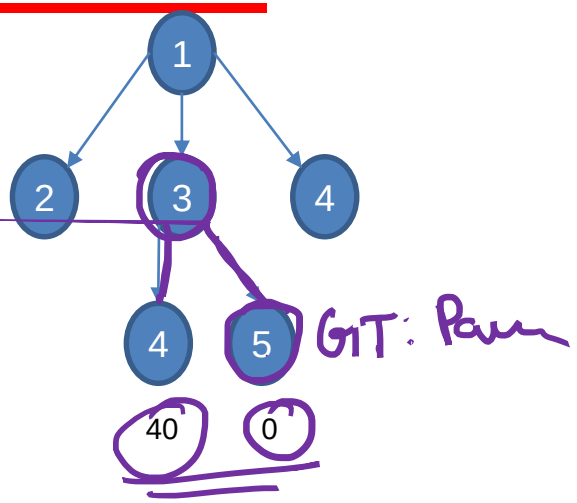
- Often ends up in state with a dead end as the heuristic doesn't guarantee a path but is only an approximation

Time and Space Complexity - $\mathcal{O}(b^m)$ where m – max depth of search tree

Greedy Best First Search





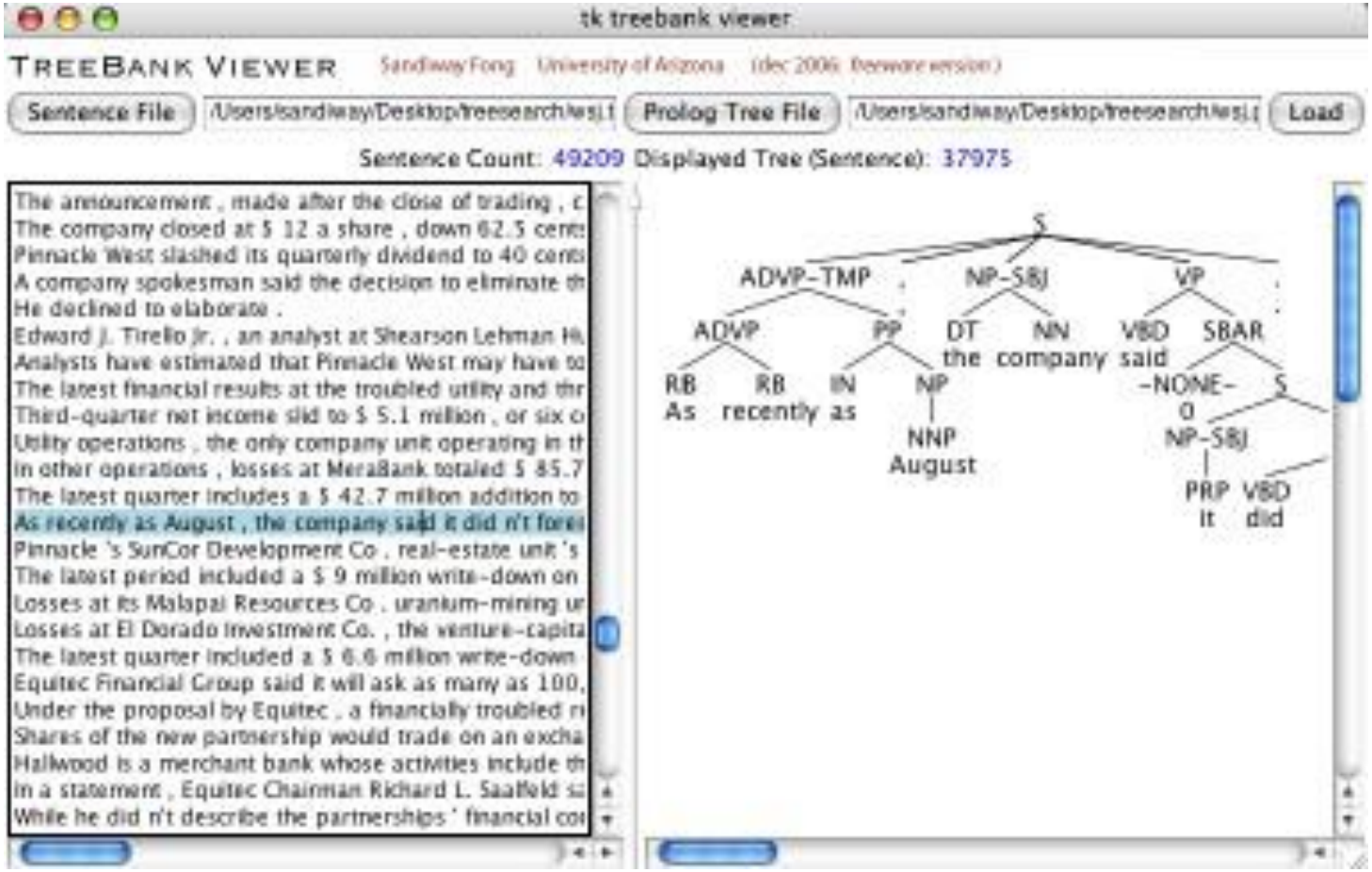


n	h(n)
1	60
2	120
3	30
4	40
5	0

(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
Idea: Optimize DFS. Choose next nearest to goal in the same hill.

Case Study – 1 Search in Treebanks



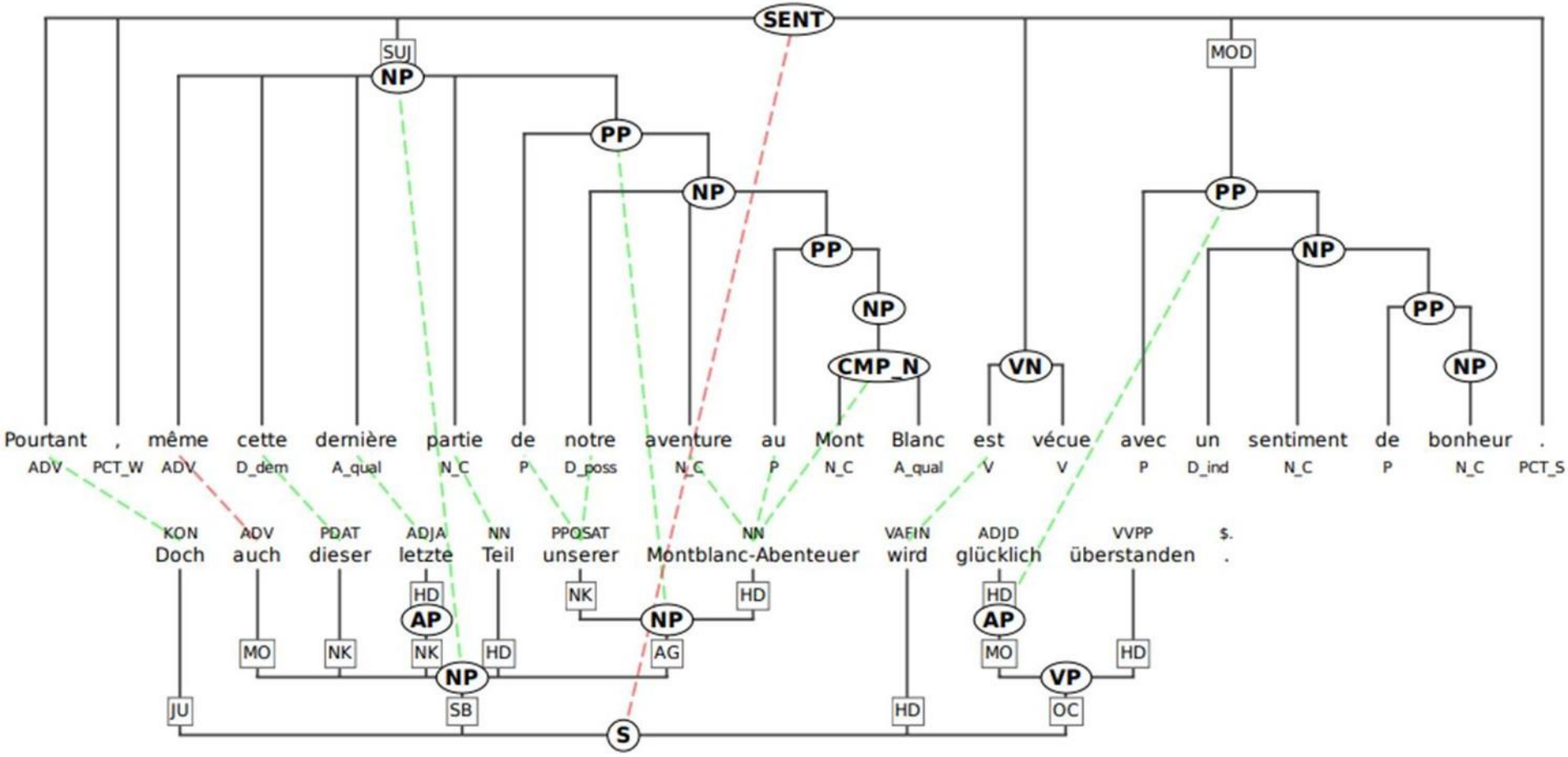
Source Credit :

https://catalog.ldc.upenn.edu/docs/LDC95T7/cl93.html

https://ufal.mff.cuni.cz/pdt3.5

BITS Pilani, Pilani Campus

Case Study – 1 Search in Treebanks



Source Credit :

https://catalog.ldc.upenn.edu/docs/LDC95T7/cl93.html

https://ufal.mff.cuni.cz/pdt3.5

BITS Pilani, Pilani Campus

A* Search

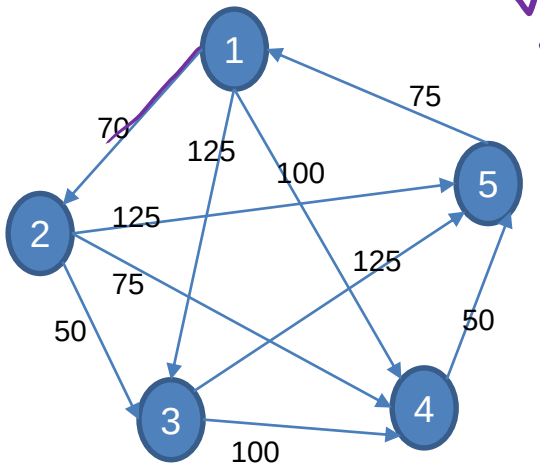
Expands the node which lies in the closest path (estimated cheapest path) to the goal

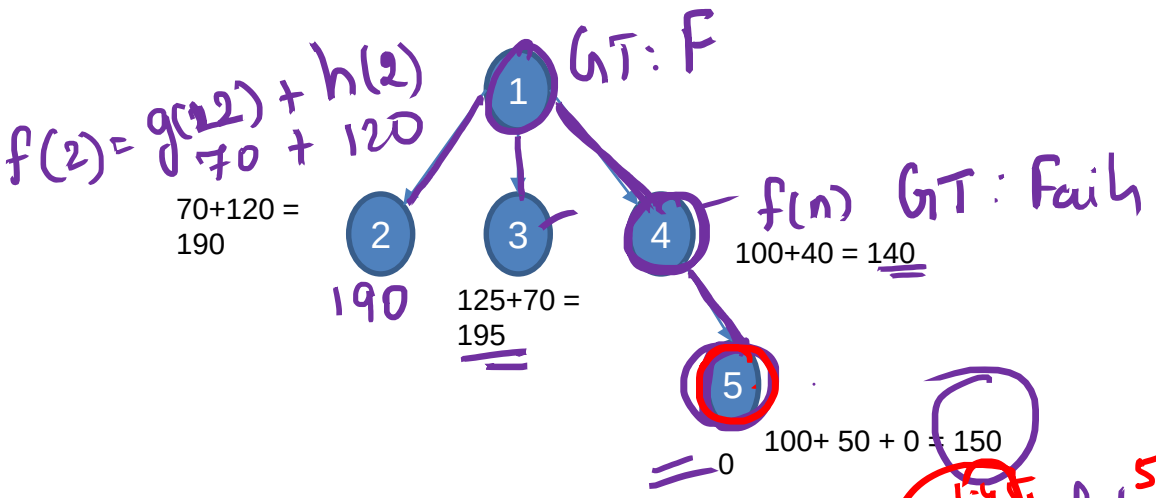
Evaluation function **f(n) = g(n) + h(n)**

- 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



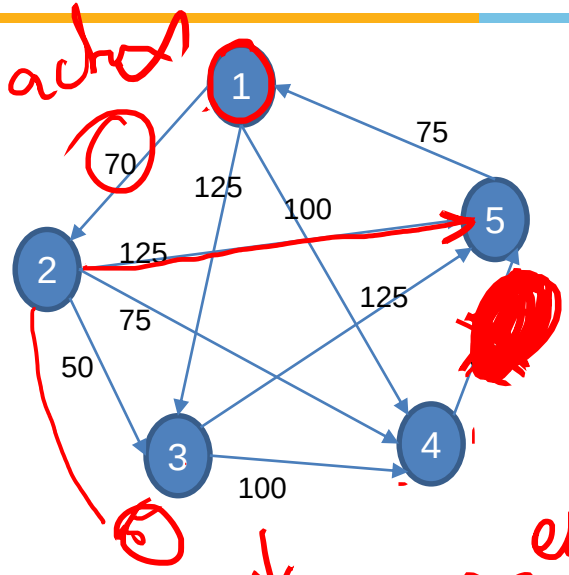


n	h(n)
1	60
2	120
3	70
4	40
5	0

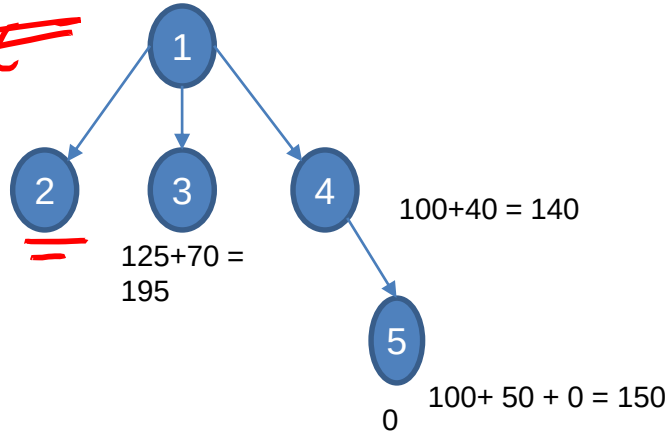
(1)
(1 4) (1 2) (1 3)
(1 4 5) (1 2) (1 3)

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

A* Search →

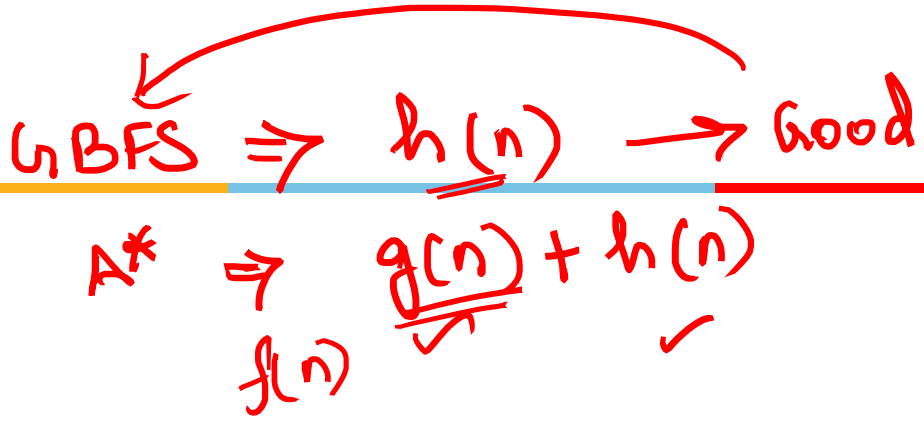


n	h(n)
1	60
2	120
3	70
4	40
5	0



(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



Optimality of A*

A* Search

Test for Admissibility

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

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

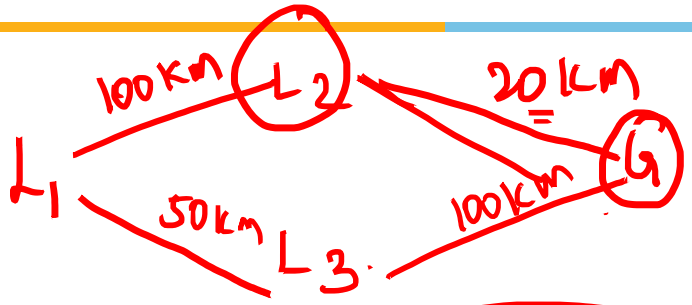
g(n) – the cost to reach the node

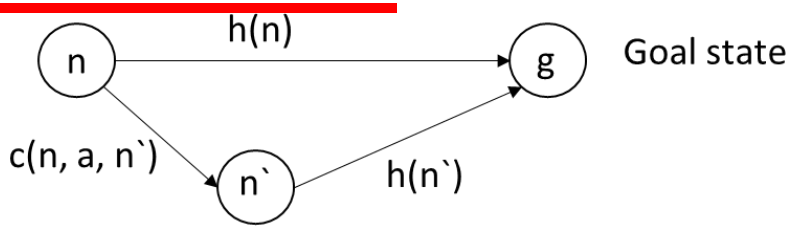
h(n) – the expected cost to go from node to goal

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

A heuristic is admissible or optimistic if , **$0 \leq h(n) \leq h^*(n)$**, where h*(n) is the actual cost to reach the goal

A* Search





	h(n)
L2	50 km
L3	30

Overestimated

Underestimation

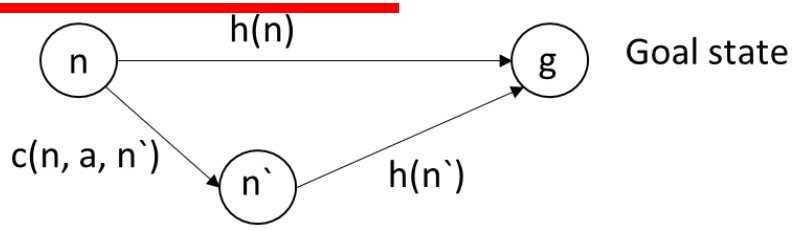
estimated $h(n)$ $\leq$ actual $h^*(n)$

$h_G(L2) < h^*_G(L2)$

$h_G(L2)$ = 50 — less informed — more deviated

$h^*_G(L2)$ = 20 — more informed

A* Search → optimal soln



Optimal on condition

h(n) must satisfies two conditions:

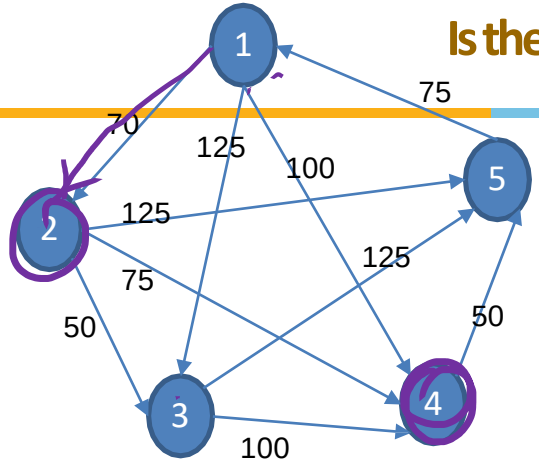
1. Admissible Heuristic – one that never overestimates the cost to reach the goal
2. Consistency – A heuristic is consistent if for every node n and every successor node n` of n generated by action a, h(n) <= c(n, a, n`) + h(n`)

Complete

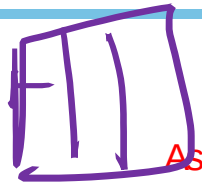
- If the number of nodes with cost <=C* is finite
- If the branching factor is finite
- A* expands no nodes with f(n) > C*, known as pruning

Time Complexity - $\mathcal{O}(b^{\Delta})$ where the absolute error $\Delta = h^* - h$

A* Search



Is the heuristic designed leads to optimal solution?

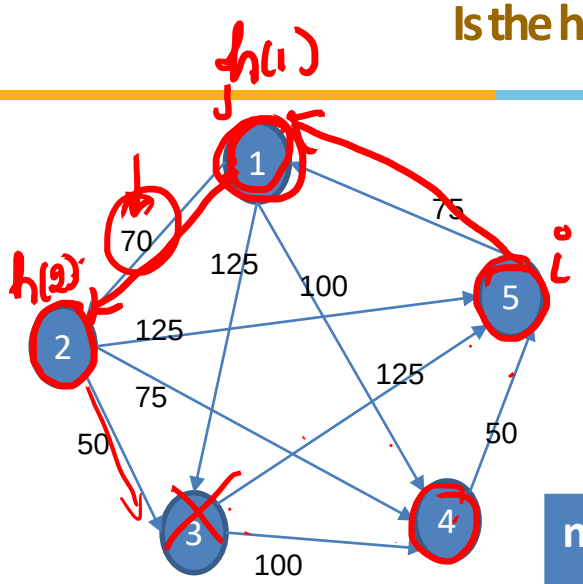


Assuming **node 3** as goal, taking only sample edges per node below is checked for consistency

n	h(n)	Is Admissible? h(n) <= h*(n)	Is Consistent? For every arc (i,j): h(i) <= g(i,j) + h(j)
1	80	F	
2	60	F	
3	0		
4	200	F	
5	190	F	

A* Search → optimal

Is the heuristic designed leads to optimal solution?



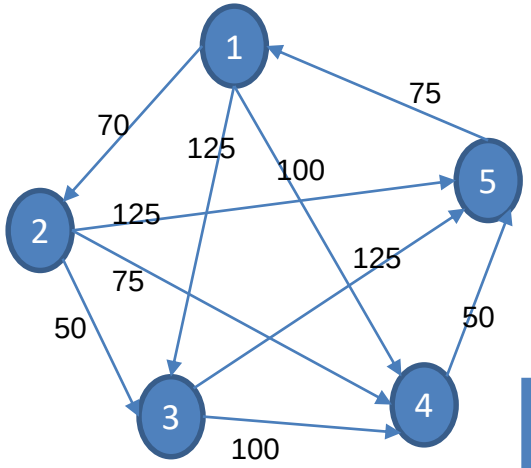
Assuming node 3 as goal, taking only sample edges per node below is checked for consistency

n	h(n)	Is Admissible? h(n) <= h*(n)	Is Consistent? For every arc (i,j): h(i) <= g(i,j) + h(j)
1	80	Y	N (5→1) : 190<=155
2	60	N	Y (1→2) : 80<=130
3	0	Y	
4	200	Y	Y (1→4) : 80<=300 Y (2→4) : 60<=275
5	190	Y	Y (2→5) : 60<=315 Y (4→5) : 200<=240

A* Search

innovate achieve lead

Is the heuristic designed leads to optimal solution?



Assuming node 3 as goal, taking only sample edges per node below is checked for consistency

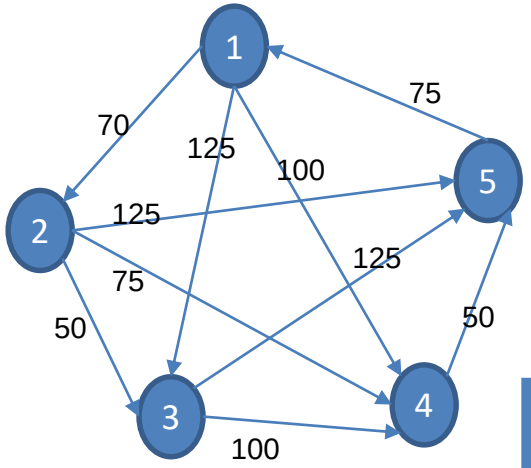
DE1 DE2 DE3

$h_1(n)$ $h_2(n)$ $h_3(n)$

n	h(n)	Is Admissible? h(n) <= h*(n)	Is Consistent? For every arc (i,j): h(i) <= g(i,j) + h(j)
1	80	Y	N (5→1) : 190<=155
2	60	N	Y (1→2) : 80<=130
3	0	Y	
4	200	Y	Y (1→4) : 80<=300 Y (2→4) : 60<=275
5	190	Y	Y (2→5) : 60<=315 Y (4→5) : 200<=240

A* Search

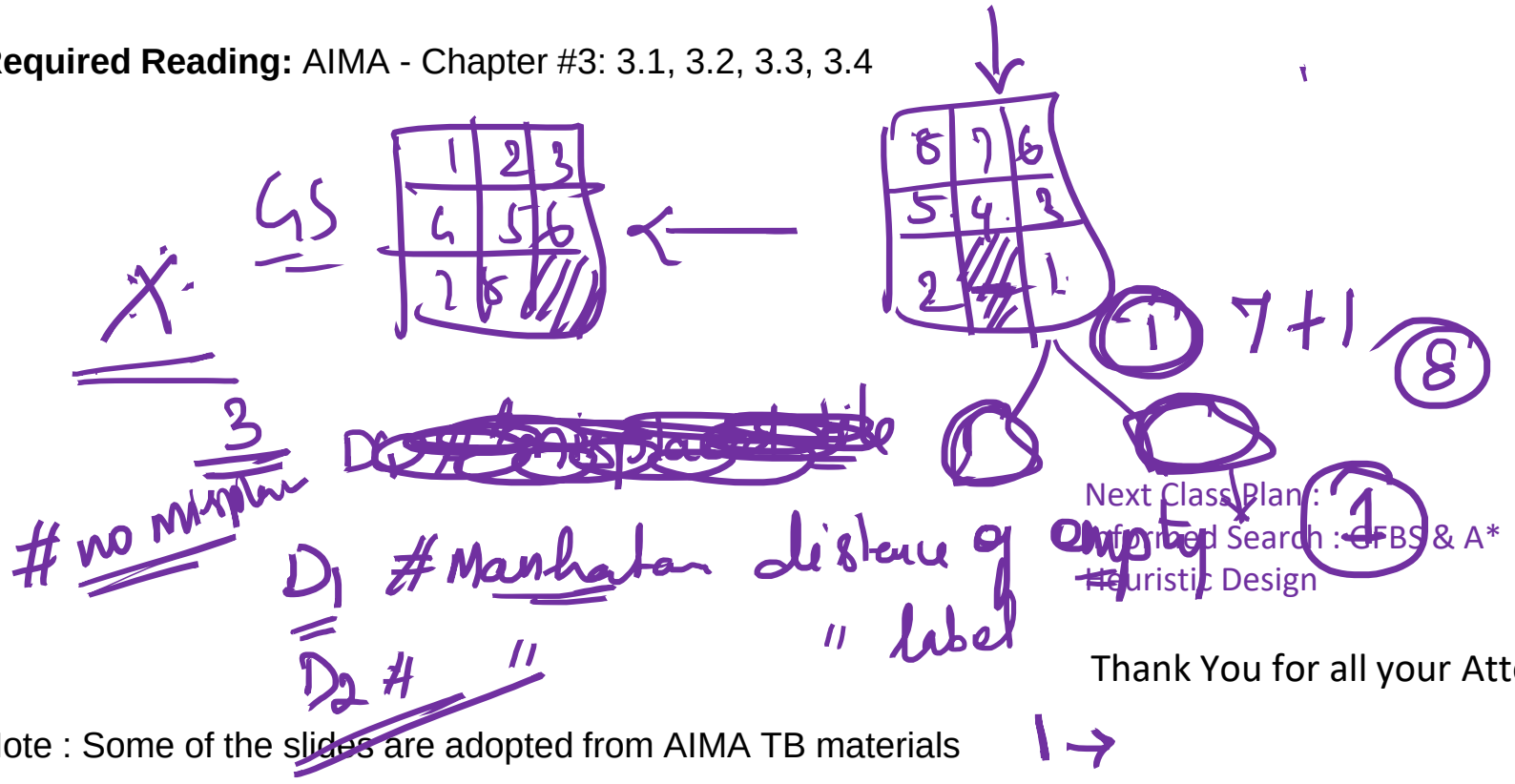
Is the heuristic designed leads to optimal solution?



Assuming node 3 as goal, taking only sample edges per node below is checked for consistency

n	h(n)	Is Admissible? h(n) <= h*(n)	Is Consistent? For every arc (i,j): h(i) <= g(i,j) + h(j)
1	80	Y	N (5→1) : 190<=155
2	60	N	Y (1→2) : 80<=130
3	0	Y	
4	200	Y	Y (1→4) : 80<=300 ✓ Y (2→4) : 60<=275
5	190	Y	Y (2→5) : 60<=315 Y (4→5) : 200<=240

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



Next Class Plan :
Informed Search : ~~GFBS~~ & A*
~~He~~uristic Design

Thank You for all your Attention

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