A: Weak pseudorandom generator

Memory limit: **128 MB**

Byteland's Homeland Security Agency is struggling fighting terrorists. There's a new terrorist organization, and they are encrypting all their communications. Secret agent Johnny infiltrated that organization, but he managed to send only a single message before Operations HQ lost all communications with him. Here's the intel: Each organization member has their own encryption key which is generated using a linear pseudorandom number generator. For given parameters p, x_0, a, b it generates the sequence $f_0 = x_0$, $f_{i+1} = (a \cdot f_i + b) \mod p$. The organization leader uses key number f_i, and i^{th} of the members uses key f_i. Once you become a member there's no way to leave the organization. In his message Johnny also sent the parameters of the generator and his own key x. Homeland Security Agency is now trying to decipher captured communications. You're given a bit different task: try to find out how many members the organization could possibly have, that is, find any k such that $f_k = x$. The organization is screening their candidates very carefully (this is even more true now as they probably learned Johnny's real identity) so surely they haven't inducted anyone after agent Johnny.

Input

The first and only line of input contains five integers separated by single spaces: p, x_0, a, b, x ($2 \leq p < 10^9$, p is prime, $0 \leq x_0, a, b, x < p$). The first four numbers are parameters of the generator, and the last one is agent Johnny's key.

Output

In the first and only line of output you should print number k such that $f_k = x$ and $0 \leq k \leq 10^{18}$, or the word `NIE` if such k doesn't exist. If there's more than one such k, you can print any of them.

Example

Input	Output
`5 3 2 1 0`	`2`

The generated sequence is $3, 2, 0, 1, 3, 2, 0, 1, 3, 2, 0, 1, \ldots$ In particular $f_2 = 0$.

Input	Output
`7 1 2 0 5`	`NIE`

The generated sequence is $1, 2, 4, 1, 2, 4, 1, 2, 4, \ldots$ There's no key with value 5, perhaps the organization already knew that Johnny is a double agent?

B: Home alone: Johnny lost in New York

Memory limit: **128 MB**

Johnny has just arrived in New York for a conference. He slept in the plane and now has a whole day free. He wants to go sightseeing, but in the evening he has to be on time for the conference. Johnny is well prepared – he knows that the streets of New York form a regular grid, and points of interest are at every crossing! Johnny wonders if it is possible to make a trip from the hotel at crossing s to the conference place at crossing t while visiting every crossing exactly once – he doesn't want to waste any time. Of course he visits crossing s just after leaving the hotel, even before the trip starts.

The streets of New York form a regular grid: on the map there are n vertical streets and m horizontal streets. The street names are just numbers: vertical streets are numbered from the left side, while horizontal streets are numbered from the up side, both types are numbered with consecutive integers starting with 1. Each horizontal street crosses with every vertical street, and the crossing of x^{th} vertical and y^{th} horizontal street is denoted as (x, y).

Input

The input consists of three lines, each of which contains two integers separated by single spaces. In the first line there are n and m ($4 \le n, m \le 1000$), denoting respectively the number of vertical and horizontal streets on the map. In the second line there are s_x and s_y ($1 \le s_x \le n$ and $1 \le s_y \le m$). In the third line there are t_x and t_y ($1 \le t_x \le n$ and $1 \le t_y \le m$). The hotel is at the crossing (s_x, s_y), while the conference place is at the crossing (t_x, t_y). Crossings s and t are different.

Output

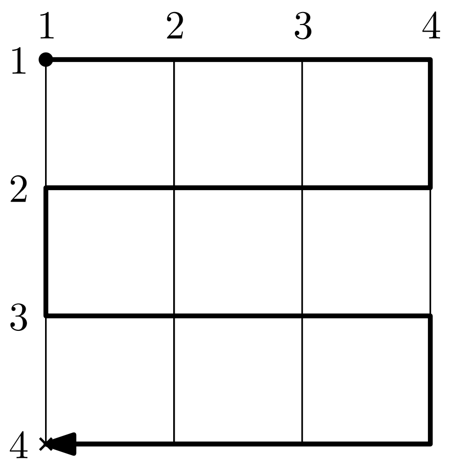
The output should consist of a single line and contain the description of a trip that Johnny could take, or the word `NIE` if it's not possible to make a trip satisfying above conditions. The description should consist of a string of $n \cdot m - 1$ letters `D`, `G`, `L`, or `P`, which mean that the next step of Johnny's trip should be to go to the nearest crossing in the direction down, up, left, or right respectively (here up, down, left, and right are the directions on the map that Johnny has).

Example

Input	Output

`4 4`
`1 1`
`1 4`
	`PPPDLLLDPPPDLLL`

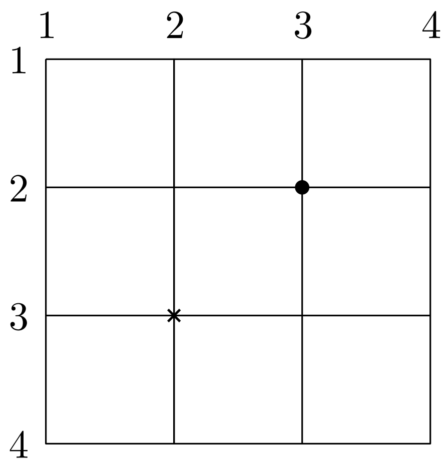
The diagram shows an example trip that Johnny could take. The hotel is marked with a dot, while the conference place is marked with a cross.



Input	Output

`4 4`
`3 2`
`2 3`
	`NIE`

The hotel is marked with a dot, while the conference place is marked with a cross. It's not possible to make a trip that would go through every crossing exactly once.



C: Generating Polygons

Memory limit: **128 MB**

Johnny has learned many new things about triangles and quadrangles in geometry class, including how to compute their perimeter and area. He isn't so good at computing these things but he really started to like drawing different polygons. Due to both aesthetic and practical reasons he does so in the following way:

- he draws an edge along one of the grid lines (horizontal or vertical) in such a way that the endpoints lie on lines perpendicular to it; this means that edges have integer lengths and are parallel to grid lines (and to axes),
- he never lifts his pencil off, so that his polygon has no "holes", and terminates at the starting point,
- each point lies on at most two edges; moreover, when a point lies on exactly two edges it is an endpoint of both of them (so it is a vertex of the polygon).

Johnny quickly noticed that computing perimeter and area of this type of polygons is very easy, so now he is wondering how to draw such polygons that would have a given perimeter p and area a.

Can you help him by providing an instruction how to draw it?

Input

The first and only line of input contains two integers p and a, separated by single space ($4 \le p \le 10^6$, $1 \le a \le 10^{12}$). They denote respectively the perimeter and area of a polygon Johnny wants to draw.

Output

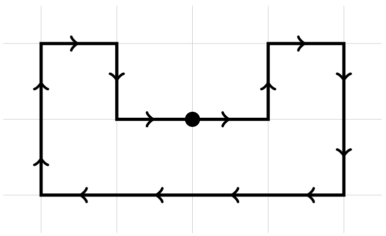
The output should consist of exactly one line. If a polygon with aforementioned properties of perimeter p and area a exists the line should contain a description of such polygon. The description should consist of p letters `L`, `B`, and `P`. They denote respectively: rotate 90 degrees left (counterclockwise), no rotation, and rotate 90 degrees right (clockwise). After performing the rotation (if any) specified by the character, a segment of length 1 will be drawn in the current direction. The starting direction is facing up.

In the opposite case, when such polygon does not exist, the only output line should contain the word `NIE`.

Example

Input	Output
`14 6`	`PLPPBPBBBPBPPL`

The instruction describes the following figure which has perimeter 14 and area 6. The starting point is marked with a dot.



Input	Output
`20 26`	`NIE`

It is easy to see that the polygon with perimeter 20 and maximum area is a 5×5 square.

D: Grapes

Memory limit: **128 MB**

Hansel received a bunch of grapes from his mother. He is supposed to share it with Gretel, who likes grapes very much. She is rather predictable, so Hansel knows that as soon as he breaks one twig of the bunch, she will insist that, because she is a girl, she should be allowed to choose one of the two obtained parts. Of course she will take the one with the larger number of berries. Hence Hansel would like to choose the twig as to guarantee that the other part (which he gets to keep) has as many berries as possible. Help him to determine how many berries he can guarantee for himself.

Hansel is using a rather peculiar terminology when talking about a bunch of grapes. Any such bunch has a tree structure: it consists of *grapes* and *twigs*, and every twig directly connects two different grapes. Any two grapes are connected with a uniquely determined sequence of twigs. One grape is distinguished and called the *root*. Grapes different than the root such that there is exactly one twig connecting them to other grapes are called the *berries*, and all other grapes are called the *joints*.

Input

The first line of input contains one integer n ($2 \le n \le 1\,000\,000$) denoting the number of grapes in the bunch. Grapes are numbered from 1 to n, where the root has number 1. The next $n-1$ lines describe the twigs, each in a separate line. Each of these lines contains two integers a and b ($a \ne b$, $1 \le a, b \le n$) denoting that grapes with the corresponding numbers are directly connected with a twig.

Output

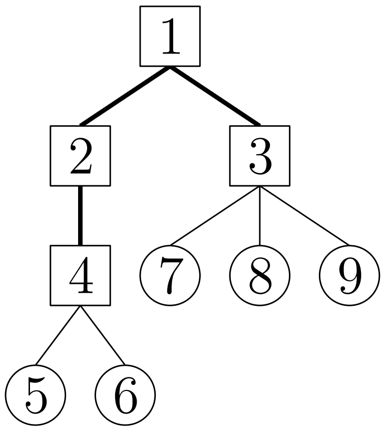
The first and only line of output should contain the maximum number of berries that Hansel can guarantee for himself by choosing the twig appropriately.

Example

Input	Output

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

The following figure shows the bunch from the example. Each berry is represented by a circle, and all other grapes (that is joints) are represented by squares. Segments connecting circles and squares represent twigs. Johnny can guarantee getting berries with numbers 5 and 6 by breaking any of the twigs represented by bold segments. The remaining berries with numbers 7, 8, 9 go to Gretel.


1
2
3
4
7
8
9
5
6

E: Taxi

Memory limit: **128 MB**

Taxi rides in Byteland are quite expensive, but at least the taxi cabs are spacious. Thus, group rides are very popular. The are n carrier companies, each characterized by three numbers:

- the maximum number c_i of passengers in a taxi cab,
- the price s_i for the first kilometer of a trip,
- the price p_i for each next kilometer of a trip.

Each company can supply any number of taxi cabs, but all cabs from one company are identical.

Johnny noticed a niche in the market and decided to become a middleman in ordering of taxis. He'll have to handle requests of the form: "order a taxi for m people on a distance of d kilometers" and has to find the cheapest possible combination of taxi cabs to fulfill it assuming that passengers are not willing to change taxis during the trip. His idea hit the jackpot, his services quickly became very popular, and he now has too many requests to handle them manually. You were hired to help with automating that process.

Input

The first line of input contains two integers n ($1 \leq n \leq 10^5$) and q ($1 \leq q \leq 10^5$), separated by a single space, and denoting respectively the number of carrier companies and the number of requests to be processed.

Each of the next n lines contains three integers separated by single spaces. In the i^{th} of those lines, the integers are c_i, s_i, and p_i, ($1 \leq c_i \leq 15$, $0 \leq s_i, p_i \leq 10^6$), denoting respectively the capacity of a single cab, the price for the first kilometer, and the price for each next kilometer.

Each of the next q lines contains two integers separated by a single space. In the i^{th} of those lines, there are m_i and d_i, ($1 \leq m_i \leq 10^6$, $1 \leq d_i \leq 10^6$), denoting respectively the number of people and the distance in i^{th} request.

Output

The output should contain exactly q lines, one for each request.

Each line of output should contain one integer, the minimum possible cost of fulfilling the request, in the same order as in the input.

Example

Input	Output

3 3
4 8 4
4 15 2
3 6 3
1 12
11 3
7 20
	37
	44
	106

For the first request (1 passenger, 12 km) the cheapest way is to order one cab from the second carrier for a price of $15 + 11 \cdot 2 = 37$.

For the second request (11 passengers, 3 km) the cheapest way is to order two cabs from the first carrier for 8 passengers, and one cab from the third carrier for the remaining 3 passengers for a total price of $2 \cdot (8+2 \cdot 4) + (6+2 \cdot 3) = 44$.

For the third request (7 passengers, 20 km) the cheapest way is to order two cabs from the second carrier for a price of $2 \cdot (15 + 2 \cdot 19) = 106$.

F: Antisocial network

Memory limit: **128 MB**

Johnny is the creator of the first ever social network, but hardly anyone knows about it. This is because while he was busy trying to find that one bug in his code, his colleague working on the project shabbily took over the company and fired Johnny. Johnny decided to take revenge by using the bug which he didn't manage to fix: he can *flip* some subsets X of users. Flipping X changes the friendship relation for all pairs of users in X, that is if a and b $(a, b \in X)$ were friends, then after flipping they no longer are, and vice versa. Johnny thinks that users who lose all their friends, once all flips are done, will delete their accounts, and that if many do so, the stolen project will surely die. Johnny spent the whole night looking for a good sequence of flips, and eventually came up with a sequence of subsets of users V_1,V_2,…,V_k. Before he finally went to sleep he was sure that the sequence he found will make all users delete their accounts, but this morning he is no longer so sure. He may have made a mistake, given his sleep deprivation, anger at the colleague, and questionable programming skills. That's why you should verify for each user if after the flips suggested by Johnny the user would have no friends.

Input

The first input line contains three integers n, m, and k $(1 \le n \le 2.5{\cdot}10^5, 1 \le k \le 10^4, 0 \le m \le \min\left\{\frac{n(n-1)}{2},\ 2 \cdot 10^6\right\})$, separated by single spaces. These denote, respectively: the number of users of the social network, the number of friend pairs in the network, and the length of sequence of operations suggested by Johnny.

In the next m lines there are pairs of integers a_i, b_i that specify the identifiers of users that are friends $(1 \le a_i, b_i \le n$, $a_i \ne b_i)$; these numbers are separated by a single space. The (unordered) pairs of friends are distinct.

The next k lines describe the sets V_i. Each of those lines contains sequence of integers s_i, v_{i_1}, …, $v_{i_{s_i}}$, separated by single spaces, where s_i $(1 \le s_i \le n)$ is the size of V_i and v_{i_j} are the elements.

The sum of s_i is not larger than 10^6.

Output

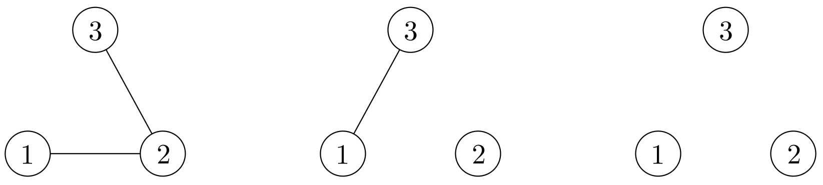
The output should contain n lines: i^{th} line should contain the word `TAK` if after the sequence of flips proposed by Johnny the i^{th} user will have no friends, and `NIE` otherwise.

Example

Input	Output

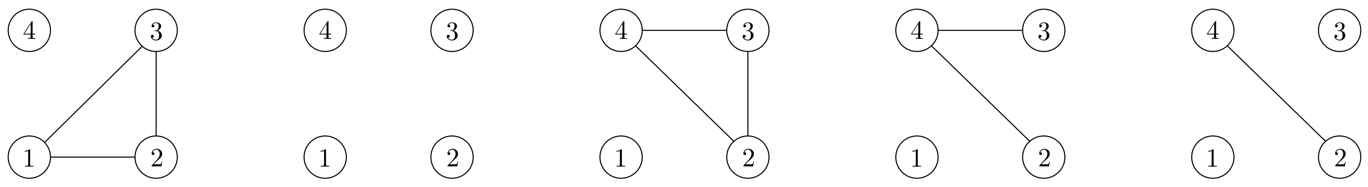
3 2 2
1 2
2 3
3 1 2 3
2 1 3
	TAK
	TAK
	TAK

The diagram showing the friendship relation initially and after each operation:



Input	Output
`4 3 4`	`TAK`
`1 2`	`NIE`
`2 3`	`TAK`
`1 3`	`NIE`
`3 1 2 3`	
`3 2 3 4`	
`2 2 3`	
`2 3 4`	

The diagram showing the friends relation in the beginning and after each operation:



G: Parking Lot

Memory limit: **128 MB**

The marketplace of town M is a popular tourist attraction, but it's hard to get there without a car due to a very peculiar traffic policy. The traffic department decided to remove all trams and buses from town centre as they were impeding car traffic. But the cars require parking places which there aren't many of in the town centre. Thus, the small streets around marketplace were converted into very long parking lots for parallel parking. Young Johnny doesn't have a car yet but he's already vigorously advocating for this plan, because he got a job as a controller for one of the new parking lots. Unfortunately he's not so good at his new job, so he'll definitely need your help.

The street Johnny was appointed to, has length d. His job is to tell the approaching drivers where they should park their cars or inform them that their car is too long to fit. Formally he's supposed to serve two kinds of events:

- a car with length l_i and license plate p_i approaches,
- the car with license plate p_i leaves.

When a car approaches, Johnny has to find the shortest available contiguous unoccupied space where the car would fit. If there is more than one such place, he should choose the one that is closest to the marketplace (that is the earliest, looking from the start of parking lot). Car drivers are the masters of parallel parking and their vehicles are so swift that they always park at the very beginning of the space that Johnny finds; to the point that they can touch the preceding car, and in extreme situations they can even touch with both the preceding and the following car. Such tight parking is not a problem even when leaving the parking lot.

Input

The first line of input contains two integers d and q, separated by a single space, which denote the length of parking lot and the number of events respectively ($1 \leq d \leq 10^9$, $1 \leq q \leq 10^6$).

Each of the following q lines begins with a single letter `P` or `O`, which denote the kind of event (respectively approach and departure of a car), followed by a single space and a license plate p_i, which is a nonempty string of up to 10 characters, each of which is either small Latin letter (`a-z`) or a digit (`0-9`).

If the line describes approach of a car, then it also contains, after a single space, an (integer) length l_i ($1 \leq l_i \leq 10^9$) of the car.

Each car will try to find a parking place at most once: tourists never visit the marketplace more than once, this is even more true if they couldn't find parking place.

If the line describes departure of a car, then it contains nothing further. However, the license plate p_i is guaranteed to correspond to a car that already tried to find parking place. Note that the car didn't necessarily find the parking place and even in that case doesn't have to depart immediately (that is other cars can depart before it) or at all.

Output

The output should contain exactly q lines, each describing the result of the next event from input.

The result of an event describing the approach of a car is a single integer o_i – the position (counting from the beginning of parking lot) on which the car should be parked, or the word `NIE`, if it's not possible to park the car.

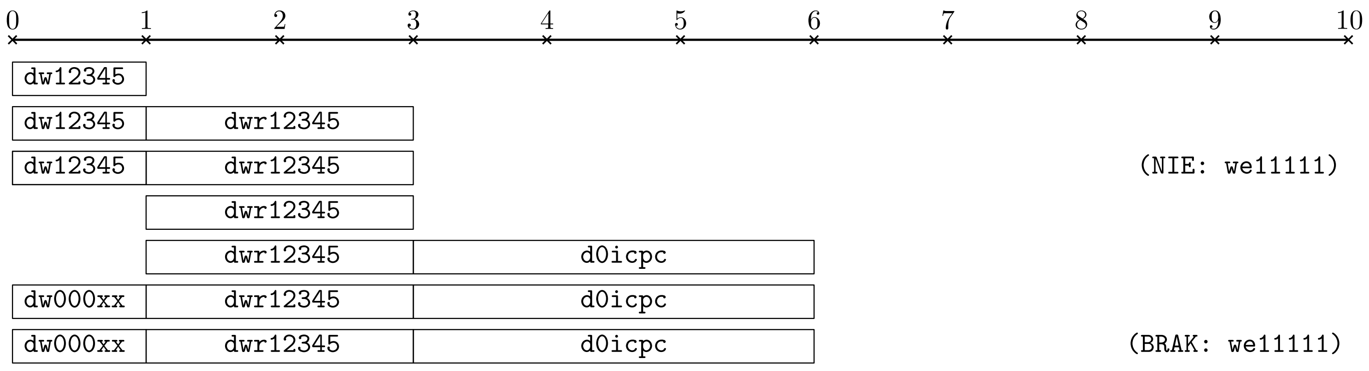
The result of an event describing the departure of a car is the word `OK`, if the car has actually parked and is now leaving or the word `BRAK` otherwise.

Example

Input	Output

`10 7`
`P dw12345 1`
`P dwr12345 2`
`P we11111 8`
`O dw12345`
`P d0icpc 3`
`P dw000xx 1`
`O we11111`
	`0`
	`1`
	`NIE`
	`OK`
	`3`
	`0`
	`BRAK`

The state of parking lot after each operation is shown in the following diagram.



The cars `dw12345` and `dwr12345` park one after another, at positions 0 and 1 respectively. The car `we11111` is too long and can't park. After the car `dw12345` departs, `d0icpc` parks just behind `dwr12345` at position 3, and then `dw000xx` parks at position 0, which has been made free by `dw12345`. Since `we11111` didn't find a parking place, the answer to its departure is `BRAK`.

H: Anagrams

Memory limit: **128 MB**

An *anagram* of a word w is any word obtained by rearranging (permuting) letters of w, including word w itself. Anagrams are a passion of Johnny ever since he started playing Scrabble: whenever he has at his disposal some collection of letters arranged as a word w he immediately starts rearranging them in all possible ways, counting how many anagrams the word w has. Apparently Johnny is more interested in anagrams than Scrabble, as it doesn't matter for him at all, whether any of w's anagrams has a meaning and occurs in dictionary. Johnny quickly noticed that from different words of the same length he can obtain different numbers of anagrams. When he decided that he knows how to efficiently compute the number of anagrams of a given word he started to ponder the opposite problem, that is, how long is the shortest word consisting only of small Latin letters (`a`–`z`) that has exactly n anagrams. This question turned out too hard for Johnny, so is counting on your help with it.

Input

The first and only input line contains a single integer n ($1 \le n \le 10^{12}$), the number of anagrams Johnny wants to obtain.

Output

In the first and only output line you should print one integer: minimum possible length of a word consisting of small Latin letters (`a`–`z`) that has exactly n anagrams.

Example

Input	Output
12	4

Word `baca` has 12 anagrams. Any shorter word has at most 6 anagrams, just like, e.g., `bac`.

I: Mountain hike

Memory limit: **128 MB**

Johnny is particularly fond of hiking. Unfortunately, his knees are not what they used to be anymore, and descending is particularly difficult for him. Hence his plans for the next Saturday are as follows: he will start the hike at the Valley of Three Lakes and then ascend the Mount Doom. Then, he will take a bus to the Valley of Five Lakes and ascend the Misty Mountain. Because of the knees problem, it is crucial that both parts of the trip (ignoring the bus drive) consist only of ascending segments. Johnny has already prepared a list of all ascending segments and determined the length of every such segment. Each segment connects two locations. As he gets bored rather easily, both parts of his trip should be also strictly disjoint. Help Johnny to determine the whole hike so that the total walking distance is as small as possible.

Input

The first line of input contains two integers n and m, separated by single space, which denote the number of locations and the number of segment respectively ($2 \leq n \leq 1\,000$, $1 \leq m \leq 10\,000$). Each of the following m lines contains three integers a_i, b_i, and x_i, separated by single spaces, which describe the i-th possible segment: from location a_i to location b_i of length x_i ($1 \leq a_i, b_i \leq n$, $1 \leq x_i \leq 10^6$); every such segment is ascending, that is height above mean sea level of its end location is strictly larger than height above mean sea level of its start location.

The next line contains integers s_1, t_1, and finally the last line contains s_2, t_2 ($1 \leq s_i, t_i \leq n$), separated by single spaces, which denote the start and end locations of both parts of the trip, that is, Valley of Three Lakes, Mount Doom, Valley of Five Lakes, and finally Misty Mountain respectively. You can assume that $s_1 \neq t_1$ and $s_2 \neq t_2$, but should not make any further assumptions.

Output

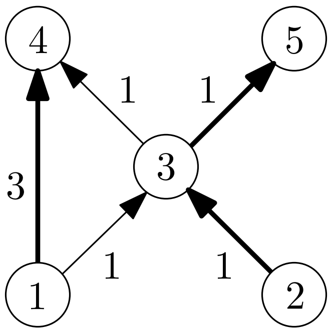
The output should consist of exactly one line, containing either the smallest possible total walking distance or the word `NIE` if it is not possible to plan the hike according to Johnny's requirements.

Example

Input	Output

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

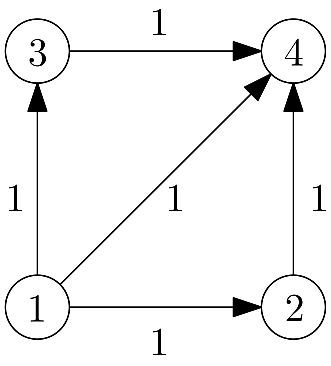
The above example describes the following locations and segments. There is exactly one way of choosing disjoint paths from 1 to 4 and from 2 to 5, shown in bold. The total walking distance of the hike corresponding to this choice is 5.



Input	Output

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

The above example describes the following locations and segments. It is not possible to start at 2 and reach 3.



J: Rock, paper, scissors

Memory limit: **128 MB**

Johnny is playing *rock, paper, scissors* with his friend Bob. The game consists of separate rounds during which both players at the same time show with their hands a symbol of either rock, paper, or scissors. The player who shows a stronger symbol wins the round; if both players show the same symbol the round is considered a tie. The following rules show the relative strength of symbols:

- scissors cut paper (scissors are stronger than paper),
- paper covers rock (paper is stronger than rock),
- rock crushes scissors (rock is stronger than scissors).

The player who wins more rounds wins the entire game.

Mastering the game is Bob's one and only goal in life. After quite a few years of training, he prepared a long list of symbols that he considers to be the best possible strategy, and memorized it. Johnny has accidentally found a printout with the sequence and of course decided to prank Bob: he wants to win the game (that is to win strictly more rounds than his friend), and he wants to do so "epically". The win is epic when Johnny changes the symbol he's showing the least number of times possible. There's not much time until the match begins and Johnny still doesn't know what is the minimal number of changes he needs to make. Help him compute it.

Input

The first and only line of input contains a string of letters of length no greater than 10^6, where letter at position i specifies the symbol that Bob will show during round i. Each letter is either `K`, `P` or `N`, which denote respectively rock, paper, and scissors.

Output

In the first and only line you should output a single integer: the minimum possible number of changes of shown symbol that Johnny needs to make in order to win the game.

Example

Input	Output
`KPNPKP`	`0`

Johnny can win this game by one point without making any changes, he just needs to show scissors all the time. Then:

- he wins three rounds (second, fourth, and sixth), in which Bob shows paper,
- he ties in third round, in which Bob also shows scissors,
- he loses two rounds (first and fifth), in which Bob shows rock.

Input	Output
`KKPPNN`	`1`

Here if Johnny would make no changes he can only end the game with a tie. If he makes a single change there are many ways to win this game. One of possible ways is to show paper in the first two rounds and then change symbol to scissors for the rest of the game. Then:

- showing paper he wins first and second round, in which Bob shows rock,
- showing scissors he wins third and fourth round, in which Bob shows paper,
- continuing to show scissors he ties fifth and sixth round, in which Bob also shows scissors.

K: Johnny-Bohr model

Memory limit: **128 MB**

Johnny has been sick lately and he missed a few classes. He didn't make much of it with a single exception: physics is his passion, so he borrowed lecture notes from his friend. From those notes he learned that he missed lecture about Bohr model of hydrogen atom and... apparently he didn't learn anything else, because based on the notes he reconstructed the following "Johnny-Bohr model"[1]

The electron orbiting around nucleus of a hydrogen atom can only stay at orbits with specific radii. Each of those radii is associated with a particular energy of the electron and its angular momentum. The angular momentum is quantized, that is, it can take any value that is a positive integer multiple of the reduced Planck constant $\hbar$ (also known as Dirac constant). Due to this it is useful to number orbits with natural numbers, such that orbit number n corresponds to angular momentum of $n \cdot \hbar$. An electron can jump from orbit n_1 to a lower orbit n_2 ($n_2 < n_1$) — in that case the energy difference is emitted as a photon with appropriate frequency and wavelength. Johnny-Bohr model does not foresee the opposite situation, that is, absorption of a photon by atom and transfer of an electron to a higher orbit. The model precisely specifies to which lower orbits an electron can jump. In his friend's notebook a multiset B is written which has the following property: an electron can jump from orbit a to an orbit $\lfloor \frac{a}{b} \rfloor$ for any $b \in B$. The electron can then jump from a new orbit a' again according to that rule. In particular the Johnny-Bohr model allows an electron to jump to the (final) orbit number 0, that is, to "fall" onto the nucleus. Just as with the original Bohr model, Johnny can't prove that this model really describes the way hydrogen atom behaves, however he can assure you that it agrees with experimental data (at least the data from Johnny's thought experiments). Proud of his deep understanding of the problem, Johnny noticed that it gives raise to a few interesting questions which may help him better understand the way atoms behave. The most important of the questions is the following: *"If an electron starts at an orbit number n, to how many different orbits (including n) can the electron move in any number (possibly zero) of jumps?"*

Input

The first line of input file contains two integers n and m ($1 \le n \le 10^{15}$, $1 \le m \le 10$), denoting the starting orbit number of the electron and the size of multiset B, respectively. The second (and last) line of input file contains sequence of m integers b_i, $1 \le b_i \le 10^{15}$, separated by single spaces; those are the elements of the multiset B.

Output

In the first and only line you should output a single integer – the number of different orbits on which the electron could end up after any number of jumps starting from orbit number n.

Example

Input	Output
20 2 2 3	8

Starting from an orbit $n = 20$ and with $B = \{2, 3\}$, an electron can jump (in 0 or more jumps) to any of the following eight orbits: $20, 10, 6, 5, 3, 2, 1, 0$.

[1] Even though the model has name of one of the greatest physicists it has not much to do with physics: Johnny's friend isn't particularly skilled when it comes to taking notes, and Johnny's fever is not helping!

L: Constitutional Tribunal

Memory limit: **128 MB**

The Constitutional Tribunal will be re-organized due to (apparently) low efficiency of their work. Before that happens the Tribunal will not accept any new pending laws until Chief Judge, the Judge number 1, makes a ruling on all the pending laws that were brought to the Tribunal before the re-organization. All of his subordinates, the Judges with numbers greater than 1, work according to old schedule: for each $i > 1$, the Judge number i comes to work at 8:00 and immediately takes c_i pending laws from his inbox (or all of them if there are less than c_i pending laws in his inbox), which he then evaluates, and at the end of the day, at 16:00, he puts all of the evaluated laws in his outbox. After 16:00 the laws are transferred to the next Judge: from outbox of Judge number i the laws are moved to inbox of his superior – Judge number p_i. Of course the final superior of all Judges is the Chief Judge, that is, the Judge number 1. The Chief Judge wants to know, when all the pending laws, will end up in his inbox. For this he came to work earlier today, and personally counted how many pending laws are in the inboxes of other Judges. It turned out that inbox of Judge number i (for $i > 1$) contains s_i pending laws. Help the Chief Judge to compute how many days will pass until all the pending laws will end up in his inbox, so that he can finally make a ruling on them.

Input

The input contains multiple testcases. The first line of input contains the number of testcases Z ($Z \geq 1$). Then all the testcases follow.

Each testcase begins with a line with a single integer n ($2 \leq n \leq 5000$), denoting the number of Judges in the Constitutional Tribunal. The next line contains $n-1$ integers $s_2, s_3, \ldots, s_n$, separated by single spaces, where s_i ($0 \leq s_i \leq 10^9$) denotes the number of pending laws that are currently in the inbox of Judge number i. The next line contains $n-1$ integers $p_2, p_3, \ldots, p_n$, separated by single spaces, where p_i ($1 \leq p_i < i$) means that Judge i is subordinate of a Judge number p_i. The last line (for this testcase) contains $n-1$ integers $c_2, c_3, \ldots, c_n$, separated by single spaces, where c_i ($1 \leq c_i \leq 10^9$) is the number of laws that Judge number i evaluates every day.

The sum of n over all testcases is no larger than 5000.

Output

For each testcase you should output a single line containing one integer: the number of days that will pass until all the pending laws will end up in Chief Judge's inbox.

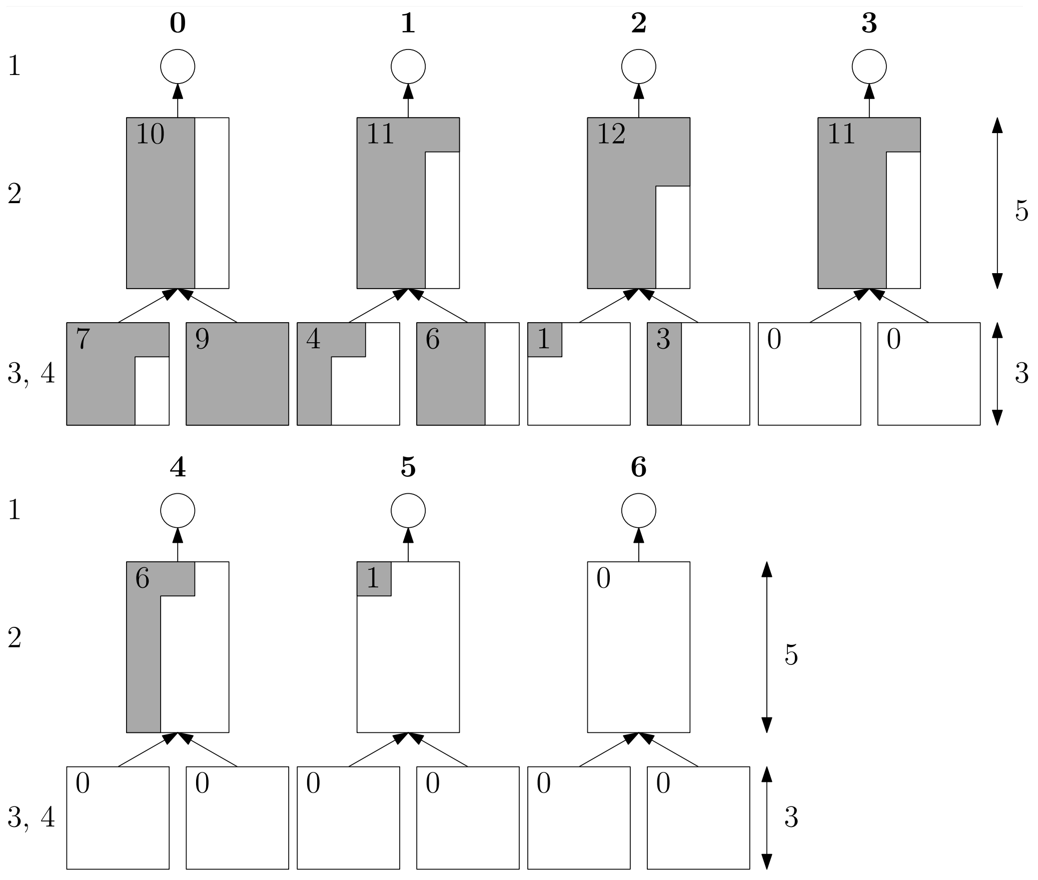
Example

Input	Output

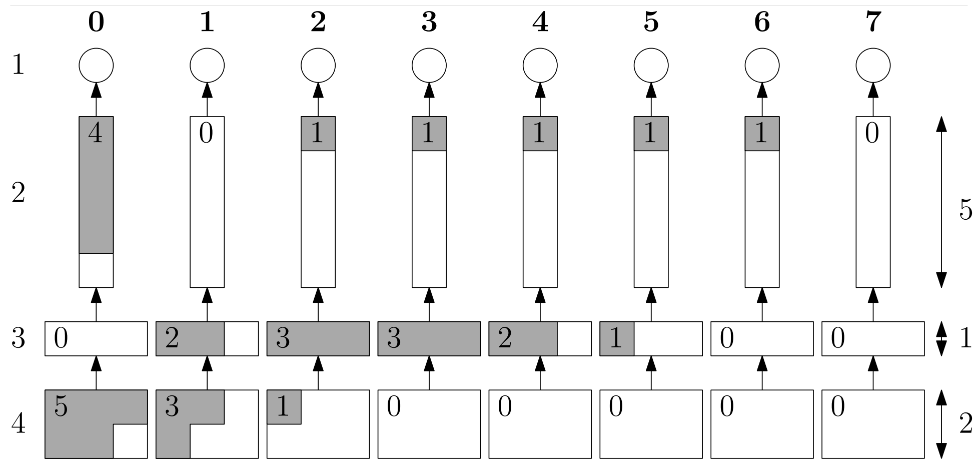
2
4
10 7 9
1 2 2
5 3 3
4
4 0 5
1 2 3
5 1 2
	6
	7

The diagram shows the state in the morning of each day. The numbers of the days are written in bold above, and the numbers of the Judges are written on the left. The Chief Judge is denoted by circle, while for all the other judges their inboxes are represented by rectangles. The arrows denote the superiority relation. The rectangle representing Judge number i inbox has height c_i, marked on the right and width large enough to hold all the pending laws in his inbox. The laws in inbox are shaded, and their number is also written in the top left corner. Judge number i ($i > 1$) evaluates up to c_i laws every day — graphically this corresponds to "removing" the first column of laws from his inbox, and "shifting" of the remaining laws to the left. Removed laws are (at the end of the day) moved to the rectangle of superior Judge p_i.

First testcase:



Second testcase:


0
1
2
3
4
5
6
7
1
2
3
4
4
0
1
1
1
1
1
0
5
0
2
3
3
2
1
0
0
1
5
3
1
0
0
0
0
0
2