NAME: JIGYASU DEO

Univ. Roll No. : 2015245

SECTION : M.L

CLASS Roll No : 32

SUBJECT : DESIGN AND ANALYSIS OF ALGORITHM

SUBJECT CODE : TCS 505

Tutorial - 5

Ans 1:

BFS	DFS
1. BFS Stands For Breadth First Search uses Queue data Structure for Finding the Shortest Path.	1. DFS Stands For Depth First Search uses Stack data Structure.
2. BFS can be used to find Single Source Shortest path in an unweighted graph, because in BFS, we reach a vertex with minimum number of edges from a Source.	2. In DFS, we might traverse through more edges to reach a destination vertex from a Source.
3. Siblings are visited before the children.	3. Children are visited before the Siblings.

Applications

BFS	DFS
1. Shortest Path and Minimum Spanning Tree for unweighted graph.	1. Path Finding
2. Peer to Peer Networks	2. Topological Sorting
3. Cycle Detection in undirected graph.	3. To test if a graph is bipartite

Ans 2 = BFS does the Search for nodes level-by-level i-e it Searches the nodes with respect to their distance from root. Here, Siblings are visited before children.

We use Queue as it is FIFO data Structure, We visit the node which is discovered first from the root.

For DFS, we retrieve it from root to the farthest node as much as possible, Some idea as LIFO. Therefore we use Stack data Structure. Here Children are visited before the Siblings.

Ans 3 = A graph with relatively few edges is sparse

Sparse Graph is a graph G(V,E) in which $|E| = O(|V|)$

Edge Vertex

A graph with many edges is dense

Dense Graph is a graph G (V,E) in which $|E| = O(|V^2|)$

Adjacency List can be used for Sparse Graphs

whereas Adjacency Matrix can be used for Dense Graphs.

Ans 4= **Detect A Cycle in a Directed Graph using BFS :-**

1. Compute in-degree (number of incoming edges) for each of the vertex present in the graph and initialize the count of visited nodes as 0.

2. Pick all the vertices with in-degree as 0 and add them into a queue (Enqueue Operation).

3. Remove a vertex from the Queue (Dequeue operation) and then

   i) Increment count of visited nodes by 1

   ii) Decrease in-degree by 1 for all its neighboring nodes.

   iii) If in-degree of a neighboring nodes is reduced to zero, then add it to the queue.

4. Repeat Step 3 until the Queue is empty.

5. If count of visited nodes is not equal to the number of nodes in the graph has cycle, otherwise not.

Detect A Cycle in a Directed Graph using DFS :-

1. Create the graph using the given number of edges and vertices.

2. Create a recursive function that initializes the current index or vertex, visited, and recursion stack.

3. Mark the current node as visited and also mark the index in recursion stack.

4. Find all the vertices which are not visited and are adjacent to the current node. Recursively call the function for those vertices, if the recursive function returns true, returns true.

5. If the adjacent vertices are already marked in the recursion Stack then return true.

6. Create a wrapper class, that calls the recursive function for all the vertices and if any function returns true return true. Else if for all vertices the function returns false return false.

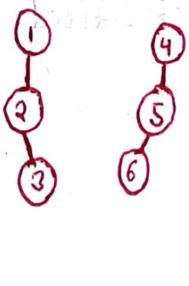
Ans 5: Disjoint Set is basically as group of Sets where no item can be in more than one Set. It Supports union and find operation on Subsets.

Assume that you have a Set of N elements that are into further Subsets and you have to track the connectivity of each element in a Specific Subset or connectivity of Subsets with each other. You can use the union-find algorithm (disjoint Set union) to achieve this.

Operations on Disjoint Set :-

$S_1 = \{1,2,3\}$

$S_2 = \{4,5,6\}$



Find() : It is used to find in which Subset a particular element is in and returns the representative of that particular Set.

$Find(1) = S_1$

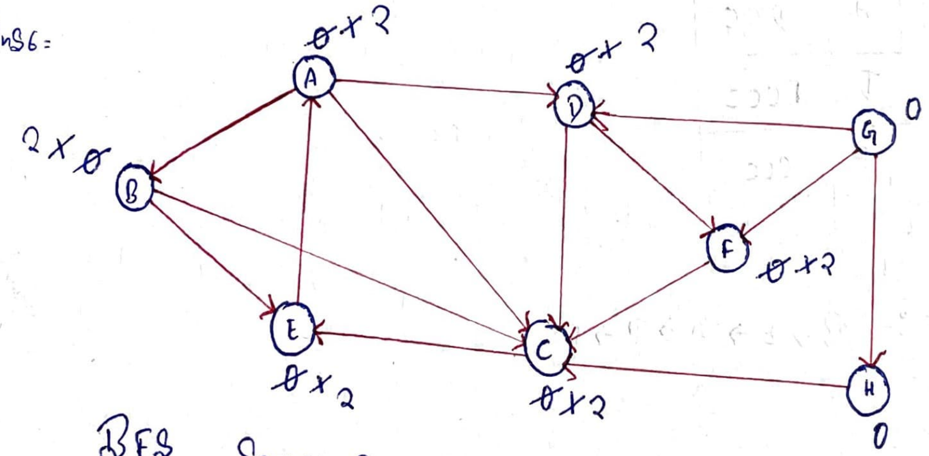
$Find(5) = S_2$

Union() : It merges two different Subsets into a Single Subset and representative of one Set becomes representative of other.

$S_1 \cup S_2$

$S_3 = \{1, 2, 3, 4, 5, 6\}$

Ans6=

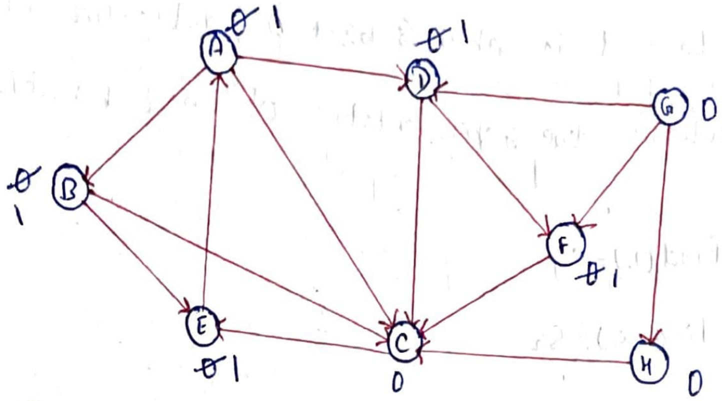


BFS Source = B Destination = F

Queue

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

Path : B→E→A→D→F



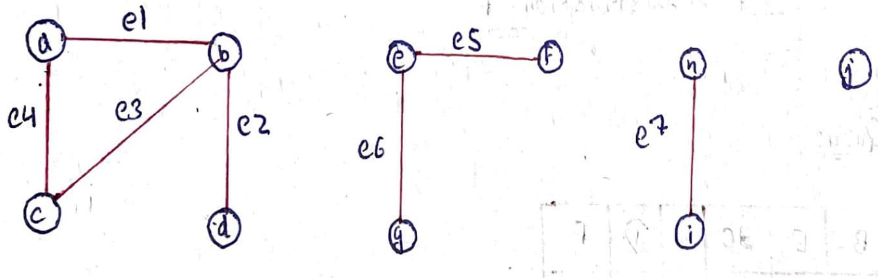
DFS Source = B Destination = F

Stack

NODE PROCESSED	STACK
-	B
B	EC
E	AC
A	DCC
D	FCCC
F	CCC

Path :- B → E → A → D → F

Ans 7



a	b	c	d	e	f	g	h	i	j
~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~	-1
~~-2~~	a	a	~~b~~	-3	e	e	-2	h	
~~-3~~	~~-1~~		a						
-4									

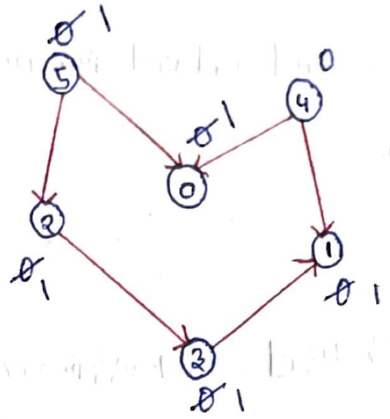
Not connected to anyone

4 connected to a

3 connected to e

2 connected to h

Ans 8:-



Topological Sort

5, 2, 4, 0, 3, 1

DFS

Source = 5

NODE PROC	STACK
-	5
5	20
2	30
3	10
1	0
0	-

STACK

Ans 9:- Heaps are great for implementing a priority queue because of the largest and smallest element at the root of the tree for a max-heap and a min-heap respectively.

We use a max-heap for a max-priority queue and a min-heap for a min-priority queue.

Applications:-

i) Dijkstra's Shortest Path Algorithm using priority queue:

When the graph is stored in the form of adjacency list or matrix, priority queue can be used to extract minimum efficiently when implementing Algo.

ii) Prim's Algorithm : It is used to implement Prim's Algo. to store keys of nodes and extract minimum key node at every step.

iii) Data Compression : It is used in Huffman codes which is used to compress data.

Ans 10=

Min Heap	Max Heap
1. In a min - Heap the key present at the root node must be less than or equal to among the keys present at all of its children.	1. In a Max-Heap the key present at the root node must be greater than or equal to among the keys present at all of its children.
2. In a Min-Heap the minimum key element present at the root.	2. In a max-Heap the maximum key element present at the root.
3. A Min-Heap uses the ascending priority.	3. A Max-Heap uses the descending priority.