Akinjay Aggarwal I-35 Tutorial-5

Qn 1-

BFS

- Stands for Breadth First Search
- It uses queue to find shortest path
- It is better when target is closer to source
- As BFS consider all neighbours so it is not suitable for decision

DFS

- Stands for Depth First Search.
- It uses stack to find shortest path.
- It is better when target is far away from source
- It is more suitable for decision tree. As with one decision we need to traverse further to argument the decision. If we search the conclusion.

Application of DFS

- Using DFS, we can find the b/w two vertices.
- We can perform topological sorting which is used to schedule jobs.
- We can use DFS to detect cycles
- Using DFS, we can find strongly connected components of a graph.

Application of BFS

- BFS may also used to detect cycles
- Finding shortest path and minimal spanning tree in unweighted graphs.
- In networking, finding a route for packet transmission.

- finding a route through GPS navigation system.

Q2- Breadth First Search (BFS) uses queue data structure.
In BFS, you mark any node in the graph as source node and start traversing from it. BFS traverses all nodes in the graph and keeps dropping them as completed. BFS visited an adjacent unvisited node, marks it as done and insert it into Queue.
DFS uses stack data structure because DFS traverses a graph in a depthwise motion and uses stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

Q3- Sparse graph
A graph in which the number of edges is much less than the possible number of edges.

Dense graph
A dense graph is a graph in which the number of edges is close to the maximum number of edges ~~in~~.

If the graph is sparse, we should store it as list of edges.
Alternatively, if a graph is dense, we should store it as adjacency matrix.

Q4. DFS can be used to detect cycle in a graph.

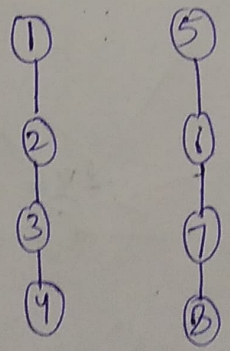
DFS for a connected graph produces a tree. There is a cycle in a graph only if there is a back edge present in the graph. A back edge is an edge that is from a node to itself or one of its ancestor in the tree produced by DFS.

BFS can be used to detect cycles. Just perform BFS, while keeping a list of previous nodes at each node visited or else constructing a tree from the starting node. If I visit a node that is ~~from~~ already marked by BFS, I found a cycle.

Q5 Disjoint set Data structure

- It allows you to find out whether the two elements are in the same set or not efficiently.
- A disjoint set can be defined as the subset when there is no common elements b/w the two sets.

Example: S1 = {1, 2, 3, 4}
S2 = {5, 6, 7, 8}.



Operations Performed

[i]. find:

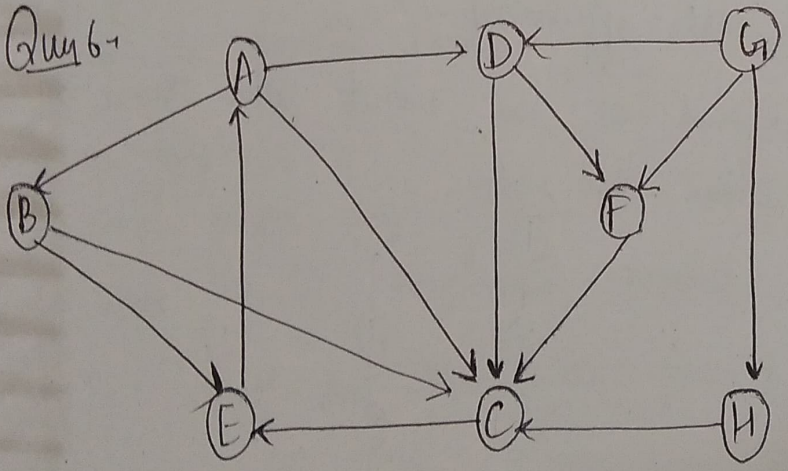
```
int find(int v)
{
    if (v == parent[v])
        return v;
```

```
    return parent[v] = find(parent[v]);
}
```

Union

```
void Union(int a, int b)
{
    a = find(a);
    b = find(b);
    if (a != b)
    {
        if (size[a] < size[b])
        {
            swap(a,b)
        }
        parent[b] = a;
        size[a] += size[b];
    }
}
```

Qus 6-



BFS

Node:	B	E	C	A	D	F
parent:	-	B	B	E	A	D

path: B → E → A → D → F

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

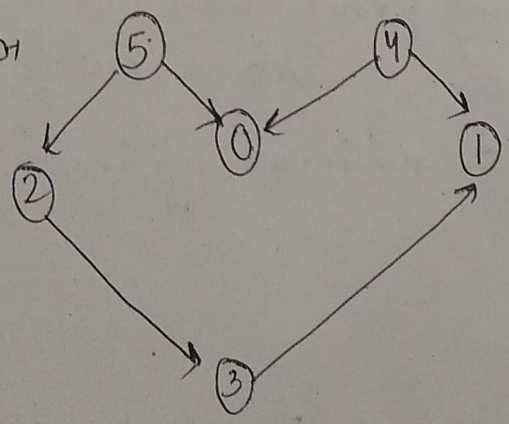
Q7 → V = {a} {b} {c} {d} {e} {f} {g} {h} {i} {j}

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

Q8 →



Adjacency 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

stack (Empty)

Step1: Topological sort (0), visited[0] = true

stack | 0 |

Step2: Topological sort(1), visited[1] = true

stack | 0 | 1 |

Step3: Topological sort (2), visited[2] = true

↓

Topological sort (3), visited[3] = true

stack | 0 | 1 | 3 | 2 |

Step4: stack | 0 | 1 | 3 | 2 | 4 |

Step5: Stack | 0 | 1 | 3 | 2 | 4 | 5 |

Step6: Print all elements of stack from top to bottom

5, 4, 3, 2, 1, 0

Q9. Algorithms that uses Priority Queue

[i] Dijkstra's shortest path Algorithm using priority Queue

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

[ii] Prim's Algo

It is used to implement prims algo to store key of nodes to extract minimum key node at every step.

[iii] Data Compression

It is used in Huffman's code which is used to compress data

Q10 Min heap

[i] In min heap, the key present at root node must be less than or equal to among the key present all its children.

[ii] Uses the ascending priority

[iii] The minimum key prsent at the root node.

Max heap

[i] In max heap, the key present at root node must be greater than or equal to the key present at all its children.

[ii] Uses descending priority.

[iii] The maximum key present at the root node.