Problem A. Dress to Impress

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

Ariana is very rich. She has several rings, she buys diamonds for her friends and she cannot resist an impulse to buy a thing once she sees it and likes it.

That is why Ariana has many clothes of different colors. Formally, there are n types of clothes or accessories and m colors for them. Each piece of clothing can be described by its type and color.

Not surprisingly, Ariana likes to be flossy. She calls a set of n pieces of clothing of different types an *attractive set* if it contains clothes of at least k colors. At the beginning of each day she picks an attractive set from the clothes she has. At the end of each day she throws this set away since she has already worn it.

You are given all the clothes Ariana has. Find out the maximal number of days she can go out without buying new clothes if she picks attractive sets optimally (it's not like she cannot afford new clothes, of course, it's just pure curiosity). Additionally, determine what to wear each day to achieve that number.

Input

The first line contains three integers n, m, c and k ($1 \le n, m \le 100$, $1 \le c \le 5000$, $1 \le k \le \min(n, m)$) denoting the number of types of clothing, the number of possible colors, the number of clothes Ariana has and the attractiveness threshold, respectively. The i-th of the next c lines contains two integers x_i and y_i ($1 \le x_i \le n$, $1 \le y_i \le m$), denoting a piece of clothing of type x_i and color y_i.

Output

On the first line output the only integer d: the answer to the problem.

On i-th of the following d lines print n integers $id_{i,1}$, $id_{i,2}$, ..., $id_{i,n}$ describing the i-th attractive set ($1 \le id_{i,j} \le c$). The value $id_{i,j}$ must be the index of a piece of type j from the i-th attractive set. **Note that the order of clothes in an attractive set matters,** because $x_{id_{i,j}} = j$ must hold. Each set must contain pieces of clothing of at least k different colors, and nothing may be used more than once.

If there are several possible lists of attractive sets of size d, print any one of them.

Examples

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

Problem B. Somewhere Over the Rainbow

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

You want to draw a rainbow. The rainbow can be represented as a sequence of **integer** heights a_0, a_1, ..., a_m and must satisfy the following constraints:

- $a_0 = a_m = 0$ (the endpoints of the rainbow are 0 meters above the horizon),
- $2a_i > a_{i-1} + a_{i+1}$ for all $0 < i < m$ (the rainbow is convex),
- $a_{x_i} \geq y_i$ for n given pairs (x_i, y_i).

You also want the rainbow to take as little space as possible, so please find the minimum possible value of $\sum_{i=0}^{m} a_i$. Since the answer may be very large, output it modulo 998 244 353.

Input

The first line of input contains two positive integers n and m ($1 \leq n \leq 2 \cdot 10^5$, $1 \leq m \leq 10^9$): the number of constraints and the length of the sequence.

Each of the next n lines contains two integers x_i ($1 \leq x_i \leq m - 1$) and y_i ($1 \leq y_i \leq 10^{18}$), which set conditions $a_{x_i} \geq y_i$.

It is guaranteed that $x_1 < x_2 < \ldots < x_n$.

Output

Print one integer: the minimum value of $\sum_{i=0}^{m} a_i$ modulo 998 244 353.

Example

standard input	standard output
3 6 1 100 4 42 5 22	310

Note

In the sample case, one optimal height sequence is $(0, 100, 82, 63, 43, 22, 0)$.

Problem C. Goldberg Machine

Input file:	*standard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	512 mebibytes

Rube has constructed a brand new obscure contraption and is now testing it. The machine has n nodes numbered from 1 to n that can accomodate a marble, and $n-1$ tracks connecting the nodes. It is possible to get from any node to any other by following the tracks, that is, the tracks form an undirected tree.

For every node, the tracks adjacent to it are ordered: if a node v has k_v tracks adjacent to it, we will write $e_{v,0}, \ldots, e_{v,k_v-1}$ for the order in question. Further, every node has an *active* adjacent track selected: we will write $t_v \in \{0, \ldots, k_v - 1\}$ to denote that the track e_{t_v} is active for the node v.

Here's how the machine operates. First, a marble is placed at the node 1. Then, one or more *steps* take place. Each step proceeds as follows:

- If the marble is currently in a node v, it moves to the other endpoint of the track e_{v,t_v} (the currently active track for the node v).
- After the ball has moved to the new node, t_v becomes equal to $(t_v + 1) \bmod k_v$ (that is, the node that contained the marble **at the start of the step** changes its active track to the next one in the cyclic ordering of its tracks).

Rube wants you to process several queries of two kinds:

- "`C` v t" ($1 \le v \le n$, $0 \le t \le k_v - 1$): set $t_v = t$;
- "`Q` x" ($1 \le x \le 10^{18}$): determine the index of the node that will contain the marble after taking exactly x steps from the node 1 starting from the current configuration. Note that this query **does not** affect the starting configuration for any further queries.

Rube is very busy tinkering with his device, so he wants to you answer fast!

Input

The first line contains a single integer n ($2 \le n \le 10^5$): the number of nodes in the machine.

The following n lines describe the tracks. The i-th of these lines contains two integers k_i and t_i ($0 \le t_i \le k_i - 1$), followed by k_i distinct integers $u_{i,0}, \ldots, u_{v,k_v-1}$ ($u_{i,j} \ne i$). This denotes:

- there are k_i tracks adjacent to the node i;
- for any $j \in \{0, \ldots, k_i - 1\}$ the track $e_{i,j}$ leads from i to $u_{i,j}$;
- the track e_{i,t_i} is currently active for the node i.

It is guaranteed that the described tracks form an undirected tree, that is:

- for any pair of distinct nodes (u, v), the node u is an endpoint of a track adjacent to v if and only if v is an endpoint of a track adjacent to u;
- there are $n - 1$ distinct undirected tracks, that is, $\sum_{i=1}^{n} k_i = 2(n - 1)$;
- it is possible to get from any node to any other by following the tracks.

The following line contains a single integer q ($1 \le q \le 10^5$): the number of queries.

The following q lines describe the queries in the format described above, one per line.

Output

Print answers for all "`Q` x" queries in the order asked, one per line.

Example

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

Note

Here are the paths of the marbles in the sample case:

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

Problem D. Grid Game

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

Consider an infinite two-dimensional grid. A white chip is placed in the cell $(0, 0)$. There are also n black chips, the i-th of which is initially in the cell (x_i, y_i). Some chips may occupy the same cells.

Two player play a game, taking turns starting with the first player. On his turn, the first player must move the white chip from its current cell (x, y) to $(x + 1, y)$ or $(x, y + 1)$. The second player must move each black chip from its current position (x, y) to $(x - 1, y)$ or $(x, y - 1)$, choosing directions for each chip independently. If at any moment (after first or second player's turn) the white chip shares its cell with any black chip, the first player immediately loses the game. If the first player manages to make 10^{100} turns without losing, he wins. Determine if the first player can win regardless of the strategy of the second player.

Input

The first line contains an integer t: the number of test cases.

Each test case starts with a line with a positive integer n denoting the number of black chips. It is followed by n lines, each containing two integers x_i and y_i ($0 \le x_i, y_i \le 100$, $x_i + y_i > 0$).

It is guaranteed that the total amount of black chips over all test cases does not exceed 10^4.

Output

For each test case, print a separate line with a single word: "`Yes`" if the first player can win, or "`No`" otherwise.

Example

standard input	standard output
3 1 3 3 2 2 0 1 2 4 2 0 2 3 1 6 5 2	No Yes Yes

Problem E. k-coloring

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

Afanasy has a simple connected undirected graph. Afanasy plays a game on this graph. Initially, there is a chip in the vertex 1. Afanasy can move the chip along the edges of the graph arbitrarily: at any moment he can choose any edge adjacent to the current vertex, including already visited edges, or the edge he used on the previous step.

Every k-th edge passed by the chip becomes colored (that is, edges will be colored on k-th step, $2k$-th step, and so on). If Afanasy tries to color an edge that is already colored, he loses. To win, he needs to color all the edges of the graph. Help Afanasy win, and find a sequence of movements of the chip such that all edges are colored exactly once, or determine that it is impossible.

Input

The first line contains three integers n, m, k: the number of vertices and edges in the graph, and the given parameter ($1 \le n \le 100\,000$, $n - 1 \le m \le 100\,000$, $1 \le k \le 10$).

The next m lines describe the edges of the graph. Each of them contains two different integers u and v: the two vertices connected by an edge ($1 \le u, v \le n$).

It is guaranteed that the graph is connected, and also contain no multiple edges and no loops.

Output

If there is no suitable path, print the only integer -1. Otherwise, on the first line, print one integer: the number of vertices on the path. On the next line, print the numbers of these vertices in the order of visiting.

The path cannot contain more than 1 000 001 vertices. The path should start at the vertex 1 and is allowed to finish at any vertex. If there are multiple suitable paths, you are allowed to output any one of them.

Examples

standard input	standard output
3 3 1 1 2 2 3 3 1	4 1 2 3 1
3 3 2 1 2 2 3 3 1	7 1 2 3 1 2 3 1

Problem F. Just Shuffle the Input

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

A *permutation* p of size n is a sequence of n pairwise distinct numbers from 1 to n. We denote the i-th of them by $p(i)$. By $p^k(i)$ we denote $\underbrace{p(p(p(\ldots(p(i))\ldots)))}_{k \text{ times}}$. A permutation is called *cyclic* if the minimal positive k for which $p^k(1) = 1$ equals n.

You are given a string s of size n, a string t of size m and a cyclic permutation p of size m. You want t to be a substring of s. To do this, you may apply the *shuffle* operation zero or more times. The shuffle operation consists of replacing t with t', such that the i-th letter of t equals the $p(i)$-th letter of t' for each i from 1 to m.

Please find out if it is possible obtain a substring of s. If it is possible, find the minimum number of shuffles required.

Recall that a string a is a substring of s if there exists some l such that $1 \le l \le |s| - |a| + 1$ and $s_{l+i-1} = a_i$ for every i from 1 to $|a|$.

Input

The first line of input contains two integers n and m ($1 \le m \le n \le 200\,000$). The second line contains m integers $p(1), \ldots, p(m)$: the permutation you can apply. The next two lines contain string s of length n and string t of length m, respectively. Both strings consist of lowercase English letters.

It is guaranteed that the permutation in the input is cyclic.

Output

If it is impossible to obtain a substring of s from t, output -1. Otherwise print the minimum number of shuffles needed to obtain a substring of s.

Examples

standard input	standard output
3 2 2 1 aba ba	0
7 4 3 4 2 1 dcabadc abcd	1

Problem G. Restoring a Permutation

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

You are given a positive integer n and two arrays a and b containing n integers each.

You need to find permutation p of length n such that for each $i \in \{1, 2, \ldots, n\}$ the following two conditions are satisfied:

- the length of longest increasing subsequence of p ending at position i is equal to a_i,
- the length of longest decreasing subsequence of p starting at position i is equal to b_i.

Input

The first line of input contains a positive integer n ($1 \le n \le 2 \cdot 10^5$), the length of the permutation.

The second line contains n integers $a_1, a_2 \ldots, a_n$, where a_i ($1 \le a_i \le n$): the length of longest increasing subsequence ending at position i.

The third line contains n integers $b_1, b_2 \ldots, b_n$, where b_i ($1 \le b_i \le n$): the length of longest decreasing subsequence starting at position i.

Output

Print a line containing n space-separated integers $p_1, p_2, \ldots, p_n$: the desired permutation.

It is guaranteed that the answer exists. If there are multiple solutions, you may print any one of them.

Examples

standard input	standard output
5 1 1 1 2 3 3 2 1 1 1	3 2 1 4 5
6 1 1 2 2 3 3 2 1 2 1 2 1	2 1 4 3 6 5

Problem H. Shadow Companion

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

You are given a number n in its binary representation with infinite number of leading zeroes. You can walk over its bits and do some operations with them.

More precisely, you can make the following moves:

- "`s`": **s**ummon a **s**hadow. Your shadow will appear one bit to the left from your current position.
- "`l`": walk one bit to the **l**eft. Your shadow, if it exists, does the same.
- "`r`": walk one bit to the **r**ight. Your shadow, if it exists, does the same.
- "`L`": if the bit you are standing at is `1` then walk one bit to the left, otherwise do nothing. Your shadow, if it exists and is standing at `1`, also walks one bit to the left. **Please note that you and your shadow behave independently of each other.** You move if you are standing at `1`, it moves if it is standing at `1`.
- "`R`": the same as the previous option, but you and your shadow move right instead of left.
- "`x`": e**x**change positions with your shadow. If it does not exist, nothing happens.
- "`f`": **f**lip some bits (flipping means replacing `0` by `1` and vice versa). You flip two bits: the bit you are standing at and the bit to the left from you. Your shadow, if it exists, is not as strong as you, and flips only the bit it is standing at. **Please note that during this move some bit may be flipped twice, thus remaining unchanged.**

Initially you are at the rightmost (the least significant) bit, and $0 \le n < 2^{10}$ holds. You are asked to write a program (that is, a sequence of moves) which satisfies the following:

- Neither you nor your shadow ever try to move to the right from the least significant bit (in other words, don't try to leave the number).
- Your program consists of no more than 500 000 commands.
- The number at the end is n^2.

Some technical details:

- If after some move you and your shadow have the same position **then your shadow disappears**.
- If you summon a shadow when you already have one **then your previous shadow disappears**.
- Your position at the end may be arbitrary.
- Your shadow is allowed to exist in the end, and its position may be arbitrary.
- During the execution, the number represented by the bits may be arbitrarily large. The only constraint on it is that it must equal n^2 in the end.
- Again, if you and your shadow flip the same bit simultaneously then it does not change.

Input

You will have no input.

Output

You should print a single string consisting of no more than 500 000 letters from the set "`slrLRxf`" representing your program.

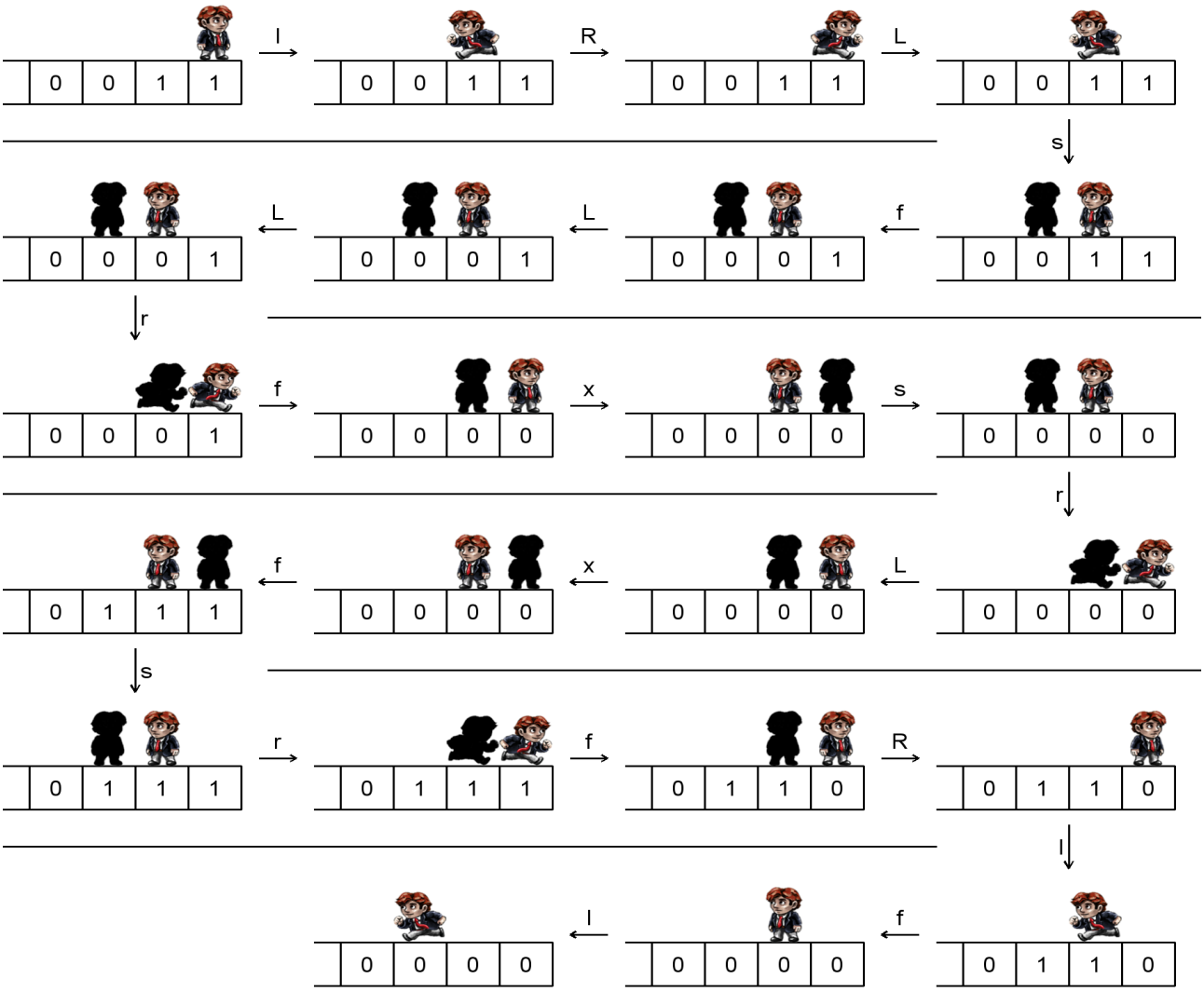
Example

standard input	standard output
	`lRLsfLLrfxsrLxfsrfRlfl`

Note

The sample output is incorrect. In fact, it is a program which, starting from the state where the two rightmost bits are x and y and the other bits are zeroes, comes to the state where all the bits are zeroes and the third bit (1-indexing) is the Sheffer stroke of x and y (which is $\neg(x \wedge y)$).

Below is an example of how it works for $x = y = 1$:



Problem I. The Older We Are, The Worse It Hurts

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

You are given a tree with n vertices. Each vertex i has a weight a_i.

You traverse the whole tree starting in an arbitrary vertex and moving along the edges so that each edge is traversed exactly once in each direction (in other words, you perform a depth-first search traversal choosing the initial vertex and the order of outgoing edges arbitrarily). Write down the list of all vertices, $(v_1, v_2, \ldots, v_n)$, sorted by the time when you first arrive at them. You get a penalty of $\sum_{i=1}^{n} i \cdot a_{v_i}$.

Your goal is to minimize the penalty. Note that $(v_1, v_2, \ldots, v_n)$ is a permutation of $(1, 2, \ldots, n)$, and v_1 is the vertex you start from.

Input

The first line contains the only integer n ($1 \le n \le 200\,000$) denoting the number of vertices. The next $n-1$ lines contain edges descriptions: i-th of them contains two integers u_i and v_i ($1 \le u_i, v_i \le n$) denoting the edge between u_i and v_i. The third line contains n space-separated integers a_i ($1 \le a_i \le 200\,000$).

It is guaranteed that the given edges represent a tree.

Output

Print a single line with a single integer on it: the minimum possible penalty.

Examples

standard input	standard output
3 1 2 1 3 1 2 3	11
5 1 2 1 3 3 4 3 5 5 4 3 2 1	35

Problem J. Three Vectors

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

You are given three distinct binary vectors of length n. Find any 2-CNF formula which satisfies the following conditions:

- The formula is true on these vectors;
- The number of vectors on which the formula is true is minimal possible;
- The formula is not too long.

Recall that a *2-CNF formula* is a propositional formula of n boolean variables $v_1, \ldots, v_n$ which looks like

$$C_1 \wedge C_2 \wedge \ldots \wedge C_m,$$

where each clause C_i is represented as a disjunction $\pm x_{i1} \vee \pm x_{i2}$ of two literals (by literal, we mean any variable or its negation). Here, $x_{ij} \in \{v_1, \ldots, v_n\}$ is one of the variables, and $-x_{ij}$ is its negation: true becomes false, and false becomes true. We say that the formula f *is true on a binary vector* v if $f(v_1, v_2, \ldots, v_n) = 1$.

If there are several valid formulas, you are allowed to output any one of them.

Input

The first line of input contains a single integer n which denotes the length of the three vectors ($2 \le n \le 10^5$). The i-th of the following three lines contains a binary string of length n denoting the i-th binary vector.

No two vectors coincide.

Output

On the first line, print a single integer m ($0 \le m \le 2 \cdot 10^5$). Then output m lines, i-th of them containing two integers a_i and b_i ($1 \le |a_i|, |b_i| \le n$), denoting that the i-th clause is a conjunction of two literals: the first is v_{a_i} if $a_i > 0$ and $-v_{|a_i|}$ otherwise, and the second is, similarly, v_{b_i} if $b_i > 0$ and $-v_{|b_i|}$ otherwise. If your formula is empty (that is, $m = 0$), it is considered to be true for every possible input vector of size n.

Please note that, if you use too many clauses, your answer will be considered incorrect.

Examples

standard input	standard output
5 00101 10011 11011	6 -1 -3 3 1 -1 4 -4 1 5 5 -2 1
3 100 010 001	3 -2 -1 -3 -1 -3 -2