Assignment - 5
Data Analysis and Algorithm

Name Harsha saini
Section F
Class roll no. 8

What is the difference between DFS & BFS write application of both the algorithm.

BFS	DFS
(1) It stands for Breadth first Search	(1) It stands for Depth first Search.
(2) It uses Queue data structure	(2) It uses Stack data structure
(3) It is more suitable for searching vertices which are closer to given source	(3) It is more suitable when there are solutions away from source.
(4) BFS consider all neighbours first & therefore not suitable for decision making trees used in games & puzzle	(4) DFS is more suitable for game or puzzle problems we make a decision then explore all paths through this decision and if decision leads to win situation we stop.
(5) Here sibling are visited before children	(5) Here children are visited before siblings.
(6) There is no concept of backtracking	(6) It is a recursive algorithm that uses backtracking.
(7) It requires more memory	(7) It requires less memory.

Applications

BFS → Bipartite graph and shortest path, peer to peer networking crawlers in search engine & GPS navigation system.

DFS → acyclic graph, topological order scheduling problems Sudoku puzzle.

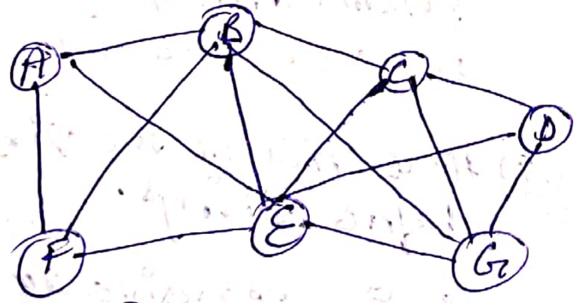
Q Which data structure used to implement BFS & DFS & why?

Ans] For implementing BFS we need a queue data structure for finding shortest path between any node. We use queue because things don't have to be processed immediately, but have to be processed in FIFO order like BFS. BFS searches for nodes level wise, it searches nodes w.r.t their distance from root (source) for this queue is better to use in BFS.

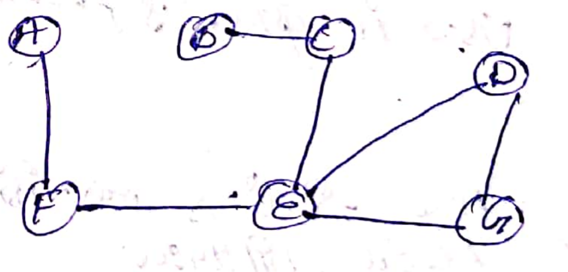
for implementing DFS we need a stack data structure as it transverses a graph in depthward motion and uses stack to remember to get next vertex to start a search when a dead end occurs in any iteration.

Ques) What do you mean by sparse and dense graph & which representation of graph is better for sparse & dense graph

Ans Dense graph is a graph in which no. of edges is close to maximal no. of edges. Sparse graph is graph in which no of edges is very less



Dense graph
(Many edges b/w nodes)



sparse graph (few edges b/w nodes)

→ for sparse graph it is preferred to use Adjacency list

→ for dense graph it is preferred to use Adjacency Matrix

Qu How can you detect a cycle in graph using BFS & DFS

Ans for detecting cycle in a graph using BFS we need to use Kahn's algorithm for topological sorting

The steps involved are:

(1) Compute in degree (no. of incoming edges) for each of vertex present in graph & initialise count of visited nodes as 0.

(2) Pick all vertices with in-degree as 0 and add them in queue

(3) Remove a vertex from queue and then
→ increment count of visited nodes by 1
→ Decrease in-degree by 1 for all its neighbouring nodes
→ If in degree of neighbouring nodes is reduced to zero then add to queue

(4) Repeat → until queue is empty

5) If count of visited nodes is not equal to no. of nodes in graph has cycle, otherwise not

for detecting cycle in graph using DFS we need to do following:

DFS for a connected graph produces a tree. there is a cycle in graph if there is a back edge present in the graph. A back edge is an edge that is from a node to itself (self-loop) or one of its ancestors in tree produced by DFS. for a disconnected graph, get DFS forest as output. To detect cycle check for a cycle in individual trees by checking back edges. To detect a back edge, keep track of vertices currently in recursion track for DFS traversal. If a vertex is reached that is already in recursion stack, then there is a cycle.

Ques. What do you mean by disjoint set data structure. Explain 3 operatives along with examples which can be performed on disjoint sets?

Ans. A disjoint set is a data structure that keeps track of set of elements partioned into several disjoint subsets. In other words, a disjoint set is a group of sets where no items can be in more than one set.

3 operations

→ find → can be implemented by recursively traversing the parent array until we hit a node who is parent to itself.

eg

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

→ union → It takes 2 element as input and find representatives of their sets using the find operation & finally puts either one of the trees under root node of other tree, effectively merging the trees & sets.

eg.

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

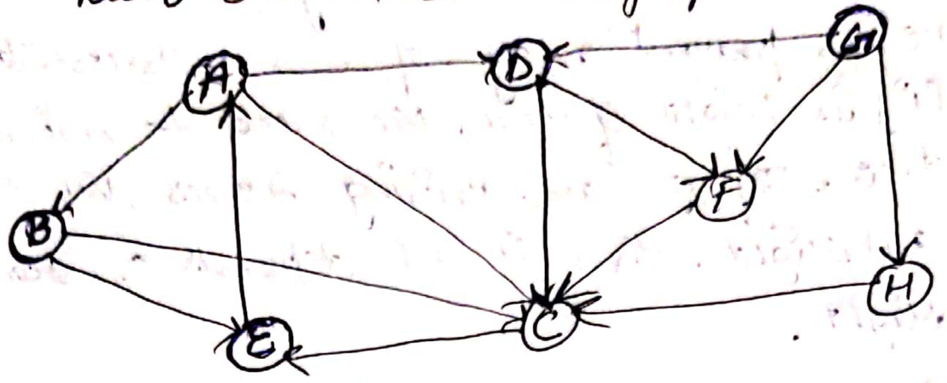
→ Union by rank → We need a new array rank[] . size of array same as parent array. If i is representative of set, rank[i] is height of tree. We need to minimize height of tree. If we are uniting 2 trees, we call them left and right, then it all depends on rank of left and right.

→ If rank of left is less than right then its best to move left under right & vice versa.

→ If ranks are equal, rank of result will always be one greater than rank of trees.

Eg:
```
void main (int i, int j)
{   int irep = this.find (i);
    int jrep = this.find (j);
        if (irep == jrep)
            return;
        irank = Rank[irep];
        jrank = Rank[jrep];
        if (irank < jrank)
    this.parent[irep] = jrep;
    else if (jrank < irank)
      this.parent[jrep] = irep;
    else {
        this.parent[irep] = jrep;
        Rank[irep]++;
    }
}
```

Que 60 . Run BFS & DFS on graph shown below



BFS

Child	G	H	D	F	C	E	A	B
Parent		G	G	G	H	C	E	A

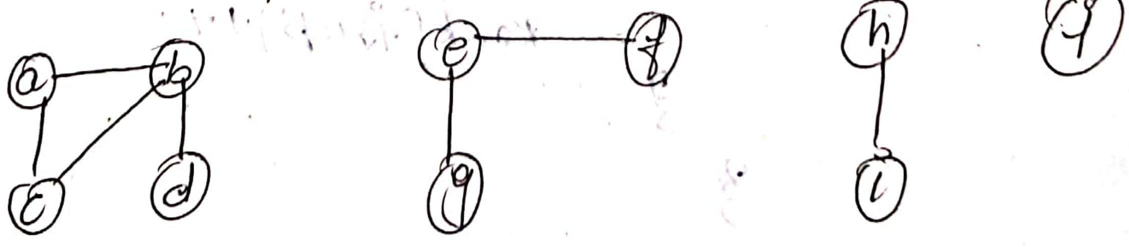
Path → G → H → C → E → A → B

DFS

Nodes visited	Stack
~~G~~	G
D	F
H	C
~~F~~	E
~~C~~	A
~~E~~	B
~~A~~	
~~B~~	

Path → G → F → C → E → A → B

70) find out no. of connected components & vertices in each component using disjoint set data structure



Ans) 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}

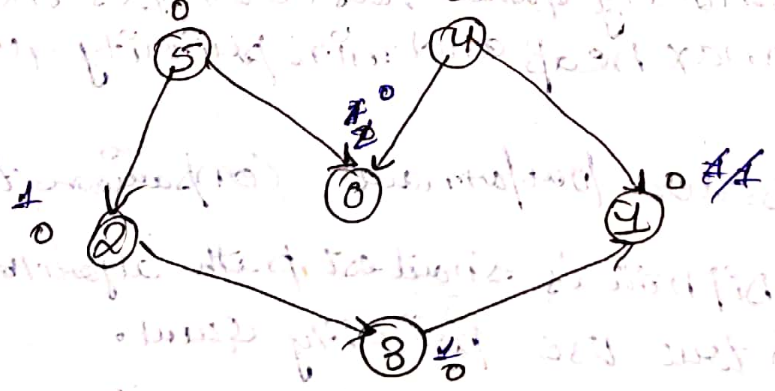
(e,f) {a,b,c,d} {e,f} {g} {h} {i} {j}

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

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

No. of connected components = 3 → Ans.

Que. Apply topological sort & DFS on graph having vertices from 0 to 5



Ans. We take source node as 5

Apply Topological Sort

q: 5/4 ; Pop 5 & decrement indegree of it by 1

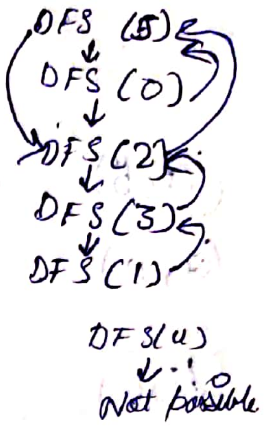
q: 4/2 ; Pop 4 & decrement indegree & Push 0

q: 2/0 Pop 2 & decrement indegree & Push 3.

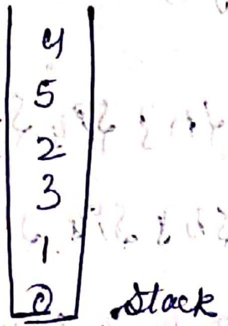
q: 0/3 Pop 0, Pop 3 Push 1

q: 1 ; Pop 1

Ans → 5 4 2 0 3 1
Topological Sort.



(DFS)



4 → 5 → 2 → 3 → 1 → 0
Ans.

Ques 9 Heap data structure can be used to implement priority queue. Name few graph algorithm where you need to use priority queue & why?

Ans Yes, heap data structure can be used to implement priority queue. It will take O(log N) time to insert & delete each element in priority queue. Based on heap structure, priority queue has two types max-priority queue based on max heap and min priority queue based on min heap.

Heaps provide better performance comparison to array.

The graphs like Dijkstra's shortest path algorithm, Prism's Minimum Spanning tree use priority queue.

→ Dijkstra's Algorithm → when graph is stored in form of adjacency list or matrix, priority queue is used to extract minimum efficiently when implementing the algorithm.

→ Prism's Algorithm → It is used to store keys of nodes & extract minimum key node at every step.

Ques10. Differentiate b/w Min-heap & Max heap

Ans.

Min - Heap	Max - Heap
1.) In min heap key present at root node must be less than or equal to among keys present at all of its children.	(1.) In max Heap the key present at root node must be greater than or equal to among keys present at all of its children
2.) The minimum key element is present at the root.	(2.) The maximum key element is present at the root
(3.) It uses ascending priority	(3.) It uses descending priority.
(4.) The smallest element has priority while construction of Min- heap.	(4.) The largest element has priority while construction of Max- heap.
(5.) The smallest element is the first to be popped from the heap.	(5.) The largest element is the first to be popped from the heap.