CODEFORCES β
Sponsored by Telegram

# Codeforces Round #778 (Div. 1 + Div. 2, based on Technocup 2022 Final Round)

## A. Maximum Cake Tastiness

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

There are $n$ pieces of cake on a line. The $i$-th piece of cake has weight $a_i$ ($1 \le i \le n$).

The tastiness of the cake is the maximum total weight of two adjacent pieces of cake (i. e., $\max(a_1 + a_2,\, a_2 + a_3,\, \ldots,\, a_{n-1} + a_n)$).

You want to maximize the tastiness of the cake. You are allowed to do the following operation at most once (doing more operations would ruin the cake):

- Choose a contiguous subsegment $a[l, r]$ of pieces of cake ($1 \le l \le r \le n$), and reverse it.

The subsegment $a[l, r]$ of the array $a$ is the sequence $a_l, a_{l+1}, \dots, a_r$.

If you reverse it, the array will become $a_1, a_2, \dots, a_{l-2}, a_{l-1}, \underline{a_r}, \underline{a_{r-1}}, \underline{\dots}, \underline{a_{l+1}}, \underline{a_l}, a_{r+1}, a_{r+2}, \dots, a_{n-1}, a_n$.

For example, if the weights are initially $[5, 2, 1, 4, 7, 3]$, you can reverse the subsegment $a[2, 5]$, getting $[5, \underline{7}, \underline{4}, \underline{1}, \underline{2}, 3]$. The tastiness of the cake is now $5 + 7 = 12$ (while before the operation the tastiness was $4 + 7 = 11$).

Find the maximum tastiness of the cake after doing the operation at most once.

### Input

The first line contains a single integer $t$ ($1 \le t \le 50$) — the number of test cases.

The first line of each test case contains a single integer $n$ ($2 \le n \le 1000$) — the number of pieces of cake.

The second line of each test case contains $n$ integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 10^9$) — $a_i$ is the weight of the $i$-th piece of cake.

### Output

For each test case, print a single integer: the maximum tastiness of the cake after doing the operation at most once.

### Example

**input**

```
5
6
5 2 1 4 7 3
3
32 78 78
3
69 54 91
8
999021 999021 999021 999021 999652 999021 999021 999021
2
1000000000 1000000000
```

**output**

```
12
156
160
1998673
2000000000
```

### Note

In the first test case, after reversing the subsegment $a[2, 5]$, you get a cake with weights $[5, \underline{7}, \underline{4}, \underline{1}, \underline{2}, 3]$. The tastiness of the cake is now $\max(5+7, 7+4, 4+1, 1+2, 2+3) = 12$. This is the maximum possible tastiness of the cake one can obtain by performing the operation at most once.

In the second test case, it's optimal not to do any operation. The tastiness is $78 + 78 = 156$.

In the third test case, after reversing the subsegment $a[1, 2]$, you get a cake with weights $[\underline{54}, \underline{69}, 91]$. The tastiness of the cake is now $\max(54+69, 69+91) = 160$. There is no way to make the tastiness of the cake greater than $160$ by performing at most one operation.

## B. Prefix Removals

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

You are given a string $s$ consisting of lowercase letters of the English alphabet. You must perform the following algorithm on $s$:

- Let $x$ be the length of the longest prefix of $s$ which occurs somewhere else in $s$ as a contiguous substring (the other occurrence may also intersect the prefix). If $x = 0$, break. Otherwise, remove the first $x$ characters of $s$, and repeat.

A prefix is a string consisting of several first letters of a given string, without any reorders. An empty prefix is also a valid prefix. For example, the string "abcd" has 5 prefixes: empty string, "a", "ab", "abc" and "abcd".

For instance, if we perform the algorithm on $s =$ "abcabdc",

- Initially, "ab" is the longest prefix that also appears somewhere else as a substring in $s$, so $s =$ "cabdc" after $1$ operation.
- Then, "c" is the longest prefix that also appears somewhere else as a substring in $s$, so $s =$ "abdc" after $2$ operations.
- Now $x = 0$ (because there are no non-empty prefixes of "abdc" that also appear somewhere else as a substring in $s$), so the algorithm terminates.

Find the final state of the string after performing the algorithm.

### Input

The first line contains a single integer $t$ ($1 \le t \le 10^4$) — the number of test cases.

This is followed by $t$ lines, each containing a description of one test case. Each line contains a string $s$. The given strings consist only of lowercase letters of the English alphabet and have lengths between $1$ and $2 \cdot 10^5$ inclusive.

It is guaranteed that the sum of the lengths of $s$ over all test cases does not exceed $2 \cdot 10^5$.

### Output

For each test case, print a single line containing the string $s$ after executing the algorithm. It can be shown that such string is non-empty.

#### Example

**input**

```
6
abcabdc
a
bbbbbbbbbb
codeforces
cffcfccffccfcffcfccfcffccffcfccf
zyzyzwxxyyxxyyzzyzzxxwzxwywxwzxxyzzw
```

**output**

```
abdc
a
b
deforces
cf
xyzzw
```

### Note

The first test case is explained in the statement.

In the second test case, no operations can be performed on $s$.

In the third test case,

- Initially, $s$ = "bbbbbbbbbb".
- After $1$ operation, $s$ = "b".

In the fourth test case,

- Initially, $s$ = "codeforces".
- After $1$ operation, $s$ = "odeforces".
- After $2$ operations, $s$ = "deforces".

## C. Alice and the Cake

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

Alice has a cake, and she is going to cut it. She will perform the following operation $n-1$ times: choose a piece of the cake (initially, the cake is all one piece) with weight $w \ge 2$ and cut it into two smaller pieces of weight $\lfloor \frac{w}{2} \rfloor$ and $\lceil \frac{w}{2} \rceil$ ($\lfloor x \rfloor$ and $\lceil x \rceil$ denote floor and ceiling functions, respectively).

After cutting the cake in $n$ pieces, she will line up these $n$ pieces on a table in an arbitrary order. Let $a_i$ be the weight of the $i$-th piece in the line.

You are given the array $a$. Determine whether there exists an initial weight and sequence of operations which results in $a$.

### Input

The first line contains a single integer $t$ ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains a single integer $n$ ($1 \le n \le 2 \cdot 10^5$).

The second line of each test case contains $n$ integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$).

It is guaranteed that the sum of $n$ for all test cases does not exceed $2 \cdot 10^5$.

### Output

For each test case, print a single line: print YES if the array $a$ could have resulted from Alice's operations, otherwise print NO.

You may print each letter in any case (for example, YES, Yes, yes, yEs will all be recognized as positive answer).

#### Example

**input**

```
14
1
327
2
869 541
2
985214736 985214737
3
2 3 1
3
2 3 3
6
1 1 1 1 1 1
6
100 100 100 100 100 100
8
100 100 100 100 100 100 100 100
8
2 16 1 8 64 1 4 32
10
1 2 4 7 1 1 1 1 7 2
10
7 1 1 1 3 1 3 3 2 3
10
```

```
1 4 4 1 1 1 3 3 3 1
10
2 3 2 2 1 2 2 2 2 2
4
999999999 999999999 999999999 999999999
```

**output**

```
YES
NO
YES
YES
NO
YES
NO
YES
YES
YES
YES
NO
NO
YES
```

### Note

In the first test case, it's possible to get the array $a$ by performing $0$ operations on a cake with weight $327$.

In the second test case, it's not possible to get the array $a$.

In the third test case, it's possible to get the array $a$ by performing $1$ operation on a cake with weight $1\,970\,429\,473$:

- Cut it in half, so that the weights are $[985\,214\,736, 985\,214\,737]$.

Note that the starting weight can be greater than $10^9$.
In the fourth test case, it's possible to get the array $a$ by performing $2$ operations on a cake with weight $6$:

- Cut it in half, so that the weights are $[3, 3]$.
- Cut one of the two pieces with weight $3$, so that the new weights are $[1, 2, 3]$ which is equivalent to $[2, 3, 1]$ up to reordering.

## D. Potion Brewing Class

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

Alice's potion making professor gave the following assignment to his students: brew a potion using $n$ ingredients, such that the proportion of ingredient $i$ in the final potion is $r_i > 0$ (and $r_1 + r_2 + \cdots + r_n = 1$).

He forgot the recipe, and now all he remembers is a set of $n-1$ facts of the form, "ingredients $i$ and $j$ should have a ratio of $x$ to $y$" (i.e., if $a_i$ and $a_j$ are the amounts of ingredient $i$ and $j$ in the potion respectively, then it must hold $a_i/a_j = x/y$), where $x$ and $y$ are positive integers. However, it is guaranteed that the set of facts he remembers is sufficient to uniquely determine the original values $r_i$.

He decided that he will allow the students to pass the class as long as they submit a potion which satisfies all of the $n-1$ requirements (there may be many such satisfactory potions), and contains a positive integer amount of each ingredient.

Find the minimum total amount of ingredients needed to make a potion which passes the class. As the result can be very large, you should print the answer modulo $998\,244\,353$.

### Input

The first line contains a single integer $t$ ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains a single integer $n$ ($2 \le n \le 2 \cdot 10^5$).

Each of the next $n-1$ lines contains four integers $i, j, x, y$ ($1 \le i, j \le n$, $i \ne j$, $1 \le x, y \le n$) — ingredients $i$ and $j$ should have a ratio of $x$ to $y$. It is guaranteed that the set of facts is sufficient to uniquely determine the original values $r_i$.

It is also guaranteed that the sum of $n$ for all test cases does not exceed $2 \cdot 10^5$.

### Output

For each test case, print the minimum total amount of ingredients needed to make a potion which passes the class, modulo $998\,244\,353$.

**Example**

**input**

```
3
4
3 2 3 4
1 2 4 3
1 4 2 4
8
5 4 2 3
6 4 5 4
1 3 5 2
6 8 2 1
3 5 3 4
3 2 2 5
6 7 4 3
17
8 7 4 16
9 17 4 5
5 14 13 12
11 1 17 14
6 13 8 9
2 11 3 11
4 17 7 2
17 16 8 6
15 5 1 14
16 7 1 10
12 17 13 10
11 16 7 2
10 11 6 4
13 17 14 6
3 11 15 8
15 6 12 8
```

**output**

```
69
359
573672453
```

### Note

In the first test case, the minimum total amount of ingredients is $69$. In fact, the amounts of ingredients $1, 2, 3, 4$ of a valid potion are $16, 12, 9, 32$, respectively. The potion is valid because

- Ingredients $3$ and $2$ have a ratio of $9 : 12 = 3 : 4$;
- Ingredients $1$ and $2$ have a ratio of $16 : 12 = 4 : 3$;
- Ingredients $1$ and $4$ have a ratio of $16 : 32 = 2 : 4$.

In the second test case, the amounts of ingredients $1, 2, 3, 4, 5, 6, 7, 8$ in the potion that minimizes the total amount of ingredients are $60, 60, 24, 48, 32, 60, 45, 30$.

## E. Arithmetic Operations

time limit per test: 5 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

You are given an array of integers $a_1, a_2, \dots, a_n$.

You can do the following operation any number of times (possibly zero):

- Choose any index $i$ and set $a_i$ to any integer (positive, negative or $0$).

What is the minimum number of operations needed to turn $a$ into an arithmetic progression? The array $a$ is an arithmetic progression if $a_{i+1} - a_i = a_i - a_{i-1}$ for any $2 \le i \le n - 1$.

### Input

The first line contains a single integer $n$ ($1 \le n \le 10^5$).

The second line contains $n$ integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 10^5$).

### Output

Print a single integer: the minimum number of operations needed to turn $a$ into an arithmetic progression.

#### Examples

**input**

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

**output**

```
6
```

**input**

```
14
19 2 15 8 9 14 17 13 4 14 4 11 15 7
```

**output**

```
10
```

**input**

```
10
100000 1 60000 2 20000 4 8 16 32 64
```

**output**

```
7
```

**input**

```
4
10000 20000 10000 1
```

**output**

```
2
```

### Note

In the first test, you can get the array $a = [11, 10, 9, 8, 7, 6, 5, 4, 3]$ by performing $6$ operations:

- Set $a_3$ to $9$: the array becomes $[3, 2, 9, 8, 6, 9, 5, 4, 1]$;
- Set $a_2$ to $10$: the array becomes $[3, 10, 9, 8, 6, 9, 5, 4, 1]$;
- Set $a_6$ to $6$: the array becomes $[3, 10, 9, 8, 6, 6, 5, 4, 1]$;
- Set $a_9$ to $3$: the array becomes $[3, 10, 9, 8, 6, 6, 5, 4, 3]$;
- Set $a_5$ to $7$: the array becomes $[3, 10, 9, 8, 7, 6, 5, 4, 3]$;
- Set $a_1$ to $11$: the array becomes $[11, 10, 9, 8, 7, 6, 5, 4, 3]$.

$a$ is an arithmetic progression: in fact, $a_{i+1} - a_i = a_i - a_{i-1} = -1$ for any $2 \le i \le n - 1$.

There is no sequence of less than $6$ operations that makes $a$ an arithmetic progression.

In the second test, you can get the array $a = [-1, 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38]$ by performing $10$ operations.

In the third test, you can get the array $a = [100000, 80000, 60000, 40000, 20000, 0, -20000, -40000, -60000, -80000]$ by performing $7$ operations.

## F. Minimal String Xoration

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

You are given an integer $n$ and a string $s$ consisting of $2^n$ lowercase letters of the English alphabet. The characters of the string $s$

are $s_0 s_1 s_2 \cdots s_{2^n-1}$.

A string $t$ of length $2^n$ (whose characters are denoted by $t_0 t_1 t_2 \cdots t_{2^n-1}$) is a *xoration* of $s$ if there exists an integer $j$ ( $0 \le j \le 2^n - 1$) such that, for each $0 \le i \le 2^n - 1$, $t_i = s_{i \oplus j}$ (where $\oplus$ denotes the operation bitwise XOR).

Find the lexicographically minimal *xoration* of $s$.

A string $a$ is lexicographically smaller than a string $b$ if and only if one of the following holds:

- $a$ is a prefix of $b$, but $a \ne b$;
- in the first position where $a$ and $b$ differ, the string $a$ has a letter that appears earlier in the alphabet than the corresponding letter in $b$.

**Input**
The first line contains a single integer $n$ ($1 \le n \le 18$).

The second line contains a string $s$ consisting of $2^n$ lowercase letters of the English alphabet.

**Output**
Print a single line containing the lexicographically minimal xoration of $s$.

**Examples**

**input**
```
2
acba
```
**output**
```
abca
```

**input**
```
3
bcbaaabb
```
**output**
```
aabbbcba
```

**input**
```
4
bdbcbccdbdbaaccd
```
**output**
```
abdbdccacbdbdccb
```

**input**
```
5
ccfcffccccccffcfcfccfffffcccccff
```
**output**
```
cccccffffcccccffccfcffcccccfffff
```

**input**
```
1
zz
```
**output**
```
zz
```

**Note**
In the first test, the lexicographically minimal xoration $t$ of $s =$"acba" is "abca". It's a xoration because, for $j = 3$,

- $t_0 = s_{0 \oplus j} = s_3 =$ "a";
- $t_1 = s_{1 \oplus j} = s_2 =$ "b";
- $t_2 = s_{2 \oplus j} = s_1 =$ "c";
- $t_3 = s_{3 \oplus j} = s_0 =$ "a".

There isn't any xoration of $s$ lexicographically smaller than "abca".
In the second test, the minimal string xoration corresponds to choosing $j = 4$ in the definition of xoration.

In the third test, the minimal string xoration corresponds to choosing $j = 11$ in the definition of xoration.

In the fourth test, the minimal string xoration corresponds to choosing $j = 10$ in the definition of xoration.

In the fifth test, the minimal string xoration corresponds to choosing either $j = 0$ or $j = 1$ in the definition of xoration.

## G. Snowy Mountain

time limit per test: 5 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

There are $n$ locations on a snowy mountain range (numbered from $1$ to $n$), connected by $n - 1$ trails in the shape of a tree. Each trail has length $1$. Some of the locations are base lodges. The height $h_i$ of each location is equal to the distance to the nearest base lodge (a base lodge has height $0$).

There is a skier at each location, each skier has initial kinetic energy $0$. Each skier wants to ski along as many trails as possible. Suppose that the skier is skiing along a trail from location $i$ to $j$. Skiers are not allowed to ski uphill (i.e., if $h_i < h_j$). It costs one unit of kinetic energy to ski along flat ground (i.e., if $h_i = h_j$), and a skier gains one unit of kinetic energy by skiing downhill (i.e., if $h_i > h_j$). For each location, compute the length of the longest sequence of trails that the skier starting at that location can ski along without their kinetic energy ever becoming negative. Skiers are allowed to visit the same location or trail multiple times.

**Input**
The first line contains a single integer $n$ ($2 \le n \le 2 \cdot 10^5$).

The second line contains $n$ integers $l_1, l_2, \ldots, l_n$ ($0 \le l_i \le 1$). If $l_i = 1$, location $i$ is a base lodge; if $l_i = 0$, location $i$ is not a base lodge. It is guaranteed that there is at least $1$ base lodge.

Each of the next $n-1$ lines contains two integers $u, v$ ($1 \leq u, v \leq n$, $u \neq v$), meaning that there is a trail that connects the locations $u$ and $v$. It is guaranteed that the given trails form a tree.

### Output

Print $n$ integers: the $i$-th integer is equal to the length of the longest sequence of trails that the skier starting at location $i$ can ski along without their kinetic energy ever becoming negative.

#### Examples

**input**

```
6
1 1 0 0 0 0
1 3
2 4
3 4
4 5
5 6
```

**output**

```
0 0 1 1 3 5
```

**input**

```
9
0 0 0 0 0 0 1 1 1
1 3
2 3
2 5
3 6
4 5
4 7
5 8
6 9
```

**output**

```
5 3 2 1 1 1 0 0 0
```

**input**

```
14
0 0 0 0 0 0 0 0 0 1 1 1 1 1
1 2
2 5
3 4
4 5
3 6
4 8
5 9
7 8
6 11
7 12
8 13
9 14
10 11
```

**output**

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

**input**

```
20
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 1
17 3
11 12
6 10
18 19
8 14
16 20
5 3
2 11
7 10
2 15
8 3
3 15
9 16
7 13
16 1
19 2
2 16
6 1
4 17
```

**output**

```
2 2 1 5 3 4 8 1 2 6 4 6 10 0 0 0 3 0 1 0
```

### Note

In the first test, $h = [0, 0, 1, 1, 2, 3]$. The skier starting from $6$ can ski along at most $5$ trails, in the path $6 \to 5 \to 4 \to 3 \to 4 \to 2$ (notice that a skier can ski multiple times along the same trail and can visit more than once the same location):

- at the location $6$, the kinetic energy is $0$;
- at the location $5$, the kinetic energy increases by $1$ (because $h_5 < h_6$), so it becomes $1$;
- at the location $4$, the kinetic energy increases by $1$ (because $h_4 < h_5$), so it becomes $2$;
- at the location $3$, the kinetic energy decreases by $1$ (because $h_3 = h_4$), so it becomes $1$;
- at the location $4$, the kinetic energy decreases by $1$ (because $h_4 = h_3$), so it becomes $0$;
- at the location $2$, the kinetic energy increases by $1$ (because $h_2 < h_4$), so it becomes $1$.

There isn't any sequence of trails of length greater than $5$ such that the kinetic energy is always non-negative.

Moreover,

- the optimal path for the skier starting from $1$ is $1$ (no trails);
- the optimal path for the skier starting from $2$ is $2$ (no trails);
- the optimal path for the skier starting from $3$ is $3 \to 1$;
- the optimal path for the skier starting from $4$ is $4 \to 2$;
- the optimal path for the skier starting from $5$ is $5 \to 4 \to 3 \to 1$.

In the second test, $h = [3, 2, 2, 1, 1, 1, 0, 0, 0]$. The skier starting from $1$ can ski along at most $5$ trails, in the path

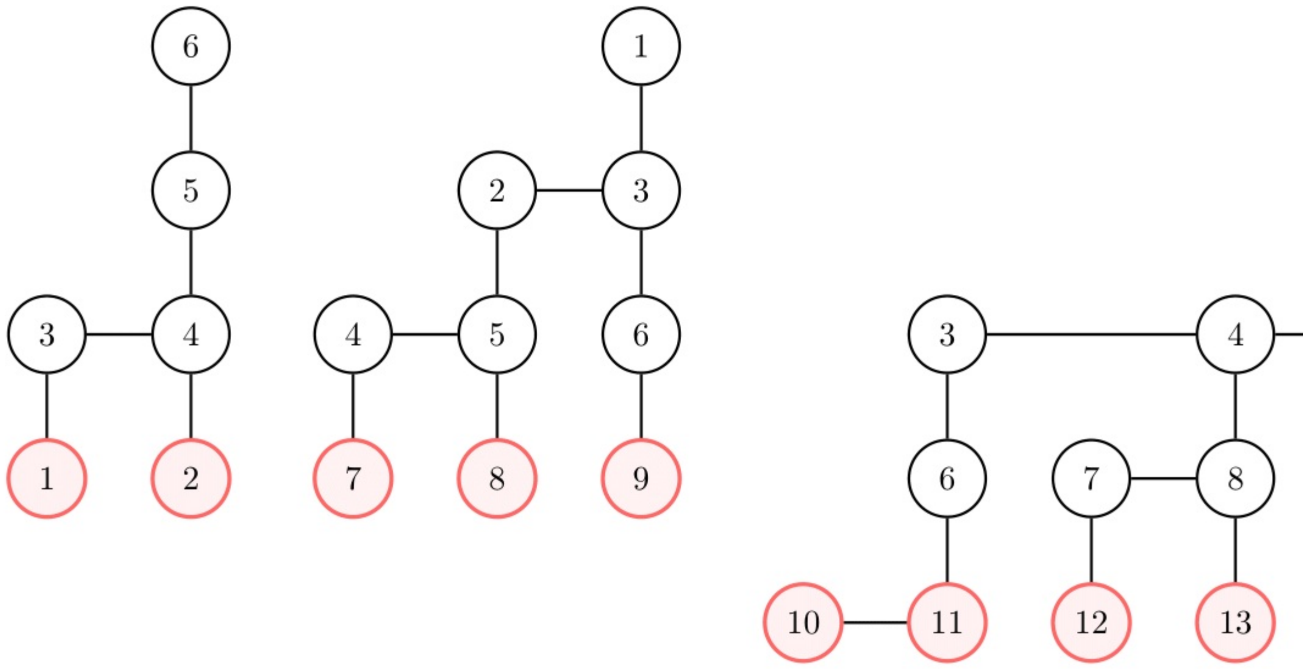
$1 \to 3 \to 2 \to 5 \to 4 \to 7$.

- at the location $1$, the kinetic energy is $0$;
- at the location $3$, the kinetic energy increases by $1$ (because $h_3 < h_1$), so it becomes $1$;
- at the location $2$, the kinetic energy decreases by $1$ (because $h_2 = h_3$), so it becomes $0$;
- at the location $5$, the kinetic energy increases by $1$ (because $h_5 < h_2$), so it becomes $1$;
- at the location $4$, the kinetic energy decreases by $1$ (because $h_4 = h_5$), so it becomes $0$;
- at the location $7$, the kinetic energy increases by $1$ (because $h_7 < h_4$), so it becomes $1$.

There isn't any sequence of trails of length greater than $5$ such that the kinetic energy is always non-negative.

In the third test, for the skier starting from vertex $1$, the optimal path is $1 \to 2 \to 5 \to 4 \to 3 \to 6 \to 11 \to 10 \to 11$.

Here are pictures of the first, second, and third test, with the base lodges shown in red:



## H. Three Minimums

time limit per test: 8 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

Given a list of distinct values, we denote with *first minimum*, *second minimum*, and *third minimum* the three smallest values (in increasing order).

A permutation $p_1, p_2, \ldots, p_n$ is *good* if the following statement holds for all pairs $(l, r)$ with $1 \le l < l + 2 \le r \le n$.

- If $\{p_l, p_r\}$ are (not necessarily in this order) the first and second minimum of $p_l, p_{l+1}, \ldots, p_r$ then the third minimum of $p_l, p_{l+1}, \ldots, p_r$ is either $p_{l+1}$ or $p_{r-1}$.

You are given an integer $n$ and a string $s$ of length $m$ consisting of characters "<" and ">".

Count the number of *good* permutations $p_1, p_2, \ldots, p_n$ such that, for all $1 \le i \le m$,

- $p_i < p_{i+1}$ if $s_i =$ "<";
- $p_i > p_{i+1}$ if $s_i =$ ">".

As the result can be very large, you should print it modulo $998\,244\,353$.

### Input

The first line contains two integers $n$ and $m$ ($2 \le n \le 2 \cdot 10^5$, $1 \le m \le \min(100, n-1)$).

The second line contains a string $s$ of length $m$, consisting of characters "<" and ">".

### Output

Print a single integer: the number of good permutations satisfying the constraints described in the statement, modulo $998\,244\,353$.

**Examples**

**input**
```
5 3
>>>
```
**output**
```
5
```

**input**
```
5 1
<
```
**output**
```
56
```

**input**

```
6 5
<<><>
```

**output**

```
23
```

**input**

```
10 5
><<><
```

**output**

```
83154
```

**input**

```
1008 20
<><<>>><<<<<>>>>>>>>
```

**output**

```
284142857
```

## Note

In the first test, there are $5$ good permutations satisfying the constraints given by the string $s$: $[4, 3, 2, 1, 5]$, $[5, 3, 2, 1, 4]$, $[5, 4, 2, 1, 3]$, $[5, 4, 3, 1, 2]$, $[5, 4, 3, 2, 1]$. Each of them

- is good;
- satisfies $p_1 > p_2$;
- satisfies $p_2 > p_3$;
- satisfies $p_3 > p_4$.

In the second test, there are $60$ permutations such that $p_1 < p_2$. Only $56$ of them are good: the permutations $[1, 4, 3, 5, 2]$, $[1, 5, 3, 4, 2]$, $[2, 4, 3, 5, 1]$, $[2, 5, 3, 4, 1]$ are not good because the required condition doesn't hold for $(l, r)$ = $(1, 5)$. For example, for the permutation $[2, 4, 3, 5, 1]$,

- the first minimum and the second minimum are $p_5$ and $p_1$, respectively (so they are $\{p_l, p_r\}$ up to reordering);
- the third minimum is $p_3$ (neither $p_{l+1}$ nor $p_{r-1}$).

In the third test, there are $23$ good permutations satisfying the constraints given by the string $s$: $[1, 2, 4, 3, 6, 5]$, $[1, 2, 5, 3, 6, 4]$, $[1, 2, 6, 3, 5, 4]$, $[1, 3, 4, 2, 6, 5]$, $[1, 3, 5, 2, 6, 4]$, $[1, 3, 6, 2, 5, 4]$, $[1, 4, 5, 2, 6, 3]$, $[1, 4, 6, 2, 5, 3]$, $[1, 5, 6, 2, 4, 3]$, $[2, 3, 4, 1, 6, 5]$, $[2, 3, 5, 1, 6, 4]$, $[2, 3, 6, 1, 5, 4]$, $[2, 4, 5, 1, 6, 3]$, $[2, 4, 6, 1, 5, 3]$, $[2, 5, 6, 1, 4, 3]$, $[3, 4, 5, 1, 6, 2]$, $[3, 4, 5, 2, 6, 1]$, $[3, 4, 6, 1, 5, 2]$, $[3, 4, 6, 2, 5, 1]$, $[3, 5, 6, 1, 4, 2]$, $[3, 5, 6, 2, 4, 1]$, $[4, 5, 6, 1, 3, 2]$, $[4, 5, 6, 2, 3, 1]$.