IARE

IARE
INSTITUTE OF
AERONAUTICAL ENGINEERING
(An Autonomous Institute affiliated to JNTUH, Hyderabad)
Dundigal, Hyderabad - 500 043

LABORATORY WORK BOOK

Name of the Student : RACHERLA SANTHOSH

Class : IT-B Semester : 03

Roll Number									
2	3	9	5	1	A	1	2	C	3

Course Code : AGSD11 Course Name : DS Laboratory

Name of the Course Faculty : Ms. K. Laxminarayanamma Faculty ID : IARE10033

Exercise Number : 12 Week Number : 12 Date : 25/11/2024

S. No.	Exercise Number	EXERCISE NAME	MARKS AWARDED						
			Aim/ Preparation	Algorithm / Procedure	Performance in the Lab	Source Code Calculations and Graphs	Program Execution Results and Error Analysis	Viva - Voce	Total
			4	4		4	4	4	20
1	12.1	Breadth First Search							
2	12.2	Depth First Search							
3	12.3	Best First Search							
4	12.4	Breadth First Traversal							
5	12.5	Depth First Search for Disconnected graph							
6			4		4	4	4	4	20
7									
8									
9									
10									
11									
12									

Santhosh

Signature of the Student

Signature of the Faculty

12. Graph Traversal.

12.1 Breadth First Search :-

AIM :- Write a Program for a given graph G, Print BFS traversal from a given Source vertex.

PROGRAM :-

```
import java.util.*;
Public class Graph {
  Private Map<Integer, List<Integer>> graph;
  Public Graph() {
    graph = new HashMap<>();
  }
  Public void addEdge (int u, int v) {
    graph.computeIfAbsent (u, K -> new ArrayList<>()).
    add(v);
  }
  Public void BFS (int Start) {
    Set<Integer> visited = new HashSet<>();
    queue<Integer> queue = new LinkedList<>();
    queue.add (Start);
```

```
        visited.add(start);
        while (!queue.isEmpty()) {
            int node = queue.poll();
            System.out.print(node + " ");
            List<Integer> neighbors = graph.getOrDefault
                (node, new ArrayList<>());
            for (int neighbor : neighbors) {
                if (!visited.contains(neighbor)) {
                    queue.add(neighbor);
                    visited.add(neighbor);
                }
            }
        }
    }

    Public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Graph graph = new Graph();
        System.out.println("Enter the number of edges:");
        int edges = scanner.nextInt();
        System.out.println("Enter the edges (u v):");
        for (int i = 0; i < edges; i++) {
```

```
        int u = scanner.nextInt();
        int v = scanner.nextInt();
        graph.addEdge(u, v);
    }
    System.out.println("Enter the starting vertex for BFS:");
    int start = scanner.nextInt();
    System.out.println("Following is Breadth First
     Traversal (starting from vertex " + start + "): ");
    graph.BFS(start);
    scanner.close();
    }
}
```

OUTPUT :-

Enter the number of edges : 5

Enter the edges (u v) : 0 1
0 2
1 2
2 0
2 3

Enter the starting vertex for BFS : 2

Following is BFS (starting from vertex 2) : 2 0 3 1 .

12.2 Depth First Search :-

AIM :- Write a Program for a given graph G, Print DFS traversal from a given Source Vertex.

PROGRAM :-

```
import java.util.*;
class Graph {
 Private Map<Integer, List<Integer>> graph;
 Public Graph() {
   graph = new HashMap<>();
}
Public void addEdge (int u, int v) {
 graph.computeIfAbsent (u, K -> new ArrayList<>()).
   add(v);
}
Private void DFSUtil (int V, Set<Integer> visited) {
 visited.add(v);
 System.out.print (v + " ");
 for (int neighbor : graph.getOrDefault
    (V, new ArrayList<>())) {
  if ( ! visited.contains (neighbor)) {
```

```
        DFSUtil(neighbor, visited);
    }
  }
}

Public void DFS (int start) {
  Set<Integer> visited = new HashSet<>();
  DFSUtil (start, visited);
}

Public static void main (String[] args) {
  Scanner scanner = new Scanner (System.in);
  Graph graph = new Graph();
  System.out.println (" Enter the number of vertices : ");
  int n = scanner.nextInt();
  System.out.println (" Enter the number of edges : ");
  int e = scanner.nextInt();
  System.out.println (" Enter the edges (u v) : ");
  for (int i= 0; i<e; i++) {
    int u = scanner.nextInt();
    int v = scanner.nextInt();
    graph.addEdge(u, v);
  }
```

```
        System.out.println("Enter the Starting Vertex
                            for DFS:");
        int start = scanner.nextInt();
        System.out.println("DFS from Vertex" +
                            start + ":");
        graph.DFS(start);
        scanner.close();
    }
}
```

OUTPUT :-

```
Enter the number of Vertices : 5
Enter the number of edges : 5
Enter the edges (u v) :
0 1
0 2
1 3
1 4
3 4
Enter the Starting Vertex for DFS : 0
DFS from Vertex 0 : 0 1 3 4 2
```

ROLL NUMBER :

12.3 Best First Search (Informed Search) :-

AIM :- The idea of BFS is to use an evaluation function to decide which adjacent is most promising and then explore.

PROGRAM :-

```
import java.util.*;
Public class BestFirstSearch {
  Static List< List < pair>> graph = new ArrayList<>();
  Static class pair {
    int node, cost;
    Pair (int node, int cost) {
      this.node = node;
      this.cost = cost;
    }
  }
  Static void addEdge (int x, int y, int cost) {
    graph.get(x).add(new Pair (y, cost));
    graph.get(y).add(new Pair (x, cost));
  }
  Static void bestFirstSearch (int Source, int target) {
```

```
PriorityQueue<Pair> pq = new PriorityQueue<>
    (Comparator.comparingInt(p -> p.cost));
boolean[] visited = new boolean[graph.size()];
Pq.add(new Pair(source, 0));
Visited[source] = true;
while (!pq.isEmpty()) {
    Pair current = pq.poll();
    int currentNode = current.node;
    System.out.print(currentNode + " ");
    if (currentNode == target) {
        System.out.println("\nGoal node" + target + "
                reached.");
        return;
    }
    for (Pair neighbor : graph.get(currentNode)) {
        if (!visited[neighbor.node]) {
            visited[neighbor.node] = true;
            pq.add(neighbor);
        }
    }
}
```

```
        System.out.println("In Goal node" + target + " not
    }                                    reachable");
    Public static void main(String[] args) {
        Scanner Scanner = new Scanner(System.in);
        System.out.println("Enter the number of vertices:");
        int vertices = Scanner.nextInt();
        System.out.println("Enter the number of edges:");
        int edges = Scanner.nextInt();
        for (int i=0; i< vertices ; i++) {
            Graph.add(new ArrayList<>());
        }
        System.out.println("Enter the edges (u v cost):");
        for (int i=0; i< edges ; i++) {
            int u = Scanner.nextInt();
            int v = Scanner.nextInt();
            int cost = Scanner.nextInt();
            addEdge(u, v, cost);
        }
        System.out.println("Enter the Source node:");
```

```
        int source = scanner.nextInt();
        System.out.println("Enter the target node:");
        int target = scanner.nextInt();
        System.out.println("Best First Search traversal:");
        bestFirstSearch(source, target);
        scanner.close();
    }
}
```

OUTPUT :-

```
Enter the number of vertices : 6
Enter the number of edges : 7
Enter the edges (u v cost):
0 1 1
0 2 4
1 3 2
1 4 5
2 4 1
3 5 3
4 5 2
Enter the Source node : 0
```

Enter the target node : 5

Best First traversal : 0 1 3 5

Goal node 5 reached.

12.4 Breadth First Traversal Of A Graph :-

AIM :- Write a program for a directed graph that the task is to do Breadth First Traversal of the graph Starting from 0.

PROGRAM :-

```
import java.util.*;
class Graph {
 Private int V;
 Private List < List < Integer >> adj;
 Graph (int v) {
  V = v;
  adj = new ArrayList <>();
  for (int i=0; i<v; i++) adj.add (new Arraylist<>());
}
void addEdge (int v, int w) {
```

```
    adj.get(v).add(w);
  }
  void BFS (int start) {
    boolean [] visited = new boolean [v];
    Queue < Integer > q = new LinkedList <> ();
    q.add (start);
    visited [ start] = true;
    System.out.print (" BFS Traversal : ");
    while (! q.isEmpty ()) {
      int node = q.poll();
      System.out.print (node + " ");
      for (int neighbor : adj.get (node)) {
        if (! visited [neighbor]) {
          visited [neighbor] = true;
          q.add (neighbor);
        }
      }
    }
  }
}
Public static void main (String [] args) {
  Scanner sc = new Scanner (System.in);
  System.out.println (" Enter vertices and edges : ");
```

```
        Graph g = new Graph (sc.nextInt());
        int edges = sc.nextInt();
        System.out.println("Enter Starting Vertex:");
        g.BFS(sc.nextInt());
        sc.close();
    }
}
```

OUTPUT :-

```
Enter Vertices and edges : 5 4
Enter edges (from to) :
0 1
0 2
0 3
2 4
Enter Starting Vertex : 0
BFS Traversal : 0 1 2 3 4
```

12.5 Depth First Search for Disconnected Graph :-

AIM :- Given a Disconnected Graph, the task is to implement DFS Algorithm.

PROGRAM :-

```
import java.util.*;
class Graph {
  Private Map<Integer, List<Integer>> adj;
: Public Graph() {
    adj = new HashMap<>();
  }
  void addEdge(int u, int v) {
    adj.putIfAbsent(u, new ArrayList<>());
    adj.putIfAbsent(v, new ArrayList<>());
    adj.get(u).add(v);
    adj.get(v).add(u);
  }
  Private void DFSUtil(int v, boolean[] Visited) {
    visited[v] = true;
    System.out.print(v + " ");
    for (int neighbor : adj.getOrDefault(v, new ArrayList<>())) {
      if (!visited[neighbor]) DFSUtil(neighbor, Visited);
    }
}
```

```
Public static void main (String[] args) {
    Scanner sc = new Scanner (System.in);
    Graph g = new Graph();
    System.out.println ("Enter number of vertices & edges:");
    int v = sc.nextInt(), E = sc.nextInt();
    System.out.println ("Enter edges (u v):");
    for (int i=0; i<E; i++) g.addEdge (sc.nextInt(),
      sc.nextInt());
    System.out.println ("DFS Traversal:");
    g.DFS();
    sc.close();
  }
}
```

OUTPUT :-

```
Enter number of vertices & edges: 6 5
Enter edges (u v):
0 1
0 2
1 3
1 4
4 5
```

DFS Traversal :

0 1 3 4 5 2.

VIVA VOCE :-

1) Define Breadth First Search (BFS) ?

A) It is a graph traversal algorithm that explores all neighbors of a node before moving to the next level neighbors. It uses a queue to ensure nodes are visited in level order, making it suitable for finding the shortest path in an unweighted graph.

2) Define Depth First Search (DFS) ?

A) It is a graph traversal algorithm that explores a path fully before backtracking to explore other paths. It uses a stack or recursion and is ideal for tasks like path finding, cycle detection, and connected component identification.

3) How does DFS handle Disconnected Graph ?

A) In a Disconnected Graph, DFS iterates over all vertices and performs DFS from any unvisited vertex, ensuring that all components are explored.

4) What data Structure is typically used in Best First Search ?

A) A Priority queue (or heap) is used in Best First Search to store and retrieve nodes based on the evaluation function value (cost).