Date.
Page No.

Tutorial 5

1. Difference b/w BFS and DFS.

BFS

→ Stands for breadth first search.

→ uses queue data structure for finding path.

→ gives shortest path from source to destination.

→ Considers all the neighbours of source node and then their neighbours.

→ Time Complexity - list : $O(V+E)$
Matrix : $O(V^2)$

→ No Concept of back tracking as it do not uses recursion.

Application: 1. Shortest path
2. MST for unweighted graph
3. Peer to peer networks
4. Social Networking Sites

Date.
Page No.

5. GPS navigation
6. Cycle detection indication.

DFS :

→ stands for depth first search.

→ uses stack data structure.

→ gives one of the possible paths from source node to destination node.

→ Considers a source node, then considers its source node till the end.

→ Time Complexity: List: $O(V+E)$
Matrix: $O(V^2)$

↔ has concept of backtracking.

Application :

1. Topological Sorting
2. Finding Strongly connected components of graph.
3. Path finding

Date.
Page No.

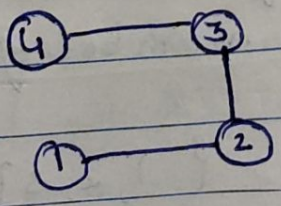
2. Which data structures are used to implement BFS & DFS.

Data structures used in case of BFS & DFS are queue & stack respectively.

→ BFS requires ~~~~ queue data structure because it traverses the graph in breadthward motion and uses a queue to remember to get the next vertex to start a search when a dead end occurs in any iteration.

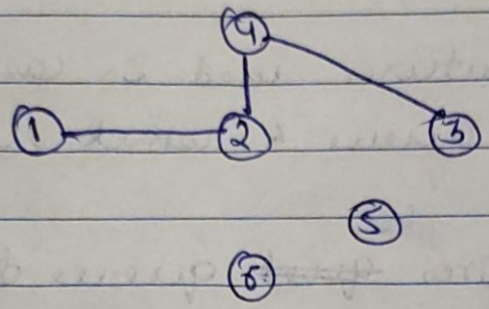
→ DFS requires stack data structure because it traverse ~~~~ a graph in a depthward motion & uses a stack to remember it to get to the next vertex to start a search, when a dead end occurs in iteration.

3. Dense graph is a graph in which number of edges is close to maximal no. of edges.



Date.
Page No.

Sparse graph is a graph in which no. of edges is close to minimal no. of edges.



→ It is ideal to use sparse graph by adjacency list and dense graph by adjacency matrix.

4. Disjoint Set Data Structure:

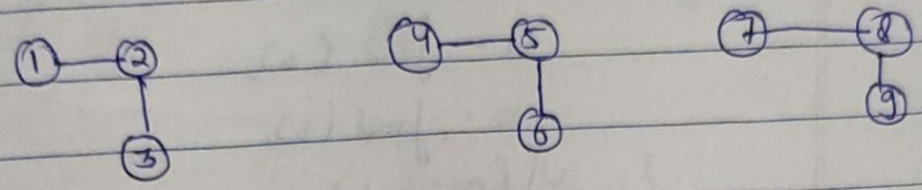
A disjoint set data structure also called as union find data structure or merge. Find data structure is a data structure that stores a partition of set into disjoint subsets.

It provides operations for adding sets, merging sets and finding a representative member of a set.

Eg. $S1 = \{1, 2, 3\}$
$S2 = \{4, 5, 6\}$

Date.
Page No.

S3 = {7, 8, 9}



Operations Performed:

(i) **Find:** It can be implemented recursively by traversing the parent array until we hit a node who is parent to itself. Here, path comparision can be achieved by keeping track or size of node.

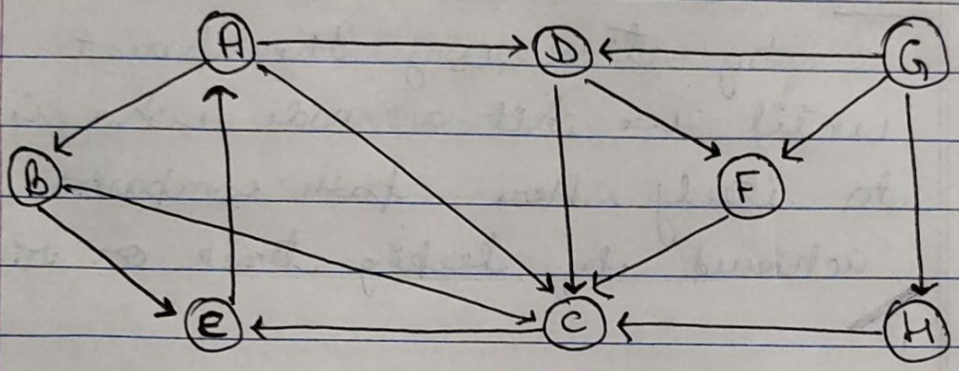
```
int find (int i)
{
    if ( parent[ i ] == i )
        return i;
    return parent [i] = find ( parent [i] );
}
```

(ii) **Union:** It takes two elements as input and finds the parent of the inputs using find operation and performs merging of the one child to the parent node.

Date.
Page No.

```
void union (int a, int b)
{
    a = find (a);
    b = find (b);
    if (a == b)
        parent [a] = b;
}
```

6



BFS

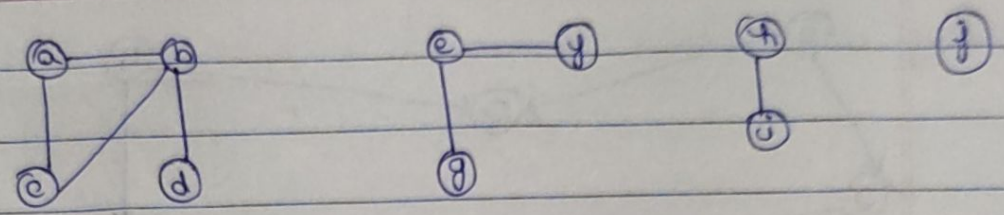
Node	B	E	C	A	D	F
Parent	-	B	B	E	A	D

DFS

B	E	A	D	F	C

Date.
Page No.

7.



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

Here, make, find and union operations will be performed.

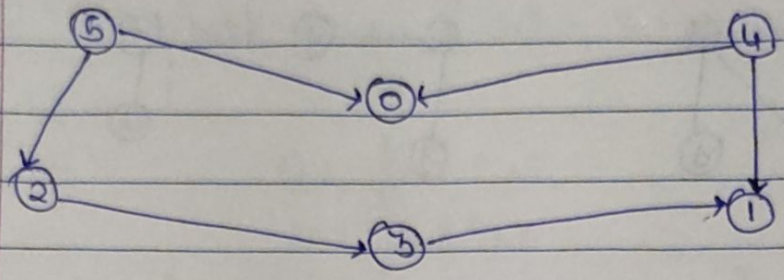
→ First each vertex will be parent of itself

→ If an edge exists between vertices, then we perform find operation & check if their parent is same and perform union operation on them.

$S_1 = \{ a, b, c, d \}$
$S_2 = \{ e, f, g \}$
$S_3 = \{ h, i \}$
$S_4 = \{ j \}$

Date.
Page No.

8.



toposort (0): push (0)

toposort (1): push (1)

toposort (2): toposort (3)

first push(3), then push 2

toposort (4): push (4)

toposort (5): push (5)

∴ 5 4 2 3 1 0

9. We can use heaps to implement the priority queue. It will take O(log N) time to insert & delete each element in priority queue.

→ Based on heap structure, priority queue also two types - max & min priority

Date.
Page No.

Some algorithm's where we need to use priority queue :

i) Dijkstra's :

→ used for calculating shortest path in algorithm.
→ When the graph is stored in the form of adjacency list or matrix, priority queue can be used to extract efficiently.

ii) Prim's : used to implement prim's algo to store keys of nodes and extract minimum key node at every step.

iii) Data Compression :

→ Used in Huffman's coding which is used to compress data.

Date.
Page No.

10.	Min heap	Max heap
•	In a min heap the key present at root must be less than or equal to the children.	In max heap, key at root must be greater than the key at the children
•	Min key element is at root of heap.	Max key element is at root of heap.
•	Used ascending priority.	Uses descending priority
•	Smallest element has a priority.	Largest element has a priority
•	Smallest element is to be popped from heap.	Largest element is to be popped from heap.