Design & Analysis of Algorithms

Tutorial-5

Name- Satyam Kumar
Section-D
Roll no:- 34
2017004.

Ans1

BFS:- BFS stands for Breadth First Search, is a vertex-based technique for finding the shortest path in the graph. It uses a Queue data structure that follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and stored in the queue.

DFS:- It stands for Depth first Search, is an edge-based technique. It uses the stack data structure and performs two stages, first visited vertices are pushed into the stack, and second if there are no vertices then visited vertices are popped.

BFS	DFS
• In BFS, we reach a vertex with minimum number of edges from a source vertex.	• In DFS, we might traverse through more edges to reach a destination vertex from a source.
• Time Complexity of BFS is $O(V+E)$ [Adj. List], $O(V^2)$ [Adj. Matrix].	• Time Complexity of DFS is, $O(V+E)$ [Adj. List] $O(V^2)$ [Adj. Matrix]
• There is no concept of backtracking	• It is a recursive algorithm that uses idea of backtracking.
• It requires more memory.	• It requires less memory.

• Applications:-

→ DFS:-

- Cycles in a graph may be detected using DFS.
- A path may be find between u and v vertices.
- It may be used to perform topological sorting.

→ BFS:-

- BFS is used to find all neighbour nodes,
- Using GPS navigation system BFS is used to find neighbouring places.
- In networking, to broadcast packets, BFS is used.

Ans 2

• BFS:-

Queue data structure, based on First In First Out, is used to implement BFS (Breadth First Search).

BFS algorithm traverses a graph in a 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. Queue will ensure that those things that were discovered first will be explored first, before exploring those that were discovered subsequently.

• DFS:-

DFS algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration. For keeps tracking on the current node it requires the depth of a node then all the nodes will be popped out of stack. Next it searches for adjacent nodes which are not visited yet.

Ans 3

A dense graph is a graph in which the number of edges is close to the maximal number of edges.

The sparse graph is a graph with very few edges.

- For sparse graphs, adjacency list representation is good.
- For dense graphs, adjacency matrix representation is good.

Ans 4

The existence of cycle in directed and undirected graph can be determined by whether depth-first search (DFS) finds an edge that points to an ancestor of the current vertex (it contains a back edge). All the back edges which DFS skips are part of cycles.

Ans 5

Disjoint Set data structure :-

- It allows to find out whether the two elements are in the same set or not efficiently.
- The disjoint set can be defined as the subsets where there is no common element b/w the two sets.
- Operations performed :-

i) Find :- can be implemented by recursively traversing the parent array until we hit a node who is parent to itself.

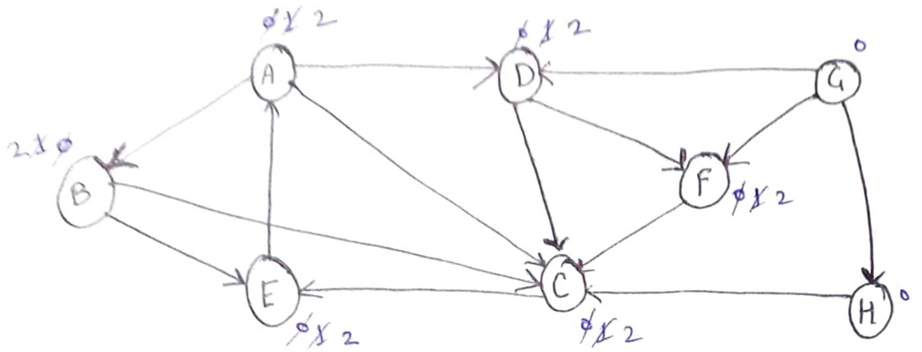
```
int find (int i)
{
 if (parent [i] == i)
  {
    return i;
   }
  else
  {
    return find (parent [i]);
   }
  }
```

ii) Union :- It takes, as input, two elements. And finds the representatives of their sets using the find operations, and finally puts either one of the trees (representing the set) under the root node of the other tree, effectively merging the trees, and the sets.

```
Void union (int i, int j)
{
   int irep = this. Find (i);
   int jrep = this. Find (j);
   this parent [irep] = jrep;
 }.
```

iii) Path Compression (Modifications to Find()):- It speeds up the data structure by compressing the height of the tree. It can be achieved by inserting a small caching mechanism into find operation.

Ans 6



B.F.S :-

Node :-	B	E	C	A	D	E
Parent :-	—	B	B	E	A	D

Unvisited Nodes :- G H

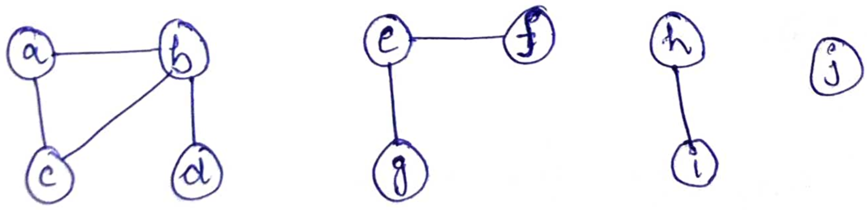
Path = B ⟶ E ⟶ A ⟶ D ⟶ E

DFS :-

Node Processed :-	B	B	C	E	A	D	F
Stack :	B	CE	EE	AE	DE	FE	E

Path :- B ⟶ C ⟶ E ⟶ A ⟶ D ⟶ F

Ans 7



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

⇒ $S_1 = \{a\}$

edge $(a,b) \Rightarrow$ $S_1 = \{a, b\}$

$(b,d) \Rightarrow$ $S_1 = \{a, b, d\}$

$(a,c) \Rightarrow$ $S_1 = \{a, b, c, d\}$

$S_2 = \{e\}$

$(e,i) \Rightarrow S_2 = \{e, f\}$

$(e,g) \Rightarrow S_2 = \{e, g, f\}$

$S_3 = \{h\}$

$(h, i) \Rightarrow \{h, i\}$

$S_4 = \{j\}$

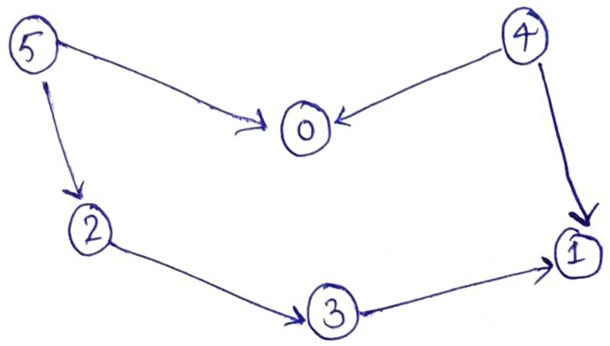
$\Rightarrow \{a, b, c, d\}\{e, f, g\}\{h, i\}\{j\}$

$\Rightarrow$ Connected components = 3

Non-connected Components = 1

Total = 3+1 = 4.

Ans 8



$\Rightarrow$ 5, 4, 2, 3, 1, 0.

Ans 9. We can use heaps to implement the priority queue. It will take O(log N) time to insert and delete each element in the priority queue. Based on heap structure, priority queue has also two types — max priority queue and min priority queue. Some algorithms where we need to use priority queue are:-

i) Dijkstra's Shortest path algorithm using priority queue: when the graph is sorted in the form of adjacency list or matrix, priority queue can be used to extract minimum efficiently when implementing Dijkstra's algorithm.

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

iii> Data Compression:- It is used in Huffman's Code which is used to compress data.

Ans 10:-

Min Heap	Max Heap
• In a min heap the key present at the root must be less than or equal to among the keys present at all of its children.	• 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.
• The minimum key element present at the root.	• The maximum key element present at the root.
• Uses the ascending priority	• Uses descending priority.
• In a construction of min heap, the smallest element has priority.	• In the construction, the largest element has priority.