Snakes and Ladders: The Quickest Way Up

by idlecool

Markov takes out his Snakes and Ladders game and stares at the board, and wonders: If he had absolute control on the die (singular), and could get it to generate any number (in the range $1 - 6$) he desired, what would be the least number of rolls of the die in which he'd be able to reach the destination square (Square Number 100) after having started at the base square (Square Number 1)?

Rules

1. Markov has total control over the die and the face which shows up every time he tosses it. You need to help him figure out the minimum number of moves in which he can reach the target square (100) after starting at the base (Square 1).
2. A die roll which would cause the player to land up at a square greater than 100, goes wasted, and the player remains at his original square. Such as a case when the player is at Square Number 99, rolls the die, and ends up with a 5.
3. If a person reaches a square which is the base of a ladder, (s)he has to climb up that ladder, and he cannot come down on it. If a person reaches a square which has the mouth of the snake, (s)he has to go down the snake and come out through the tail - there is no way to climb down a ladder or to go up through a snake.

Constraints

The board is always of the size 10×10 and Squares are always numbered 1 to 100.

$1 <= T <= 10$
$1 <= \text{Number of Ladders} <= 15$
$1 <= \text{Number of Snakes} <= 15$

Square number 1 and 100 will not be the starting point of a ladder or a snake.
No square will have more than one of the starting or ending point of a snake or ladder (e.g. snake 56 to 44 and ladder 44 to 97 is not possible because 44 has both ending of a snake and a starting of a ladder)

Input Format

The first line contains the number of tests, T. T testcases follow.

For each testcase, the first line contain N(Number of ladders) and after that N line follow. Each of the N line contain 2 integer representing the starting point and the ending point of a ladder respectively.

The next line contain integer M(Number of snakes) and after that M line follow. Each of the M line contain 2 integer representing the starting point and the ending point of a snake respectively.

Output Format

For each of the T test cases, output one integer, each in a new line, which is the least number of moves (or rolls of the die) in which the player can reach the target square (Square Number 100) after starting at the base (Square Number 1). This corresponds to the ideal sequence of faces which show up

when the die is rolled.
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: To traverse the board via the shortest route, the player first rolls the die to get a 5, and ends up at square 6. He then rolls the die to get 6. He ends up at square 12, from where he climbs the ladder to square 98. He then rolls the die to get '2', and ends up at square 100, which is the target square. So, the player required 3 rolls of the die for this shortest and best case scenario. So the answer for the first test is 3.

Submissions: 11482
Max Score: 50
Difficulty: Medium

Rate This Challenge:
☆☆☆☆☆

More

Current Buffer (saved locally, editable)

Java 8

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

class Graph{

    private int V;
    private LinkedList<Integer> adj[];

    public Graph(int V){
        this.V = V;
        adj = new LinkedList[V];

        for(int i = 0 ; i < V ; i++){
            adj[i] = new LinkedList<Integer>();
        }
    }


    public void addEdge(int vertex, int sourceVertex){
        adj[sourceVertex].add(vertex);
    }

    public int doBFS(int startingNode){

        LinkedList<Integer> queue = new LinkedList<Integer>();
        queue.add(startingNode);

        boolean[] isVisited = new boolean[V];

        int[] output = new int[100];

        for(int i = 0 ; i < 100 ; i++){
            output[i] = Integer.MAX_VALUE;
        }

        output[0] = 0;

        while(queue.size() != 0){

            int s = queue.poll();

            if(s != 99){
                isVisited[s] = true;
            }

            Iterator<Integer> i = adj[s].listIterator();

            while(i.hasNext()){
                int ver = i.next();

                if(!isVisited[ver]){
                    queue.add(ver);

                    if(output[ver] > output[s] + 1){
                        output[ver] = output[s] + 1;
                    }
                }
            }
        }

        return output[99];
    }
}




public class Solution {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);
        int tst = scan.nextInt();

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

            HashMap<Integer,Integer> laders = new HashMap<Integer,Integer>();
            int ladrsCNT = scan.nextInt();

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

                int low = scan.nextInt() - 1;
                int high = scan.nextInt() - 1;

                laders.put(low,high);

```

```java
            }

            int snksCNT = scan.nextInt();
            HashMap<Integer,Integer> snakes = new HashMap<Integer,Integer>();

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

                int high = scan.nextInt() - 1;
                int low = scan.nextInt() - 1;
                snakes.put(high,low);
            }

            if(snakes.containsKey(99) || (snakes.containsKey(98) && snakes.containsKey(97) && snakes.containsKey(96)
&& snakes.containsKey(95) && snakes.containsKey(94) && snakes.containsKey(93))){
                System.out.println("-1");
            }
            else{

            Graph graph = new Graph(100);

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

                if(j == 99 || laders.containsKey(j) || snakes.containsKey(j)){
                    continue;
                }

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

                    if((j+k) > 99){
                        break;
                    }

                    if(snakes.containsKey(j + k)){
                        graph.addEdge(snakes.get(j+k),j);
                    }
                    else if(laders.containsKey(j + k)){
                        graph.addEdge(laders.get(j+k),j);
                    }
                    else{
                        graph.addEdge((j+k),j);
                    }
                }
            }


            System.out.println(graph.doBFS(0));

            }
        }

    }


}
```

Line: 1 Col: 1

Upload Code as File   Test against custom input   Run Code   Submit Code