Sk Sohail F,29

DAA Tutorial - 5

1. What is diff. b/w DFS & BFS. Please write the applications of both algorithms.

1. BFS

① BFS stands for Breadth first search.

② BFS uses Queue data structure for finding the shortest path.

③ BFS consider all neighbours first & therefore not suitable for decision making trees used in games or puzzles.

DFS

① DFS stands for Depth first search.

② DFS wes stack data structure.

③ DFS is more suitable when there are sol^{n} among from source.

BFS application:

1. Path finding algo is based on BFS.
2. Used in ford-fulkerson algo to find max. flow in a network.

DFS application:

1. If we perform DFS on unweighted graph, then it will create mini. spanning.

2. Using DFS we can find path b/w two given vertices u & v.

2. Which data structure are used to implem. BFS & DFS & why?

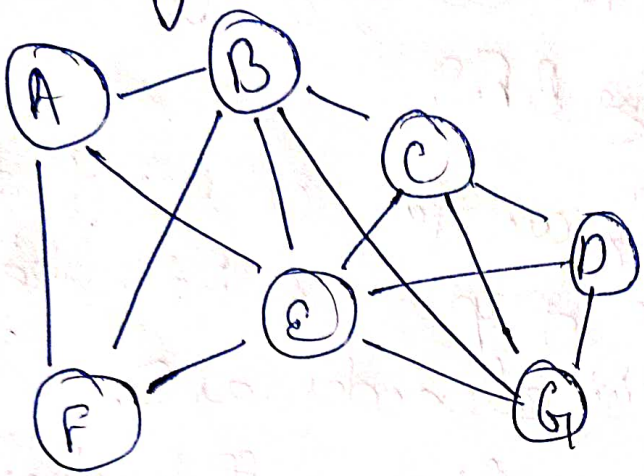
BFS uses the queue data structure to traverse a graph in a ~~search~~ breadth forward & uses a queue to remember to get the next vertex to start & search, when a end occur in any the queue follow the queue concept the grant are discounted pass will be explained.

DFS uses the stack data structure to traverse the graph to depth the motion & uses stack to remember,

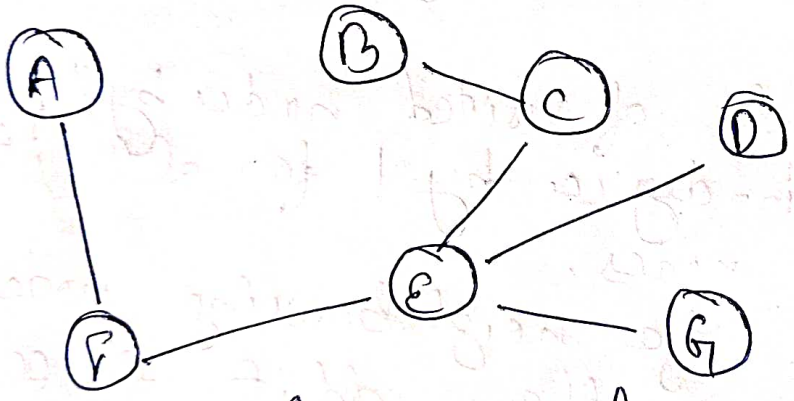
3. What do you mean by sparse & dense graphs? Which representation of graph is better for sparse & done graphs?

A dense graph is which no. of edges is close to the max. no. of edges.

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



Dense graph
(many edges b/w nodes)



Sparse graph (few edges b/w nodes)

1. for sparse graph it is preferred to use adjacency list.
2. for dense graph it is preferred to use adjacency matrix.

4. How can you detect a cycle in graph using BFS & DFS?

Steps involved in detecting cycle in a directed graph using BFS.

Step1: Compute in-degree for each of the vertex present in the graph & initialize the count of visited nodes as 0.

Step 2: Pick all the vertices with in-degree as 0 & add them into a queue.

Step 3: Remove a vertex from the queue & then.

1. Incr. 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.

Step4: Repeat step 3, until the queue is empty.

Step5: If count of visited nodes is not equal to the no. of nodes in the graph has cycle, otherwise not.

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

The disjoint set data structure is also known as union-find data structure & merge-find set. It is a data structure that contains a collection of disjoint.

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 & find representation of the sets using the find operations & finally puts either one of the tree under root of other tree efficiently merging the tree & sets.

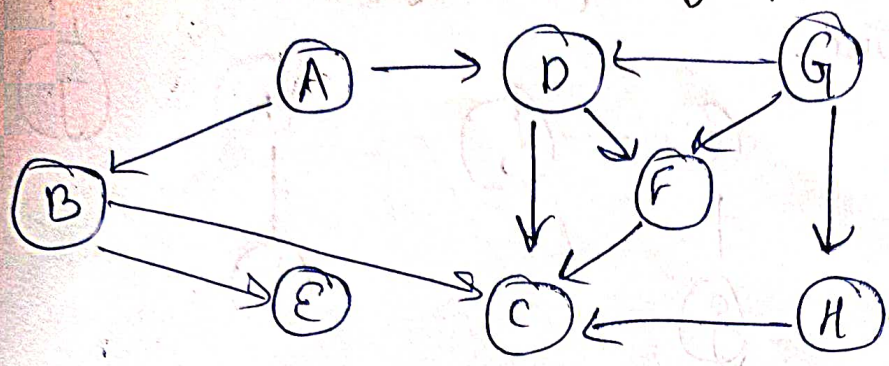
```
void union(int i, int j)
{
    int irep = this.Find(i);
    int jrep = this.Find(j);
    this.parent[irep] = jrep;
}
```

- **Union by Rank** : We need a new array rank[]. Size of array same as parrent array
  - If rank of left is less than right then, it's least to more, left under right, vice-verse.
  - If ranks are equal, rank of result will always be 1 greater than rank of trees

Eg.

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

6. Run BFS & DFS on graph shown below.

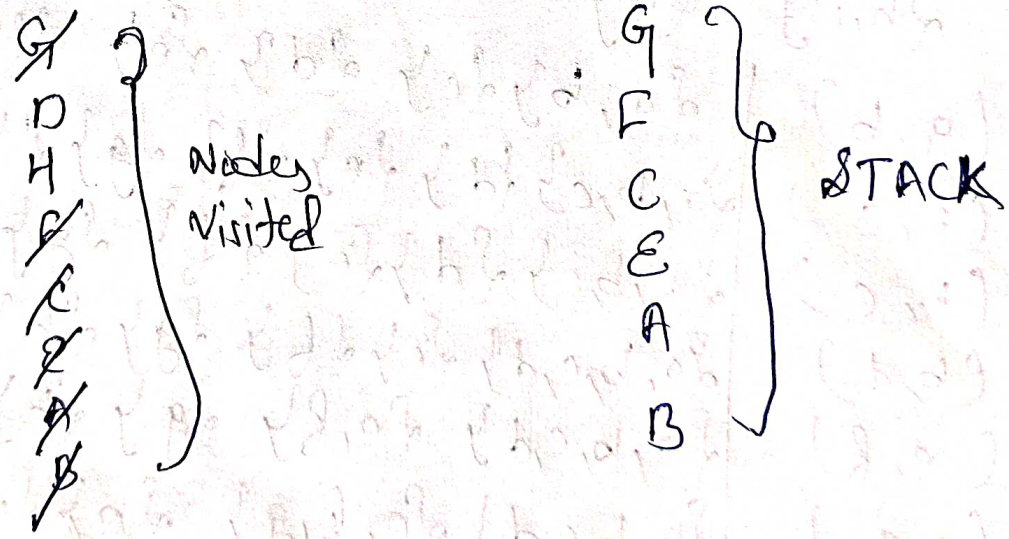


BFS

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

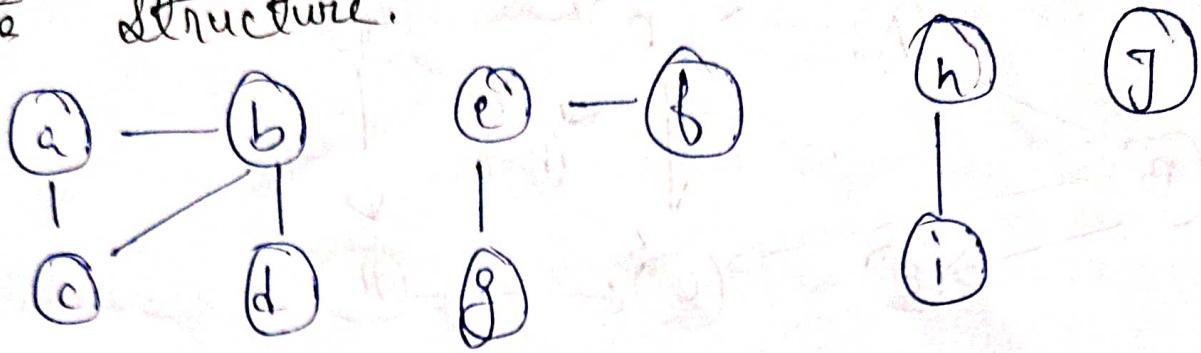
Path → G → H → C → E → A → B

DFS



Path → G → F → C → E → A → B

7. Find out no. of connected components & ve in each component using disjoint set data structure.



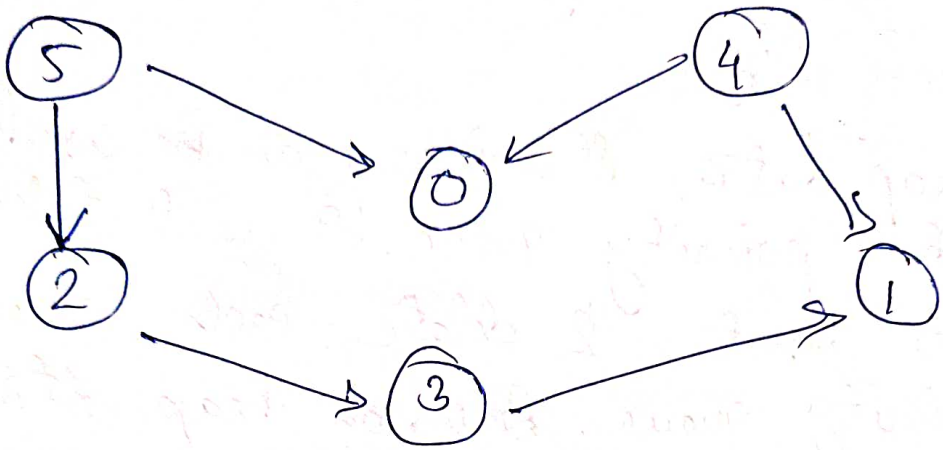
V = {a} {b} {c} {d} {e} {f} {g} {h} {i} {j}

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}

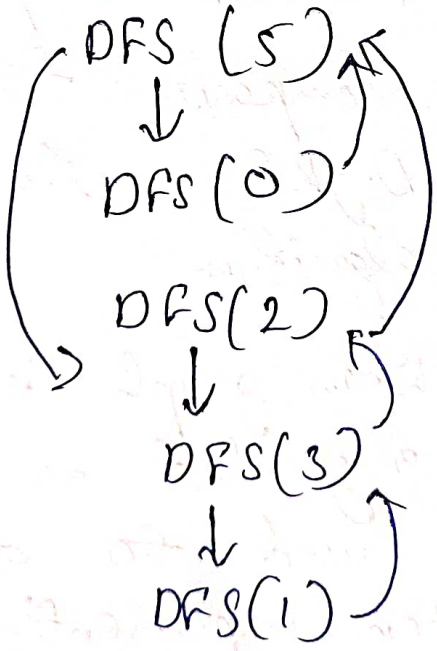
No. of connected components = 3

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



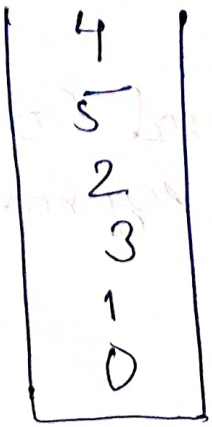
We take source node as 5

Applying topological sort



DFS (4)
↓
Not possible

DFS



Stack

4 → 5 → 2 → 3 → 1 → 0

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

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 & min. priority queue based on min-heap. Heaps provide better performance comparison to array. The graphs like Dijkstra's shortest path algorithm, prim's algorithm

- Dijkstra's algorithm: When graph is stored in the form of adjacency list or matrix, priority queue, is used to extract min efficiently when implementing the algorithm.
- Prim's algorithm: It is used to store keys of nodes & extract min. key node at every step.

Differentiate b/w Minheap & Maxheap

Minheap	Maxheap
In min-heap, key present at root node must be less than or equal to among keys present at all of its children.	In max-heap the key present at root node must be greater than or equal to among keys present at all of its children.
The mini key element is present at the root.	The max key element is present at root.
The smallest element has priority while construction of Min-heap.	The largest element has priority while constr. of Max-heap.