TUTORIAL - 5

NAME - AKSHIT SINGH
SECTION - F
ROLL. NO. - 42
UNIV. ROLL.NO. - 2016609

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

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 verticle which are closure given search.	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 & puzzles.	4) DFS is more suitable for game & puzzles problems. We make a decision then explore all paths through this decision and if decision leads to win situation, we stop
5) Here siblings are visited before children	5) Here, children are visited before siblings.
6) There is no concept of backtracking	6) It is a recursive algo. that uses ~~algo~~. backtracking
7) It require more memory	7) It require less memory.

Applications

a) BFS - Bipartied graph & shortest path, peer to peer networking, crawlers in search engines & GPS nevigation system

b) DFS - acyclic graph, topological graph, schedulling problem, sudoku puzzle.

Q2 → Which data structure is used to implement BFS and DFS and Why?

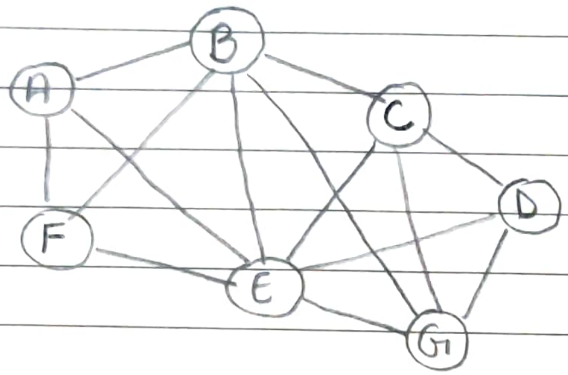
Ans - For implementing BFS we need a queue data structure for finding shortest path between any node. We ~~a~~ use queue because things don't have to be processed immediatly, but have to be processed in FIFO order like BFS. BFS searches for nodes level wise, i.e. it searches node w.r.t their distance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

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

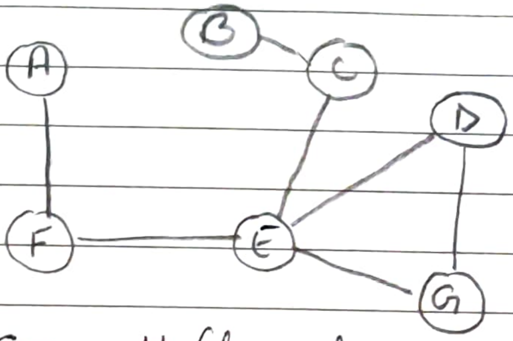
Q3) What do you mean by Sparse graph and dense graph? which representation of graph~~s~~ is better for sparse & dense

Dense graph is a graph in which no. of edges is close to maximal no. of edges.

Sparse graph is a graph in which no of edges is very less.



Dense Graph
(many edge b/w nodes)



Spare graph (few edges b/w nodes)

•) For spare graph it is prefered to use adjacency list

•) For dence graph it is prefered to use Adjacency.

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

Ans - For detecting cycle in a graph using BFS we need to use Kahn's algorithm for Topo logical Sorting.

The steps involved are:

1) Compute in-degree (no. of incoming edges) for each of vertex present in graph & initalize 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 added to queue.

4) Repeat.

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

(Q5) What do you mean by disjoint set data structure? Explain operations along with ~~operations~~ example that that can be performed on disjoint set?

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

3 operations:

- Find → can be implemented recursively traversing the parent away 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 elements as input and find representatives of their sets using the find operation and finally puts either one of the tree under root node of other tree, effectively merging the trees and sets

eg.

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

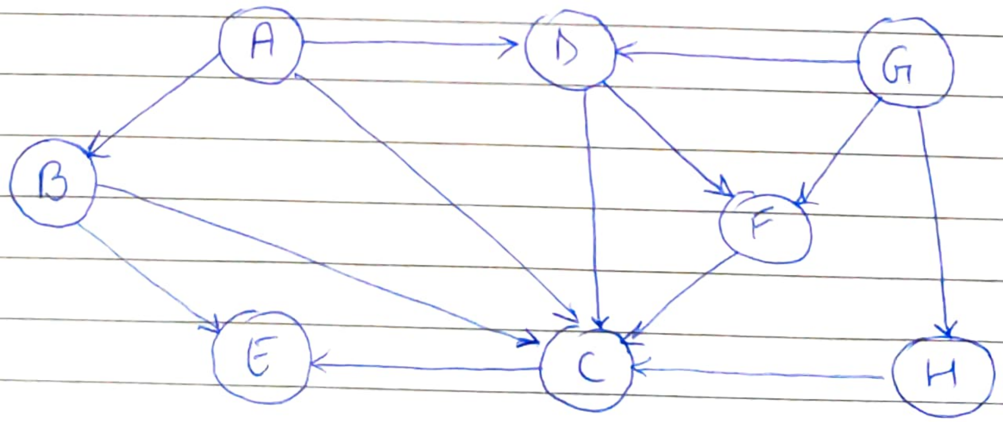
- Union by Rank → We need new array rank[]. Size of array same as parent array. If i is representative of of set, rank[i] is height of tree. We ~~a~~ need to minimize height of tree. If we are uniting 2 trees, call them left & right. Then it all depends on rank of left and right.
- If rank of left is less than right then it's 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 union (int i, int j) {
    int iup = this.find(i);
    int jup = this.find(j);
        if(iup=jup)
                return;
        irank = Rank[iup];
        jrank = Rank[jup];
        if(irank<jrank)
            this.parent[iup]=jup;
        else if (jrank <irank)
            this.parent[jup]=iup;
        else {
            this.parent[iup]= jup;
            Rank[jup]++;
        }
}
```

Q 6) Run BFS and 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:

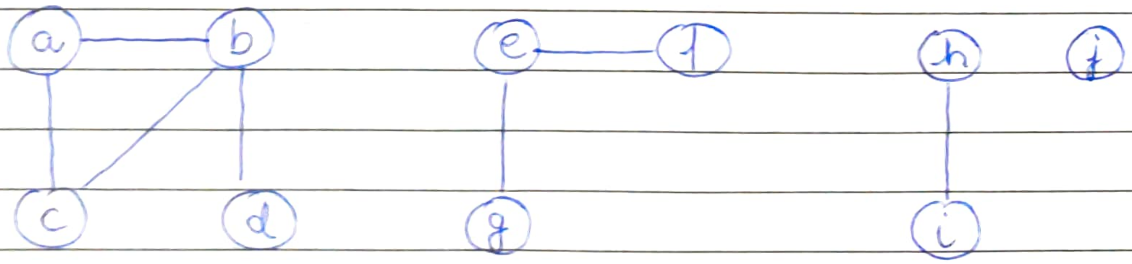
~~G~~
D
H
~~F~~
~~C~~
~~E~~
~~A~~
~~B~~
} NODES VISITED

G
F
C
E
A
B
} STACK

Path → G → F → C → E → A → B

7) Find out no. connected components and vertices in each component using disjoint set data structure.

Ans →



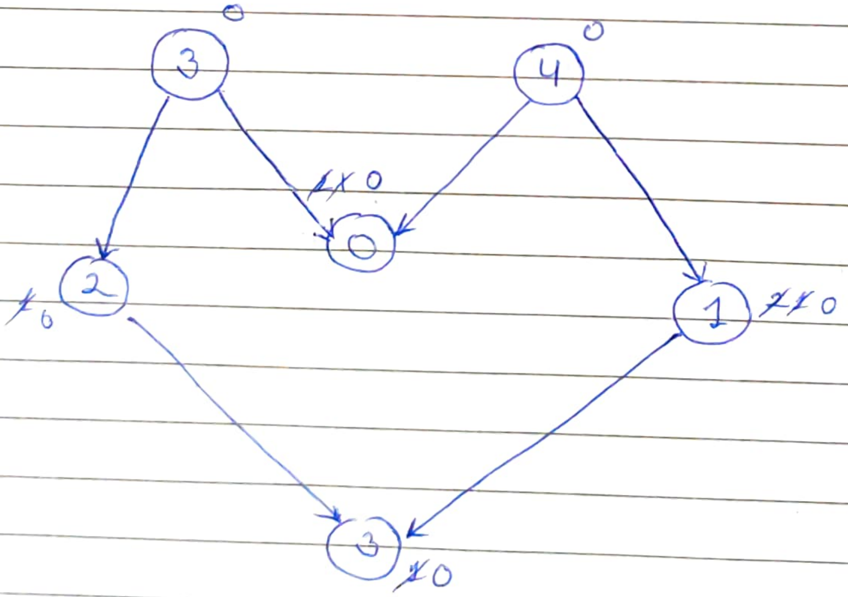
Ans → V = {a} {b} {c} {d} {e} {f} {g} {h} {i} {j}

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

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



Ans → We take souce node as 5.

Applying Topogical Sort →

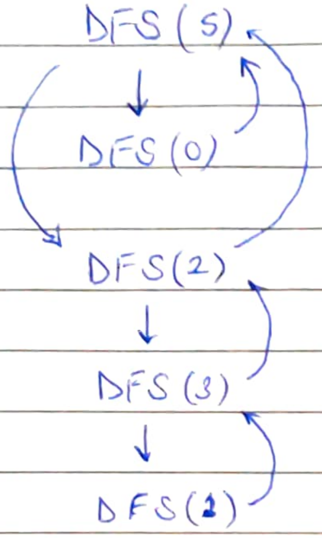
q : 5/4 ; Pop 5 of decremented indegree of it by 1

q : 4/2 ; Pop 4 & decremented in degree & push 0

q : 2/0 Pop 2 & decrement indegree & push

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

q : 1 ; pop 1



DFS(4)
↓
Not possible

Answer : $\underbrace{5\ 4\ 2\ 0\ 3\ 1}_{\text{Topological Sort}}$

DFS

4
3
2
3
1
0

Stack

4 → 5 → 2 → 3 → 1 → 0

Ans

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

Ans - Yes, heap data structure can be used to implement priority queue. It will take O(log N) time to insert and 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 maxheap and min priority queue based on min-heap. Heaps provide better performance comparison to array & L.L.

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 efficiency when implementing the algorithm.

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

10) Differentiate between Min-heap and Max-heap.

↳

Min-heap	Max-heap
i) In min-heap, key present at root node must be less than or equal to among keys present at all of its children	i) In max-heap, the key present at root node must be greater than or equal to among keys present at all present at all of its children

•) The minimum key element is present at the root	•) The maximum key element is present at the root
•) It uses ascending priority	•) It uses descending priority
•) The smallest element has preority while consstuction of Min-heap	•) The largest element has priority while construction of Max-heap.
•) The smallest element is the first do be popped from the heap	•) The largest element in the to be popped from the heap.