# Tutorial-5 (DAA)

Name- Shubham Gupta
Section-F Roll No.-28
Uni Roll No.- 2017050

**Qu-1** What is difference b/w DFS & BFS. Please write the application of both the algorithm.

**Sol^n:**

DFS

- DFS stands for Depth First Search
- It uses the data structure Stack
- It uses the concept of backtracking
- less memory required
- It is used to find a path from source to distination node

BFS

- BFS stands for Breadth First Search
- It use the data structure queue
- No concept of backtracking
- Requires more memory
- It is used to find single source shortest path in unweighted graph

Applications:

1. BFS
- It is used for detecting cycles in a graph
- It is used for finding routs from GPS
- It is used for finding shortest & min path in unweighted graph

2. DFS
- It is used for detecting cycles in a graph
- It is used for finding path between two vertices
- It is used for job scheduling process

Soln 2 In DFS we need to traverse a whole branch of a tree. So to keep track on the current node, it requires last in first out approach which can be implemented using stack. After it reaches depth of the node, then all the nodes will be popped out of stack.

In BFS, we've to go through the nodes with min no of nodes in between. So we don't have to look for all nodes. Therefore it uses queue data structure. If it uses stack then it will go through all the adjacent nodes which consumes more time & thus do not find min path

Ans-3 Dense graph is a graph in which number of edges is close to the maximal number of edges

Sparse graph is a graph in which number of edges is close to the minimal number of edges

For sparse graph, adjacency list is used
for Dense graph, adjacency matrix is used

Ans-4 Using DFS:

1. Create a graph using given no of edges & vertices
2. Create a recursive function that have current index visited array & parent node
3. Mark current node as visited
4. Find all vertices which are not visited & are adjacent to current node. Recursively function for those vertices
5. If the adjacent node is not parent node and is already visited, then return true
6. Call the recursive function for all the vertices & if any function return true, return true
7. Else for all the vertices, the function returns false, return false

Using BFS:

1. Pick all vertices with status 0 and add them with a queue
2. Remove a vertex from queue & add them
   Increment count by 1 for all its neighbouring nodes
   Decrease status by 1 for all its neighbouring nodes
   . If status is reduced to 0, the add it to queue
3. Repeat step 2 until queue is empty
4. If count is not equal to the no of nodes then a graph then cycle occurs not.

Ans-5 Disjoint set are similar to sets in maths but they are modified for the usage in algorithms, In other words a disjoint set is a gp of sets where no item can be in more than one set.

3 operations:

- Find - can be implemented by recursively traversing the parent array until we hit a node which is present to itself.

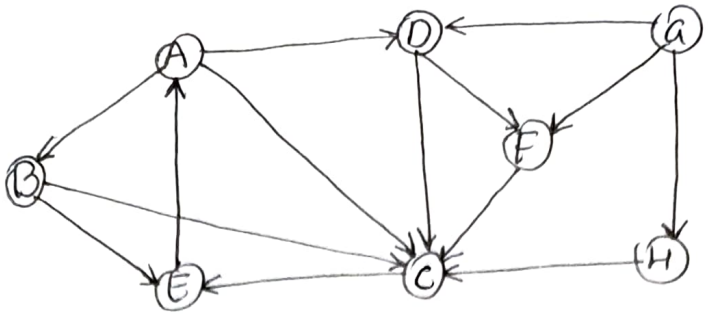
```
eg  int find(int i){
        if(parent[i]==i)
            return i;
        else
            return find(parent[i]);
    }
```

- Unione - It takes two elements as input & find representatives of thos sets using the find operation and finally puts either one of the trees under root node of other tree, effectively merging the trees & sets.

- Union by Rank - We need a necessarily rank[] since of array same as parent array. If it is representative of set. rank[i] is height of tree. We need to minimize height of tree.

```
eg  void union(int i, int j){
        int irep = this.find(i);
        int jrep = this.find(j);
        if(irep == jrep) return
        irank = rank[irep]
        jrank.Rank[jrep]
```

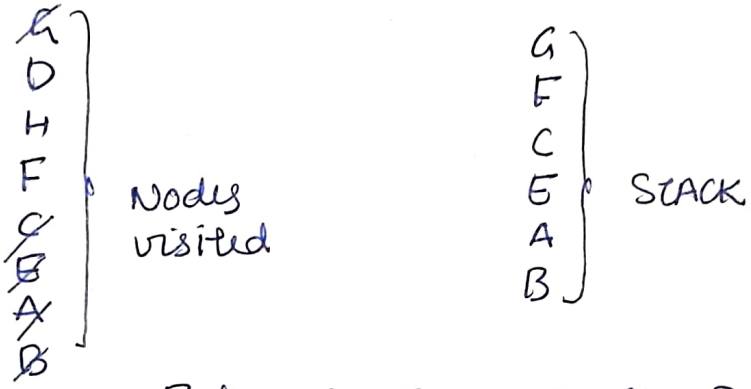
Ans-6




BFS:

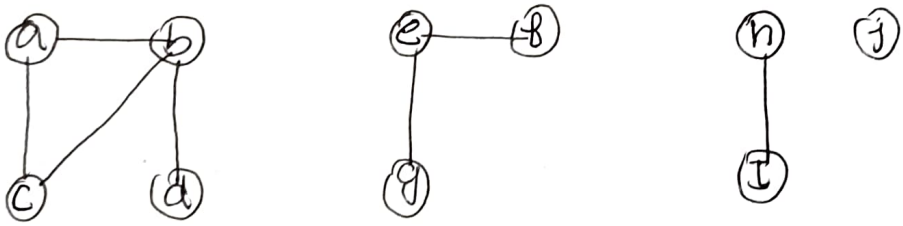
| Child | G | H | D | F | C | E | A | B |
|---|---|---|---|---|---|---|---|---|
| Parent | | G | G | G | H | C | E | A |

DFS:




Path → G → F → C → E → A → B

Ans-7




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

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

(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}

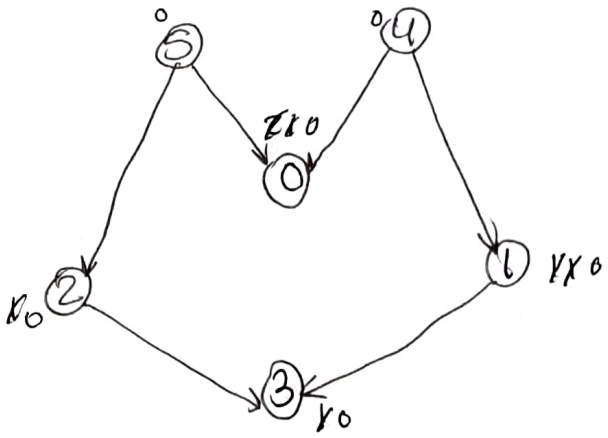
(e,f) {a,b,c,d}{e,f}{g}{h}{i}{j}

(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

Ans-6




we take source node as 5

Applying Topological Sort

q: 5/4 ; pop 5 & dec indegree of it adj

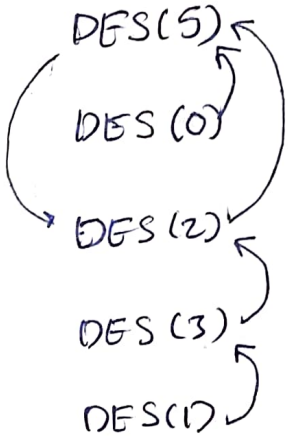
q: 4/2 ; pop 4 & dec indegree & push 0

q: 2/0 : pop 2

q: 0/8 ; pop 0, p03 push 1

q: 1 ; pop 1

Answer : 5 4 8 2 0 3 1




DFS


4 → 5 → 2 → 3 → 1 → 0 Ans

Ans-10

## Min-Heap

- In min heap, key present at root node must be less than or equal to among key present at all of its children.
- The minimum key element is present at the root
- It uses ascending priority
- The smallest element has priority while construction of min heap
- The smallest element is the first to be popped from heap

## Max-Heap

- In max heap the key present at root must be greater than or equal to among keys present at all its children
- The max key element is present at the root
- It uses decending priority
- The largest element has Plority while construction of max heap
- The largest elment is the first to be popped