Tutorial-5

Q1 What is the difference b/w DFS and BFS. Applications of Both Algo.

BFS	DFS
→ stands for Breadth first search.	DFS stands for Depth first search.
→ BFS is more suitable for searching vertices which are closer to source.	DFS is more suitable for searching vertices away from source.
→ BFS considers all neighbours first, so not suitable for desicion making trees in games/puzzles.	In DFS we make a desicion, then explore all the paths through this descision, if desicion = win, stop, therefore suitable in games/puzzles.
→ siblings are visited before children.	children are visited before sibling.
→ Implemented using FIFO list.	Implemented using LIFO list.
→ Requires more m/m as compared to DFS.	Requires less m/m.
→ No Backtracking required.	Need of Backtracking
→ slower than DFS.	Faster than BFS
→ O(V+E) V= Vertices, E= Edges	O(V+E)
→ visited according to tree level.	visited according to tree depths.

Date.
Page No.

Application of BFS -
→ Peer to Peer Networks
→ Web Crawlers
→ Network Broadcasting.

Application of DFS -
- Detecting cycle in graph
- Topological sort.
- Solving puzzles with only one solution

Q2 Which Datastructure is used to implement and why?
→ BFS → queue
DFS → stack

Q3. What do you mean by sparse and dense graphs? Which representation of graph is better for sparse and dense graphs?

Dense graph is a graph in which the no of edges is close to the maximal no of edges. Adjacency matrix is preferred.

Sparse graph is a graph in which no of edges is close to the minimal no of edges.
sparse graph can be a disconnected graph
Adjacency list is required.

Q4 How can you d
using BFS and
DFS :
DFS for a conne
There is a cycle
a back edge pre
↳ An edge
itself (
in the t

BFS : For every
an adjacent
visited and vis
there is a cycle
of any vertex is
cycle.

Q5 What do you
Explain three op
examples which
disjoint sets.
→ Also known as un
merge find dat
track of set of e
a number of di

Disjoint set can be

there is no common element b/w the two sets.

operations - !

① Making new sets

The MakeSet operation adds a new element

The element is placed into a new set

containing only new element and new set is added to ds.

② Finding the representative of set containing a ~~new~~ given element

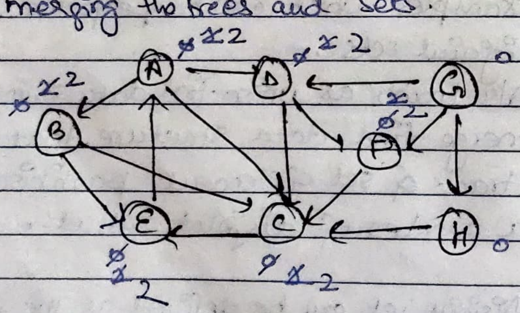
can be implemented by recursively traversing parent array until we hit a node who is parent to itself.

"Find".

③ Union - It takes two elements a input and finds the representatives using "find" operation and puts either one of the trees under the root node of other tree, merging the trees and sets.

Q6

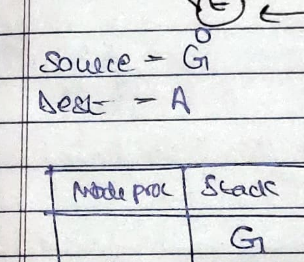
BFS



Node	~~A~~	B
Parent	-	A

* No path b/w

DFS



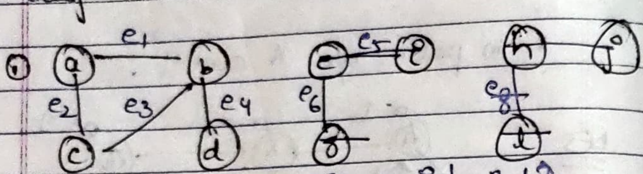
Source - G

Dest - A

Node proc	Stack
	G
G	D F H
D	~~E C F H~~
F	C C F H
C	E C F H
E	A C F H
A	B C F

G → D —

Q7 Find out no of connected components and vertices in each component using disjoint set Data Structure.

(1) 

$U = \{a, b, c, d\}$ ~~e, g, i, h, j, l~~

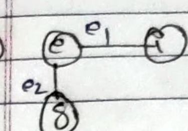
(1) Add e1 (a,b)
find (a) = U
find (b) = U
$S_1 = \{a, b\}$
$U = \{c, d\}$

(2) Add e_2 (a, c)
find (a) = S_1
find (c) = U
$S_1 = \{a, b, c\}$
$U = \{d\}$

(3) Add e_3 (b, c)
$\left.\begin{array}{l}\text{find (b)} = S_1 \\ \text{find (c)} = S_1\end{array}\right\}$ cycle detected

(4) Add e_4 (b, d)
find (b) = S_1
find (d) = U

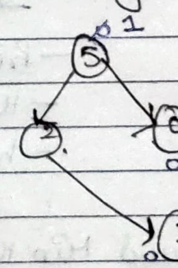
$S_1 = \{a, b, c, d$
$U = \{\phi\}$

(2) 

(1) Add e_1 (e, i)
find (e) = U
$S_1 = \{e, i\}$
$U = \{g\}$

(2) Add e_2 (e, g
find(e) = S_1
find (g) = U
$S_1 = \{e, i, g\}$

Q8 Apply topologic
graph having



(1) Topological sor
(2) DFS

Date.
Page No.

Node	Stack
	5
5	20
2	30
3	10
1	0

5→2→3→1

Q9. Heap DS can be used to implement priority queue? Name few graph algo where you need to use priority queue and why?

Yes, we can use heaps to implement the priority queue.

O(nlogn) to insert and delete.

Priority queue is used in Dijkstra's Algo

- Prim's Algo
- Huffman code
- heap sort

Q10 Diff between Max and Min Heap?

Min Heap	Max Heap
In Min Heap, key ~~is~~ present at the root node, and it must be <= all keys present on its children.	Key ~~present is~~ present at root must be >= keys present at all of its children.

→ Min key element is present at the root

→ Ascending priority

→ Smallest element is popped first.

→ Get Maximum Element, Extract Maximum and insert.