**Experiment 2.1**

**Competitive Coding Lab 5(Graphs)**

**Student Name: Sahul Kr. Parida UID: 20BCS4919**

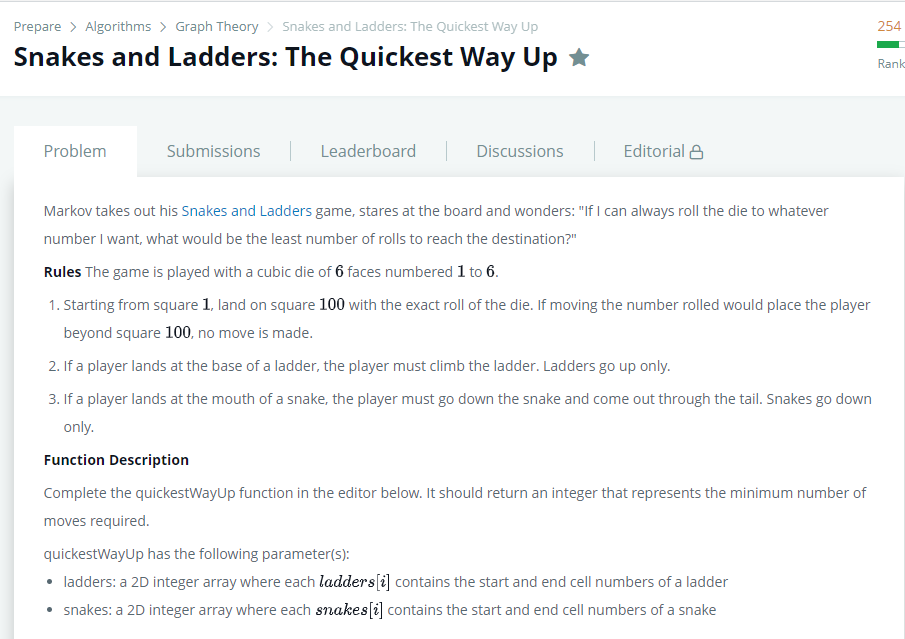
**Branch: CSE Section/Group: WM-904/B**

**Semester: 5th Date of Performance: 10/10/22**

**Subject Name: Competitive Coding(CC) Subject Code: 20CSP-314**

**PROBLEM STATEMENT 5.1: -**

<https://www.hackerrank.com/challenges/the-quickest-way-up/problem?isFullScreen=true>



**SOLUTION:**

import java.util.ArrayList;

import java.util.Scanner;

public class Solution {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int t = scanner.nextInt();

        for (int i = 0; i < t; i++) {

            String[] data = scanner.next().split(",");

            int numberOfLadders = Integer.valueOf(data[0]);

            int numberOfSnakes = Integer.valueOf(data[1]);

            ArrayList <Integer> ladderStartingPoints = new ArrayList <Integer> ();

            ArrayList <Integer> ladderEndingPoints = new ArrayList <Integer> ();

            ArrayList <Integer> snakeStartingPoints = new ArrayList <Integer> ();

            ArrayList <Integer> snakeEndingPoints = new ArrayList <Integer> ();

            for (int j = 0; j < numberOfLadders; j++) {

                String[] ladderData = scanner.next().split(",");

                ladderStartingPoints.add(Integer.valueOf(ladderData[0]));

                ladderEndingPoints.add(Integer.valueOf(ladderData[1]));

            }

            for (int j = 0; j < numberOfSnakes; j++) {

                String[] snakeData = scanner.next().split(",");

                snakeStartingPoints.add(Integer.valueOf(snakeData[0]));

                snakeEndingPoints.add(Integer.valueOf(snakeData[1]));

            }

            State initialState = new State(1, 0);

            ArrayList <Integer> visited = new ArrayList <Integer> ();

            ArrayList <State> list = new ArrayList <State> ();

            list.add(initialState);

            visited.add(1);

 while (!list.isEmpty()) {

                State top = list.get(0);

                list.remove(0);

                if (top.currentCell == 100) {

                    System.out.println(top.moves);

                    break;

                }

                for (int j = 1; j <= 6; j++) {

                    int temp = top.currentCell + j;

                    int ladderIndex = ladderStartingPoints.indexOf(temp);

                    int snakeIndex = snakeStartingPoints.indexOf(temp);

                    if (ladderIndex != -1) {

                        temp = ladderEndingPoints.get(ladderIndex);

                    }

                    else if (snakeIndex != -1) {

                        temp = snakeEndingPoints.get(snakeIndex);

                    }

                    if (!visited.contains(temp)) {

                        list.add(new State(temp, top.moves + 1));

                        visited.add(temp);

                    }

                }

            }

        }

        scanner.close();

    }

    public static class State {

        public int currentCell;

        public int moves;

        public State(int currentCell, int moves) {

            this.currentCell = currentCell;

            this.moves = moves;

        }

        public String toString() {

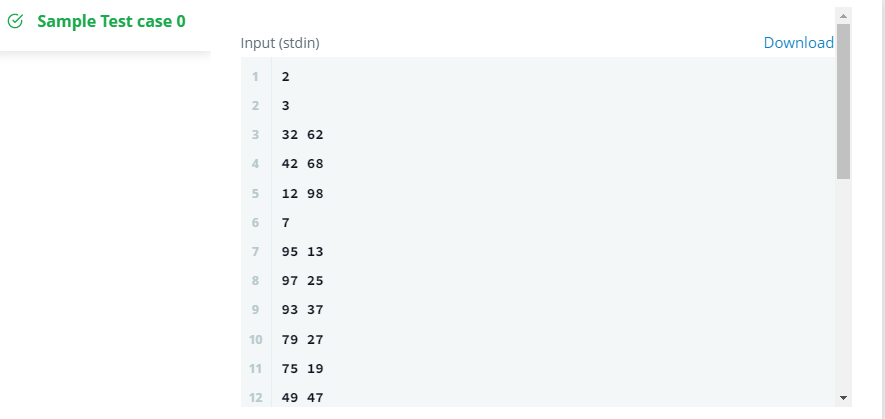
            return currentCell + " " + moves;

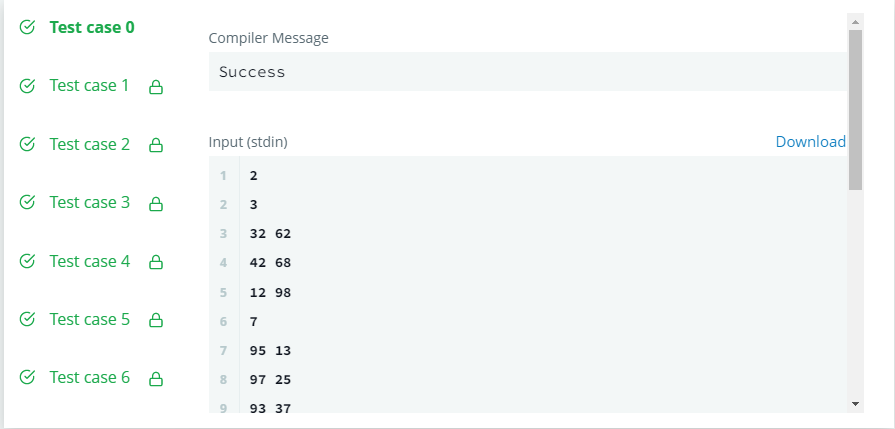
        }

    }

}

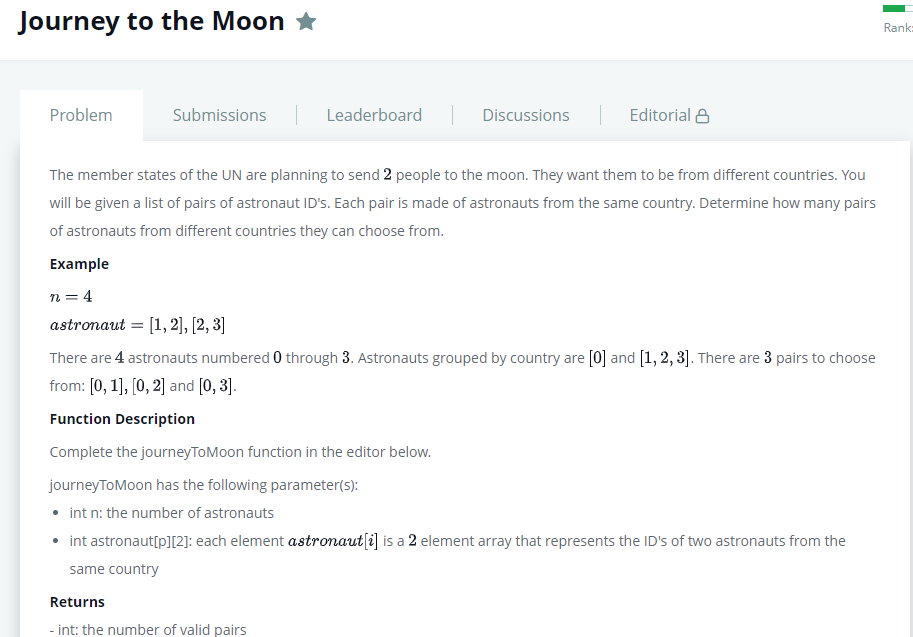
# TEST CASES:





**PROBLEM STATEMENT 5.2: -**

<https://www.hackerrank.com/challenges/journey-to-the-moon/problem?isFullScreen=false>



**SOLUTION:**

import java.io.\*;

import java.util.\*;

public class Solution

{

    public static void main(String[] args)

    {

        Scanner in = new Scanner(System.in);

        int N = in.nextInt();

        Graph graph = new Graph(N);

        int P = in.nextInt();

        for(int i = 0; i < P; i++)

        {

            int source = in.nextInt();

            int destination = in.nextInt();

            graph.addEdge(source,destination);

        }

        boolean[] visited = new boolean[N];

        List<Integer> countries = new ArrayList<Integer>();

        long combinations = 0;

        // store size of each country by traversing each cluster

        for(int i = 0; i < N; i++){

            if(!visited[i]){

                countries.add(graph.dfs(i, visited));

            }

        }

        /\* Let size of each country be A, B, C, D ...

         \* Combinations: AB + AC + AD + BC + BD + CD + ...

         \* => AB + (A+B)C + (A+B+C)D

         \*/

        int sum = 0;

        for(int country : countries){

            combinations += sum\*country;

            sum += country;

        }

        System.out.println(combinations);

    }

}

class Graph

{

    List<Integer>[] vertices;

    public Graph(int count)

    {

        vertices = new ArrayList[count];

        for(int i = 0; i < count; i++)

        {

            vertices[i] = new ArrayList<Integer>();

        }

    }

    public void addEdge(int source, int destination)

    {

        vertices[source].add(destination);

        vertices[destination].add(source);

    }

    // modified DFS to return number of vertices traversed

    public int dfs(int source, boolean[] visited)

    {

        visited[source] = true;

        int count = 1;

        for(Integer vertex: vertices[source])

        {

            if(!visited[vertex])

            {

                count += dfs(vertex, visited);

            }

        }

        return count;

    }

}

# TEST CASES:

# 

# 