TUTORIAL - 5

Que 1) What is difference between DFS and BFS. Also write application of both.

Ans

BFS	DFS
(i) BFS stands for Breadth First Search.	(ii) DFS stands for Depth First Search
(ii) BFS uses Queue data structure for finding shortest path	(ii) DFS uses Stack data structure
(iii) BFS is more suitable for searching vertices which are closer to the given source	(iii) DFS is more suitable when there are solutions away from source.
(iv) 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 vertex	(iv) In DFS, we might traverse through more edges to reach a destination vertex from a source.
(v) The Time Complexity of BFS is $O(V+E)$ when Adjacency List is used and $O(V^2)$ when adjacency matrix is used	(v) The Time Complexity of DFS is $O(V+E)$ when Adjacency List is used and $O(V^2)$ when adjacency matrix is used.

Application of BFS

(i) BFS is used for detecting cycles in a graph.

(ii) finding shortest path and minimal spanning trees in unweighted graph.

(iii) In networking, to find a route for packet transmission.

(iv) finding a route through GPS navigation system with minimum number of crossings.

(v) In building the index by search engine crawlers.

Application of DFS

(i) It is used for detecting cycle in a graph.

(ii) Used to find a path between two given vertices.

(iii) To test if a graph is bipartite.

(iv) finding strongly connected components of a graph.

(v) Solving puzzles with only one solution such as mazes.

Que 2) Which Data Structures are used to implement BFS and DFS and why?

Ans BFS uses Queue data Structure for finding the Shortest path and DFS uses Stack data Structure.

Que 3) What do you mean by sparse & dense graphs? Which representation of graph is better for sparse & dense graphs?

Ans Sparse Graph - A graph in which the number of edges are much less than the possible number of edges. In this we should store it as a list of edges.

Dense Graph - A graph in which the number of edges is close to the maximal number of edges. In this we should store it as adjacency matrix.

Que 4) How can you detect a cycle in a graph using BFS and DFS?

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

DFS can be used to detect a cycle in a graph. DFS for a connected graph produces a tree. There is a cycle in a graph only if there is a back edge that is from a node to itself or one of its ancestors in the tree produced by DFS.

Detect Cycle in an Undirected graph

Run a DFS for every invisited node. DFS can be used to detect a cycle in a graph. DFS for a connected graph produces a tree. There is a cycle in a graph only if there is a back edge present in the graph.

A back edge is an edge that is joining a node to itself (self-loop) or one of its ancestors in the tree produced by DFS.

To find the back edge to any of its ancestors keep and visit array. And if there is a back edge to any visited node then there is a loop and returns true.

Que 5) What do you mean by disjoint set data structure? Explain 3 operations with examples.

Ans) Disjoint Set data structure is also known as union-find data structure & merge-find set.

It is a data structure that contains a collection of disjoint or non-overlapping sets.

The disjoint set means that when the set is partitioned into the disjoint subsets. The various operations can be performed on the disjoint sets.

Operations —:

(i) find - Can be implemented by recursively traversing the parent array until we hit a node who is parent of 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 operation, & finally puts either one of the trees under the root node of the other tree, effectively merging the trees & the sets.

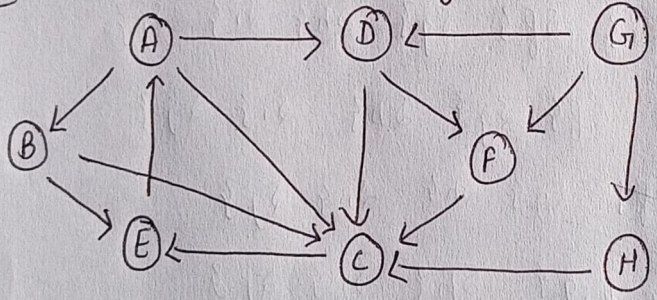
```
void union ( int i , int j )
{
    int irep = this.find(i)
    int jrep = this.find(j);
    this.Parent[irep] = jrep;
}
```

(ii) Path Compression - It speeds up the data structure by compressing the height of the trees.

It can be achieved by inserting a small caching mechanism into the find operation.

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

Que 6) Run BFS & DFS on graph shown -:



BFS

Rear →

Node	B	C	E	A	D	F
Parent	—	B	B	E	A	D

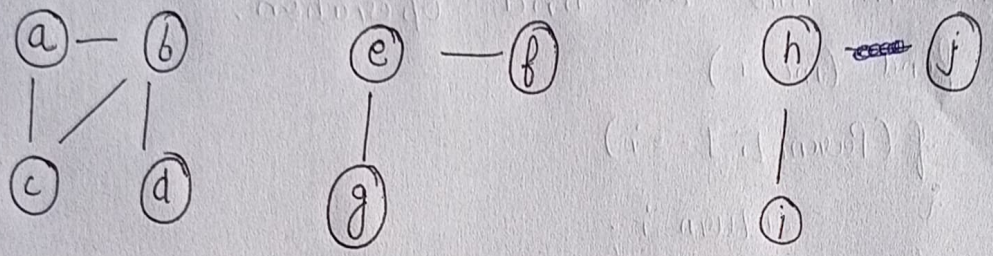
B → C → E → A → D → F

DFS

Node Processed	Stack ←
	B
B	C E
C	E E
E	A E
A	D E
D	F E
F	

B → C → E → A → D → F

Que 7) Find out the number of connected components & vertices in each component using disjoint set data structure.



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

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

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

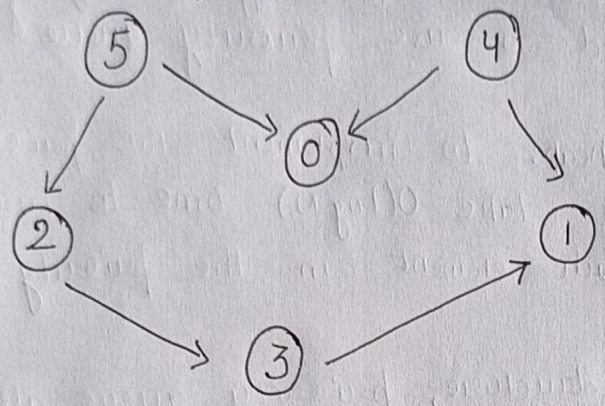
(e,f) $\{a,b,c,d\}\{e,f\}\{g\}\{h\}\{i\}\{j\}$

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

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

Number of connected components = 3

Que 8) Apply Topological Sorting and DFS on graph having vertices from 0 to 5.



Adjacent List

0 →
1 →
2 → 3
3 → 1
4 → 0, 1
5 → 2, 0

Visited

0	1	2	3	4	5
false	false	false	false	false	false

S1: Topological Sort (0), visited [0] = true; list is empty, no more recursion call
stack [0]

S2: Topological Sort (1), visited [1] = true; list empty, no more recursion call
stack [0 | 1]

S3: Topological Sort (2), visited [2] = true
Topological Sort (3), visited [3] = true
'1' is already visited; No more recursion call
Stack [0 | 1 | 3 | 2]

S4: Topological Sort (4), visited [4] = true
'0', '1', already visited, No more recursion call
Stack [0 | 1 | 3 | 2 | 4]

S5: Topological Sort (5), visited [5] = true
'2' & '0' already visited. No more recursion call
Stack [0 | 1 | 3 | 2 | 4 | 5]

S6: Print all elements of stack from top to bottom
5, 4, 2, 3, 1, 0

Que 9) Heap data structure can be used to implement priority queue? Name few graph algorithms where you need to use priority queue & why?

Ans 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 also has two types max-priority queue and min-priority queue.

Algorithms where we can use priority queue:-

(i) 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 while 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 Codes which is used to compresses data.

Que 10) What is the difference between Max and Min Heap?

Min Heap	Max Heap
(i) 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.	(i) 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.
(ii) Minimum key element is present at the root	(ii) Maximum key element is present at the root.
(iii) It uses the ascending priority.	(iii) It uses the descending priority.
(iv) In the construction of Min Heap, the smallest element has priority	(iv) In the construction of Max Heap, the largest element has priority.
(v) In this, the smallest element is the first to be popped from the heap	(v) In this, the largest element is the first to be popped from the heap.