Codeforces Round #501 (Div. 3)

A. Points in Segments

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

You are given a set of n segments on the axis Ox, each segment has integer endpoints between 1 and m inclusive. Segments may intersect, overlap or even coincide with each other. Each segment is characterized by two integers l_i and r_i ($1 \le l_i \le r_i \le m$) — coordinates of the left and of the right endpoints.

Consider all integer points between 1 and m inclusive. Your task is to print all such points that don't belong to any segment. The point x belongs to the segment $[l; r]$ if and only if $l \le x \le r$.

Input

The first line of the input contains two integers n and m ($1 \le n, m \le 100$) — the number of segments and the upper bound for coordinates.

The next n lines contain two integers each l_i and r_i ($1 \le l_i \le r_i \le m$) — the endpoints of the i-th segment. Segments may intersect, overlap or even coincide with each other. Note, it is possible that $l_i = r_i$, i.e. a segment can degenerate to a point.

Output

In the first line print one integer k — the number of points that don't belong to any segment.

In the second line print exactly k integers in *any* order — the points that don't belong to any segment. All points you print should be distinct.

If there are no such points at all, print a single integer 0 in the first line and either leave the second line empty or do not print it at all.

Examples

input
3 5 2 2 1 2 5 5
output
2 3 4

input
1 7 1 7
output
0

Note

In the first example the point 1 belongs to the second segment, the point 2 belongs to the first and the second segments and the point 5 belongs to the third segment. The points 3 and 4 do not belong to any segment.

In the second example all the points from 1 to 7 belong to the first segment.

B. Obtaining the String

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

You are given two strings s and t. Both strings have length n and consist of lowercase Latin letters. The characters in the strings are numbered from 1 to n.

You can successively perform the following move any number of times (possibly, zero):

- swap any two adjacent (neighboring) characters of s (i.e. for any $i = \{1, 2, \ldots, n - 1\}$ you can swap s_i and s_{i+1}).

You can't apply a move to the string t. The moves are applied to the string s one after another.

Your task is to obtain the string t from the string s. Find any way to do it with at most 10^4 such moves.

You do not have to minimize the number of moves, just find any sequence of moves of length 10^4 or less to transform s into t.

Input
The first line of the input contains one integer n ($1 \le n \le 50$) — the length of strings s and t.

The second line of the input contains the string s consisting of n lowercase Latin letters.

The third line of the input contains the string t consisting of n lowercase Latin letters.

Output
If it is impossible to obtain the string t using moves, print "-1".

Otherwise in the first line print one integer k — the number of moves to transform s to t. Note that k must be an integer number between 0 and 10^4 inclusive.

In the second line print k integers c_j ($1 \le c_j < n$), where c_j means that on the j-th move you swap characters s_{c_j} and s_{c_j+1}.

If you do not need to apply any moves, print a single integer 0 in the first line and either leave the second line empty or do not print it at all.

Examples

input
6 abcdef abdfec
output
4 3 5 4 5

input
4 abcd accd
output
-1

Note
In the first example the string s changes as follows: "abcdef" $\rightarrow$ "abdcef" $\rightarrow$ "abdcfe" $\rightarrow$ "abdfce" $\rightarrow$ "abdfec".

In the second example there is no way to transform the string s into the string t through any allowed moves.

C. Songs Compression

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

Ivan has n songs on his phone. The size of the i-th song is a_i bytes. Ivan also has a flash drive which can hold at most m bytes in total. Initially, his flash drive is empty.

Ivan wants to copy all n songs to the flash drive. He can compress the songs. If he compresses the i-th song, the size of the i-th song reduces from a_i to b_i bytes ($b_i < a_i$).

Ivan can compress any subset of the songs (possibly empty) and copy all the songs to his flash drive if the sum of their sizes is at most m. He can compress *any* subset of the songs (not necessarily contiguous).

Ivan wants to find the minimum number of songs he needs to compress in such a way that all his songs fit on the drive (i.e. the sum of their sizes is less than or equal to m).

If it is impossible to copy all the songs (even if Ivan compresses all the songs), print "-1". Otherwise print the minimum number of songs Ivan needs to compress.

Input
The first line of the input contains two integers n and m ($1 \le n \le 10^5, 1 \le m \le 10^9$) — the number of the songs on Ivan's phone and the capacity of Ivan's flash drive.

The next n lines contain two integers each: the i-th line contains two integers a_i and b_i ($1 \le a_i, b_i \le 10^9$, $a_i > b_i$) — the initial size of the i-th song and the size of the i-th song after compression.

Output
If it is impossible to compress a subset of the songs in such a way that all songs fit on the flash drive, print "-1". Otherwise print the

minimum number of the songs to compress.

Examples

input
4 21 10 8 7 4 3 1 5 4
output
2

input
4 16 10 8 7 4 3 1 5 4
output
-1

Note

In the first example Ivan can compress the first and the third songs so after these moves the sum of sizes will be equal to $8 + 7 + 1 + 5 = 21 \le 21$. Also Ivan can compress the first and the second songs, then the sum of sizes will be equal $8 + 4 + 3 + 5 = 20 \le 21$. Note that compressing any single song is not sufficient to copy all the songs on the flash drive (for example, after compressing the second song the sum of sizes will be equal to $10 + 4 + 3 + 5 = 22 > 21$).

In the second example even if Ivan compresses all the songs the sum of sizes will be equal $8 + 4 + 1 + 4 = 17 > 16$.

D. Walking Between Houses

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

There are n houses in a row. They are numbered from 1 to n in order from left to right. Initially you are in the house 1.

You have to perform k moves to other house. In one move you go from your current house to some other house. You can't stay where you are (i.e., in each move the new house differs from the current house). If you go from the house x to the house y, the total distance you walked increases by $|x - y|$ units of distance, where $|a|$ is the absolute value of a. It is possible to visit the same house multiple times (but you can't visit the same house in sequence).

Your goal is to walk exactly s units of distance in total.

If it is impossible, print "NO". Otherwise print "YES" and any of the ways to do that. Remember that you should do exactly k moves.

Input

The first line of the input contains three integers n, k, s ($2 \le n \le 10^9$, $1 \le k \le 2 \cdot 10^5$, $1 \le s \le 10^{18}$) — the number of houses, the number of moves and the total distance you want to walk.

Output

If you cannot perform k moves with total walking distance equal to s, print "NO".

Otherwise print "YES" on the first line and then print exactly k integers h_i ($1 \le h_i \le n$) on the second line, where h_i is the house you visit on the i-th move.

For each j from 1 to $k - 1$ the following condition should be satisfied: $h_j \ne h_{j+1}$. Also $h_1 \ne 1$ should be satisfied.

Examples

input
10 2 15
output
YES 10 4

input
10 9 45
output
YES 10 1 10 1 2 1 2 1 6

input

10 9 81
output
YES 10 1 10 1 10 1 10 1 10

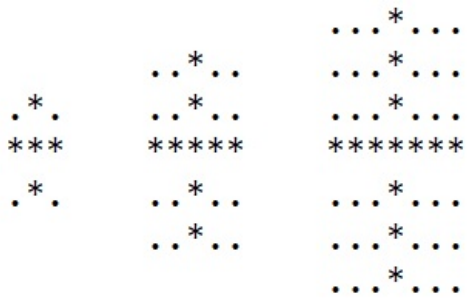
input
10 9 82
output
NO

E1. Stars Drawing (Easy Edition)

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

A *star* is a figure of the following type: an asterisk character '*' in the center of the figure and four rays (to the left, right, top, bottom) of the same positive length. The size of a *star* is the length of its rays. The size of a star must be a positive number (i.e. rays of length 0 are not allowed).

Let's consider empty cells are denoted by '.', then the following figures are *stars*:



The leftmost figure is a *star* of size 1, the middle figure is a *star* of size 2 and the rightmost figure is a *star* of size 3.

You are given a rectangular grid of size $n \times m$ consisting only of asterisks '*' and periods (dots) '.'. Rows are numbered from 1 to n, columns are numbered from 1 to m. Your task is to draw this grid using *any* number of *stars* or find out that it is impossible. *Stars* can intersect, overlap or even coincide with each other. The number of *stars* in the output can't exceed $n \cdot m$. Each star should be completely inside the grid. You can use stars of same and arbitrary sizes.

In this problem, you do not need to minimize the number of stars. Just find any way to draw the given grid with at most $n \cdot m$ stars.

Input

The first line of the input contains two integers n and m ($3 \le n, m \le 100$) — the sizes of the given grid.

The next n lines contains m characters each, the i-th line describes the i-th row of the grid. It is guaranteed that grid consists of characters '*' and '.' only.

Output

If it is impossible to draw the given grid using *stars* only, print "-1".

Otherwise in the first line print one integer k ($0 \le k \le n \cdot m$) — the number of *stars* needed to draw the given grid. The next k lines should contain three integers each — x_j, y_j and s_j, where x_j is the row index of the central *star* character, y_j is the column index of the central *star* character and s_j is the size of the *star*. Each *star* should be completely inside the grid.

Examples

input
6 8 ....*... ...**... ..*****. ...**... ....*... ....
output
3 3 4 1 3 5 2 3 5 1

input
5 5 .*... ****. .****

..**.
.....

output

3
2 2 1
3 3 1
3 4 1

input

5 5
.*...
***..
.*...
.*...
.....

output

-1

input

3 3
.
.*.
.

output

-1

Note

In the first example the output

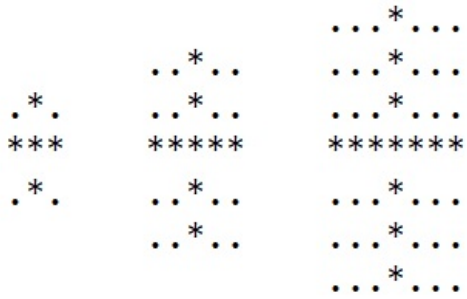
2
3 4 1
3 5 2
is also correct.

E2. Stars Drawing (Hard Edition)

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

A *star* is a figure of the following type: an asterisk character '*' in the center of the figure and four rays (to the left, right, top, bottom) of the same positive length. The size of a *star* is the length of its rays. The size of a star must be a positive number (i.e. rays of length 0 are not allowed).

Let's consider empty cells are denoted by '.', then the following figures are *stars*:



The leftmost figure is a *star* of size 1, the middle figure is a *star* of size 2 and the rightmost figure is a *star* of size 3.

You are given a rectangular grid of size $n \times m$ consisting only of asterisks '*' and periods (dots) '.'. Rows are numbered from 1 to n, columns are numbered from 1 to m. Your task is to draw this grid using *any* number of *stars* or find out that it is impossible. *Stars* can intersect, overlap or even coincide with each other. The number of *stars* in the output can't exceed $n \cdot m$. Each star should be completely inside the grid. You can use stars of same and arbitrary sizes.

In this problem, you do not need to minimize the number of stars. Just find any way to draw the given grid with at most $n \cdot m$ stars.

Input

The first line of the input contains two integers n and m ($3 \le n, m \le 1000$) — the sizes of the given grid.

The next n lines contains m characters each, the i-th line describes the i-th row of the grid. It is guaranteed that grid consists of characters '*' and '.' only.

Output

If it is impossible to draw the given grid using *stars* only, print "-1".

Otherwise in the first line print one integer k ($0 \le k \le n \cdot m$) — the number of *stars* needed to draw the given grid. The next k lines should contain three integers each — x_j, y_j and s_j, where x_j is the row index of the central *star* character, y_j is the column index of the central *star* character and s_j is the size of the *star*. Each *star* should be completely inside the grid.

Examples

input
6 8 ....*... ...**... ..*****. ...**... ....*... ....
output
3 3 4 1 3 5 2 3 5 1

input
5 5 .*... ****. .**** ..**. .....
output
3 2 2 1 3 3 1 3 4 1

input
5 5 .*... ***.. .*... .*... .....
output
-1

input
3 3 *.* .*. *.*
output
-1

Note

In the first example the output

2
3 4 1
3 5 2

is also correct.

F. Bracket Substring

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

You are given a bracket sequence s (not necessarily a regular one). A bracket sequence is a string containing only characters '(' and ')'.

A regular bracket sequence is a bracket sequence that can be transformed into a correct arithmetic expression by inserting characters '1' and '+' between the original characters of the sequence. For example, bracket sequences "()()" and "(())" are regular (the resulting expressions are: "(1)+(1)" and "((1+1)+1)"), and ")(", "(" and ")" are not.

Your problem is to calculate the number of regular bracket sequences of length $2n$ containing the given bracket sequence s as a substring (consecutive sequence of characters) modulo $10^9 + 7$ (1000000007).

Input

The first line of the input contains one integer n ($1 \le n \le 100$) — the half-length of the resulting regular bracket sequences (the resulting sequences must have length equal to $2n$).

The second line of the input contains one string s ($1 \le |s| \le 200$) — the string s that should be a substring in each of the resulting regular bracket sequences ($|s|$ is the length of s).

Output

Print only one integer — the number of regular bracket sequences containing the given bracket sequence s as a substring. Since this number can be huge, print it modulo $10^9 + 7$ (1000000007).

Examples

input
5 ()))()
output
5

input
3 (()
output
4

input
2 (((
output
0

Note

All regular bracket sequences satisfying the conditions above for the first example:

- "(((()))())";
- "(((())()))";
- "(((()))()())";
- "((()()))()";
- "()((()))()".

All regular bracket sequences satisfying the conditions above for the second example:

- "((()))";
- "(()())";
- "(())()";
- "()(())".

And there is no regular bracket sequences of length 4 containing "(((" as a substring in the third example.