Croc Champ 2013 - Qualification Round

A. Spyke Talks

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

Polycarpus is the director of a large corporation. There are n secretaries working for the corporation, each of them corresponds via the famous Spyke VoIP system during the day. We know that when two people call each other via Spyke, the Spyke network assigns a unique ID to this call, a positive integer session number.

One day Polycarpus wondered which secretaries are talking via the Spyke and which are not. For each secretary, he wrote out either the session number of his call or a 0 if this secretary wasn't talking via Spyke at that moment.

Help Polycarpus analyze these data and find out the number of pairs of secretaries that are talking. If Polycarpus has made a mistake in the data and the described situation could not have taken place, say so.

Note that the secretaries can correspond via Spyke not only with each other, but also with the people from other places. Also, Spyke conferences aren't permitted — that is, one call connects exactly two people.

Input

The first line contains integer n ($1 \le n \le 10^3$) — the number of secretaries in Polycarpus's corporation. The next line contains n space-separated integers: $id_1, id_2, ..., id_n$ ($0 \le id_i \le 10^9$). Number id_i equals the number of the call session of the i-th secretary, if the secretary is talking via Spyke, or zero otherwise.

Consider the secretaries indexed from 1 to n in some way.

Output

Print a single integer — the number of pairs of chatting secretaries, or -1 if Polycarpus's got a mistake in his records and the described situation could not have taken place.

Examples

input
6 0 1 7 1 7 10
output
2

input
3 1 1 1
output
-1

input
1 0
output
0

Note

In the first test sample there are two Spyke calls between secretaries: secretary 2 and secretary 4, secretary 3 and secretary 5.

In the second test sample the described situation is impossible as conferences aren't allowed.

B. Command Line Arguments

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

The problem describes the properties of a command line. The description somehow resembles the one you usually see in real operating systems. However, there are differences in the behavior. Please make sure you've read the statement attentively and use it as a formal document.

In the Pindows operating system a strings are the lexemes of the command line — the first of them is understood as the name of the program to run and the following lexemes are its arguments. For example, as we execute the command `" run.exe one, two . "`, we give four lexemes to the Pindows command line: `"run.exe"`, `"one,"`, `"two"`, `"."`. More formally, if we run a command that can be represented as string S (that has no quotes), then the command line lexemes are maximal by inclusion substrings of string S that contain no spaces.

To send a string with spaces or an empty string as a command line lexeme, we can use double quotes. The block of characters that should be considered as one lexeme goes inside the quotes. Embedded quotes are prohibited — that is, for each occurrence of character `"` we should be able to say clearly that the quotes are opening or closing. For example, as we run the command `""run.exe o" "" " ne, " two . " " "`, we give six lexemes to the Pindows command line: `"run.exe o"`, `""` (an empty string), `" ne, "`, `"two"`, `"."`, `" "` (a single space).

It is guaranteed that each lexeme of the command line is either surrounded by spaces on both sides or touches the corresponding command border. One of its consequences is: the opening brackets are either the first character of the string or there is a space to the left of them.

You have a string that consists of uppercase and lowercase English letters, digits, characters `".,?!""` and spaces. It is guaranteed that this string is a correct OS Pindows command line string. Print all lexemes of this command line string. Consider the character `"` to be used only in order to denote a single block of characters into one command line lexeme. In particular, the consequence is that the given string has got an even number of such characters.

Input

The single line contains a non-empty string S. String S consists of at most 10^5 characters. Each character is either an uppercase or a lowercase English letter, or a digit, or one of the `".,?!""` signs, or a space.

It is guaranteed that the given string is some correct command line string of the OS Pindows. It is guaranteed that the given command line string contains at least one lexeme.

Output

In the first line print the first lexeme, in the second line print the second one and so on. To make the output clearer, print the "<" (less) character to the left of your lexemes and the ">" (more) character to the right. Print the lexemes in the order in which they occur in the command.

Please, follow the given output format strictly. For more clarifications on the output format see the test samples.

Examples

input
"RUn.exe O" "" " 2ne, " two! . " "
output
<RUn.exe O> <> < 2ne, > <two!> <.> < >

input
firstarg second ""
output
<firstarg> <second> <>

C. Network Mask

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

The problem uses a simplified TCP/IP address model, please make sure you've read the statement attentively.

Polycarpus has found a job, he is a system administrator. One day he came across n IP addresses. Each IP address is a 32 bit number, represented as a group of four 8-bit numbers (without leading zeroes), separated by dots. For example, the record `0.255.1.123` shows a correct IP address and records `0.256.1.123` and `0.255.1.01` do not. In this problem an arbitrary group of four 8-bit numbers is a correct IP address.

Having worked as an administrator for some time, Polycarpus learned that if you know the IP address, you can use the subnet mask to get the address of the network that has this IP addess.

The *subnet mask* is an IP address that has the following property: if we write this IP address as a 32 bit string, that it is representable as "`11...11000..000`". In other words, the subnet mask first has one or more one bits, and then one or more zero bits (overall there are 32 bits). For example, the IP address `2.0.0.0` is not a correct subnet mask as its 32-bit record looks as `00000010000000000000000000000000`.

To get the network address of the IP address, you need to perform the operation of the bitwise "and" of the IP address and the subnet mask. For example, if the subnet mask is `255.192.0.0`, and the IP address is `192.168.1.2`, then the network address equals `192.128.0.0`. In the bitwise "and" the result has a bit that equals 1 if and only if both operands have corresponding bits equal to one.

Now Polycarpus wants to find all networks to which his IP addresses belong. Unfortunately, Polycarpus lost subnet mask. Fortunately, Polycarpus remembers that his IP addresses belonged to exactly k distinct networks. Help Polycarpus find the subnet mask, such that his IP addresses will belong to exactly k distinct networks. If there are several such subnet masks, find the one whose bit record contains the least number of ones. If such subnet mask do not exist, say so.

Input

The first line contains two integers, n and k ($1 \le k \le n \le 10^5$) — the number of IP addresses and networks. The next n lines contain the IP addresses. It is guaranteed that all IP addresses are distinct.

Output

In a single line print the IP address of the subnet mask in the format that is described in the statement, if the required subnet mask exists. Otherwise, print `-1`.

Examples

input
5 3 0.0.0.1 0.1.1.2 0.0.2.1 0.1.1.0 0.0.2.3
output
255.255.254.0

input
5 2 0.0.0.1 0.1.1.2 0.0.2.1 0.1.1.0 0.0.2.3
output
255.255.0.0

input
2 1 255.0.0.1 0.0.0.2
output
-1

D. Parallel Programming

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

Polycarpus has a computer with n processors. Also, his computer has n memory cells. We'll consider the processors numbered by integers from 1 to n and that the memory cells are consecutively numbered by integers from 1 to n.

Polycarpus needs to come up with a parallel program model. For each memory cell number i this program must record the value $n - i$ to this cell. In other words, for each cell you've got to find the distance to cell n.

Let's denote the value that is written in the i-th cell as a_i. Initially, $a_i = 1$ $(1 \le i < n)$ and $a_n = 0$. We will consider that only processor i can write values in the memory cell number i. All processors can read an information from some cell (several processors can read an information from some cell simultaneously).

The parallel program is executed in several steps. During each step we execute the *parallel version of the increment operation*. Executing the parallel version of the increment operation goes as follows:

1. Each processor independently of the other ones chooses some memory cell. Let's say that processor i has chosen a cell with number c_i $(1 \le c_i \le n)$.
2. All processors *simultaneously* execute operation $a_i = a_i + a_{c_i}$.

Help Polycarpus come up with the parallel program model that is executed in exactly k steps. Calculate the operations that need to be executed. Note that after k steps for all i's value a_i must be equal $n - i$.

Input

The first line contains two space-separated integers n and k $(1 \le n \le 10^4, 1 \le k \le 20)$.

It is guaranteed that at the given n and k the required sequence of operations exists.

Output

Print exactly $n \cdot k$ integers in k lines. In the first line print numbers $c_1, c_2, \ldots, c_n$ $(1 \le c_i \le n)$ for the first increment operation. In the second line print the numbers for the second increment operation. In the k-th line print the numbers for the k-th increment operation.

As a result of the printed operations for any i value a_i must equal $n - i$.

Examples

input
1 1
output
1

input
3 2
output
2 3 3 3 3 3

E. Tree-String Problem

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

A *rooted tree* is a non-directed connected graph without any cycles with a distinguished vertex, which is called the tree root. Consider the vertices of a rooted tree, that consists of n vertices, numbered from 1 to n. In this problem the tree root is the vertex number 1.

Let's represent the length of the shortest by the number of edges path in the tree between vertices v and u as $d(v, u)$.

A *parent* of vertex v in the rooted tree with the root in vertex r ($v \neq r$) is vertex p_v, such that $d(r, p_v) + 1 = d(r, v)$ and $\boldsymbol{d(p_v, v) = 1}$. For example, on the picture the parent of vertex $v = 5$ is vertex $p_5 = 2$.

One day Polycarpus came across a rooted tree, consisting of n vertices. The tree wasn't exactly ordinary: it had strings written on its edges. Polycarpus positioned the tree on the plane so as to make all edges lead from top to bottom if you go from the vertex parent to the vertex (see the picture). For any edge that lead from vertex p_v to vertex v ($1 < v \leq n$), he knows string s_v that is written on it. All strings are written on the edges from top to bottom. For example, on the picture s_7="ba". The characters in the strings are numbered starting from 0.

An example of Polycarpus's tree (corresponds to the example from the statement)

Polycarpus defines the *position* in this tree as a specific letter on a specific string. The position is written as a pair of integers (v, x) that means that the position is the x-th letter of the string s_v ($1 < v \leq n$, $0 \leq x < |s_v|$), where $|s_v|$ is the length of string s_v. For example, the highlighted letters are positions $(2, 1)$ and $(3, 1)$.

Let's consider the pair of positions (v, x) and (u, y) in Polycarpus' tree, such that the way from the first position to the second goes down on each step. We will consider that the pair of such positions defines string z. String z consists of all letters on the way from (v, x) to (u, y), written in the order of this path. For example, in the picture the highlighted positions define string "bacaba".

Polycarpus has a string t, he wants to know the number of pairs of positions that define string t. Note that the way from the first position to the second in the pair must go down everywhere. Help him with this challenging tree-string problem!

Input

The first line contains integer n ($2 \leq n \leq 10^5$) — the number of vertices of Polycarpus's tree. Next $n - 1$ lines contain the tree edges. The i-th of them contains number p_{i+1} and string s_{i+1} ($1 \leq p_{i+1} \leq n$; $p_{i+1} \neq (i + 1)$). String s_{i+1} is non-empty and consists of lowercase English letters. The last line contains string t. String t consists of lowercase English letters, its length is at least 2.

It is guaranteed that the input contains at most $3 \cdot 10^5$ English letters.

Output

Print a single integer — the required number.

Please, do not use the `%lld` specifier to read or write 64-bit integers in C++. It is preferred to use the `cin`, `cout` streams or the `%I64d` specifier.

Examples

input
7 1 ab 5 bacaba 1 abacaba 2 aca 5 ba 2 ba aba
output
6

input
7 1 ab 5 bacaba 1 abacaba 2 aca 5 ba 2 ba bacaba
output

```
4
```

Note

In the first test case string "aba" is determined by the pairs of positions: (2, 0) and (5, 0); (5, 2) and (6, 1); (5, 2) and (3, 1); (4, 0) and (4, 2); (4, 4) and (4, 6); (3, 3) and (3, 5).

Note that the string is not defined by the pair of positions (7, 1) and (5, 0), as the way between them doesn't always go down.