Assignment - 3

1. What is the difference between DFS and BFS. Write the applications of both the algorithms.

BFS	DFS
i) BFS stands for Breadth First Search.	DFS stands for Depth First Search.
ii) BFS uses queue data structure for finding the shortest path.	DFS uses stack data structure.
iii) 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.	In DFS, we might traverse through more edges to reach a destination vertex from a source.
iv) BFS is more suitable for searching vertices which are closer to given source.	DFS is more suitable when there are solutions away from source.
v) BFS considers all neighbours first and therefore not suitable for decision making trees used in games or puzzles.	DFS is more suitable for game or puzzle problems. We make a decision, then explore all paths through this decision. And if this decision leads to win situation, we stop.
vi) T.C. of BFS is $O(V+E)$ when Adj. List is used and $O(V^2)$ when Adj. Matrix is used.	T.C. of DFS is also $O(V+E)$ when Adj. List is used and $O(V^2)$ when Adj. Matrix is used.
vii) Here siblings are visited before the children.	Here, children are visited before the siblings.

viii) In BFS, there is no concept of backtracking.	DFS algorithm is a recursive algorithm that uses idea of backtracking.
ix) BFS is used in various applications such as bipartite graph, and shortest path etc.	DFS is used in various applications such as acyclic graph and topological order, etc.
x) BFS require more memory.	DFS requires less memory.

Applications of BFS:

1) Shortest path and Minimum spanning tree for unweighted graph.

2) Peer to Peer networks: In Peer to Peer Networks like BitTorrent BFS is used to find all neighbour nodes.

3) Crawlers in Search Engine: Crawler use build index using Breadth first. The idea is to start from source page and follow all links from source and keep doing same. DFS can be used, but the advantage with BFS is, depth or levels of the built tree can be limited.

4) Social Networking Websites: In social networks, we can find people within a given distance 'k' from a person using BFS till 'k' levels.

5) GPS navigation systems: BFS is used to find all neighbouring locations.

6) Broadcasting in Network: In networks, a broadcaste packets follows BFS to reach all nodes.

7) Cycle detection in undirected graph.

8) To test graph is bipartite.

Applications of DFS:

i) Detecting cycle in a graph: A graph has cycle if and only if we see a back edge during DFS. So we can run DFS for the graph and check for back edges.

ii) Path finding:

iii) Topological Sorting

iv) To test if graph is bipartite

v) Finding Strongly connected components of a graph

vi) Solving puzzle with only one solution such as mazes.

Q.2. Which Data structures are used to implement BFS and DFS and why?

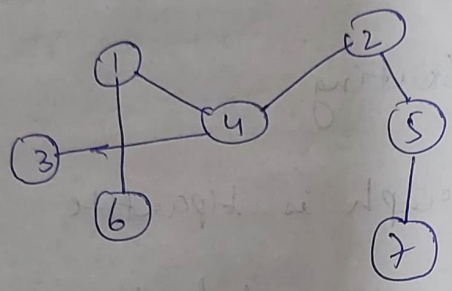
- DFS algorithm traverses the graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

- The data structure used in BFS is queue. The algorithm makes sure that every node is visited not more than once.

3. What do you mean by sparse and dense graphs? Which representation of graph is better for sparse and dense graphs?

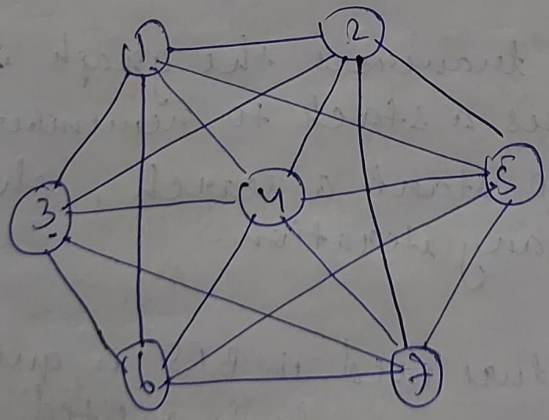
Sparse graph :- Sparse graph is a graph in which the number of edges is close to the minimal number of edges. If the graph is sparse, we should store it as a list of edges.

Eg:



Dense graph: A graph is a graph in which the number of edges is close to the maximal number of edges. If the graph is dense we should store it as an adjacency matrix.

Eg:



8. Detect a cycle in a graph using BFS and DFS.

Cycle in a directed graph using BFS:-

Algo:

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

Step 2: Pick all the vertices with in-degree as 0 and add them into a queue (enqueue operation)

Step 3: Remove a vertex from the queue (Dequeue operation) and then

1. Increment count of visited nodes by 1.
2. Decrease in-degree by 1 for all its neighbouring nodes.
3. If in-degree of a neighbouring nodes is reduced to zero, then add it to the queue.

Step 4: Repeat step 3 until the queue is empty.

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

Cycle in a directed graph using DFS:

Algo:

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

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

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

Step: 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, return true.

Step: 5 If the adjacent vertices are already marked in the recursion stack that returns true.

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

Q.5. What do you mean by disjoint set data structure? Explain 3 operation along with examples, which can be performed on disjoint sets.

- It is also known as union-find data structure and merge-find set. It is a data structure that contains a collection of disjoint or non-disjoint sets. It also allows to find out whether the two elements are in the same set or not efficiently.

Three operations performed on disjoint sets:-

i) Making new sets

```
function Makeset(x) is
    if x is not already in the forest then
        x.parent := x
        x.size := 1      // if nodes store size
        x.rank := 0      // if nodes store rank
    end if
end function
```

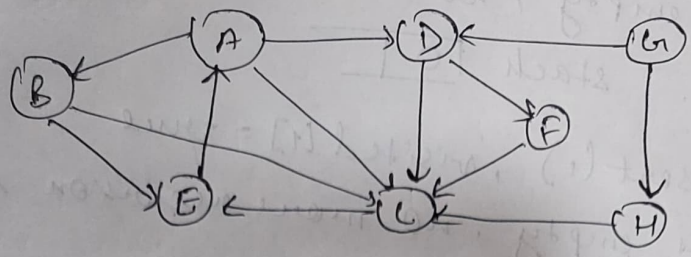
ii) Finding set representative

```
function Find(x) is
    if x.parent ≠ x then
        x.parent := Find(x.parent)
        return x.parent
    else
        return x
    end if
end function
```

iii) Merging two sets

```
function Union(x, y) is
    // Replaces nodes by roots
    x := Find(x)
    y := Find(y)
    if x = y then
        return // x and y are already in the same set.
    end if
    // If necessary, rename variables to ensure that
    // x has at least as many descendants as y.
    if x.size < y.size then
        (x, y) := (y, x)
    end if
    // Make x the new root
    y.parent := x
    // Update the size of x
    x.size := x.size + y.size
end function
```

8.6. Run BFS and DFS on given graph.



BFS:

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

Unvisited nodes - G and H

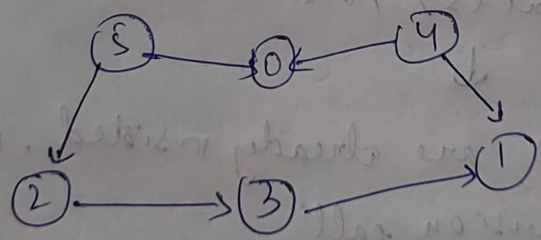
Path → B → E → A → D → F

DFS:-

node processed:	B	B	C	E	A	D	F
stack:	B	CE	FE	AE	DE	FE	E

Path: B → C → E → A → D → F

8.8 Apply topological sorting and DFS on graph having vertices from 0 to 5.



Adj. list 0 →
1 →
2 → 3
3 → 1
4 → 0, 1
5 → 2, 0

visited:

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

stack: empty

Step 1: Topo-sort (0) visited[0] = true.
list is empty, no more recursion calls

stack | 0 |

Step 2: Topo-sort (1), visited[1] = true
list is empty. no more recursion call.

stack | 0 | 1 |

Step 3: Topo-sort (2) visited[2] = true

↓

Topo-sort (3), visited[3] = true
1 is already visited no more recursion call
stack:

| 0 | 1 | 3 | 2 |

Step 4: Topological sort (4), visited[4] = true
0, 1 are already visited. No more recursion call.
stack: | 0 | 1 | 3 | 2 | 4 |

Step 5: Topo-sort (5), visited[5] = true

↓

2, 0 are already visited, no more recursion call

stack | 0 | 1 | 3 | 2 | 4 | 5 |

step 6: Print all elements of stack from top to bottom.

5, 4, 2, 3, 1, 0

Q.9: Heap data structure can be used to implement priority queue? Name few graph algorithms where you need to use priority queue and why?

Ans: Yes, 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. Heaps are great for implementing 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 min-priority queue.

Few graph algorithms:

a) Dijkstra's:- When the graph is stored in the form of adj. list or matrix, priority queue can be used to extract minimum efficiently ~~string~~ when implementing Dijkstra's algorithm.

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

c) Heap Sort: Heap sort is typically implemented using Heap which is implementation of Priority Queue.