

**UNIVERSITY INSTITUTE OF ENGINEERING**

**Department of Computer Science & Engineering**

**Subject Name:** Competitive Coding

**Subject Code:** 20CSP-314

**Submitted to: Submitted by:**

Er. Mamta Punia Name: Sahil Kaundal

UID: 21BCS8197

Section: 616

Group: A

**INDEX**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Ex. No** | **List of Experiments** | **Conduct (MM: 12)** | **Viva**  **(MM: 10)** | **Record (MM: 8)** | **Total**  **(MM: 30)** | **Remarks/Signature** |
| 1 | To demonstrate the concept of Array. |  |  |  |  |  |
| 2 | To demonstrate the concept of Stack and Queue. |  |  |  |  |  |
| 3 | **To demonstrate the concept of Linked List.** |  |  |  |  |  |
| 4 | Sorting and Searching: Implement the concept of Searching and Sorting techniques. |  |  |  |  |  |
| 5 | To implement the concept of Graphs. |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Experiment 5.1**

**Student Name:** Sahil Kaundal **UID:** 21BCS8197

**Branch:** BE CSE (Lateral Entry) **Section/Group:** 616/A

**Semester:** 5th **Date of Performance:** 07/10/2022

**Subject Name:** CC Lab **Subject Code:** 20CSP-314

1. **Aim/Overview of the practical:**

To implement the concept of Graphs.

Consider an undirected graph where each edge weighs 6 units. Each of the nodes is labeled consecutively from 1 to n.

You will be given a number of queries. For each query, you will be given a list of edges describing an undirected graph. After you create a representation of the graph, you must determine and report the shortest distance to each of the other nodes from a given starting position using the *breadth-first search* algorithm ([BFS](https://en.wikipedia.org/wiki/Breadth-first_search)). Return an array of distances from the start node in node number order. If a node is unreachable, return  for that node.

<https://www.hackerrank.com/challenges/bfsshortreach/problem?isFullScreen=true>

1. **Apparatus / Simulator Used:**

* Windows 7 or above
* Google Chrome

1. **Objective:**

* To understand the concept of graphs.
* To implement the concept of Graphs.

**4. Code:**

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

#include <queue>

#include<limits>

using namespace std;

struct entity

    {

    int node;

    int weight;

};

int main() {

    int T, N , M, from, to, s;

    entity e, e1;

    cin >> T;

    for(int i= 0; i < T; i++)

    {

        cin >> N >> M;

    vector<vector<int>> aList(N);

    vector<int> output(N,numeric\_limits<int>::max());

    vector<int> finished(N, -1);

    vector<int>::iterator it;

    for(int i = 0 ; i < M; i++)

        {

            cin >> from >> to;

            it = find (aList[from-1].begin(), aList[from-1].end(), to -1);

            if (it == aList[from-1].end())

            {

                aList[from-1].push\_back(to - 1);

            aList[to-1].push\_back(from - 1);

            }

    }

    cin >> s;

        output[s-1] = 0;

   //     cout << s << endl;

    queue<entity> myqueue;

  /\*      for(int i = 0 ; i < N; i++)

            {

            cout << i << "\t";

            for(int j = 0; j < aList[i].size(); j++)

                {

                cout << aList[i][j] << " ";

            }

            cout << endl;

        }\*/

    for(int v : aList[s-1])

        {

       // cout << v;

        e.node = v;

        e.weight = 6;

        myqueue.push(e);

    }

        finished[s-1] = 1;

    while (!myqueue.empty())

  {

        e = myqueue.front();

        //cout << e.node << " " << e.weight << endl;

        if (e.weight < output[e.node])

            {

            output[e.node] = e.weight;

        }

        finished[e.node] = 1;

        myqueue.pop();

        for(int v: aList[e.node])

            {

            if(finished[v] != 1)

                {

            e1.node = v;

            e1.weight = 6 + e.weight;

            myqueue.push(e1);

            }

        }

    }

    for( int i = 0 ; i < output.size() ;  i++)

        {

        if(i!= s-1)

            {

        if( output[i]!= numeric\_limits<int>::max())

            cout << output[i] << " ";

        else

            cout << -1 << " ";

        }

    }

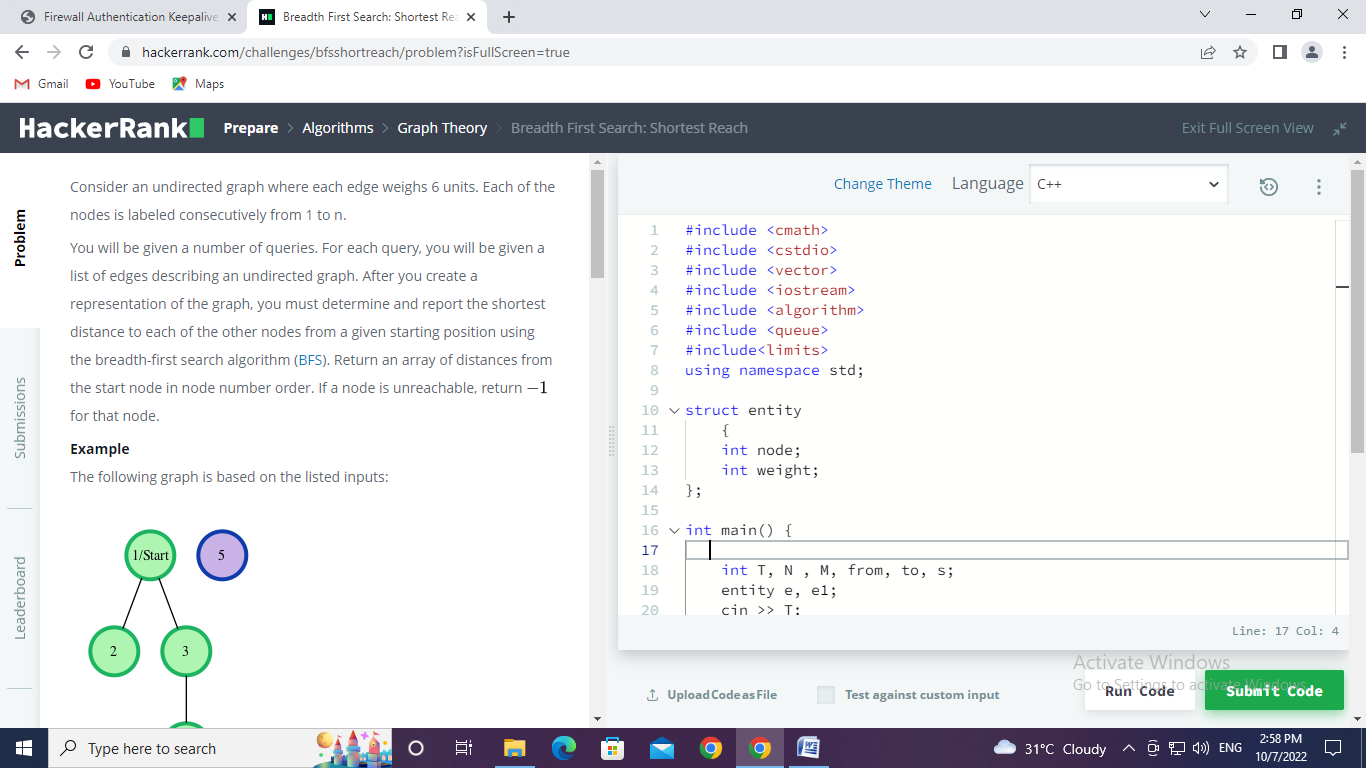
        cout << endl;

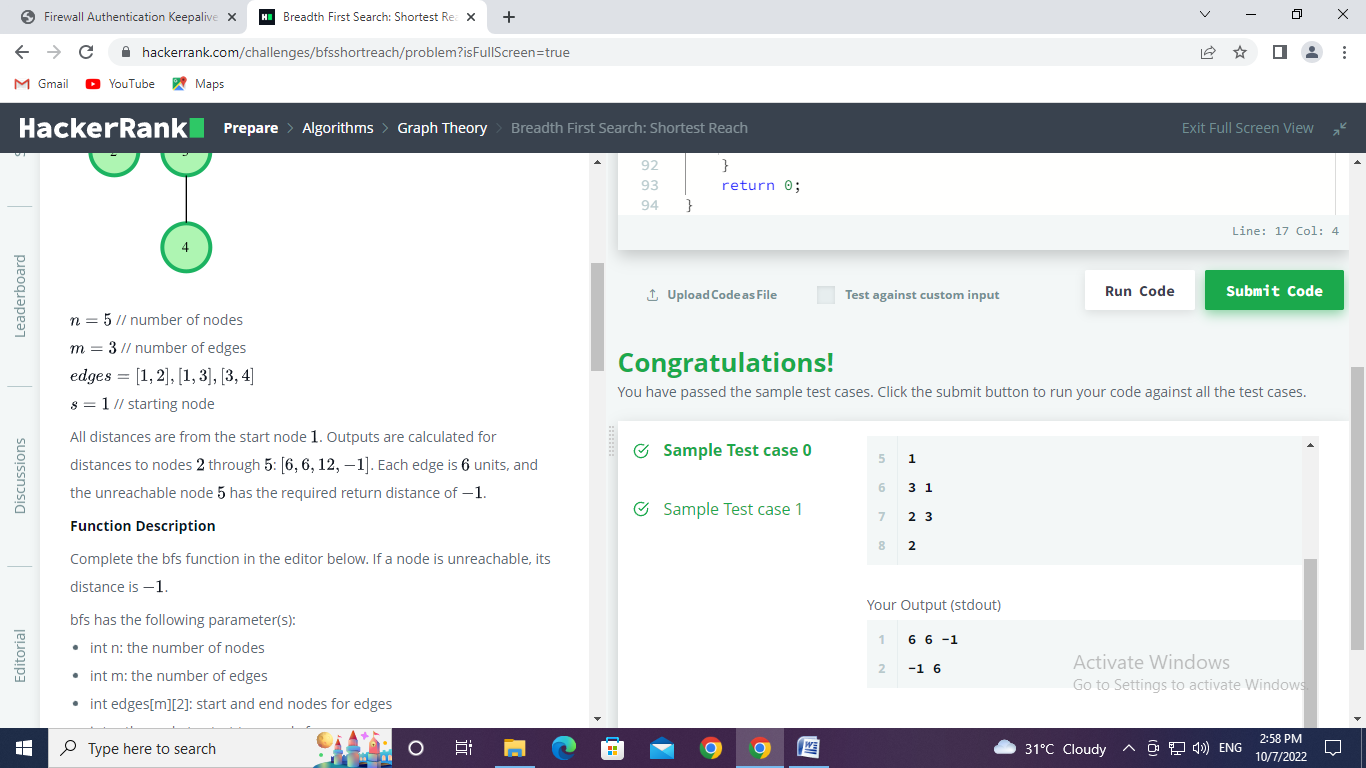
    }

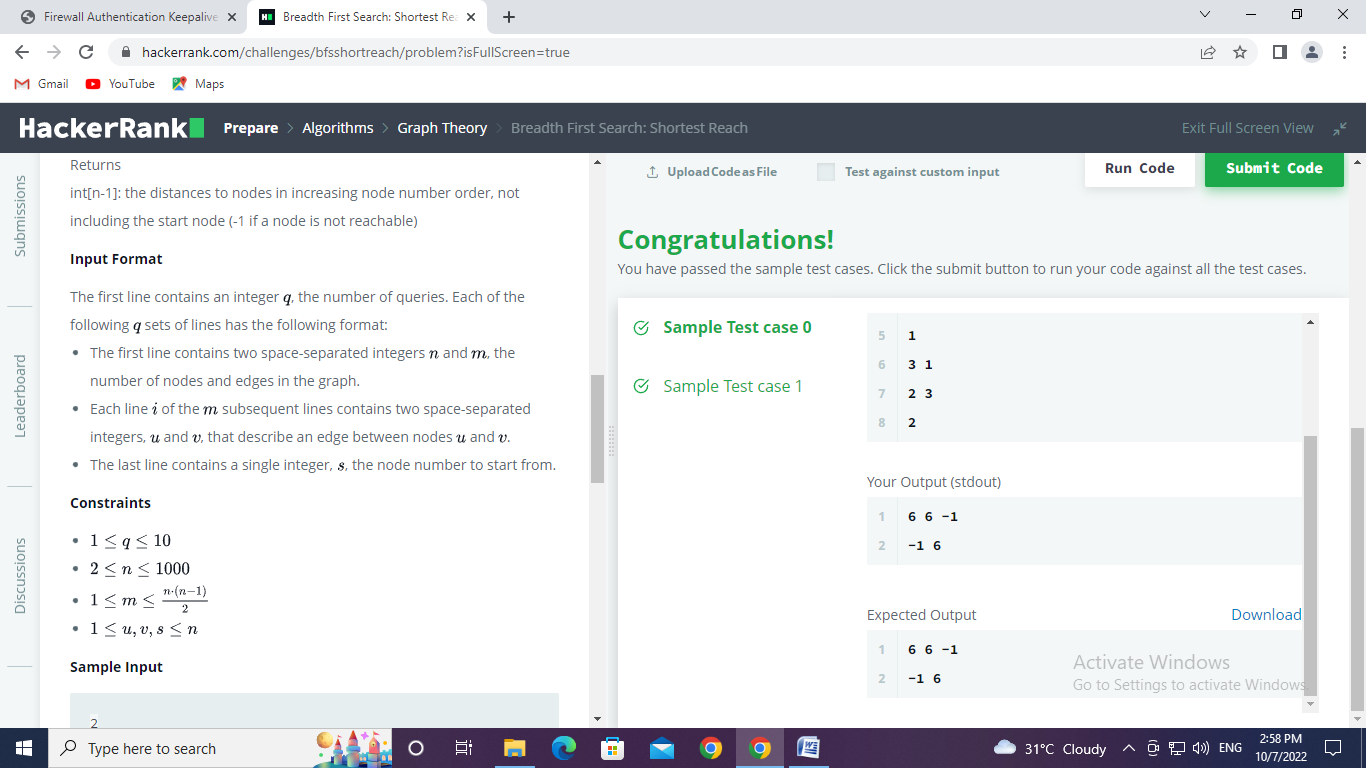
    return 0;

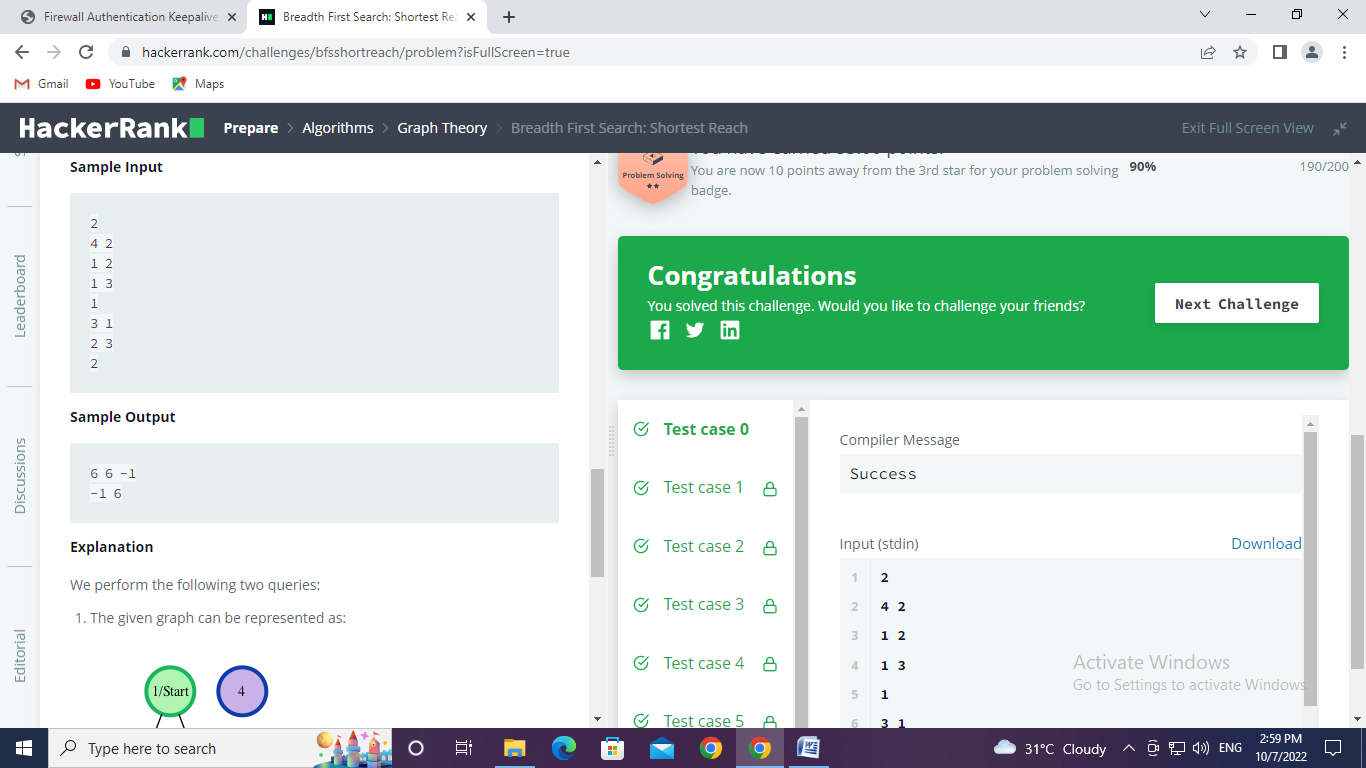
}

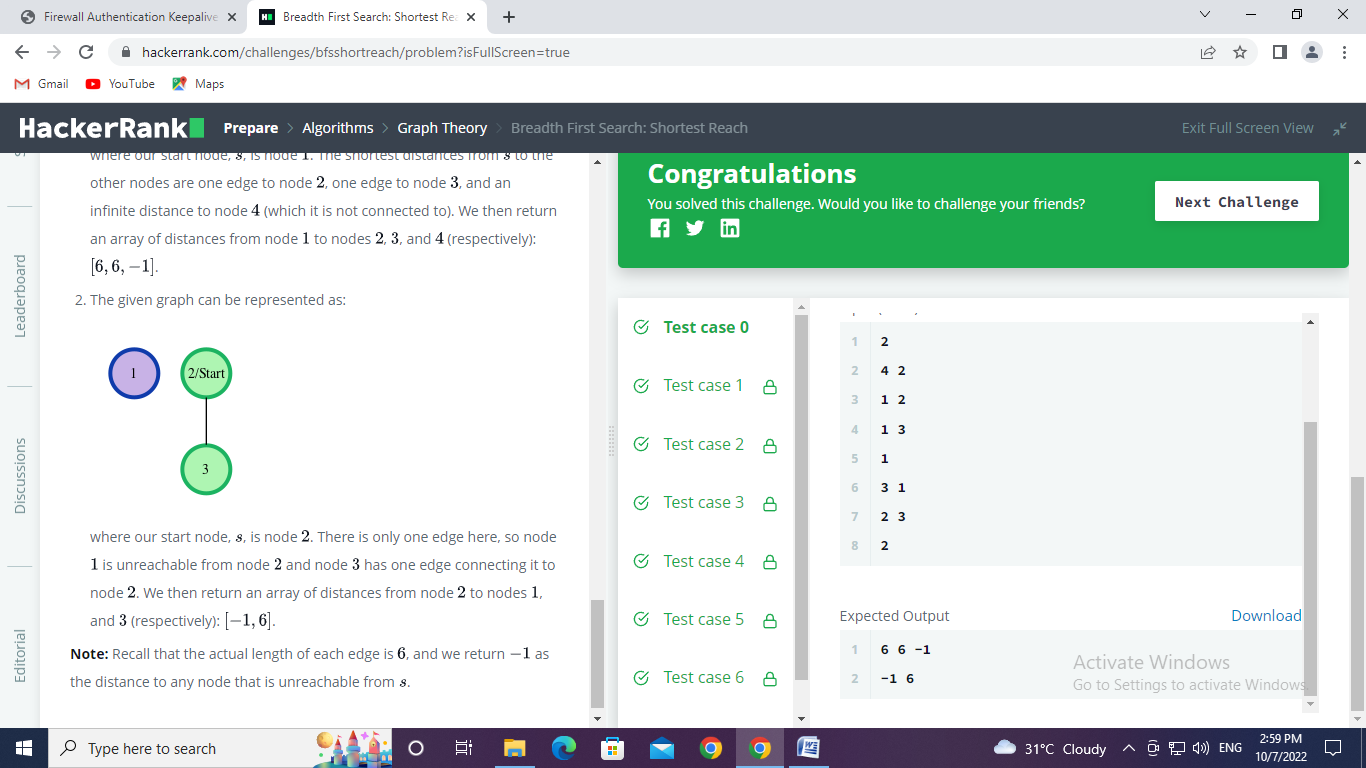
**5. Result/Output/Writing Summary:**











**Experiment 5.2**

1. **Aim/Overview of the practical:**

To implement the concept of Graphs.

Markov takes out his [Snakes and Ladders](http://en.wikipedia.org/wiki/Snakes_and_Ladders) game, stares at the board and wonders: "If I can always roll the die to whatever number I want, what would be the least number of rolls to reach the destination?"

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

1. **Apparatus / Simulator Used:**

* Windows 7 or above
* Google Chrome

1. **Objective:**

* To understand the concept of graphs.
* To implement the concept of Graphs.

1. **Code:**

import java.io.\*;

import java.util.\*;

public class Solution {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int T = sc.nextInt();

        int M,N;

        for (int i = 0; i < T; i++){

            N = sc.nextInt();

            HashMap<Integer,Integer> ladders = new HashMap<>();

            int start, end;

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

                start = sc.nextInt();

                end = sc.nextInt();

                ladders.put(start,end);

            }

            HashMap<Integer,Integer> snakes = new HashMap<>();

            M = sc.nextInt();

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

                start = sc.nextInt();

                end = sc.nextInt();

                snakes.put(start, end);

            }

            int[] distances = new int[100];

            for (int j = 0; j < 100; j++){

                distances[j] = Integer.MAX\_VALUE;

            }

            getShortestPathToEnd(getGameGraph(ladders, snakes), 1, distances, 0);

            System.out.println(distances[99] == Integer.MAX\_VALUE ? -1 : distances[99]);

        }

    }

    private static int getShortestPathToEnd(HashMap<Integer,HashSet<Integer>> graph, int start, int[] distances, int depth){

       if (distances[start-1] > depth){

           distances[start-1] = depth;

       }

       else{

           return 0;

       }

       if (!graph.get(start).isEmpty()){

           for (Integer child : graph.get(start)){

               //System.out.println(start + " - " + child);

               getShortestPathToEnd(graph, child, distances, depth + 1);

           }

           return 0;

       }

       else{

           return -1;

       }

    }

    private static HashMap<Integer,HashSet<Integer>> getGameGraph(HashMap<Integer,Integer> ladders, HashMap<Integer,Integer> snakes){

        HashMap<Integer, HashSet<Integer>> graph = new HashMap<>();

        HashSet<Integer> neighbours;

        for (int i = 1; i <= 100; i++){

            neighbours = new HashSet<Integer>();

            for (int j = 1; j <= 6 && (i + j <= 100); j++){

                if(ladders.containsKey(i+j)){

                    neighbours.add(ladders.get(i+j));

                }

                else if (snakes.containsKey(i+j)){

                    neighbours.add(snakes.get(i+j));

                }

                else{

                    neighbours.add(i+j);

                }

            }

            graph.put(i, neighbours);

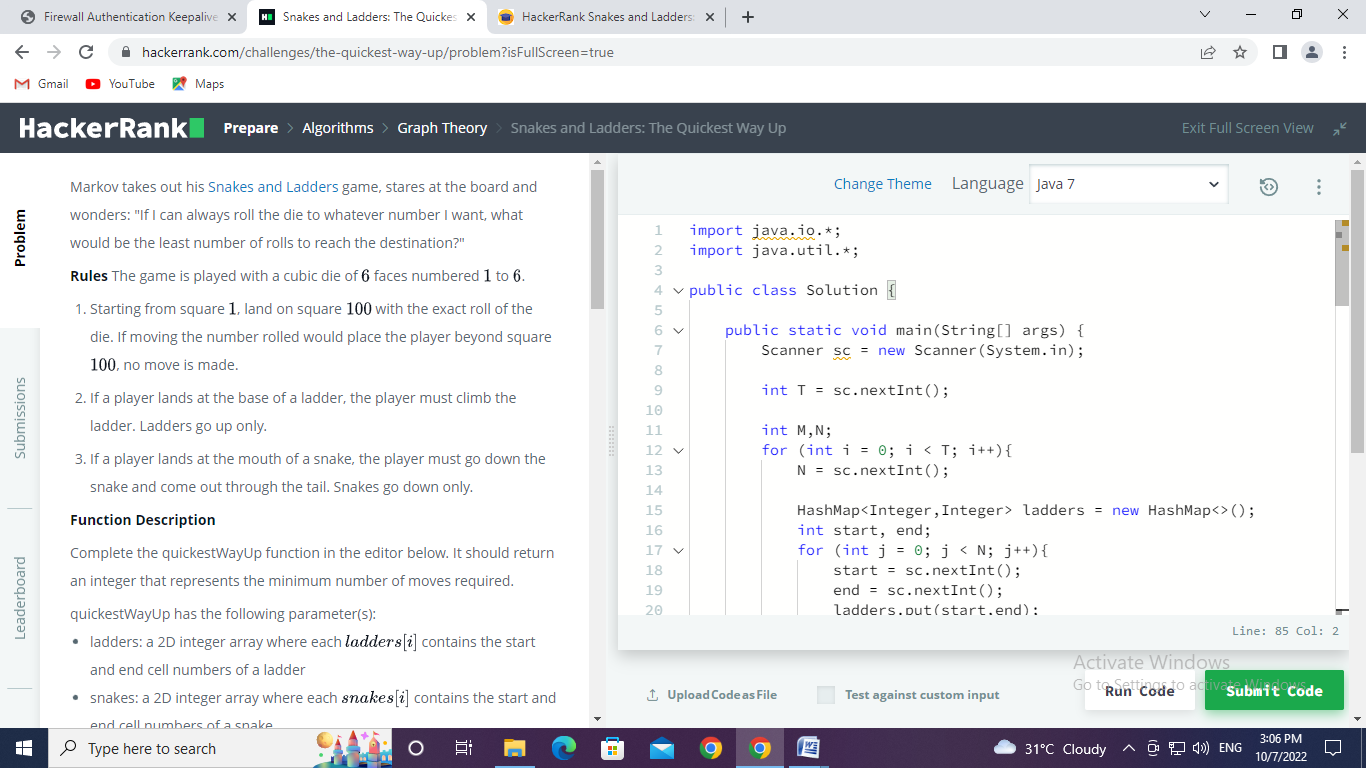
        }

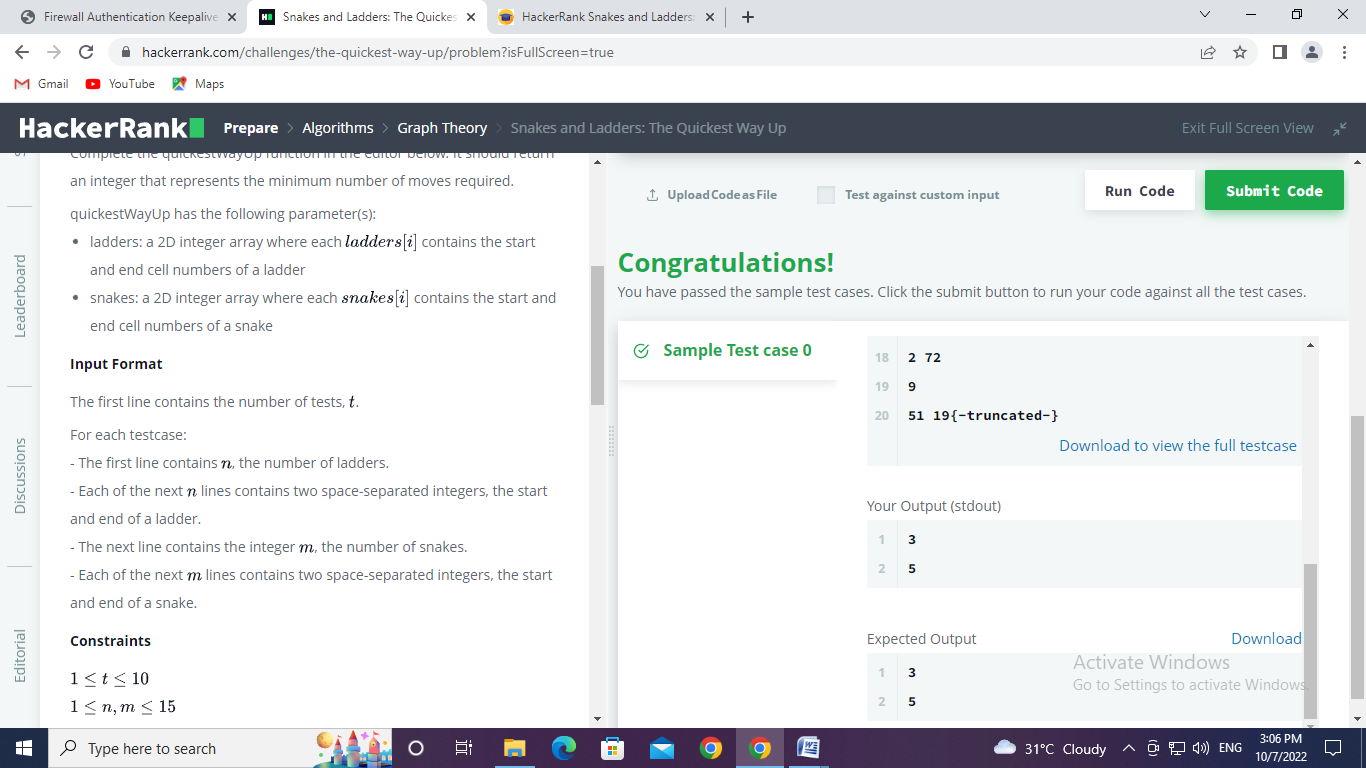
        return graph;

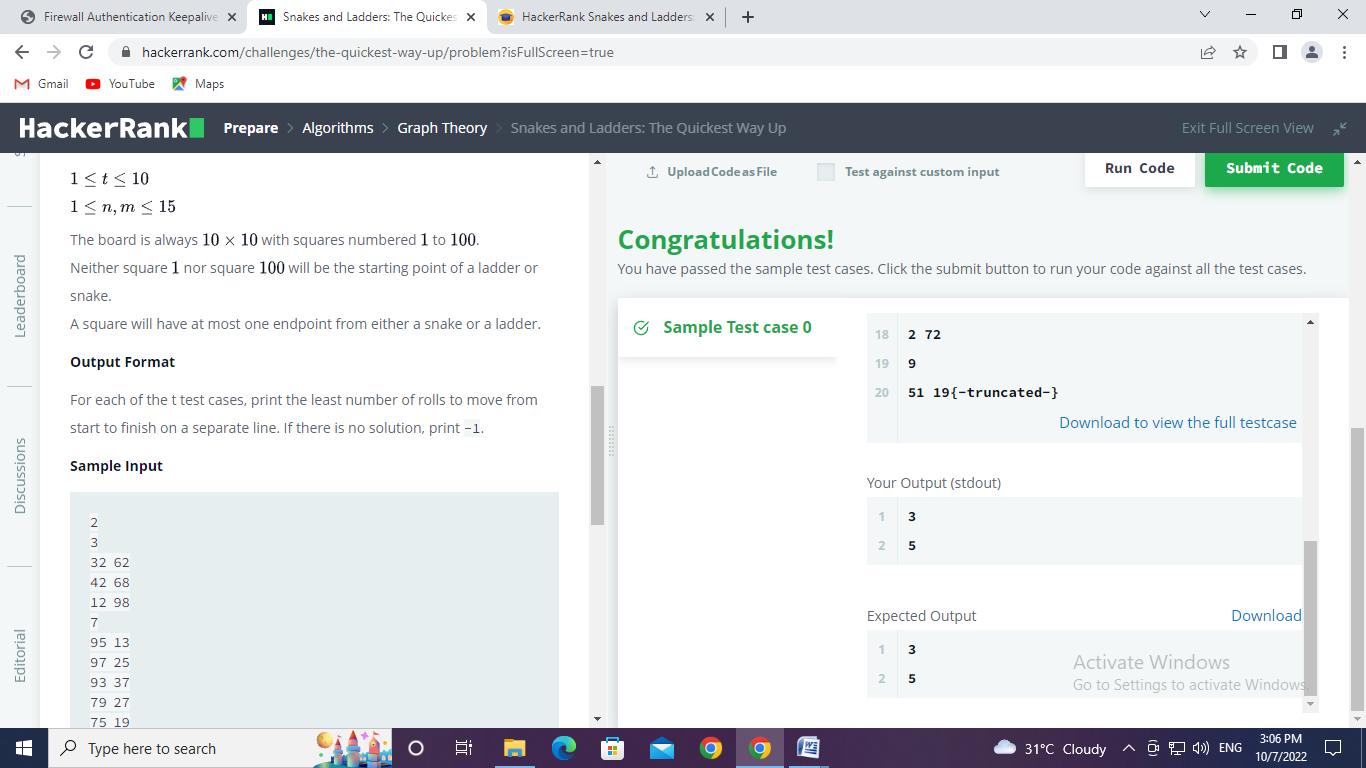
    }

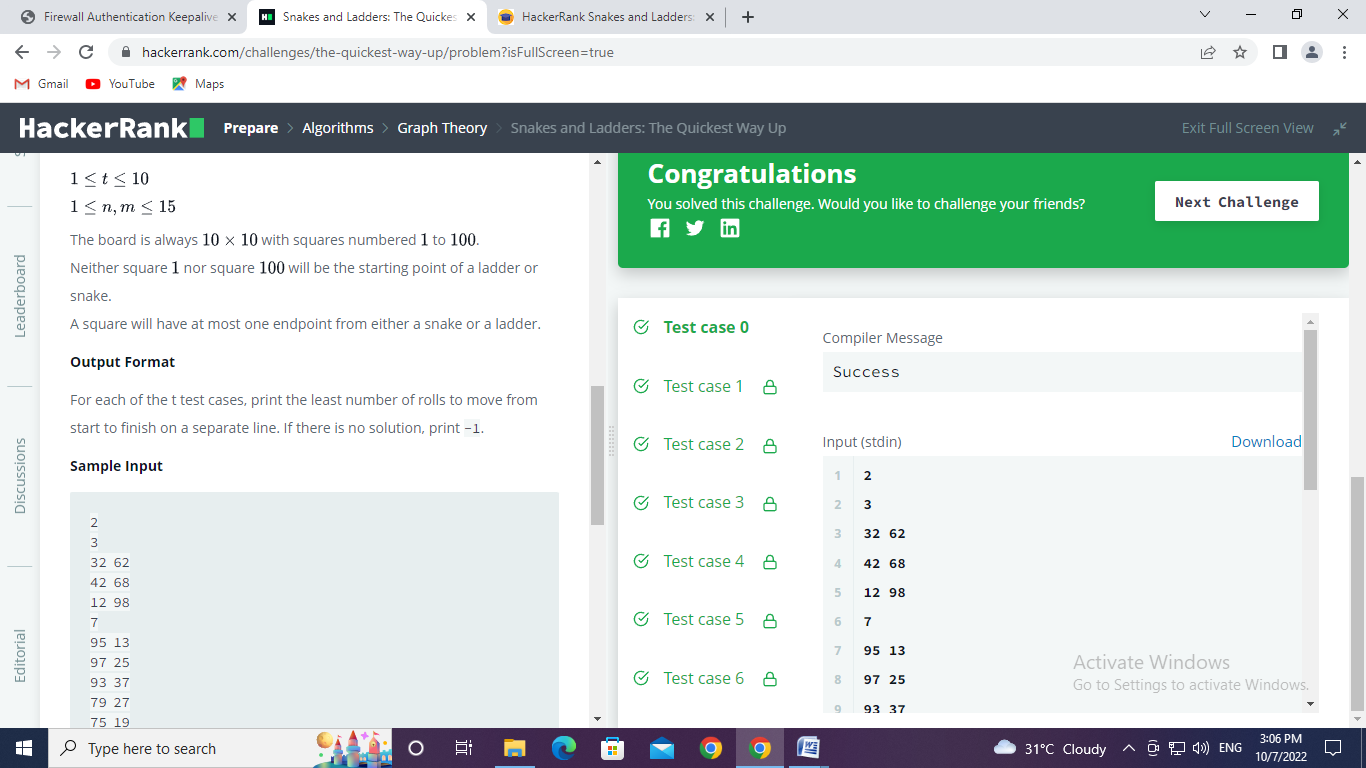
}

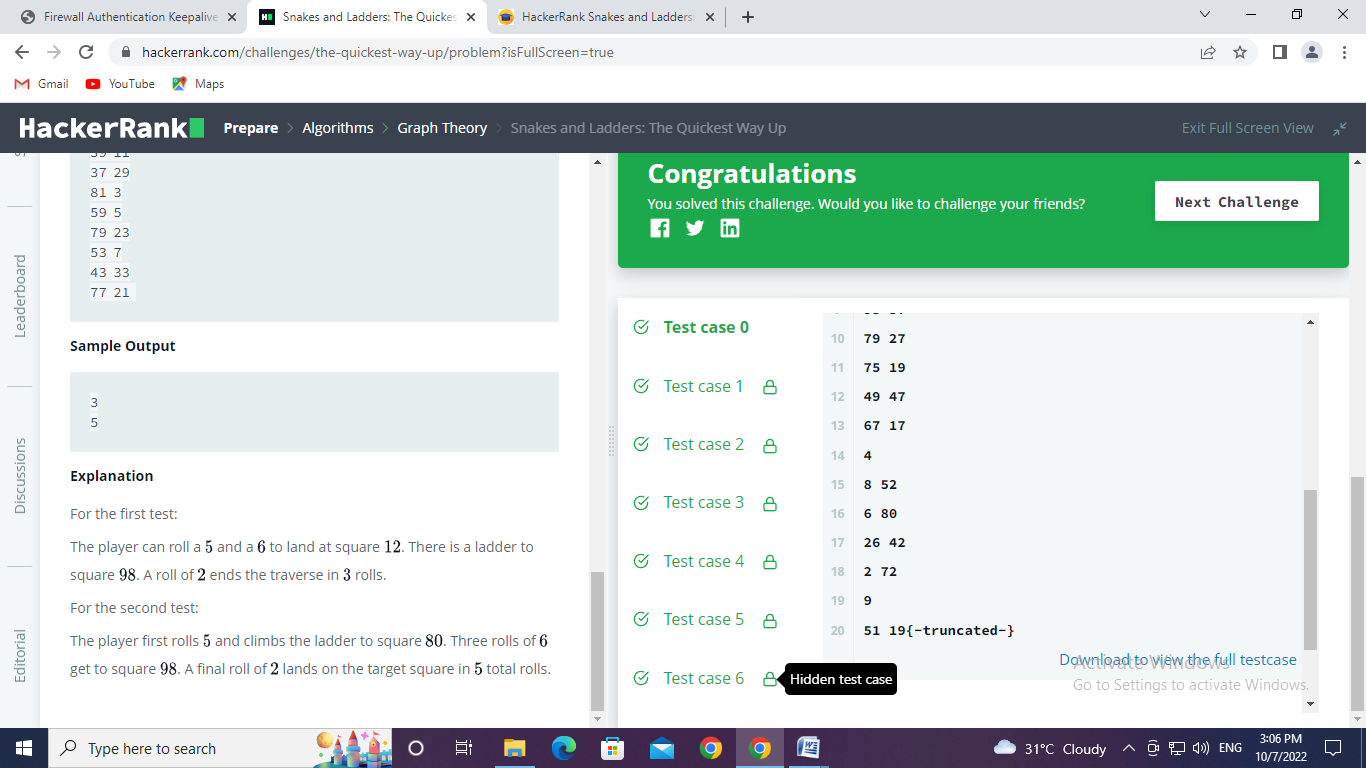
1. **Result/Output/Writing Summary:**

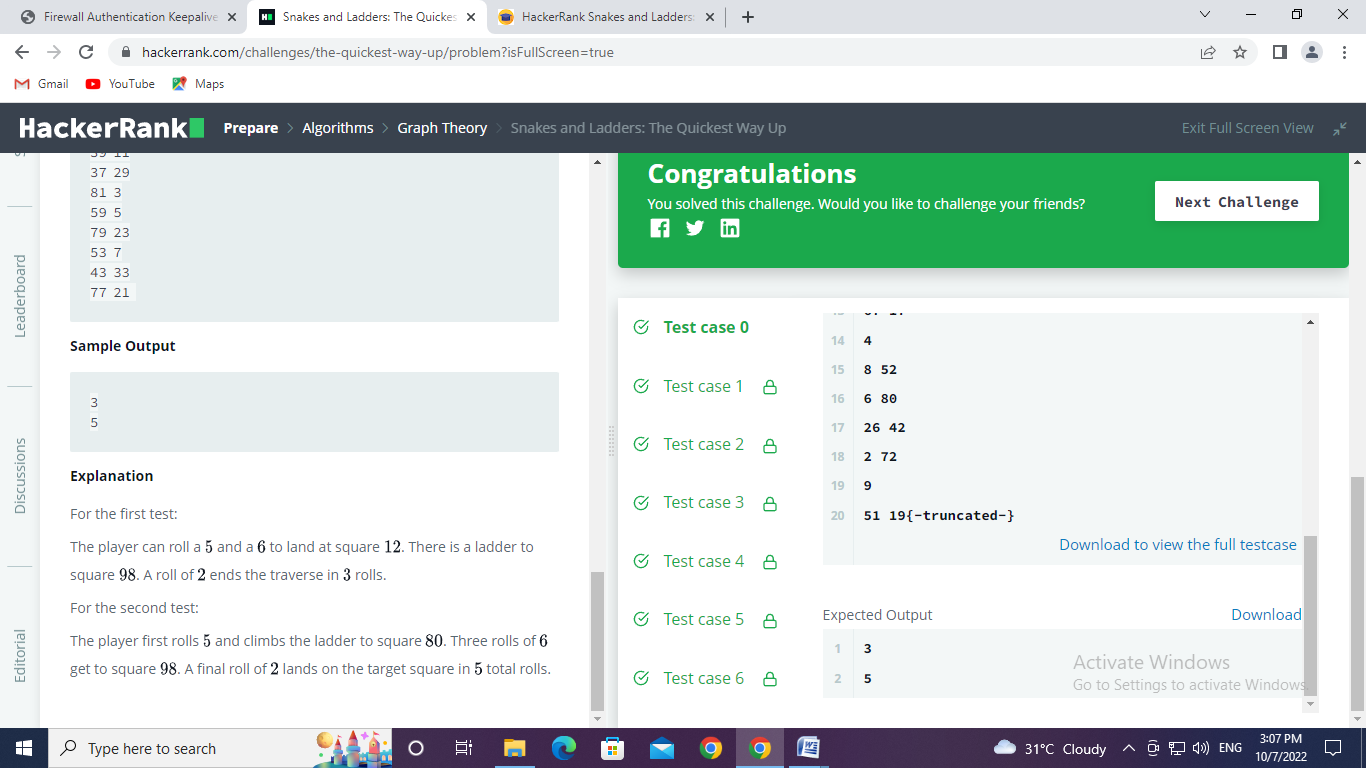
****

****

****

****

****

****

**Learning outcomes (What I have learnt):**

* Learned about the concept of graphs.
* Learned about implement the concept of Graphs.
* Learned about BFS.
* Learned about the snake and ladder concept using Graph

**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
| 1. |  |  |  |
| 2. |  |  |  |
| 3. |  |  |  |
|  |  |  |  |