Tutorial 5

Name -> Tanvi Nautiyal
Section -> G
Roll No. -> 47

Q.1.

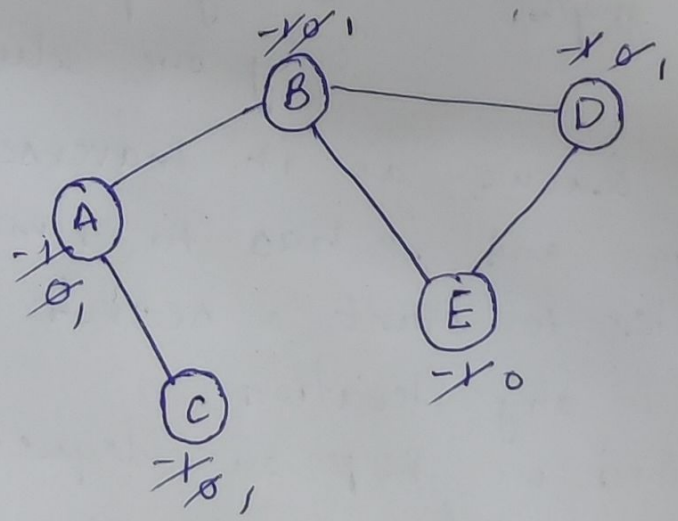
BFS	DFS
• Breadth First Search	• Depth First search
• Uses Queue	• Uses Stack
• Used to find single source shortest path in an unweighted graph.	• Used to find one of the possible path from source to destination.
• There is no concept of backtracking.	• We can backtrack with the help of stack.
• Requires more memory.	• Requires less memory.
• Used in GPS navigation, garbage collection, to find a spanning tree in an unweighted graph.	• Used in topological sorting, scheduling problems, cycle detection in graphs & solving puzzles with only one solution.

Q.2. In BFS we use Queue as it traverses in a breadthwise motion and it has to remember to get the next vertex to start a search, when a dead end occurs in any iteration.
As per the algorithm we keep on dequeueing in order to get all unvisited nodes. When the queue gets emptied the program is over.

8. DFS algorithm traverses a graph in a depthwise motion and uses a stack to remember to get the next vertex to start a search when a dead end occurs in any iteration. At last we do not have any unvisited adjacent nodes so we keep popping the stack until we find a node that has an unvisited adjacent node.

8.3. Dense graph is a graph in which the number of edges is close to the maximal number of edges. Sparse graph is a graph in which the number of edges is close to the minimal no. of edges sparse graph can be a disconnected graph.
It is ideal to represent sparse graph by adjacency list and dense graph by an adjacency matrix.

8.4. Cycle detection in Undirected graph (BFS)



-1 = Unvisited
0 = into the queue
1 = traversed

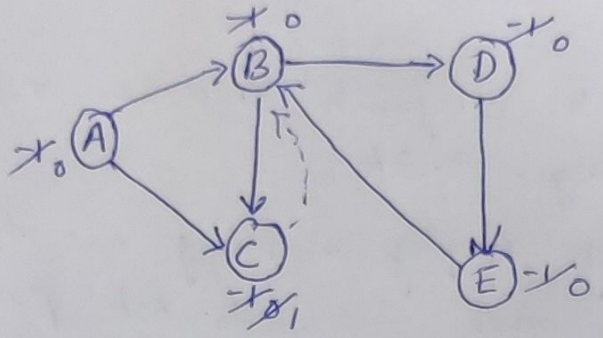
Queue : | A | B | C | D | E |

Visited Set : | A | B | C | D |

When D checks it adjacent vertices it finds E with 0

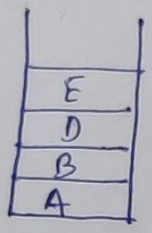
→ If any vertex finds the adjacent vertex with flag 0, then it contains cycle.

Cycle Detection in Directed Graphs (DFS)



-1 = unvisited
0 = visited & in stack
1 = visited & popped out from stack

Stack:



Visited Set:
A B C D E

⇒ B → D → E → B

Here E finds B (adjacent vertex of E) with 0.

→ It contains a cycle.

Parent Map

Vertex	Parent
A	-
B	A
C	B
D	B
E	D

Q.5. The disjoint set data structure is also known as union-find data structure & merge-find set. It is a data structure that contains a coll^n of disjoint or non-overlapping sets.

The disjoint set means that when the set is partioned into the disjoint subsets, various operations can be performed on it.

In this case, we can add new sets, we can merge the sets, & we can also find the representation member of a set. It also allows to find out whether the two elements are in

the same set or not efficiently.

Operation on disjoint set

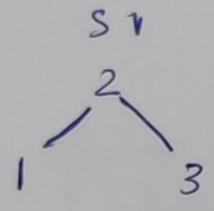
1. Union

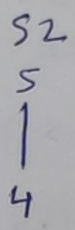
a) If S1 & S2 are two disjoint sets, their union S1 ∪ S2 is a set of all elements x such that x is in either S1 or S2.

b) As the sets should be disjoint S1 ∪ S2 replaces S1 & S2 which no longer exists.

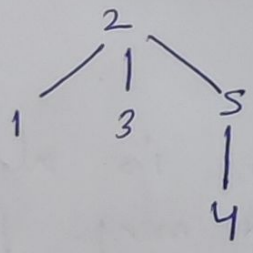
c) Union is achieved by simply making one of the trees as a subtree of other i.e. to set parent field of one of the roots of the trees to other root.

Eg.





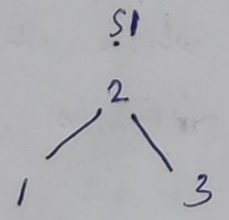
S1 ∪ S2

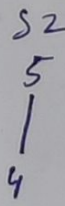


Merge the sets containing x & y into one

Find

Given an element x, to find the set containing it.



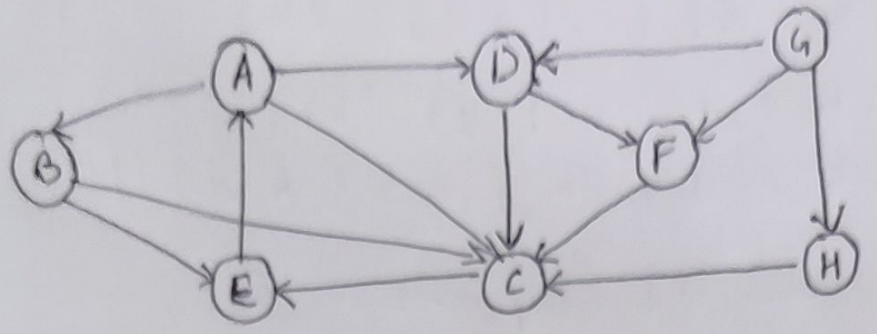


find (3) → S1
find (5) → S2

return in which set x belongs

Make-set (x): creates a set containing x.

Q.6



BFS

Child	G	H	D	F	C	E	A	B
Parent		G	G	G	H	C	E	A

Path : G → H → C → E → A → B

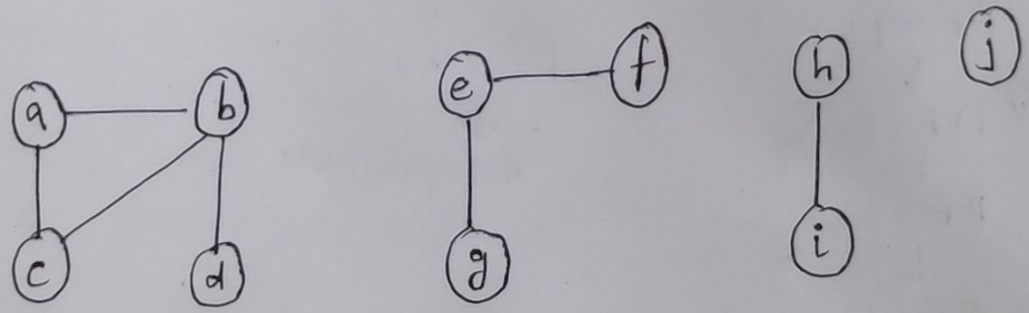
DFS

~~G~~
D
H
~~F~~
~~C~~
~~E~~
~~A~~
~~B~~
} Nodes Visited

G
F
C
E
A
B
} Stack

Path : G → F → C → E → A → B

Q.7.



V = {a} {b} {c} {d} {e} {f} {g} {h} {i} {j}

E = {a, b}, {a, c}, {b, c}, {b, d}, {e, f}, {e, g}, {h, i}, {j}

(a, b)	{a, b} {c} {d} {e} {f} {g} {h} {i} {j}
(a, c)	{a, b, c} {d} {e} {f} {g} {h} {i} {j}
(b, c)	{a, b, c} {d} {e} {f} {g} {h} {i} {j}
(b, d)	{a, b, c, d} {e} {f} {g} {h} {i} {j}

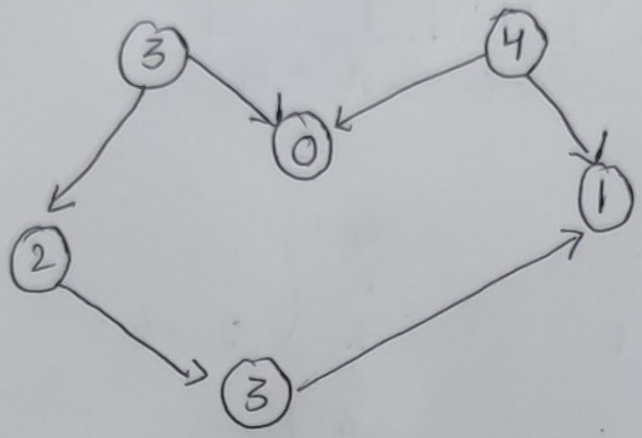
(e,f) {a,b,c,d} {e,f} {g} {h} {i} {j}

(e,g) {a,b,c,d} {e,f,g} {h} {i} {j}

(h,i) {a,b,c,d} {e,f,g} {h,i} {j}

No. of connected components = 3

Q.8



We take source node as 5

Applying Topological sort

DFS (5)

↓

DFS (0)

↓

DFS (2)

↓

DFS (3)

↓

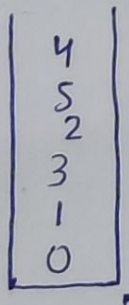
DFS (1)

DFS(4)

↓

Not possible

DFS



4 → 5 → 2 → 3 → 1 → 0

Q9. Heap is generally preferred for priority queue implementation because heap provide better performance compared to arrays or linked list.

Algorithm where priority queue is used:

1. Dijkstra's Shortest Path Algorithm: When the graph is stored in the form of adjacency list or matrix, priority queue can be used to extract minimum efficiently when implementing Dijkstra's algorithm.
2. Prim's Algorithm: To store keys of nodes & extract minimum key node at every step.

Q.10.

Min Heap	Max Heap
• For every pair of the parent & descendant child node, the parent node always has lower value than descendant child node.	• For every pair of the parent & descendant child node, the parent node has greater value than descendant child node
• The value of nodes increases as we traverse from root to leaf node.	• The value of nodes decreases as we traverse from root to leaf node
• Root node has the lowest value.	• Root node has the greatest value.