Tutorial -5 Tutorial 5

Name → Kavishna Rathi

Section → F

Roll No. → 62

University Roll no. :→ 2016822

Q1)) What is the difference b/w DFS & BFS. Write applications of both the algorithms.

Answer)

BFS	DFS
(a) It stands for Breadth first search	(a) It stands for Depth first search
(b) It uses queue	(b) It uses stack
(c) It is more suitable for searching	(c) It is more suitable when there are solutions away from source.
(d) BFS considers all neighbours first of therefore not suitable for decision making trees used in games of puzzles.	(d) DFS is more suitable for game or puzzle problem. We make a decision, then explore all paths through this decision. And if decision leads to using situations we stop.
(e) Here siblings are visited before children.	(e) Here children are visited before sibling.
(f) There is no concept of Back-tracking.	(f) It is a recursive algorithm that uses back-tracking.
(g) It requires more memory.	(g) It requires less memory.

Applications :-

(a) BFS → Bipartite graph and shortest path, peer to peer networking, crawlers in search engine of GPS navigation system.

(b) DFS → acyclic graph, topological order, scheduling problems sudoku puzzle.

PTO

(Q2) → Which data structure are used to implement BFS and why?

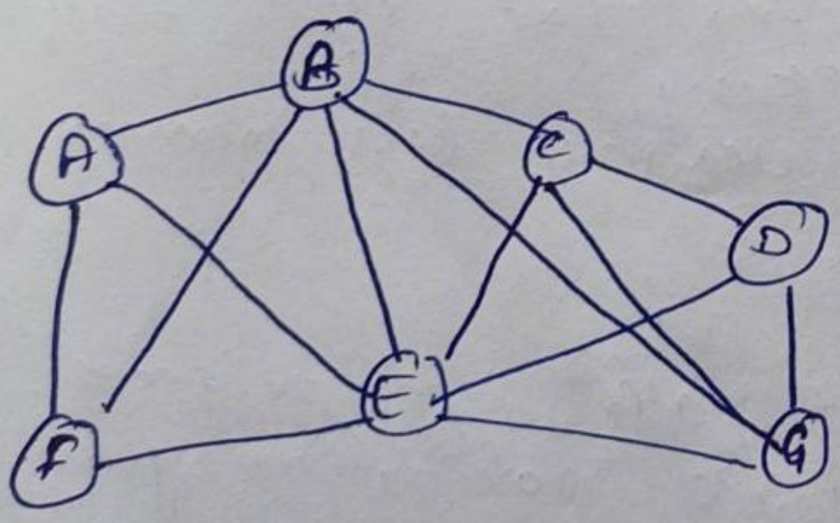
↳ for implementing BFS we need a queue data structure for finding a shortest path b/w any node. We use queue because things don't have to be processed immediately, but have in FIFO order like BFS. BFS searches for nodes level wise i.e it searches nodes w.r.t their distance from root (source) for this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

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

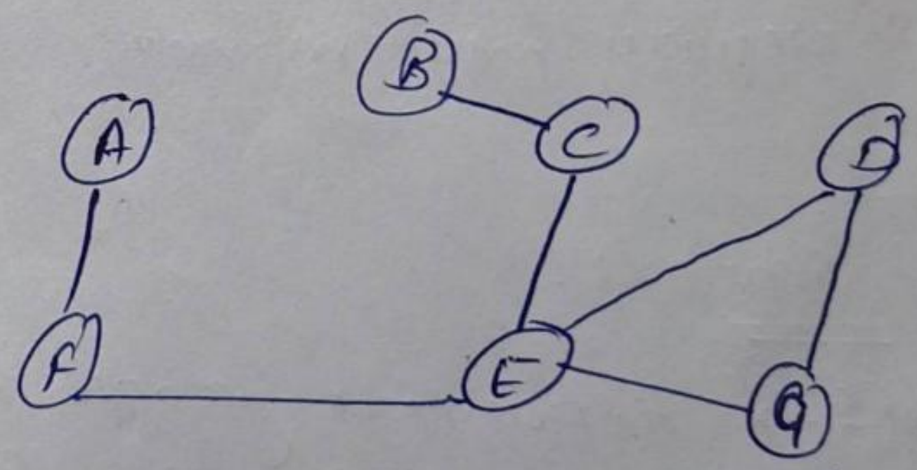
(Q3) What do you mean by sparse graph and dense graph? Which representation of graph is better for sparse and dense graph?

↳ 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



Sparse graph

(1) for sparse graph it is preferred to use Adjency List.
(2) for dense graph it is preferred to use Adjacency Matrix

→ How can you detect a cycle in a graph 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 & initialize 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 node 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 3) 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 a graph use DFS, do the following, DFA 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 ancestor 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 recursive stack for DFS traversal.

(Q6) → What do you mean by disjoint set data structure? Explain 3 operations along with examples which can be performed on disjoint set.

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

(operations):

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

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

(2) Union → It takes 2 elements as input. And find representation of this set using the find operation and finally puts either one of the trees under root node of the 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;
}
```

(3) 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 two minimize height of tree. If we are uniting 2 trees, we call them left and right.

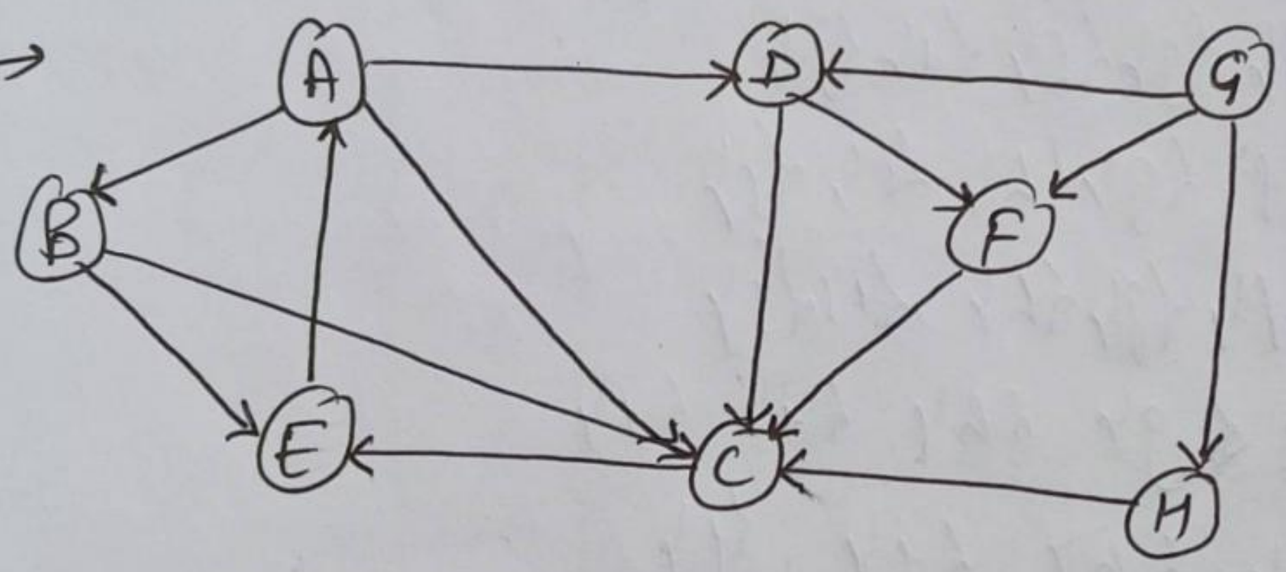
- If rank of left is less than right then its best to move left under right of vice versa.
- If ranks are equal, rank of result will always be one greater than rank of trees.

eg →
```
void union (int i, int j) {
    int 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- [jrep] = jrep;
    ~~ets~~ Rank [jrep] ++;
    }
}
```

(Q6) → Run Bfs and Dfs on graph shown below;

↳

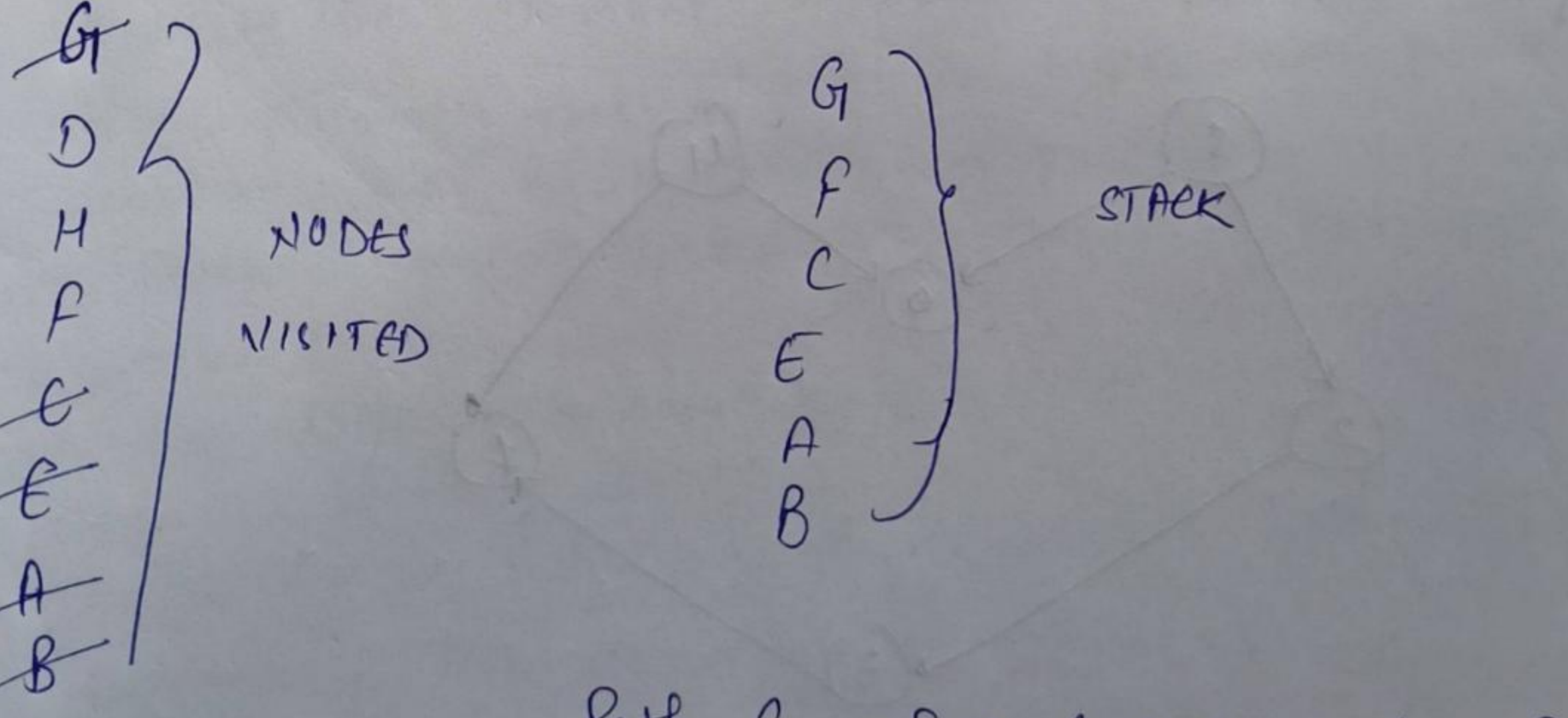


BFS

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

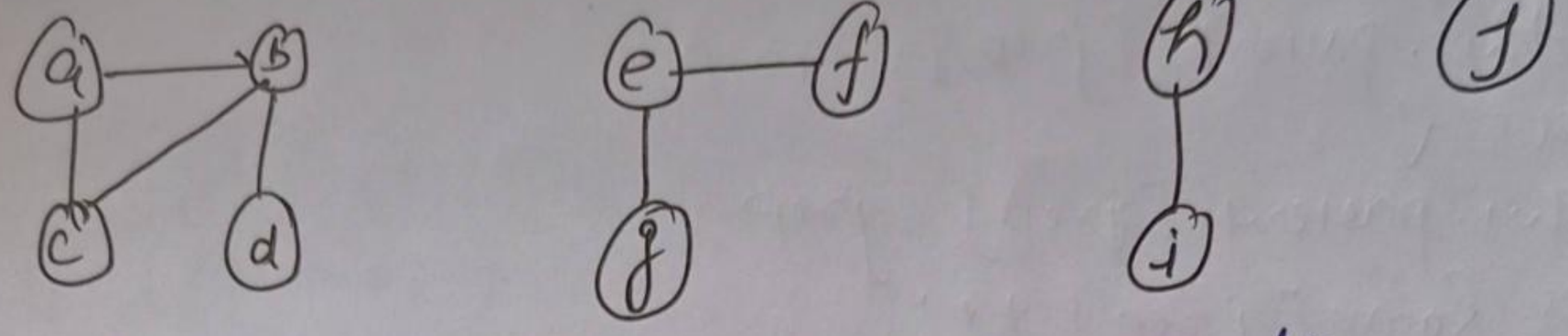
Path → G → H → C → E → A → B

DFS



Path G → F → C → E → A → B

Q7) find out no. of connected components 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}

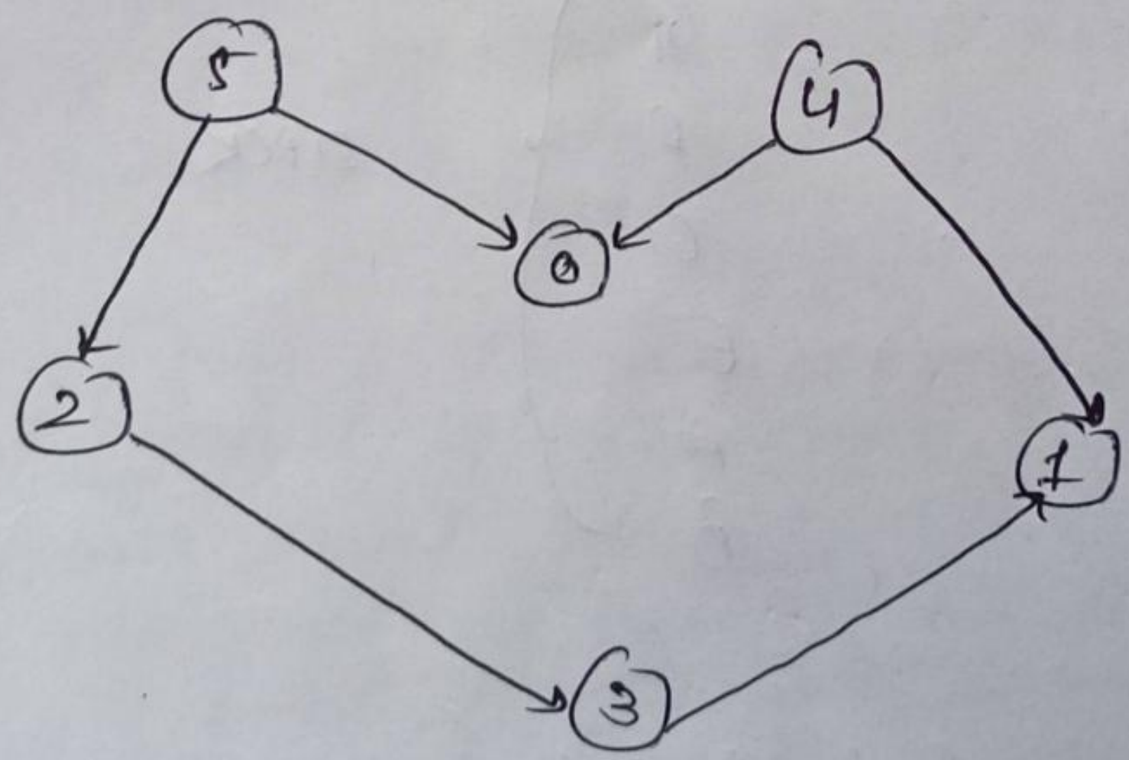
(e,f) {a,b,c,d} {e,f} {g} {h} {i} {j}

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

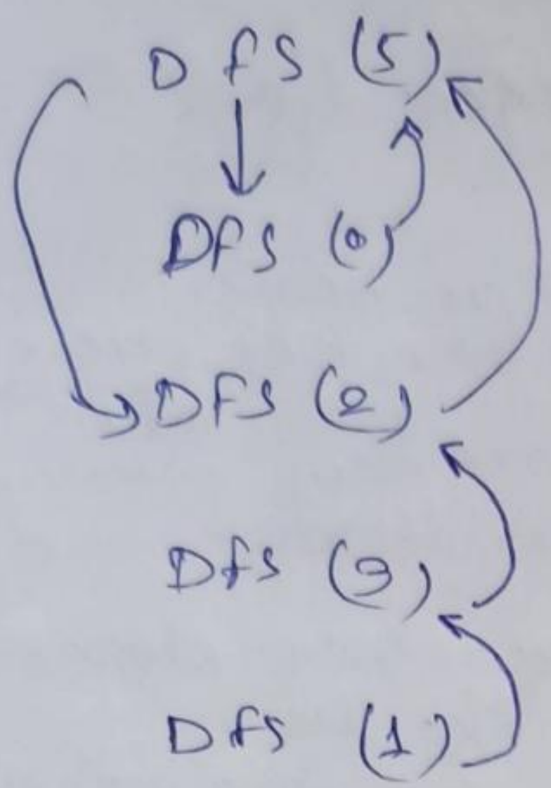
(h,i) {a,b,c,d} {e,f,g} {h,i} {j}

No. of connected components = 3 → Ans

Q8) → Apply topological sort of DFS on graph having vertices from 0 to 15.

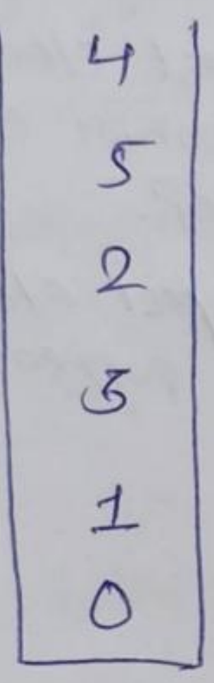


We take source node as 5
Applying Topological sort



DFS (4)
↓
Not possible

DFS



STACK

4 → 5 → 2 → 3 → 1 → 0

Ans

(Q9) 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, the statement is true. It will take O(logN) 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 based on min heap. Heaps provide better performance comparison to array of linked list.

The graph like Dijkstroo's shortest path algorithm, Prim's Min. Spanning Tree use Priority Queue.

- Dijkstroo's Algorithm → When graph is stored in the 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, extract minimum key node at every step.

(Q10) Difference b/w Min-Heap and Max heap.

Min heap	Max heap
→ 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 childeren.
→ The minimum key element is present at root.	→ The maximum key element is present at the root.
→ It uses ascending priority.	→ It uses decending priority.
→ The smallest element has priority while construction of min-heap.	→ The largest element has priority, while construction of Max heap.
→ The smallest element is the first to be popped from the heap.	→ The largest element is the first to be popped from the heap.