Problem A
Equivalence

Input File: A.in
Output File: standard output
Time Limit: 0.2 seconds (C/C++)
Memory Limit: 256 megabytes

A **propositional formula** is generated by the following grammar:

<*formula*> ::= <*variable*> | ~<*formula*> | (<*formula*>) | <*formula*> <*operator*> <*formula*>
<*operator*> ::= `^` | `V`
<*variable*> ::= `a-zA-Z` (except character `V`)

where '`^`' encodes boolean **`AND`**, '`V`' encodes boolean **`OR`**.

An ***interpretation*** is a truth-assignment for all variables ocurring in a formula. The truth-value of a formula with respect to an interpretation can be determined by applying boolean operations on the values of variables, in the standard way.

Two propositional formulae are ***equivalent*** if they produce the same truth-value for all possible interpretations.

The input file will contain two formulae generated by the above grammar. The formulae are separated by the newline character. Variables are encoded by alphabetic characters, except the character '**`V`**' which is reserved for encoding **`OR`** . Each formula will have at most **`51`** variables. Whitespaces can occur freely anywhere in the input.

The output must be **`1`** if the two formulae are equivalent, and **`0`** otherwise.

In your implementation, you do not need to take into account operator precedence (priority). For instance, a formula such as:

`x ^ y V z`

will be presented as either

`(x ^ y) V z` or
`x ^ (y V z)`.

Sample input	Sample output
`x ^ y` `y ^ x`	`1`
`A ^ (~y V z)` `(A^~y) V (A ^ z)`	`1`
`a V (b ^ ~b) V (c ^ ~c)` `a`	`1`
`~x ^ ~y` `~(x V y)`	`1`
`a ^ b ^ c ^ d` `a V b V c V d`	`0`

Problem B
Tree of Almost Clean Money

Input File: B.in
Output File: standard output
Time Limit: 4 seconds (C/C++)
Memory Limit: 256 megabytes

The tree of Almost Clean Money (or ACM Tree, for short) consists of N ($1 \le N \le 500000$) vertices in which, well, (almost clean) money is growing (contrary to the old saying that money doesn't grow on trees). The vertices are numbered from 0 to $N-1$, with vertex 0 being the root of the tree. Every vertex i except vertex 0 has a parent $p(i)$ in the tree, such that $p(i) < i$. Initially, every vertex contains $v(i)$ ($0 \le v(i) < 1000000007$) monetary units. Due to its special properties, the tree has attracted the attention of a large money laundering organization, who wants to use the tree for its money "cleansing" business. This organization wants to execute Q ($1 \le Q \le 50000$) operations on the tree. Each operation consists of two steps:

1) In step 1, K ($1 \le K \le 1000$) vertices from the tree are chosen: $x(1), \ldots, x(K)$ ($0 \le x(i) \le N-1$) – the same vertex may be selected multiple times here. In each of these vertices, an amount of monetary units is added (thus increasing the amount of monetary units in them). More exactly, $y(i)$ ($0 \le y(i) < 1000000007$) monetary units are added to the selected vertex $x(i)$ ($1 \le i \le K$).
2) In step 2, two vertices u and v ($0 \le u, v \le N-1$) are chosen and the organization wants to know the total amount of money found in the vertices located on the unique path in the tree between the vertices u and v (with u and v inclusive).

The organization hired you to find the answer for step 2 of each of the Q operations and promised you a hefty amount of money if you succeed.

Input
The first line of input contains the number of tree vertices N. The next $N-1$ lines contain two space-separated integers, $p(i)$ and i, each describing an edge of the tree. The next line contains N space-separated values: the initial amount of monetary units in each vertex, $v(0), \ldots, v(N-1)$. The next line contains the number of operations Q. Each of the next Q lines describes an operation. Each operation is described by 9 space-separated integers, in this order: K, $x(1)$, $y(1)$, A, B, C, D, u, v ($0 \le A,B,C,D < 1000000007$). The values $x(2 \le i \le K)$ and $y(2 \le i \le K)$ are generated as follows:

$$x(i) = (A * x(i-1) + B) \bmod N$$
$$y(i) = (C * y(i-1) + D) \bmod 1000000007$$

Output
For each of the Q operations print a line containing the answer to step 2 of the operation. When computing the answer for an operation, the effects of steps 1 from previous operations need to be considered, too (i.e. after adding $y(i)$ monetary units to a vertex $x(i)$, these units remain added to the vertex when executing subsequent operations, too).

Sample input	Sample output	Explanation
4 0 1 0 3 1 2 1 2 3 4 3 1000 1 1 1 0 1 0 0 2 2 0 5 1 1 2 2 2 3 1 3 7 999 999 999 999 1 3	1006 1027 1031	In the first operation the value 1 is added 1000 times to vertex 1 (note A=C=1, B=D=0). The path between 0 and 2 contains the vertices 0, 1 and 2. The total amount of monetary units in them is 1006. In operation 2: x(1)=0, y(1)=5, x(2)=1, y(2)=12. The path between 2 and 3 contains all the vertices of the tree. In operation 3: K=1, so A, B, C, D are irrelevant.

Problem C
Primes

Input File: C.in
Output File: standard output
Time Limit: 1 second (C/C++)
Memory Limit: 256 megabytes

Define a *mass split* operation for the multiset of positive integers **K**: for each integer $\mathbf{k_i}$ in the multiset we will replace it with the pair $\mathbf{d_i}$ and $\mathbf{k_i/d_i}$, where $\mathbf{d_i}$ is the random integer divisor of $\mathbf{k_i}$, which is greater than 1, and less than $\mathbf{k_i}$. If $\mathbf{k_i}$ is prime, it remains untouched. All divisors can be chosen equiprobably.

For example, let's take the multiset {2, 10, 12, 12}. Then {2, 2, 3, 3, 4, 4, 5}, {2, 2, 2, 3, 4, 5, 6} and {2, 2, 2, 2, 5, 6, 6} will be the possible outcomes of the first mass split (first and third with probability 0.25, second with probability 0.5), and {2, 2, 2, 2, 2, 2, 3, 3, 5} will be the only possible outcome of the second mass split.

If we start with a multiset containing one integer **N**, find the expected number of mass splits needed to obtain a multiset with prime numbers only, where the expected number is the probability-weighted average of all possible values.

Input
First line of the input contains integer **T** ($1 \le \mathbf{T} \le 10^4$) – number of test cases. Each test case consists of one integer **N** – the starting multiset ($2 \le \mathbf{N} \le 10^{10}$).

Output
For each test case, print one number – the expected number of mass splits, with absolute or relative
error not worse than $\mathbf{10^{-6}}$.

Sample input	Sample output
3 3 12 48	0 2.0 3.3333333

Problem D
LCM

Input File: D.in
Output File: standard output
Time Limit: 0.1 seconds (C/C++)
Memory Limit: 256 megabytes

You are given two natural numbers **A** and **B**. Determine the natural number **N** such that the least common multiple of the numbers **A** + **N** and **B** + **N** is minimal.

Input
The only line of the input contains two natural numbers: **A** and **B**. None of them is exceeding 10^9.

Output
Output the natural number **N** such that LCM(**A** + **N**, **B** + **N**) is minimal. If there are several values of **N** which yield the minimum, output the smallest one.

Sample input	Sample output
4 10	2

Problem E
Taxies

Input File: E.in
Output File: standard output
Time Limit: 1 second (C/C++)
Memory Limit: 256 megabytes

Some employees of the company had to work overtime and finished their work late in the night. **K** ($2 \le K \le 15$) of the employees, who commonly use public transport, ask the company manager to have the taxi fee reimbursed for them. There are a lot of available taxis, and the manager can order an individual taxi for each employee. However, he finds it too expensive, because a taxi can take some passengers (ranging from 1 to 4), and a ride in the same taxi for people who live close to each other is much cheaper. On the other hand, the manager decides that it would be too impolite to make some employees wait outdoors as the taxi driver gives their colleagues a lift and returns back to pick them up. Therefore, the manager wants to choose the cheapest method to transport all the employees to their homes, satisfying such constraints:

- All **K** employees are divided into groups, where each group contains 1, 2, 3 or 4 people; the manager decides how people are grouped.
- Employees of the same group share the same taxi.
- All the people from the group go to one of the group member's home, where this person gets out of the taxi; the rest of the people from the group (while there are any) go to the next person's home, and so on. The order (which person will be the 1st, which the 2nd, and so on) is also defined by the manager.
- The manager himself is not one of the **K** employees and does not belong to any group. He drives his own car, and does not want to share it with any employee.

The company and employees' homes are located in the vertices of a weighted graph. Most of edges of the graph are undirected (two-way roads), but some may be directed (one-way). The weight of each edge of the graph is the fee of traveling in a taxi along the edge. The graph is strongly connected, i. e. there exists a path from each vertex of the graph to any other vertex.
The taxi costs include the distance fee and the boarding fee. The boarding fee is charged per car, without reference to the distance covered and the number of passengers.

Input
The 1st line contains the number of vertices **N** ($5 \le N \le 20000$) and the number of edges **M** ($N \le M \le 50000$); afterwards, there are **M** lines, each of which contains four integers: the 1st number is 1 or 2, denoting a one-way or a two-way road; then, two numbers **u**, **v** ($u \ne v$, $1 \le v \le N$, $1 \le u \le N$) denoting which vertices are connected (if the road is one-way, it's directed from **u** to **v**); the 4th number (ranging from 5 to 5000) denotes the weight (cost of traveling in a taxi) of the edge.
The next line of input data contains the boarding fee (integer number ranging from 500 to 50000).
The next line contains the index (ranging from 1 to **N**) of the vertex where the company is located.
The next line contains the number of employees **K** ($2 \le K \le 15$). The next line is the last and contains **K** numbers (each ranging from 1 to **N**) — indices of the vertices where the employees live. Different employees may live in the same vertex, but nobody lives in the vertex where the company is located.

Output
Print one number — the minimal total cost of taking all employees to their homes.

Sample input	Sample output
6 7 2 1 2 200 2 1 3 1000 2 1 4 1200 2 2 3 900 2 6 2 1300 2 6 4 200 2 4 5 100 1000 1 4 2 3 5 6	4500
6 7 2 1 2 200 2 1 3 1000 2 1 4 1200 2 2 3 900 2 6 2 1300 2 6 4 200 2 4 5 100 500 1 4 2 3 5 6	3700

Note In the 1st sample, the minimal total cost 4500 can be reached when one taxi car takes all the 4 employees and goes in the order: the 2nd employee's home, then the 1st's, the 4th's and the 3rd's. In the 2nd sample, the minimal cost 3700 can be reached, when two taxi cars are used, and one of the cars goes first to the 1st employee's home, then to the 2nd's, another car — first to the 3rd's, then to the 4th's.

Problem F
Irrational Roots

Input File: F.in
Output File: standard output
Time Limit: 0.1 seconds (C/C++)
Memory Limit: 256 megabytes

Let **n** be a natural number, **n** ≤ 8. Consider the following equation:

$$x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \ldots + c_1x + c_0 = 0$$

where $c_{n-1}, c_{n-2}, \ldots, c_1, c_0$ are integers and $c_0 \neq 0$.

It is known that all the **n** roots of the equation are real numbers. We consider that each root **r** of the equation satisfies the condition: -10 ≤ **r** ≤ 10. Also, there might be roots that appear more than once.

Find the number of irrational roots of the equation (an irrational root is a root that is an irrational number).

Input
The input file contains a single test. The first line of the input file contains the value of **n**. The second line contains the values of $c_{n-1}, c_{n-2}, \ldots, c_1, c_0$: each two consecutive values are separated by a single space.

Output
The result will be written to standard output.

Sample input	Sample output
6 12 -12 -454 -373 3754 1680	2

Problem G
Race

Input File: G.in
Output File: standard output
Time Limit: 2 seconds (C/C++)
Memory Limit: 256 megabytes

Bob is member of the organizing committee of the annual motorcycling race. His job is to label the contest map. All the roads of the map are bidirectional and connect two places. Each place on the map must be labeled ***regular*** or ***service***, such that no place has more than ***max*** neighbors of its own label. Let's define ***degree*** the maximum number of roads reaching a place. Then ***max* = *degree* / 2**. Your job is to help Bob.

The input file starts with the number ***n*** **(**0<***n***<1001**)** of places on the map, on a separate line. Each place is identified by a natural number from **0** to ***n* - 1**. Then follows ***n*** lines containing the description of places, each on a separate line. Line ***i***, ***i*** = **0**,.., ***n*-1**, describing place ***i***, has the following form:

***number_of_neighbors: neighbor*$_1$ *neighbor*$_2$ *... neighbor*$_m$**

The ouput file must contain the labeled map. The format is the same except that each line describing a place starts with the label of the place: `0` for regular places and `1` for service places.

***label number_of_neighbors: neighbor*$_1$ *neighbor*$_2$ *... neighbor*$_m$**

The sample describes a map with 3 places that are all connected. The first line of the input contains the number of places. The following lines contain the description of places. For example the line `2: 1 2` stands for the place identified by `0`, that has `2` neighbors identified as `1` and `2`. In the output file the line `1 2: 1 2` stands for the place `0` that has label `1`.

Sample input	Sample output
3 2: 1 2 2: 0 2 2: 0 1	3 1 2: 1 2 0 2: 0 2 0 2: 0 1

Problem H

Railway Tickets

Input File: H.in
Output File: standard output
Time Limit: 0.5 seconds (C/C++)
Memory Limit: 256 megabytes

The common rule for transporting passengers in long-distance trains at many railways is that each ticket should specify a reserved place. It's rather convenient for passengers to know that they will have a guarantee for a vacant place beforehand. However, such a rule can pose the problem of false lack of places.

Consider such an unlikely but still possible case. A train with two seats goes from A to C, with only one intermediate stop at station B. Suppose seat 1 is bought for the A-B segment, and seat 2 is bought for the B-C segment. Therefore, there is no available seat between A and C. However, a traveler can buy seat 2 for the A-B segment and seat 1 for the B-C segment, and move from one place to another during the trip.

Your task is to write a program, which, knowing which tickets are already sold, finds the number of source-destination pairs of stations which can still be reached by buying two or more tickets and switching places. You may assume that any seat which is not sold between a specific source-destination may be bought.

Input

The 1st input line contains the number **K** ($3 \le K \le 1000$) of places in the train; the 2nd line contains the number **N** ($3 \le N \le 10000$) of stations in the train's route; the 3rd line contains the number of the tickets already sold **T** ($0 \le T \le \min(10^5, K(N-1))$). Each of the next **T** lines contains three integer numbers **pl**, **st**, **fn**, describing tickets: **pl** stands for the number of the reserved place (assuming that places are sequentially numbered in range 1 to **K** over all the train, without dividing by carriages), and **st, fn** are the departure, destination stations, respectively (stations are numbered along the train's route consequently from 1 to **N**). In each ticket, **st** < **fn**, i. e. arrival station's index is strictly greater than departure station's one. Different tickets for the same place are possible only if their travel ranges do not overlap (each next ticket can start either at the station where the previous ticket for the place ends, or somewhere later on the route).

Output

Print one integer — the requested number of stations pairs.

Sample input	Sample output
3 10 6 2 9 10 3 5 9 1 2 10 2 1 4 2 7 8 3 1 2	10

Note These 10 pairs are: (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (2, 6), (2, 7), (3, 6), (3, 7), and (8, 10). For other pairs of stations, it's either possible to buy a direct ticket with specific reserved place or otherwise the train has no vacant places, even for a passenger, who is ready to buy separate tickets and change his or her place during the trip.

Problem I
Olympic Parade

Input File: I.in
Output File: standard output
Time Limit: 0.5 seconds (C/C++)
Memory Limit: 4 megabytes

SEERC organizing committee decided to make this year's opening ceremony in unusual way – organize a parade of contestants on the city streets. This year ***N*** people (contestants, coaches and guests), that represent universities from participating countries, will go on the streets, trying to impress spectators by original costumes and loud songs. Each university is represented by a group of people and has a unique identifier (ID) that is carried by each person in the group. To make the parade well organised and entertaining, each group should be lined up in several rows, each consisting of ***K*** people.

Only one university was not able to line up according to the mentioned rule, and you need to find it's ID.

Input
The first line at input contains integer **N** and **K**, separated by a single space ($1 \le N \le 1\,000\,000$, $2 \le K \le 1$). Following **N** lines contain IDs $C_1, C_2, \ldots, C_n$ ($0 \le C_i \le 1\,000\,000\,000$, $1 \le i \le N$) of **N** people.

Output
The single line at output should contain one integer – the answer for the problem.

Sample input	Sample output
10 3 1 1 2 3 1 3 3 2 2 2	2

Problem J
Word by mouth

Input File: J.in
Output File: standard output
Time Limit: 2.5 seconds (C/C++)
Memory Limit: 256 megabytes

A group of ***N*** friends are playing a word-by-mouth game, where one friend (the leader) starts saying a word (either 'hat' or 'cat') to each of the other friends. ***m*** of the ***N*** friends pronounce the words incorrectly such that instead of '`hat`' a listening friend may understand '`cat`'. For simplicity we assume that such a friend will always say '`cat`' to his other friends, no matter what another friend said to him. The leader may also have problems pronouncing, in which case he will say different things to each friend.

In order to try to reach a consensus, so that each friend recovers the same message, they try the following Word By Mouth algorithm WBM(m):

Algorithm WBM(m), m > 0
1) The leader sends his word to every other friend.
2) For each friend *i*, let v_i be value that friend *i* receives from the leader. Then, friend *i* acts as the leader in Algorithm WBM(m-1) to send the value v_i to each of the other friends. A message originated from friend *i* cannot reach *i* again.
3) For each *i* and $j \neq i$, let v_j be the value that friend *i* received from friend *j* in step (2) using Algorithm WBM(m-1). Then friend *i* uses as his value the majority of the values (v_1, v_2, …, v_{N-1}). In case of equality of values, the friends decide for '`cat`'.

Algorithm WBM(0)
1) The leader sends his word to very other friend
2) Each friend uses the word received from the leader

The game starts by the leader running the algorithm WBM(m).

The following examples show how the friends play in different scenarios. Except from the messages sent by the leader, each time a friend sends a message, he adds his ID to the message, so that the receiver knows which path the message came from.

Example 1:
As shown in Figure 1, there are *N*=`4` friends with the leader transmitting the word cat to friend `2` but the word hat to friends `3` and `4`. Friend `2` receives the following: `cat` (from `1`), `hat` (from `3`), `hat` (from `4`). He decides for `hat`. Friend `3` receives the words: `hat` (from `1`), `cat` (from `2`), `hat` (from `4`). He decides for `hat`. Friend `4` receives the words: `hat` (from `1`), `cat` (from `2`), `hat` (from `3`). He decides for hat. All the *N-1* friends (excluding the leader) decide for the same word, hat, even though the leader sent different words to each friend.

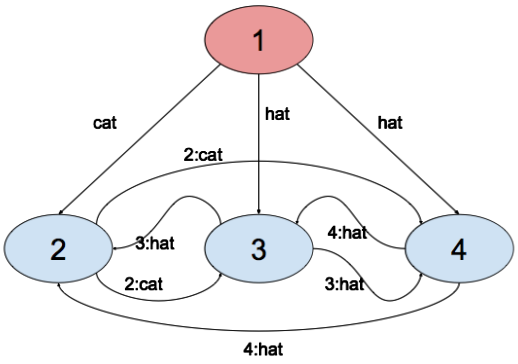

Fig. 2. N=4 friends with leader i=1 sending an incorrect word.

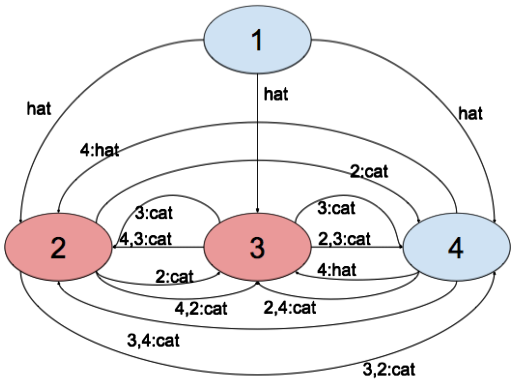

Fig. 2. N=4 friends with friends i=2 and i=3 sending an incorrect word.

Example 2:
In Figure 2, we have again *N*=4 friends but *m*=2 of them send incorrect words. We see that in this case there are more messages being exchanged. Friend `2` sends `cat` to friend `3` and then friend `3` forwards this to friend `4` (shown by '`2,3:cat`' in the figure). Friend `2` also sends the message `cat` directly to friend `4` (shown by '`2:cat`' in the figure). Therefore, friend `4` receives the messages `cat` (directly, '`2:cat`') and `cat` (indirectly, '`2,3:cat`') from friend `2` and decides that friend `2` said `cat`. Similarly, friend `4` receives from friend `3` the words `cat` (directly) and `cat` (indirectly, '`3,2:cat`') and decides that friend `3` said `cat`. Friend `4` received hat from the leader. In the end, friend `4` decides for the word `cat`, which is the majority (friends `2` and `3` said `cat`, friend `1` said `hat`).

Your task is to determine the words decided by the friends that do not confuse the words, excluding the leader. If the leader pronounces correctly, then there are ***N-m-1*** such friends (friend 4, in example 2), else there are ***N-m*** such friends (friends 2, 3 and 4, in example 1).

Input
The first line contains the numbers ***N*** *(2<**N**<101) and* ***m*** *(0<**m**<8)*.
The second line contains ***m*** positive integer numbers, representing the IDs of the friends that will modify the messages (the leader always has ID=1).
The following ***N*-1** lines contain the word ('`cat`' or '`hat`') that the leader (i=1) sends to each of the other ***N*-1** friends.

Output
On the output you should put ***N-m-1*** (if the leader pronounces correctly) or ***N-m*** (if the leader can pronounce incorrectly) words ('`cat`' or '`hat`'), one per line, representing the words understood by the friends that pronounce correctly (this excludes the leader).

Sample input	Sample output
4 1 **1** **cat** **hat** **hat**	**hat** **hat** **hat**