Codeforces Beta Round #92 (Div. 1 Only)

A. Prime Permutation

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

You are given a string s, consisting of small Latin letters. Let's denote the length of the string as $|s|$. The characters in the string are numbered starting from 1.

Your task is to find out if it is possible to rearrange characters in string s so that for any prime number $p \leq |s|$ and for any integer i ranging from 1 to $|s| / p$ (inclusive) the following condition was fulfilled $s_p = s_{p \times i}$. If the answer is positive, find one way to rearrange the characters.

Input
The only line contains the initial string s, consisting of small Latin letters ($1 \leq |s| \leq 1000$).

Output
If it is possible to rearrange the characters in the string so that the above-mentioned conditions were fulfilled, then print in the first line "YES" (without the quotes) and print on the second line one of the possible resulting strings. If such permutation is impossible to perform, then print the single string "NO".

Sample test(s)

input
abc
output
YES abc

input
abcd
output
NO

input
xxxyxxx
output
YES xxxxxxy

Note
In the first sample any of the six possible strings will do: "abc", "acb", "bac", "bca", "cab" or "cba".

In the second sample no letter permutation will satisfy the condition at $p = 2$ ($s_2 = s_4$).

In the third test any string where character "y" doesn't occupy positions 2, 3, 4, 6 will be valid.

B. Squares

time limit per test: 0.5 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given an infinite checkered field. You should get from a square (x_1; y_1) to a square (x_2; y_2). Using the shortest path is not necessary. You can move on the field squares in four directions. That is, when you are positioned in any square, you can move to any other side-neighboring one.

A square (x; y) is considered bad, if at least one of the two conditions is fulfilled:

- $|x + y| \equiv 0 \ (mod\ 2a)$,
- $|x - y| \equiv 0 \ (mod\ 2b)$.

Your task is to find the minimum number of bad cells one will have to visit on the way from (x_1; y_1) to (x_2; y_2).

Input

The only line contains integers a, b, x_1, y_1, x_2 and y_2 — the parameters of the bad squares, the coordinates of the initial and the final squares correspondingly ($2 \le a, b \le 10^9$ and $|x_1|,|y_1|,|x_2|,|y_2| \le 10^9$). It is guaranteed that the initial and the final square aren't bad.

Output

Print a single number — the minimum number of bad cells that one will have to visit in order to travel from square (x_1; y_1) to square (x_2; y_2).

Sample test(s)

input
2 2 1 0 0 1
output
1

input
2 2 10 11 0 1
output
5

input
2 4 3 -1 3 7
output
2

Note

In the third sample one of the possible paths in (3;-1)->(3;0)->(3;1)->(3;2)->(4;2)->(4;3)->(4;4)->(4;5)->(4;6)->(4;7)->(3;7). Squares (3;1) and (4;4) are bad.

C. Brackets

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

A two dimensional array is called a *bracket* array if each grid contains one of the two possible brackets — "(" or ")". A path through the two dimensional array cells is called *monotonous* if any two consecutive cells in the path are side-adjacent and each cell of the path is located below or to the right from the previous one.

A two dimensional array whose size equals $n \times m$ is called a *correct bracket* array, if any string formed by writing out the brackets on some monotonous way from cell $(1, 1)$ to cell (n, m) forms a correct bracket sequence.

Let's define the operation of comparing two correct bracket arrays of equal size (a and b) like that. Let's consider a given two dimensional array of priorities (c) — a two dimensional array of same size, containing different integers from 1 to nm. Let's find such position (i, j) in the two dimensional array, that $a_{i,j} \neq b_{i,j}$. If there are several such positions, let's choose the one where number $c_{i,j}$ is minimum. If $a_{i,j} =$ "(", then $a < b$, otherwise $a > b$. If the position (i, j) is not found, then the arrays are considered equal.

Your task is to find a k-th two dimensional correct bracket array. It is guaranteed that for the given sizes of n and m there will be no less than k two dimensional correct bracket arrays.

Input

The first line contains integers n, m and k — the sizes of the array and the number of the sought correct bracket array ($1 \le n, m \le 100$, $1 \le k \le 10^{18}$). Then an array of priorities is given, n lines each containing m numbers, number $p_{i,j}$ shows the priority of character j in line i ($1 \le p_{i,j} \le nm$, all $p_{i,j}$ are different).

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

Output

Print the k-th two dimensional correct bracket array.

Sample test(s)

input

```
1 2 1
1 2
```

output

```
()
```

input

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

output

```
(()
())
```

input

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

output

```
()
)(
()
```

Note

In the first sample exists only one correct two-dimensional bracket array.

In the second and in the third samples two arrays exist.

A bracket sequence is called regular if it is possible to obtain correct arithmetic expression by inserting characters «+» and «1» into this sequence. For example, sequences «(()())», «()» and «(()(()))» are regular, while «)(», «(()» and «(())(» are not.

D. String

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

You are given a string s. Each pair of numbers l and r that fulfill the condition $1 \le l \le r \le |s|$, correspond to a substring of the string s, starting in the position l and ending in the position r (inclusive).

Let's define the function of two strings $F(x, y)$ like this. We'll find a list of such pairs of numbers for which the corresponding substrings of string x are equal to string y. Let's sort this list of pairs according to the pair's first number's increasing. The value of function $F(x, y)$ equals the number of non-empty continuous sequences in the list.

For example: $F(babbabbababbab, babb) = 6$. The list of pairs is as follows:

$(1, 4), (4, 7), (9, 12)$

Its continuous sequences are:

- $(1, 4)$
- $(4, 7)$
- $(9, 12)$
- $(1, 4), (4, 7)$
- $(4, 7), (9, 12)$
- $(1, 4), (4, 7), (9, 12)$

Your task is to calculate for the given string s the sum $F(s, x)$ for all x, that x belongs to the set of all substrings of a string s.

Input
The only line contains the given string s, consisting only of small Latin letters ($1 \le |s| \le 10^5$).

Output
Print the single number — the sought sum.

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

Sample test(s)

input
aaaa
output
20

input
abcdef
output
21

input
abacabadabacaba
output
188

Note
In the first sample the function values at x equal to "a", "aa", "aaa" and "aaaa" equal 10, 6, 3 and 1 correspondingly.

In the second sample for any satisfying x the function value is 1.

E. Maze

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

A maze is represented by a tree (an undirected graph, where exactly one way exists between each pair of vertices). In the maze the entrance vertex and the exit vertex are chosen with some probability. The exit from the maze is sought by Deep First Search. If there are several possible ways to move, the move is chosen equiprobably. Consider the following pseudo-code:

```
DFS(x)
    if x == exit vertex then
        finish search
    flag[x] <- TRUE
    random shuffle the vertices' order in V(x) // here all permutations have equal probability to be chosen
    for i <- 1 to length[V] do
        if flag[V[i]] = FALSE then
            count++;
            DFS(y);
    count++;
```

$V(x)$ is the list vertices adjacent to x. The $flag$ array is initially filled as `FALSE`. DFS initially starts with a parameter of an entrance vertex. When the *search is finished*, variable $count$ will contain the number of moves.

Your task is to count the mathematical expectation of the number of moves one has to do to exit the maze.

Input

The first line determines the number of vertices in the graph n ($1 \le n \le 10^5$). The next $n - 1$ lines contain pairs of integers a_i and b_i, which show the existence of an edge between a_i and b_i vertices ($1 \le a_i, b_i \le n$). It is guaranteed that the given graph is a tree.

Next n lines contain pairs of non-negative numbers x_i and y_i, which represent the probability of choosing the i-th vertex as an entrance and exit correspondingly. The probabilities to choose vertex i as an entrance and an exit equal $x_i / \sum_{j=1}^{n} x_j$ and $y_i / \sum_{j=1}^{n} y_j$ correspondingly. The sum of all x_i and the sum of all y_i are positive and do not exceed 10^6.

Output

Print the expectation of the number of moves. The absolute or relative error should not exceed 10^{-9}.

Sample test(s)

input
2 1 2 0 1 1 0
output
1.0000000000000000000

input
3 1 2 1 3 1 0 0 2 0 3
output
2.0000000000000000000

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

```
4.04081632653
```

Note

In the first sample the entrance vertex is always 1 and the exit vertex is always 2.

In the second sample the entrance vertex is always 1 and the exit vertex with the probability of 2/5 will be 2 of with the probability if 3/5 will be 3. The mathematical expectations for the exit vertices 2 and 3 will be equal (symmetrical cases). During the first move one can go to the exit vertex with the probability of 0.5 or to go to a vertex that's not the exit vertex with the probability of 0.5. In the first case the number of moves equals 1, in the second one it equals 3. The total mathematical expectation is counted as $2 / 5 \times (1 \times 0.5 + 3 \times 0.5) + 3 / 5 \times (1 \times 0.5 + 3 \times 0.5)$