Tutorial-5

Name - Harshita Pratap

Section - G

Roll no - 36

Subject - Design and Analysis of Algorithms

Q1. What is the difference between DFS and BFS write application of both the algorithms.

BFS	DFS
1) It stands for breadth first search	It stands for depth first search
2) It uses queue data structure	It uses stack data structure
3) It is more suitable for searching	It is more suitable when there are solutions away from source.
4) BFS considers all the neighbours first of therefore not suitable for decision making trees used in games and puzzles.	DFS 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 children	Here children are visited before siblings.
6) There is no concept of backtracking	It is a recursive algorithm that uses backtracking

Applications -

BFS → Bipartite graph and shortest path, peer to peer networking

DFS → acyclic graphs, topological order, scheduling problems, sudoku puzzle.

Q2. Which data structure are 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 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 wrt their distance from root (source). For this queue is better to use in BFS.

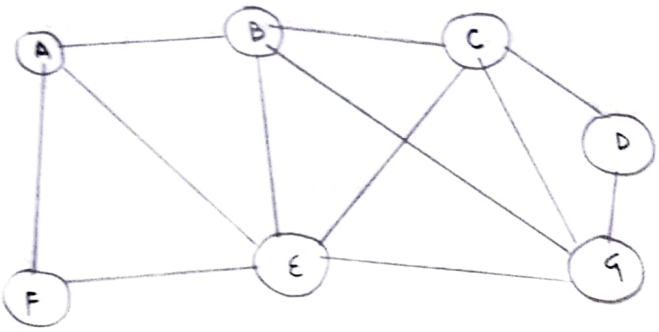
For implementing DFS, we need a stack data structure as it traverses a graph in depthward motion and uses stack to resembles to get the next vertex to start a search, when a dead end occurs in any iteration.

C
–
B

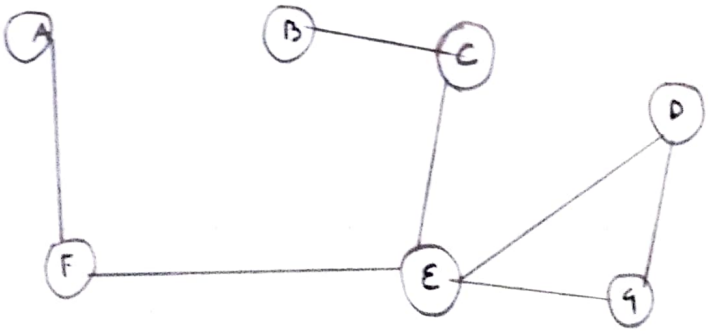
Q3. What do you mean by sparse and dense graphs. Which representation of graph is better for sparse and dense graphs.

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 are very less.



Dense graph
(many edges b|w nodes)



Sparse graphs
(five edges b|w nodes)

For sparse graph it is preferred to use Adjacency List.

For dense graph it is preferred to use Adjacency Matrix.

Q4. How can you detect cycle in a graph using BFS and DFS.

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 and initialize count of visited array node as zero

2) Pick all vertices with in degree as 0 and add them in queue

3) Rename 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 it 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.

3 ope

a. F. can arr

For detecting cycle in graph using DFS we need to do following :

DFS for a connected graph produces a tree. Then is cycle in graph if there is back edge present in the graph. A back edge is a 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 in dividual trees by checking back edges. To detect a back edge, keep track of vertices currently in recursion track, DFS traversal of 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 3 operations along with examples which can be performed on disjoint set.

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

3 operations

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]);
}
```

b. Union.

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

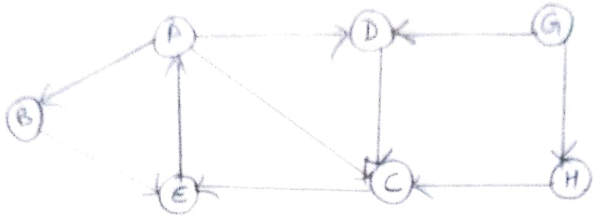
```
void union (int i, int j) {
    int irep = this.Find (i);
    int jrep = this.Find (j);
    this.parent[irep] = jrep;
}
```

Union by Rank

If i is representative of set rank i is height of tree.

```
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 [ irep]= jrep;
         Rank [jrep ] ++;

    }
}
```

Q6. Run BFS and DFS on graph schema 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

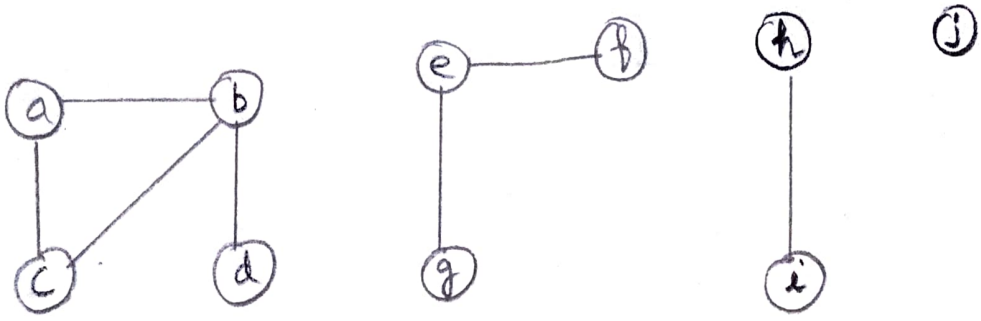
DFS

NODES VISITED: G, D, H, F, C, E, A, B

STACK: G, F, C, E, A, B

Path → G → F → C → E → A → B

Q7. Find out no. of connected components and vertices in each component using disjoint set.



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}

(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}

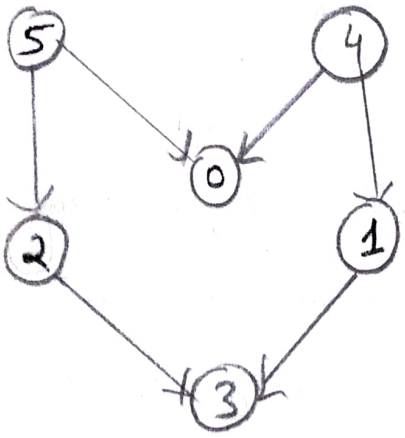
(h, i) {a b c d} {e f g} {h i} {j}

No. of connected components = 3

↓

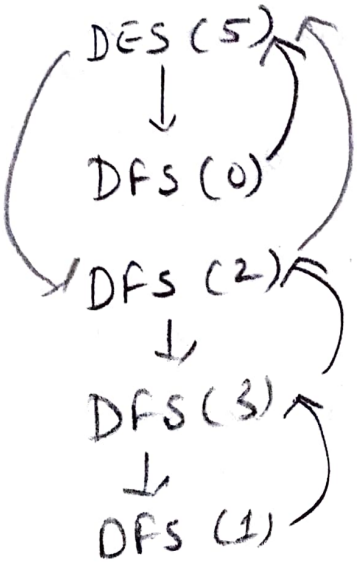
Ans.

Q8. Apply Topological sort of DFS on graph having vertices from 0 to 15



Ans We take source node as 5

Applying Topological sort



DFS (4)
↓
Not Possible

DFS

4
5
2
3
1
0

stack

4 → 5 → 2 → 3 → 1 → 0 Ans

Q9. Heap DS can be used to implement priority queue. Name few graph Algorithms where you need to use priority queue and why?

Yes, heap DS 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 min heap

Heaps provide better performance comparison to array

The graph like - Dijistra's shortest path Algorithm

Minimum spamming Tree uses priority queue.

Dijistra's Algorithm

when graph is stored in form of adjacency list or matrix, priority queue is used to extract minimum efficiently when implementing algorithm.

Prim's Algorithm

It is used to store keys of nodes

10. Differentiate between 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 children
The min key element is present at root	The maximum key element is present at root.
It uses ascending priority	It uses descending 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 heap