A. Doggo Recoloring

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

Panic is rising in the committee for doggo standardization — the puppies of the new brood have been born multi-colored! In total there are 26 possible colors of puppies in the nature and they are denoted by letters from 'a' to 'z' inclusive.

The committee rules strictly prohibit even the smallest diversity between doggos and hence all the puppies should be of the same color. Thus Slava, the committee employee, has been assigned the task to recolor some puppies into other colors in order to eliminate the difference and make all the puppies have one common color.

Unfortunately, due to bureaucratic reasons and restricted budget, there's only one operation Slava can perform: he can choose a color x such that there are currently **at least two** puppies of color x and recolor **all** puppies of the color x into some arbitrary color y. Luckily, this operation can be applied multiple times (including zero).

For example, if the number of puppies is 7 and their colors are represented as the string "`abababc`", then in one operation Slava can get the results "`zbzbzbc`", "`bbbbbbc`", "`aaaaaac`", "`acacacc`" and others. However, if the current color sequence is "`abababc`", then he can't choose x='`c`' right now, because currently only one puppy has the color '`c`'.

Help Slava and the committee determine whether it is possible to standardize all the puppies, i.e. after Slava's operations all the puppies should have the same color.

Input

The first line contains a single integer n ($1 \le n \le 10^5$) — the number of puppies.

The second line contains a string s of length n consisting of lowercase Latin letters, where the i-th symbol denotes the i-th puppy's color.

Output

If it's possible to recolor all puppies into one color, print "`Yes`".

Otherwise print "`No`".

Output the answer without quotation signs.

Examples

input

```
6
aabddc
```

output

```
Yes
```

input

```
3
abc
```

output

```
No
```

input

```
3
jjj
```

output

```
Yes
```

Note

In the first example Slava can perform the following steps:

1. take all puppies of color 'a' (a total of two) and recolor them into 'b';
2. take all puppies of color 'd' (a total of two) and recolor them into 'c';
3. take all puppies of color 'b' (three puppies for now) and recolor them into 'c'.

In the second example it's impossible to recolor any of the puppies.

In the third example all the puppies' colors are the same; thus there's no need to recolor anything.

B. Weakened Common Divisor

time limit per test: 1.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

During the research on properties of the greatest common divisor (*GCD*) of a set of numbers, Ildar, a famous mathematician, introduced a brand new concept of the weakened common divisor (*WCD*) of a list of pairs of integers.

For a given list of pairs of integers (a_1, b_1), (a_2, b_2), ..., (a_n, b_n) their *WCD* is arbitrary integer greater than 1, such that it divides at least one element in each pair. WCD may not exist for some lists.

For example, if the list looks like $[(12, 15), (25, 18), (10, 24)]$, then their WCD can be equal to 2, 3, 5 or 6 (each of these numbers is strictly greater than 1 and divides at least one number in each pair).

You're currently pursuing your PhD degree under Ildar's mentorship, and that's why this problem was delegated to you. Your task is to calculate *WCD* efficiently.

Input

The first line contains a single integer n ($1 \le n \le 150\,000$) — the number of pairs.

Each of the next n lines contains two integer values a_i, b_i ($2 \le a_i, b_i \le 2 \cdot 10^9$).

Output

Print a single integer — the *WCD* of the set of pairs.

If there are multiple possible answers, output any; if there is no answer, print -1.

Examples

input

```
3
17 18
15 24
12 15
```

output

```
6
```

input

```
2
10 16
7 17
```

output

```
-1
```

input

```
5
90 108
45 105
75 40
165 175
33 30
```

output

```
5
```

Note

In the first example the answer is 6 since it divides 18 from the first pair, 24 from the second and 12 from the third ones. Note that other valid answers will also be accepted.

In the second example there are no integers greater than 1 satisfying the conditions.

In the third example one of the possible answers is 5. Note that, for example, 15 is also allowed, but it's not necessary to maximize the output.

C. Plasticine zebra

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

Is there anything better than going to the zoo after a tiresome week at work? No wonder Grisha feels the same while spending the entire weekend accompanied by pretty striped zebras.

Inspired by this adventure and an accidentally found plasticine pack (represented as a sequence of black and white stripes), Grisha now wants to select several consequent (contiguous) pieces of alternating colors to create a zebra. Let's call the number of selected pieces the length of the zebra.

Before assembling the zebra Grisha can make the following operation 0 or more times. He splits the sequence in some place into two parts, then reverses each of them and sticks them together again. For example, if Grisha has pieces in the order "`bwbbw`" (here 'b' denotes a black strip, and 'w' denotes a white strip), then he can split the sequence as `bw|bbw` (here the vertical bar represents the cut), reverse both parts and obtain "`wbwbb`".

Determine the maximum possible length of the zebra that Grisha can produce.

Input

The only line contains a string s ($1 \le |s| \le 10^5$, where $|s|$ denotes the length of the string s) comprised of lowercase English letters 'b' and 'w' only, where 'w' denotes a white piece and 'b' denotes a black piece.

Output

Print a single integer — the maximum possible zebra length.

Examples

input
```
bwwwbwwbw
```
output
```
5
```

input
```
bwwbwwb
```
output
```
3
```

Note

In the first example one of the possible sequence of operations is `bwwwbww|bw` $\to$ `w|wbwwwbwb` $\to$ **`wbwbw`**`wwbw`, that gives the answer equal to 5.

In the second example no operation can increase the answer.

D. Recovering BST

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

Dima the hamster enjoys nibbling different things: cages, sticks, bad problemsetters and even trees!

Recently he found a binary search tree and instinctively nibbled all of its edges, hence messing up the vertices. Dima knows that if Andrew, who has been thoroughly assembling the tree for a long time, comes home and sees his creation demolished, he'll get extremely upset.

To not let that happen, Dima has to recover the binary search tree. Luckily, he noticed that any two vertices connected by a direct edge had their greatest common divisor value exceed 1.

Help Dima construct such a **binary search tree** or determine that it's impossible. The definition and properties of a binary search tree can be found here.

Input

The first line contains the number of vertices n ($2 \le n \le 700$).

The second line features n distinct integers a_i ($2 \le a_i \le 10^9$) — the values of vertices **in ascending order**.

Output

If it is possible to reassemble the binary search tree, such that the greatest common divisor of any two vertices connected by the edge is greater than 1, print "`Yes`" (quotes for clarity).

Otherwise, print "`No`" (quotes for clarity).

Examples

input
```
6
3 6 9 18 36 108
```
output
```
Yes
```

input

```
2
7 17
```

output

```
No
```

input

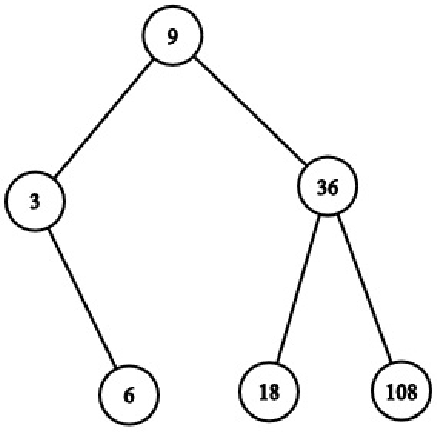
```
9
4 8 10 12 15 18 33 44 81
```

output

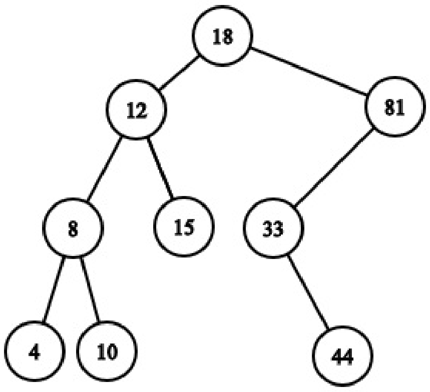
```
Yes
```

Note

The picture below illustrates one of the possible trees for the first example.



The picture below illustrates one of the possible trees for the third example.



E. Colored Cubes

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

Vasya passes all exams! Despite expectations, Vasya is not tired, moreover, he is ready for new challenges. However, he does not want to work too hard on difficult problems.

Vasya remembered that he has a not-so-hard puzzle: m colored cubes are placed on a chessboard of size $n \times n$. The fact is that $m \leq n$ and all cubes have distinct colors. Each cube occupies exactly one cell. Also, there is a designated cell for each cube on the board, the puzzle is to place each cube on its place. The cubes are fragile, so in one operation you only can move one cube onto one of four neighboring by side cells, if only it is empty. Vasya wants to be careful, so each operation takes exactly one second.

Vasya used to train hard for VK Cup Final, so he can focus his attention on the puzzle for at most 3 hours, that is 10800 seconds. Help Vasya find such a sequence of operations that all cubes will be moved onto their designated places, and Vasya won't lose his attention.

Input

The first line contains two integers n and m ($1 \leq m \leq n \leq 50$).

Each of the next m lines contains two integers x_i, y_i ($1 \le x_i, y_i \le n$), the initial positions of the cubes.

The next m lines describe the designated places for the cubes in the same format and order.

It is guaranteed that all initial positions are distinct and all designated places are distinct, however, it is possible that some initial positions coincide with some final positions.

Output

In the first line print a single integer k ($0 \le k \le 10800$) — the number of operations Vasya should make.

In each of the next k lines you should describe one operation: print four integers x_1, y_1, x_2, y_2, where x_1, y_1 is the position of the cube Vasya should move, and x_2, y_2 is the new position of the cube. The cells x_1, y_1 and x_2, y_2 should have a common side, the cell x_2, y_2 should be empty before the operation.

We can show that there always exists at least one solution. If there are multiple solutions, print any of them.

Examples

input

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

output

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

input

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

output

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

input

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

output

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

input

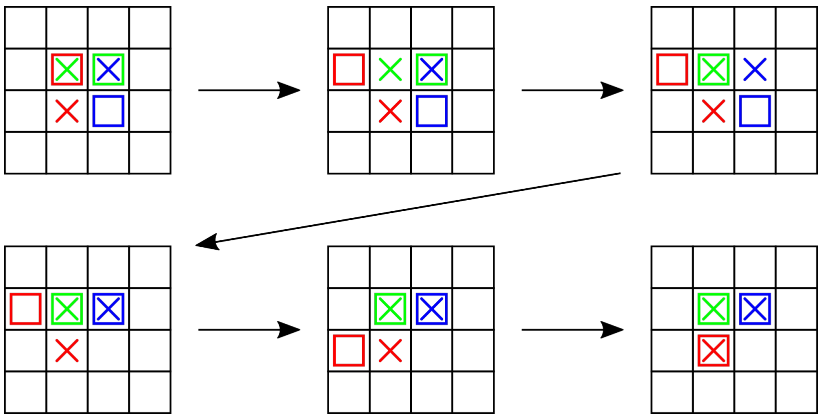
```
4 3
2 2
2 3
3 3
3 2
2 2
2 3
```

output

```
9
2 2 1 2
1 2 1 1
2 3 2 2
3 3 2 3
2 2 1 2
1 1 2 1
2 1 3 1
3 1 3 2
1 2 2 2
```

Note

In the fourth example the printed sequence of movements (shown on the picture below) is valid, but not shortest. There is a solution in 3 operations.



F. Disjoint Triangles

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

A point *belongs* to a triangle if it lies inside the triangle or on one of its sides. Two triangles are *disjoint* if there is no point on the plane that belongs to both triangles.

You are given n points on the plane. No two points coincide and no three points are collinear.

Find the number of different ways to choose two disjoint triangles with vertices in the given points. Two ways which differ only in order of triangles or in order of vertices inside triangles are considered equal.

Input

The first line of the input contains an integer n ($6 \le n \le 2000$) – the number of points.

Each of the next n lines contains two integers x_i and y_i ($|x_i|, |y_i| \le 10^9$) – the coordinates of a point.

No two points coincide and no three points are collinear.

Output

Print one integer – the number of ways to choose two disjoint triangles.

Examples

input

```
6
1 1
2 2
4 6
4 5
7 2
5 3
```

output

```
6
```

input

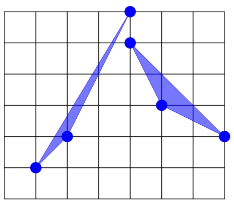
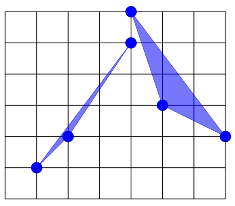
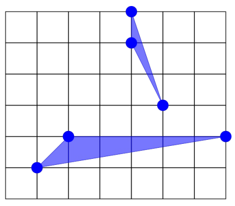
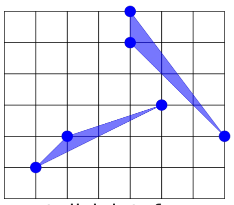
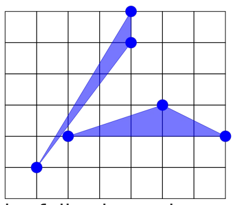
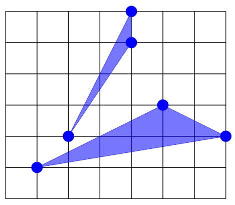
```
7
0 -1000000000
-5 -5
5 -5
-5 0
5 0
-2 2
2 2
```

output

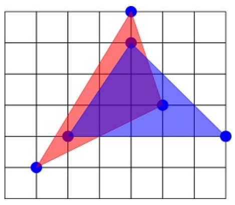
```
21
```

Note

In the first example there are six pairs of disjoint triangles, they are shown on the picture below.

All other pairs of triangles are not disjoint, for example the following pair:



G. Company Acquisitions

time limit per test: 1 second

memory limit per test: 256 megabytes

input: standard input

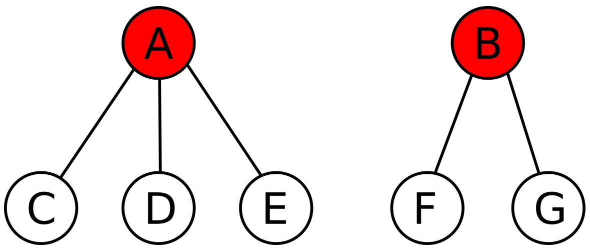
output: standard output

There are n startups. Startups can be *active* or *acquired*. If a startup is *acquired*, then that means it has exactly one *active* startup that it is following. An active startup can have arbitrarily many acquired startups that are following it. An active startup cannot follow any other startup.

The following steps happen until there is exactly one active startup. The following sequence of steps takes exactly 1 day.

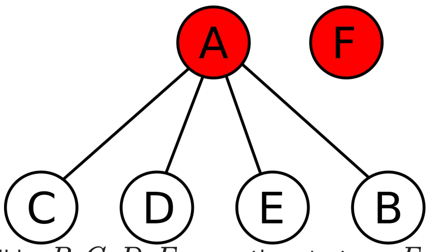
1. Two distinct active startups A, B, are chosen uniformly at random.
2. A fair coin is flipped, and with equal probability, A acquires B or B acquires A (i.e. if A acquires B, then that means B's state changes from active to acquired, and its starts following A).
3. When a startup changes from active to acquired, all of its previously acquired startups become active.

For example, the following scenario can happen: Let's say A, B are active startups. C, D, E are acquired startups under A, and F, G are acquired startups under B:

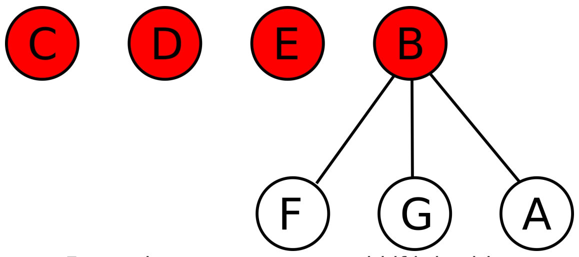


Active startups are shown in red.

If A acquires B, then the state will be A, F, G are active startups. C, D, E, B are acquired startups under A. F and G have no acquired startups:



If instead, B acquires A, then the state will be B, C, D, E are active startups. F, G, A are acquired startups under B. C, D, E have no acquired startups:



You are given the initial state of the startups. For each startup, you are told if it is either acquired or active. If it is acquired, you are also given the index of the active startup that it is following.

You're now wondering, what is the expected number of days needed for this process to finish with exactly one active startup at the end.

It can be shown the expected number of days can be written as a rational number P/Q, where P and Q are co-prime integers, and $Q \not\equiv 0 \pmod{10^9+7}$. Return the value of $P \cdot Q^{-1}$ modulo 10^9+7.

Input

The first line contains a single integer n ($2 \leq n \leq 500$), the number of startups.

The next line will contain n space-separated integers $a_1, a_2, \ldots, a_n$ ($a_i = -1$ or $1 \leq a_i \leq n$). If $a_i = -1$, then that means startup i is active. Otherwise, if $1 \leq a_i \leq n$, then startup i is acquired, and it is currently following startup a_i. It is guaranteed if $a_i \neq -1$, then $a_{a_i} = -1$ (that is, all startups that are being followed are active).

Output

Print a single integer, the expected number of days needed for the process to end with exactly one active startup, modulo 10^9+7.

Examples

input

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

output

```
3
```

input

```
2
2 -1
```

output

```
0
```

input

```
40
3 3 -1 -1 4 4 -1 -1 -1 -1 -1 10 10 10 10 10 10 4 20 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 3 3 3 3 3 3 3 3
```

output

```
755808950
```

Note

In the first sample, there are three active startups labeled 1, 2 and 3, and zero acquired startups. Here's an example of how one scenario can happen

1. Startup 1 acquires startup 2 (This state can be represented by the array $[-1, 1, -1]$)
2. Startup 3 acquires startup 1 (This state can be represented by the array $[3, -1, -1]$)
3. Startup 2 acquires startup 3 (This state can be represented by the array $[-1, -1, 2]$).
4. Startup 2 acquires startup 1 (This state can be represented by the array $[2, -1, 2]$).

At this point, there is only one active startup, and this sequence of steps took 4 days. It can be shown the expected number of days is 3.

For the second sample, there is only one active startup, so we need zero days.

For the last sample, remember to take the answer modulo 10^9+7.