Assignment-1

BFS (Breadth-First Search)

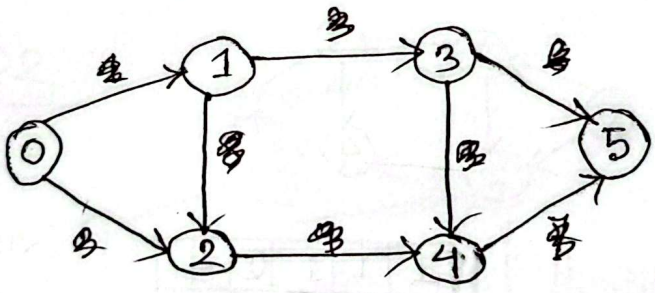
BFS is an uninformed search strategy which used to explore nodes & edges of a graph or tree systematically. It works by FIFO technique

Steps of BFS:

i) Start with the source node and enqueue it.
ii) Dequeue a node from the queue, mark it as visited, and enqueue all its unvisited neighbors.
iii) Repeat step (ii) until the queue is empty.

Example: 1

Step: 0

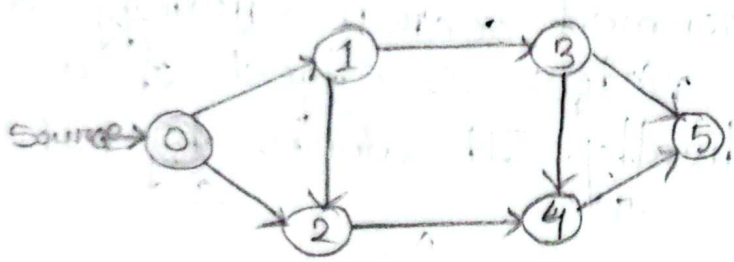


visited[ ] = | 0 | 0 | 0 | 0 | 0 | 0 |

rest[ ] =

queue :

step: 1



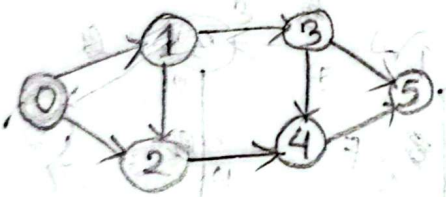
visited[ ] = | 1 | 0 | 0 | 0 | 0 | 0 |

rest[ ] = | | | | | | |

queue = | 0 | | | | |

front

step - 2



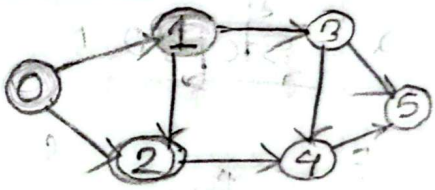
visited[ ] = | 1 | 1 | 1 | 0 | | |

rest[ ] = | 0 | | | | | |

queue = | 1 | 2 | | | |

front

step - 3



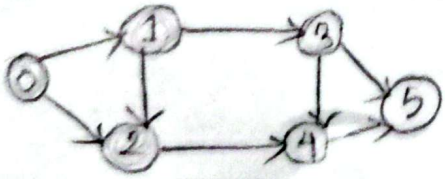
visited[ ] = | 1 | 1 | 1 | 1 | 0 | 0 |

rest[ ] = | 0 | 1 | | | | |

queue = | 2 | 3 |

front

step - 4

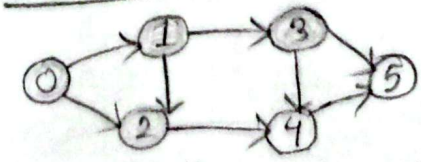


visited[ ] = | 1 | 1 | 1 | 1 | 1 | 0 |

rest[ ] = | 0 | 1 | 2 | | | |

queue = | 3 | 4 |

step - 5

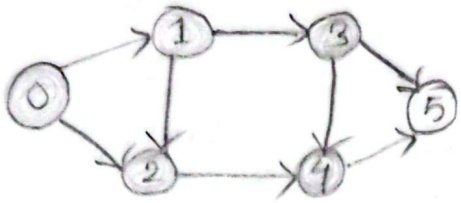


visited[ ] = | 1 | 1 | 1 | 1 | 1 | 1 |

rest[ ] = | 0 | 1 | 2 | 3 | | |

queue = | 4 | 5 |

step-6

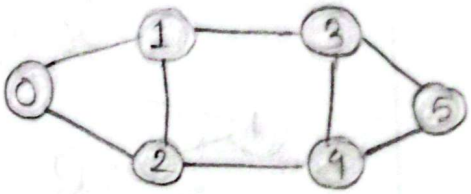


visited [ ] = | 1 | 1 | 1 | 1 | 1 | 1 |

rest [ ] = | 0 | 1 | 2 | 3 | 4 | |

queue = | 5 | | | | | |

step-7



visited [ ] = | 1 | 1 | 1 | 1 | 1 | 1 |

rest [ ] = | 0 | 1 | 2 | 3 | 4 | 5 |

queue = | | | | | | |

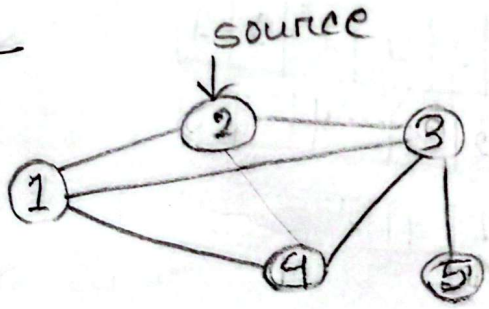
visited array → 0 means unvisited

→ 1 means visited

Final Output : [0, 1, 2, 3, 4, 5]

Example – 02

step-0

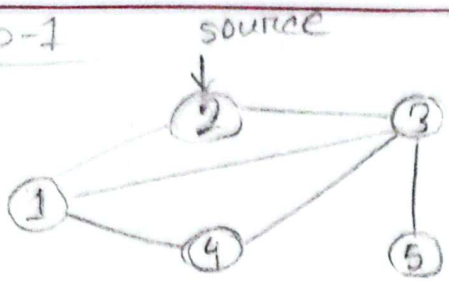


visited [ ] = | 0 | 0 | 0 | 0 | 0 |

rest [ ] = | | | | | | |

queue = | | | | | | |

step-1

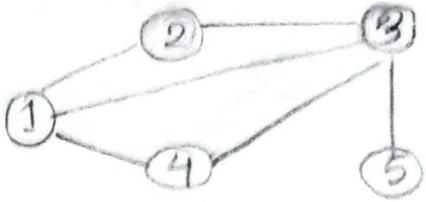


visited[ ] = | 1 | 0 | 0 | 0 | 0 |

rest[ ] = | | | | | |

queue = | 2 | | | | |

step-2

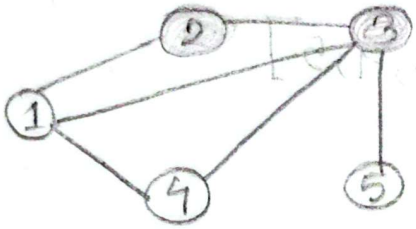


visited[ ] = | 1 | 1 | 1 | 0 | 0 |

rest[ ] = | 2 | | | | |

queue = | 3 | 1 |

step-3

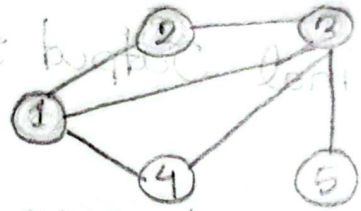


visited[ ] = | 1 | 1 | 1 | 1 | 1 |

rest[ ] = | 2 | 3 | | | |

queue = | 1 | 4 | 5 |

step-4

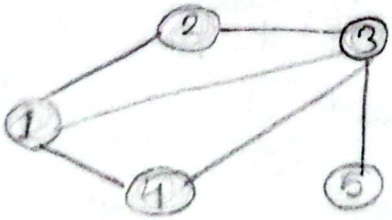


visited[ ] = | 1 | 1 | 1 | 1 | 1 |

rest[ ] = | 2 | 3 | 1 | | |

queue = | 4 | 5 |

step-5

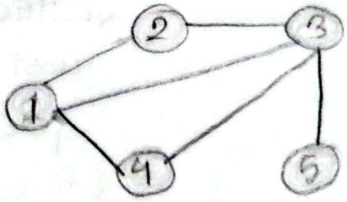


visited[ ] = | 1 | 1 | 1 | 1 | 1 |

rest[ ] = | 2 | 3 | 1 | 4 | |

queue = | 5 | | | | |

step-6



visited[ ] = | 1 | 1 | 1 | 1 | 1 |

rest[ ] = | 2 | 3 | 1 | 4 | 5 |

queue = | | | | | |

Our final Output: [2, 3, 1, 4, 5]

Complexity of BFS:

I) Time Complexity: O (V+E)

II) Space (queue & visited): O(V)

Here, V = vertex/node
E = edge

Applications of BFS:

I) Shortest path in unweighted graphs.

II) Solving Puzzles: Explore all possible moves layer by layer.

III) Broadcasting in Networks

DFS (Depth First Search)

DFS is an uninformed search strategy ~~that~~ which used to traverse through data structure like trees and graphs.

Steps of DFS:

I) Start at the root node.

II) Visit the node and mark it as visited.

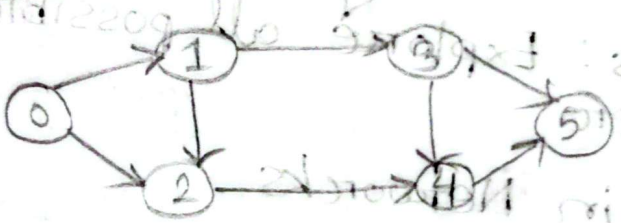
iii) Recursively (or using LIFO (stack)) visit the next unvisited neighbor.

iv) Continue this process until ~~all nodes are visited~~. a node with no unvisited neighbours.

v) Backtrack to the previous node until all nodes are ~~unvisited~~. visited.

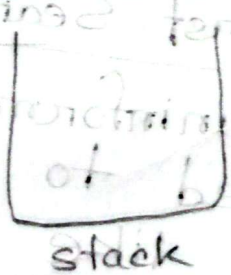
Example -1

step -0

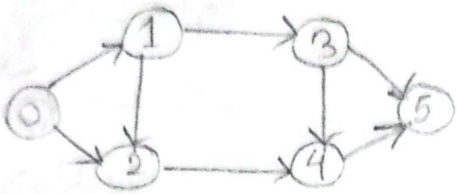


visited[ ] = | 0 | 0 | 0 | 0 | 0 | 0 |

rest[ ] =

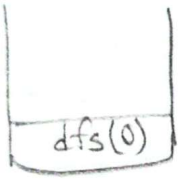


step-1

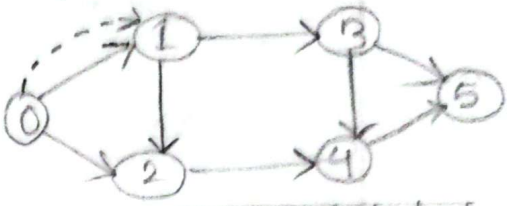


visited[ ] = | 1 | 0 | 0 | 0 | 0 | 0 |

rest[ ] = | 0 | | | | | |

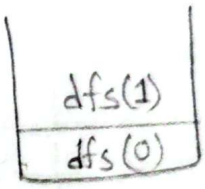


step-2

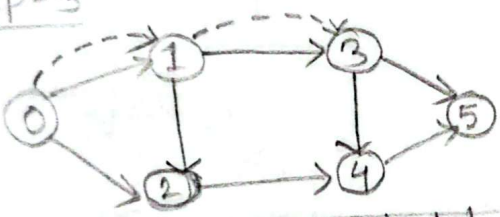


visited[ ] = | 1 | 1 | 0 | 0 | 0 | 0 |

rest[ ] = | 0 | 1 | | | | |

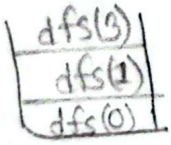


step-3

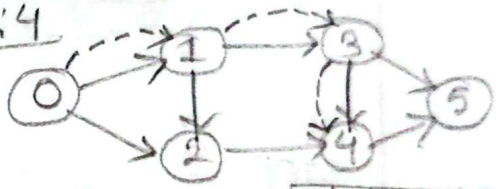


visited[ ] = | 1 | 1 | 0 | 1 | 0 | 0 |

rest[ ] = | 0 | 1 | 3 | | | |

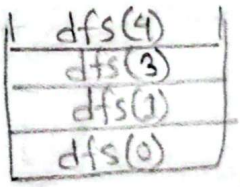


step:4

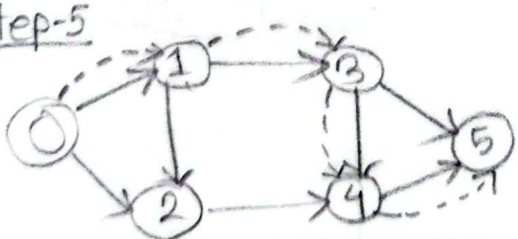


visited[ ] = | 1 | 1 | 0 | 1 | 1 | 0 |

rest[ ] = | 0 | 1 | 3 | 4 | | |

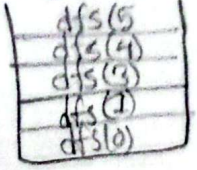


step-5

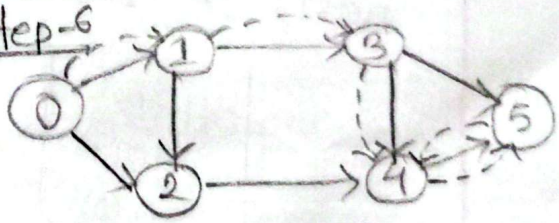


visited[ ] = | 1 | 1 | 0 | 1 | 1 | 1 |

rest[ ] = | 0 | 1 | 3 | 4 | 5 | |

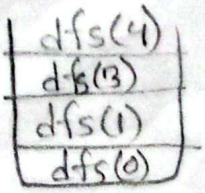


step-6

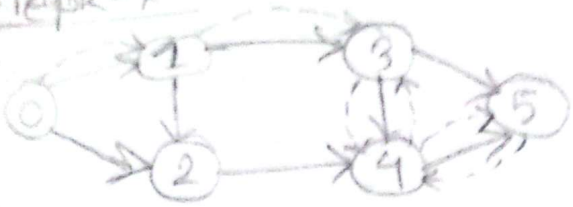


visited[ ] = | 1 | 1 | 0 | 1 | 1 | 1 |

rest[ ] = | 0 | 1 | 3 | 4 | 5 | |

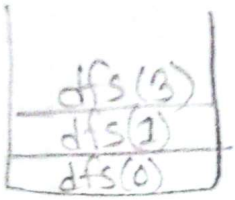


step-7



visited[ ] = | 1 | 1 | 0 | 1 | 1 | 1 |

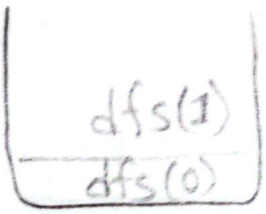
rest[ ] = | 0 | 1 | 3 | 4 | 5 | |



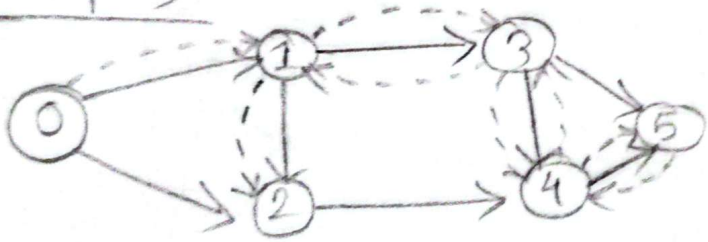
step-8

visited[ ] = | 1 | 1 | 0 | 1 | 1 | 1 |

rest[ ] = | 0 | 1 | 3 | 4 | 5 | |

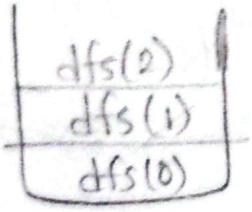


step-9

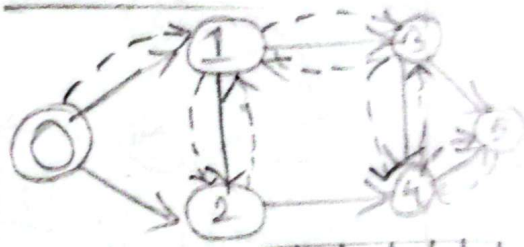


visited[ ] = | 1 | 1 | 1 | 1 | 1 | 1 |

rest[ ] = | 0 | 1 | 3 | 4 | 5 | 2 |

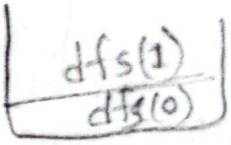


step-10

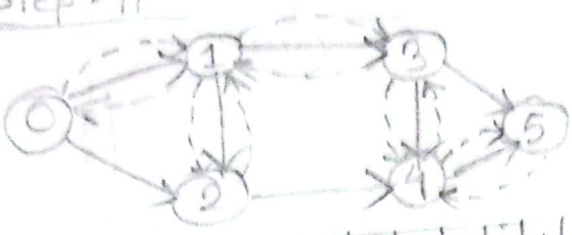


visited[ ] = | 1 | 1 | 1 | 1 | 1 | 1 |

rest[ ] = | 0 | 1 | 3 | 4 | 5 | 2 |

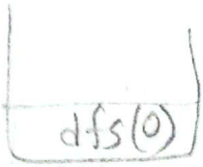


step - 11

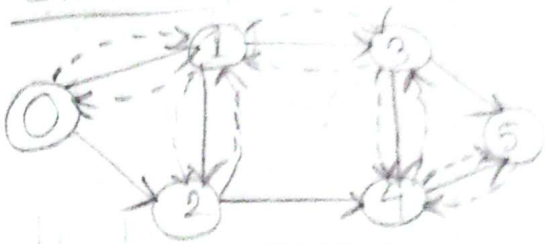


visited[ ] = | 1 | 1 | 1 | 1 | 1 | 1 |

rest[ ] = | 0 | 1 | 3 | 4 | 5 | 2 |



step - 12



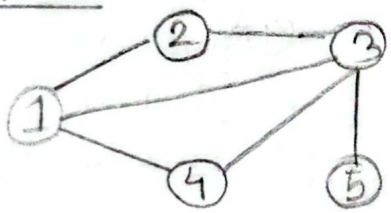
visited[ ] = | 1 | 1 | 1 | 1 | 1 | 1 |

rest[ ] = | 0 | 1 | 3 | 4 | 5 | 2 |

* Final output : [0, 1, 3, 4, 5, 2]

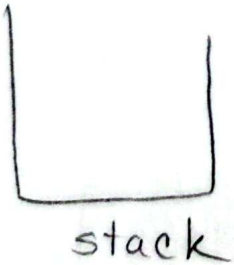
Example - 02

step - 0

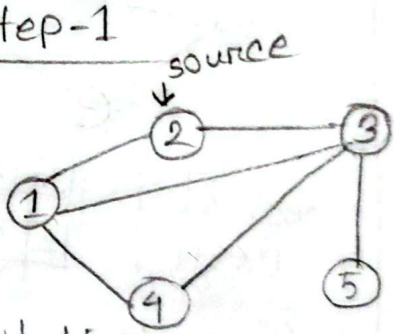


visited[ ] = | 0 | 0 | 0 | 0 | 0 |

rest[ ] = | | | | | |

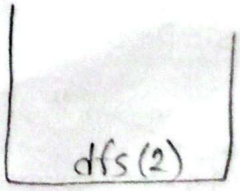


step - 1

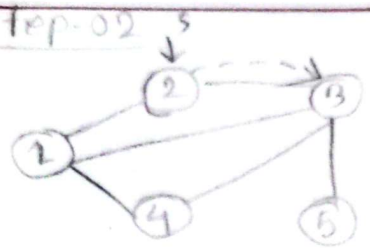


visited[ ] = | 0 | 1 | 0 | 0 | 0 |

rest[ ] = | 2 | | | | |



step-02

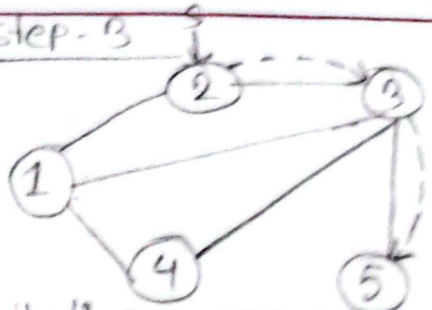


visited[ ] = | 0 | 1 | 1 | 0 | 0 |

rest[ ] = | 2 | 3 | | | |

dfs(3)
dfs(2)

step-3

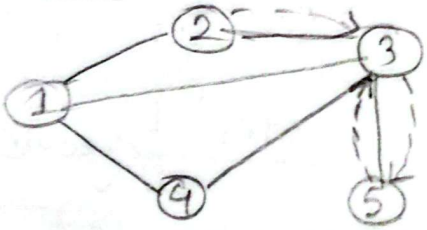


visited[ ] = | 0 | 1 | 1 | 0 | 1 |

rest[ ] = | 2 | 3 | 5 | | |

dfs(5)
~~dfs(3)~~
dfs(2)

step-4

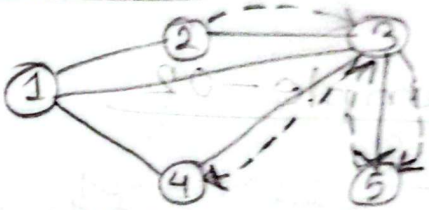


visited[ ] = | 0 | 1 | 1 | 0 | 1 |

rest[ ] = | 2 | 3 | 5 | | |

dfs(3)
dfs(2)

step-5

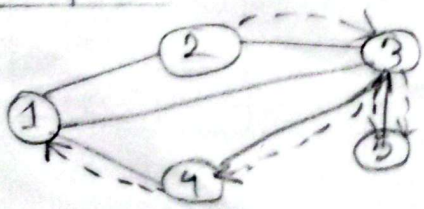


visited[ ] = | 0 | 1 | 1 | 1 | 1 |

rest[ ] = | 2 | 3 | 5 | 4 | |

dfs(4)
dfs(3)
dfs(2)

step-6

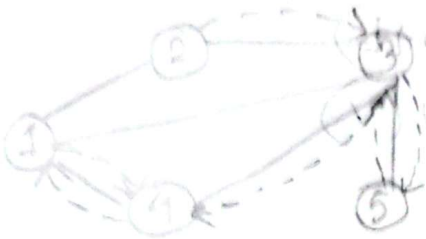


visited[ ] = | 1 | 1 | 1 | 1 | 1 |

rest[ ] = | 2 | 3 | 5 | 4 | 1 |

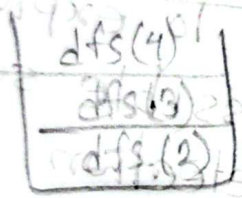
dfs(1)
dfs(4)
dfs(3)
dfs(2)

step-7



visited[ ] = | 1 | 1 | 1 | 1 | 1 |

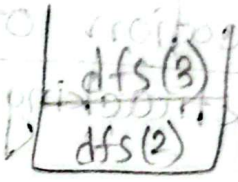
rest[ ] = | 2 | 3 | 5 | 4 | 1 |



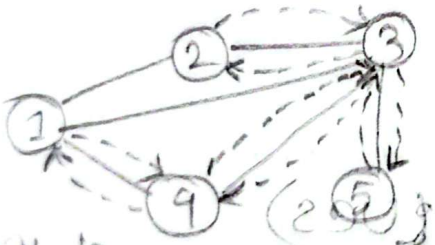
step-8

visited[ ] = | 1 | 1 | 1 | 1 | 1 |

rest[ ] = | 2 | 3 | 5 | 4 | 1 |

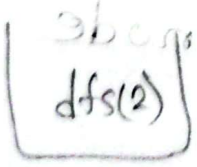


step-9

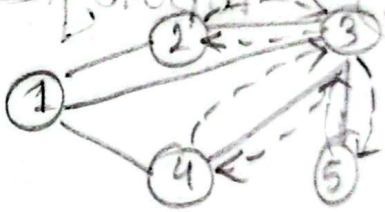


visited[ ] = | 1 | 1 | 1 | 1 | 1 |

rest[ ] = | 2 | 3 | 5 | 4 | 1 |

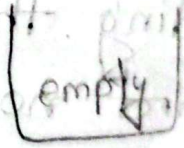


step-10



visited[ ] = | 1 | 1 | 1 | 1 | 1 |

rest[ ] = | 2 | 3 | 5 | 4 | 1 |



Final Output: [2, 3, 5, 4, 1]

~~Time~~ Complexity of DFS:

i) Time complexity : $O(V+E)$

ii) Space complexity : $O(V)$

Here, $V \rightarrow$ vertex
$E \rightarrow$ edge

Application of DFS:

i) Backtracking & Puzzles : To explore all possible solutions.

ii) Cycle & connectivity detection

iii) Topological sorting & SCCs.

Uniform Cost Search (UCS)

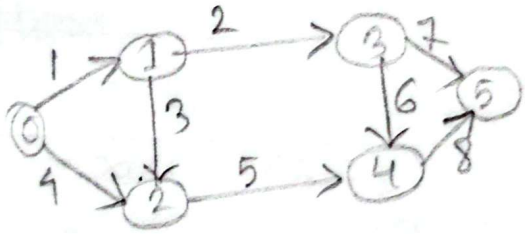
UCS is a search algorithm used for finding the least-cost path from a start node to a goal node in a weighted graph.

~~Example 1~~

Steps of UCS:

i) Finds the least-cost path from a start node to a goal node in a weighted graph.
ii) Explores nodes with the lowest cost first.
iii) Uses priority queue to manage nodes
iv) Repeat step (ii) until we get our goal node.

Example – 01



Find the minimum path from 0 to 5 node.

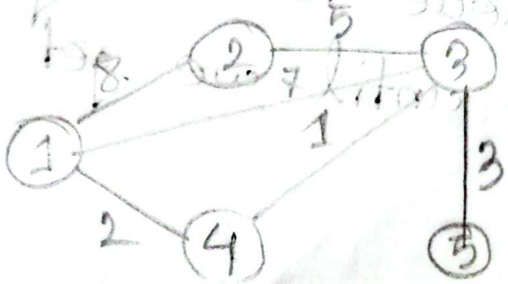
0	1	2	3	4	5
(0,0)	(1,0)	(4,0)	∞	∞	∞
(0,0)		(4,0)	(2,1)	∞	∞
		(4,0)		(6,3)	(7,3)
				(6,3)	(7,3)
					(7,3)

Minimum Path: $0 \rightarrow 1 \rightarrow 3 \rightarrow 5$

Cost = 1 + 2 + 7

= 10

Example-2:



Find the minimum path from 1 to 3

node → / (x,y) ↓	1	2	3	4	5
	(0,0)	(8,0)	∞	(2,0)	∞
		(8,0)	(1,4)		∞
		(8,0)			(3,3)
		(8,0)			

Here, (x,y) → x: cost, y: via node

Path of 1 to 3 = $1 \rightarrow 4 \rightarrow 3$

cost = 2 + 1 = 3

Complexity of UCS:

i) Time complexity: $O(b^{c^*/\epsilon})$ → Tree graph

~~Here, $c^* =$~~

ii) Time complexity: $O(m^{(1+\text{floor}(l/e))})$

Here, m = maximum number of neighbors
l = Length of the shortest path to the goal state
e = least cost of an edge.

* Space complexity: $O(b^{c^*/\epsilon})$

Here, b = branching factor
c^* = cost of the optimal solution path
ϵ = minimum positive edge cost

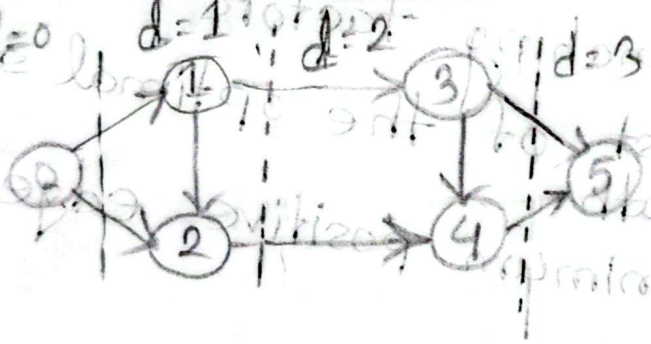
Applications of UCS:

i) Optimal path finding in Maps.
ii) AI & game solving
iii) Network Routing

Iterative Deepening ~~Depth-first~~ Search (IDS):

Iterative Deepening Search (IDS) is a search strategy algorithm that combines the benefits of DFS & BFS. It is used in tree or graph traversal specially where the search space is large.

Example - 01

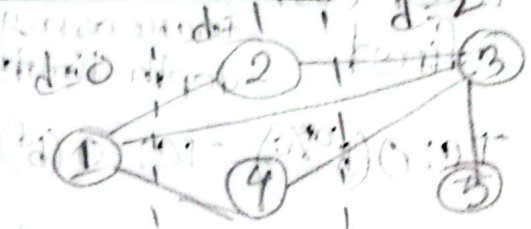


Depth	IDS
0	0
1	0 1 2
2	0 1 3 2 4 3 4
3	0 1 2 3 4 5

Example-02 Goal = 5

Depth	IDS	Goal found
0	0	No
1	0 1 2	No
2	0 1 2 3 2 4	No
3	0 1 2 3 4 5	Yes

Example-02



Goal = 5

Depth	IDS	Goal found
0	1	No
1	1 2 4	No
2	1 2 3 4 5	Yes

Time complexity: $O(b^d)$

Space complexity: $O(d)$

Here, b = branching node

d = depth of the shallowest goal node.

Application:

i) Robotics & Path Planning.
ii) Puzzle solving
iii) Game AI

Comparison among BFS, DFS, UCS & IDS:

BFS	DFS	UCS	IDS
Level-order traversal	Deepest node first	Least cost first	DFS with increasing depth limits
TC: $O(b^d)$	TC: $O(b^m)$	TC: $O(b^{c^*/\varepsilon})$	TC: $O(b^d)$
SC: $O(b^d)$	SC: $O(m)$	SC: $O(b^d)$	SC: $O(d)$
Best for finding shortest path.	Best for finding deep solutions	Best for varying step costs.	Optimal + low memory
Worst for using high memory.	Wost for ~~use~~ may get stuck	slow if many low-cost paths	Repeats work per depth.

Here,
b = branching factor
d = depth of the shallowest solution
m = max depth
c^* = cost of optimal solution, ε = smallest step cost

Situation		Most Optimal Algorithm
i) All actions have equal cost	⟹	BFS or IDS
ii) Actions of different costs	⟹	UCS
iii) Very limited memory	⟹	IDS
iv) Want fast deep solutions	⟹	DFS