CHANDIGARH UNIVERSITY

UNIVERSITY INSTITUTE OF NGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

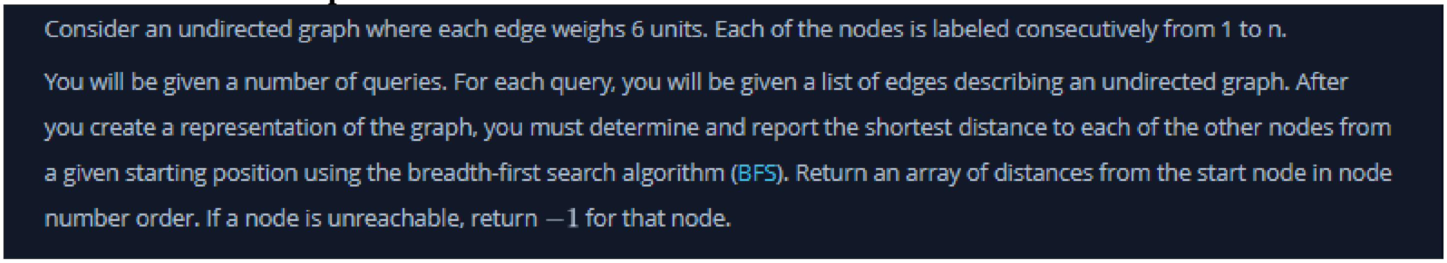
Submitted By: Vivek Kumar(21BCS8129)	**Submitted To:** Mamta Punia(E12337)
Subject Name	Competitive Coding - I
Subject Code	20CSP-314
Branch	Computer Science and Engineering
Semester	5th

Experiment No. - 5

Student Name: Vivek Kumar **UID: 21BCS8129**
Branch: BE-CSE(LEET) **Section/Group: WM-20BCS-616/A**
Semester: 5th **Date of Performance: 07/10/2022**
Subject Name: Competitive coding - I **Subject Code: 20CSP-314**

Breadth First Search: Shortest Reach:

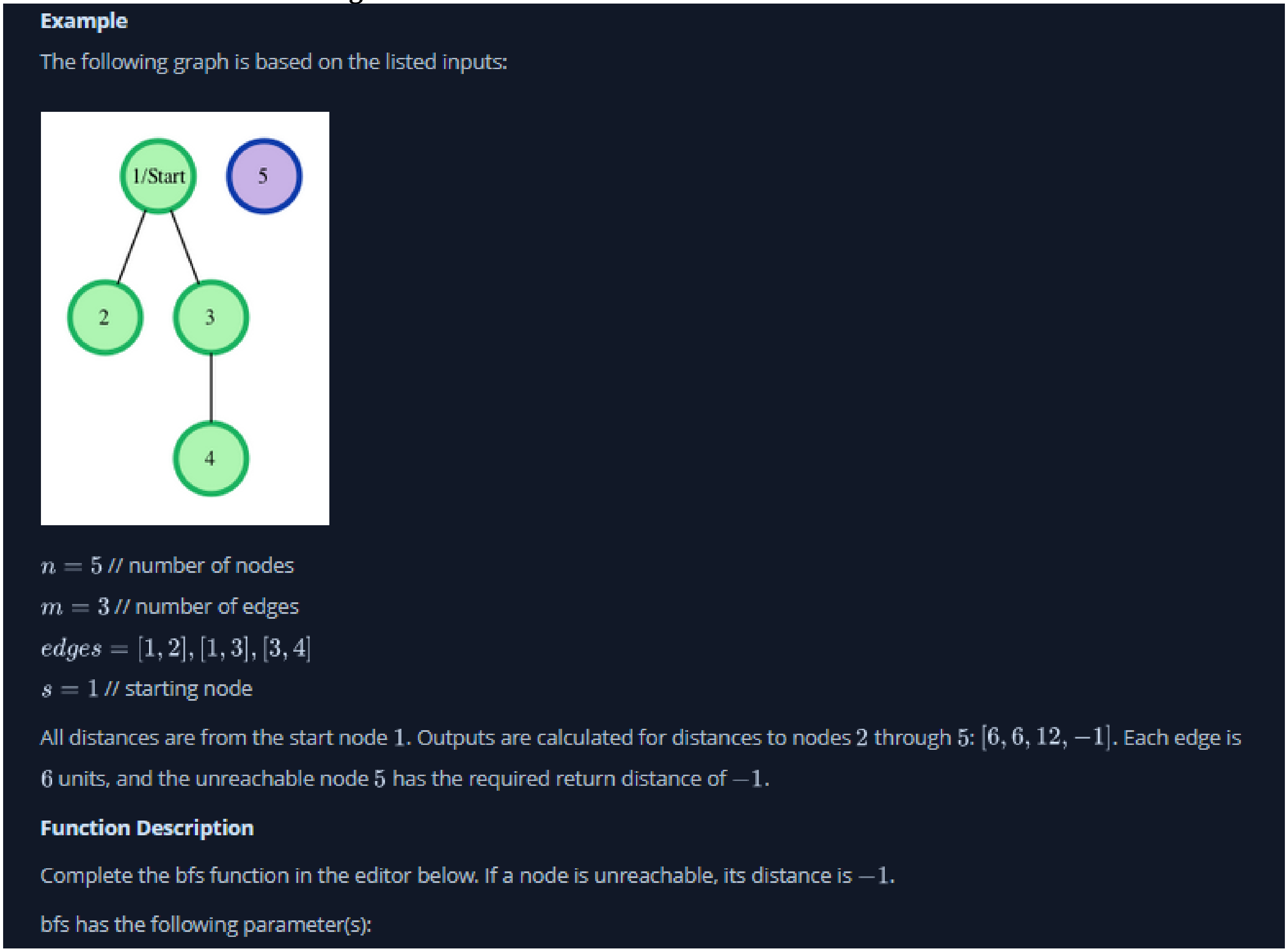
1. Aim/Overview of the practical:



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). Return an array of distances from the start node in node number order. If a node is unreachable, return -1 for that node.

2. Task to be done/ Which logistics used:



Example

The following graph is based on the listed inputs:

$n = 5$ // number of nodes

$m = 3$ // number of edges

$edges = [1, 2], [1, 3], [3, 4]$

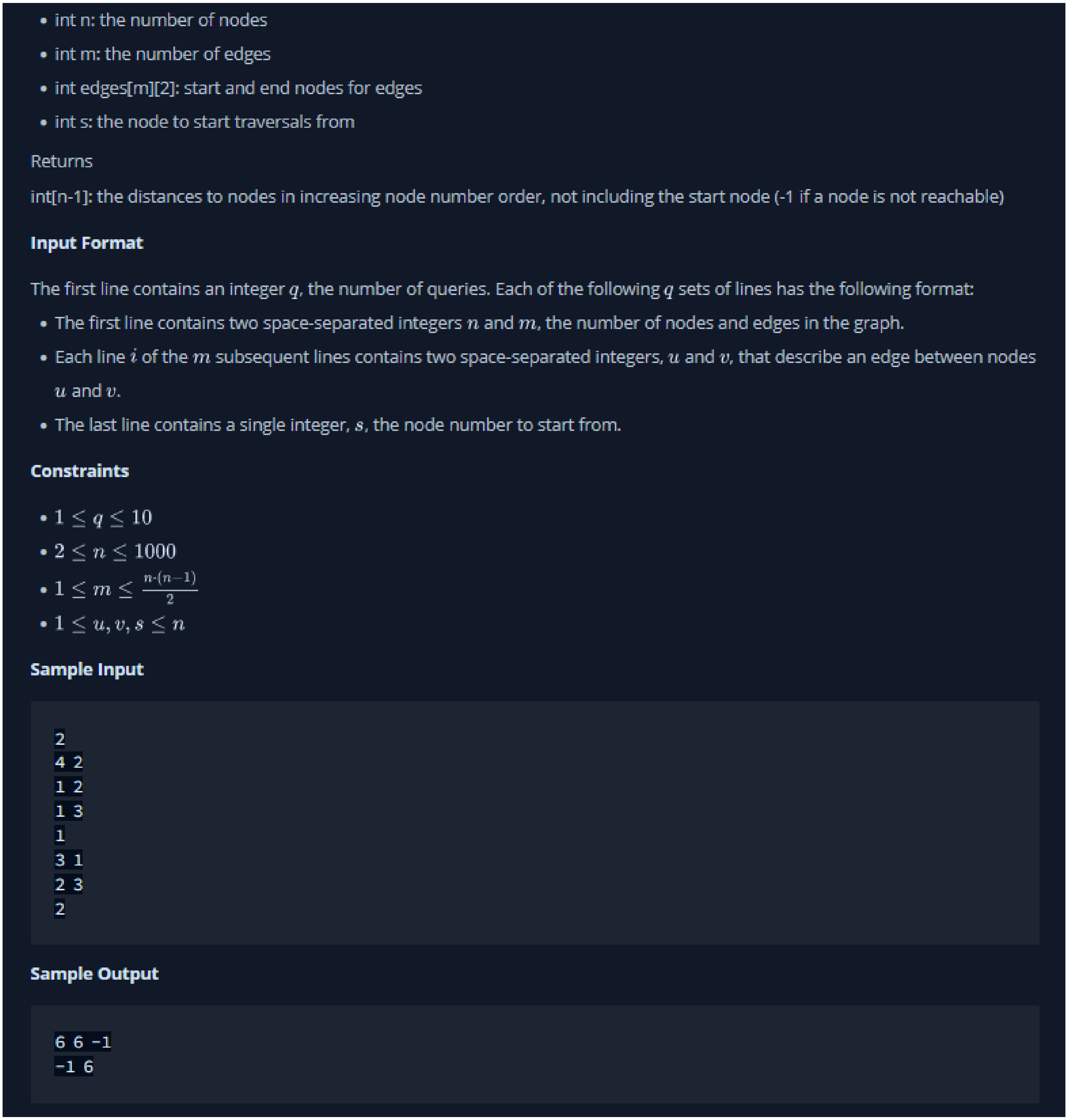
$s = 1$ // starting node

All distances are from the start node 1. Outputs are calculated for distances to nodes 2 through 5: $[6, 6, 12, -1]$. Each edge is 6 units, and the unreachable node 5 has the required return distance of -1.

Function Description

Complete the bfs function in the editor below. If a node is unreachable, its distance is -1.

bfs has the following parameter(s):



- int n: the number of nodes
- int m: the number of edges
- int edges[m][2]: start and end nodes for edges
- int s: the node to start traversals from

Returns

int[n-1]: the distances to nodes in increasing node number order, not including the start node (-1 if a node is not reachable)

Input Format

The first line contains an integer q, the number of queries. Each of the following q sets of lines has the following format:

- The first line contains two space-separated integers n and m, the number of nodes and edges in the graph.
- Each line i of the m subsequent lines contains two space-separated integers, u and v, that describe an edge between nodes u and v.
- The last line contains a single integer, s, the node number to start from.

Constraints

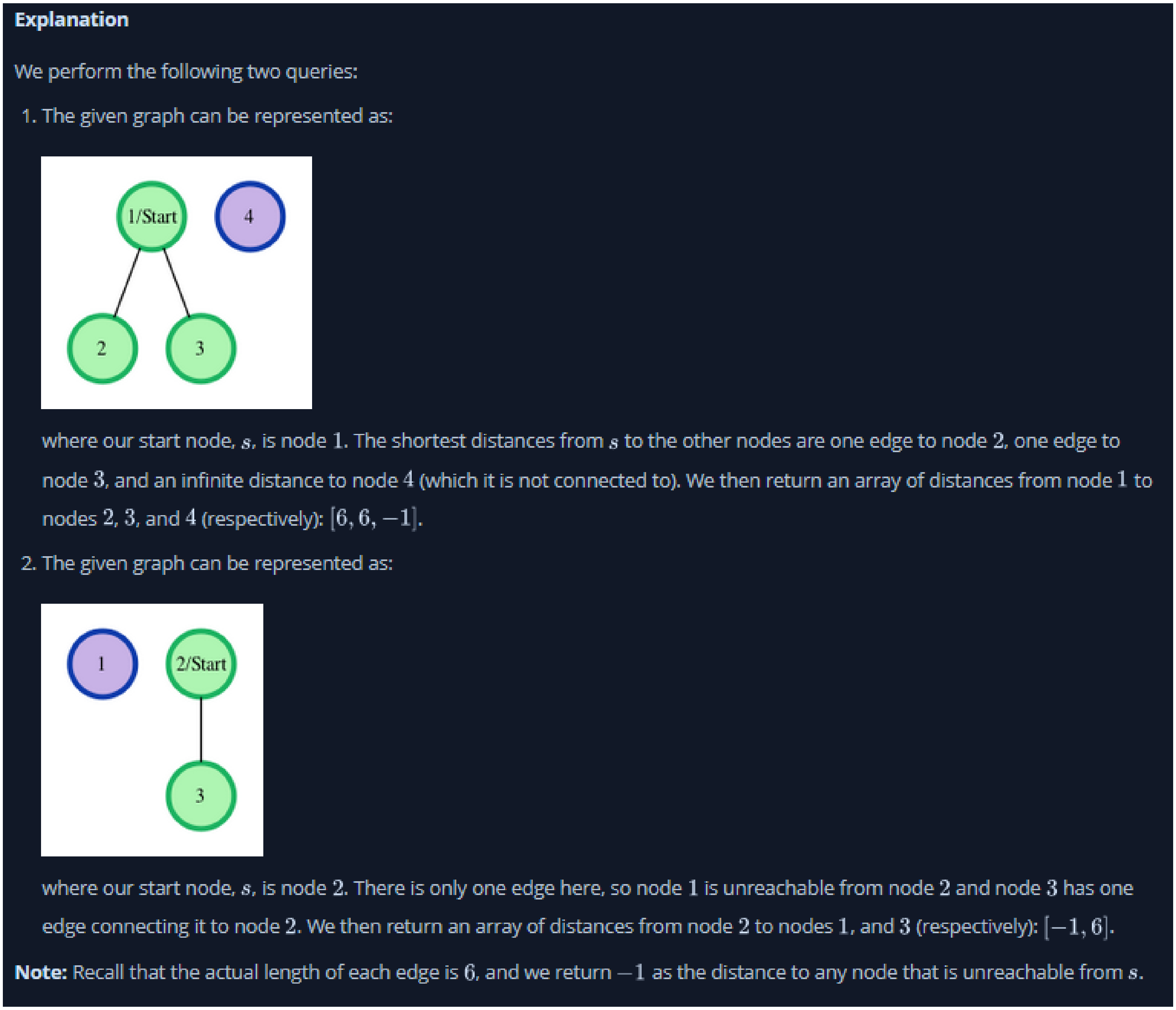
- $1 \leq q \leq 10$
- $2 \leq n \leq 1000$
- $1 \leq m \leq \frac{n\cdot(n-1)}{2}$
- $1 \leq u, v, s \leq n$

Sample Input

```
2
4 2
1 2
1 3
1
3 1
2 3
2
```

Sample Output

```
6 6 -1
-1 6
```



Explanation

We perform the following two queries:

1. The given graph can be represented as:

   where our start node, s, is node 1. The shortest distances from s to the other nodes are one edge to node 2, one edge to node 3, and an infinite distance to node 4 (which it is not connected to). We then return an array of distances from node 1 to nodes 2, 3, and 4 (respectively): $[6, 6, -1]$.

2. The given graph can be represented as:

   where our start node, s, is node 2. There is only one edge here, so node 1 is unreachable from node 2 and node 3 has one edge connecting it to node 2. We then return an array of distances from node 2 to nodes 1, and 3 (respectively): $[-1, 6]$.

Note: Recall that the actual length of each edge is 6, and we return -1 as the distance to any node that is unreachable from s.

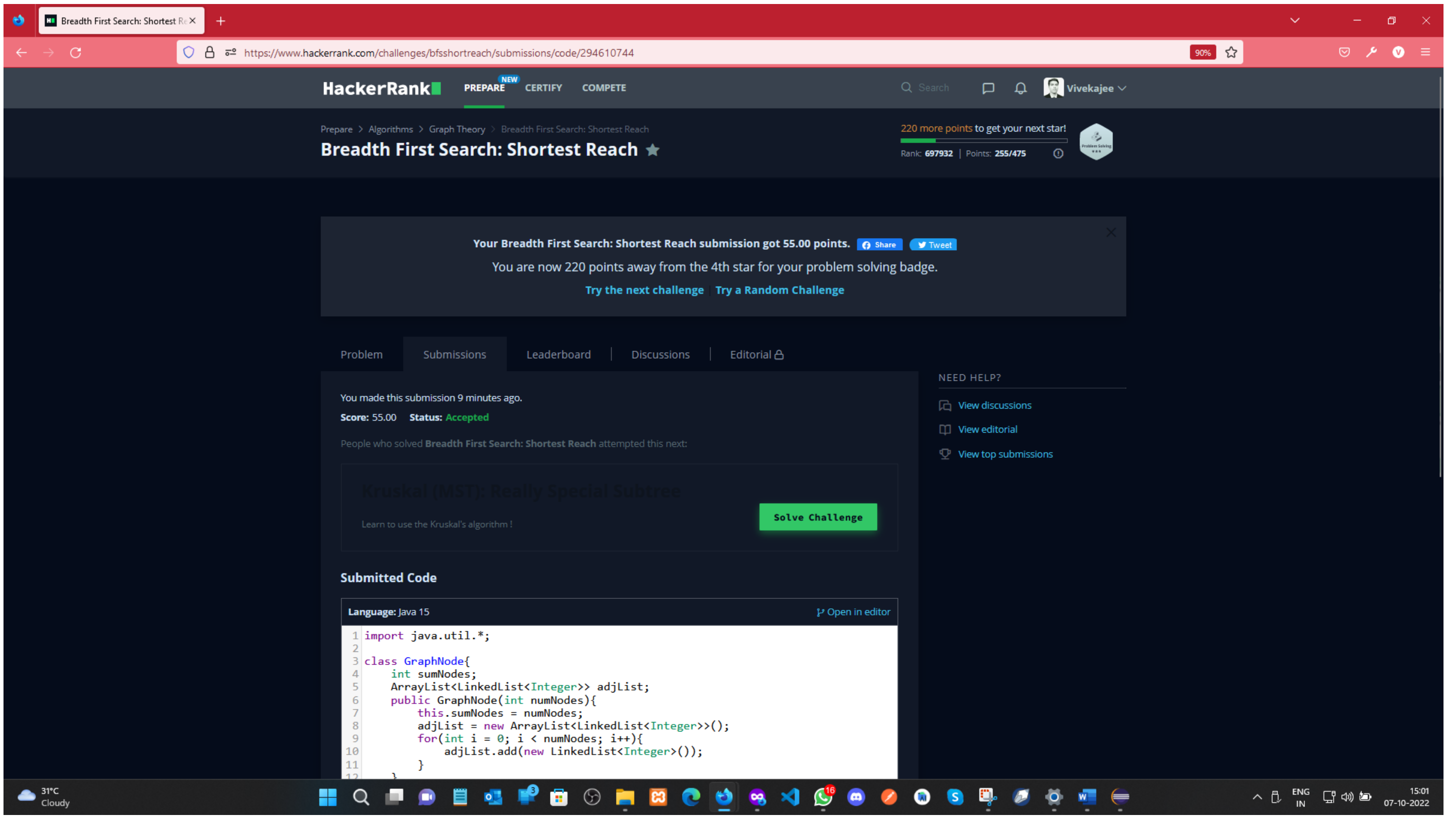
3. Hardware and Software Requirements (For programming-based labs)**:**

- Laptop or Desktop
- Hacker-Rank Account

4. 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:



Breadth First Search: Shortest Reach

220 more points to get your next star!

Rank: 697932 | Points: 255/475

Your Breadth First Search: Shortest Reach submission got 55.00 points.

You are now 220 points away from the 4th star for your problem solving badge.

Try the next challenge | Try a Random Challenge

You made this submission 9 minutes ago.

Score: 55.00 **Status:** Accepted

People who solved **Breadth First Search: Shortest Reach** attempted this next:

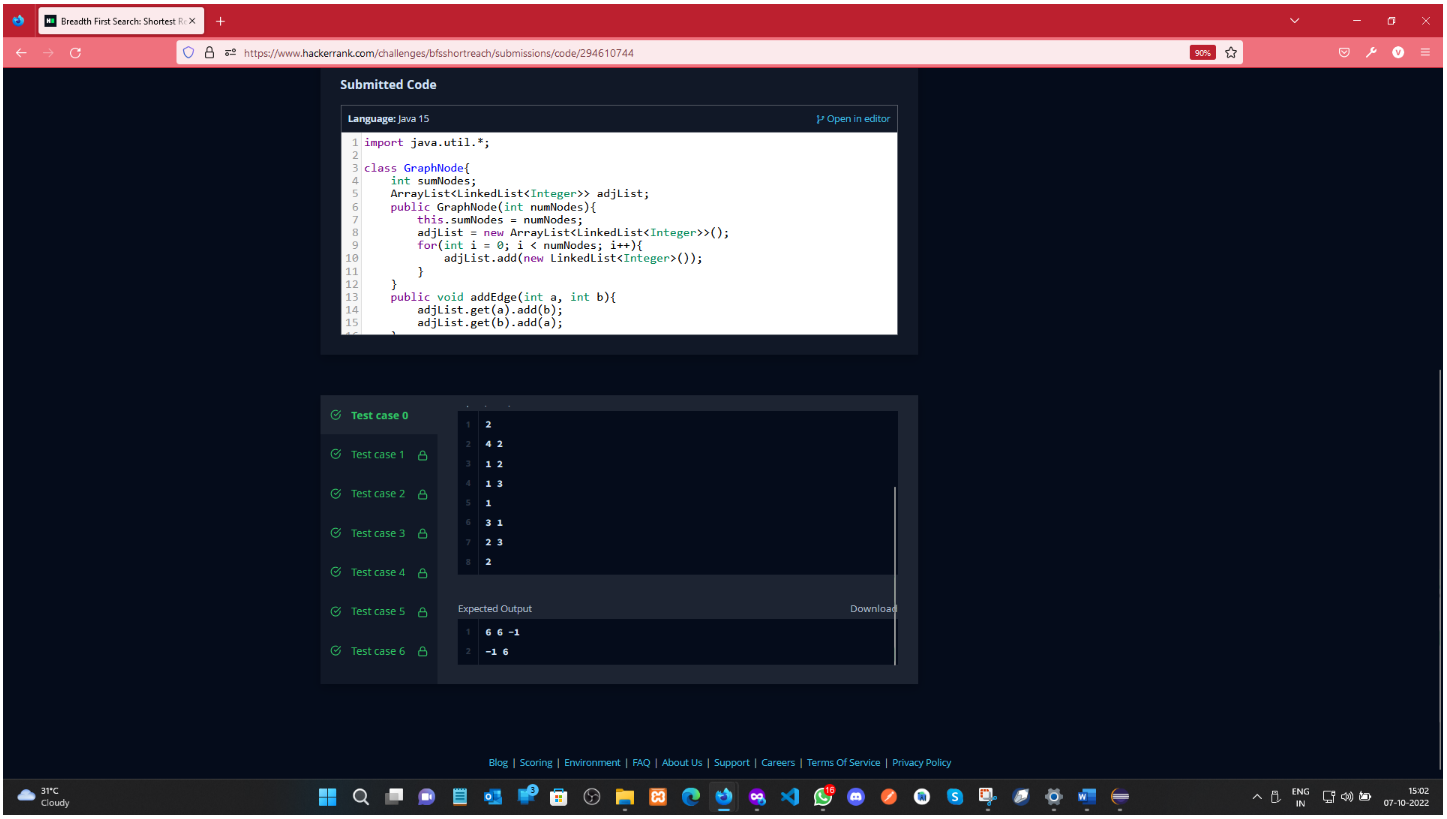
Kruskal (MST): Really Special Subtree

Learn to use the Kruskal's algorithm !

Submitted Code

Language: Java 15

```
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>());
        }
    }
```



Breadth First Search: Shortest R
https://www.hackerrank.com/challenges/bfsshortreach/submissions/code/294610744
90%

Submitted Code

Language: Java 15

```java
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);
```

Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6

```
2
4 2
1 2
1 3
1
3 1
2 3
2
```

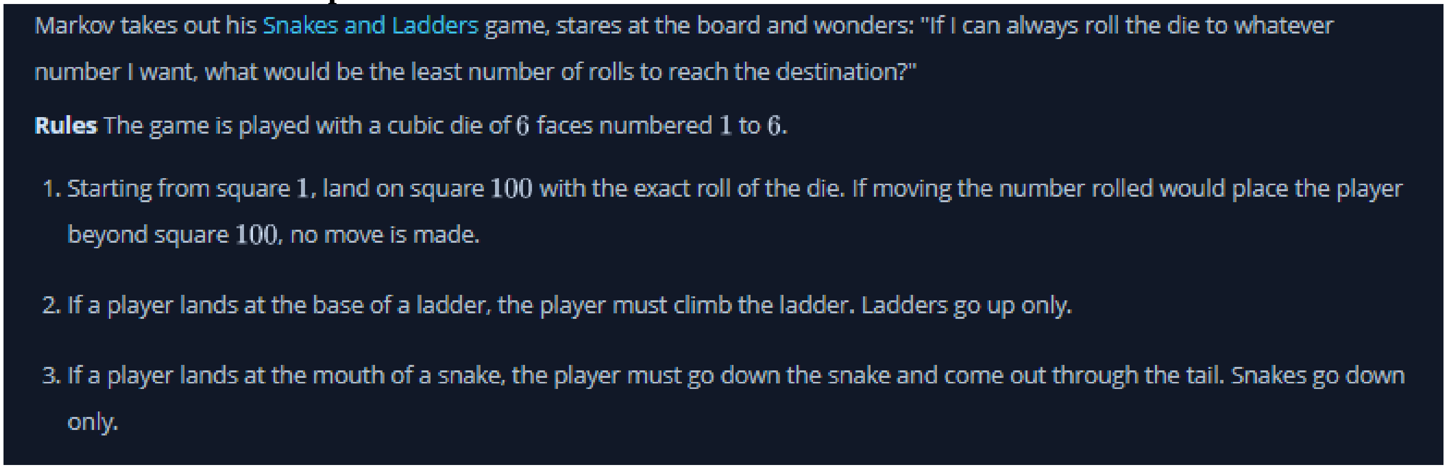
Expected Output

```
6 6 -1
-1 6
```

Blog | Scoring | Environment | FAQ | About Us | Support | Careers | Terms Of Service | Privacy Policy
31°C
Cloudy
ENG
IN
15:02
07-10-2022

Snakes and Ladders: The Quickest Way Up:

1. Aim/Overview of the practical:

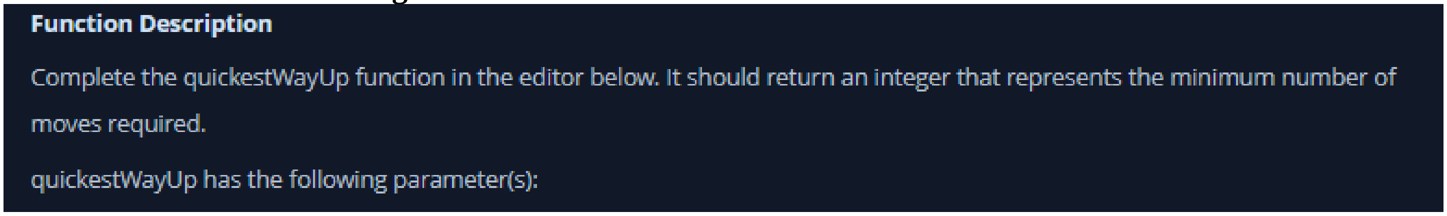


Markov takes out his Snakes and Ladders game, stares at the board and wonders: "If I can always roll the die to whatever number I want, what would be the least number of rolls to reach the destination?"

Rules The game is played with a cubic die of 6 faces numbered 1 to 6.

1. Starting from square 1, land on square 100 with the exact roll of the die. If moving the number rolled would place the player beyond square 100, no move is made.
2. If a player lands at the base of a ladder, the player must climb the ladder. Ladders go up only.
3. If a player lands at the mouth of a snake, the player must go down the snake and come out through the tail. Snakes go down only.

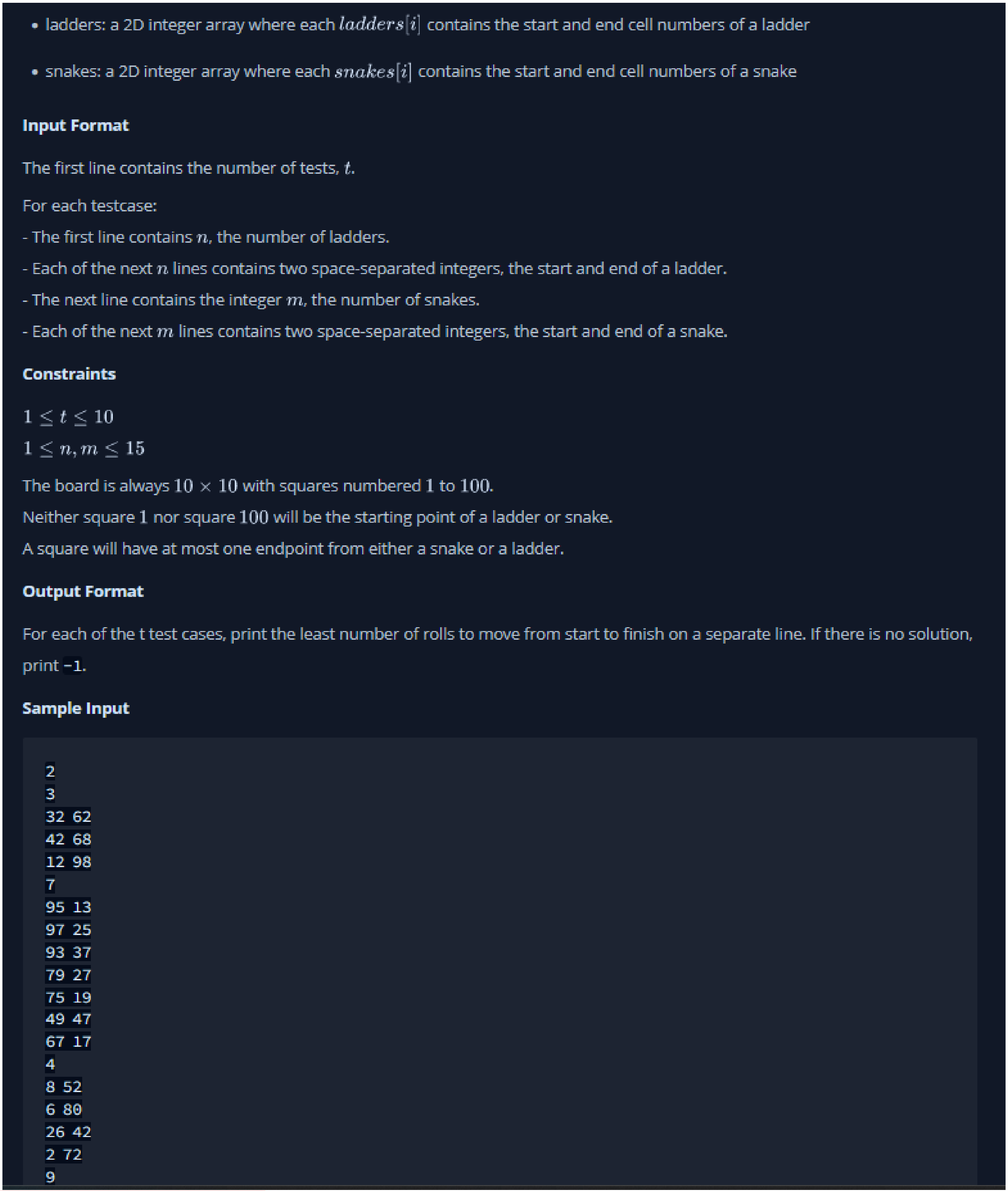
2. Task to be done/ Which logistics used:



Function Description

Complete the quickestWayUp function in the editor below. It should return an integer that represents the minimum number of moves required.

quickestWayUp has the following parameter(s):



- ladders: a 2D integer array where each $ladders[i]$ contains the start and end cell numbers of a ladder
- snakes: a 2D integer array where each $snakes[i]$ contains the start and end cell numbers of a snake

Input Format

The first line contains the number of tests, t.

For each testcase:

- The first line contains n, the number of ladders.

- Each of the next n lines contains two space-separated integers, the start and end of a ladder.

- The next line contains the integer m, the number of snakes.

- Each of the next m lines contains two space-separated integers, the start and end of a snake.

Constraints

$1 \leq t \leq 10$
$1 \leq n, m \leq 15$

The board is always 10×10 with squares numbered 1 to 100.

Neither square 1 nor square 100 will be the starting point of a ladder or snake.

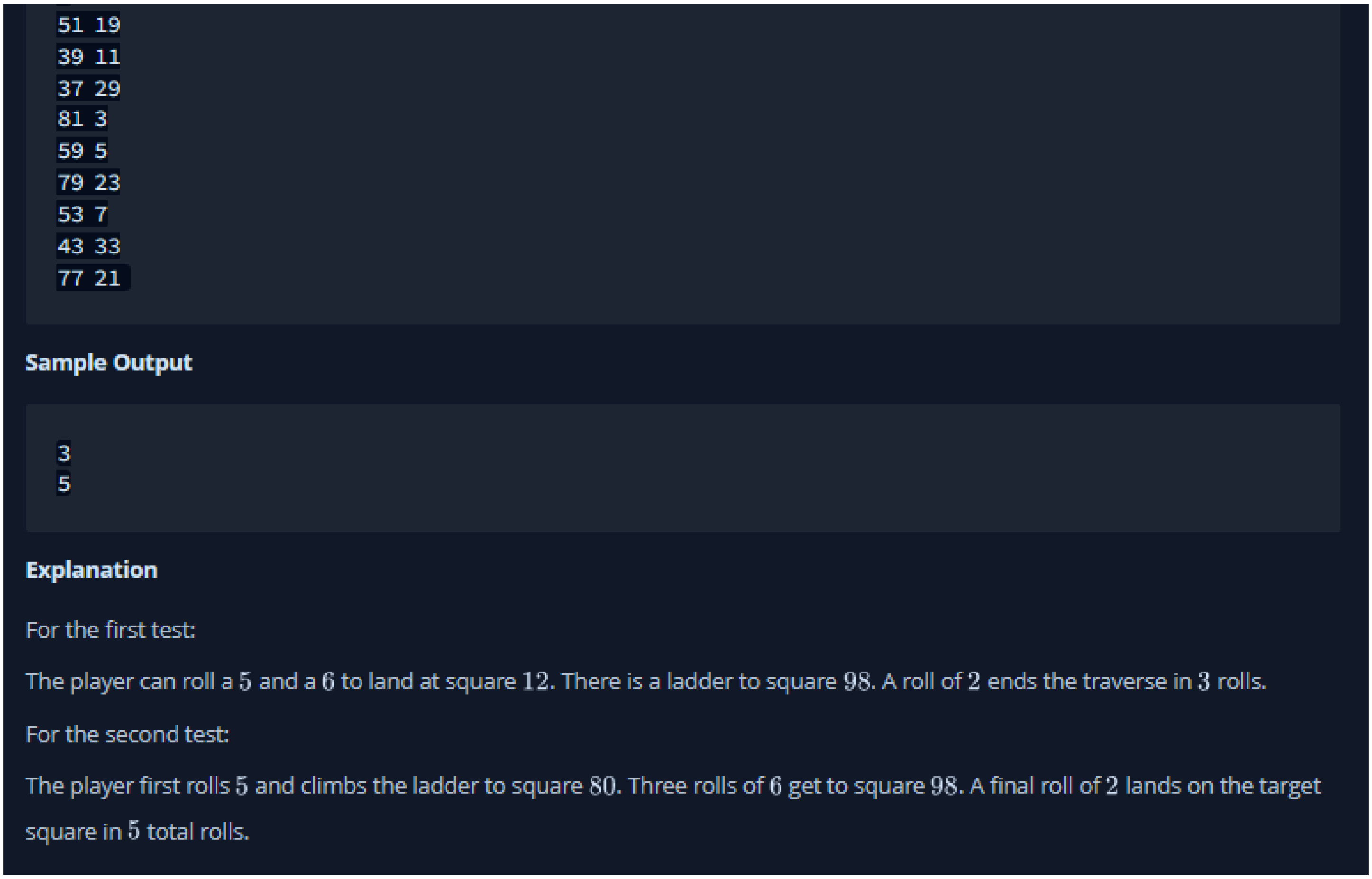
A square will have at most one endpoint from either a snake or a ladder.

Output Format

For each of the t test cases, print the least number of rolls to move from start to finish on a separate line. If there is no solution, print `-1`.

Sample Input

```
2
3
32 62
42 68
12 98
7
95 13
97 25
93 37
79 27
75 19
49 47
67 17
4
8 52
6 80
26 42
2 72
9
```



```
51 19
39 11
37 29
81 3
59 5
79 23
53 7
43 33
77 21
```

Sample Output

```
3
5
```

Explanation

For the first test:

The player can roll a 5 and a 6 to land at square 12. There is a ladder to square 98. A roll of 2 ends the traverse in 3 rolls.

For the second test:

The player first rolls 5 and climbs the ladder to square 80. Three rolls of 6 get to square 98. A final roll of 2 lands on the target square in 5 total rolls.

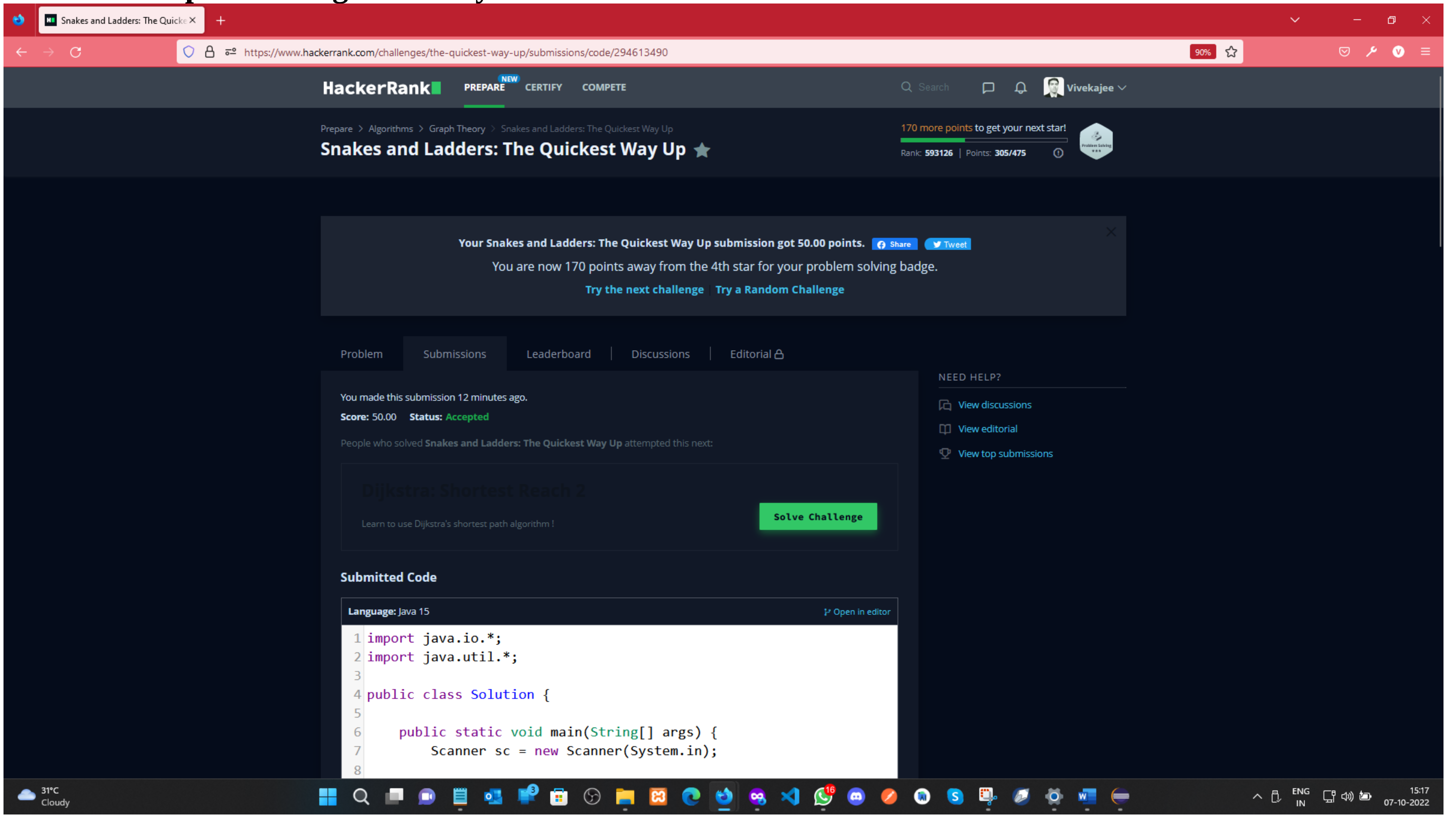
3. Hardware and Software Requirements (For programming-based labs):

- Laptop or Desktop
- Hacker-Rank Account

4. 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;
    }
}
```

6. Result/Output/Writing Summary:



Snakes and Ladders: The Quickest Way Up

170 more points to get your next star!

Rank: 593126 | Points: 305/475

Your Snakes and Ladders: The Quickest Way Up submission got 50.00 points.

You are now 170 points away from the 4th star for your problem solving badge.

Try the next challenge | Try a Random Challenge

You made this submission 12 minutes ago.

Score: 50.00 **Status:** Accepted

People who solved **Snakes and Ladders: The Quickest Way Up** attempted this next:

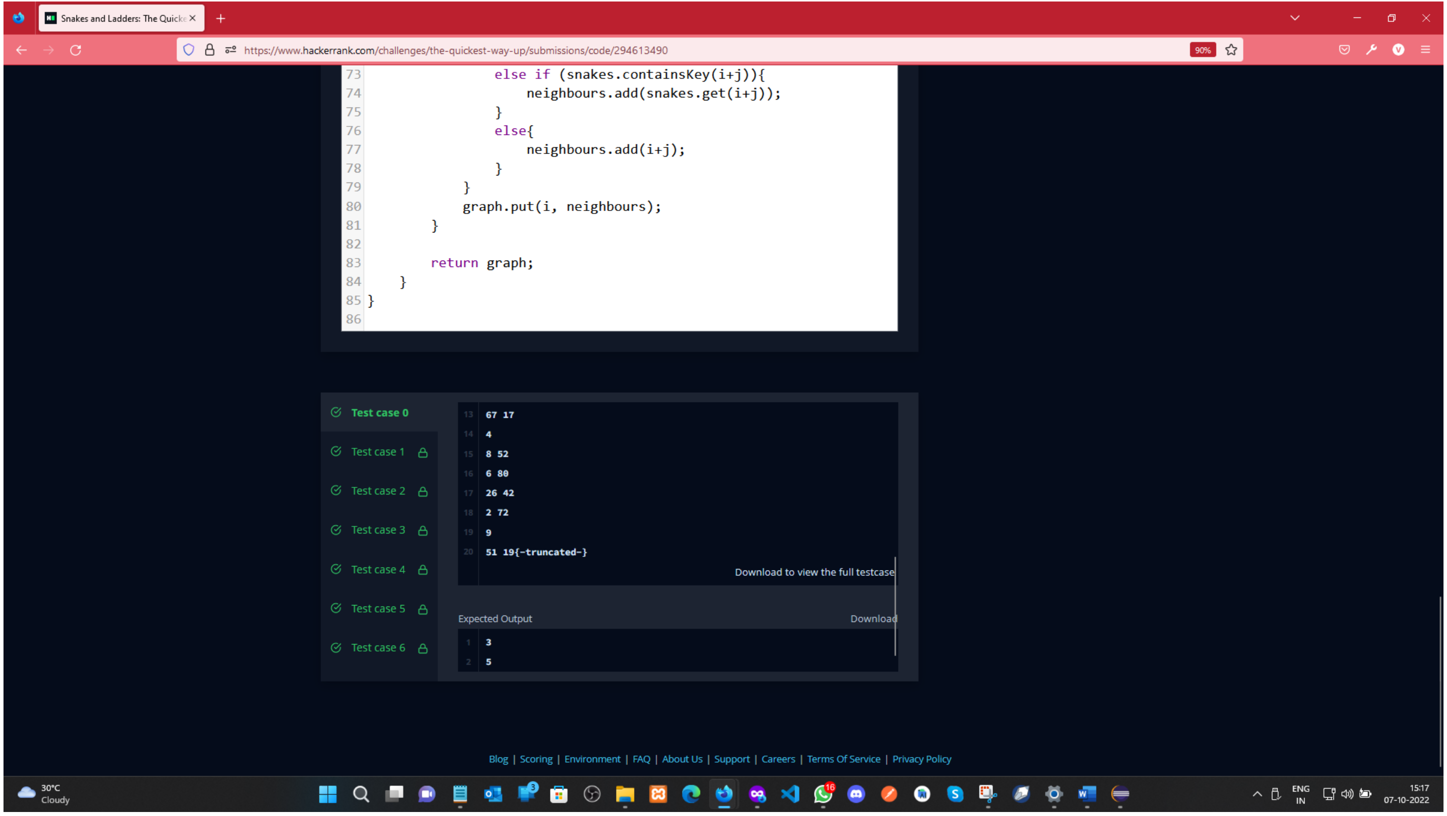
Dijkstra: Shortest Reach 2

Learn to use Dijkstra's shortest path algorithm !

Submitted Code

Language: Java 15

```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

```


Snakes and Ladders: The Quicke
https://www.hackerrank.com/challenges/the-quickest-way-up/submissions/code/294613490
90%
Test case 0
Test case 1
Test case 2
Test case 3
Test case 4
Test case 5
Test case 6
Download to view the full testcase
Download
Blog | Scoring | Environment | FAQ | About Us | Support | Careers | Terms Of Service | Privacy Policy
30°C
Cloudy
ENG
IN
15:17
07-10-2022

```java
                else if (snakes.containsKey(i+j)){
                    neighbours.add(snakes.get(i+j));
                }
                else{
                    neighbours.add(i+j);
                }
            }
            graph.put(i, neighbours);
        }

        return graph;
    }
}

```

```
67 17
4
8 52
6 80
26 42
2 72
9
51 19{-truncated-}
```

Expected Output

```
3
5
```

Learning outcomes (What I have learnt):

a. Learnt about Graph concept.

b. Learnt about BFS.

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.			