A. Bear and Three Balls

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

Limak is a little polar bear. He has n balls, the i-th ball has size t_i.

Limak wants to give one ball to each of his three friends. Giving gifts isn't easy — there are two rules Limak must obey to make friends happy:

- No two friends can get balls of the same size.
- No two friends can get balls of sizes that differ by more than 2.

For example, Limak can choose balls with sizes 4, 5 and 3, or balls with sizes 90, 91 and 92. But he can't choose balls with sizes 5, 5 and 6 (two friends would get balls of the same size), and he can't choose balls with sizes 30, 31 and 33 (because sizes 30 and 33 differ by more than 2).

Your task is to check whether Limak can choose three balls that satisfy conditions above.

Input

The first line of the input contains one integer n ($3 \le n \le 50$) — the number of balls Limak has.

The second line contains n integers $t_1, t_2, \ldots, t_n$ ($1 \le t_i \le 1000$) where t_i denotes the size of the i-th ball.

Output

Print "YES" (without quotes) if Limak can choose three balls of distinct sizes, such that any two of them differ by no more than 2. Otherwise, print "NO" (without quotes).

Examples

input
```
4
18 55 16 17
```
output
```
YES
```

input
```
6
40 41 43 44 44 44
```
output
```
NO
```

input
```
8
5 972 3 4 1 4 970 971
```
output
```
YES
```

Note

In the first sample, there are 4 balls and Limak is able to choose three of them to satisfy the rules. He must must choose balls with sizes 18, 16 and 17.

In the second sample, there is no way to give gifts to three friends without breaking the rules.

In the third sample, there is even more than one way to choose balls:

1. Choose balls with sizes 3, 4 and 5.
2. Choose balls with sizes 972, 970, 971.

B. Bear and Compressing

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

Limak is a little polar bear. Polar bears hate long strings and thus they like to compress them. You should also know that Limak is so young that he knows only first six letters of the English alphabet: 'a', 'b', 'c', 'd', 'e' and 'f'.

You are given a set of q possible operations. Limak can perform them in any order, any operation may be applied any number of times. The i-th operation is described by a string a_i of length two and a string b_i of length one. No two of q possible operations have the same string a_i.

When Limak has a string s he can perform the i-th operation on s if the first two letters of s match a two-letter string a_i. Performing the i-th operation removes first two letters of s and inserts there a string b_i. See the notes section for further clarification.

You may note that performing an operation decreases the length of a string s exactly by 1. Also, for some sets of operations there may be a string that cannot be compressed any further, because the first two letters don't match any a_i.

Limak wants to start with a string of length n and perform n - 1 operations to finally get a one-letter string "a". In how many ways can he choose the starting string to be able to get "a"? Remember that Limak can use only letters he knows.

Input

The first line contains two integers n and q ($2 \le n \le 6$, $1 \le q \le 36$) — the length of the initial string and the number of available operations.

The next q lines describe the possible operations. The i-th of them contains two strings a_i and b_i ($|a_i| = 2$, $|b_i| = 1$). It's guaranteed that $a_i \neq a_j$ for $i \neq j$ and that all a_i and b_i consist of only first six lowercase English letters.

Output

Print the number of strings of length n that Limak will be able to transform to string "a" by applying only operations given in the input.

Examples

input

```
3 5
ab a
cc c
ca a
ee c
ff d
```

output

```
4
```

input

```
2 8
af e
dc d
cc f
bc b
da b
eb a
bb b
ff c
```

output

```
1
```

input

```
6 2
bb a
ba a
```

output

```
0
```

Note

In the first sample, we count initial strings of length 3 from which Limak can get a required string "a". There are 4 such strings: "abb", "cab", "cca", "eea". The first one Limak can compress using operation 1 two times (changing "ab" to a single "a"). The first operation would change "abb" to "ab" and the second operation would change "ab" to "a".

Other three strings may be compressed as follows:

- "cab" $\longrightarrow$ "ab" $\longrightarrow$ "a"
- "cca" $\longrightarrow$ "ca" $\longrightarrow$ "a"
- "eea" $\longrightarrow$ "ca" $\longrightarrow$ "a"

In the second sample, the only correct initial string is "eb" because it can be immediately compressed to "a".

C. Bear and Up-Down

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

The life goes up and down, just like nice sequences. Sequence $t_1, t_2, \ldots, t_n$ is called *nice* if the following two conditions are satisfied:

- $t_i < t_{i+1}$ for each odd $i < n$;
- $t_i > t_{i+1}$ for each even $i < n$.

For example, sequences $(2, 8)$, $(1, 5, 1)$ and $(2, 5, 1, 100, 99, 120)$ are nice, while $(1, 1)$, $(1, 2, 3)$ and $(2, 5, 3, 2)$ are not.

Bear Limak has a sequence of positive integers $t_1, t_2, \ldots, t_n$. This sequence **is not nice** now and Limak wants to fix it by a single swap. He is going to choose two indices $i < j$ and swap elements t_i and t_j in order to get a nice sequence. Count the number of ways to do so. Two ways are considered different if indices of elements chosen for a swap are different.

Input

The first line of the input contains one integer n ($2 \le n \le 150\,000$) — the length of the sequence.

The second line contains n integers $t_1, t_2, \ldots, t_n$ ($1 \le t_i \le 150\,000$) — the initial sequence. It's guaranteed that the given sequence is not nice.

Output

Print the number of ways to swap two elements exactly once in order to get a nice sequence.

Examples

input

```
5
2 8 4 7 7
```

output

```
2
```

input

```
4
200 150 100 50
```

output

```
1
```

input

```
10
3 2 1 4 1 4 1 4 1 4
```

output

```
8
```

input

```
9
1 2 3 4 5 6 7 8 9
```

output

```
0
```

Note

In the first sample, there are two ways to get a nice sequence with one swap:

1. Swap $t_2 = 8$ with $t_4 = 7$.
2. Swap $t_1 = 2$ with $t_5 = 7$.

In the second sample, there is only one way — Limak should swap $t_1 = 200$ with $t_4 = 50$.

D. Delivery Bears

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

Niwel is a little golden bear. As everyone knows, bears live in forests, but Niwel got tired of seeing all the trees so he decided to move to the city.

In the city, Niwel took on a job managing bears to deliver goods. The city that he lives in can be represented as a directed graph with n nodes and m edges. Each edge has a weight capacity. A delivery consists of a bear carrying weights with their bear hands on a simple path from node 1 to node n. The total weight that travels across a particular edge must not exceed the weight capacity of that edge.

Niwel has **exactly** x bears. In the interest of fairness, no bear can rest, and the weight that each bear carries must be exactly the same. However, each bear may take different paths if they like.

Niwel would like to determine, what is the maximum amount of weight he can deliver (it's the sum of weights carried by bears). Find the maximum weight.

Input

The first line contains three integers n, m and x ($2 \le n \le 50$, $1 \le m \le 500$, $1 \le x \le 100\,000$) — the number of nodes, the number of directed edges and the number of bears, respectively.

Each of the following m lines contains three integers a_i, b_i and c_i ($1 \le a_i, b_i \le n$, $a_i \ne b_i$, $1 \le c_i \le 1\,000\,000$). This represents a directed edge from node a_i to b_i with weight capacity c_i. There are no self loops and no multiple edges from one city to the other city. More formally, for each i and j that $i \ne j$ it's guaranteed that $a_i \ne a_j$ or $b_i \ne b_j$. It is also guaranteed that there is at least one path from node 1 to node n.

Output

Print one real value on a single line — the maximum amount of weight Niwel can deliver if he uses exactly x bears. Your answer will be considered correct if its absolute or relative error does not exceed 10^{-6}.

Namely: let's assume that your answer is a, and the answer of the jury is b. The checker program will consider your answer correct if $\frac{|a-b|}{max(1,b)} \le 10^{-6}$.

Examples

input

```
4 4 3
1 2 2
2 4 1
1 3 1
3 4 2
```

output

```
1.5000000000
```

input

```
5 11 23
1 2 3
2 3 4
3 4 5
4 5 6
1 3 4
2 4 5
3 5 6
1 4 2
2 5 3
1 5 2
3 2 30
```

output

```
10.2222222222
```

Note

In the first sample, Niwel has three bears. Two bears can choose the path $1 \to 3 \to 4$, while one bear can choose the path $1 \to 2 \to 4$. Even though the bear that goes on the path $1 \to 2 \to 4$ can carry one unit of weight, in the interest of fairness, he is restricted to carry 0.5 units of weight. Thus, the total weight is 1.5 units overall. Note that even though Niwel can deliver more weight with just 2 bears, he must use exactly 3 bears on this day.

E. Bear and Forgotten Tree 2

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

A tree is a connected undirected graph consisting of n vertices and $n - 1$ edges. Vertices are numbered 1 through n.

Limak is a little polar bear. He once had a tree with n vertices but he lost it. He still remembers something about the lost tree though.

You are given m pairs of vertices $(a_1, b_1), (a_2, b_2), ..., (a_m, b_m)$. Limak remembers that for each i there was **no edge** between a_i and b_i. He also remembers that vertex 1 was incident to exactly k edges (its degree was equal to k).

Is it possible that Limak remembers everything correctly? Check whether there exists a tree satisfying the given conditions.

Input

The first line of the input contains three integers n, m and k ($2 \le n \le 300\,000, 0 \le m \le min(300\,000, \frac{n \cdot (n-1)}{2}), 1 \le k \le n - 1$) — the number of vertices in Limak's tree, the number of forbidden pairs of vertices, and the degree of vertex 1, respectively.

The i-th of next m lines contains two distinct integers a_i and b_i ($1 \le a_i, b_i \le n, a_i \ne b_i$) — the i-th pair that is **forbidden**. It's guaranteed that each pair of vertices will appear at most once in the input.

Output

Print "`possible`" (without quotes) if there exists at least one tree satisfying the given conditions. Otherwise, print "`impossible`" (without quotes).

Examples

input

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

output

```
possible
```

input

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

output

```
impossible
```

Note

In the first sample, there are $n = 5$ vertices. The degree of vertex 1 should be $k = 2$. All conditions are satisfied for a tree with edges $1 - 5$, $5 - 2$, $1 - 3$ and $3 - 4$.

In the second sample, Limak remembers that none of the following edges existed: $1 - 2$, $1 - 3$, $1 - 4$, $1 - 5$ and $1 - 6$. Hence, vertex 1 couldn't be connected to any other vertex and it implies that there is no suitable tree.

F. Paper task

time limit per test: 3 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

Alex was programming while Valentina (his toddler daughter) got there and started asking many questions about the round brackets (or parenthesis) in the code. He explained her a bit and when she got it he gave her a task in order to finish his code on time.

For the purpose of this problem we consider only strings consisting of opening and closing round brackets, that is characters '(' and ')'.

The sequence of brackets is called *correct* if:

1. it's empty;
2. it's a correct sequence of brackets, enclosed in a pair of opening and closing brackets;
3. it's a concatenation of two correct sequences of brackets.

For example, the sequences "()()" and "((()))(())" are correct, while ")(()", "(((((" and "())" are not.

Alex took a piece of paper, wrote a string s consisting of brackets and asked Valentina to count the number of **distinct** non-empty substrings of s that are correct sequences of brackets. In other words, her task is to count the number of non-empty correct sequences of brackets that occur in a string s as a **substring** (don't mix up with subsequences).

When Valentina finished the task, Alex noticed he doesn't know the answer. Help him don't loose face in front of Valentina and solve the problem!

Input

The first line of the input contains an integer n ($1 \leq n \leq 500\,000$) — the length of the string s.

The second line contains a string s of length n consisting of only '(' and ')'.

Output

Print the number of **distinct** non-empty correct sequences that occur in s as substring.

Examples

input

```
10
()()()()()
```

output

```
5
```

input

```
7
)(())()
```

output

```
3
```

Note

In the first sample, there are 5 distinct substrings we should count: "()", "()()", "()()()", "()()()()" and "()()()()()".

In the second sample, there are 3 distinct substrings we should count: "()", "(())" and "(())()".

G. Move by Prime

time limit per test: 5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Pussycat Sonya has an array consisting of n positive integers. There are 2^n possible subsequences of the array. For each subsequence she counts the minimum number of operations to make all its elements equal. Each operation must be one of two:

- Choose some element of the subsequence and multiply it by some prime number.
- Choose some element of the subsequence and divide it by some prime number. The chosen element must be divisible by the chosen prime number.

What is the sum of minimum number of operations for all 2^n possible subsequences? Find and print this sum modulo $10^9 + 7$.

Input

The first line of the input contains a single integer n ($1 \le n \le 300\,000$) — the size of the array.

The second line contains n integers $t_1, t_2, \ldots, t_n$ ($1 \le t_i \le 300\,000$) — elements of the array.

Output

Print the sum of minimum number of operation for all possible subsequences of the given array modulo $10^9 + 7$.

Examples

input

```
3
60 60 40
```

output

```
6
```

input

```
4
1 2 3 4
```

output

```
24
```

Note

In the first sample, there are 8 possible subsequences: $(60, 60, 40)$, $(60, 60)$, $(60, 40)$, $(60, 40)$, (60), (60), (40) and $()$ (empty subsequence).

For a subsequence $(60, 60, 40)$ we can make all elements equal by two operations — divide 40 by 2 to get 20, and then multiply 20 by 3 to get 60. It's impossible to achieve the goal using less operations and thus we add 2 to the answer.

There are two subsequences equal to $(60, 40)$ and for each of them the also need to make at least 2 operations.

In each of other subsequences all numbers are already equal, so we need 0 operations for each of them. The sum is equal to $2 + 2 + 2 = 6$.