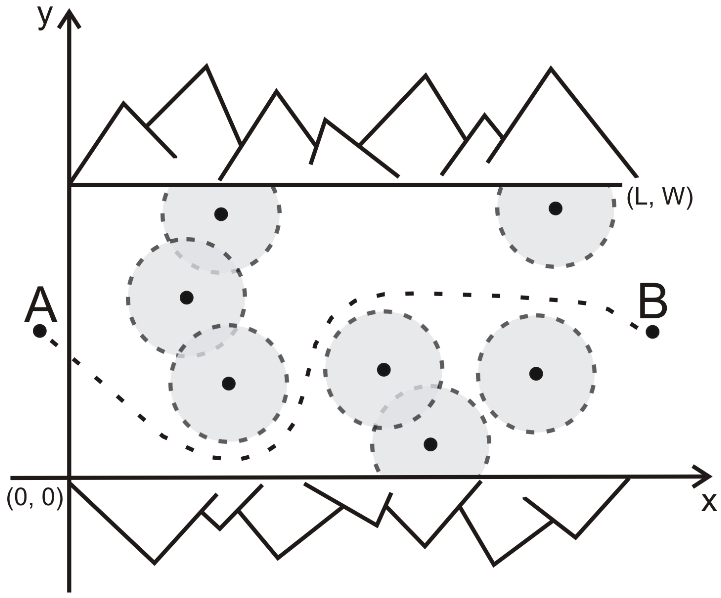
Escape

A group of war prisoners are trying to escape from a prison. They have thoroughly planned the escape from the prison itself, and after that they hope to find shelter in a nearby village. However, the village (marked as B, see picture below) and the prison (marked as A) are separated by a canyon which is also guarded by soldiers. These soldiers sit in their pickets and rarely walk; the range of view of each soldier is limited to exactly 100 meters. Thus, depending on the locations of soldiers, it may be possible to pass the canyon safely, keeping the distance to the closest soldier *strictly larger* than 100 meters at any moment.



You are to write a program which, given the width and the length of the canyon and the coordinates of every soldier in the canyon, and assuming that soldiers do not change their locations, first determines whether prisoners can pass the canyon unnoticed. If this is impossible then the prisoners (having seen enough violence) would like to know the minimum number of soldiers that have to be eliminated in order to pass the canyon safely. A soldier may be eliminated regardless of whether he is visible to any other soldier or not.

Input

The input is read from a text file named `escape.in`. The first line contains three integers L, W, and N – the length and the width of the canyon, and the number of soldiers, respectively. Each of the following N lines contains a pair of integers X_i and Y_i – the coordinates of i-th soldier in the canyon ($0 \le X_i \le L$, $0 \le Y_i \le W$). The coordinates are given in meters, relative to the canyon: the southwestern corner of the canyon has coordinates $(0, 0)$, and the northeastern corner of the canyon has coordinates (L, W), as seen in the picture above.

Note that passing the canyon may start at coordinate $(0, y_s)$ for any $0 \le y_s \le W$ and end at coordinate (L, y_e) for any $0 \le y_e \le W$. Neither y_s nor y_e need to be integer.

Output

The output is written into a text file named `escape.out`. In the first and only line of the output file the program should print the minimum number of soldiers that have to be eliminated in order for the prisoners to pass the canyon safely. If the prisoners can escape without any elimination, the program should print 0 (zero).

Example

`escape.in`	`escape.out`

`130 340 5`
`10 50`
`130 130`
`70 170`
`0 180`
`60 260`
	`1`

Constraints

$1 \le W \le 50{,}000 \quad 1 \le L \le 50{,}000 \quad 1 \le N \le 250$

Grading

Your program will receive partial credits if it can only determine whether the prisoners need to eliminate any guard at all in order to escape. For this, several test runs will be grouped to one test group. You will receive 30% of a test groups's credits in case you determine for each test run correctly whether any guards need to be eliminated (0 means no guards need to be eliminated, any integer > 0 means that any number of guards need to be eliminated). You will receive 100% of a test group's credits in case you determine for each test run correctly how many guards need to be eliminated for the prisoners' escape.

Ranklist Sorting

You are given the scores of several players in a competition. Your task is to create a ranklist of the players, sorted in decreasing order by score.

Unfortunately, the data structure used for the list of players supports only one operation, which moves a player from position i to position j without changing the relative order of other players. If $i > j$, the positions of players at positions between j and $i - 1$ increase by 1, otherwise if $i < j$ the positions of players at positions between $i + 1$ and j decrease by 1.

This operation takes i steps to locate the player to be moved, and j steps to locate the position where he or she is moved to, so the overall cost of moving a player from position i to position j is $i + j$. Here, positions are numbered starting with 1.

Determine a sequence of moves to create the ranklist such that the sum of the costs of the moves is minimized.

Input

The input is read from a text file named `sorting.in` . The first line contains n ($2 \leq n \leq 1000$), the number of players. Each of the following n lines contains one non-negative integer s_i ($0 \leq s_i \leq$ 1,000,000), the scores of the players in the current order. You may assume that all scores are distinct.

Output

The output is written into a text file named `sorting.out` . In the first line of the output print the number of moves used to create the ranklist. The following lines should specify the moves in the order in which they are applied. Each move should be described by a line containing two integers i and j, which means that the player at position i is moved to position j. The numbers i and j must be separated by a single space.

Example

sorting.in	sorting.out

5
20
30
5
15
10
	2
	2 1
	3 5

Grading

30% of the test cases have values of $n \leq 10$

The Sound of Silence

In digital recording, sound is described by a sequence of numbers representing the air pressure, measured at a rapid rate with a fixed time interval between successive measurements. Each value in the sequence is called a *sample*.

An important step in many voice-processing tasks is breaking the recorded sound into chunks of non-silence separated by silence. To avoid accidentally breaking the recording into too few or too many pieces, the silence is often defined as a sequence of m samples where the difference between the lowest and the highest value does not exceed a certain treshold c.

Write a program to detect silence in a given recording of n samples according to the given parameter values m and c.

Input

The input is read from a text file named `sound.in`.

The first line of the file contains three integers: n ($1 \leq n \leq 1{,}000{,}000$), the number of samples in the recording; m ($1 \leq m \leq 10{,}000$), the required length of the silence; and c ($0 \leq c \leq 10{,}000$), the maximal noise level allowed within silence.

The second line of the file contains n integers a_i ($0 \leq a_i \leq 1{,}000{,}000$ for $1 \leq i \leq n$), separated by single spaces: the samples in the recording.

Output

The output is written into a text file named `sound.out`.

The file should list all values of i such that $\max(a[i \dots i+m-1]) - \min(a[i \dots i+m-1]) \leq c$. The values should be listed in increasing order, each on a separate line.

If there is no silence in the input file, write `NONE` on the first and only line of the output file.

Example

sound.in	sound.out
7 2 0 0 1 1 2 3 2 2	2 6