Task 1. Game on two heaps

Two players are playing the following game:

They sit in front of two non-empty heaps of balls. For clarity, let us denote by *A* the less (or mostly equal) count of balls in the heaps. The other one is denoted by *B* (i.e., $A \le B$). The starting ratio $A : B = \alpha$ is important throughout the course of the concrete game and remains intact, no matter how the numbers in the heaps change. The players move consecutively, taking at least one ball from at least one of the heaps. The one of them who cannot make a move, or makes a wrong move, loses the game. So the one, who has played the last correct move, wins.

Let us denote by *x* the number of balls, taken from one of the heaps, and by *y* the number of balls, taken from the other heap. We assume denoted by *x* the less (or mostly equal to the other) count of taken balls. So we can state a simple rule for a move to be correct:

- **A move is correct only if $x : y \ne \alpha$, where $0 \le x \le y$** and **$y > 0$**;
- **There is, naturally, no way to take from a heap more balls than those left in it.**

Let us consider an example, where *A*=12, *B*=18. In this concrete game ***the wrong*** ratio of taken balls is $x:y = \alpha = 12:18 = 2:3$. Every other ratio of taken balls is correct. Here are some **wrong** (and therefore – losing) moves for the first player:

1. To take all balls, i. e. *x*=12, *y*=18. This move is obviously wrong, because $x:y = 12:18 = 2:3 = \alpha$.
2. To take three balls from heap *A*, leaving there 9, and two balls from heap *B*, leaving in it 16 balls. This move is wrong, because by it *x*=2, *y*=3 and $x:y = 2:3 = \alpha$. ATTENTION! After a move of this kind the ratio of the balls in the heaps would become 9:16, but this DOES NOT CHANGE the fixed ratio α, it remains 2:3, the same as before the start of this concrete game!
3. To take 8 balls from *A* and 12 from *B*, i. e. *x*=8, *y*=12, $x:y=8:12=2:3$. After this move the balls in *A* would become 4, and in *B* – 6. (The balls' ratio in the heaps after this move does not change – 4:6=2:3. But we emphasize again that the moves do not influence the important ratio α for this concrete game, which is defined before its first move and stays constant until its end.)

We can indicate many correct moves for the first player: to take one ball from any heap ($0:1 = 0 \ne \alpha$); to take a whole heap (for example – the second one, $0:18 = 0 \ne \alpha$); to take maximum balls (12) from each heap ($12:12 = 1 \ne \alpha$); to take 10 balls from *A* and 5 from *B* ($5:10 = 1:2 \ne \alpha$) etc. Of course, it is wrong if the player wants to take from a heap more balls than those left in it. And do not forget to take at least one ball from any heap. The "move" $x = y = 0$ (i.e., "I am leaving the situation intact.") is wrong (hence losing).

Design a program **arelgame**, which calculates the number of the winning first moves for the first player. A move is said to be "winning" if it leads to success, no matter how good or bad are the moves of the opponent.

Input. One line is read from the standard input, which contains only two space separated positive integers. These are the number of balls in the first, and the number of balls in the second heap, respectively.

Output. The program should send to the standard output one line, containing only one non-negative integer: the number of the winning first moves for the first player.

Constraints. The number of balls in each heap does not exceed 10^{18}.

In 30% of the test groups it is less than 2000.

Evaluation. The tests are packed in groups of three. The points for each group are given only if the three tests in the group are solved correctly.

Examples

Input	Output	*Explanation*
3 2	0	There is no winning move for the first player, while each correct move of his side leads to a direct loss – the second player can take everything left.
6 18	2	α = 6:18 = 1:3 The winning moves for the first player are as follows: - Taking 16 balls from the second heap. This move is correct, because 0:16≠1:3. After this move there are 6 balls remaining in the first, and 2 balls remaining in the second heap. The second player is not allowed to take all, because 2:6=1:3=α. So each correct move of his leads to a loss; - Taking 3 balls from the first and 17 from the second heap (3:17≠1:3, so the move is correct). There are now 3 balls left in the first and 1 ball left in the second heap and the second player has no winning move.

.

Task 2. Rain

Waterproof barriers of different heights are placed perpendicular to the length of a rectangular box. The distance between any two adjacent barriers is one centimeter. The box is without a lid and when there is enough rain on top, it is filled with as much water as possible. For some barriers, we can increase their heights by integer values no larger than the pre-specified ones. What is the minimum count of barriers at which we need to increase their heights, so that the maximum amount of water can be collected in the box? Write program **rain** to find the answer.

Input: The first line of the standard input contains the count *N* of the barriers. The second line contains, according to their location in the box from left to right, the heights in centimeters of the barriers. The next line contains the count *K* of barriers at which we can increase the heights. It follows as many lines as there are barriers at which we can increase the heights. Each of these lines contains a barrier number and a maximum allowed increase in height in centimeters. Barrier numbers start at zero.

Output: The program should print on the standard output two integers separated by exactly one space, equal respectively to the found minimum number of barriers at which we increase the heights and the maximum amount of water that the box can collect after increasing the heights of the barriers.

Remark: We calculate the amount of water in cubic centimeters because we assume the width of the box to be one centimeter. We consider the front and back walls of the box (the one that is to us and the one to the rear) to be higher than the height of each barrier, together with any possible increase in height of the barriers. The left and right walls of the box coincide with the left and right barriers respectively and with the barriers after a possible increase of the heights.

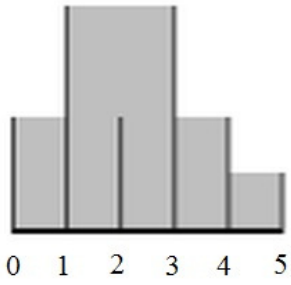
Constraints: $1 < N < 1\ 000\ 000$; $0 < K \leq N$; The initial height of each barrier is a positive integer less than 1 000 000; The maximum allowed increase in height of a barrier is a positive integer less than 1 000 000;

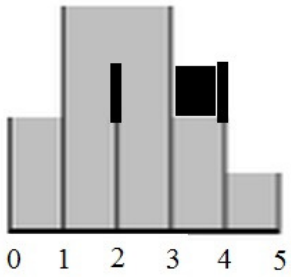
Evaluation. Each test is evaluated separately.

Example.

Input

```
6
2 4 2 4 2 1
2
2 1
4 1
```





Output

```
1 14
```

Explanation of the Example: We do not increase the height of the barrier at position 2 because this will not change the amount of water. We increase by 1 centimeter the barrier height at position 4. This increases the total amount of water by 1 cubic centimeter.

Task 3. Field

You are given a field which is a square table *N* x *N* filled with non-negative integers. Write a program **field** that calculates the minimum number *M*, such that all squares of size *M* x *M* that cover the whole number of cells in the field have their sum greater than or equal to *K*. The boundary of the square may lie on the boundary of the field.

Input. The first line of the standard input contains two integers separated by a space - *N* and *K*. The next *N* lines of the input contain *N* integers each, which represent the elements of the field.

Output. The program should send to the standard output one line, containing only one integer equal to the found minimum value of *M* or -1, if no such value exists.

Constraints.

$1 \le N \le 5000$
$0 \le$ each element in the field ≤ 50
$1 \le K \le 2 \cdot 10^9$

Evaluation.

Subtask	Points	*N*
1	5	≤ 20
2	10	≤ 100
3	20	≤ 500
4	35	≤ 2 000
5	30	≤ 5 000

Points for a given subtask are obtained only if all tests for it are correctly solved.

Example.

Input	Output	Explanation of the example
5 10 1 2 3 4 5 10 0 7 0 4 0 0 2 1 3 5 0 3 0 2 0 5 0 8 0	3	For *M*=1 the 1x1 square: 2 has a sum of just 2 < 10. For *M*=2 the 2x2 square: 0 7 0 2 has a sum of 9 < 10. For *M*=3 each 3x3 square has a sum ≥ 10. Thus *M*=3 is the minimal *M* such that each square of size *M* x *M* has a sum of at least 10.