The 2nd Universal Cup

Stage 19: Estonia

January 20-21, 2024

This problem set should contain 10 problems on 17 numbered pages.

Based on

Osijek Competitive Programming Camp

Problem A. Alternating Paths

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	256 megabytes

Next year, the Overly Complicated Problem Colloquium will take place in the Greatly Magnificent City. In order to impress the guests, several construction projects are underway. This includes painting all the city streets.

As one might expect, the city consists of n junctions and m bidirectional streets connecting them. The junctions are indexed $1 \ldots n$. No pair of junctions is connected by more than one street, no street connects a junction to itself and it is possible to walk from any junction to any other using these streets.

Each street should be painted either red or blue. The mayor thinks that it's a lot more interesting to walk around the city if every street is different from the last. Therefore, the mayor has issued an additional constraint: "if p and q are different junctions, then it must be possible to walk from p to q such that every street on the path is painted in a different color from the street before it." Such a path may also visit some streets or junctions multiple times.

Your task is to suggest a way to paint all the streets so that the mayor is satisfied or to claim that this isn't possible.

Input

The first line contains one integer t ($1 \leq t \leq 1000$) — the number of test cases. t test cases follow. Each test case is described as follows.

The first line of the test case contains two integers n and m — the number of junctions and streets, respectively ($2 \leq n \leq 100$, $1 \leq m \leq 300$). Each of the following m lines consists of two integers u and v ($1 \leq u \leq n$, $1 \leq v \leq n$), denoting a street between junctions u and v.

It is guaranteed that in each input file, there are at most 100 test cases where $n > 50$ or $m > 150$.

Output

For each test case, print the answer on a separate line as follows:

- If satisfying the mayor isn't possible, print `IMPOSSIBLE`.
- Otherwise, print a string of length m. The i-th character in the string should be `B` if the i-th road is blue, and `R` otherwise.

Example

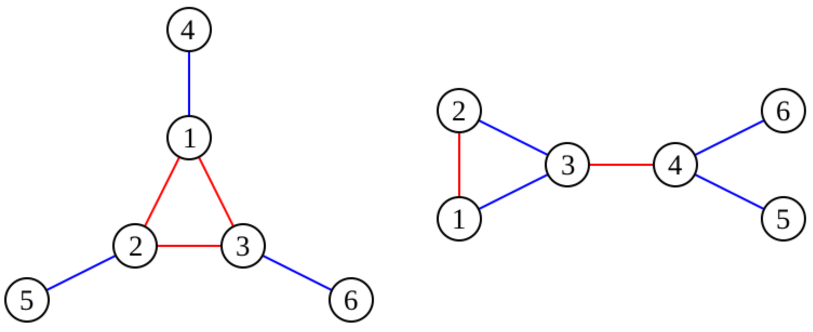
standard input	standard output

3
6 6
1 2
2 3
3 1
4 1
5 2
6 3
 	
6 6
1 2
2 3
3 1
3 4
4 5
4 6
 	
4 3
1 2
4 2
2 3
	RRRBBB
	RBBRBB
	IMPOSSIBLE

Note

The extra line breaks between test cases in the example are to aid with reading, they are not present in the actual input.

The following graphs illustrate a possible coloring for the first two test cases.



In the third test case, however you paint the edges, there will always be a pair of leaves with the same color leading into them, making the condition impossible to satisfy.

Problem B. Binary Sequence

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

A well-known puzzle is: "what is the next element in the sequence $1, 11, 21, 1211, 111221, 312211$?". In this problem we consider the binary variant.

The first element of the *binary look and say sequence* is 1. Each next element is generated by looking at the previous element and counting the number of digits in each group of consecutive similar digits. For example, if an element of the sequence is 110001, then the next element will be 10111011 (10 1, 11 0, 1 1, that is: two 1s, three 0s, one 1). Notice that the counts of similar digits are written down in binary.

For example, the first few elements of the sequence are generated as follows:

- 1 is read as "one 1", thus the second element is 11.
- 11 is read as "two 1s" i.e. "10 1", thus the third element is 101.
- 101 is read as "one 1, one 0, one 1", i.e, "1 1, 1 0, 1 1", thus the fourth element is 111011.
- 111011 is read as "three 1-s, one 0, two 1-s", i.e. "11 1, 1 0, 10 1", thus the fifth element is 11110101.

Your task is to calculate the m-th binary digit from the right of the n-th element in the sequence. The first element of the sequence has index 1; the rightmost binary digit of an element has index 0. If the element has m or fewer binary digits, output 0.

Input

The first line of the input contains one integer t ($1 \le t \le 10^5$) — the number of test cases. t test cases follow.

Each test case consists of two integers n and m ($1 \le n \le 10^{18}$, $0 \le m < 10^6$).

Output

For each test case, print the answer on a separate line.

Example

standard input	standard output

10
4 0
4 1
4 2
4 3
4 4
4 5
4 6
6 3
6 7
118999881999119725 3
	1
	1
	0
	1
	1
	1
	0
	1
	1
	0

Note

The fourth element of the sequence is 111011. The first six tests print out this number in reverse: as we are increasing m, we are moving from right to left. The seventh test, `4 6` returns 0 as there are only six digits in the number 111011.

Problem C. Yet Another Balanced Coloring Problem

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

You are given two rooted trees with n and m vertices, respectively. The vertices are indexed $1 \dots n$ (resp. $1 \dots m$) and the root is the vertex n (resp. m). Both trees have k leaves and in both trees, the leaves are precisely the vertices with indices $1 \dots k$. Here, the root of a tree isn't considered a leaf, even if it has only one neighbor.

For each i in $1 \dots k$, you have to choose red or blue. Then you have to paint the i-th vertex in both trees with the selected color.

After coloring the leaves, the following must hold in both trees:

- For each vertex u, the number of red leaves in the subtree of u must not differ from the number of blue leaves in the subtree of u by more than one.

Input

The first line contains one integer t ($1 \leq t \leq 10^5$) — the number of test cases. t test cases follow. Each test case is described as follows.

The first line of the test case contains two integers n and m ($3 \leq n, m \leq 10^5$).

The second line contains $n-1$ integers $p_1, \dots, p_{n-1}$ ($i < p_i \leq n$); the i-th of them denotes an edge between i and p_i in the first tree.

The third line contains $m-1$ integers $q_1, \dots, q_{m-1}$ ($i < q_i \leq m$); the i-th of them denotes an edge between i and q_i in the second tree.

It is guaranteed that in both trees, exactly the vertices $1 \dots k$ are leaves. It is guaranteed that the sum of $n+m$ over all test cases doesn't exceed $2 \cdot 10^5$.

Output

For each test case, print the answer on a separate line as follows.

- If there is no solution, print `IMPOSSIBLE`.
- Otherwise, print a string with length k. The i-th character in the string should be `B` if the i-th leaf is blue, and `R` otherwise.

Example

standard input	standard output

2
7 7
5 5 6 6 7 7
5 6 5 6 7 7
5 4
4 4 5 5
4 4 4
	RBBR
	RBB

Problem D. Filesystem

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

The Red Monster is preparing a contest for the Overly Complicated Problem Colloquium. The Red Monster has a number of files that need to be uploaded to the contest preparation system, Polytope. All of the files are in the same folder; each file has a creation date and a file name. All file names and creation dates are distinct.

The interface for uploading files looks like this:

File name	Creation date
checker.cpp	17.03.2023
clever_generator.cpp	18.09.2022
doit.sh	15.12.2022
examples.txt	10.12.2021
generator.cpp	07.05.2021
monsters.Inc.3D.2001.1080p.BluRay.Half-OU.DTS-ES.x264-HDMaNiAcS.avi	17.10.2021
not_a_virus.exe	07.06.2022
sol.cpp	18.06.2021
spxG7HoMSMHH225xjadbnA.tmp	06.07.2023
tutorial.tex	03.02.2021
xxx_my_password_DO_NOT_HACK.docx	10.01.2023
validator.cpp	15.01.2023

In one operation, the Red Monster does all of the following:

- Sorts the files in the folder, either alphabetically by file name or by the creation date.
- Chooses a contiguous segment of files and uploads those files to Polytope.

Note that files are not deleted after they are uploaded. Only a subset of files has to be uploaded to Polytope. The other files may be embarrassing and therefore must not be uploaded. For example, in the table above, only the files with bold file names should be uploaded; the rest have clearly nothing to do with the contest.

Each file must only be uploaded once, that is, there should not be any file that is uploaded in several operations. What is the minimum number of operations needed to upload exactly the necessary files?

Input

The first line contains one integer t ($1 \leq t \leq 1000$) — the number of test cases. t test cases follow. Each test case is described as follows.

Let n be the total number of files. We index the files $1 \ldots n$ and assume that the order of files when sorting by file name is $1, 2, 3, \ldots, n$.

The first line of the the test case consists of two integers n and k ($1 \leq k \leq n \leq 1000$) — the total number of files and the number of files that need to be uploaded.

The second line of the the test case consists of k integers $u_1, u_2, \ldots, u_k$ ($1 \leq u_i \leq n$, for all i; all u_i are pairwise distinct). These are the indices of the files that must be uploaded.

The third line consists of a permutation $p_1, p_2, \ldots, p_n$ of $1 \ldots n$. This indicates that the order of files when sorted by creation date is $p_1, p_2, \ldots, p_n$.

It is guaranteed that the sum of n over all test cases doesn't exceed 1000.

Output

For each test case, print the answer on a separate line — a single integer, the minimum number of operations needed to upload all files.

Example

standard input	standard output

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

Note

The first example test case corresponds to the example in the statement. Ordered by creation date, it looks as follows:

ID	File name	Creation date
10	**tutorial.tex**	03.02.2021
5	**generator.cpp**	07.05.2021
8	**sol.cpp**	18.06.2021
6	monsters.Inc.3D.2001.1080p.BluRay.Half-OU.DTS-ES.x264-HDMaNiAcS.avi	17.10.2021
4	**examples.txt**	10.12.2021
7	not_a_virus.exe	07.06.2022
2	**clever_generator.cpp**	18.09.2022
3	**doit.sh**	15.12.2022
11	xxx_my_password_DO_NOT_HACK.docx	10.01.2023
12	**validator.cpp**	15.01.2023
1	**checker.cpp**	17.03.2023
9	spxG7HoMSMHH225xjadbnA.tmp	06.07.2023

Observe that if we only ever sorted by the file name, we would need to use 4 operations. The same holds if we only ever sorted by creation date. A solution in 3 operations looks as follows:

- Sort by file name. Upload files 2, 3 and 4 (clever_generator.cpp, doit.sh and examples.txt). Note that if we also uploaded file 5 (generator.cpp) in this step, we would mess up the next step.
- Sort by creation date. Upload files 10, 5 and 8 (tutorial.tex, generator.cpp and sol.cpp).
- Sort by creation date. Upload files 12 and 1 (validator.cpp and checker.cpp).

There are other solutions with 3 operations. It can be proven that there is no solution with 2 operations.

In the second example test case, we want to upload exactly the files with odd indices. Whichever way we sort, there is never even a situation where two files we want to upload are consecutive. Therefore, every file must be uploaded separately.

Problem E. Freshman's Dream

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Everyone knows that `(a + b)^n` is never equal to `a^n + b^n` for positive integers a, b and n if $n \geq 2$. Or is it? Look again.

Given an integer $n \geq 2$, you have to find positive integers a and b such that `(a + b)^n` is equal to `a^n + b^n`, where **every symbol is interpreted as it is in C++**, including operator precedence. In other words, you have to find a and b such that

$$(a + b) \oplus n = a \oplus (n + b) \oplus n$$

holds, where $\oplus$ is the bitwise XOR operation.

Input

The first line contains one integer t ($1 \leq t \leq 10^5$) — the number of test cases. t test cases follow.

Each test case consists of one integer n ($2 \leq n < 2^{60}$).

Output

For each test case, print the answer on a separate line as follows.

- If there is no solution, print -1.
- Otherwise, print positive integers a and b ($1 \leq a, b < 2^{60}$) such that the equation in the problem statement holds. Under the constraints of the problem, it can be proven that if there is a solution, then there is also a solution with $a, b < 2^{60}$. If there are multiple solutions, you can print any one of them.

Example

standard input	standard output

5
2
3
6
10
18
	1 1
	-1
	3 5
	7 3
	11 39

Problem F. When Anton Saw This Task He Reacted With 😩

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	256 megabytes

Given two three-dimensional vectors $v = (v_x, v_y, v_z)$ and $w = (w_x, w_y, w_z)$, recall that their *cross product* is another three-dimensional vector defined as

$$v \times w = (v_y w_z - v_z w_y, v_z w_x - v_x w_z, v_x w_y - v_y w_x).$$

The cross product has length $|v|\,|w|\,|\sin \alpha|$, where α is the angle between v and w and its direction is such that $v \times w$ is at a right angle with both v and w. Notice that if v and w have integer coordinates, then so does $v \times w$.

Recall that the cross product is not associative: that is, $(u \times v) \times w$ is not necessarily the same as $u \times (v \times w)$. Therefore, to reformulate classical data structure problems such as "update some values, calculate the maximum/sum/gcd of the array" with cross products, we need to specify which way the brackets are positioned. You're about to solve one such problem. To avoid expression parsing, we reformulate the problem in terms of trees (see how considerate we are!).

You are given a tree with n vertices. The vertices are indexed $1 \ldots n$. Vertex 1 is the root. Each vertex is one of the following two types:

- *Internal vertex.* The vertex has exactly two children: a left child l and a right child r.
- *Leaf.* The vertex has no children. A vector (x, y, z) is written in the vertex.

The *value* $\text{val}(v)$ of a vertex v is a three-dimensional vector defined as follows:

- The value of an internal vertex v is defined as $\text{val}(l) \times \text{val}(r)$, where l and r are the left and right child of v, respectively.
- The value of a leaf v is defined as (x, y, z), where (x, y, z) is the vector written in the vertex v.

You are given q queries of the following form:

- Given a leaf v and a vector (x, y, z). Replace the vector written in the vertex v with (x, y, z). Then print the value of the root vertex, i.e. $\text{val}(1)$.

For each query, output the coordinates of the result modulo $P = 998\,244\,353$. That is, if the answer to a query is (w_x, w_y, w_z), you may output any triple (u_x, u_y, u_z) such that $w_x \equiv u_x \pmod{P}$, $w_y \equiv u_y \pmod{P}$ and $w_z \equiv u_z \pmod{P}$.

Input

The first line of the input consists of two integers n and q ($3 \le n \le 2 \cdot 10^5$, $1 \le q \le 10^5$).

The following n lines describe the vertices; the i-th of them describes the vertex i.

- If the i-th vertex is an internal vertex, then the line is of the form `x` l r ($2 \le l, r \le n$), where l and r are the indices of the left and right child, respectively.
- If the i-th vertex is a leaf, then the line is of the form `v` x y z ($0 \le x, y, z < 998\,244\,353$), where (x, y, z) is the vector written in the vertex i.

It is guaranteed that these lines describe a tree with root vertex 1.

The following q lines describe the queries. Each of them consists of four integers v, x, y, and z ($1 \le v \le n$, $0 \le x, y, z < 998\,244\,353$). It is guaranteed that v is a leaf.

Output

For each query, print the answer on a separate line — three integers x, y and z ($-2^{31} \le x, y, z < 2^{31}$), the coordinates of the answer to the query, modulo 998 244 353.

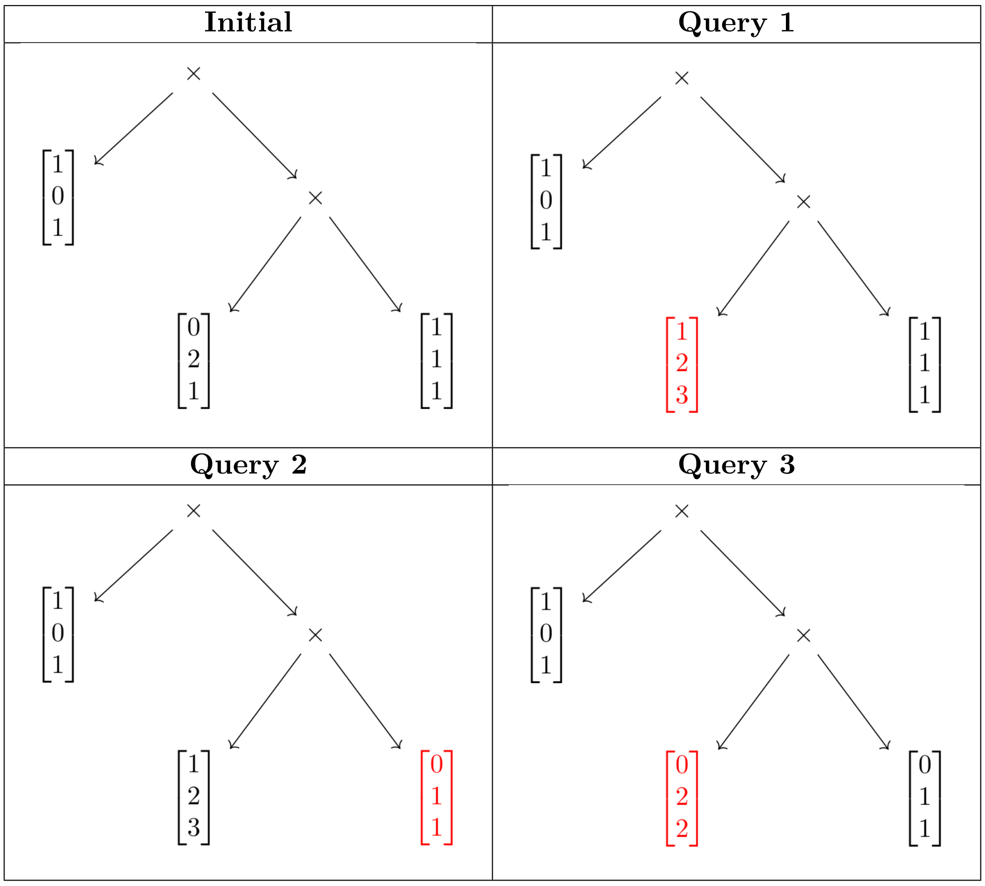
Example

standard input	standard output

`5 3`
`x 2 3`
`v 1 0 1`
`x 4 5`
`v 0 2 1`
`v 1 1 1`
`4 1 2 3`
`5 0 1 1`
`4 0 2 2`
	`998244351 0 2`
	`1 -2 998244352`
	`0 0 0`

Note

The following figures illustrate the state of the tree initially and after each query.



After the first query, the value of the root node is $(1, 0, 1) \times ((1, 2, 3) \times (1, 1, 1)) = (-2, 0, 2)$.
After the second query, the value of the root node is $(1, 0, 1) \times ((1, 2, 3) \times (0, 1, 1)) = (1, -2, -1)$.

Notice that, as per the problem statement, both `-2` and `998244351` are valid ways of outputting -2.

Problem G. LCA Counting

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

You are given a rooted tree with n vertices and ℓ leaves. The vertices are indexed $1 \dots n$ and the root vertex is 1. Exactly k leaves $u_1, \dots, u_k$ are chosen. Here, the root of the tree isn't considered a leaf, even if it has only one neighbor. What is the maximum cardinality of the set

$$\{\operatorname{lca}(u_i, u_j) \mid 1 \le i, j \le k\}?$$

Here, $\operatorname{lca}(u, v)$ refers to the lowest common ancestor of vertices u and v. Solve this problem for each k in $1 \dots \ell$, where ℓ is the number of leaves in the tree.

Input

The first line of the input consists of a single integer n ($2 \le n \le 2 \cdot 10^5$) — the number of vertices.

The second line consists of $n - 1$ integers $p_2, p_3, \dots, p_n$ ($1 \le p_i < i$), denoting an edge between p_i and i.

Output

Let ℓ be the number of leaves in the tree.

Print ℓ integers on a single line, the k-th of which is the answer to the problem if exactly k leaves are chosen.

Example

standard input	standard output
7 1 1 2 4 2 2	1 3 5 6

Problem H. Minimum Cost Flow^2

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

You may be familiar with the minimum cost flow problem. In the minimum cost flow problem, every edge e has a cost c; if you put f flow on that edge, you have to pay cf for using that edge. In other words, the cost of using an edge is linear in the amount of flow you put on it. This may be limiting. Maybe in some applications it is better to use a different kind of cost function.

In this problem you will solve one such variant: the cost of sending 1 unit of flow through a network where the cost of putting flow on an edge is quadratic in the amount of flow you put on it.

Formally, you are given an directed graph with n vertices and m edges. The vertices are indexed $1 \ldots n$, where vertex 1 is the source and vertex n is the sink. For each edge e, you are given a nonnegative cost c_e. You have to decide, for each edge e, a real number f_e: the amount of flow running through this edge. The numbers f_e must satisfy the following:

- $\displaystyle\sum_{e \in \text{out}(1)} f_e - \sum_{e \in \text{in}(1)} f_e = 1;$
- $\displaystyle\sum_{e \in \text{out}(u)} f_e - \sum_{e \in \text{in}(u)} f_e = 0$ for all $1 < u < n$;
- Among all valid choices, $\displaystyle\sum_e c_e f_e^2$ must be minimized.

Notice that there are no edge capacities. You may put as much flow on each edge as you like. Here, $\text{out}(u)$ and $\text{in}(u)$ represent the sets of outgoing and incoming edges to vertex u, respectively. You are allowed to set f_e to a negative number, this represents flow going in the opposite direction.

You have to find the minimum possible value of $\sum_e c_e f_e^2$. It can be proven that under the constraints of this problem, this is always a rational number. Output this rational number modulo 998 244 353.

Input

The first line contains one integer t ($1 \leq t \leq 100$) — the number of test cases. t test cases follow. Each test case is described as follows.

The first line of the test case consists of two integers n and m ($2 \leq n \leq 100$, $1 \leq m \leq 300$).

Each of the following m lines consists of three integers u, v and c ($1 \leq u, v \leq n$, $1 \leq c < 998\,244\,353$), representing an edge from u to v with cost c. There are no self-loops, parallel or antiparallel edges. It is guaranteed that the underlying undirected graph is connected.

The sum of $n + m$ over all test cases doesn't exceed 400.

The tests are constructed in such a way that there exists at least one optimal flow f, such that for each edge e, f_e is a rational number that can be expressed as $\frac{p}{q}$, where p and q are integers and $q \not\equiv 0 \pmod{998\,244\,353}$.

Output

For each test case, print the answer on a separate line — a single integer, the answer to the problem modulo 998 244 353.

Formally, let $P = 998\,244\,353$. It can be shown that the answer can be expressed as an irreducible fraction $\frac{p}{q}$, where p and q are integers. It can be shown that under the constraints of this problem, $q \not\equiv 0 \pmod{P}$. Output any integer equal to $p \cdot q^{-1} \pmod{P}$. In other words, output an integer x so that $-2^{31} \leq x < 2^{31}$ and $x \cdot q \equiv p \pmod{P}$.

Example

standard input	standard output

```
4
4 4
1 2 1
2 4 1
1 3 1
3 4 1
3 3
1 2 1
1 3 1
2 3 1
5 6
1 2 1
2 3 3
2 4 1
3 4 1
3 5 1
1 5 8
4 5
1 2 1
1 3 2
2 3 1
3 4 1
4 2 8
```
	```
	1
	665496236
	713031683
	614304219
	```

Problem I. Rebellious Edge

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

Given a directed graph where each edge has a nonnegative weight, and a root vertex r, the *directed minimum spanning tree problem* asks to choose a subset of edges such that:

- Using the selected edges, there exists a directed path from r to u for every vertex u.
- The chosen subset of edges forms a spanning tree of the graph if the directions of the edges are ignored.
- The total weight of the chosen edges is minimized.

Although there exist efficient algorithms for computing the directed minimum spanning tree, we look at a simple special case in this problem. Given is a directed graph with n vertices and m directed edges. The vertices are indexed $1 \ldots n$. All edges, **except for exactly one**, are directed from the vertex with the smaller index to the vertex with the bigger index. There are no parallel edges, i.e. for every pair of vertices (u, v), there is at most one edge $u \to v$. Find the directed minimum spanning tree, with root vertex 1.

Input

The first line contains one integer t ($1 \le t \le 10^5$) — the number of test cases. t test cases follow. Each test case is described as follows.

The first line of the test case consists of two integers n and m ($3 \le n \le 2 \cdot 10^5$, $n - 1 \le m \le 5 \cdot 10^5$). The following m lines each consist of three integers u, v and w, denoting an edge $u \to v$ with weight w ($1 \le u, v \le n$, $u \ne v$, $0 \le w \le 10^9$). For all but one of these lines $u < v$.

It is guaranteed that the sum of n over all test cases doesn't exceed $2 \cdot 10^5$ and the sum of m over all test cases doesn't exceed $5 \cdot 10^5$.

The tests are constructed in such a way that a directed spanning tree always exists.

Output

For each test case, print the answer on a separate line — a single integer, the total weight of edges in the directed minimum spanning tree.

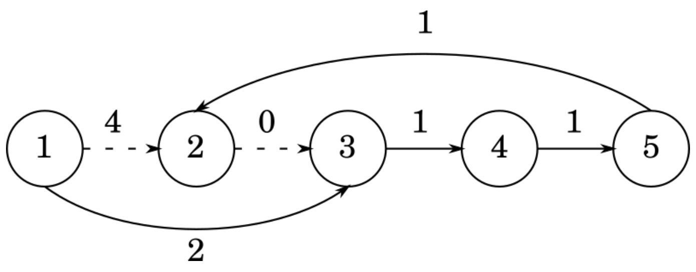
Example

standard input	standard output

3
5 6
1 2 4
1 3 2
2 3 0
3 4 1
4 5 1
5 2 1
4 4
1 2 4
1 3 6
1 4 8
4 2 1000000
3 3
1 2 100
2 1 10
2 3 1000
	5
	18
	1100

Note

The figure below illustrates the first example test case. The solid edges are the ones in the directed minimum spanning tree. Notice that if we didn't use the backwards edge, the total weight would have to be at least 6.



In the second example, the backwards edge has a very high weight, so there is no point in considering it.

Notice that although the statement guarantees that there are no parallel edges, antiparallel edges (such as seen in the third example) are allowed.

Problem J. 'Ello, and What Are You After, Then?

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	256 megabytes

You are playing *RunEscape*. *Slayer* is an activity where you repeatedly complete tasks of the form "slay x monsters of type y". You ponder what is the fastest way to train slayer.

There are n slayer masters you can get tasks from. The i-th slayer master has m_i tasks they can give you. You know the following information about each task:

- f_{ij} — the frequency of the task;
- t_{ij} — how many minutes the task takes to complete;
- e_{ij} — how much XP you gain per minute while doing the task.

When you complete a task you receive c slayer points. When you get a task you can spend s slayer points to skip it. You loop through the following steps:

1. Select a slayer master. Let i be the index of the chosen master.
2. Block up to b of their tasks leaving at least one unblocked. Let B be the set of indices of the tasks you blocked. The probability of receiving the j-th task becomes
$$P(j) = \begin{cases} \dfrac{f_{ij}}{\sum_{k \notin B} f_{ik}} & \text{if } j \notin B \\ 0 & \text{if } j \in B. \end{cases}$$
3. The slayer master randomly assigns you a task using P. You can either skip the task and lose s points or complete it and receive c points.
4. Go back to step 1.

You start with 0 slayer points. Calculate e, the maximum possible expected XP gain per minute such that you never go below 0 slayer points, assuming you make all of your choices optimally. See the Note section below for a formal definition of e.

Input

The first line of input contains three integers b ($0 \le b \le 3 \cdot 10^4$), c and s ($1 \le c, s \le 10^4$).

The second line of input contains a single integers n ($1 \le n \le 10^3$) — the number of slayer masters. The description of their tasks follows.

The first line for each slayer master contains a single integer m_i ($1 \le m_i \le 3 \cdot 10^4$) — the number of tasks the slayer master has.

The following n_i lines contain three integers f_{ij}, t_{ij} and e_{ij} ($1 \le f_{ij}, t_{ij}, e_{ij} \le 10^4$).

It is guaranteed that the sum of all m_i does not exceed $3 \cdot 10^4$.

Output

Print a single floating-point number: e.

Your answer will be considered correct if its absolute or relative error does not exceed 10^{-6}. Formally, let your answer be a, and the jury's answer be b. Your answer is accepted if and only if

$$\frac{|a-b|}{\max(1, |b|)} \le 10^{-6}.$$

Examples

standard input	standard output

`0 1 6`
`2`
`1`
`1 1 1`
`2`
`1 10 1`
`1 10 10`
	`7.000000000000`

`2 1 2`
`1`
`4`
`10 2 1`
`10 1 1`
`1 10 1`
`1 1 10`
	`5.909090909091`

Note

We now define e formally.

Let q be a natural number. Consider going through the loop for exactly q iterations. A *strategy* is a sequence of q tuples (i_k, B_k, t_k). The k-th of those tuples describes your actions on the k-th iteration:

- i_k is the index of the slayer master you will go to on the k-th step.
- B_k is the set of tasks you will block on the k-th step.
- t_k is a function $\mathbb{N}_0 \to 2^{\{1,2,\ldots,m_{i_k}\}}$. $t_k(p)$ describes the set of tasks you will skip if you have exactly p slayer points. If $p < s$, then $t_k(p) = \varnothing$.

For each strategy, its *efficiency* is defined as the expected amount of XP you gain, divided by the expected amount of time it will take. Let e_q be the maximum efficiency among all strategies.

Then

$$e = \lim_{q \to \infty} e_q.$$