A. Reachable Numbers

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

Let's denote a function $f(x)$ in such a way: we add 1 to x, then, while there is at least one trailing zero in the resulting number, we remove that zero. For example,

- $f(599) = 6$: $599 + 1 = 600 \to 60 \to 6$;
- $f(7) = 8$: $7 + 1 = 8$;
- $f(9) = 1$: $9 + 1 = 10 \to 1$;
- $f(10099) = 101$: $10099 + 1 = 10100 \to 1010 \to 101$.

We say that some number y is **reachable** from x if we can apply function f to x some (possibly zero) times so that we get y as a result. For example, 102 is reachable from 10098 because $f(f(f(10098))) = f(f(10099)) = f(101) = 102$; and any number is reachable from itself.

You are given a number n; your task is to count how many different numbers are reachable from n.

Input

The first line contains one integer n ($1 \le n \le 10^9$).

Output

Print one integer: the number of different numbers that are reachable from n.

Examples

input
```
1098
```
output
```
20
```

input
```
10
```
output
```
19
```

Note

The numbers that are reachable from 1098 are:

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1098, 1099.$$

B. Long Number

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

You are given a long decimal number a consisting of n digits from 1 to 9. You also have a function f that maps every digit from 1 to 9 to some (possibly the same) digit from 1 to 9.

You can perform the following operation **no more than once**: choose a non-empty **contiguous subsegment** of digits in a, and replace each digit x from this segment with $f(x)$. For example, if $a = 1337$, $f(1) = 1$, $f(3) = 5$, $f(7) = 3$, and you choose the segment consisting of three rightmost digits, you get 1553 as the result.

What is the maximum possible number you can obtain applying this operation no more than once?

Input

The first line contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the number of digits in a.

The second line contains a string of n characters, denoting the number a. Each character is a decimal digit from 1 to 9.

The third line contains exactly 9 integers $f(1)$, $f(2)$, ..., $f(9)$ ($1 \le f(i) \le 9$).

Output

Print the maximum number you can get after applying the operation described in the statement no more than once.

Examples

input

```
4
1337
1 2 5 4 6 6 3 1 9
```

output

```
1557
```

input

```
5
11111
9 8 7 6 5 4 3 2 1
```

output

```
99999
```

input

```
2
33
1 1 1 1 1 1 1 1 1
```

output

```
33
```

C1. Increasing Subsequence (easy version)

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

The only difference between problems C1 and C2 is that all values in input of problem C1 are distinct (this condition may be false for problem C2).

You are given a sequence a consisting of n integers. **All these integers are distinct, each value from 1 to n appears in the sequence exactly once.**

You are making a sequence of moves. During each move you must take either the leftmost element of the sequence or the rightmost element of the sequence, write it down and remove it from the sequence. Your task is to write down a **strictly** increasing sequence, and among all such sequences you should take the longest (the length of the sequence is the number of elements in it).

For example, for the sequence $[2, 1, 5, 4, 3]$ the answer is 4 (you take 2 and the sequence becomes $[1, 5, 4, 3]$, then you take the rightmost element 3 and the sequence becomes $[1, 5, 4]$, then you take 4 and the sequence becomes $[1, 5]$ and then you take 5 and the sequence becomes $[1]$, the obtained increasing sequence is $[2, 3, 4, 5]$).

Input

The first line of the input contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the number of elements in a.

The second line of the input contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le n$), where a_i is the i-th element of a. **All these integers are pairwise distinct**.

Output

In the first line of the output print k — the maximum number of elements in a **strictly** increasing sequence you can obtain.

In the second line print a string s of length k, where the j-th character of this string s_j should be 'L' if you take the leftmost element during the j-th move and 'R' otherwise. If there are multiple answers, you can print any.

Examples

input

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

output

```
4
LRRR
```

input

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

output

```
7
```

```
LRLRLLL
```

input

```
3
1 2 3
```

output

```
3
LLL
```

input

```
4
1 2 4 3
```

output

```
4
LLRL
```

Note

The first example is described in the problem statement.

C2. Increasing Subsequence (hard version)

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

The only difference between problems C1 and C2 is that all values in input of problem C1 are distinct (this condition may be false for problem C2).

You are given a sequence a consisting of n integers.

You are making a sequence of moves. During each move you must take either the leftmost element of the sequence or the rightmost element of the sequence, write it down and remove it from the sequence. Your task is to write down a **strictly** increasing sequence, and among all such sequences you should take the longest (the length of the sequence is the number of elements in it).

For example, for the sequence $[1, 2, 4, 3, 2]$ the answer is 4 (you take 1 and the sequence becomes $[2, 4, 3, 2]$, then you take the rightmost element 2 and the sequence becomes $[2, 4, 3]$, then you take 3 and the sequence becomes $[2, 4]$ and then you take 4 and the sequence becomes $[2]$, the obtained increasing sequence is $[1, 2, 3, 4]$).

Input

The first line of the input contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the number of elements in a.

The second line of the input contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 2 \cdot 10^5$), where a_i is the i-th element of a.

Output

In the first line of the output print k — the maximum number of elements in a **strictly** increasing sequence you can obtain.

In the second line print a string s of length k, where the j-th character of this string s_j should be 'L' if you take the leftmost element during the j-th move and 'R' otherwise. If there are multiple answers, you can print any.

Examples

input

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

output

```
4
LRRR
```

input

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

output

```
6
LRLRRR
```

input

```
3
2 2 2
```

output

```
1
R
```

input

```
4
1 2 4 3
```

output

```
4
LLRR
```

Note

The first example is described in the problem statement.

D. N Problems During K Days

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

Polycarp has to solve exactly n problems to improve his programming skill before an important programming competition. But this competition will be held very soon, most precisely, it will start in k days. It means that Polycarp has exactly k days for training!

Polycarp doesn't want to procrastinate, so he wants to solve at least one problem during each of k days. He also doesn't want to overwork, so if he solves x problems during some day, he should solve no more than $2x$ problems during the next day. And, at last, he wants to improve his skill, so if he solves x problems during some day, he should solve at least $x+1$ problem during the next day.

More formally: let $[a_1, a_2, \dots, a_k]$ be the array of numbers of problems solved by Polycarp. The i-th element of this array is the number of problems Polycarp solves during the i-th day of his training. Then the following conditions must be satisfied:

- sum of all a_i for i from 1 to k should be n;
- a_i should be **greater than zero** for each i from 1 to k;
- the condition $a_i < a_{i+1} \le 2a_i$ should be satisfied for each i from 1 to $k-1$.

Your problem is to find **any** array a of length k satisfying the conditions above or say that it is impossible to do it.

Input

The first line of the input contains two integers n and k ($1 \le n \le 10^9, 1 \le k \le 10^5$) — the number of problems Polycarp wants to solve and the number of days Polycarp wants to train.

Output

If it is impossible to find any array a of length k satisfying Polycarp's rules of training, print "NO" in the first line.

Otherwise print "YES" in the first line, then print k integers $a_1, a_2, \dots, a_k$ in the second line, where a_i should be the number of problems Polycarp should solve during the i-th day. If there are multiple answers, you can print any.

Examples

input

```
26 6
```

output

```
YES
1 2 4 5 6 8
```

input

```
8 3
```

output

```
NO
```

input

```
1 1
```

output

```
YES
1
```

input

```
9 4
```

output

```
NO
```

E. Minimum Array

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

You are given two arrays a and b, both of length n. All elements of both arrays are from 0 to $n-1$.

You can reorder elements of the array b (if you want, you may leave the order of elements as it is). After that, let array c be the array of length n, the i-th element of this array is $c_i = (a_i + b_i) \% n$, where $x \% y$ is x modulo y.

Your task is to reorder elements of the array b to obtain the **lexicographically** minimum possible array c.

Array x of length n is lexicographically less than array y of length n, if there exists such i ($1 \le i \le n$), that $x_i < y_i$, and for any j ($1 \le j < i$) $x_j = y_j$.

Input

The first line of the input contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the number of elements in a, b and c.

The second line of the input contains n integers $a_1, a_2, \dots, a_n$ ($0 \le a_i < n$), where a_i is the i-th element of a.

The third line of the input contains n integers $b_1, b_2, \dots, b_n$ ($0 \le b_i < n$), where b_i is the i-th element of b.

Output

Print the **lexicographically** minimum possible array c. Recall that your task is to reorder elements of the array b and obtain the **lexicographically** minimum possible array c, where the i-th element of c is $c_i = (a_i + b_i) \% n$.

Examples

input

```
4
0 1 2 1
3 2 1 1
```

output

```
1 0 0 2
```

input

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

output

```
0 0 0 1 0 2 4
```

F. Maximum Balanced Circle

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

There are n people in a row. The height of the i-th person is a_i. You can choose **any** subset of these people and try to arrange them into a **balanced circle**.

A **balanced circle** is such an order of people that the difference between heights of any adjacent people is no more than 1. For example, let heights of chosen people be $[a_{i_1}, a_{i_2}, \dots, a_{i_k}]$, where k is the number of people you choose. Then the condition $|a_{i_j} - a_{i_{j+1}}| \le 1$ should be satisfied for all j from 1 to $k-1$ and the condition $|a_{i_1} - a_{i_k}| \le 1$ should be also satisfied. $|x|$ means the absolute value of x. It is obvious that the circle consisting of one person is balanced.

Your task is to choose the maximum number of people and construct a **balanced circle** consisting of all chosen people. It is obvious that the circle consisting of one person is balanced so the answer always exists.

Input

The first line of the input contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the number of people.

The second line of the input contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 2 \cdot 10^5$), where a_i is the height of the i-th person.

Output

In the first line of the output print k — the number of people in the maximum **balanced circle**.

In the second line print k integers $res_1, res_2, \dots, res_k$, where res_j is the **height** of the j-th person in the maximum **balanced circle**. The condition $|res_j - res_{j+1}| \le 1$ should be satisfied for all j from 1 to $k-1$ and the condition $|res_1 - res_k| \le 1$ should be also satisfied.

Examples

input

```
7
```

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

output

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

input

```
5
3 7 5 1 5
```

output

```
2
5 5
```

input

```
3
5 1 4
```

output

```
2
4 5
```

input

```
7
2 2 3 2 1 2 2
```

output

```
7
1 2 2 2 2 3 2
```

G. Inverse of Rows and Columns

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

You are given a binary matrix a of size $n \times m$. A binary matrix is a matrix where each element is either 0 or 1.

You may perform some (possibly zero) operations with this matrix. During each operation you can inverse the row of this matrix or a column of this matrix. Formally, inverting a row is changing all values in this row to the opposite (0 to 1, 1 to 0). Inverting a column is changing all values in this column to the opposite.

Your task is to sort the initial matrix by some sequence of such operations. The matrix is considered **sorted** if the array $[a_{1,1}, a_{1,2}, \dots, a_{1,m}, a_{2,1}, a_{2,2}, \dots, a_{2,m}, \dots, a_{n,m-1}, a_{n,m}]$ is sorted *in non-descending order*.

Input

The first line of the input contains two integers n and m ($1 \le n, m \le 200$) — the number of rows and the number of columns in the matrix.

The next n lines contain m integers each. The j-th element in the i-th line is $a_{i,j}$ ($0 \le a_{i,j} \le 1$) — the element of a at position (i, j).

Output

If it is impossible to obtain a **sorted** matrix, print "NO" in the first line.

Otherwise print "YES" in the first line. In the second line print a string r of length n. The i-th character r_i of this string should be '1' if the i-th row of the matrix is inverted and '0' otherwise. In the third line print a string c of length m. The j-th character c_j of this string should be '1' if the j-th column of the matrix is inverted and '0' otherwise. If there are multiple answers, you can print any.

Examples

input

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

output

```
YES
00
10
```

input

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

output

```
YES
010
0000
```

input

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

output

```
NO
```