CS1020 Take-home Lab #5
Exercise #2: Obstacle Course

http://www.comp.nus.edu.sg/~cs1020/3_ca/takehomelabs.html

Objective:

- Solving an optimisation problem using recursion.

Recursion is the objective of this exercise. Hence, you must use recursion. If recursion is not used (or it is used but not in solving the problem but in other side quests), no attempt mark will be awarded.

Task statement:

An obstacle course comprises a sequence of Blocks of different heights. Traversing the obstacle course involves hopping from one Block to the next.

Your task is to write a program, **ObstacleCourse.java**, to compute the smallest possible number of hops required to go from the first Block to the last Block in the obstacle course.

An example of user input to be read by the program is as shown on the left:

```
5
1 3
2 4
3 3
4 4
1 0
```

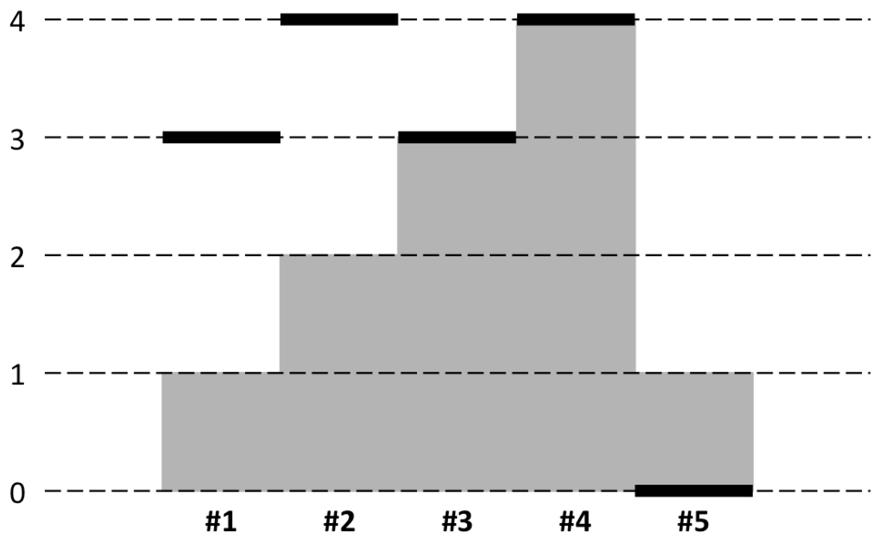
The first line contains a single positive integer, ***N***, the total number of Blocks in the obstacle course.

The first line is followed by ***N*** lines that correspond to ***N*** Blocks. Each line contains two non-negative integers, separated by a space. The first integer, H_i, is the height of the Block. The second integer, R_i, is the "hopping range" of the Block. The hopping range of the Block is the maximum height of a Block that we can hop to.

One of the following two conditions must be true if we want to hop from some Block X to some other Block Y:

- X and Y are adjacent Blocks.
- X and Y are *not* adjacent Blocks. The height of Y must be ≤ the hopping range of X. The height of each Block between X and Y must also be ≤ the hopping range of X.

The output consists of a single line containing an integer, which is the smallest possible number of hops.



The aforementioned rules work out in our example as follows:

- If we are currently on Block #1 (with $\boldsymbol{R_1}$ = 3), then…
  - we can hop to Block #2 because it is adjacent to Block #1.
  - we can hop to Block #3 because $\boldsymbol{H_2}$ = 2 ≤ $\boldsymbol{R_1}$ (ie. Block #2 does not obstruct Block #3), and because $\boldsymbol{H_3}$ = 3 ≤ $\boldsymbol{R_1}$.
  - we cannot hop to Block #4 because $\boldsymbol{H_4}$ = 4 > $\boldsymbol{R_1}$.
  - we cannot hop to Block #5 because it is obstructed by Block #4.
- If we are currently on Block #2 (with $\boldsymbol{R_2}$ = 4), then…
  - we can hop to Block #3 because it is adjacent to Block #2.
  - we can hop to Block #4 because $\boldsymbol{H_3}$ = 3 ≤ $\boldsymbol{R_2}$ (ie. Block #3 does not obstruct Block #4), and because $\boldsymbol{H_4}$ = 4 ≤ $\boldsymbol{R_2}$.
  - we can hop to Block #5 because $\boldsymbol{H_5}$ = 1 ≤ $\boldsymbol{R_2}$.
- If we are currently on Block #3 (with $\boldsymbol{R_3}$ = 3), then…
  - we can hop to Block #4 because it is adjacent to Block #3.
  - we cannot hop to Block #5 because $\boldsymbol{H_4}$ = 4 > $\boldsymbol{R_3}$ (ie. Block #4 obstructs Block #5).
- If we are currently on Block #4 (with hopping range $\boldsymbol{R_4}$ = 4), then…
  - we can hop to Block #5 because it is adjacent to Block #4.
- If we are currently on Block #5 (with hopping range $\boldsymbol{R_5}$ = 0), then…
  - we have completed the obstacle course because Block #5 is the last Block in the sequence.

In our example, the path from Block #1 to Block #5 that requires the least hops is precisely:

#1 → #2 → #5

This path has **2** hops. Therefore, the output of our program is simply **2**. (See sample run #1 below.)

The code for processing the user input is given. Your task is to complete the recursive **countHops** method of the **ObstacleCourse** class.

Number of submissions:

You are given **10** submissions. Only the final submission will be graded.

Sample run #1:

```
5
1  3
2  4
3  3
4  4
1  0
2
```

Sample run #2:

```
3
1  3
3  0
2  0
1
```

Sample run #3:

```
1
1  0
0
```

END