**Experiment No. – 5**

**Student Name: SUMIT KUMAR UID: 20BCS8226**

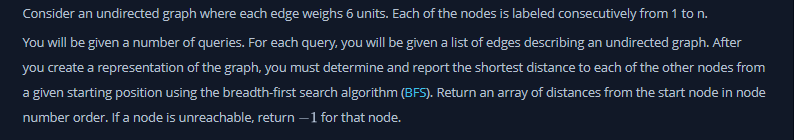
**Branch: BE-CSE Section/Group: MM 808 A**

**Semester: 5th Date of Submission: 12-11-2022**

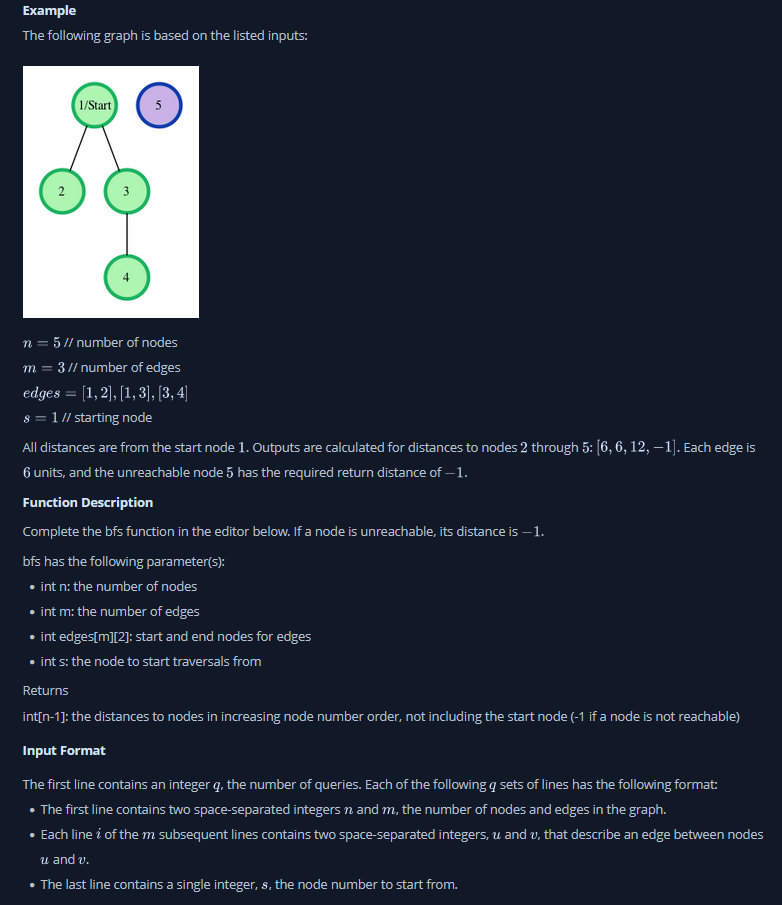
**Subject Name: Competitive coding - I Subject Code: 20CSP-314**

# **Breadth First Search: Shortest Reach:**

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

****

**2. Task to be done/ Which logistics used:**







1. **Hardware and Software Requirements (For programming-based labs):**

* Laptop or Desktop
* Hacker-Rank Account

1. **Steps for experiment/practical/Code:**

import java.util.\*;

class GraphNode{

int sumNodes;

ArrayList<LinkedList<Integer>> adjList;

public GraphNode(int numNodes){

this.sumNodes = numNodes;

adjList = new ArrayList<LinkedList<Integer>>();

for(int i = 0; i < numNodes; i++){

adjList.add(new LinkedList<Integer>());

}

}

public void addEdge(int a, int b){

adjList.get(a).add(b);

adjList.get(b).add(a);

}

}

public class Solution {

public static void getDistance(ArrayList<LinkedList<Integer>> adjList, int[] results, int s){

LinkedList<Integer> q = new LinkedList();

boolean[] isVisited = new boolean[adjList.size()];

q.add(s);

isVisited[s] = true;

int count = 0;

while(!q.isEmpty()){

int qSize = q.size();

for(int i = 0; i < qSize; i++){

int removed = q.poll();

results[removed] = count;

for(int x: adjList.get(removed)){

if(!isVisited[x]){

q.add(x);

isVisited[x] = true;

}

}

}

count += 6;

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int q = sc.nextInt();

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

int n = sc.nextInt();

int m = sc.nextInt();

GraphNode g = new GraphNode(n);

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

int a = sc.nextInt();

int b = sc.nextInt();

g.addEdge(a-1, b-1);

}

int s = sc.nextInt();

int[] results = new int[n];

Arrays.fill(results, -1);

getDistance(g.adjList, results, s-1);

for(int k = 0; k < n; k++){

if(k != s-1) System.out.print(results[k]+ " ");

}

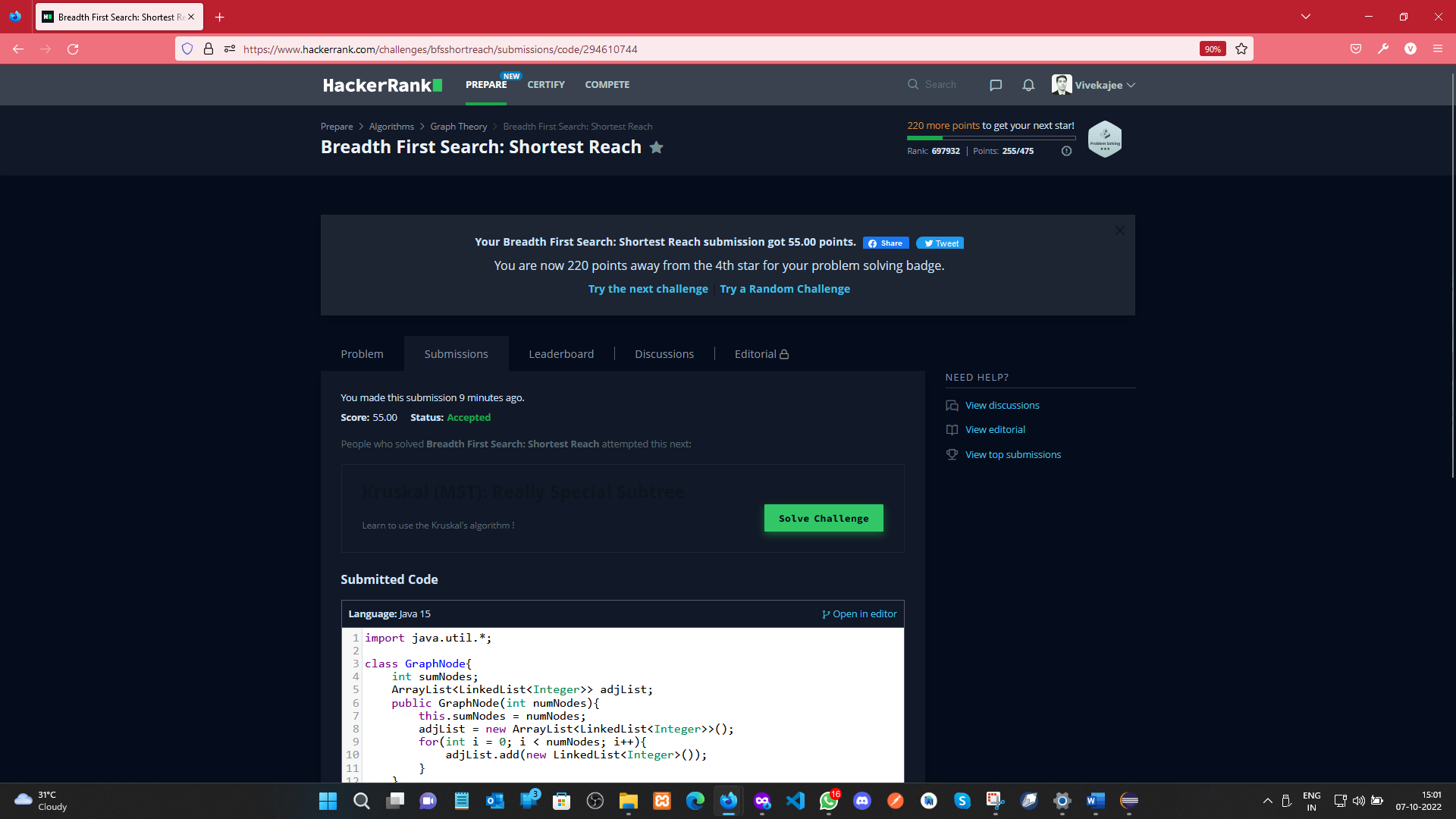
System.out.println();

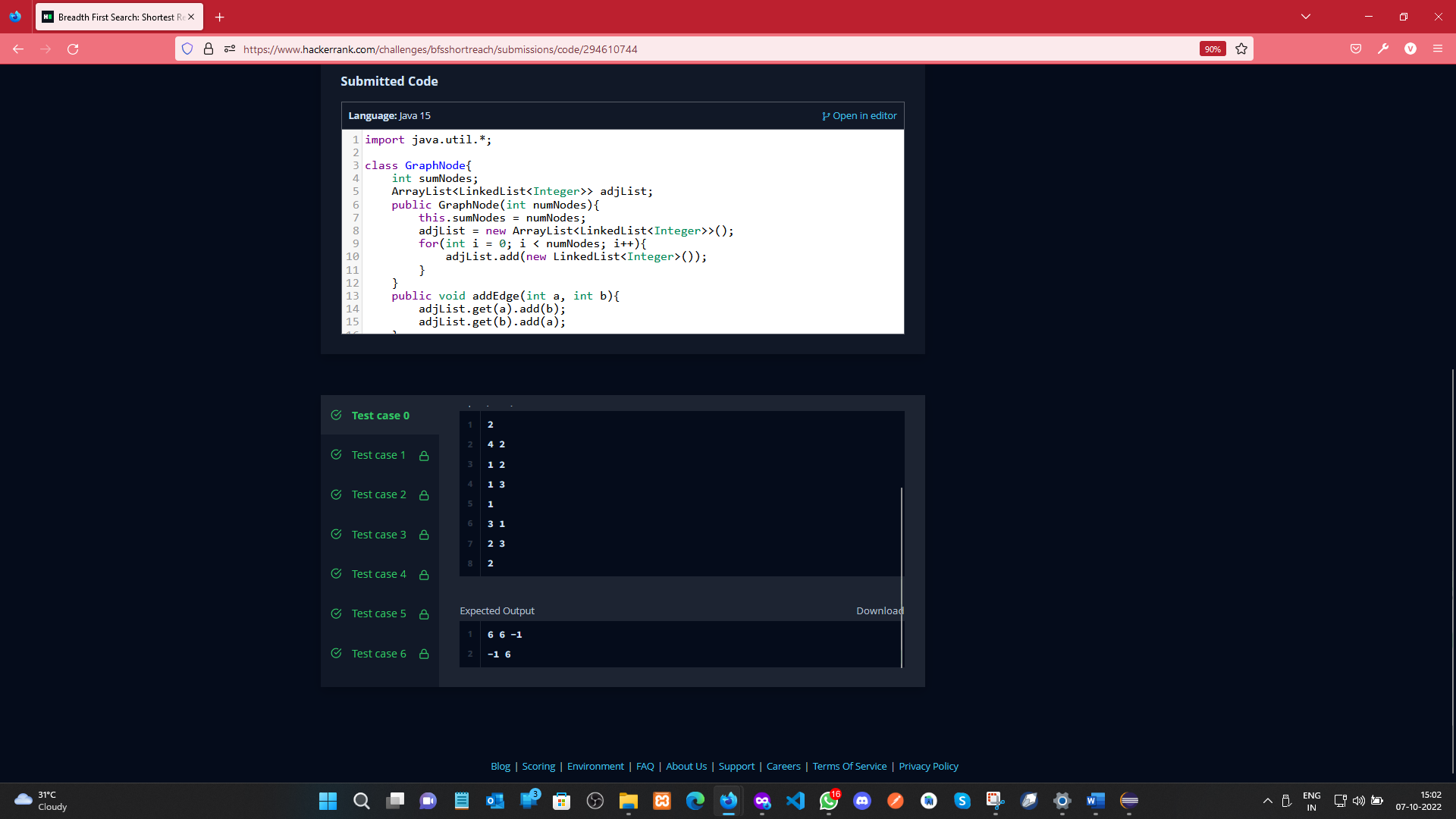
}

}

}

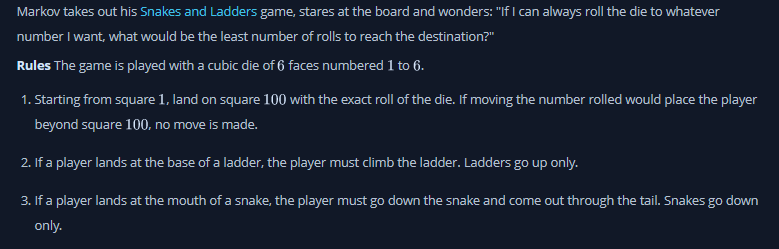
**5. Result/Output/Writing Summary:**



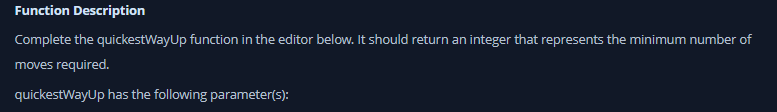
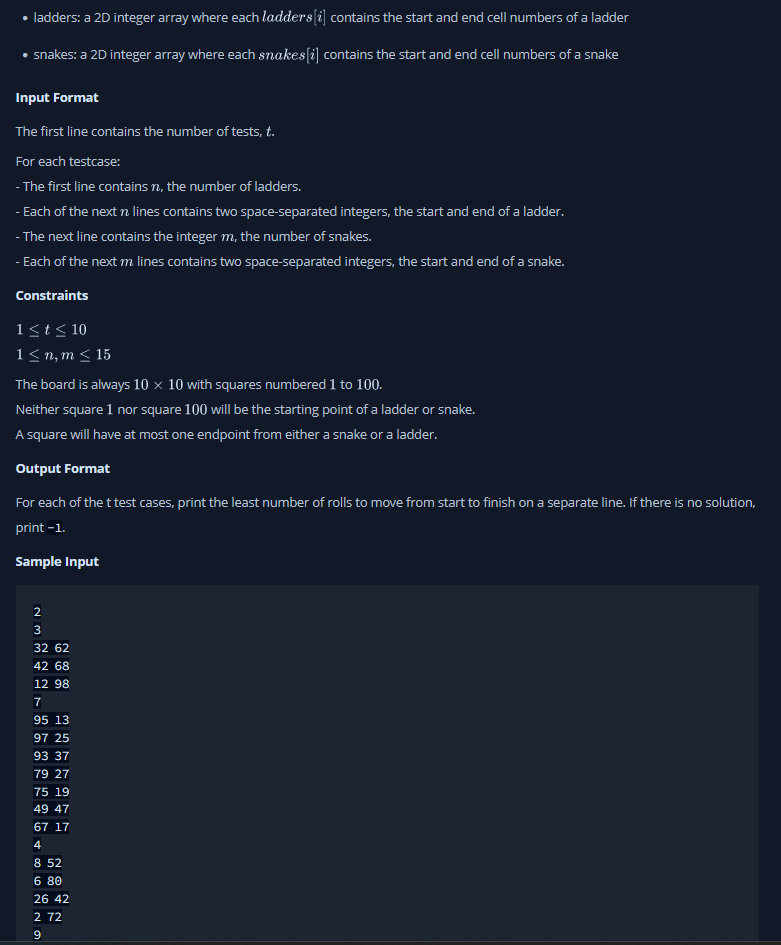


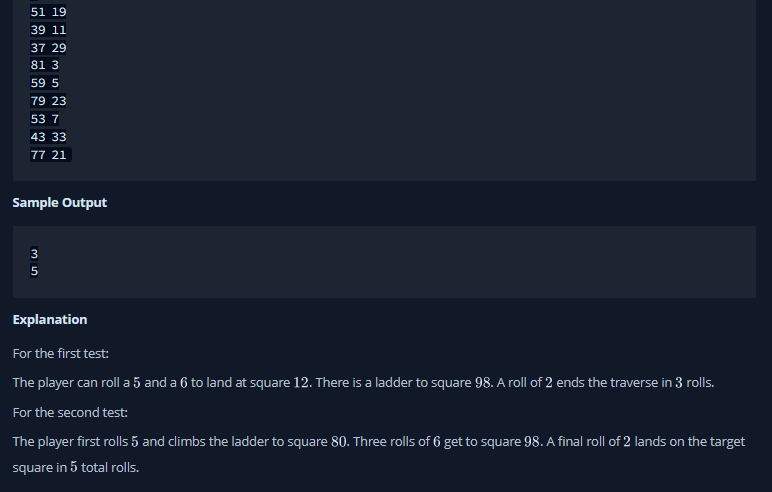
**Snakes and Ladders: The Quickest Way Up:**

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



1. **Task to be done/ Which logistics used:**



1. **Hardware and Software Requirements (For programming-based labs):**

* Laptop or Desktop
* Hacker-Rank Account

1. **Steps for experiment/practical/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);

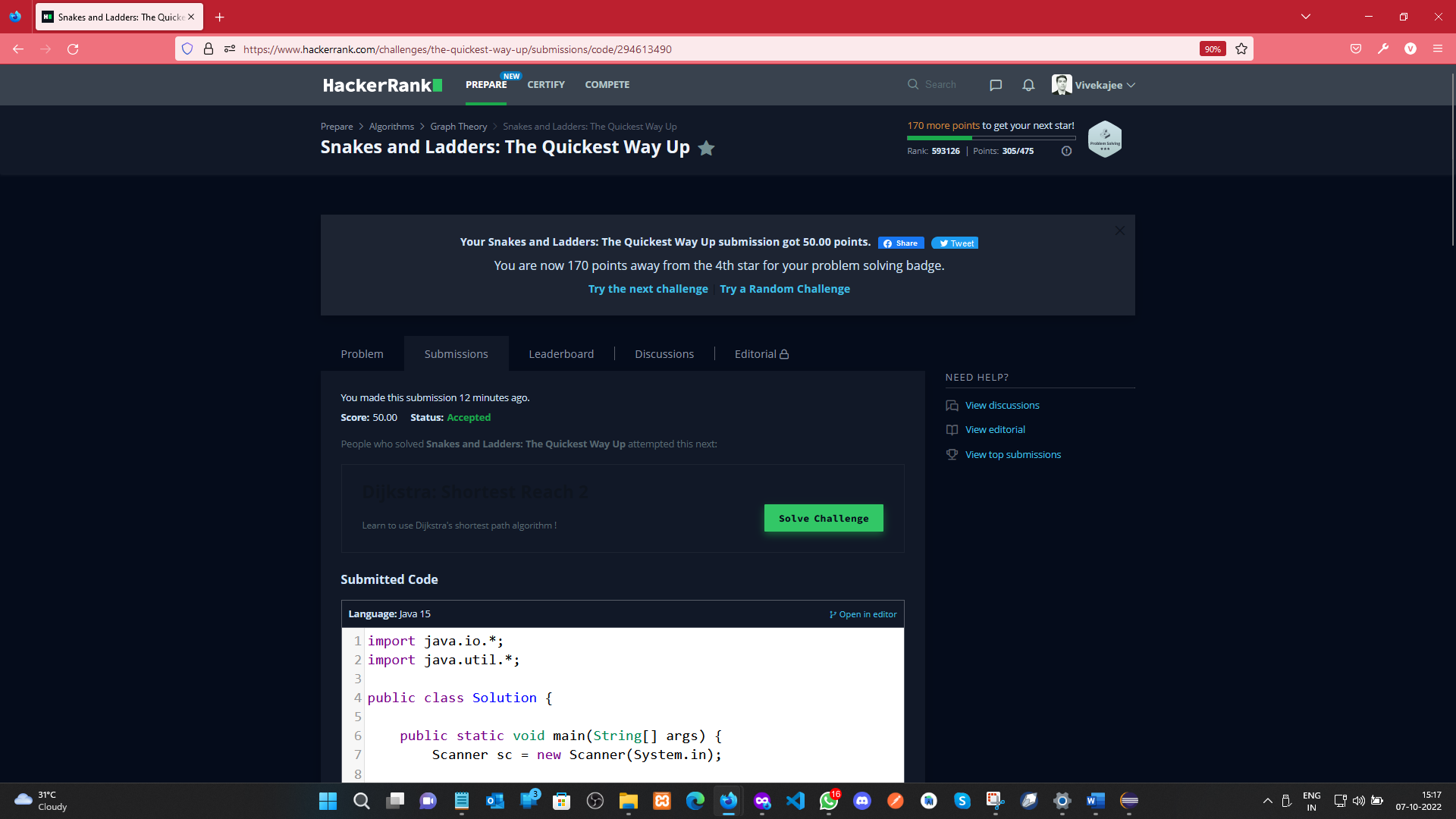
        }

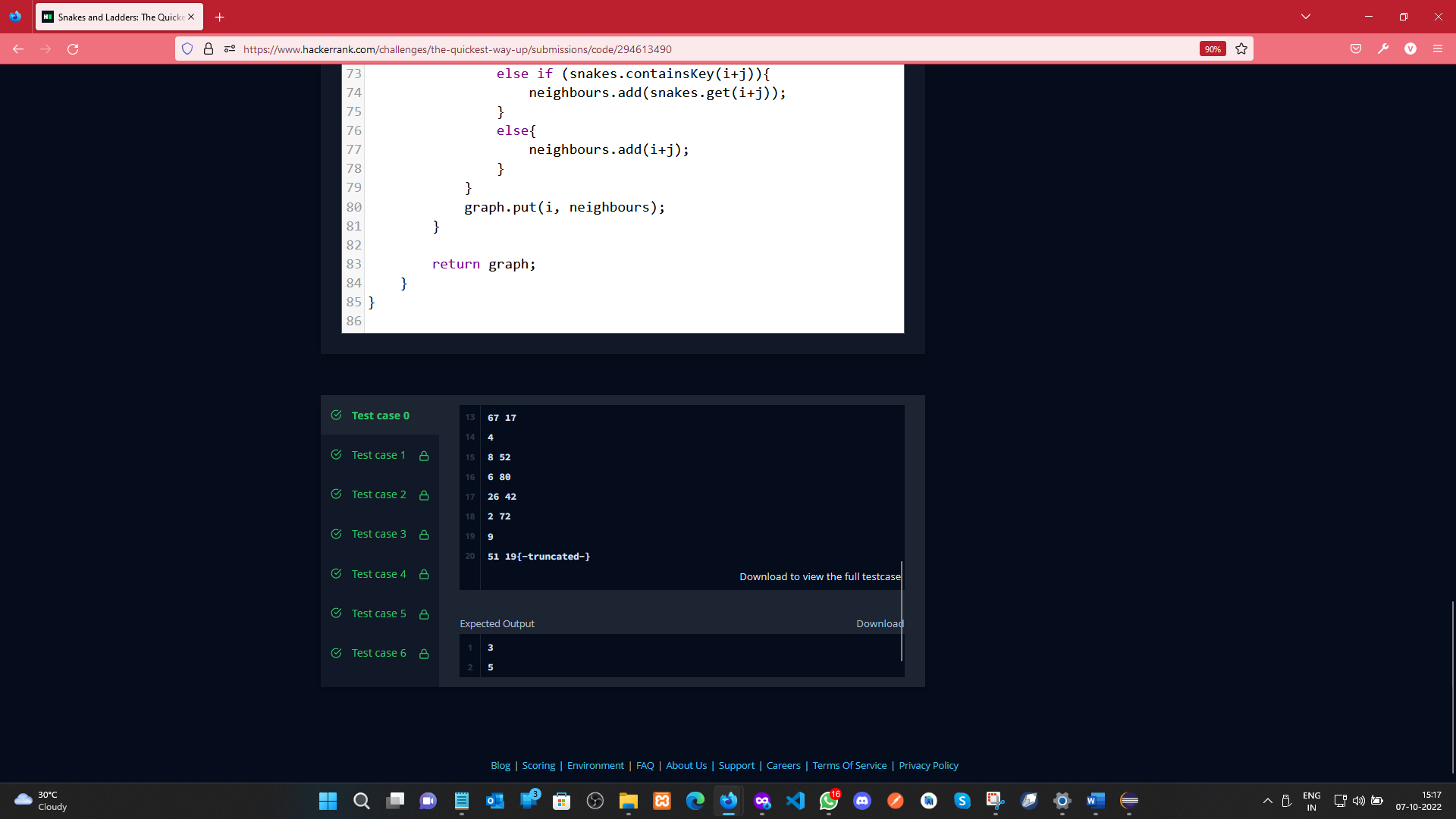
        return graph;

    }

}

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





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

a. Learnt about Graph concept.

1. b. Learnt about BFS.
2. c. Learn 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. |  |  |  |
|  |  |  |  |