Codeforces Round #410 (Div. 2)

A. Mike and palindrome

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

Mike has a string s consisting of only lowercase English letters. He wants to **change exactly one** character from the string so that the resulting one is a palindrome.

A palindrome is a string that reads the same backward as forward, for example strings " `z`", "`aaa`", "`aba`", "`abccba`" are palindromes, but strings "`codeforces`", "`reality`", "`ab`" are not.

Input

The first and single line contains string s ($1 \leq |s| \leq 15$).

Output

Print "`YES`" (without quotes) if Mike can change **exactly** one character so that the resulting string is palindrome or "`NO`" (without quotes) otherwise.

Examples

input
abccaa
output
YES

input
abbcca
output
NO

input
abcda
output
YES

B. Mike and strings

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

Mike has n strings $s_1, s_2, ..., s_n$ each consisting of lowercase English letters. In one move he can choose a string s_i, erase the first character and append it to the end of the string. For example, if he has the string "coolmike", in one move he can transform it into the string "oolmikec".

Now Mike asks himself: what is minimal number of moves that he needs to do in order to make all the strings equal?

Input

The first line contains integer n $(1 \le n \le 50)$ — the number of strings.

This is followed by n lines which contain a string each. The i-th line corresponding to string s_i. Lengths of strings are equal. Lengths of each string is positive and don't exceed 50.

Output

Print the minimal number of moves Mike needs in order to make all the strings equal or print -1 if there is no solution.

Examples

input
4 xzzwo zwoxz zzwox xzzwo
output
5

input
2 molzv lzvmo
output
2

input
3 kc kc kc
output
0

input
3 aa aa ab
output
-1

Note

In the first sample testcase the optimal scenario is to perform operations in such a way as to transform all strings into "zwoxz".

C. Mike and gcd problem

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

Mike has a sequence $A = [a_1, a_2, ..., a_n]$ of length n. He considers the sequence $B = [b_1, b_2, ..., b_n]$ beautiful if the gcd of all its elements is bigger than 1, i.e. .

Mike wants to change his sequence in order to make it beautiful. In one move he can choose an index i ($1 \le i < n$), delete numbers a_i, a_{i+1} and put numbers a_i - a_{i+1}, $a_i + a_{i+1}$ in their place instead, in this order. He wants perform as few operations as possible. Find the minimal number of operations to make sequence A beautiful if it's possible, or tell him that it is impossible to do so.

is the biggest non-negative number d such that d divides b_i for every i ($1 \le i \le n$).

Input

The first line contains a single integer n ($2 \le n \le 100\,000$) — length of sequence A.

The second line contains n space-separated integers $a_1, a_2, ..., a_n$ ($1 \le a_i \le 10^9$) — elements of sequence A.

Output

Output on the first line "YES" (without quotes) if it is possible to make sequence A beautiful by performing operations described above, and "NO" (without quotes) otherwise.

If the answer was "YES", output the minimal number of moves needed to make sequence A beautiful.

Examples

input
2 1 1
output
YES 1

input
3 6 2 4
output
YES 0

input
2 1 3
output
YES 1

Note

In the first example you can simply make one move to obtain sequence $[0, 2]$ with .

In the second example the gcd of the sequence is already greater than 1.

D. Mike and distribution

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

Mike has always been thinking about the harshness of social inequality. He's so obsessed with it that sometimes it even affects him while solving problems. At the moment, Mike has two sequences of positive integers $A = [a_1, a_2, ..., a_n]$ and $B = [b_1, b_2, ..., b_n]$ of length n each which he uses to ask people some quite peculiar questions.

To test you on how good are you at spotting inequality in life, he wants you to find an "*unfair*" subset of the original sequence. To be more precise, he wants you to select k numbers $P = [p_1, p_2, ..., p_k]$ such that $1 \le p_i \le n$ for $1 \le i \le k$ and elements in P are distinct. Sequence P will represent indices of elements that you'll select from both sequences. He calls such a subset P "*unfair*" if and only if the following conditions are satisfied: $2 \cdot (a_{p_1} + ... + a_{p_k})$ is **greater** than the sum of all elements from sequence A, and $2 \cdot (b_{p_1} + ... + b_{p_k})$ is **greater** than the sum of all elements from the sequence B. Also, k should be smaller or equal to because it will be to easy to find sequence P if he allowed you to select too many elements!

Mike guarantees you that a solution will always exist given the conditions described above, so please help him satisfy his curiosity!

Input

The first line contains integer n $(1 \le n \le 10^5)$ — the number of elements in the sequences.

On the second line there are n space-separated integers $a_1, ..., a_n$ $(1 \le a_i \le 10^9)$ — elements of sequence A.

On the third line there are also n space-separated integers $b_1, ..., b_n$ $(1 \le b_i \le 10^9)$ — elements of sequence B.

Output

On the first line output an integer k which represents the size of the found subset. k should be less or equal to .

On the next line print k integers $p_1, p_2, ..., p_k$ $(1 \le p_i \le n)$ — the elements of sequence P. You can print the numbers in any order you want. Elements in sequence P should be distinct.

Example

input
5 8 7 4 8 3 4 2 5 3 7
output
3 1 4 5

E. Mike and code of a permutation

time limit per test: 4 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

Mike has discovered a new way to encode permutations. If he has a permutation $P = [p_1, p_2, ..., p_n]$, he will encode it in the following way:

Denote by $A = [a_1, a_2, ..., a_n]$ a sequence of length n which will represent the code of the permutation. For each i from 1 to n sequentially, he will choose the smallest unmarked j ($1 \le j \le n$) such that $p_i < p_j$ and will assign to a_i the number j (in other words he performs $a_i = j$) and will mark j. If there is no such j, he'll assign to a_i the number -1 (he performs $a_i = -1$).

Mike forgot his original permutation but he remembers its code. Your task is simple: find **any** permutation such that its code is the same as the code of Mike's original permutation.

You may assume that there will always be at least one valid permutation.

Input

The first line contains single integer n ($1 \le n \le 500\,000$) — length of permutation.

The second line contains n space-separated integers $a_1, a_2, ..., a_n$ ($1 \le a_i \le n$ or $a_i = -1$) — the code of Mike's permutation.

You may assume that all positive values from A are different.

Output

In first and only line print n numbers $p_1, p_2, ..., p_n$ ($1 \le p_i \le n$) — a permutation P which has the same code as the given one. Note that numbers in permutation are distinct.

Examples

input
6 2 -1 1 5 -1 4
output
2 6 1 4 5 3

input
8 2 -1 4 -1 6 -1 8 -1
output
1 8 2 7 3 6 4 5

Note

For the permutation from the first example:

$i = 1$, the smallest j is 2 because $p_2 = 6 > p_1 = 2$.

$i = 2$, there is no j because $p_2 = 6$ is the greatest element in the permutation.

$i = 3$, the smallest j is 1 because $p_1 = 2 > p_3 = 1$.

$i = 4$, the smallest j is 5 (2 was already marked) because $p_5 = 5 > p_4 = 4$.

$i = 5$, there is no j because 2 is already marked.

$i = 6$, the smallest j is 4 because $p_4 = 4 > p_6 = 3$.