Lab Assignment - 4

21BCE9905

1. Write a program to find the Strongly connected components in a digraph.

```
Public class digraph {
  Public Static void main(String[] args) {
      GFG obj = new GFG();
      int v=5;
      List < List <Integer >> edges = new ArrayList<>();
  edges.add(new ArrayList<>(List.of(1, 3)));
  edges.add(new ArrayList<>(List.of( )));
  edges.add(new ArrayList<>(List.of( )));
  edges.add(new ArrayList<>(List.of( )));
  edges.add(new ArrayList<>(List.of( )));
  List <List <Integer>> ans = obj.findScc(v, edges);
  System.out.println("Strongly connected components are:");
  for (List < Integer> x : ans) {
      for (int y : x) {
          System.out.println(y + " ");
      }
      System.out.println();
  }
  }
}
```

```
Class GFG {
boolean dfs (int curr, int des, List<List<Integer>> adj,
                                List<Integer> vis ){
 if (curr == des) {
  return true;
 }
 vis.set(curr, element : 1);
 for (int x : adj.get(curr)) {
  if (vis.get(x) == 0){
   if (dfs (x, des, adj, vis)) {
     return true;
   }
  }
 }
 return false;
}

boolean isPath (int src, int des, List<List<Integer>> adj){
 List<Integers vis = new ArrayList<>(adj.size() + 1);
for (int i=0; i<=adj.size(); i++){
    vis.add(0);
}
return dfs(src, des, adj, vis);
}

List<List<Integer>> findSCC (int n, List<List<Integer>>adj
 List<List<Integers> ans = new ArrayList<>();
 List<Integer> is_SCC = new ArrayList<>();
for (int i=0; i<=n; i++){
  adj.add(new ArrayList<>());
 }
```

```
for (List<Integer> edge){
  adj.get(edge.get(0)).add(edge.get(1));
}
for (int i=1; i<=n; i++){
  if (is_scc.get(i)==0){
    List<Integer> scc = new ArrayList<>();
    scc.add(i);
    for (int j=i+1; j<=n; j++){
      if (is_scc.get(j)==0 && isPath(i,j,adj)
          && isPath(j,i,adj)){
        is_scc.set(j,1);
        scc.add(j);
      }
    }
    ans.add(scc);
  }
  return ans;
}
}
```

Output: Strongly Connected components are;

1 2 3
4
5

2. Write a program to implement Dynamic programming algorithm to Solve all pairs shortest problem.

```
import java.util.*;
import java.lang.*;
import java.io.*;

class AllpairShortestPath {
  final static int INF = 99999, V = 4;
  void floydWarshall(int graph[][]) {
    int dist[][] = new int[V][V];
    int i, j, k;
    for(i=0; i<V; i++){
      for(j=0; j<V; j++)
        dist[i][j] = graph[i][j];

    for(k=0; k<V; k++) {
      for(i=0; i<V; i++){
        for(j=0; j<V; j++){
          if(dist[i][k] + dist[k][j] < dist[i][j])
            dist[i][j] = dist[i][k] + dist[k][j];
        }
      }
    }
    printSolution(dist);
  }
  void printSolution(int dist[][]) {
    System.out.println("The following matrix shows the shortest
                        "distance between every pair of vertices");
```

```
for (int i=0; i<V; ++i){
    for (int j=0; j<V; ++j){
        if (dist[i][j] == INF)
            System.out.print("INF");
        else
            System.out.print(dist[i][j]+"    ");
    }
    System.out.print(n());
  }
}
public static void main(String[] args) {
    int graph[][] = { {0, 5, INF, 10},
                      {INF, 0, 3, INF},
                      {INF, INF, 0, 1},
                      {INF, INF, INF, 0}
                    };
    AllPairShortestPath a = new AllPairShortestPath();
    a.floydWarshall(graph);
  }
}
```

output

```
0    5    8    9
INF  0    3    4
INF  INF  0    1
INF  INF  INF  0
```

3. Write a program to find shortest paths to other vertices from a given vertex in a weighted connected graph, using Dijkstra's algorithm.

```
import java.io.*;
import java.lang.*;
import java.util.*;
Public class ShortestPath {
  Static final int V=9;
  int minDistance(int dist[], Boolean sptSet[]){
    int min = Integer.MAX_VALUE, min_index=-1;
    for(int v=0; v<V; v++){
      if (sptSet[v] == false && dist[v] <= min){
        min = dist[v];
        min_index = v;
      }
    } return min_index;
  }
  void printSolution(int dist[]){
    System.out.println(
      "vertex \t\t Distance from Source");
    for(int i=0; i<V; i++){
    System.out.println(i+ "\t\t" + dist[i]);
  }
  void dijkstra(int graph[][], int src){
  int dist[] = new int[V];
  Boolean sptSet[] = new Boolean[V];
```

```
for (int i=0; i<V; i++){
    dist[i] = Integer.MAX_VALUE;
    sptSet[i] = false;
}
dist[src] = 0;
for (int count=0; count<V-1; count++){
  int u = minDistance(dist, sptSet);
  sptSet[u] = true;
  for (int v=0; v<V; v++){
    if (!sptSet[v] && graph[u][v] != 0
      && dist[u] != Integer.MAX_VALUE
      && dist[u] + graph[u][v] < dist[v])
    dist[v] = dist[u] + graph[u][v];
}
Public Static void main(String[] args)
{ int graph[][] =
new int[][]{{0 4 0 0 0 0 0 8 0}
            {4 0 8 0 0 0 0 11 0}
            {0 8 0 7 0 4 0 0 2}
            {0 0 7 0 9 14 0 0 0}
            {0 0 0 9 0 10 0 0 0}
            {0 0 4 14 10 0 2 0 0}
            {0 0 0 0 0 2 0 1 6}
            {8 11 0 0 0 0 1 0 7}
            {0 0 2 0 0 0 6 7 0}};
Shortest path t = new Shortestpath();
    t.dijkstra(graph,0);
}
}
```

Output :

Vertex	Distance From Source
0	0
1	4
2	12
3	19
4	21
5	11
6	9
7	8
8	14