Tutorial No-5

Question 1. what is difference b/w DFS and BFS. write applications of both the algorithm.

Ans

BFS	DFS
1) It stands for Breadth First Search.	1) It stands for Depth first search.
2) It uses Queue Data Structures	2) It uses stack data structure.
3) It is more suitable for searching vertices which are close to given source.	3) It is more suitable when there ar solutions away from sources.
4) It considers all neighbours first ~~of~~ and therefore not suitable for decision making trees used in games and puzzles.	4) It 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 siblings are visited before the offsprings	5) Here offsprings are visited before siblings.
6) Backtracking is possible.	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 search engine ~~of~~ and GPS navigation system.

DFS → Acyclic graph, topological order, scheduling problems, sudoku puzzles.

Question-2 Which data structures are used to implement BFS and DFS and why?

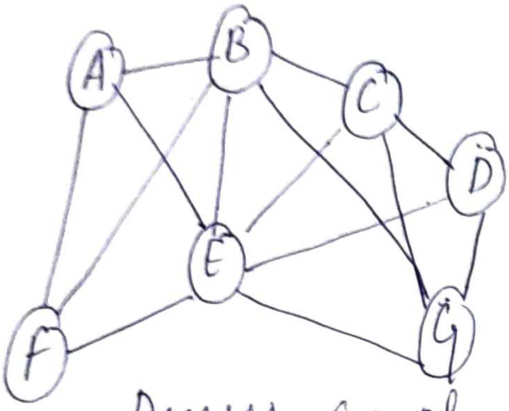
For implementing BFS we used queue data structure. for finding shortest path b/w any node. We use queue because things don't have to be processed immediately, but have to be processed in FIFO order like BFS. BFS searches for nodes level wise, i.e it searches nodes w.r.t their distance from root (source).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

For implementing DFS are used a stack data structure as it transverse a graph in depthward motion and uses stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

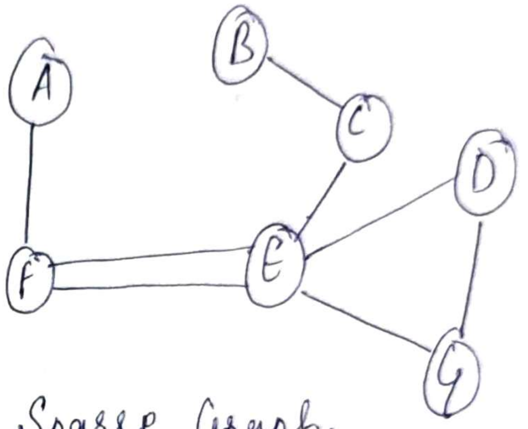
Ques-3: what do you mean by sparse and dense graph? which representation of graph is letter for sparse and dense graph?

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

Space graph is graph in which no. of edges is very less



Denser Graph
(many edges b/w nodes)



Sparse Graph
(Few edges b/w nodes)

→ For sparse graph, it is prefered to use Adjacency list.

→ For denser graph, it is prefered to use Adjacency Matrix

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

Ans For detecting cycle in a graph using BFS ~~as~~ 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 and intialize 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
→ Decreases in degree by 1 for all it's neighbouring nodes
→ If in-degree of neighbouring nodes is reduced to zero then add to queue.

4) Repeat step (3) until queue's 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 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 another in the 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 stack for DFS traversal. If a vertex is reached that is already in recursion stack, then there is a cycle.

Question 5. What do you mean by disjoint set data structure? Explain three operation along with ~~eam~~ example which can be performed on disjoint sets?

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

Three operations are -

a) Find — can be implemented by recursively traversing the parent array until we hit a node who is parent to itself

```
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 this sets using the find operation and finally puts either one of the trees under root of other tree, effectively merging the trees and sets.

```
Ex -  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 used to minimize height of tree . If we are limiting two 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 it's best to move left under right and vice versa.
- If rank are equal, rank of result will always be one greater than rank of trees.

Ex →

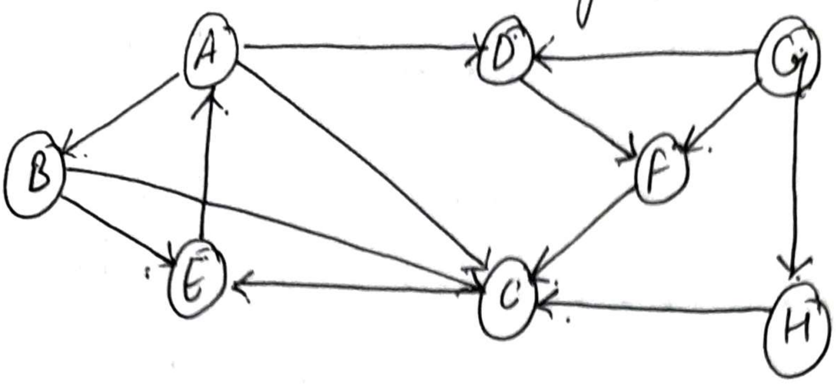
```
void union( int i, int j)
  {
    int irep = this.find (i);
    int jrep = this.find (j);
    if (irep ==jrep)
    return;
    irank = Rank [irep ];
    jrank = Rank [jrep];
  if (irank < jrank)
     this.parent [irep ] =jrep;
  else if (jrank <irank)
     this.parent [jrep ] = irep;

  else
  { this.parent [irep ] =jrep;
     Rank [jrep ]++;

  }

}.
```

Ques 6. Run BFS and DFS on graph below.



BFS

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

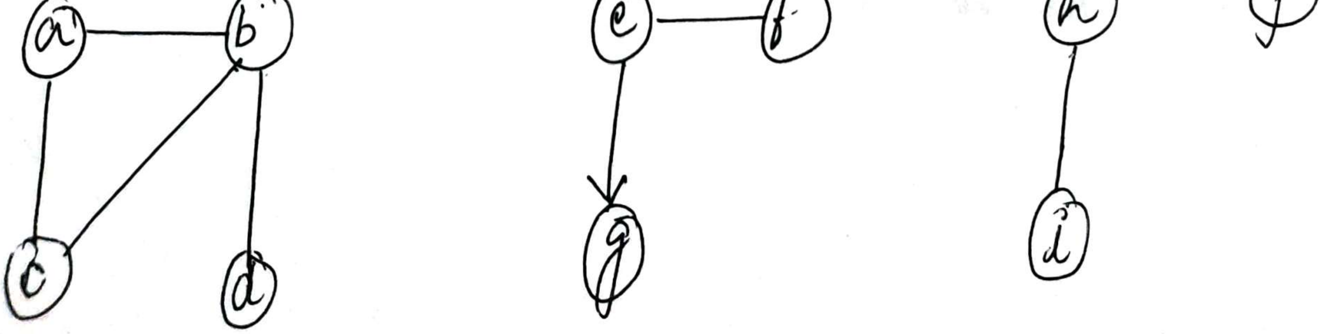
DFS

~~G~~
D
H
~~F~~
~~C~~
~~E~~
~~A~~
~~B~~
} nodes visited

G
F
C
E
A
B
} Stack

Path → G → F → C → E → A → B

Ques 7. Find out no. of connected components and vertices 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}

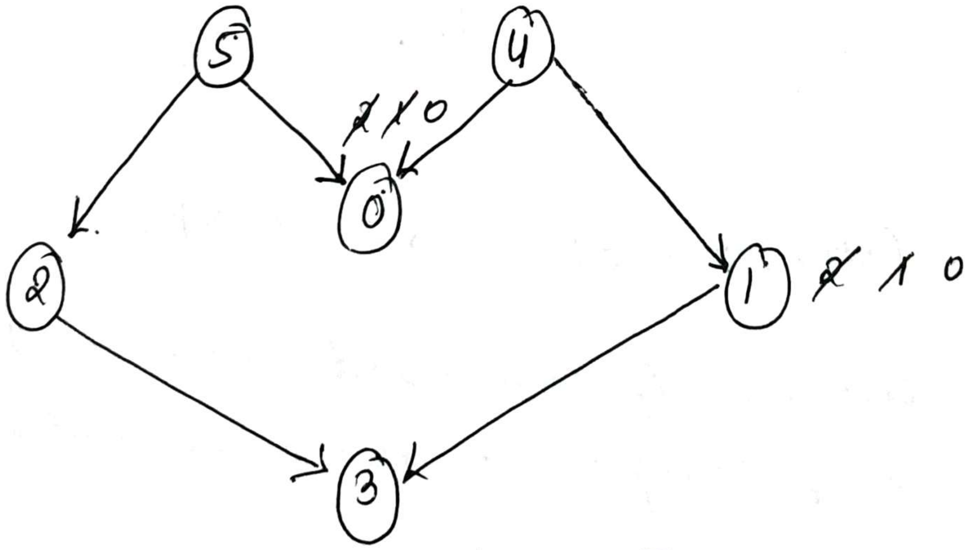
(e, f) {a, b, c, d} {e, f} {g} {h} {i} {j}

(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 — Ans

Ques 8. Apply topological sort and DFS on graph having vertices from 0 to 5



We take source node as 5

Applying Topological sort

→ q : 5/4 ; pop 5 and decreament in degree of It by 1.

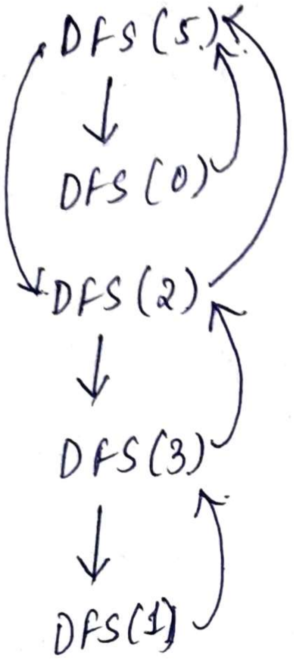
→ q : 4/2 ; Pop 4 and decrement in degree and push 0

→ q : 2/0 ; Pop and decrement in degree and push 3

→ q : 0/3 Pop 0, Pop 3
Push 1

q : 1; pop 1

Answer – $\underbrace{5\ 4\ 2\ 0\ 3\ 1}_{\text{Topological sort}}$



DFS(4)
↓
Not possible.

DFS

4
5
2
3
1
0
Stack

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 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 type max priority queue based on max heap and min. priority queue based on min-heap. Heap provides better performance compairitevely to array and holo

The graph like Dijkstra's shortest path algorithm Prim's Minimum Spanning Tree uses 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.

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

Ques 10 Differentiate b/w Min heap and Max heap.

Ans.

Min heap	Max heap
1) In min-heap, key present at root node must be less than or.	1) 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 max 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 the heap.