Problem A. Ascent Sequences

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Consider a sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of non-negative integers. An *ascent* in the sequence is a pair of adjacent elements such that the element with greater index has greater value. For example, there are two ascents in sequence $\langle 0, 2, 3, 1, 0 \rangle$: $a_1 = 0$ to $a_2 = 2$, and $a_2 = 2$ to $a_3 = 3$. Let us denote the number of ascents among the first k elements of the sequence by A_k. In the given example, $A_1 = 0$, $A_2 = 1$, $A_3 = 2$, $A_4 = 2$ and $A_5 = 2$.

Sequence a is called an *ascent sequence* if $a_1 = 0$ and for every $i \geq 2$ inequality $a_i \leq A_{i-1} + 1$ is satisfied. For example, sequence $\langle 0, 2, 3, 1, 0 \rangle$ is not an ascent sequence because $a_2 = 2$ and $A_1 = 0$. Sequence $\langle 0, 1, 0, 2, 3 \rangle$ is, in turn, an ascent sequence because $A_1 = 0$, $A_2 = 1$, $A_3 = 1$, $A_4 = 2$.

Sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of non-negative integers *avoids pattern 201* if there are no i, j and k such that $i < j < k$ and $a_j < a_k < a_i$. For example, sequence $\langle 0, 1, 0, 2, 3 \rangle$ avoids pattern 201, while $\langle 0, 1, 2, 3, 1, 0, 2 \rangle$ does not avoid pattern 201 because for $i = 4$, $j = 6$, $k = 7$ we have $a_j = 0 < a_k = 2 < a_i = 3$.

You are given two integers n and p. Find the number of ascent sequences of length n avoiding pattern 201, and output this number modulo p.

Input

The only line of the input contains two integers n and p ($1 \leq n \leq 500$; $2 \leq p \leq 10^9 + 123$; p is a prime).

Output

Output a single integer — the number of ascent sequences of length n avoiding pattern 201, modulo p.

Examples

standard input	standard output
3 23	5
5 239	52

Note

In the first example test case, there are five ascent sequences of length 3 avoiding pattern 201: $\langle 0, 0, 0 \rangle$, $\langle 0, 0, 1 \rangle$, $\langle 0, 1, 0 \rangle$, $\langle 0, 1, 1 \rangle$, $\langle 0, 1, 2 \rangle$.

In the second example test case, there are 53 ascent sequences of length 5 and all of them except $\langle 0, 1, 2, 0, 1 \rangle$ avoid pattern 201.

Problem B. Buggy Combination Lock

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

You're a full-time bomb defuser. Your duties include keeping talking and not letting anyone explode.

This time, you're stumbled upon the last obstacle before defusing the bomb — a combination lock. The lock contains n rotating discs, with each of the discs containing all integers between 0 and $m-1$, inclusive, in increasing order. One forward rotation of a disc causes the number on it to increase by one (except when the number on the disc is $m-1$, it changes to 0). Similarly, one backward rotation of a disc causes the number on it to decrease by one (except when the number on the disc is 0, it changes to $m-1$).

You see the initial state of the lock, and you perfectly know the code combination which opens the lock. Unfortunately, the lock is buggy. Whenever you rotate the i-th disc forward or backward, the $i+1$-th disc gets rotated in the same direction as well. Similarly, whenever you rotate the n-th disc, the first disc gets rotated in the same direction too.

On one hand, this might help you open the lock sooner; on the other hand, this might prevent you from opening the lock at all; who knows? Well, of course you do. Find the minimum number of disc rotations you need to perform to open the lock, or determine that it's impossible (and someone is about to explode).

Input

The first line of the input contains two integers n and m ($2 \le n \le 2 \cdot 10^5$; $2 \le m \le 10^9$) — the number of discs in the lock and the range of numbers on the discs, respectively. The second line contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i < m$) — the initial numbers on the first, second, ..., n-th disc. The third line contains n integers $b_1, b_2, \ldots, b_n$ ($0 \le b_i < m$) — the target numbers on the first, second, ..., n-th disc.

Output

If it's impossible to open the lock, output -1. Otherwise, output a single integer — the minimum number of disc rotations you need to perform to open the lock.

Examples

standard input	standard output
6 3 0 1 0 1 0 1 1 0 1 0 1 0	4
3 7 4 2 1 1 2 5	3
2 10 7 7 3 5	-1

Note

In the first example test case, one possible solution is to rotate the first and the second discs forward, both once, and the fourth and the fifth discs backward, both once.

In the second example test case, the fastest way to open the lock is to rotate the third disc backward three times.

In the third example test case, whichever disc you rotate, the numbers on the discs will always remain equal.

Problem C. Cyclic Shifts

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Let $s = s_1 s_2 \dots s_k$ be a string of length k. For any integer i between 0 and $k-1$, inclusive, we define the i-th *cyclic shift* of s as the string $s_{i+1} s_{i+2} \dots s_k s_1 \dots s_i$. For example, the 4-th cyclic shift of "`wellplayed`" is "`playedwell`", while the 0-th cyclic shift of "`metro`" is "`metro`" itself.

Let's define a function $f(s)$ which, for a string of length k, is equal to i such that the i-th cyclic shift of s is the lexicographically smallest among all its cyclic shifts. If there are several such i's, then $f(s)$ is equal to the smallest of them. For example, $f($"`acabbac`"$) = 2$, while $f($"`cabcab`"$) = 1$.

Let's define a function $g(s)$ which, for a string of length n, is equal to the sum of $f(s_1 s_2 ... s_k) \cdot 8753^{k-1}$ over all k between 1 and n, inclusive.

For each given string s, find the value of $g(s)$ modulo $10^9 + 123$.

Input

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

Each of the next t lines contains a non-empty string consisting of lowercase English letters.

The total length of the input strings doesn't exceed 10^6.

Output

For each string s in order of input, output a single integer — the value of $g(s)$ modulo $10^9 + 123$.

Example

standard input	standard output
2 aab acabbac	0 38098220

Note

In the first example test case, $f($"`a`"$) = 0$, $f($"`aa`"$) = 0$, and $f($"`aab`"$) = 0$. Therefore, $g($"`aab`"$) = 0$.

Here is the list of values of $f(s_1 s_2 \dots s_k)$ for the second example test case:

- $f($"`a`"$) = 0$;
- $f($"`ac`"$) = 0$;
- $f($"`aca`"$) = 2$;
- $f($"`acab`"$) = 2$;
- $f($"`acabb`"$) = 2$;
- $f($"`acabba`"$) = 5$;
- $f($"`acabbac`"$) = 2$.

Thus, $g($"`acabbac`"$) = 2 \cdot 8753^2 + 2 \cdot 8753^3 + 2 \cdot 8753^4 + 5 \cdot 8753^5 + 2 \cdot 8753^6 = 899695598935764095704157$, which is equal to 38098220 modulo $10^9 + 123$.

Problem D. Distribution of Prize Money

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

The most famous Zurumbian programming contest, Zurumbia Open, has been recently held online. The total prize fund of the contest is x Zurumbian roubles!

There were n contestants taking part, and each of them was ranked from 1-st to n-th. No two contestants had the same rank.

Unfortunately, the distribution of prize money among the contestants is not known yet. It's only known that every contestant will receive a non-negative integer amount of Zurumbian roubles, and no contestant will receive less money than another contestant with greater rank.

Some of the contestants are your friends. You've decided to find the least possible part of the prize fund which will be definitely received by them in total, among all valid distributions of prize money. Also, you are interested in the *worst* distribution — that is, the one that leads to the least possible amount of money received by your friends. Write a program to help yourself.

Input

The first line of the input contains a single integer x ($1 \le x \le 10^9$) — the total prize fund, in Zurumbian roubles. The second line contains a string of length n ($1 \le n \le 400$) consisting of uppercase English letters "`Y`" and "`N`" only. The i-th (1-based) character of the string equals to "`Y`" if the contestant ranked i-th is your friend, and "`N`" otherwise.

Output

Output two lines.

The first line must contain the least possible amount of money, in Zurumbian roubles, which will be definitely received by your friends in total.

The second line must contain a non-increasing sequence of n non-negative integers — the prizes of the contestants ranked first, second, ..., n-th in the worst distribution of prize money. The sum of these n integers must be equal to x. If there are several worst distributions, you may output any of them.

Example

standard input	standard output
23 YNYNNYY	10 7 7 3 3 3 0 0

Note

In the distribution from the example test case, your friend ranked first will receive 7 Zurumbian roubles, the third-ranked one will receive 3 Zurumbian roubles, and two other friends will receive nothing, resulting in the grand total of 10 Zurumbian roubles received by your friends. It can be proved that in any distribution of prize money, your friends will always receive at least 10 Zurumbian roubles.

Problem E. Exclusive Training

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	512 mebibytes

You're managing "Squash It", an elite squash club. There are n players in your club, the i-th of them has club rating r_i and pleasantness p_i.

You're looking to conduct an exclusive training with one of your players as the leader and several other lower-rated players as the learners. Let's number the upcoming days in such a manner that tomorrow has index 1, the day after tomorrow has index 2, and so on. Being informed about your plans, each player provided a range of day indices $[a_i; b_i]$ such that he will be able to attend the training only if it is held on any of the days between a_i and b_i, inclusive.

Not only do you want the training to be useful for the attendees, but also you want it to be as pleasant as possible. Of course, it will be too harsh to tell someone he's not invited simply because he's not pleasant enough. Instead, you can set an *upper bound* on the rating of the invitees, announcing that this training is designed for low-rated players.

For each player i, you'd like to find a way to organize the training in such a way that the total pleasantness of the invited players is as high as possible. You're able to choose any day between a_i and b_i, inclusive, and any rating bound lower than r_i. Then, you will invite everyone who can attend the training on the chosen day and whose rating is not higher than the chosen upper bound (and, of course, the i-th player himself). Note that you can even set the upper bound on rating to 0 — in this case, no one except the leader will be invited, but sometimes that may be more pleasant than inviting *that rude guy*.

Input

The first line of the input contains a single integer n ($1 \le n \le 2 \cdot 10^5$) — the number of players in "Squash It". Each of the next n lines contains four integers r_i, p_i, a_i and b_i ($1 \le r_i \le 10^6$; $-10^6 \le p_i \le 10^6$; $1 \le a_i \le b_i \le 10^6$) — the club rating, the pleasantness and the range of day indices of the i-th player, respectively.

All player ratings are pairwise distinct.

Output

For each player in order of input, output a single line containing the maximum possible total pleasantness of training invitees in case this player is chosen to be the leader of the training.

Example

standard input	standard output
4 15 22 2 5 13 -7 5 8 9 -5 3 7 12 13 5 6	30 1 -5 13

Note

In the example test case, if the training is held on day 5 with the first player as the leader and everyone with rating not exceeding 12 invited, then the first, the third and the fourth players will receive an invitation, resulting in the total invitee pleasantness of 30.

Problem F. Fruit Game

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Apfelmann and Bananenfrau are good friends. Today, Apfelmann brought some apples, and Bananenfrau brought some bananas. Also, they found a coconut.

The friends decided to play a fruit game. They put their apples, bananas and the only coconut in a row on a table. Players take turns, Apfelmann is the first to make a move.

An apple or a banana is considered *tasty* if and only if there is no other fruit lying on the table between this apple or banana and the coconut.

On his turn, Apfelmann must take a tasty apple from the table and eat it. If there are no tasty apples at the moment, Apfelmann skips his turn. Similarly, on her turn, Bananenfrau must take a tasty banana from the table and eat it. If there are no tasty bananas at the moment, Bananenfrau skips her turn.

The player who eats all his or her fruit before the opponent does the same is declared to be the winner of the game.

Given the initial placement of the fruit on the table, determine who will win the game if both players play perfectly and strive to win.

Input

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

Each of the next t lines contains a string consisting of uppercase English letters "`A`", "`B`" and "`C`" — the initial placement of the fruit in the row, in order from left to right. "`A`" stands for an apple, "`B`" stands for a banana, and "`C`" stands for the coconut. There is at least one apple, at least one banana and exactly one coconut in the row.

The total length of the input strings doesn't exceed 10^6.

Output

For each test case, output a single line containing the name of the winner of the game.

Example

standard input	standard output
3 AAACBB CABAB BBBBCBA	Bananenfrau Apfelmann Bananenfrau

Note

In the first example test case, Apfelmann and Bananenfrau take turns eating apples and bananas from different sides of the coconut. There are less bananas than apples, therefore, Bananenfrau wins.

In the second example test case, the players eat apples and bananas in turns from left to right. Apfelmann finishes with apples first.

In the third example test case, Apfelmann has to skip his first turn. Then, Bananenfrau has a choice to eat either the banana to the left or the banana to the right of the coconut. Once Bananenfrau eats the rightmost banana, Apfelmann will eat the only apple in the next turn and win. It's better for Bananenfrau to eat all the bananas to the left of the coconut first, one by one, making Apfelmann skip his turns. After that Bananenfrau will be able to win by eating the rightmost banana.

Problem G. Greedy Coach

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

A certain contest system contains n problemsets, numbered from 1 to n. Every problemset can be used for exactly one training. Sometimes, the coach wants to hold a training for a team of three of his students. For that, the coach has to assign a problemset for the training such that none of the students on the team took part in a training with the same set before. In case this is impossible, the training is not conducted.

Consider the following two greedy strategies of assigning a problemset to a team of students for training.

The first strategy, which we'll call *strategy A*, is to assign the lowest-numbered problemset which hasn't yet been seen by any of the students on the team.

The second strategy, which we'll call *strategy B*, is to assign the problemset which has been seen by the largest number of students in total during the previous trainings and, at the same time, has not yet been seen by any of the students on the team. In case there are several such problemsets, strategy B is to assign the lowest-numbered of those.

We want to find out if one of the strategies is strictly better than the other. First, you're asked to report there's none or find such integer n and such a sequence of team formations that strategy A allows to assign problemsets for all trainings, while strategy B doesn't allow to assign a problemset for at least one training. Second, the same question is asked with regard to a situation where strategy B allows to assign problemsets for all trainings while strategy A doesn't.

Input

The only line of the input contains a single integer t ($0 \le t \le 2$). If $t = 0$, you're allowed to output either -1 or any valid sequence of team formations. If $t = 1$, you have to find a situation where strategy A allows to assign problemsets for all trainings, while strategy B doesn't. If $t = 2$, you have to find a situation where strategy B allows to assign problemsets for all trainings, while strategy A doesn't.

Test case 1 has $t = 0$, test case 2 has $t = 1$, and test case 3 has $t = 2$.

Output

If the required situation is impossible, output a single integer -1. Otherwise, output two integers n and q ($1 \le n \le 100$; $1 \le q \le 10^4$) — the number of problemsets and the number of trainings. Each of the following q lines must contain three non-empty strings a_i, b_i and c_i — the identificators of students on the team. These lines should correspond to teams going for a training in chronological order. Student identificators must consist of lowercase English letters and digits. Different identificators must belong to different students, and each team must consist of three different students. It's guaranteed that if the required situation is possible, there also exists one with $n \le 100$ and $q \le 10^4$.

Example

standard input	standard output
0	5 4 eatmore maxbuzz winger cerealguy eatmore niyaznigmatul cerealguy niyaznigmatul tourist qwerty787788 tourist vartem

Note

In the example test case, if the coach follows any of the two strategies, the first and the third trainings will be held on problemset 1, while the second and the fourth trainings will be held on problemset 2.

Problem H. Highly Composite Permutations

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Positive integer x is called *composite* if it has strictly more than two positive integer divisors. For example, integers 4, 30 and 111 are composite, while 1, 7 and 239 are not.

Integer sequence $p = \langle p_1, p_2, \ldots, p_n \rangle$ is called a *permutation of length* n if it contains every integer between 1 and n, inclusive, exactly once.

We'll call permutation $p = \langle p_1, p_2, \ldots, p_n \rangle$ *highly composite* if for every i between 1 and n, inclusive, the sum of the first i elements of p (that is, $p_1 + p_2 + \ldots + p_i$) is composite.

Given a single integer n, find a highly composite permutation of length n.

Input

The only line of the input contains a single integer n ($1 \le n \le 100$).

Output

If no highly composite permutation of length n exists, output a single integer -1. Otherwise, output n integers $p_1, p_2, \ldots, p_n$ such that $p = \langle p_1, p_2, \ldots, p_n \rangle$ is a highly composite permutation.

If there are multiple highly composite permutations of length n, you may output any of them.

Examples

standard input	standard output
13	9 13 6 5 3 2 8 4 1 12 11 10 7
2	-1

Note

In the first example test case, the first element of the permutation, 9, is composite, the sum of the first two elements of the permutation, $9 + 13 = 22$, is composite, the sum of the first three elements of the permutation, $9 + 13 + 6 = 28$, is composite, and so on.

In the second example test case, only two permutations of the required length exist, $\langle 1, 2 \rangle$ and $\langle 2, 1 \rangle$, and neither of them is highly composite.

Problem I. Inversions in Lexicographical Order

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

An *inversion* in permutation $p = \langle p_1, p_2, \ldots, p_n \rangle$ is a pair of integers (i, j) such that $i < j$ and $p_i > p_j$.

Consider a lexicographical order on positive integers. Under this ordering, integers are compared lexicographically as strings of digits. For example, 628 comes before 7, 239 comes before 271, and 42 comes before 427.

You are given a single positive integer n. Let's sort all integers from 1 to n, inclusive, in lexicographical order. We'll get a permutation p of length n, where p_1 is the lexicographically smallest integer between 1 and n (actually, $p_1 = 1$ for any n), p_2 is the second smallest one, and so on.

How many inversions does p contain?

Input

The only line of the input contains a single positive integer n without leading zeroes.

The value of n will be between 1 and $10^{250\,000} - 1$, inclusive. That is, n will consist of no more than 250 000 decimal digits.

Output

Output a single integer without leading zeroes — the number of inversions in p.

Examples

standard input	standard output
11	16
427	26576

Note

Indeed, in the first example test case, $p = \langle 1, 10, 11, 2, 3, 4, 5, 6, 7, 8, 9 \rangle$ contains 16 inversions.

Problem J. Jogging in the Park

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

The Outer Park is very well suited for jogging. Inside the park, there are n glades, conveniently numbered from 1 to n. The glades are connected by m trails, the i-th trail connects glades a_i and b_i and has length c_i meters. All trails can be walked in both directions. The main entrance is situated at glade 1.

Your friends decided to have a nice run together over the park. Each of them prepared her own route starting from the entrance and moving to subsequent glades by trails.

At the end of jogging, all your friends want to reach glade n with a beautiful cafe at the same time. Not everyone planned her route accordingly, though: some of the routes do not end at glade n, and the routes might have different length as well. Luckily, all your friends run at the same speed.

You decided to help your friends and write a program to provide each of them with an *extended* route — that is, a route which starts with the same sequence of glades as her original one, but ends at glade n and has exactly the same length, in meters, as all the other extended routes. Note that both the original and the extended routes may contain the same trail multiple times.

Input

The first line of the input contains two integers n and m ($2 \le n \le 50$; $1 \le m \le \frac{n(n-1)}{2}$) — the number of glades and trails in the park. Each of the next m lines contains three integers a_i, b_i and c_i ($1 \le a_i, b_i \le n$; $a_i \ne b_i$; $1 \le c_i \le 10^6$) — the indices of glades connected by the i-th trail and its length, respectively.

The next line of the input contains a single integer k ($2 \le k \le 50$) — the number of your jogging friends. Each of the next k lines contains an integer l_i ($2 \le l_i \le 50$) — the number of glades in the route of your i-th friend, followed by l_i integers $g_{i,1}, g_{i,2}, \ldots, g_{i,l_i}$ ($1 \le g_{i,j} \le n$) — the indices of glades in the route.

All unordered pairs (a_i, b_i) are distinct. For every i, $g_{i,1} = 1$. For every j between 1 and $l_i - 1$, there exists a trail between glades $g_{i,j}$ and $g_{i,j+1}$.

Output

If it is impossible to extend all the routes according to the problem statement, output a single integer -1. Otherwise, print k lines. The i-th of these lines must contain an integer p_i ($p_i \ge l_i$) followed by p_i integers $h_{i,1}, h_{i,2}, \ldots, h_{i,p_i}$ ($1 \le h_{i,j} \le n$) — the indices of glades in the i-th extended route.

For every i, h_{i,p_i} must be equal to n. For every j between 1 and $p_i - 1$, there must exist a trail between glades $h_{i,j}$ and $h_{i,j+1}$. For every j between 1 and l_i, $h_{i,j}$ must be equal to $g_{i,j}$. The total length of all the extended routes, in meters, must be the same.

The sum of all p_i must not exceed $2 \cdot 10^6$. It is guaranteed that if a valid route extension exists, there also exists one with the sum of all p_i not exceeding $2 \cdot 10^6$.

Example

standard input	standard output
4 4 1 2 10 2 3 30 2 4 30 1 4 100 3 2 1 2 3 1 2 3 2 1 4	5 1 2 4 2 4 5 1 2 3 2 4 2 1 4

Problem K. Keep Distance

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	512 mebibytes

You are participating in an infamous TV-show. Show participants are usually asked to do ridiculous stuff, and you're not an exception.

Right now, there are several colored cubes arranged in a single row in front of you. Your quest goes as follows. The show host announces her favorite color among those present on the cubes. Then, you have to rearrange the cubes as fast as possible in such a way that the row becomes *beautiful* with regard to the host's favorite color.

Let's number the cubes in the row with $1, 2, \ldots$ from left to right. Suppose there are k cubes of color c in the row, namely, at positions $p_1 < p_2 < \ldots < p_k$. The row is called *beautiful* with regard to color c if the neighboring cubes of this color are placed at equal distances from each other, that is, if the following equation is satisfied: $p_2 - p_1 = p_3 - p_2 = \ldots = p_k - p_{k-1}$. Note that if $k \le 2$, the row is beautiful with regard to color c as well.

During the rearrangement of the cubes, you're only allowed to perform the following operation any number of times: take out any two cubes of different colors and exchange their positions in the row. Note that these two cubes do not have to be adjacent.

You already see the row of the cubes but you don't know the favorite color of the host. Now that there's still some time before the ultimate quest, you decided to calculate the minimum number of cube swaps required to make the row beautiful with regard to every present color.

Input

The first line of the input contains a single integer t ($1 \le t \le 2\,500$) — the number of test cases.

Each of the next t lines contains a non-empty string consisting of lowercase English letters, representing the colors of the cubes in the row in order from left to right. Here, "`a`" stands for amaranth, "`b`" stands for bittersweet, "`c`" stands for crimson, and so on.

The total length of the input strings doesn't exceed 250 000.

Output

For each input row, output a sequence of integers. For every color in order from "`a`" to "`z`" present in the row, output the minimum number of swaps required to make the row beautiful with regard to this color.

Example

standard input	standard output
1 vvrrcvrvvr	0 1 2

Note

In the example test case, the row is already beautiful with regard to crimson ("`c`"). To make the row beautiful with regard to red ("`r`"), it's enough to swap the first and the third cubes — the row will look like "`rvvrcvrvvr`". To make the row beautiful with regard to vermilion ("`v`"), it's enough, for example, to swap the first and the fifth cubes, and then swap the second and the seventh cubes — the row will look like "`crrrvvvvvr`".