Codeforces Round #584 - Dasha Code Championship - Elimination Round (rated, open for everyone, Div. 1 + Div. 2)

A. Paint the Numbers

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

You are given a sequence of integers $a_1, a_2, \dots, a_n$. You need to paint elements in colors, so that:

- If we consider any color, all elements of this color must be divisible by the minimal element of this color.
- The number of used colors must be minimized.

For example, it's fine to paint elements $[40, 10, 60]$ in a single color, because they are all divisible by 10. You can use any color an arbitrary amount of times (in particular, it is allowed to use a color only once). The elements painted in one color do not need to be consecutive.

For example, if $a = [6, 2, 3, 4, 12]$ then two colors are required: let's paint 6, 3 and 12 in the first color (6, 3 and 12 are divisible by 3) and paint 2 and 4 in the second color (2 and 4 are divisible by 2). For example, if $a = [10, 7, 15]$ then 3 colors are required (we can simply paint each element in an unique color).

Input

The first line contains an integer n ($1 \le n \le 100$), where n is the length of the given sequence.

The second line contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 100$). These numbers can contain duplicates.

Output

Print the minimal number of colors to paint all the given numbers in a valid way.

Examples

input
```
6
10 2 3 5 4 2
```
output
```
3
```

input
```
4
100 100 100 100
```
output
```
1
```

input
```
8
7 6 5 4 3 2 2 3
```
output
```
4
```

Note

In the first example, one possible way to paint the elements in 3 colors is:

- paint in the first color the elements: $a_1 = 10$ and $a_4 = 5$,
- paint in the second color the element $a_3 = 3$,
- paint in the third color the elements: $a_2 = 2$, $a_5 = 4$ and $a_6 = 2$.

In the second example, you can use one color to paint all the elements.

In the third example, one possible way to paint the elements in 4 colors is:

- paint in the first color the elements: $a_4 = 4$, $a_6 = 2$ and $a_7 = 2$,
- paint in the second color the elements: $a_2 = 6$, $a_5 = 3$ and $a_8 = 3$,
- paint in the third color the element $a_3 = 5$,
- paint in the fourth color the element $a_1 = 7$.

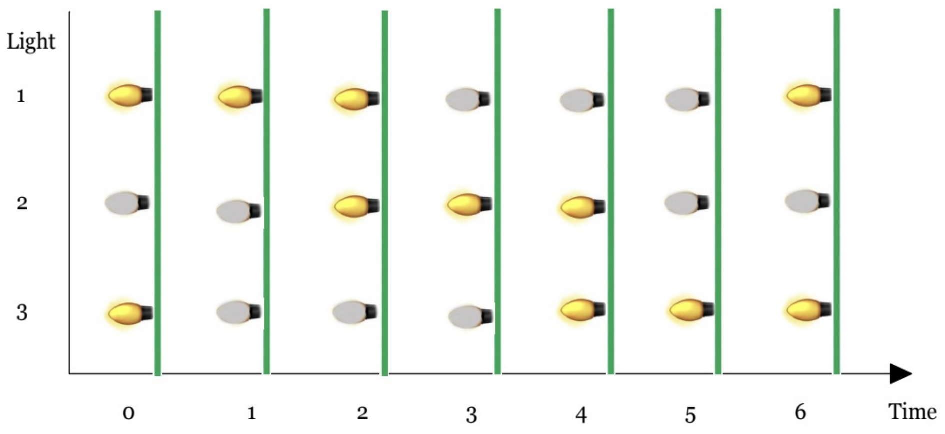
B. Koala and Lights

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

It is a holiday season, and Koala is decorating his house with cool lights! He owns n lights, all of which flash periodically.

After taking a quick glance at them, Koala realizes that each of his lights can be described with two parameters a_i and b_i. Light with parameters a_i and b_i will toggle (on to off, or off to on) every a_i seconds starting from the b_i-th second. In other words, it will toggle at the moments b_i, $b_i + a_i$, $b_i + 2 \cdot a_i$ and so on.

You know for each light whether it's initially on or off and its corresponding parameters a_i and b_i. Koala is wondering what is the maximum number of lights that will ever be on at the same time. So you need to find that out.



Here is a graphic for the first example.

Input

The first line contains a single integer n ($1 \le n \le 100$), the number of lights.

The next line contains a string s of n characters. The i-th character is "1", if the i-th lamp is initially on. Otherwise, i-th character is "0".

The i-th of the following n lines contains two integers a_i and b_i ($1 \le a_i, b_i \le 5$) — the parameters of the i-th light.

Output

Print a single integer — the maximum number of lights that will ever be on at the same time.

Examples

input
```
3
101
3 3
3 2
3 1
```
output
```
2
```

input
```
4
1111
3 4
5 2
3 1
3 2
```
output
```
4
```

input
```
6
011100
5 3
5 5
2 4
3 5
4 2
1 5
```
output
```
6
```

Note

For first example, the lamps' states are shown in the picture above. The largest number of simultaneously on lamps is 2 (e.g. at the moment 2).

In the second example, all lights are initially on. So the answer is 4.

C. Paint the Digits

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

You are given a sequence of n digits $d_1d_2 \dots d_n$. You need to paint all the digits in two colors so that:

- each digit is painted either in the color 1 or in the color 2;
- if you write in a row from left to right all the digits painted in the color 1, and then after them all the digits painted in the color 2, then the resulting sequence of n digits will be non-decreasing (that is, each next digit will be greater than or equal to the previous digit).

For example, for the sequence $d = 914$ the only valid coloring is 211 (paint in the color 1 two last digits, paint in the color 2 the first digit). But 122 is not a valid coloring (9 concatenated with 14 is not a non-decreasing sequence).

It is allowed that either of the two colors is not used at all. Digits painted in the same color are not required to have consecutive positions.

Find any of the valid ways to paint the given sequence of digits or determine that it is impossible to do.

Input

The first line contains a single integer t ($1 \le t \le 10000$) — the number of test cases in the input.

The first line of each test case contains an integer n ($1 \le n \le 2 \cdot 10^5$) — the length of a given sequence of digits.

The next line contains a sequence of n digits $d_1d_2 \dots d_n$ ($0 \le d_i \le 9$). The digits are written in a row without spaces or any other separators. The sequence can start with `0`.

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

Output

Print t lines — the answers to each of the test cases in the input.

If there is a solution for a test case, the corresponding output line should contain any of the valid colorings written as a string of n digits $t_1t_2 \dots t_n$ ($1 \le t_i \le 2$), where t_i is the color the i-th digit is painted in. If there are several feasible solutions, print any of them.

If there is no solution, then the corresponding output line should contain a single character '`-`' (the minus sign).

Example

input

```
5
12
040425524644
1
0
9
123456789
2
98
3
987
```

output

```
121212211211
1
222222222
21
-
```

Note

In the first test case, $d = 040425524644$. The output $t = 121212211211$ is correct because 0022444 (painted in 1) concatenated with 44556 (painted in 2) is 002244444556 which is a sorted sequence of n given digits.

D. Cow and Snacks

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

The legendary Farmer John is throwing a huge party, and animals from all over the world are hanging out at his house. His guests are hungry, so he instructs his cow Bessie to bring out the snacks! Moo!

There are n snacks flavors, numbered with integers $1, 2, \ldots, n$. Bessie has n snacks, one snack of each flavor. Every guest has exactly two favorite flavors. The procedure for eating snacks will go as follows:

- First, Bessie will line up the guests in some way.
- Then in this order, guests will approach the snacks one by one.
- Each guest in their turn will eat all remaining snacks of their favorite flavor. In case no favorite flavors are present when a guest goes up, they become very sad.

Help Bessie to minimize the number of sad guests by lining the guests in an optimal way.

Input

The first line contains integers n and k ($2 \le n \le 10^5$, $1 \le k \le 10^5$), the number of snacks and the number of guests.

The i-th of the following k lines contains two integers x_i and y_i ($1 \le x_i, y_i \le n$, $x_i \ne y_i$), favorite snack flavors of the i-th guest.

Output

Output one integer, the smallest possible number of sad guests.

Examples

input

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

output

```
1
```

input

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

output

```
0
```

Note

In the first example, Bessie can order the guests like this: $3, 1, 2, 4$. Guest 3 goes first and eats snacks 1 and 4. Then the guest 1 goes and eats the snack 2 only, because the snack 1 has already been eaten. Similarly, the guest 2 goes up and eats the snack 3 only. All the snacks are gone, so the guest 4 will be sad.

In the second example, one optimal ordering is $2, 1, 3, 5, 4$. All the guests will be satisfied.

E1. Rotate Columns (easy version)

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

This is an easier version of the next problem. The difference is only in constraints.

You are given a rectangular $n \times m$ matrix a. In one move you can choose any column and cyclically shift elements in this column. You can perform this operation as many times as you want (possibly zero). You can perform this operation to a column multiple times.

After you are done with cyclical shifts, you compute for every row the maximal value in it. Suppose that for i-th row it is equal r_i. What is the maximal possible value of $r_1 + r_2 + \ldots + r_n$?

Input

The first line contains an integer t ($1 \le t \le 40$), the number of test cases in the input.

The first line of each test case contains integers n and m ($1 \le n \le 4$, $1 \le m \le 100$) — the number of rows and the number of columns in the given matrix a.

Each of the following n lines contains m integers, the elements of a ($1 \le a_{i,j} \le 10^5$).

Output

Print t integers: answers for all test cases in the order they are given in the input.

Example

input

```
2
2 3
2 5 7
4 2 4
3 6
4 1 5 2 10 4
8 6 6 4 9 10
5 4 9 5 8 7
```

output

```
12
29
```

Note

In the first test case, you can shift the third column down by one, this way there will be $r_1 = 5$ and $r_2 = 7$.

In the second case you can don't rotate anything at all, this way there will be $r_1 = r_2 = 10$ and $r_3 = 9$.

E2. Rotate Columns (hard version)

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

This is a harder version of the problem. The difference is only in constraints.

You are given a rectangular $n \times m$ matrix a. In one move you can choose any column and cyclically shift elements in this column. You can perform this operation as many times as you want (possibly zero). You can perform this operation to a column multiple times.

After you are done with cyclical shifts, you compute for every row the maximal value in it. Suppose that for i-th row it is equal r_i. What is the maximal possible value of $r_1 + r_2 + \ldots + r_n$?

Input

The first line contains an integer t ($1 \le t \le 40$), the number of test cases in the input.

The first line of each test case contains integers n and m ($1 \le n \le 12$, $1 \le m \le 2000$) — the number of rows and the number of columns in the given matrix a.

Each of the following n lines contains m integers, the elements of a ($1 \le a_{i,j} \le 10^5$).

Output

Print t integers: answers for all test cases in the order they are given in the input.

Example

input

```
3
2 3
2 5 7
4 2 4
3 6
4 1 5 2 10 4
8 6 6 4 9 10
5 4 9 5 8 7
3 3
9 9 9
1 1 1
1 1 1
```

output

```
12
29
27
```

Note

In the first test case you can shift the third column down by one, this way there will be $r_1 = 5$ and $r_2 = 7$.

In the second case you can don't rotate anything at all, this way there will be $r_1 = r_2 = 10$ and $r_3 = 9$.

F. Koala and Notebook

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

Koala Land consists of m bidirectional roads connecting n cities. The roads are numbered from 1 to m by order in input. It is guaranteed, that one can reach any city from every other city.

Koala starts traveling from city 1. Whenever he travels on a road, he writes its number down in his notebook. He doesn't put spaces between the numbers, so they all get concatenated into a single number.

Before embarking on his trip, Koala is curious about the resulting number for all possible destinations. For each possible destination, what is the smallest number he could have written for it?

Since these numbers may be quite large, print their remainders modulo $10^9 + 7$. Please note, that you need to compute the remainder of the minimum possible number, **not** the minimum possible remainder.

Input

The first line contains two integers n and m ($2 \le n \le 10^5, n - 1 \le m \le 10^5$), the number of cities and the number of roads, respectively.

The i-th of the following m lines contains integers x_i and y_i ($1 \le x_i, y_i \le n$, $x_i \ne y_i$), representing a bidirectional road between cities x_i and y_i.

It is guaranteed, that for any pair of cities there is at most one road connecting them, and that one can reach any city from every other city.

Output

Print $n - 1$ integers, the answer for every city except for the first city.

The i-th integer should be equal to the smallest number he could have written for destination $i + 1$. Since this number may be large, output its remainder modulo $10^9 + 7$.

Examples

input

```
11 10
1 2
2 3
3 4
4 5
5 6
6 7
7 8
8 9
9 10
10 11
```

output

```
1
12
123
1234
12345
123456
1234567
12345678
123456789
345678826
```

input

```
12 19
1 2
2 3
2 4
2 5
2 6
2 7
2 8
2 9
2 10
3 11
11 12
1 3
1 4
1 5
1 6
1 7
1 8
1 9
1 10
```

output

```
1
12
13
14
15
16
17
18
19
```

```
1210
121011
```

input

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

output

```
1
12
13
134
1345
13456
1498
149
14
1410
141011
```

G1. Into Blocks (easy version)

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

This is an easier version of the next problem. In this version, $q = 0$.

A sequence of integers is called *nice* if its elements are arranged in blocks like in $[3, 3, 3, 4, 1, 1]$. Formally, if two elements are equal, everything in between must also be equal.

Let's define *difficulty* of a sequence as a minimum possible number of elements to change to get a nice sequence. However, if you change at least one element of value x to value y, you must also change all other elements of value x into y as well. For example, for $[3, 3, 1, 3, 2, 1, 2]$ it isn't allowed to change first 1 to 3 and second 1 to 2. You need to leave 1's untouched or change them to the same value.

You are given a sequence of integers $a_1, a_2, \ldots, a_n$ and q updates.

Each update is of form "i x" — change a_i to x. Updates are not independent (the change stays for the future).

Print the difficulty of the initial sequence and of the sequence after every update.

Input

The first line contains integers n and q ($1 \le n \le 200\,000$, $q = 0$), the length of the sequence and the number of the updates.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 200\,000$), the initial sequence.

Each of the following q lines contains integers i_t and x_t ($1 \le i_t \le n$, $1 \le x_t \le 200\,000$), the position and the new value for this position.

Output

Print $q + 1$ integers, the answer for the initial sequence and the answer after every update.

Examples

input

```
5 0
3 7 3 7 3
```

output

```
2
```

input

```
10 0
1 2 1 2 3 1 1 1 50 1
```

output

```
4
```

input

```
6 0
6 6 3 3 4 4
```

output

```
0
```

input

```
7 0
3 3 1 3 2 1 2
```

output

```
4
```

G2. Into Blocks (hard version)

time limit per test: 5 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

This is a harder version of the problem. In this version $q \leq 200\,000$.

A sequence of integers is called *nice* if its elements are arranged in blocks like in $[3, 3, 3, 4, 1, 1]$. Formally, if two elements are equal, everything in between must also be equal.

Let's define *difficulty* of a sequence as a minimum possible number of elements to change to get a nice sequence. However, if you change at least one element of value x to value y, you must also change all other elements of value x into y as well. For example, for $[3, 3, 1, 3, 2, 1, 2]$ it isn't allowed to change first 1 to 3 and second 1 to 2. You need to leave 1's untouched or change them to the same value.

You are given a sequence of integers $a_1, a_2, \ldots, a_n$ and q updates.

Each update is of form "i x" — change a_i to x. Updates are not independent (the change stays for the future).

Print the difficulty of the initial sequence and of the sequence after every update.

Input

The first line contains integers n and q ($1 \leq n \leq 200\,000$, $0 \leq q \leq 200\,000$), the length of the sequence and the number of the updates.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 200\,000$), the initial sequence.

Each of the following q lines contains integers i_t and x_t ($1 \leq i_t \leq n$, $1 \leq x_t \leq 200\,000$), the position and the new value for this position.

Output

Print $q+1$ integers, the answer for the initial sequence and the answer after every update.

Example

input

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

output

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

H. Moving Walkways

time limit per test: 2.5 seconds

memory limit per test: 256 megabytes

input: standard input

output: standard output

Airports often use moving walkways to help you walking big distances faster. Each such walkway has some speed that effectively increases your speed. You can stand on such a walkway and let it move you, or you could also walk and then your effective speed is your walking speed plus walkway's speed.

Limak wants to get from point 0 to point L on a straight line. There are n disjoint walkways in between. The i-th walkway is described by two integers x_i and y_i and a real value s_i. The i-th walkway starts at x_i, ends at y_i and has speed s_i.

Every walkway is located inside the segment $[0, L]$ and no two walkways have positive intersection. However, they can touch by endpoints.

Limak needs to decide how to distribute his energy. For example, it might make more sense to stand somewhere (or to walk slowly) to then have a lot of energy to walk faster.

Limak's initial energy is 0 and it must never drop below that value. At any moment, he can walk with any speed v in the interval $[0, 2]$ and it will cost him v energy per second, but he continuously recovers energy with speed of 1 energy per second. So, when he walks with speed v, his energy increases by $(1 - v)$. Note that negative value would mean losing energy.

In particular, he can walk with speed 1 and this won't change his energy at all, while walking with speed 0.77 effectively gives him 0.23 energy per second.

Limak can choose his speed arbitrarily (any real value in interval $[0, 2]$) at every moment of time (including the moments when he is located on non-integer positions). Everything is continuous (non-discrete).

What is the fastest time Limak can get from 0 to L?

Input

The first line contains integers n and L ($1 \le n \le 200\,000$, $1 \le L \le 10^9$), the number of walkways and the distance to walk.

Each of the next n lines contains integers x_i, y_i and real value s_i ($0 \le x_i < y_i \le L$, $0.1 \le s_i \le 10.0$). The value s_i is given with at most 9 digits after decimal point.

It's guaranteed, that no two walkways have a positive intersection. The walkways are listed from left to right. That is, $y_i \le x_{i+1}$ for $1 \le i \le n - 1$.

Output

Print one real value, the fastest possible time to reach L. Your answer will be considered correct if its absolute or relative error won't exceed 10^{-9}.

Examples

input

```
1 5
0 2 2.0
```

output

```
3.000000000000
```

input

```
1 5
2 4 0.91
```

output

```
3.808900523560
```

input

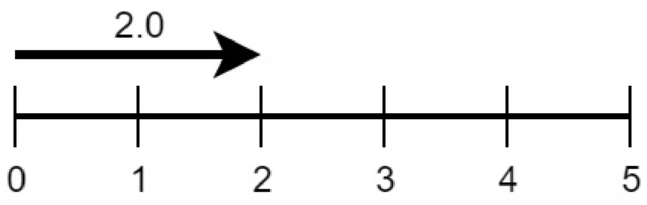
```
3 1000
0 990 1.777777
995 996 1.123456789
996 1000 2.0
```

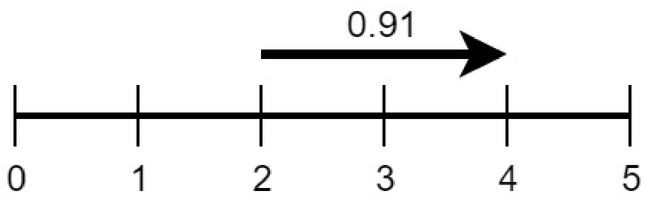
output

```
361.568848429553
```

Note

The drawings show the first two examples. In the first one, there is a walkway from 0 to 2 with speed 2.0 and Limak wants to get to point 5. The second example has a walkway from 2 to 4 with speed 0.91.





In the first example, one of optimal strategies is as follows.

- Get from 0 to 2 by standing still on the walkway. It moves you with speed 2 so it takes 1 second and you save up 1 energy.
- Get from 2 to 4 by walking with max speed 2 for next 1 second. It takes 1 second again and the energy drops to 0.
- Get from 4 to 5 by walking with speed 1. It takes 1 second and the energy stays constant at the value 0.

The total time is $1 + 1 + 1 = 3$.