Problem A. No Thanks!

Source file name:	Nothanks.c, Nothanks.cpp, Nothanks.java, Nothanks.py
Input:	`Standard`
Output:	`Standard`

In the card game "No Thanks," the deck of cards consists of 36 cards numbered 1–36, and players collect cards to their score pile as the game is played. A player's final score is the sum of the numbers on their collected cards, with one exception: if a player has collected any cards with two or more consecutive numbers, only the smallest number of that group counts toward the score. Your job is to compute the score for a single player's pile of cards, though here we allow play with a deck much larger than 36 cards.

Input

The first line contains one integer, n, representing the number of cards collected. The second line contains n integers representing the numbers on the collected cards. You may assume that $1 \leq n \leq 9 \cdot 10^4$, all card values are in the range $1 \ldots 9 \cdot 10^4$ inclusive, and no card value is repeated.

Output

Output a single line containing the score for the given set of cards

Example

Input	Output
5 1 7 5 3 4	11
6 2 1 3 8 4 5	9

Problem B. Exam Manipulation

Source file name:	Exam.c, Exam.cpp, Exam.java, Exam.py
Input:	`Standard`
Output:	`Standard`

A group of students is taking a True/False exam. Each question is worth one point. You, as their teacher, want to make your students look as good as possible–so you cheat! (I know, you would never actually do that.) To cheat, you manipulate the answer key so that the lowest score in the class is as high as possible.

What is the best possible lowest score you can achieve?

Input

The first line of input contains two integers n ($1 \leq n \leq 10^3$) and k ($1 \leq k \leq 10$), where n is the number of students, and k is the number of True/False questions on the exam.

Each of the next n lines contains a string of length k, consisting only of upper-case 'T' and uppercase 'F'. This string represents the answers that a student submitted, in the order the questions were given.

Output

Output, on a single line, the best possible lowest score in the class.

Example

Input	Output

`5 4`
`TFTF`
`TFFF`
`TFTT`
`TFFT`
`TFTF`
	`2`

`3 5`
`TFTFT`
`TFTFT`
`TFTFT`
	`5`

Problem C. Painted Corridors

Source file name:	Painted.c, Painted.cpp, Painted.java, Painted.py
Input:	`Standard`
Output:	`Standard`

The Institute of Colorfully Painted Corridors is planning the construction of a new building. The building has numerous junctions, and corridors that each connect a pair of junctions. The corridors will be painted by amazing new painting robots that drive along the corridors and paint all the walls as they go. The architect has specified the colors of some of the corridors, which may be red, orange, yellow, green, blue, or purple. However, there is only a budget for three painting robots, so there will be a single robot for each primary color (red, yellow, or blue). In addition, these robots are the cheapest possible version, and cannot turn their paint sprayer off (though they can go as fast or as slow as desired with no problems; they can even stop moving entirely).

If a corridor needs to be painted a secondary color (orange, green, or purple), in order for the paints to mix properly, the two robots with the appropriate primary colors must travel down the corridor in the same direction at the same time to create the correct color. The color mixing rules are: $orange = red + yellow$, $green = yellow + blue$, and $purple = red + blue$. A corridor that is unspecified in the plan may be painted any color, or left unpainted.

Corridors may be painted multiple times, provided that each time they are painted with the correct color. Corridors with no specified color can be painted multiple times with different colors. All corridors can be travelled along in both directions. The robots may end up at any junctions after painting all the corridors.

Given the architect's design, is it possible for the painting robots to paint the corridors the desired colors?

Input

The first line of input contains five integers, n ($2 \leq n \leq 100$), m $\left(1 \leq m \leq \frac{n \cdot (n-1)}{2}\right)$, r, b and y ($1 \leq r, b, y \leq n$), where n is the number of junctions, m is the number of corridors, and r, b and y are the initial junctions of the red, blue, and yellow painting robots respectively. Junctions are numbered 1 through n. Each of the next m lines contains two integers i, j ($1 \leq i < j \leq n$), and a single character c which is one of `R`, `O`, `Y`, `G`, `B`, `P`, `X`. The integers i, j indicate that there is a corridor between junction i and junction j, with c indicating the desired color. (`R`, `O`, `Y`, `G`, `B`, `P`, `X`, corresponding to Red, Orange, Yellow, Green, Blue, Purple, and Unspecified, respectively.) There is at most one corridor between each pair of junctions.

Output

Output a single integer, 1 if it is possible to paint the corridors as described and 0 otherwise.

Example

Input	Output

`6 5 1 2 5`
`1 3 X`
`2 3 X`
`3 4 P`
`4 5 X`
`4 6 Y`
	`1`

`6 5 1 2 5`
`1 3 X`
`2 3 X`
`3 4 O`
`4 5 X`
`4 6 Y`
	`0`

`6 5 1 2 5`
`1 3 X`
`2 3 X`
`3 4 P`
`4 5 X`
`4 6 G`
	`1`

Problem D. Basic Basis

Source file name:	Basic.c, Basic.cpp, Basic.java, Basic.py
Input:	`Standard`
Output:	`Standard`

You are given a sequence of n bit strings $b_1, b_2, \ldots, b_n$, each with $k \times 4$ bits.

You are also given another sequence of m bit strings $a_1, a_2, \ldots, a_m$, each also with $k \times 4$ bits.

Let $f(x)$ denote the minimum index i such that it is possible to take a non-empty subset of $b_1, b_2, \ldots, b_i$, XOR them all together, and get x. If there is no such index, $f(x) = -1$.

Print the values $f(a_1), f(a_2), \ldots, f(a_m)$.

Input

The first line of input contains three integers n $(1 \leq n \leq 1000)$, m $(1 \leq m \leq 1000)$ and k $(1 \leq k \leq 40)$, where n is the length of sequence b, m is the length of sequence a, and the elements of both sequences are bit strings with $k \times 4$ bits.

Each of the next n lines contains a hexadecimal representation of b_i as a string of length k. The strings consist only of hexadecimal digits ('0'-'9' and 'a'-'f').

Then, each of the next m lines contains a hexadecimal representation of a_i in the same format as above.

Output

Output m lines with a single integer on each line, where the integer on the ith line is $f(a_i)$.

Example

Input	Output
3 5 2 02 e1 fa 02 e3 1b e1 ff	1 2 3 2 -1

5 6 2
01
02
04
08
10
01
02
03
04
05
64
	1
	2
	2
	3
	3
	-1

Problem E. Bitonic Ordering

Source file name:	Bitonic.c, Bitonic.cpp, Bitonic.java, Bitonic.py
Input:	`Standard`
Output:	`Standard`

Noah suggests the following card game: You are given a deck of cards, each with a distinct positive integer value written on it. The cards are shuffled and placed in a row. Your objective is to arrange the cards in the row so that the values are monotonically increasing initially, and then monotonically decreasing for the remainder of the sequence.

The only move that is allowed is that two neighboring cards may swap positions. Cards may only swap positions if they are adjacent to each other.

Note that in the final ordered sequence, the initial increasing portion of the sequence may be empty (such that the whole sequence is in descending order). Likewise it is allowed for the decreasing portion of the sequence to be empty.

What is the fewest number of moves needed to get the cards arranged in the proper order?

Input

The first line of input contains a single integer n ($1 \le n \le 3 \cdot 10^5$), which is the number of cards.

Each of the next n lines contains a single integer c ($1 \le c \le 10^9$). These are the cards, in their initial order. They will all be distinct.

Output

Output a single integer, which is the fewest number of moves needed to arrange the cards as specified.

Example

Input	Output
8 7 4 8 10 1 2 6 9	7

Problem F. Derangement Rotations

Source file name:	Derangement.c, Derangement.cpp, Derangement.java, Derangement.py
Input:	`Standard`
Output:	`Standard`

A Derangement is a permutation p of $1, 2, \ldots, n$ where $p_i \neq i$ for all i from 1 to n.

A rotation of a sequence a_1, a_2, …, a_n with offset k ($1 \leq k \leq n$) is equal to the sequence a_k, a_{k+1}, …, a_n, a_1, a_2, …, a_{k-1}. A sequence of length n has at most n distinct rotations.

Given a derangement D, let $f(D)$ denote the number of distinct rotations of D that are also derangements. For example, $f([2, 1]) = 1$, $f([3, 1, 2]) = 2$.

Given n and a prime number p, count the number of derangements D of $1, 2, \ldots, n$ such that $f(D) = n-2$, modulo p.

Input

The single line of input contains two integers n ($3 \leq n \leq 10^6$) and p ($10^8 \leq p \leq 10^9 + 7$), where n is a permutation size, and p is a prime number.

Output

Output a single integer, which is the number of derangements D of size n with $f(D) = n - 2$, modulo p.

Example

Input	Output
3 1000000007	0
6 999999937	20

Problem G. Ant Typing

Source file name:	Anttyping.c, Anttyping.cpp, Anttyping.java, Anttyping.py
Input:	`Standard`
Output:	`Standard`

Consider a configurable keyboard where keys can be moved about. An ant is walking on the top row of this keyboard and needs to type a numeric string. The ant starts on the leftmost key of the top row, which contains 9 keys, some permutation of the digits from 1 to 9. On a given second, the ant can perform one of three operations:

1. Stay on that key. The digit corresponding to that key will be entered.
2. Move one key to the left. This can only happen if the ant is not on the leftmost key.
3. Move one key to the right. This can only happen if the ant is not on the rightmost key.

Compute the minimum number of seconds needed for the ant to type out the given numeric string, over all possible numeric key permutations.

Input

The single line of input contains a single string s ($1 \leq |s| \leq 10^5$) consisting only of numeric digit characters from 1 to 9. This is the numeric string that the ant needs to type.

Output

Output a single integer, which is the minimum number of seconds needed for the ant to type out the given numeric string, over all possible numeric key permutations.

Example

Input	Output
`78432579`	`20`

Problem H. Dominating Duos

Source file name:	Dominating.c, Dominating.cpp, Dominating.java, Dominating.py
Input:	`Standard`
Output:	`Standard`

A group of people are standing in a line. Each person has a distinct height. You would like to count the number of unordered pairs of people in the line such that they are taller than everyone in between them in the line.

More formally, let d be a sequence of the heights of the people in order from left to right. We want to count the number of pairs of indices i and j with $i < j$ such that for all k with $i < k < j$, $d_i > d_k$ and $d_j > d_k$. Note that if $j = i + 1$ (i.e., there are no k's between i and j), it is trivially true.

Input

The first line of input contains an integer n ($2 \leq n \leq 10^6$), which is the number of people.

Each of the next n lines contains a single integer d_i ($1 \leq d_i \leq n$). These are the heights of the people in the group, in the order in which they're standing. The sequence is guaranteed to be a permutation of the integers 1 through n.

Output

Output a single integer, which is the number of pairs of people who are taller than everyone between them.

Example

Input	Output
3 2 1 3	3
6 1 3 2 6 4 5	7

Problem I. TripTik

Source file name:	Triptik.c, Triptik.cpp, Triptik.java, Triptik.py
Input:	`Standard`
Output:	`Standard`

Have you ever been on a long road journey? AAA (the American Automobile Association) has a tool for long road trips. It's called a TripTik, and it follows the highways, showing points of interest.

You are building a TripTik app, which allows users to see what's on their route. It models a highway as a straight line, and points of interest as points along that line. All points have an integer coordinate as well as a unique integral weight. Your app provides a viewport, which can scale in and out. Also, to prevent the display from becoming too cluttered, only a small number of the points with the highest weights are shown. The initial viewport is centered at 0.0, and shows from -1.0 to 1.0 on the line.

There are three valid operations for changing your viewport:

1. **Zoom out**: double the dimensions of your viewport while keeping the center the same; this can always be done regardless of the current dimensions of the viewport.

2. **Zoom in:** halve the dimensions of your viewport while keeping the center the same; this can always be done regardless of the current dimensions of the viewport.

3. **Recenter:** change the center of your viewport to be equal to a point of interest visible in your viewport (including the boundary).

There is an important caveat: Your TripTik app will not render all points of interest in a given viewport; instead, it will only render a certain number of points in the viewport with the highest weights. The remaining points with lower weight are not visible, and therefore are not valid targets for the recenter operation.

For each point of interest, determine the minimum number of operations needed to go from the starting viewport to a viewport where that point of interest is centered and visible. Consider each point of interest independently

Input

The first line of input contains two integers n ($1 \leq n \leq 10^5$) and k ($1 \leq k \leq 4$), where n is the number of points, and k is the maximum number of points visible in the viewport.

Each of the next n lines contains a single integer x ($|x| \leq 10^8,\ x \neq 0$). These are the points of interest. The weight of each point is equal to its position in the list, with lower weights earlier in the list. All points will be distinct.

Output

Output n lines, each with a single integer, indicating the minimum number of operations necessary to get from the starting viewport to a viewport which is centered on the corresponding input point and can see that point. Output -1 if this isn't possible. The order of output lines should correspond to the order of inputs for which they're the answer

Example

Input	Output

4 2
100
4
1
3
	-1
	5
	1
	3

Problem J. Longest Common Subsequence

Source file name:	Longest.c, Longest.cpp, Longest.java, Longest.py
Input:	`Standard`
Output:	`Standard`

You are given n strings, each a permutation of the first k upper-case letters of the alphabet.

String s is a subsequence of string t if and only if it is possible to delete some (possibly zero) characters from the string t to get the string s.

Compute the length of the longest common subsequence of all n strings.

Input

The first line of input contains two integers n ($1 \leq n \leq 10^5$) and k ($1 \leq k \leq 26$), where n is the number of strings, and the strings are all permutations of the first k upper-case letters of the alphabet.

Each of the next n lines contains a single string t. It is guaranteed that every t contains each of the first k upper-case letters of the alphabet exactly once.

Output

Output a single integer, the length of the longest subsequence that appears in all n strings.

Example

Input	Output
2 3 BAC ABC	2
3 8 HGBDFCAE ADBGHFCE HCFGBDAE	3
6 8 AHFBGDCE FABGCEHD AHDGFBCE DABHGCFE ABCHFEDG DGABHFCE	4

Problem K. Condorcet

Source file name:	Condorcet.c, Condorcet.cpp, Condorcet.java, Condorcet.py
Input:	`Standard`
Output:	`Standard`

Consider an election where there are some candidates, one winner, and each voter has a complete ranked preference of the candidates. One possible method of determining the winner is to examine each pair of candidates and see which would win in a head-to-head matchup (*i.e.*, which candidate has more voters that rank them higher than their opponent) and then see if there is a single candidate that wins all their head-to-head matchups. This is called the *Condorcet Method.*

For simplicity, let `ABC` denote a vote that prefers `A` over `B`, and `B` over `C`. As an example, imagine that there are three votes: `ABC`, `BAC`, and `CAB`. Then `A` wins in a head-to-head matchup against `B` 2:1, and `A` also wins against `C` 2:1, so we could declare `A` the winner overall.

Note that a winner does not always exist; for example, imagine that the three votes were `ABC`, `BCA`, and `CAB` instead. Then `A` wins against `B` 2:1, `B` wins against `C` 2:1, and `C` wins against `A` 2:1. There is no single candidate that wins all their head-to-head matchups.

You are given the candidates and a set of votes. What is the minimum number of additional voters you would need to add (whose preferences you can individually control) in order to ensure that no overall winner exists? Assume that ties are broken by some tiebreaker you do not control and cannot predict. Therefore, you need to have every candidate lose a head-to-head matchup with some other candidate.

Input

The first line of input contains two integers, n ($3 \leq n \leq 5$) and m ($1 \leq m \leq n!$), where n is the number of candidates, and m is the number of vote tally lines. The candidates are represented by the first n upper-case letters of the alphabet.

Each of the next m lines contains a string s and an integer k ($1 \leq k \leq 10^6$). These are the vote tallies, which each consist of a string s defining the vote, and an integer count k indicating how many votes of type s are represented by that tally line. The string s describing a vote contains the first n upper-case letters, each exactly once, in some order. The votes in the vote tally lines are unique.

Output

Output a single integer, which is the minimum number of additional voters needed to ensure that no overall winner exists

Example

Input	Output
3 6 ABC 1 ACB 2 BAC 3 BCA 4 CAB 5 CBA 6	6

Problem L. Kth Subtree

Source file name:	Kthsub.c, Kthsub.cpp, Kthsub.java, Kthsub.py
Input:	`Standard`
Output:	`Standard`

You are given an unrooted labeled tree. A subtree is a connected subgraph of this tree. The size of a subtree is the number of nodes in the subtree. Two subtrees are different if there is at least one node which is in one but not the other. The largest subtree is the original tree itself.

Compute the size of the K^{th} smallest non-empty subtree.

Input

The first line of input contains two integers n $(1 \leq n \leq 5,000)$ and K $(1 \leq K \leq 10^{18})$, where n is the number of nodes in the tree, and you're looking for the size of the K^{th} smallest subtree. The nodes are numbered 1 through n.

Each of the next $n-1$ lines contains a pair of integers u and v $(1 \leq u, v \leq n,\ \ u \neq v)$, which represents an undirected edge between nodes u and v. All edges are distinct. It is guaranteed that the edges form a single tree.

Output

Output a single integer, which is the number of nodes in the K^{th} smallest non-empty subtree of the input tree. If there are fewer than K non-empty subtrees of the given tree, output -1.

Example

Input	Output

`2 1`
`1 2`
	`1`

`2 3`
`1 2`
	`2`

`5 10`
`1 2`
`2 3`
`3 4`
`4 5`
	`3`

Problem M. Bad Packing

Source file name:	Badpack.c, Badpack.cpp, Badpack.java, Badpack.py
Input:	`Standard`
Output:	`Standard`

We have a knapsack of integral capacity and some objects of assorted integral sizes. We attempt to fill the knapsack up, but unfortunately, we are really bad at it, so we end up wasting a lot of space that can't be further filled by any of the remaining objects. In fact, we are optimally bad at this! How bad can we possibly be?

Figure out the least capacity we can use where we cannot place any of the remaining objects in the knapsack. For example, suppose we have 3 objects with weights 3, 5 and 3, and our knapsack has capacity 6. If we foolishly pack the object with weight 5 first, we cannot place either of the other two objects in the knapsack. That's the worst we can do, so 5 is the answer.

Input

The first line of input contains two integers n ($1 \leq n \leq 1,000$) and c ($1 \leq c \leq 10^5$), where n is the number of objects we want to pack and c is the capacity of the knapsack.

Each of the next n lines contains a single integer w ($1 \leq w \leq c$). These are the weights of the objects.

Output

Output a single integer, which is the least capacity we can use where we cannot place any of the remaining objects in the knapsack.

Example

Input	Output

3 6
3
5
3
	5

Problem N. Exciting Tournament

Source file name:	Exciting.c, Exciting.cpp, Exciting.java, Exciting.py
Input:	`Standard`
Output:	`Standard`

A group of players compete in a no-holds-barred tournament.

Each player has a unique skill level (represented as an integer). In each game, two players play, and the player of higher skill level wins. The player of lower skill level is immediately eliminated from the tournament! The tournament continues until there is only one player left.

Due to scheduling constraints, each player has a limit on the maximum number of games they can play. Interestingly, this is the only constraint that the tournament bracket needs to meet. In other words, the bracket may not necessarily have the shape of a balanced binary tree, as long as every player plays at most their maximum number of games before getting eliminated or winning the entire tournament.

As a tournament organizer, you are free to choose any valid bracket. Given the list of participants, you wonder how exciting (or not exciting) the tournament can get. Concretely, the *excitement of a game* is defined as the bitwise XOR of the two players' skill levels. The *excitement of the tournament* is simply the sum of the *excitement* of each game.

Compute the minimum and maximum possible *excitement* values of the entire tournament.

Input

The first line of input contains a single integer n ($3 \leq n \leq 100$), which is the number of players in the tournament.

Each of the next n lines contains two integers s ($0 \leq s < 2^{30}$) and g ($2 \leq g < n$). Each line describes a single player; s is the skill level of the player, and the g is the limit on the number of games that player can play.

Output

Output two space-separated integers on a single line, which are the minimum and maximum possible *excitement* values of the entire tournament, minimum first.

Example

Input	Output

4
41 2
13 2
36 3
17 3
	94 110

6
66 5
628 4
216 5
78 4
230 5
74 3
	882 2650

Problem O. Rainbow Numbers

Source file name:	Rainbow.c, Rainbow.cpp, Rainbow.java, Rainbow.py
Input:	`Standard`
Output:	`Standard`

Define a *rainbow number* as an integer that, when represented in base 10 with no leading zeros, has no two adjacent digits the same.

Given lower and upper bounds, count the number of rainbow numbers between them (inclusive)

Input

The first line of input contains a single integer L $\left(1 \le L < 10^{10^5}\right)$, which is the lower bound.

The second line of input contains a single integer U $\left(1 \le U < 10^{10^5}\right)$, which is the upper bound.

It is guaranteed that $L \le U$. Note that the limits are not a misprint; L and U can be up to 10^5 digits long.

Output

Output a single integer, which is the number of rainbow numbers between L and U (inclusive). Because this number may be very large, output it modulo $998,244,353$.

Example

Input	Output
1 10	10
12345 65432	35882