Design and Analysis of Algorithms Lab (CS263)

Lab Assignment 8

November 9, 2019

Course Instructor: Dr. Keyur Parmar

Student and ID : Anvay Mishra 201851026

- **Write a code to solve each of the following problems For each problem, solve it using the following methods.**

  1. **Adjacency matrix Depth-first search**

```
class DFS_Matrix
{
        private int V;
    private int adj[][];
        DFS_Matrix(int v)
        {
                V = v;
                adj =new int[v][v];
        }
        void addEdge(int v, int w)
        {
        adj[v][w]=1;
        }
        void DFSUtil(int v,boolean visited[])
        {
                visited[v] = true;
                System.out.print(v+" ");
                for(int i=0;i<V;i++)
                {
            if(adj[v][i]==1&&visited[i]==false){
                DFSUtil(i,visited);
            }
        }
        }
        void DFS(int v)
        {
                boolean visited[] = new boolean[V];
                DFSUtil(v, visited);
        }

        public static void main(String args[])
        {
                DFS_Matrix g = new DFS_Matrix(5);

                g.addEdge(0, 1);
                g.addEdge(0, 2);
                g.addEdge(2, 4);
                g.addEdge(0, 3);
        g.addEdge(1, 2);


                System.out.println("Following is Depth First Traversal "+
                                                "(starting from vertex 2)");

        g.DFS(2);
        System.out.println();
        }
}
```

2. **Adjacency matrix Breadth-first search**

```
import java.util.*;
import java.io.*;
class BFS_Matrix
{
        private int V;
    private int adj[][];
        BFS_Matrix(int v)
        {
                V = v;
                adj =new int[v][v];
        }
        void addEdge(int v, int w)
        {
        adj[v][w]=1;
        }
        void BFSUtil(int v,boolean visited[])
        {

        Queue arr= new LinkedList<Integer>();
        arr.add(v);
        visited[v] = true;
        while(!arr.isEmpty()){
            int m=(int)arr.peek();
            System.out.print(m+" ");
            arr.poll();
                    for(int i=0;i<V;i++)
                    {
                if(adj[m][i]==1&&visited[i]==false){
                    arr.add(i);
                }
            }
        }
        }
        void BFS(int v)
        {
        boolean visited[] = new boolean[V];
                BFSUtil(v, visited);
        }

        public static void main(String args[])
        {
                BFS_Matrix g = new BFS_Matrix(6);

                g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 3);
        g.addEdge(1, 4);
        g.addEdge(2, 5);


                System.out.println("Following is Depth First Traversal "+
                                                "(starting from vertex 2)");

        g.BFS(0);
        // System.out.println(Arrays.toString(arr));
        }
}
```

3. **Adjacency List Depth-first search**

```
import java.util.*;
import java.util.LinkedList;
class Graph
{
        private int V;
    private LinkedList<Integer> adj[];
        Graph(int v)
        {
                V = v;
                adj = new LinkedList[v];
                for (int i=0; i<v; ++i)
                        adj[i] = new LinkedList();
        }
        void addEdge(int v, int w)
        {
                adj[v].add(w);
        }
        void DFSUtil(int v,boolean visited[])
        {
                visited[v] = true;
                System.out.print(v+" ");
                Iterator<Integer> i = adj[v].listIterator();
                while (i.hasNext())
                {
                        int n = i.next();
                        if (!visited[n])
                                DFSUtil(n, visited);
                }
        }
        void DFS(int v)
        {
                boolean visited[] = new boolean[V];
                DFSUtil(v, visited);
        }

        public static void main(String args[])
        {
                Graph g = new Graph(4);

                g.addEdge(0, 1);
                g.addEdge(0, 2);
                g.addEdge(1, 2);
                g.addEdge(2, 0);
                g.addEdge(2, 3);
                g.addEdge(3, 3);

                System.out.println("Following is Depth First Traversal "+
                                                "(starting from vertex 2)");

        g.DFS(2);
        System.out.println();
        }
}
```

4. **Adjacency List Breadth-first search**

```
import java.io.*;
import java.util.*;
class BFS_List
{
        private int V;
        private LinkedList<Integer> adj[];
```

```
BFS_List(int v)
{
        V = v;
        adj = new LinkedList[v];
        for (int i=0; i<v; ++i)
                adj[i] = new LinkedList();
}
void addEdge(int v,int w)
{
        adj[v].add(w);
}
void BFS(int s)
{
        boolean visited[] = new boolean[V];
        LinkedList<Integer> queue = new LinkedList<Integer>();
        visited[s]=true;
        queue.add(s);

        while (queue.size() != 0)
        {
                s = queue.poll();
                System.out.print(s+" ");
                Iterator<Integer> i = adj[s].listIterator();
                while (i.hasNext())
                {
                        int n = i.next();
                        if (!visited[n])
                        {
                                visited[n] = true;
                                queue.add(n);
                        }
                }
        }
}
public static void main(String args[])
{
        BFS_List g = new BFS_List(4);

        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        System.out.println("Following is Breadth First Traversal "+
                                                "(starting from vertex 2)");
```

```
          g.BFS(2);
          System.out.println();
          }
}
```