A. The number of positions

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

Petr stands in line of n people, but he doesn't know exactly which position he occupies. He can say that there are no less than a people standing in front of him and no more than b people standing behind him. Find the number of different positions Petr can occupy.

Input

The only line contains three integers n, a and b ($0 \le a, b < n \le 100$).

Output

Print the single number — the number of the sought positions.

Sample test(s)

input
```
3 1 1
```
output
```
2
```

input
```
5 2 3
```
output
```
3
```

Note

The possible positions in the first sample are: 2 and 3 (if we number the positions starting with 1).

In the second sample they are 3, 4 and 5.

B. Permutations

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

You are given n k-digit integers. You have to rearrange the digits in the integers so that the difference between the largest and the smallest number was minimum. Digits should be rearranged by the same rule in all integers.

Input

The first line contains integers n and k — the number and digit capacity of numbers correspondingly ($1 \le n, k \le 8$). Next n lines contain k-digit positive integers. Leading zeroes are allowed both in the initial integers and the integers resulting from the rearranging of digits.

Output

Print a single number: the minimally possible difference between the largest and the smallest number after the digits are rearranged in all integers by the same rule.

Sample test(s)

input

```
6 4
5237
2753
7523
5723
5327
2537
```

output

```
2700
```

input

```
3 3
010
909
012
```

output

```
3
```

input

```
7 5
50808
36603
37198
44911
29994
42543
50156
```

output

```
20522
```

Note

In the first sample, if we rearrange the digits in numbers as (3,1,4,2), then the 2-nd and the 4-th numbers will equal 5237 and 2537 correspondingly (they will be maximum and minimum for such order of digits).

In the second sample, if we swap the second digits and the first ones, we get integers 100, 99 and 102.

C. 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 $p \le |s|$ 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 ($1 \le |s| \le 1000$).

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.

D. 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.

E. 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 ($1 \leq n, m \leq 100$, $1 \leq k \leq 10^{18}$). 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 ($1 \leq p_{i,j} \leq nm$, 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.