USACO DEC15 Problem 'lightson' Analysis

by Nick Wu

In this problem, Bessie is traveling in a grid where certain squares are either on or off. Bessie can only walk on squares that are on. When Bessie is on certain squares, she is able to turn other squares on - our job is to figure out how many squares Bessie can turn on.

With the grid being small, we can simulate Bessie's moving through the grid as follows:

1. For a given square that Bessie can visit, turn on all squares that Bessie can. If Bessie turns on a square that is adjacent to one that she has already visited, recursively visit that square.
2. After turning on all squares, look at the squares that Bessie is next to. If anyone of them are on, but Bessie has not yet visited them, visit them.

Here is my code illustrating this process:

```
import java.io.*;
import java.util.*;
public class lightson {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader("lightson.in"));
        PrintWriter pw = new PrintWriter(new BufferedWriter(new
FileWriter("lightson.out")));
        StringTokenizer st = new StringTokenizer(br.readLine());
        int n = Integer.parseInt(st.nextToken());
        int m = Integer.parseInt(st.nextToken());
        switches = new ArrayList[n][n];
        on = new boolean[n][n];
        visited = new boolean[n][n];
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                switches[i][j] = new ArrayList<Pair>();
            }
        }
        for(int i = 0; i < m; i++) {
            st = new StringTokenizer(br.readLine());
            int a = Integer.parseInt(st.nextToken())-1;
            int b = Integer.parseInt(st.nextToken())-1;
            int c = Integer.parseInt(st.nextToken())-1;
            int d = Integer.parseInt(st.nextToken())-1;
            switches[a][b].add(new Pair(c, d));
        }
        LinkedList<Pair> q = new LinkedList<Pair>();
        q.add(new Pair(0, 0));
        on[0][0] = true;
        // start by searching the top-left corner
        search(0, 0);
        int ret = 0;
        for(boolean[] row: on)
            for(boolean col: row)
                if(col) ret++;
        pw.println(ret);
        pw.close();
    }

    static boolean[][] on;
    static boolean[][] visited;
    static int[] dx = new int[]{-1,1,0,0};
    static int[] dy = new int[]{0,0,-1,1};
    static ArrayList<Pair>[][] switches;

    public static void search(int x, int y) {
        // if we've already searched this square, don't do it again to save time
        if(isVisited(x, y)) return;
        visited[x][y] = true;
        for(Pair next: switches[x][y])
            // step 1 - turn on all squares that Bessie can
            if(!on[next.x][next.y]) {
                on[next.x][next.y] = true;
                if(hasVisitedNeighbor(next.x, next.y))
                    // this square is next to one Bessie can visit, so go search it
                    search(next.x, next.y);
            }

        for(int k = 0; k < dx.length; k++) {
            // check which neighbors of (x, y) are on
            int nx = x + dx[k];
            int ny = y + dy[k];
            if(isOn(nx, ny))
                // this is a neighboring square that was already on, so it can be visited
                search(nx, ny);
        }
    }

    // Returns true if and only if there is a visited neighbor of (x, y)
    public static boolean hasVisitedNeighbor(int x, int y) {
        for(int k = 0; k < dx.length; k++) {
            if(isOn(x + dx[k], y + dy[k]) && isVisited(x + dx[k], y + dy[k])) {
                return true;
            }
        }
        return false;
    }

    // Returns true if and only if square (x, y) is on.
    public static boolean isOn(int x, int y) {
        return x >= 0 && x < on.length && y >= 0 && y < on[x].length && on[x][y];
    }

    // Returns true if and only if square (x, y) has been visited
    public static boolean isVisited(int x, int y) {
        return x >= 0 && x < visited.length && y >= 0 && y < visited[x].length &&
visited[x][y];
    }

    static class Pair {
        public int x,y;
        public Pair(int x, int y) {
            this.x = x;
            this.y = y;
        }
    }
}
```