TASK	**KAMPANJA**	**MJESEC**	**SETNJA**	**TRAMPOLIN**
source code	`kampanja.pas` `kampanja.c` `kampanja.cpp`	`mjesec.pas` `mjesec.c` `mjesec.cpp`	`setnja.pas` `setnja.c` `setnja.cpp`	`trampolin.pas` `trampolin.c` `trampolin.cpp`
input	standard input (*stdin*)			
output	standard output (*stdout*)			
time limit	1 second	5 seconds	1 second	0.5 seconds
memory limit	128 MB	256 MB	256 MB	64 MB
point value	**100**	**100**	**100**	**100**
	400			

The elections are nearing, so President Amabo Kcarab is planning a tour of the States, with speeches in WDC and LA. To provide adequate security, the Secret Service needs to constantly monitor **all cities** that the President will pass through (including WDC and LA).

Of course, the federal budget needs to be spent responsibly, so the President will not be using AF1, but will be travelling by car. Also, the Secret Service will plan the President's tour from WDC to LA **and back to WDC** such that the **least possible** number of cities needs to be monitored.

For this problem, assume that the States consist of **N** cities, denoted by numbers from 1 to **N**, and **M** **unidirectional** Interstates, with each Interstate linking two different cities. WDC is city number 1, LA number 2.

Write a program to compute the minimum number of cities that need to be monitored such that there exists a path from WDC to LA and back to WDC passing only through monitored cities.

INPUT

The first line of input contains the two integers **N** and **M** ($2 \le$ **N** ≤ 100, $2 \le$ **M** ≤ 200), the number of cities and the number of Interstates linking the cities, respectively.

Each of the following **M** lines contains two different integers **A**, **B** ($1 \le$ **A**, **B** $\le$ **N**), the beginning and ending city served by the given Interstate. No two Interstates link the same two cities in the same direction, but two Interstates can link the same two cities in opposite directions.

OUTPUT

The first and only line of output must contain the minimum number of cities that need to be monitored.

SCORING

In test data worth a total of 20 points, **N** will be at most 20.

Note: Test data will ensure that a solution will always exist.

SAMPLE TESTS

input	input

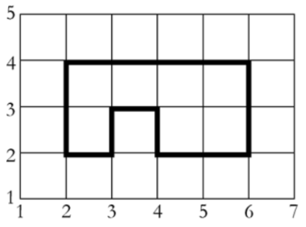
```
6 7
1 3
3 4
4 5
5 1
4 2
2 6
6 3
```
	```
	9 11
	1 3
	3 4
	4 2
	2 5
	5 3
	3 6
	6 1
	2 7
	7 8
	8 9
	9 1
	```

output	**output**
6	6

First sample description: The President can take the following route: **1** -> 3 -> 4 -> **2** -> 6 -> 3 -> 4 -> 5 -> **1**. Since he needs to pass through each city at least once, the solution is 6.

The fully automated lunar station's upkeep is the responsibility of the service robot M1RK8. The robot is moving along a **circular** track (a track forming a cycle). The track consists of **straight segments** with north, south, east, or west orientations, and in-place 90-degree **turns** to the left or right. The length of each straight segment is a positive integer number of metres, so the entire track can be shown in a Cartesian plane with segments passing through **lattice** (integer-coordinates) points and **parallel** to the coordinate axes. The track **never intersects** with itself, and no two segments touch or overlap (except, of course, at adjacent endpoints, i.e. turns).

Here is an example track with 8 turns:



A meteorite has struck the Moon near the station, so most sensors went offline. Thus, neither the **position** of the robot nor the **direction** that it is facing is currently known. We cannot begin repair work on the station until we have established the robot's position along the track.

This task will not be easy since the only command available for moving the robot is "**move K**", where **K** is the positive integer number of metres that the robot must move along the track. After executing the command, the robot returns **two integers**: **L** and **R**. The number **L** is the number of **left** turns that the robot had to make **while executing the last move command**, and **R** is the number of **right** turns.

Notes: The robot's starting position, as well as the position after each command, is always a **lattice** (integer-coordinates) point. If the robot ends the move command at a turn point, it will turn in the direction of the rest of the track **before** ending the move command, so the next move command will start with going straight.

For example, if the robot is at the coordinates (3, 3) and facing east, after executing the command "move 4" it will end up at (6, 2) facing north and return the result "2 1", meaning that it has made two left and one right turns while executing the command. After that, the command "move 1" will move the robot to (6, 3) and return "0 0".

Write a program that will determine the position of the robot using **at most 5000** commands of the form "move **K**". The program has to finish with the command "**stop X Y**", where **X** and **Y** are the **final coordinates of the robot** after executing all comands.

INTERACTION

Before interacting with the robot, the program needs to read the following data from the input:

- The first line of input contains the positive integer **N** ($6 \le N \le 10\,000$), the number of turns on the track, which is also equal to the number of straight segments.
- The following **N** lines of input contain the coordinates of turns in order that they appear along the track, one per line, given as pairs of integers **X** and **Y** ($1 \le X \le 100\,000$, $1 \le Y \le 100\,000$).

Notes: The **direction** of the robot along the track can be in order that the coordinates are given, or the opposite order. The track will be such that tke robot's position is uniquely determinable, i.e. the test data will never contain two track points with different orientation that give the same result for every possible "move" command sequence.

After reading the data above, it is possible to give commands to the robot using standard output. Make sure to *flush* the output after each "move **K**" command; after flushing, you can read in the two numbers **L** and **R** from standard input. **K** must be between 1 and 1 000 000 000, inclusive.

After determining the coordinates of the robot, your program must output "stop **X Y**" to standard output, *flush* the output, and regularly finish execution.

SCORING

The following constraints will hold in a subset of the test data:

- worth a total of 40 points: **N** ≤ 100 and **X**, **Y** ≤ 50
- worth a total of 60 points: **N** ≤ 100 and **X**, **Y** ≤ 100 000
- worth a total of 80 points: **N** ≤ 2000 and **X**, **Y** ≤ 100 000

TESTING

Your solution can be tested either locally or using the evaluator system. For both methods you first need to create a file describing the desired test case.

- The first line must contain the positive integer **N**, the number of turns.
- The following **N** lines must contain the coordinates of turns in the order that they appear along the track: two integers, **X** and **Y**, for each line.
- The following line must contain the two integers **X0** and **Y0** and the word **S**. **X0** and **Y0** represent the starting coordinates of the robot, while **S** describes the robot's direction. **S** can be one of the following: "**same**" if the robot is moving along the track in the order of turns given in input, or "**opposite**" if the robot is moving in the opposite direction. The starting coordinates can be either at a turn or along a straight segment of the track.

An example of a test input file corresponding to the track example given earlier:

```
8
2 4
6 4
6 2
4 2
4 3
3 3
3 2
2 2
3 3 opposite
```

If you wish to test using the evaluator system, you first need to submit the source code using the SUBMIT page, and then send the test input file using the TEST page.

Local testing (Unix only) can be carried out using the mjesec_test file available for download from the evaluator system. The testing is done using the following command:

```
./mjesec_test ./solution_executable test_input_file
```

Whatever testing method you choose, the output will contain feedback on whether your program has correctly solved the test case, as well as information about the commands issued by your program, responses received, and robot positions after each command.

Mirko is taking all his friends to a Zaz concert soon to be held in Zagreb. He has already obtained the tickets, and is now walking around the neighbourhood delivering them to everyone.

Mirko's neighbourhood can be represented by a Cartesian plane. While walking, Mirko is stepping only on lattice (integer-coordinates) points of that plane. He takes a **single step** to move to any one of **eight** adjacent lattice points (up, down, left, right, or diagonally in any direction).

Each one of Mirko's friends also lives at some lattice point (**x**, **y**), and is willing to walk some distance to meet him. Specifically, Mirko will meet a given friend at some lattice point with distance of **at most P steps** from this friend's house, with **P** depending on the specific friend's laziness.

After finishing his stroll, Mirko has recalled the **order** in which he has met with his friends. Compute the **minimum possible number of steps** that Mirko had to make during his stroll. Mirko's starting and ending position are not known.

INPUT

The first line of input contains the positive integer **N** (2 ≤ **N** ≤ 200 000), the number of Mirko's friends.

Each of the following **N** lines contains three integers describing one friend: **x**, **y**, and **P** (0 ≤ **x**, **y**, **P** ≤ 200 000). **Friends are given in the order in which Mirko has met them.**

OUTPUT

The first and only line of output must contain the required minimum possible number of steps that Mirko had to make.

SCORING

In test data worth a total of 30 points, all numbers in the input (including **N**) will be at most 200.

In test data worth an additional 30 points, no friend will have **P** greater than 10.

SAMPLE TESTS

Input 3 3 10 2 8 4 2 2 5 2 **output** 4	**input** 4 3 3 5 7 11 5 20 8 10 30 18 3 **output** 19

First sample description: Mirko could have started his stroll at coordinates (4, 8), met the first friend, taken two steps to the point (6, 6), met the second friend, taken two steps to the point (4, 5) and met the third friend there.

There are many action superheroes out there: Batman, Spiderman, Superman, Icantwriteman etc. Among them, there is one gentleman called Kickass. Today he wants to mimic Spiderman, so he has chosen a row of tall skyscrapers to jump around on.

Specifically, he has chosen a sequence of **N** skyscrapers numbered 1 through **N** from left to right. He is initially located on the **K**th skyscraper. Unfortunately, Kickass has very limited powers, and can therefore jump only to the **adjacent** skyscraper to the left or right, and only if that skyscraper's height is **not greater** than the height of the skyscraper he is currently on. However, anticipating this and not wanting to look weak, he has positioned **trampolines** on top of some skyscrapers, and from these skyscrapers he can jump onto **any other** skyscraper, no matter how tall or where that skyscraper is.

Find the **maximum number of different skyscrapers** Kickass can visit in a chain of jumps starting from the skyscraper numbered **K**. If a skyscraper is visited more than once, we still count it only once. Moreover, skyscraper **K** is counted even if we never return to it.

INPUT

The first line of input contains the two integers **N** and **K** ($3 \le$ **N** $\le 300\,000$, $1 \le$ **K** $\le$ **N**), the total number of skyscrapers and the starting skyscraper, respectively.

The second line of input contains **N** integers less than 10^6, the heights of skyscrapers in order from left to right.

The third line of input contains a sequence of **N** characters '.' or 'T'. If the **i**th character is 'T', then there is a trampoline positioned on the top of skyscraper **i**.

OUTPUT

The first and only line of output must contain the required maximum number of visited skyscrapers.

SAMPLE TESTS

Input
 	
`6 4`
`12 16 16 16 14 14`
`.T....`
 	
output
 	
`5`
	input
	
	`10 1`
	`10 7 3 1 1 9 8 2 4 10`
	`..T..T....`
	
	output
	
	`7`

Second sample description: the sequence of visited skyscrapers could be the following:

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 10 \rightarrow 9 \rightarrow 8.$$