Codeforces Round #372 (Div. 2)

A. Crazy Computer

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

ZS the Coder is coding on a crazy computer. If you don't type in a word for a c consecutive seconds, everything you typed disappear!

More formally, if you typed a word at second a and then the next word at second b, then if $b - a \le c$, just the new word is appended to other words on the screen. If $b - a > c$, then everything on the screen disappears and after that the word you have typed appears on the screen.

For example, if $c = 5$ and you typed words at seconds $1, 3, 8, 14, 19, 20$ then at the second 8 there will be 3 words on the screen. After that, everything disappears at the second 13 because nothing was typed. At the seconds 14 and 19 another two words are typed, and finally, at the second 20, one more word is typed, and a total of 3 words remain on the screen.

You're given the times when ZS the Coder typed the words. Determine how many words remain on the screen after he finished typing everything.

Input

The first line contains two integers n and c ($1 \le n \le 100\,000$, $1 \le c \le 10^9$) — the number of words ZS the Coder typed and the crazy computer delay respectively.

The next line contains n integers $t_1, t_2, ..., t_n$ ($1 \le t_1 < t_2 < ... < t_n \le 10^9$), where t_i denotes the second when ZS the Coder typed the i-th word.

Output

Print a single positive integer, the number of words that remain on the screen after all n words was typed, in other words, at the second t_n.

Examples

input
6 5 1 3 8 14 19 20
output
3

input
6 1 1 3 5 7 9 10
output
2

Note

The first sample is already explained in the problem statement.

For the second sample, after typing the first word at the second 1, it disappears because the next word is typed at the second 3 and $3 - 1 > 1$. Similarly, only 1 word will remain at the second 9. Then, a word is typed at the second 10, so there will be two words on the screen, as the old word won't disappear because $10 - 9 \le 1$.

B. Complete the Word

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

ZS the Coder loves to read the dictionary. He thinks that a word is *nice* if there exists a **substring** (contiguous segment of letters) of it of length 26 where each letter of English alphabet appears exactly once. In particular, if the string has length strictly less than 26, no such substring exists and thus it is not nice.

Now, ZS the Coder tells you a word, where some of its letters are missing as he forgot them. He wants to determine if it is possible to fill in the missing letters so that the resulting word is nice. If it is possible, he needs you to find an example of such a word as well. Can you help him?

Input

The first and only line of the input contains a single string s ($1 \le |s| \le 50\,000$), the word that ZS the Coder remembers. Each character of the string is the uppercase letter of English alphabet ('A'-'Z') or is a question mark ('?'), where the question marks denotes the letters that ZS the Coder can't remember.

Output

If there is no way to replace all the question marks with **uppercase letters** such that the resulting word is nice, then print -1 in the only line.

Otherwise, print a string which denotes a possible nice word that ZS the Coder learned. This string should match the string from the input, except for the question marks replaced with uppercase English letters.

If there are multiple solutions, you may print any of them.

Examples

input
ABC??FGHIJK???OPQR?TUVWXY?
output
ABCDEFGHIJKLMNOPQRZTUVWXYS

input
WELCOMETOCODEFORCESROUNDTHREEHUNDREDANDSEVENTYTWO
output
-1

input
??????????????????????????
output
MNBVCXZLKJHGFDSAQPWOEIRUYT

input
AABCDEFGHIJKLMNOPQRSTUVW??M
output
-1

Note

In the first sample case, `ABCDEFGHIJKLMNOPQRZTUVWXYS` is a valid answer beacuse it contains a substring of length 26 (the whole string in this case) which contains all the letters of the English alphabet exactly once. Note that there are many possible solutions, such as `ABCDEFGHIJKLMNOPQRSTUVWXYZ` or `ABCEDFGHIJKLMNOPQRZTUVWXYS`.

In the second sample case, there are no missing letters. In addition, the given string does not have a substring of length 26 that contains all the letters of the alphabet, so the answer is -1.

In the third sample case, any string of length 26 that contains all letters of the English alphabet fits as an answer.

C. Plus and Square Root

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

ZS the Coder is playing a game. There is a number displayed on the screen and there are two buttons, '$+$' (plus) and '' (square root). Initially, the number 2 is displayed on the screen. There are $n+1$ levels in the game and ZS the Coder start at the level 1.

When ZS the Coder is at level k, he can :

1. *Press the '$+$' button*. This increases the number on the screen by exactly k. So, if the number on the screen was x, it becomes $x+k$.
2. *Press the '' button*. Let the number on the screen be x. After pressing this button, the number becomes . After that, ZS the Coder levels up, so his current level becomes $k+1$. This button can only be pressed when x is a **perfect square**, i.e. $x=m^2$ for some positive integer m.

Additionally, after each move, if ZS the Coder is at level k, and the number on the screen is m, then ***m* must be a multiple of *k***. Note that this condition is only checked after performing the press. For example, if ZS the Coder is at level 4 and current number is 100, he presses the '' button and the number turns into 10. Note that at this moment, 10 is not divisible by 4, but this press is still valid, because after it, ZS the Coder is at level 5, and 10 is divisible by 5.

ZS the Coder needs your help in beating the game — he wants to reach level $n+1$. In other words, he needs to press the '' button n times. Help him determine the number of times he should press the '$+$' button before pressing the '' button at each level.

Please note that ZS the Coder wants to find just any sequence of presses allowing him to reach level $n+1$, but not necessarily a sequence minimizing the number of presses.

Input

The first and only line of the input contains a single integer n ($1 \le n \le 100\,000$), denoting that ZS the Coder wants to reach level $n+1$.

Output

Print n non-negative integers, one per line. i-th of them should be equal to the number of times that ZS the Coder needs to press the '$+$' button before pressing the '' button at level i.

Each number in the output should not exceed 10^{18}. However, the number on the screen **can be greater** than 10^{18}.

It is guaranteed that at least one solution exists. If there are multiple solutions, print any of them.

Examples

input
3
output
14 16 46

input
2
output
999999999999999998 44500000000

input
4
output
2 17 46 97

Note

In the first sample case:

On the first level, ZS the Coder pressed the '$+$' button 14 times (and the number on screen is initially 2), so the number became $2+14 \cdot 1 = 16$. Then, ZS the Coder pressed the '' button, and the number became .

After that, on the second level, ZS pressed the '$+$' button 16 times, so the number becomes $4+16 \cdot 2 = 36$. Then, ZS pressed the '' button, levelling up and changing the number into .

After that, on the third level, ZS pressed the '$+$' button 46 times, so the number becomes $6+46 \cdot 3 = 144$. Then, ZS pressed the '' button, levelling up and changing the number into .

Note that 12 is indeed divisible by 4, so ZS the Coder can reach level 4.

Also, note that pressing the ' + ' button 10 times on the third level before levelling up does not work, because the number becomes $6 + 10 \cdot 3 = 36$, and when the '' button is pressed, the number becomes and ZS the Coder is at Level 4. However, 6 is not divisible by 4 now, so this is **not a valid solution.**

In the second sample case:

On the first level, ZS the Coder pressed the ' + ' button 999999999999999998 times (and the number on screen is initially 2), so the number became $2 + 999999999999999998 \cdot 1 = 10^{18}$. Then, ZS the Coder pressed the '' button, and the number became .

After that, on the second level, ZS pressed the ' + ' button 44500000000 times, so the number becomes $10^9 + 44500000000 \cdot 2 = 9 \cdot 10^{10}$. Then, ZS pressed the '' button, levelling up and changing the number into .

Note that 300000 is a multiple of 3, so ZS the Coder can reach level 3.

D. Complete The Graph

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

ZS the Coder has drawn an undirected graph of n vertices numbered from 0 to n - 1 and m edges between them. Each edge of the graph is weighted, each weight is a **positive integer**.

The next day, ZS the Coder realized that some of the weights were erased! So he wants to reassign **positive integer** weight to each of the edges which weights were erased, so that the length of the shortest path between vertices s and t in the resulting graph is exactly L. Can you help him?

Input

The first line contains five integers n, m, L, s, t ($2 \le n \le 1000,\ 1 \le m \le 10\,000,\ 1 \le L \le 10^9,\ 0 \le s, t \le n$ - $1,\ s \ne t$) — the number of vertices, number of edges, the desired length of shortest path, starting vertex and ending vertex respectively.

Then, m lines describing the edges of the graph follow. i-th of them contains three integers, u_i, v_i, w_i ($0 \le u_i, v_i \le n$ - $1,\ u_i \ne v_i,\ 0 \le w_i \le 10^9$). u_i and v_i denote the endpoints of the edge and w_i denotes its weight. If w_i is equal to 0 then the weight of the corresponding edge was erased.

It is guaranteed that there is at most one edge between any pair of vertices.

Output

Print "NO" (without quotes) in the only line if it's not possible to assign the weights in a required way.

Otherwise, print "YES" in the first line. Next m lines should contain the edges of the resulting graph, with weights assigned to edges which weights were erased. i-th of them should contain three integers u_i, v_i and w_i, denoting an edge between vertices u_i and v_i of weight w_i. The edges of the new graph must coincide with the ones in the graph from the input. The weights that were not erased must remain unchanged whereas the new weights can be any **positive integer** not exceeding 10^{18}.

The order of the edges in the output doesn't matter. The length of the shortest path between s and t must be equal to L.

If there are multiple solutions, print any of them.

Examples

input

```
5 5 13 0 4
0 1 5
2 1 2
3 2 3
1 4 0
4 3 4
```

output

```
YES
0 1 5
2 1 2
3 2 3
1 4 8
4 3 4
```

input

```
2 1 123456789 0 1
0 1 0
```

output

```
YES
0 1 123456789
```

input

```
2 1 999999999 1 0
0 1 1000000000
```

output

```
NO
```

Note

Here's how the graph in the first sample case looks like :

In the first sample case, there is only one missing edge weight. Placing the weight of 8 gives a shortest path from 0 to 4 of length 13.

In the second sample case, there is only a single edge. Clearly, the only way is to replace the missing weight with 123456789.

In the last sample case, there is no weights to assign but the length of the shortest path doesn't match the required value, so the answer is "NO".

E. Digit Tree

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

ZS the Coder has a large tree. It can be represented as an undirected connected graph of n vertices numbered from 0 to n - 1 and n - 1 edges between them. There is a single **nonzero** digit written on each edge.

One day, ZS the Coder was bored and decided to investigate some properties of the tree. He chose a positive integer M, which is **coprime** to 10, i.e. .

ZS consider an **ordered pair** of distinct vertices (u, v) *interesting* when if he would follow the shortest path from vertex u to vertex v and write down all the digits he encounters on his path in the same order, he will get a decimal representaion of an integer divisible by M.

Formally, ZS consider an ordered pair of distinct vertices (u, v) interesting if the following states true:

- Let $a_1 = u, a_2, ..., a_k = v$ be the sequence of vertices on the shortest path from u to v in the order of encountering them;
- Let d_i $(1 \le i < k)$ be the digit written on the edge between vertices a_i and a_{i+1};
- The integer is divisible by M.

Help ZS the Coder find the number of interesting pairs!

Input

The first line of the input contains two integers, n and M $(2 \le n \le 100\,000, 1 \le M \le 10^9,)$ — the number of vertices and the number ZS has chosen respectively.

The next n - 1 lines contain three integers each. i-th of them contains u_i, v_i and w_i, denoting an edge between vertices u_i and v_i with digit w_i written on it $(0 \le u_i, v_i < n,\ 1 \le w_i \le 9)$.

Output

Print a single integer — the number of interesting (by ZS the Coder's consideration) pairs.

Examples

input
6 7 0 1 2 4 2 4 2 0 1 3 0 9 2 5 7

output
7

input
5 11 1 2 3 2 0 3 3 0 3 4 3 3

output
8

Note

In the first sample case, the interesting pairs are $(0, 4), (1, 2), (1, 5), (3, 2), (2, 5), (5, 2), (3, 5)$. The numbers that are formed by these pairs are $14, 21, 217, 91, 7, 7, 917$ respectively, which are all multiples of 7. Note that $(2, 5)$ and $(5, 2)$ are considered different.

In the second sample case, the interesting pairs are $(4, 0), (0, 4), (3, 2), (2, 3), (0, 1), (1, 0), (4, 1), (1, 4)$, and 6 of these pairs give the number 33 while 2 of them give the number 3333, which are all multiples of 11.