SKILL-6

2100031407

Sec no:10

1.

```
 class Skill6 {
  private Map<Integer, List<Integer>> graph;
   private Map<Integer, Integer> rank;
   private Map<Pair<Integer, Integer>, Boolean> connDict;

   public List<List<Integer>> criticalConnections(int n, List<List<Integer>>
connections)
 {
     this.formGraph(n, connections);
       this.dfs(0, 0);

       List<List<Integer>> result = new ArrayList<List<Integer>>();
       for (Pair<Integer, Integer> criticalConnection : this.connDict.keySet()) {
           result.add(new
ArrayList<Integer>(Arrays.asList(criticalConnection.getKey(),
criticalConnection.getValue())));
       }
```

```
        return result;
    }


    private int dfs(int node, int discoveryRank)
        if (this.rank.get(node) != null)
{
 return this.rank.get(node);
  }
      this.rank.put(node, discoveryRank);
       int minRank = discoveryRank + 1;
       for (Integer neighbor : this.graph.get(node))
{
            Integer neighRank = this.rank.get(neighbor);
            if (neighRank != null && neighRank == discoveryRank - 1)
{
continue;
          }
             int recursiveRank = this.dfs(neighbor, discoveryRank + 1);
            if (recursiveRank <= discoveryRank) {
                int sortedU = Math.min(node, neighbor), sortedV = Math.max(node,
neighbor);
                this.connDict.remove(new Pair<Integer, Integer>(sortedU, sortedV));
    }
           minRank = Math.min(minRank, recursiveRank);
        }


        return minRank;
    }

    private void formGraph(int n, List<List<Integer>> connections) {

        this.graph = new HashMap<Integer, List<Integer>>();
        this.rank = new HashMap<Integer, Integer>();
        this.connDict = new HashMap<Pair<Integer, Integer>, Boolean>();
                for (int i = 0; i < n; i++) {
            this.graph.put(i, new ArrayList<Integer>());
            this.rank.put(i, null);
        }

        for (List<Integer> edge : connections) {
            int u = edge.get(0), v = edge.get(1);
            this.graph.get(u).add(v);
            this.graph.get(v).add(u);
            int sortedU = Math.min(u, v), sortedV = Math.max(u, v);
            connDict.put(new Pair<Integer, Integer>(sortedU, sortedV), true);
        }
    }
}
```

2.

```
import java.io.*;
import java.util.*;
 class Solution {
    private int G;
   private ArrayList<ArrayList<Integer> > adj;
    Graph(int g)
    {
        G= g;
        adj = new ArrayList<ArrayList<Integer> >(g);
        for (int i = 0; i < g; ++i)
            adj.add(new ArrayList<Integer>());
    }
    void addEdge(int g, int w) { adj.get(g).add(w); }
     void topologicalSortUtil(int g, boolean visited[],
                             Stack<Integer> stack)
    {
        visited[g] = true;
        Integer i;
        Iterator<Integer> it = adj.get(g).iterator();
        while (it.hasNext()) {
            i = it.next();
            if (!visited[i])
            topologicalSortUtil(i, visited, stack);
        }
        stack.push(new Integer(g));
    }
    void topologicalSort()
    {
        Stack<Integer> stack = new Stack<Integer>
        boolean visited[] = new boolean[V];
        for (int i = 0; i < G; i++)
            visited[i] = false;
         for (int i = 0; i < G; i++)
            if (visited[i] == false)
                topologicalSortUtil(i, visited, stack);
         while (stack.empty() == false)
            System.out.print(stack.pop() + " ");
    }
     public static void main(String args[])
    {
        Graph h= new Graph(6);
        h.addEdge(5, 2);
        h.addEdge(5, 0);
        h.addEdge(4, 0);
        h.addEdge(4, 1);
        h.addEdge(2, 3);
        h.addEdge(3, 1);
System.out.println("Following is a Topological + "sort of the given graph");
h.topologicalSort();
```

```
}}
```