A. Odds and Ends

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Where do odds begin, and where do they end? Where does hope emerge, and will they ever break?

Given an integer sequence $a_1, a_2, \ldots, a_n$ of length n. Decide whether it is possible to divide it into an odd number of non-empty subsegments, the each of which has an odd length and begins and ends with odd numbers.

A *subsegment* is a contiguous slice of the whole sequence. For example, $\{3, 4, 5\}$ and $\{1\}$ are subsegments of sequence $\{1, 2, 3, 4, 5, 6\}$, while $\{1, 2, 4\}$ and $\{7\}$ are not.

Input

The first line of input contains a non-negative integer n $(1 \le n \le 100)$ — the length of the sequence.

The second line contains n space-separated non-negative integers $a_1, a_2, \ldots, a_n$ $(0 \le a_i \le 100)$ — the elements of the sequence.

Output

Output "`Yes`" if it's possible to fulfill the requirements, and "`No`" otherwise.

You can output each letter in any case (upper or lower).

Examples

input
```
3
1 3 5
```
output
```
Yes
```

input
```
5
1 0 1 5 1
```
output
```
Yes
```

input
```
3
4 3 1
```
output
```
No
```

input
```
4
3 9 9 3
```
output
```
No
```

Note

In the first example, divide the sequence into 1 subsegment: $\{1, 3, 5\}$ and the requirements will be met.

In the second example, divide the sequence into 3 subsegments: $\{1, 0, 1\}$, $\{5\}$, $\{1\}$.

In the third example, one of the subsegments must start with 4 which is an even number, thus the requirements cannot be met.

In the fourth example, the sequence can be divided into 2 subsegments: $\{3, 9, 9\}$, $\{3\}$, but this is not a valid solution because 2 is an even number.

B. Tell Your World

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Connect the countless points with lines, till we reach the faraway yonder.

There are n points on a coordinate plane, the i-th of which being (i, y_i).

Determine whether it's possible to draw two parallel and non-overlapping lines, such that every point in the set lies on **exactly one** of them, and each of them passes through **at least one** point in the set.

Input

The first line of input contains a positive integer n $(3 \le n \le 1\ 000)$ — the number of points.

The second line contains n space-separated integers $y_1, y_2, \ldots, y_n$ $(-10^9 \le y_i \le 10^9)$ — the vertical coordinates of each point.

Output

Output "Yes" (without quotes) if it's possible to fulfill the requirements, and "No" otherwise.

You can print each letter in any case (upper or lower).

Examples

input

```
5
7 5 8 6 9
```

output

```
Yes
```

input

```
5
-1 -2 0 0 -5
```

output

```
No
```

input

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

output

```
No
```

input

```
5
1000000000 0 0 0 0
```

output

```
Yes
```

Note

In the first example, there are five points: $(1, 7)$, $(2, 5)$, $(3, 8)$, $(4, 6)$ and $(5, 9)$. It's possible to draw a line that passes through points $1, 3, 5$, and another one that passes through points $2, 4$ and is parallel to the first one.

In the second example, while it's possible to draw two lines that cover all points, they cannot be made parallel.

In the third example, it's impossible to satisfy both requirements at the same time.

C. From Y to Y

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

From beginning till end, this message has been waiting to be conveyed.

For a given unordered multiset of n lowercase English letters ("multi" means that a letter may appear more than once), we treat all letters as strings of length 1, and repeat the following operation $n - 1$ times:

- Remove any two elements s and t from the set, and add their concatenation $s + t$ to the set.

The cost of such operation is defined to be $\sum_{c \in \{\text{'a'}, \text{'b'}, \ldots, \text{'z'}\}} f(s, c) \cdot f(t, c)$, where $f(s, c)$ denotes the number of times character c appears in string s.

Given a non-negative integer k, construct any valid non-empty set of no more than $100\ 000$ letters, such that the minimum accumulative cost of the whole process is **exactly** k. It can be shown that a solution always exists.

Input

The first and only line of input contains a non-negative integer k $(0 \le k \le 100\ 000)$ — the required minimum cost.

Output

Output a non-empty string of no more than $100\ 000$ lowercase English letters — any multiset satisfying the requirements, concatenated to be a string.

Note that the printed string doesn't need to be the final concatenated string. It only needs to represent an unordered multiset of letters.

Examples

input
```
12
```
output
```
abababab
```

input
```
3
```
output
```
codeforces
```

Note

For the multiset {'a', 'b', 'a', 'b', 'a', 'b', 'a', 'b'}, one of the ways to complete the process is as follows:

- {"ab", "a", "b", "a", "b", "a", "b"}, with a cost of 0;
- {"aba", "b", "a", "b", "a", "b"}, with a cost of 1;
- {"abab", "a", "b", "a", "b"}, with a cost of 1;
- {"abab", "ab", "a", "b"}, with a cost of 0;
- {"abab", "aba", "b"}, with a cost of 1;
- {"abab", "abab"}, with a cost of 1;
- {"abababab"}, with a cost of 8.

The total cost is 12, and it can be proved to be the minimum cost of the process.

D. Rooter's Song

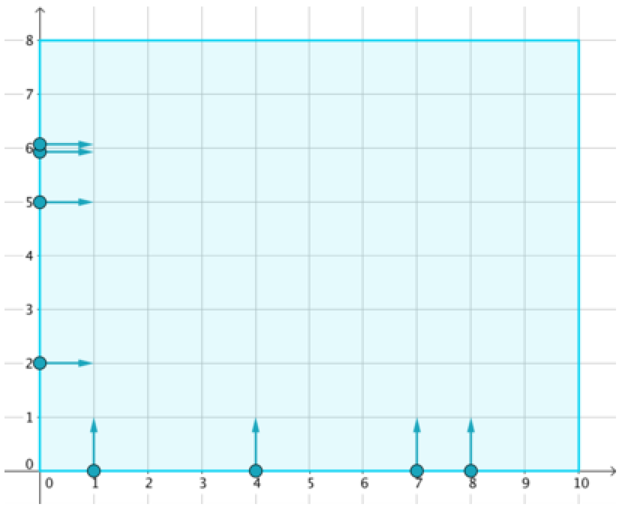
time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Wherever the destination is, whoever we meet, let's render this song together.

On a Cartesian coordinate plane lies a rectangular stage of size $w \times h$, represented by a rectangle with corners $(0, 0)$, $(w, 0)$, (w, h) and $(0, h)$. It can be seen that no collisions will happen before one enters the stage.

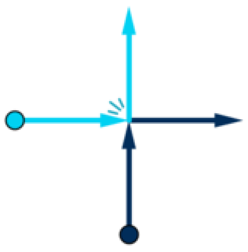
On the sides of the stage stand n dancers. The i-th of them falls into one of the following groups:

- **Vertical**: stands at $(x_i, 0)$, moves in positive y direction (upwards);
- **Horizontal**: stands at $(0, y_i)$, moves in positive x direction (rightwards).



According to choreography, the i-th dancer should stand still for the first t_i milliseconds, and then start moving in the specified direction at 1 unit per millisecond, until another border is reached. It is guaranteed that no two dancers have the same group, position and waiting time at the same time.

When two dancers collide (i.e. are on the same point at some time when both of them are moving), they immediately exchange their moving directions and go on.



Dancers stop when a border of the stage is reached. Find out every dancer's stopping position.

Input

The first line of input contains three space-separated positive integers n, w and h $(1 \le n \le 100\,000,\ 2 \le w, h \le 100\,000)$ — the number of dancers and the width and height of the stage, respectively.

The following n lines each describes a dancer: the i-th among them contains three space-separated integers g_i, p_i, and t_i $(1 \le g_i \le 2,\ 1 \le p_i \le 99\,999,\ 0 \le t_i \le 100\,000)$, describing a dancer's group g_i ($g_i = 1$ — vertical, $g_i = 2$ — horizontal), position, and waiting time. If $g_i = 1$ then $p_i = x_i$; otherwise $p_i = y_i$. It's guaranteed that $1 \le x_i \le w - 1$ and $1 \le y_i \le h - 1$. It is guaranteed that no two dancers have the same group, position and waiting time at the same time.

Output

Output n lines, the i-th of which contains two space-separated integers (x_i, y_i) — the stopping position of the i-th dancer in the input.

Examples

input

```
8 10 8
1 1 10
1 4 13
1 7 1
1 8 2
2 2 0
2 5 14
2 6 0
2 6 1
```

output

```
4 8
10 5
8 8
10 6
10 2
1 8
7 8
10 6
```

input

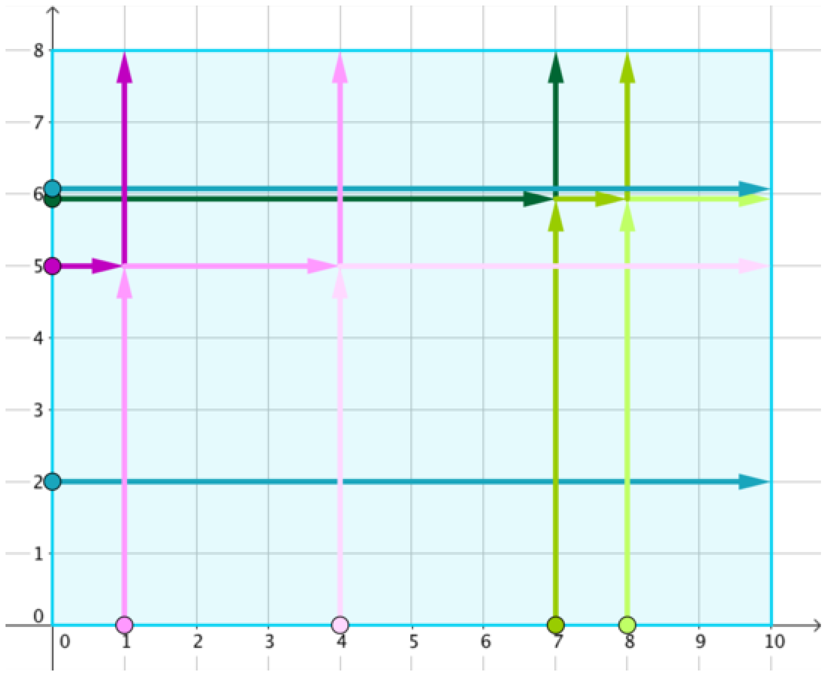
```
3 2 3
1 1 2
2 1 1
1 1 5
```

output

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

Note

The first example corresponds to the initial setup in the legend, and the tracks of dancers are marked with different colours in the following figure.



In the second example, no dancers collide.

E. Goodbye Souvenir

time limit per test: 6 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

I won't feel lonely, nor will I be sorrowful... not before everything is buried.

A string of n beads is left as the message of leaving. The beads are numbered from 1 to n from left to right, each having a shape numbered by integers between 1 and n inclusive. Some beads may have the same shapes.

The *memory* of a shape x in a certain subsegment of beads, is defined to be the difference between the last position and the first position that shape x appears in the segment. The *memory* of a subsegment is the sum of *memories* over all shapes that occur in it.

From time to time, shapes of beads change as well as the *memories*. Sometimes, the past secreted in subsegments are being recalled, and you are to find the *memory* for each of them.

Input

The first line of input contains two space-separated integers n and m $(1 \leq n, m \leq 100\ 000)$ — the number of beads in the string, and the total number of changes and queries, respectively.

The second line contains n integers $a_1, a_2, \ldots, a_n$ $(1 \leq a_i \leq n)$ — the initial shapes of beads $1, 2, \ldots, n$, respectively.

The following m lines each describes either a change in the beads or a query of subsegment. A line has one of the following formats:

- `1 p x` $(1 \leq p \leq n, 1 \leq x \leq n)$, meaning that the shape of the p-th bead is changed into x;
- `2 l r` $(1 \leq l \leq r \leq n)$, denoting a query of *memory* of the subsegment from l to r, inclusive.

Output

For each query, print one line with an integer — the *memory* of the recalled subsegment.

Examples

input

```
7 6
1 2 3 1 3 2 1
2 3 7
2 1 3
1 7 2
1 3 2
2 1 6
2 5 7
```

output

```
5
0
7
1
```

input

```
7 5
1 3 2 1 4 2 3
1 1 4
2 2 3
1 1 7
2 4 5
1 1 7
```

output

```
0
0
```

Note

The initial string of beads has shapes $(1, 2, 3, 1, 3, 2, 1)$.

Consider the changes and queries in their order:

1. `2 3 7`: the *memory* of the subsegment $[3, 7]$ is $(7 - 4) + (6 - 6) + (5 - 3) = 5$;
2. `2 1 3`: the *memory* of the subsegment $[1, 3]$ is $(1 - 1) + (2 - 2) + (3 - 3) = 0$;
3. `1 7 2`: the shape of the 7-th bead changes into 2. Beads now have shapes $(1, 2, 3, 1, 3, 2, 2)$ respectively;
4. `1 3 2`: the shape of the 3-rd bead changes into 2. Beads now have shapes $(1, 2, 2, 1, 3, 2, 2)$ respectively;
5. `2 1 6`: the *memory* of the subsegment $[1, 6]$ is $(4 - 1) + (6 - 2) + (5 - 5) = 7$;
6. `2 5 7`: the *memory* of the subsegment $[5, 7]$ is $(7 - 6) + (5 - 5) = 1$.

Codeforces (c) Copyright 2010-2017 Mike Mirzayanov
The only programming contests Web 2.0 platform