Problem Set

Please check that you have 11 problems that are spanned across 32 pages in total (including Korean translation and this cover page).

Problem A
Cleaning Robot
Time Limit: 1 Second

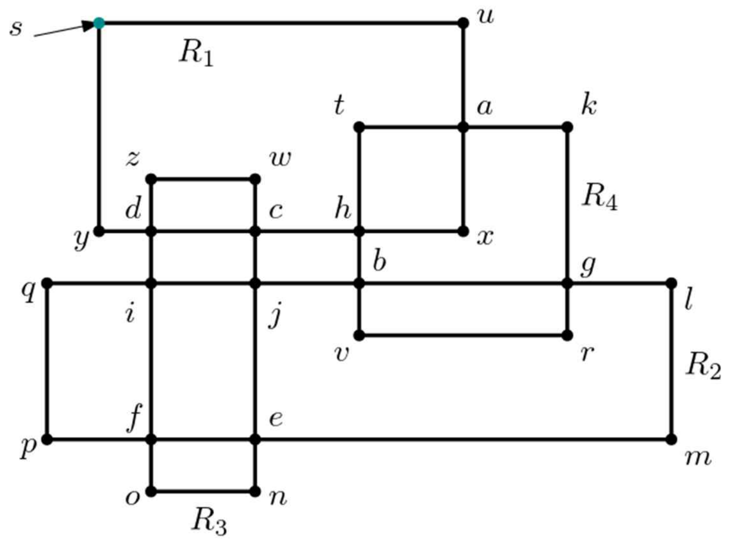
The road network of a city consists of a set of axis-parallel rectangles. Each rectangle road may overlap with others as shown in the figure below. The city operates an autonomous cleaning robot that is responsible to clean the entire roads every day. During cleaning, each rectangle is classified into one of three types as follows.

1) Uncleared (UC): A rectangle that the cleaning robot has never visited.
2) Partially cleared (PC): A rectangle that has been cleaned partially.
3) All cleared (AC): A rectangle that has been completely cleaned.

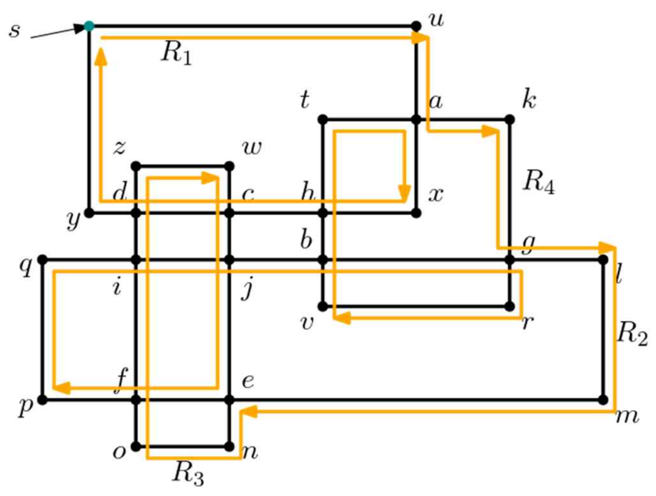
The cleaning robot works according to the following routing algorithm.

- The robot moves in a clockwise direction while cleaning each rectangle.
- If the robot encounters a road of a new UC rectangle during the cleaning of a rectangle, it will turn at the crossing point and move towards the UC rectangle.
- When a rectangle is completely cleaned as the robot reaches a point (x, y) and another PC rectangle is encountered at (x, y), the robot resumes cleaning the PC rectangle at that same point (x, y).
- If the robot returns to the starting point, it will stay at that point afterwards.
- It takes the robot one second to move a unit distance along edges, and takes two seconds in changing direction at a crossing point or corner.

Let us explain the routing algorithm with the following example. The example road network consists of four rectangles $\{R_1, R_2, R_3, R_4\}$ where the starting point is s of R_1.



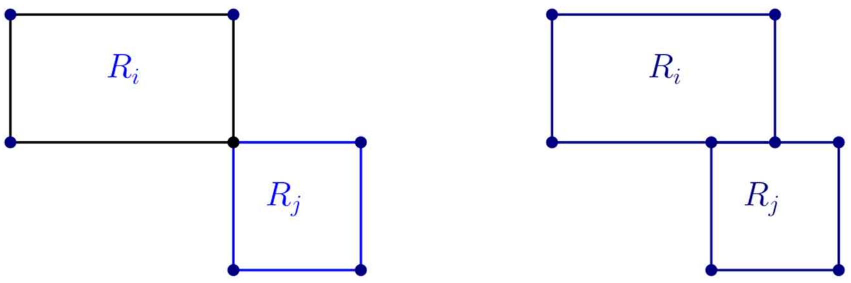
The orange lines in the following figure depict the trajectory of the cleaning robot starting from s of R_1 .



Given information about n rectangles, we want to know the exact locations of the robot for the five query points $t_i (1 \le i \le 5)$ in time. Note that the starting point is designated to the upper left corner of the rectangle R_1.

Input

Your program is to read from standard input. The input starts$1 \le i \le 5$ with a line containing one integer, n $(1 \le n \le 50)$, where n is the number of rectangles R_i. The second line gives the five query points $t_i (1 \le i \le 5)$ in time where $1 \le t_i \le 100{,}000$. The i-th line of following n lines gives four integers x_l, y_l, x_u, y_u for R_i where (x_l, y_l) is the lower left corner and (x_u, y_u) is the upper right corner where $1 \le x_l < x_u \le 1{,}000$, and $1 \le y_l < y_u \le 1{,}000$. Note that the intersection points between rectangles are all in the middle of edges, not at corner points, and there are no edge overlaps among rectangles, so the configurations below are not given in this problem. Also note that rectangles are not necessarily all connected into one component.



Output

Your program is to write to standard output. Print exactly five lines. The line should contain two integers x_i, y_i where (x_i, y_i) is the location of the cleaning robot at time $t_i (1 \le i \le 5)$.

The following shows sample input and output for three test cases. Note that at time $t = 0$, the robot is ready to start at the starting point.

Sample Input 1	Output for the Sample Input 1

4
1234 10000 700 3000 5000
100 500 300 800
100 100 900 400
150 30 190 550
230 350 700 700
	900 376
	100 800
	696 700
	190 452
	230 476

Sample Input 2

```
7
9001 8002 7003 6004 5005
200 100 300 800
350 100 500 800
600 100 700 800
100 600 800 700
100 400 800 500
100 200 800 300
900 200 1000 700
```

Output for the Sample Input 2

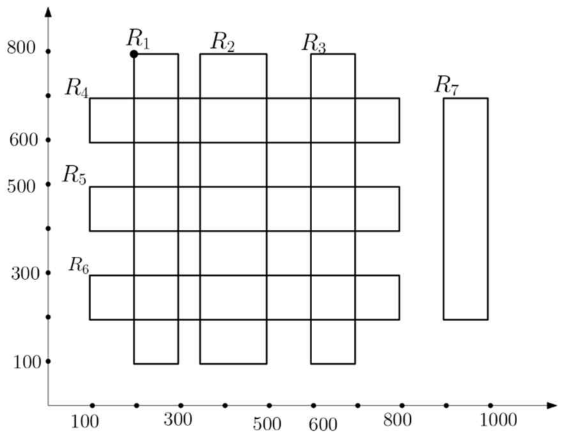
```
263 100
154 600
551 700
500 142
235 400
```

Sample Input 3

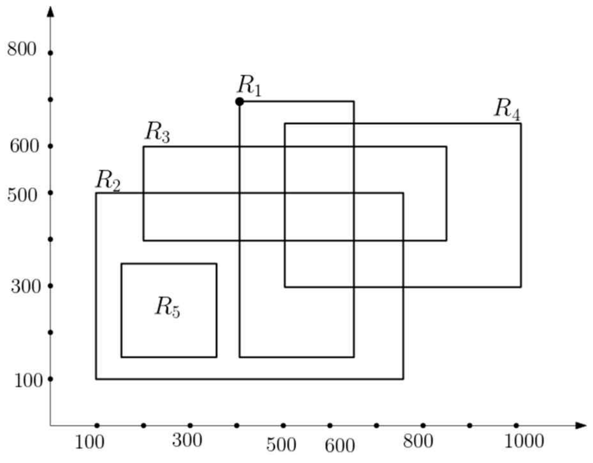
```
5
1298 5574 3332 1794 7141
400 150 650 700
100 100 750 500
200 400 850 600
500 300 1000 650
150 150 350 350
```

Output for the Sample Input 3

```
750 262
500 540
812 600
418 100
400 699
```



Sample Input 2



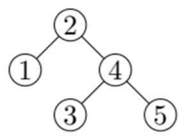
Sample Input 3

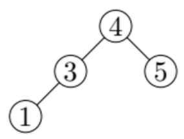
Problem B
Complexity Measure
Time Limit: 3 Seconds

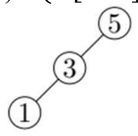
As the number of International Criminals of Poor Coding (ICPC) has been increasing rapidly, the government established a specialized prison. Now these ICPCs, the programmers who have written extremely bad codes, are supposed to be arrested and brutally punished in this place. One of the harsh punishments is drawing binary search trees. Every morning, the prisoners are given an integer sequence, and they are forced to draw several binary search trees according to the certain rules.

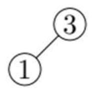
More precisely, given a sequence $X = x_1, x_2, \cdots, x_n$ of n distinct integers, let $T(X)$ be the binary search tree constructed by performing the standard way of insertion of each element of X in order; note that it holds for every $1 \leq i < j \leq n$ that the node with key x_i cannot be a descendant of the node with key x_j. Then the prisoners must draw $n - 1$ binary search trees $T(X[1..n]), T(X[2..n]), \cdots, T(X[n-1..n])$ where $X[i..n]$ is the suffix sequence $x_i, x_{i+1}, \cdots, x_n$ of X. For ease of verification, the prisoners must fill a table representing the trees. This table has $n - 1$ rows and n columns and the cell in the i-th row and j-th column has to be filled with the key of the parent of the node with key x_j in the binary search tree $T(X[i..n])$.

For example, if $X = 2,4,5,3,1$, the cell in the second row and the fifth column needs to be filled with 3, because the parent of the node with key $x_5 (= 1)$ in $T(X[2..5])$ has the key 3. The binary search trees and the entire table for X are as in Fig. 1.


(a) $T(X[1..5])$


(b) $T(X[2..5])$


(c) $T(X[3..5])$

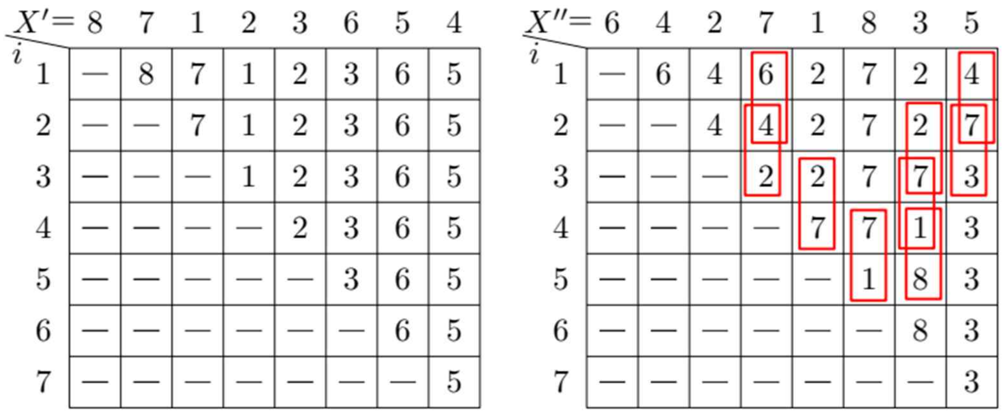

(d) $T(X[4..5])$

i \ $X=$	2	4	5	3	1
1	—	2	4	4	2
2	—	—	4	4	3
3	—	—	—	5	3
4	—	—	—	—	3

(e) Table

Fig. 1 Binary search trees for $\boldsymbol{X = 2, 4, 5, 3, 1}$ and the table representation.

One day, the prison administrators noticed that some sequences are too easy to fill the table while others are not. For example, consider a sequence $X' = 8,7,1,2,3,6,5,4$. When looking at its table, one can notice that each row is a suffix of the first row (Fig. 2(a)). Such an easy case should not be given to these guilty prisoners. On the other hand, another sequence $\boldsymbol{X'} = 6,4,2,7,1,8,3,5$ is more complicated in the sense that the table has many pairs of adjacent nonempty cells in the same column with different values (indicated with red boxes in Fig. 2(b)). Such pairs are called *critical changes*. In short, X' has no critical changes while X'' has 9 critical changes, from which one can say that X'' is more difficult than X'. The prison administrators want to use the number of critical changes as a complexity measure of a sequence so that they can control the difficulty of the tree-drawing task.



i \ $X'=$	8	7	1	2	3	6	5	4
1	—	8	7	1	2	3	6	5
2	—	—	7	1	2	3	6	5
3	—	—	—	1	2	3	6	5
4	—	—	—	—	2	3	6	5
5	—	—	—	—	—	3	6	5
6	—	—	—	—	—	—	6	5
7	—	—	—	—	—	—	—	5

(a) Table for X'

i \ $X''=$	6	4	2	7	1	8	3	5
1	—	6	4	6	2	7	2	4
2	—	—	4	4	2	7	2	7
3	—	—	—	2	2	7	7	3
4	—	—	—	—	7	7	1	3
5	—	—	—	—	—	1	8	3
6	—	—	—	—	—	—	8	3
7	—	—	—	—	—	—	—	3

(b) Table for X''

Fig. 2 Tables for two sequences $\boldsymbol{X'} = \mathbf{8,7,1,2,3,6,5,4}$ and $\boldsymbol{X'} = \mathbf{6,4,2,7,1,8,3,5}$.
The numbers of critical changes for $\boldsymbol{X'}$ are $\boldsymbol{X''}$ are **0** and **9**, respectively (indicated with red boxes).

Given a sequence of n distinct integers, write a program to output the number of critical changes of the sequence.

Input

Your program is to read from standard input. The input consists of two lines. The first line contains an integer n ($2 \le n \le 250{,}000$), where n is the length of the sequence. The following line contains n integers that comprise the input sequence. The integers are distinct and range from 1 to n.

Output

Your program is to write to standard output. Print exactly one integer indicating the number of critical changes of the sequence.

The following shows sample input and output for three test cases.

Sample Input 1	Output for the Sample Input 1
5 2 4 5 3 1	2

Sample Input 2	Output for the Sample Input 2
8 8 7 1 2 3 6 5 4	0

Sample Input 3	Output for the Sample Input 3
8 6 4 2 7 1 8 3 5	9

Problem C
Covers
Time Limit: 1 Second

You are given two strings T and P. Your aim is to create T with P. You first begin with an empty string. Then you can do one of three operations shown below.

- Put P at the end of the current string. This operation costs 0.
- Put a character at the end of the current string. This operation costs 1.
- Delete characters at the end of the current string and put P. This operation costs the number of characters you deleted.

For example, assume that $T = aabaabaa$ and $P = aaba$. There are two ways to create T with P as follows, where ϵ denotes an empty string.

- $\epsilon \rightarrow aaba \rightarrow aabaa \rightarrow aabaab \rightarrow aabaaba \rightarrow aabaabaa$, or
- $\epsilon \rightarrow aaba \rightarrow aab \rightarrow aabaaba \rightarrow aabaabaa$.

The former costs four as it first puts P (cost 0) and four characters (a, b, a, and a, cost 4). The latter costs two as it first puts P (cost 0), then deletes one character (a, cost 1) and puts P (cost 0), and finally puts a character (a, cost 1). We choose the latter and we can see that it has the minimum cost.

Given T and P, write a program which computes the minimum cost to create T with P.

Input
Your program is to read from standard input. The input starts with a line containing two integers, m and n ($1 \leq m \leq 100{,}000$, $1 \leq n \leq 200{,}000$), where m is the length of P and n is that of T. The second line contains P and the third line contains T. Both P and T are in English lower-case letters.

Output
Your program is to write to standard output. Print exactly one line. The line should contain the minimum cost to create T with P.

The following shows sample input and output for three test cases.

Sample Input 1	Output for the Sample Input 1

`4 8`
`aaba`
`aabaabaa`
	`2`

Sample Input 2	Output for the Sample Input 2

`4 8`
`aaba`
`ccdccdcc`
	`8`

Sample Input 3	Output for the Sample Input 3

`4 8`
`abab`
`abababab`
	`0`

Problem D
Diagonal Flipping
Time Limit: 1 Second

We are given an $m \times n$ grid that consists of 0s and 1s. We have two types of diagonal operations like the following two figures. The type A diagonal flipping operation to a grid position (i, j) is to flip all the elements in the positions $(i + k, j - k)$ of the grid for any integer k. If we flip the element 0, then it becomes 1. If we flip the element 1, then it becomes 0. The type B diagonal flipping operation to a grid position (i, j) is to flip all the elements in the positions $(i + k, j + k)$ of the grid for any integer k. Note that a grid position (p, q) is valid only when $0 \le p \le m - 1, 0 \le q \le n - 1$.

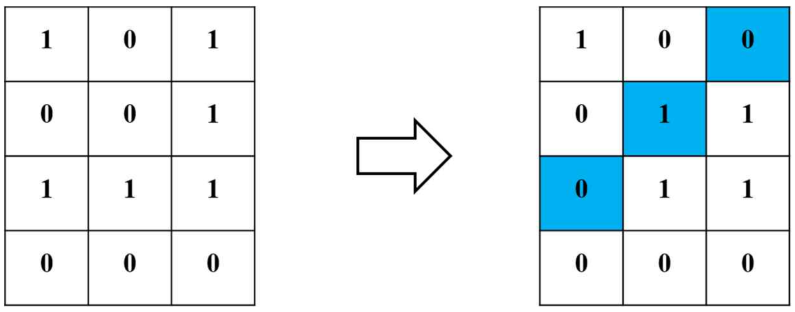


Fig 1. Type ***A*** diagonal operation to the grid position $(\mathbf{2}, \mathbf{0})$

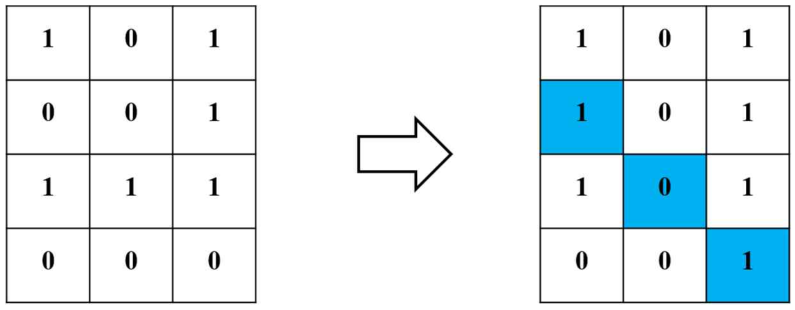


Fig 2. Type ***B*** diagonal operation to the grid position $(\mathbf{2}, \mathbf{1})$

Fig 1 shows the type A diagonal flipping operation to the grid position $(2, 0)$. Note that the type A diagonal flipping operations to the grid positions $(1, 1)$ or $(0, 2)$ have the same effect. Fig 2 shows the type B diagonal flipping operation to the grid position $(2, 1)$. The type B diagonal flipping operations to the grid positions $(1, 0)$ or $(3, 2)$ have the same effect.

Given an information of an $m \times n$ grid, write a program to output the minimum number of the diagonal operations to make all the elements in the grid to zeros.

Input
Your program is to read from standard input. The first line of input contains two positive integers $m (1 \le m \le 1{,}000)$ and $n (1 \le n \le 1{,}000)$ where m and n indicate the number of rows and columns of the grid, respectively. The rows of the grid are numbered from 0 to $m - 1$ and the columns are numbered from 0 to $n -$

1. In the following m lines, the i-th line contains n numbers, separated by spaces, which are zero or one that correspond to the row $i - 1$ of the grid.

Output

Your program is to write to standard output. Print exactly one line. The line should contain the minimum number of the diagonal operations to make all the elements of the grid to zeros. If it is not possible to make all the elements of the grid zeros, the program should print the number `-1`.

The following shows sample input and output for four test cases.

Sample Input 1

```
4 3
1 0 1
0 0 1
1 1 1
0 0 0
```

Output for the Sample Input 1

```
4
```

Sample Input 2

```
2 2
1 1
1 1
```

Output for the Sample Input 2

```
2
```

Sample Input 3

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

Output for the Sample Input 3

```
3
```

Sample Input 4

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

Output for the Sample Input 4

```
-1
```

Problem E
Matrix Game
Time Limit: 1 Second

Hoon and Chan are playing a matrix game. Each of both has $N \times N$ matrices $A = (a_{ij})$ and $B = (b_{ij})$, respectively. The game is composed of M rounds and in each round, Hoon and Chan simultaneously call the numbers α and β, respectively, which are integers between 1 and N. Then Hoon and Chan obtain the points of $a_{\alpha\beta}$ and $b_{\alpha\beta}$, respectively.

In the last night's dream, Hoon looked at M numbers $\beta_1, \beta_2, \ldots, \beta_M$ which Chan will call in the rounds of game. Thus Hoon may select M numbers $\alpha_1, \alpha_2, \ldots, \alpha_M$ to maximize the term Diff, where Diff is defined as $\sum_{k=1}^{M} \left| a_{\alpha_k \beta_k} - b_{\alpha_k \beta_k} \right|$, that is, the total sum of absolute differences of their points.

Given M numbers which Chan calls in the rounds, write a program to output the maximum value of Diff when Hoon chooses M numbers to maximize it.

Input
Your program is to read from standard input. The input starts with a line containing two integers, N and M ($1 \le N \le 1{,}000$, $1 \le M \le 1{,}000{,}000$), where N is the size of both matrices which Hoon and Chan have in the game and M is the number of rounds of the game. In the following N lines, the i-th line contains N integers to represent the values on the i-th row of Hoon's matrix. In the following N lines, the i-th line contains N integers to represent the values on the i-th row of Chan's matrix. The values of both matrices are between 0 and 1,000. The next line contains M integers between 1 and N that Chan calls in the rounds of the game.

Output
Your program is to write to standard output. Print exactly one line. The line should contain the maximum value when Hoon selects M numbers such that Diff is maximized.

The following shows sample input and output for two test cases.

Sample Input 1
```
2 2
1 2
0 1
3 1
2 1
2 1
```

Output for the Sample Input 1
```
3
```

Sample Input 2
```
2 3
5 0
2 6
3 3
3 2
1 1 2
```

Output for the Sample Input 2
```
8
```

Problem E
행렬 게임

제한 시간: 1 초

훈과 찬은 행렬 게임을 하고 있다. 두 사람은 각각 $N \times N$ 행렬 $A = (a_{ij})$와 $B = (b_{ij})$를 가지고 있다. 게임은 M개 라운드로 구성되고, 각 라운드에서 훈과 찬은 동시에 각각 숫자 α와 β를 부른다. 여기서, α와 β는 1과 N사이의 정수이다. 그러면 훈과 찬은 각각 $a_{\alpha\beta}$와 $b_{\alpha\beta}$의 점수를 얻는다.

전날 밤 꿈에서, 훈은 찬이 게임의 각 라운드에서 부를 M개 숫자 $\beta_1, \beta_2, \ldots, \beta_M$를 보았다. 따라서 훈은 Diff 가 최대가 되는 M개 숫자 $\alpha_1, \alpha_2, \ldots, \alpha_M$을 선택할 수 있다. 여기서, Diff 는 $\sum_{k=1}^{M} \left|a_{\alpha_k \beta_k} - b_{\alpha_k \beta_k}\right|$로 정의하고, 이것은 둘의 각 라운드 점수의 절대값 차이의 총 합이다.

각 라운드에서 찬이 부르는 M개 숫자가 주어지면, 훈이 Diff 를 최대화하는 M개 숫자들을 선택할 때, Diff 의 최대값을 출력하는 프로그램을 작성하시오.

Input

입력은 표준 입력을 사용한다. 첫 번째 줄에 두 정수 N과 M $(1 \le N \le 1{,}000,\ 1 \le M \le 1{,}000{,}000)$이 주어진다. 여기서, N은 훈과 찬이 게임에서 가지고 있는 행렬의 크기이고 M은 게임의 라운드 개수이다. 다음 이어지는 N개 줄의 i번째 줄에는 훈의 행렬의 i번째 행의 값들을 나타내는 N개 정수가 주어진다. 다음 이어지는 N개 줄의 i번째 줄에는 찬의 행렬의 i번째 행의 값들을 나타내는 N개 정수가 주어진다. 이 행렬들의 값은 0과 $1{,}000$ 사이의 정수이다. 다음 줄에는 찬이 각 라운드에서 부르는 1과 N 사이의 M개 정수가 주어진다.

Output

출력은 표준 출력을 사용한다. 훈이 Diff 를 최대화하는 M개 숫자를 선택할 때, Diff 의 최대값을 출력한다.

다음은 두 테스트 경우에 대한 입출력 예이다.

Sample Input 1

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

Output for the Sample Input 1

```
3
```

Sample Input 2

```
2 3
5 0
2 6
3 3
3 2
1 1 2
```

Output for the Sample Input 2

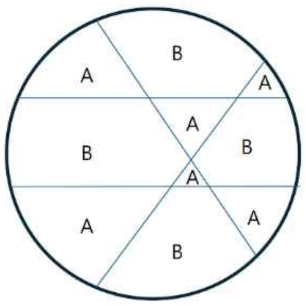
```
8
```

Problem F
Mining Rights
Time Limit: 1 Second

There is a circular piece of land where very valuable resources are buried at two distinct locations. Due to the immense value of these resources, many companies competed to obtain the mining rights, and two companies were selected through a rigorous review process. Your company has been chosen as one of the final two companies granted the mining rights. To avoid favoritism, the government devised a fair method for allocating the subareas of the land to the companies.

The method for dividing the land is as follows:

- **Rule 1 (Division by Company A)**: Company **A** is allowed to divide the circular land into smaller areas using one or more line segments. (The number of line segments is less than $10,000$). Each line segment starts from one point on the circumference and ends at another point on the circumference.



- **Rule 2 (No Cross-Border Right)**: Adjacent areas sharing a straight boundary must be developed by different companies.

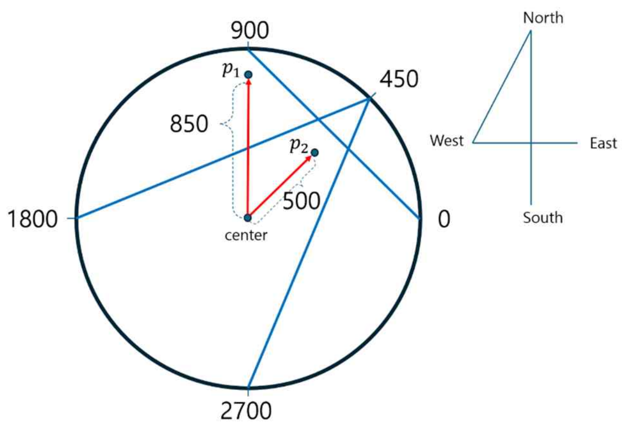
- **Rule 3 (Acceptance and Selection by Company B)**: The other company, Company **B**, has the right to either accept or reject the division proposed by Company **A**. If Company **B** accepts, it also has the right to choose one of the areas. All the remaining areas will be automatically assigned to the two companies based on Rule 2.

You are already aware of two locations, p_1 and p_2, where valuable resources are buried. Your goal is to ensure that both locations are included within your granted areas. To achieve this, as Company **B**, you will decide whether to accept or reject the division proposed by Company **A.**

Given a division proposed by Company **A** that is a set of line segments and the locations of the buried resources, write a program to decide whether you, company **B**, accepts the division or not.

Input
Your program is to read from standard input. The input starts with a line containing an integer number n, representing the number of line segments that divide the circle. The integer n is less than $10,000$. In the following n lines, each line describes one line segment. Each line contains two integers representing the indices of two points on the circumference of the circle, which is divided into 3,600 equidistant units. The point directly east of the center of the circle is indexed as 0, and the indices increase counterclockwise, ranging from 0 to 3,599. No identical line segments are given in the input.

After the n lines representing the dividing line segments, two additional lines provide information about two distinct locations p_1 and p_2 where the resources are buried. Each location is represented by two integers. The first integer indicates the direction, represented by the index of the point where a ray from the center, passing through the buried location, meets the circumference. The second integer represents the distance from the center to the buried resource. The distance is measured by dividing the radius of the circle into $1{,}000$ equidistant units, so the center has a distance of 0 , and a point on the circumference has a distance of $1{,}000$. It is guaranteed that the locations are within the circle and no locations lie on any of the line segments.



Output

Your program is to write to standard output. Print exactly one line. The line should be "`YES`" if company **B** can obtain both resource locations p_1 and p_2. If not, the line should be "`NO`".

The following shows input and output for two test cases.

Sample Input 1	Output for the Sample Input 1

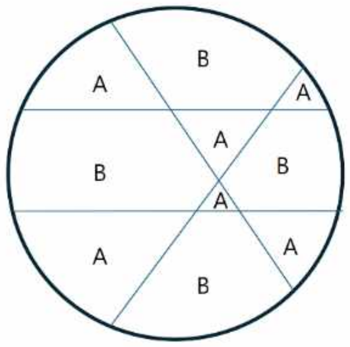
3
450 1800
900 0
450 2700
900 850
450 500
	NO

Sample Input 2	Output for the Sample Input 2

4
450 1800
0 1350
1350 2700
2700 0
450 500
3150 950
	YES

Problem F
채굴권 분할
제한 시간: 1 초

어느 원형의 땅 안에는 서로 다른 두 위치에 매우 귀중한 자원이 매장되어 있다. 이 자원들의 엄청난 가치 때문에 많은 회사가 채굴권을 얻기 위해 경쟁했지만, 엄격한 심사 과정을 통해 두 개의 기업만 선정되었다. 여러분의 회사는 최종적으로 채굴권을 얻은 두 회사에 포함되었다. 특혜 시비를 피하기 위해 정부는 두 회사에게 이 땅을 배분하는 공정한 방법을 고안했는데, 그 방법은 다음과 같다:



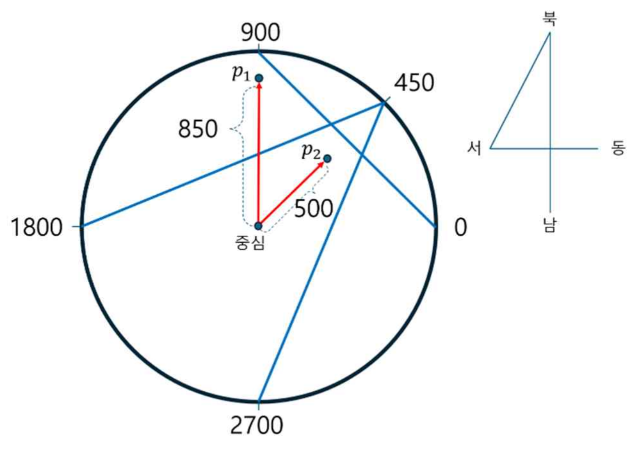
- **규칙 1 (A 사의 분할)**: 한 회사 **A** 는 10,000 개보다 적은 수의 선분을 사용하여 원형의 토지를 더 작은 영역으로 나눌 수 있다. 각 선분은 원의 둘레 위에 있는 한 점에서 시작하여 원의 둘레 위에 있는 다른 한 점에서 끝난다.
- **규칙 2 (경계 침범 금지)**: 직선 경계를 공유하며 맞닿아 있는 영역은 서로 다른 회사가 개발해야 한다.
- **규칙 3 (B 사의 수락과 선택권)**: **B** 회사는 **A** 회사가 제안한 분할을 수락하거나 거부할 권리가 있다. **B** 회사가 수락할 경우, **B** 회사는 분할된 영역 중 하나를 선택할 권리가 있다. 그 외의 나머지 영역들은 **규칙 2** 에 따라 두 회사에 자동으로 할당된다.

당신은 자원이 매장된 두 위치 p_1과 p_2를 이미 알고 있다. 당신은 **B** 회사를 대표해 위치 모두가 **B** 회사에 할당된 영역 안에 놓일 수 있도록 **A** 회사의 분할을 수락할지 거부할지를 결정해야 한다.

선분의 집합으로 표현되는 회사 **A** 의 분할 제안과 자원이 묻혀 있는 두 위치가 주어질 때, 회사 **B** 가 이를 수락해야 하는지 그렇지 않은지를 결정하는 프로그램을 작성하라.

Input
입력은 표준 입력을 사용한다. 첫 번째 줄에 원을 분할하는 선분의 개수를 나타내는 10,000보다 작은 정수 n이 주어진다. 그 다음 n개 줄에 n개 선분에 대한 정보가 주어진다. 각 줄은 하나의 선분이 갖는

두 끝점을 나타내는 두 개의 정수 인덱스로 이루어진다. 원의 둘레는 3,600 개의 단위로 나누어지며, 원의 중심에서 정확히 동쪽에 있는 원 둘레 위의 점이 인덱스 0 을 갖고, 이 인덱스는 반시계 방향으로 증가하여 3,599 까지의 값을 가질 수 있다. 여기서 중복되는 선분은 존재하지 않는다. 선분들을 표현하는 n 개의 줄이 끝나면, 자원이 매장된 서로 다른 두 위치 p_1과 p_2의 정보를 담은 두 줄이 추가로 주어진다. 각 위치는 두 정수로 표시된다. 첫 번째 정수는 방향을 나타내며, 중심에서 출발하여 매장된 위치를 통과하는 반직선이 원의 둘레와 만나는 지점의 인덱스로 표현된다. 두 번째 정수는 중심에서 매장된 자원까지의 거리를 나타낸다. 거리는 원의 반지름을 1,000 개의 단위로 나누어 측정되며, 중심의 거리는 0이고 원의 둘레에 있는 점들의 거리는 1,000이다. 매장된 자원은 원 안에 있으며, 선분 위에 놓이는 일도 없다는 것이 보장된다.



Output

출력은 표준 출력을 사용한다. 출력은 정확히 한 줄로 이루어진다. **B** 회사가 두 자원의 위치 p_1과 p_2 모두를 획득할 수 있다면 “`YES`”, 그렇지 않으면 “`NO`”를 출력한다.

다음은 두 가지 테스트 케이스에 대한 입력과 출력을 보여준다.

Sample Input 1	**Output for the Sample Input 1**

3
450 1800
900 0
450 2700
900 850
450 500
	NO

Sample Input 2	**Output for the Sample Input 2**

4
450 1800
0 1350
1350 2700
2700 0
450 500
3150 950
	YES

Problem G
New Megacity
Time Limit: 2.5 Seconds

You are involved in a huge project to design a new megacity connecting n cities using a given set of m potential roads, each with an associated cost. All cities should be connected with the minimum total cost, that is, they form a Minimum Spanning Tree (MST). However, not all roads are equally important. Your job is to determine the importance of each road on the following categories:

- **Type 1:** The road is included in **every possible MST** that connects all cities. This means that this road is essential for the optimal solution.
- **Type 2:** The road appears in **at least one MST** but **does not in all**.
- **Type 3:** The road is **never used in any MST**; it does not contribute to the least costly connection of all cities.

Given a road network of n cities with m roads with associated costs, write a program to output the type of each road.

Input

Your program is to read from standard input. The input starts with a line containing two integers n and m, where n is the number of cities $(2 \le n \le 100{,}000)$ and m is the number of potential roads $(n-1 \le m \le \min(100{,}000, n(n-1)/2))$.

In the following m lines, the i-th road is given by three integers x, y, z where x and y are the cities connected by the road $(1 \le x, y \le n, x \ne y)$, and z is the cost of building that road $(1 \le z \le 100{,}000)$. Each pair of cities is connected by at most one road, that is, there are no multiple edges between the same pair of cities.

It is guaranteed that at least one MST always exists for the given input.

Output

Your program is to write to standard output. Print m lines. The i-th line should contain a single integer representing the type of the i-th road (`1`, `2`, or `3`), in the same order as the input.

The following shows sample input and output for two test cases.

Sample Input 1

```
4 6
1 2 3
1 4 4
2 4 6
2 3 2
4 3 5
3 1 3
```

Output for the Sample Input 1

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

Sample Input 2

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

Output for the Sample Input 2

```
2
2
2
2
2
```

Problem H
Number Allocation
Time Limit: 0.1 Seconds

0	A	B	C	D
E	a	b	c	d
F	e	f	g	h
G	i	j	k	0
H	l	m	0	0

A grid consisting of 25 squares in a 5×5 size is given, as shown in the figure above. For convenience, let's assign the coordinates $(1,1)$ to the top-left square and $(5,5)$ to the bottom-right square. Initially, the squares at $(1,1)$, $(4,5)$, $(5,4)$, and $(5,5)$ are filled with 0, and certain numbers are pre-filled in all squares of the first row and the first column. That is, A to H are the pre-filled numbers. Now, the remaining empty squares must be filled with different numbers from 1 to 13. In other words, a to m are different numbers from 1 to 13. At this point, the following rules must be observed between the numbers.

- $A = a + e + i + l$
- $B = b + f + j + m$
- $C = c + g + k$
- $D = d + h$
- $E = a + b + c + d$
- $F = e + f + g + h$
- $G = i + j + k$
- $H = l + m$

That is, except for $(1,1)$, the value of each square in the first row is the sum of the values in the same column, and the value of each square in the first column is the sum of the values in the same row.

For example, consider the values A to H as shown in the right figure. If we assign the numbers $1, 2, 9, 4, 13, 10, 6, 5, 8, 7, 12, 11$, and 3 to variables a to m in that order, the rules mentioned above will be satisfied. However, consider the case where the values of A to H are $3, 22, 27, 9, 16, 34, 27$, and 14 in that order. In this scenario, there is no way to assign numbers that satisfy the rules mentioned above.

0	33	22	27	9
16	1	2	9	4
34	13	10	6	5
27	8	7	12	0
14	11	3	0	0

Given eight numbers of A to H, write a program to output the number of possible ways to assign different numbers from 1 to 13 to a through m in such a way that the above rules are satisfied.

Input
Your program is to read from standard input. The values of integers A, B, C, D, E, F, G, and H are given in order on a single line. Each integer is between 3 and 46 inclusive.

Output
Your program is to write to standard output. Print exactly one line. The line should contain the total number of all possible ways to assign 13 different integers from 1 to 13 to a through m.

The following shows sample input and output for two test cases.

Sample Input 1

```
33 22 27 9 16 34 27 14
```

Output for the Sample Input 1

```
126
```

Sample Input 2

```
3 22 27 9 16 34 27 14
```

Output for the Sample Input 2

```
0
```

Problem H

숫자 할당

제한 시간: 0.1 초

0	A	B	C	D
E	a	b	c	d
F	e	f	g	h
G	i	j	k	0
H	l	m	0	0

위 그림과 같이 5×5 크기의 총 25칸으로 이루어진 격자판이 주어져 있다. 편의상 맨 왼쪽 위 칸의 좌표를 (1,1)이라 하고 맨 오른쪽 아래 칸의 좌표를 (5,5)라 하자. 초기에 (1,1), (4,5), (5,4), (5,5)에는 0 이 채워져 있고, 1행의 모든 칸과 1열의 모든 칸은 미리 어떤 숫자(정수)들이 채워져 있다. 즉, A~H는 미리 채워진 숫자들이다. 이제 나머지 빈 칸들에 1부터 13까지의 서로 다른 수들을 채워야 한다. 즉, a~m은 1부터 13까지의 서로 다른 숫자이다. 이때, 각 숫자들 사이에는 다음과 같은 규칙이 성립해야 한다.

- $A = a + e + i + l$
- $B = b + f + j + m$
- $C = c + g + k$
- $D = d + h$
- $E = a + b + c + d$
- $F = e + f + g + h$
- $G = i + j + k$
- $H = l + m$

즉, (1,1)을 제외하고, 첫 번째 행의 각 칸에 있는 수들은 같은 열에 있는 수들의 합이며, 첫 번째 열의 각 칸에 있는 수들은 같은 행에 있는 수들의 합이다.

예를 들어, A부터 H까지의 숫자가 오른쪽 그림과 같이 주어졌다고 하자. 만약 a부터 m까지의 숫자를 그림과 같이 할당하면 위의 규칙이 만족된다. 하지만 만약 A부터 H까지의 숫자가 3, 22, 27, 9, 16, 34, 27, 14로 주어졌다면 위의 규칙을 만족하도록 a~m에 할당할 수 있는 방법이 없다.

0	33	22	27	9
16	1	2	9	4
34	13	10	6	5
27	8	7	12	0
14	11	3	0	0

A부터 H까지 8개의 숫자가 주어졌을 때, 위의 규칙이 만족되도록 1부터 13까지의 서로 다른 숫자를 a~m에 할당하는 모든 경우의 수를 출력하는 프로그램을 작성하시오.

Input
입력은 표준입력을 사용한다. 정수 A, B, C, D, E, F, G, H의 값이 한 줄에 차례로 주어진다. 각 정수는 3 이상 46이하이다.

Output
출력은 표준출력을 사용한다. 1부터 13까지의 서로 다른 13개의 정수를 a~m에 할당할 수 있는 모든 경우의 수를 한 줄에 출력한다.

다음은 두 테스트 케이스에 대한 입출력 예이다.

Sample Input 1	Output for the Sample Input 1
`33 22 27 9 16 34 27 14`	`126`

Sample Input 2	Output for the Sample Input 2
`3 22 27 9 16 34 27 14`	`0`

Problem I
Polygon Discovery
Time Limit: 4 Seconds

You are a member of the research team **CPCI** (Convex Polygon Computational Investigation), tasked with uncovering the properties of an unknown simple polygon P located on a two-dimensional coordinate plane. The exact coordinates of the polygon P are unknown, but you can use a special equipment that allows you to query the number of intersections between a given query line and P.

The polygon P is known to be a **convex** polygon, meaning every line segment between two points inside the polygon is fully contained within its interior. Additionally, P is **non-degenerate**, ensuring that its area is strictly greater than zero. The polygon also contains the origin $(0, 0)$ strictly within its interior. All vertices of P have integer coordinates, with each coordinate lying within the range $[-1{,}024, 1{,}023]$ inclusive.

Write a program to compute the area of the polygon P by invoking a series of queries.

Query
You can query with any straight line l on the coordinate plane by selecting two distinct integer points (x_1, y_1) and (x_2, y_2) on the line l. For each query, you will receive the number of intersection points between the line l and the boundary of the polygon P.

You are allowed to request at most $4{,}096$ queries. Using the intersection counts from these queries, you need to determine the exact area of the polygon P.

Interaction
This is an **interactive problem**. Your submitted program will interact with an **interactor** inside the grading server, which reads input from and writes output to your program.

The interaction proceeds in rounds. In each round, your program must first write a line containing one of two types of requests as follows:

1. A query: "`?` x_1 y_1 x_2 y_2"
   - This query request starts with a character '`?`', followed by four integers x_1, y_1, x_2, and y_2.
   - This asks for the number of intersection points between the straight line passing through the two distinct points (x_1, y_1) and (x_2, y_2) and the polygon P.
   - The values x_1, y_1, x_2, and y_2 must be integers in the range $[-10^9, 10^9]$ inclusive.
2. A final output: "`!` *area* "
   - This output request starts with a character '`!`', followed by a real number *area* .
   - This signals that your program has determined the area of the polygon P.

After the request is asked, an input line is followed, containing the response of the interactor to your request, and the response is available on standard input. If your request is a query, the response will be an integer representing the number of intersection points. If there are infinitely many intersection points, the response will be "`infinity`". If your request is a final output, the response will be "`correct`" if *area* is within an absolute error of 10^{-3} of the actual area. Otherwise, the response will be "`wrong`".

If your program violates the interaction protocol (e.g., issues a malformed request or makes more than $4{,}096$ queries), the interactor will immediately terminate the interaction. Also, your program must request exactly one final output, and this must be the last request. After making the output request, your program must terminate gracefully. **Do not forget to flush the output** after each request.

The polygon P is fixed throughout the interaction; the interactor is **not adaptive**.

The time and memory used by the interactor is also included in the calculation of your program's execution time and memory usage. You can assume that the maximum time used by the interactor is 1 second and the maximum amount of memory is 4 MiB.

Sample Interaction 1

Read (Interactor's Response)	**Write (Your Request)**
	`? 0 0 1 1`
`2`	
	`? -3 -1 0 2`
`0`	
	`? 0 -4 4 0`
`1`	
	`? 4 3 7 5`
`infinity`	
	`! 5.5`
`correct`	

In this example, the coordinates of the polygon P were $(1, 1)$, $(-2, -1)$, and $(2, -2)$.

To flush, you need to do the following right after writing a request and a line break:

- `fflush(stdout)` in C;
- `std::cout << std::flush` in C++;
- `System.out.flush()` in Java;
- `sys.stdout.flush()` in Python.

A testing tool is provided to help you develop your solution, so it is not mandatory to use it. If you want to use it, you can download the attachment `testing_tool.py` from the *DOMjudge Problemset* page. You can run `python3 testing_tool.py -f input.in ./sol` to see how your program interacts for a specific polygon P. **This testing tool can be used regardless of the language of your program as follows:** the same explanation is also found in the comments of `testing_tool.py`.

Usage: `python3 testing_tool.py -f inputfile <program>`

Use the `-f` parameter to specify the input file, e.g. `input.in`.

Format of the input file: The first line contains an integer n (≥ 3) that is the number of vertices of the polygon P. Each of the next n lines contains two integers x_i and y_i that are x- and y-coordinates of the i-th vertex of P, respectively. The coordinates of the vertices of P must be given in counterclockwise order.

Example: A triangle $(n = 3)$ with vertices $(1, 1), (-2, -1), (2, -2)$ given in counterclockwise order.

```
3
1 1
-2 -1
2 -2
```

If you have a `C++` solution stored in a file called "`sol.cpp`", you must first compile using "`g++ sol.cpp -o sol`" and then invoke the testing tool with:

```
python3 testing_tool.py -f input.in ./sol
```

If you have a `Python` solution that you would run using "`pypy3 solution.py`", you could invoke the testing tool with:

```
python3 testing_tool.py -f input.in pypy3 solution.py
```

If you have a `Java` solution that you would run using "`java MyClass`", you could invoke the testing tool with:

```
python3 testing_tool.py -f input.in java MyClass
```

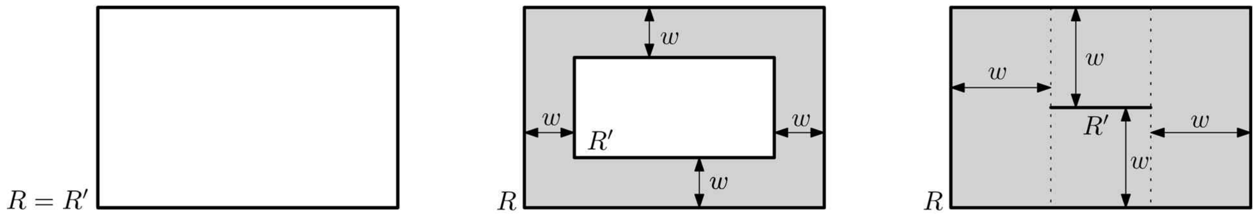
The tool is provided as-is, and you should feel free to make whatever alterations or augmentations you like to it. **Note that it is not guaranteed that a program that passes the testing tool will be accepted**.

Problem J
Two Rings
Time Limit: 2 Seconds

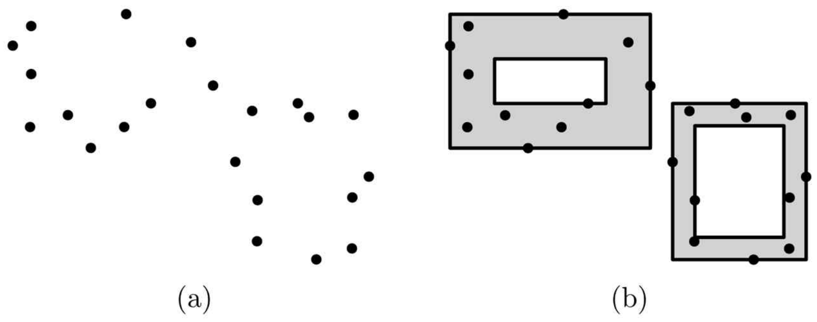
As a planar shape, a ring can be described as the area *between* two concentric circles. This concept can of course be generalized to any possible shapes. For example, one might define rectangular rings to be the area between two rectangles. A precise definition of rectangular rings is as follows: Here, any rectangles we discuss is assumed to be axis-aligned, so the four sides of a rectangle are horizontal or vertical, and are distinguished as the left, right, top, and bottom sides. Any vertical or horizontal segment is also considered a rectangle with empty interior. A *rectangular ring* is the closed area between two rectangles R and R', including the boundary, such that the following conditions are satisfied:

1. R' is contained in R, including the boundary of R.
2. The following four values are equal: the distance between the top side of R and the top side of R', the distance between the left side of R and the left side of R', the distance between the bottom side of R and the bottom side of R', and the distance between the right side of R and the right side of R'.

The distance described in the second condition is called the *width* of the rectangular ring defined by two rectangles R and R'. The figure below illustrates three rectangular rings of width w defined by two rectangles R and R'. Note that the first and third ones show two extreme and degenerate cases: $R' = R$ (thus, $w = 0$ and the rectangular ring is the boundary of $R' = R$) and R' is a line segment (thus, R is the rectangular ring).



Given a finite set P of points in the plane, write a program that finds two *non-penetrating* rectangular rings A_1 and A_2 such that $P \subset A_1 \cup A_2$ and the larger of their widths is minimized. Two rectangular rings are called non-penetrating if one's boundary neither intersects the other's interior nor crosses the other's boundary. Note that two non-penetrating rectangular rings still may touch in their boundaries in such a way that every point in the intersection between their boundaries lies in the intersection between two parallel sides of their defining rectangles or a corner.



The above figure shows (a) an example set P of 22 points and (b) an optimal pair of two non-penetrating rectangular rings containing P that minimizes the larger value of their widths.

Input

Your program is to read from standard input. The input starts with a line containing an integer, n ($1 \leq n \leq 300{,}000$), where n is the number of input points in P. In each of the following n lines, there are two integers between -10^9 and 10^9, separated by a space, that describe the coordinates of a point in P. You may assume that no two input points are identical.

Output

Your program is to write to standard output. Print exactly one line. The line should contain an integer that describes the minimum possible value for the larger width of two non-penetrating rectangular rings that include all points of P.

The following shows sample input and output for two test cases.

Sample Input 1

```
13
0 1
1 2
0 5
4 6
5 0
6 2
8 3
11 1
12 -1
13 2
12 4
10 3
3 4
```

Output for the Sample Input 1

```
1
```

Sample Input 2

```
13
0 1
1 2
0 5
4 6
5 0
6 2
8 3
11 -2
12 -4
13 -1
12 1
10 0
3 4
```

Output for the Sample Input 2

```
2
```

Problem K
WEB Machine
Time Limit: 1 Second

Tim has a machine for sorting balls, namely *WEB machine*. The WEB machine has a wheel with n slots. Each slot may have a white ball (`W`), a blue ball (`B`), or it can be empty (`E`). Over a certain slot, the machine has a head to identify the status of the slot. The head can determine the color of the ball in the slot. It can also pick the ball in the slot or drop the ball into the slot. The head, however, can hold at most one ball at a time. The WEB machine can rotate the wheel of slots clockwise or counterclockwise. Fig. 1 shows an example of WEB machine.

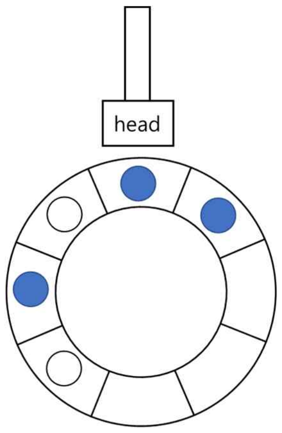


Fig 1. An example of a WEB machine

The WEB machine operates according to the control instructions, namely *WEB instructions*, and one can write a program as a sequence of the WEB instructions. The set of WEB instructions and their meaning is defined as follows:

- `Pick`: picks up and holds the ball in the current slot under the head,
- `Drop`: drops the ball of the head to the current slot,
- `Left`: rotates the wheel clockwise by a slot,
- `Right`: rotates the wheel counterclockwise by a slot,
- `LeftStar` C: rotates the wheel clockwise while condition C holds and returns the number of slots rotated,
- `RightStar` C: rotates the wheel counterclockwise while condition C holds and returns the number of slots rotated,
- `Jump` n: jumps to the next nth instruction (If n is 2, the next instruction is skipped and the next of the next instruction is executed),
- `Jump` n `if` C: jumps to the next nth instruction if condition C holds and do nothing otherwise,
- $X = E$: evaluates the expression E and stores it to variable X, and
- `Stop`: stops the machine.

Executing `Pick` or `Drop`, the machine does nothing for improper conditions: the slot is empty while executing `Pick` or the slot is not empty while executing `Drop`. The variable X should be a capital letter other than `W`, `E`, and `B`. The condition C in `LeftStar` C, `RightStar` C, or `Jump` n `if` C is either one of `W`, `E`, `B`, and X (a variable) or one of the negated forms (`!W`, `!E`, `!B`, and `!`X). The condition `W` implies that the ball in the current slot under the head is white, `B` does it is blue, and `E` does the slot is empty. The condition X holds if X is nonzero ($X \neq 0$). The negated condition holds if the unattributed condition does not hold, say `!E` is true when `E` does not hold. For instance, when executing `RightStar !E`, the head searches for an empty slot by rotating the wheel counterclockwise. As a result, there will be an empty slot under the head if the machine has at least one empty slot. The instruction incurs an infinite loop if the machine has no empty slot.

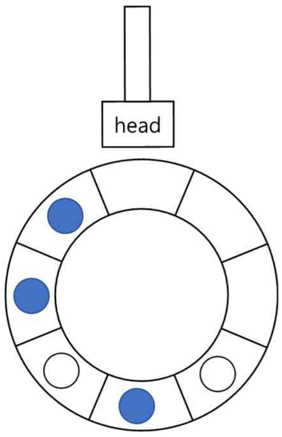


Fig 2. After RightStar B is executed

The number n in unconditional jump (`Jump` n) and conditional jump (`Jump` n `if` C) can be a negative integer. For instance, `Jump -1` branches to the previous instruction. Beware that you should not execute `Jump 0` which incurs an infinite loop.

The assignment instruction $X = E$ evaluates the integral expression E and stores the value in variable X. The expression's value is in the range between -200 and $+200$ inclusive ($|E| \leq 200$). The expression can use any variable that has been defined before. For example, `K = K + 1` will increase the variable `K` by one if `K` is defined before; it is an error, otherwise. The expression can be a star instruction, either `LeftStar` or `RightStar`; the assignment will store the number of slots rotated in the target variable. For example, executing the following assignment:

```
X = RightStar B
```

on the machine of the state in Fig. 1 will make the state in Fig. 2, and the value of variable `X` will be two.

As another example, the following code will restore the state in Fig. 1 from the state in Fig. 2:

```
LeftStar !W
Right
Pick
Drop
Stop
```

where the third and fourth instructions are listed to demonstrate the relationship between them; `Pick` and `Drop` are the inverse operations of one another.

Note that the expression cannot be nested, i.e. it can include at most one binary arithmetic operator (either + or $-$) or the star instruction. From the arithmetic operators, only the addition (+) and subtraction ($-$) operators are allowed; the multiplication, division, and modulus operators are invalid in the WEB expressions. The assignment and arithmetic operators should be separated from their operands by space. A space should not follow the sign symbol for an integer literal, say "`-12`" is valid but "`- 12`" is not.

Tim wants to sort the balls in the WEB machine using a WEB program, a sequence of WEB instructions ending with `Stop`. Initially, the balls are mixed in any order though they are grouped in the wheel. Tim wants the balls to be sorted into a group of white balls and blue ones separated by a group of empty slots reading clockwise starting from the head. Though the groups of balls are separated by a group of empty slots clockwise, they are not separated counterclockwise since they are in the wheel of the machine. Let's help Tim by writing a sequence of WEB instructions to make the balls in the wheel sorted in `W`, `E`, and `B` order reading clockwise starting from the head.

The WEB program consists of several lines, each of which contains a single WEB instruction. Therefore, the above WEB programs are valid, but the following one is invalid, i.e. a syntax error:

```
RightStar E X
= 10 + 2
```

since the assignment is spanned over two lines; the first instruction has an extra variable at the end and the second instruction is missing the target variable of the assignment.

Make a WEB program to sort the balls as Tim wanted and submit the WEB program. Your WEB program should convert the initial configuration of a WEB machine into the final one with the balls sorted by color and must end with `Stop`. The initial configuration is given as input and the final one is given as output. In the initial configuration, assume that the wheel of the WEB machine has at least two empty slots. **Initially, all empty slots are grouped in a sequence over the slots**. Assume also that the machine has at least one white ball and one blue ball. In the final configuration, the white and blue balls should be separated by a sequence of empty slots with the head located over the leftmost white ball assuming the clockwise reading of the slots.

Initial configuration of the WEB machine

The initial configuration of the WEB machine is given from standard input. The first line of input contains a positive integer s ($5 \leq s \leq 100$), the number of slots of the machine. The second line contains a sequence of s characters indicating the initial state of the wheel. The character in the second line is one of `W`, `B`, and `E` indicating a white ball, a blue ball, and an empty slot, respectively. The characters are given in clockwise and separated by space. The first character indicates the slot under the head of the machine. The character `E`'s are given consecutively in groups.

Final configuration of the WEB machine

The final configuration of the machine is given from standard output. Your WEB program is to sort the balls in the initial WEB machine resulting in the final configuration. The final configuration is a sequence of s characters indicating the final state of the wheel. The balls should be sorted in the order with the white ball(s), the empty slots, and the blue ball(s) clockwise starting from the head.

The following shows sample initial and final configurations of the WEB machine for three test cases.

Sample Initial Configuration 1	**Final Configuration for the Sample Initial Configuration 1**

`5`
`B W B E E`
	`W E E B B`

Sample Initial Configuration 2	**Final Configuration for the Sample Initial Configuration 2**

`7`
`W B W E E E W`
	`W W W E E E B`

Sample Initial Configuration 3	**Final Configuration for the Sample Initial Configuration 3**

`8`
`W E E W B W B B`
	`W W W E E B B B`

Submit

You have to submit a WEB program, say `sort.web`, to operate on the WEB machine of a given initial configuration. The WEB program should convert the initial configuration to the final configuration as Tim wanted. You should select *WEB* as the submission language in the *DOMjudge* auto-judge system; otherwise, the system can reject your program with no penalty.

Simulator

For testing purposes, you are provided with a WEB machine simulator, which can read the WEB program file and the initial configuration of the WEB machine. The simulator is just for your convenience, so it is not mandatory to use it. If you want to use it, you can download the attachment `wm.py` from *DOMjudge Problemset* page. It can detect some kinds of errors, either syntax or run-time errors; it shows the final configuration resulting from the execution of the WEB program given an initial configuration; it counts the number of steps required to execute the WEB program — see the function σ described later. The final configuration is shown in the standard output and the errors are shown in the standard error: the syntax error is reported as `Compile Error` and the run-time error is reported as `Runtime Error`. The number of steps is also shown in the standard error. Though the simulator is useful for testing WEB programs, it may not provide complete information for evaluating your code. You just use it as a reference for testing a WEB program. Note that passing the simulator does not guarantee that the WEB program will be accepted by the auto-judge system.

The simulator assumes the WEB program file as the first argument in the command line and the initial configuration is read from the standard input. For example, the following command will execute the WEB program file named `sort.web` for the first sample input using the simulator `wm.py`:

```
> python wm.py sort.web
8
B B E E E W B W
```

where the first character in the first line (>) denotes the prompt. The simulator is executable on a Python interpreter of version greater than or equal to 3.9. You may test the following code for `sort.web` to get the configuration in Fig. 2:

```
X = RightStar B
Stop
```

The simulator will force to terminate the execution of the WEB program within a certain number of steps. The number of steps of instructions is calculated by the function σ, which is defined as follows:

- $\sigma(\texttt{Pick}) = 1$,
- $\sigma(\texttt{Drop}) = 1$,
- $\sigma(\texttt{Left}) = 1$,
- $\sigma(\texttt{Right}) = 1$,
- $\sigma(\texttt{LeftStar}\ C) = k$ where k is the number of slots rotated clockwise,
- $\sigma(\texttt{RightStar}\ C) = k$ where k is the number of slots rotated counterclockwise,
- $\sigma(\texttt{Jump}\ 0) = \infty$,
- $\sigma(\texttt{Jump}\ n) = 1$ for nonzero n,
- $\sigma(\texttt{Jump}\ 0\ \texttt{if}\ C) = \infty$ if C is true and 1 otherwise,
- $\sigma(\texttt{Jump}\ n\ \texttt{if}\ C) = 1$ for nonzero n,
- $\sigma(X = e_1 + e_2) = 1$,
- $\sigma(X = e_1 - e_2) = 1$,
- $\sigma(X = e) = 1 + \sigma(e)$ where $\sigma(e) = \sigma(I)$ if e is a star instruction I and $\sigma(e) = 0$ otherwise, and
- $\sigma(\texttt{Stop}) = 1$.

Note that the number of steps of the star instructions (`LeftStar` and `RightStar`) can vary depending on the condition. Also, note that the number of steps of the jump instructions for 0, say `Jump 0` or `Jump 0 if` C for any valid condition C, is infinite. A copying assignment, say `X = Y`, is covered by the case of $\sigma(X = e)$ where e is not a star instruction. Therefore, $\sigma(\texttt{X = Y}) = 1$.

Constraints

Your WEB program should complete the execution in 70,000 steps. If the number of steps exceeds the limit, 70,000 steps, the auto-judge and the simulator force to terminate the WEB program — it is regarded as time out.