NOI 2007 TASKS OVERVIEW

	Executable file	*Input file*	*Output file*
Task 1: GIFT	GIFT.EXE	GIFT.IN	GIFT.OUT
Task 2: RECT	RECT.EXE	RECT.IN	RECT.OUT
Task 3: CIPHER	CIPHER.EXE	CIPHER.IN	CIPHER.OUT
Task 4: HOLE	HOLE.EXE	HOLE.IN	HOLE.OUT
Task 5: JAWBREAK	JAWBREAK.EXE	JAWBREAK.IN	JAWBREAK.OUT
Task 6: STREET	STREET.EXE	STREET.IN	STREET.OUT

Notes:

1. **Each task will be tested on 5 data sets.**
2. **The maximum execution time for every task is 10 seconds.**
3. **Each data set is worth 20 points.**
4. **For each question, the 5 data sets vary in size, starting from small, and ending with sizes around the limits given in each question.**
5. **Either zero mark or full mark (20 points) is awarded to your answer to each data set. There is no partial credit per data set.**
6. **The task statements are contained on 11 pages following this overview page.**
7. **In the event of tie breaking, a bigger-numbered task is favoured over a smaller-numbered one.**

HAPPY PROGRAMMING!

TASK 4: HOLE

Background: A group of scientists want to monitor a huge forest. They plan to airdrop small sensors to the forest. Due to many unpredictable conditions during airdropping, each sensor will land in a random location in the forest. After all sensors have landed, there will be square regions in the forest that do not contain any sensor. Let us call such a region a *hole*. It is desirable that all holes are small. This can be achieved by airdropping very large number of sensors. On the other hand, those sensors are expensive. Hence, the scientists want to conduct a computer simulation to determine how many sensors should be airdropped, so that the chances of having a large hole are small. To conduct this simulation, a subroutine is required that, given the locations of the sensors, outputs the size of the largest hole. This subroutine has to be very efficient (i.e. fast) since the simulation will be repeated many times with different parameters. You are tasked to write this efficient subroutine.

Problem formulation: Consider an n by n array A of 0 and 1's. We say that there is a hole in A at (x, y) with width d if the value of $A[i, j]$ is 0, where

$$\begin{aligned} i &= x, (x+1), (x+2), \ldots, (x+d-1) \\ j &= y, (y+1), (y+2), \ldots, (y+d-1), \end{aligned}$$

and $(x + d - 1) < n, (y + d - 1) < n$. That is, all values within the "square" at (x, y) with width d are 0's. Note that the indices of the array start from 0.

Given the array A, we want to find the width of the largest hole.

Example

The following figure shows an array where the entry $A[i, j]$ is at the i-th row and j-th column. The arrow points to the location of $A[3, 7]$.

In this array, there is a hole of width 3 at (0,0), and it is not possible to have a hole of width 4 at (0,0). The largest hole has width 5 and it is at (1,2), which is highlighted in the figure.

	0	1	2	3	4	5	6	7	
0	0	0	0	0	0	1	0	0	
1	0	0	**0**	**0**	**0**	**0**	**0**	0	
2	0	0	**0**	**0**	**0**	**0**	**0**	0	
3	0	1	**0**	**0**	**0**	**0**	**0**	0	$\leftarrow A[3, 7]$
4	0	0	**0**	**0**	**0**	**0**	**0**	0	
5	0	0	**0**	**0**	**0**	**0**	**0**	1	
6	0	0	0	0	1	0	0	0	
7	1	0	0	0	0	0	0	0	

Input File: `HOLE.IN`

The input file represents the input array A. The following is the input file for the array shown above.

```
8
5
3 1
7 0
6 4
0 5
5 7
```

The first line contains a positive integer $n \leq 1{,}024$ which is the width of the array. The second line contains an integer k, which is the number of 1's in the array. Next, it is followed by k more lines which specify the entries with value 1. Each line contains two integers x and y, which indicates that $A[x, y] = 1$.

Since the value of n may be as large as $1{,}024$, your program has to run fast enough to handle arrays of that size.

Output File: HOLE.OUT

The output file consists of an integer, which is the width of the largest hole. If there is no hole in the array, the output is 0. The following is the output of our example.

```
5
```

Hints

There are different fast methods that find the largest hole. Below are hints to two different methods.

Hint 1: Consider these 2 values:

d_1: the width of the largest possible hole at (1,1), and
d_2: the width of the largest possible hole at (2,2).

What is the relationship between d_1 and d_2? Suppose we know the value of d_2, is there a fast method to compute d_1?

Hint 2: Consider these 2 values:

s_1: the number of 1's within the square at (0,0) with width 7, and
s_2: the number of 1's within the square at (0,1) with width 7.

What is the relationship between s_1 and s_2? Suppose we know the value of s_1, is there a fast method to compute s_2?