Problem A. Who Can Win

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

In ICPC contests, the scoreboard is frozen during the last hour. Submissions made during the freeze are shown with result `Unknown`.

You are given multiple submission logs from old ICPC contests. However, the results of submissions made after the freeze are hidden. For each contest, determine all teams that could possibly be the champion.

Rules

Each submission is represented as a tuple:

$$(\text{teamname}, \text{problemid}, \text{time}, \text{result})$$

- **teamname**: a string of at most 10 characters, consisting of uppercase or lowercase letters and digits. It identifies the team that made the submission.
- **problemid**: a single uppercase letter (A–Z) that identifies the problem being attempted.
- **time**: an integer between 0 and 299, inclusive, representing the number of minutes from the start of the contest.
- **result**: the outcome of the submission:
  - `Accepted`: the submission correctly solves the problem.
  - `Rejected`: the submission does not solve the problem.
  - `Unknown`: the submission result is unknown because it was made after the scoreboard was frozen.
  - **Note: a submission has result Unknown if and only if its time $\geq$ 240.**

For a contest where all results are known, a team's score is determined by:

1. **Solved problems**: the number of problems for which this team has at least one `Accepted` submission.
2. **Penalty time**: the total time consumed by all problems that the team solved (0 if none).
   - For each solved problem, calculate the time as follows:
     (a) Take the submission time of the team's first Accepted submission for that problem.
     (b) Add 20 minutes for each of the team's submissions to the same problem that **occurred earlier in time** and had result `Rejected`.
   - A submission is considered earlier than another if its submission time is smaller.
   - **Note: a team will never make more than one submission in the same minute.**

A **champion team** is one with the highest number of solved problems and, among them, the smallest penalty time. Multiple champion teams may exist if they have the same number of solved problems and the same penalty time.

Please determine all teams that could possibly be champions for any scenario where each Unknown submission is replaced independently by either Accepted or Rejected.

Input

The first line contains an integer T ($1 \le T \le 10000$) — the number of contests.

For each contest:

- The first line contains an integer s ($1 \le s \le 10^5$) — the number of submissions for this contest.
- Each of the next s lines describes a submission in the format:

teamname problemid time result

It is guaranteed that:

- $T \le 10000$ and $\sum s \le 10^5$ over all contests.
- `teamname` consists of at most 10 uppercase/lowercase letters and digits.
- `problemid` is a single uppercase letter (A–Z).
- `time` is an integer between 0 and 299, inclusive.
- `result` is `Accepted`, `Rejected`, or `Unknown`.
- Each contest has at least one `Accepted` submission.
- A team will never make more than one submission in the same minute.

Output

Output T lines, one for each contest. Each line contains all team names that could possibly be champions for that contest, in **lexicographical order**, separated by spaces.

Example

standard input	standard output
3 4 ASYYPIbf7 D 268 Unknown 3NhYHv8w B 13 Accepted ASYYPIbf7 B 173 Accepted dnrkAsPqrA A 107 Accepted 6 T1 A 10 Rejected T1 A 241 Unknown T1 B 200 Accepted T2 A 100 Accepted T3 C 50 Accepted T4 D 250 Unknown 4 Alpha A 100 Accepted Beta B 100 Accepted Gamma C 240 Unknown Delta D 299 Unknown	3NhYHv8w ASYYPIbf7 T1 T3 Alpha Beta

Problem B. Creating Chaos

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

As a *Cappuccino Assassino*, you despise order. You want everything in the world to be as chaotic as possible.

In front of you are n cups of cappuccino, arranged in a line and numbered from 1 to n. You wish to drink k of them, so that the remaining cappuccinos are as disordered as possible.

Suppose the positions of the remaining cappuccinos are $a_1, a_2, \ldots, a_{n-k}$. The degree of order is defined as: $\sum\limits_{i=1}^{n-k} \sum\limits_{j=i+1}^{n-k} \gcd(|a_i - a_j|, n)$, where gcd represents the greatest common divisor.

You need to minimize this degree of order and output the sequence of cappuccinos you choose to drink.

If there are multiple answers, you can output any one of them.

Input

Given two integers n, k ($1 \le k \le n \le 1000$), separated by a space.

Output

k integers, representing the indices of the cappuccinos that are drunk.

Examples

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

Problem C. Canvas Painting

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

In the Kingdom of Flealand, there is a long canvas divided into n segments, numbered from 1 to n. Each segment is initially painted with a unique color, such that the color of segment i is $a_i = i$.

The royal artist Crysflea has received m magical paint spells. Each spell affects a continuous interval $[l, r]$ on the canvas. When a spell is cast, the artist may choose **any two positions** u and v within this interval ($l \le u, v \le r$), and repaint segment u with the color of segment v, i.e., perform $a_u = a_v$.

The spells can be used in **any order**, and each spell may be used at most once.

After all spells are used, the artist wishes to admire the canvas. Help the artist **minimize the number of distinct colors** that remain on the canvas.

Input

The input contains multiple test cases.

The first line contains a single integer T ($1 \le T \le 2 \times 10^5$) — the number of test cases.

For each test case: - The first line contains two integers m ($1 \le m \le 2 \times 10^5$) and n ($1 \le n \le 10^9$) — the number of spells and the length of the canvas. - The next m lines each contain two integers l and r ($1 \le l \le r \le n$) — describing one paint spell.

The total number of spells across all test cases satisfies $\sum m \le 2 \times 10^5$

Output

For each test case, output a single integer — the minimum number of distinct colors on the canvas after applying the spells.

Example

standard input	standard output

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

Problem D. Min-Max Tree

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 megabytes

You are given an unrooted tree with n vertices. The i-th vertex has a weight a_i.

The value of a connected component in the tree is defined as the difference between the maximum weight and the minimum weight among all vertices in the component. Your task is to delete some edges of the tree to partition it into several connected components, such that the sum of the values of all connected components is maximized.

Output the maximum possible sum of values after the partition.

Input

The first line contains an integer n ($1 \le n \le 10^6$), the number of vertices in the tree.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$), the weights of the vertices.

Each of the next $n-1$ lines contains two integers u and v ($1 \le u, v \le n$), representing an edge between vertex u and vertex v. It is guaranteed that the given edges form a tree.

Due to large input sizes, please use fast I/O methods.

Output

Output a single integer — the maximum possible sum of values of all connected components after partitioning the tree.

Examples

standard input	standard output

`8`
`1 3 7 5 2 8 4 6`
`1 2`
`1 3`
`1 4`
`3 5`
`5 6`
`3 7`
`7 8`
	`14`

`7`
`5 9 9 6 6 3 2`
`3 5`
`4 6`
`2 6`
`5 4`
`7 2`
`4 1`
	`13`

Note

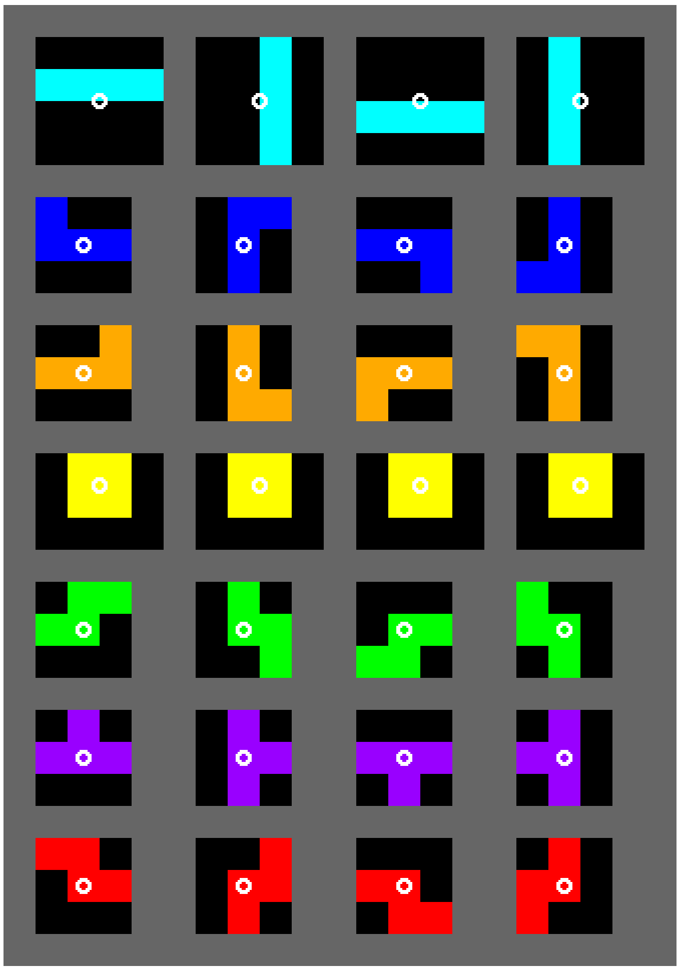
In the first test case, one of the optimal partitions is: $\{1,2,3,4\}, \{5,6\}, \{7,8\}$. The values of the 3 components are $6, 6, 2$, respectively.

Problem E. tetrart

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

You've hacked into a special version of Tetris that lets you freely choose which piece comes next. Your goal is no longer survival — it's now to create pixel art.

- The game field is a $h \times w$ rectangular grid, initially empty. Top left is $(0, 0)$ and bottom right is (h, w).**It is guaranteed that w is odd.**
- **Blocks can stack up to 20 rows above the visible grid before triggering a game over.**
- **There are 7 tetrominos, each made of 4 connected blocks, named after their distinctive shapes: I, J, L, O, S, T and Z. The figure below illustrates the 4 orientations of the 7 tetrominoes. The first column displays the initial orientation, and each subsequent column is obtained by rotating the previous column 90 degrees clockwise around the tetrominoes' center (indicated by the white circles).**



- **In each move, you can specify a triple (t, x, d), where t is an uppercase letter and x, d are integers. Then a tetromino t will appear, rotated $d \times 90°$ counterclockwise from its initial orientation, and its center coordinates, when rounded to nearest integers, are initially $(-\infty, x)$. The tetromino will drop down until it hits the bottom or any other block in the field and turn into blocks.**

- **Line Clear: When a horizontal row is completely filled with blocks, it disappears. All blocks above the cleared row shift down.**

You are given a grid the same size as the game field, representing a pixel art pattern. Your task is to construct a sequence of moves so that after performing them, all the '#'s in the grid are occupied by a block, and '.'s are empty.

Input

Each test contains multiple test cases.

The first line contains the number of test cases T ($1 \le T \le 10^4$).

The first line of each test case contains two integers, $1 \le h \le 100$ and $w \le 100$, where w is guaranteed to be odd. The following h lines each contain a string of length w, consisting of '.' and '#', representing the target pattern.

It is guaranteed that the sum of hw over all test cases does not exceed 10^5.

Output

For each test case:

If no solution exists, output -1. Otherwise, the first line should contain an integer $0 \le k \le 10hw$, denoting the number of operations in your construction. The next k lines each contain an uppercase character t, followed by two integers x and $0 \le d \le 3$, representing a move (t, x, d).

It can be proven that if a solution exists, there is one with no more than $10hw$ operations.

Example

standard input	standard output
2 3 5 ....#.. .###. 3 5 ..#.. ..... #...#	1 T 3 0 -1

Note

A checker is provided to help you verify the correctness of your solution locally. (hint: You may copy some code snippets from ‘checker.cpp‘ for writing your own solution.)

Compilation

Place checker.cpp and testlib.h in the same directory, then compile with:

```
g++ checker.cpp -o checker
```

To enable verbose mode (which prints the game field after each move), add the -DVERBOSE flag:

```
g++ checker.cpp -o checker -DVERBOSE
```

Usage

Run the checker with the following command:

```
./checker <input-file> <output-file> <answer-file>
```

Problem F. Robot

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

This is an interactive problem.

A mischievous robot is navigating an infinitely large 2D grid. However, its movement is confined to the first quadrant, meaning its coordinates (x, y) must always satisfy $x > 0$ and $y > 0$. Initially, the robot is located at grid coordinates (s_x, s_y). All grid cells are initially white.

The game proceeds in discrete time steps, up to a maximum of $T = 1000$ steps. In each time step, the following sequence occurs:

1. **Your Move:** You choose integer coordinates (x_m, y_m) such that $1 \le x_m \le T$ and $1 \le y_m \le T$. You then mark this cell black. A cell, once marked black, remains black for the rest of the game. You can mark any cell within the range, including the cell where the robot is currently located.

2. **Robot's Move:** The robot, currently at position (r_x, r_y), observes the grid (including the cell you just marked black). It knows the locations of all currently black cells. It then moves to a new position (n_x, n_y). The new position (n_x, n_y) must be one of the 8 cells immediately adjacent (horizontally, vertically, or diagonally) to (r_x, r_y). That is, $|n_x - r_x| \le 1$ and $|n_y - r_y| \le 1$, and $(n_x, n_y) \ne (r_x, r_y)$. Additionally, the robot must stay within the first quadrant, so $n_x > 0$ and $n_y > 0$. The interactor (controlling the robot) will choose one valid move for the robot. The robot's strategy is unknown to you.

3. **Outcome Check:** After the robot moves to (n_x, n_y), the system checks if the cell (n_x, n_y) is black.

- If (n_x, n_y) is black, the robot explodes.
- If (n_x, n_y) is white, the game continues to the next time step, with the robot now at (n_x, n_y).

Your goal is to make the robot explode within T time steps.

Input

The first line of input contains two integers s_x and s_y ($1 \le s_x, s_y \le 20$), the initial coordinates of the robot.

Interaction Protocol

The interaction proceeds in turns for at most T turns. In each turn t (from $t = 1$ to T):

1. Your program must print one line containing two space-separated integers x_m and y_m, representing the coordinates of the cell you choose to mark black in this turn. Remember to flush the output stream.

2. The interactor reads your chosen coordinates (x_m, y_m).

3. The interactor determines the robot's next move to (n_x, n_y) based on its current position and the set of all black cells (including the one just marked at (x_m, y_m)).

4. The interactor checks if the cell (n_x, n_y) is black.

- If it is black (robot explodes), the interactor will print a single line containing '0 0' and terminate. Your program should then read these values and terminate successfully.
- If it is white, the interactor will print a single line containing two space-separated integers n_x and n_y, the new coordinates of the robot. Your program should read these coordinates to know the robot's current position for the next turn.

If the robot has not exploded after you have made T moves, your program should terminate. Your solution will be judged as incorrect in this case (or if you exceed the turn limit or make an invalid move).

To flush your output, you can use:

- `fflush(stdout)` (if you use `printf`) or `cout.flush()` (if you use `cout`) in C and C++.
- `System.out.flush()` in Java.
- `stdout.flush()` in Python.

Note: If your program does not terminate properly, it may be judged as a Presentation Error (PE).

Example

standard input	standard output
5 5	
	6 1
5 4	
	4 1
5 3	
	6 2
5 2	
	4 2
5 1	
	5 2
0 0	

Problem G. Sorting

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

You are given m pairs (a_i, b_i), $1 \le a_i < b_i \le n$, we define the following program to sort a permutation:

```
for i = 1 to 1000000000000000000:
    for j = 1 to m:
        if p[a[j]] > p[b[j]]:
            swap(p[a[j]], p[b[j]])
```

Check whether it can sort every permutation p of length n.

Input

The first line contains two integers n and m ($2 \le n \le 2 \times 10^5$, $1 \le m \le 2 \times 10^5$). The following are m lines, every line contains two integers (a_i, b_i) $(1 \le a_i < b_i \le n)$.

Output

Output `Yes` if it can sort every permutation of length n, and `No` if it cannot.

Examples

standard input	standard output

5 7
1 3
1 3
2 4
3 5
1 4
2 5
1 5
	No

5 8
2 3
3 5
1 5
3 4
1 3
4 5
2 5
1 2
	Yes

Problem H. Walk

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

On the 2-D plane, you are initially located at $(0, 0)$ and you want to go to (N, M). If your current position is (x, y), you can go to $(x + 1, y)$ or $(x, y + 1)$ in the next step. The path you walk is defined as the $(0, 0)$ to (N, M) polyline formed by connecting the positions of each step in turn.

There are K rectangles. The lower-left corner and upper-right corner of the i-th rectangle are $(x_{i,1} - 0.5, y_{i,1} - 0.5)$ and $(x_{i,2} + 0.5, y_{i,2} + 0.5)$, respectively. If the path you walk intersects both the left and right boundaries of the rectangle, you have to bear a cost of a_i, and if it intersects both the top and bottom boundaries of the rectangle, you have to bear a cost of b_i. The total cost is defined as the sum of the costs generated by each rectangle.

You need to find the minimum total cost required to travel from $(0, 0)$ to (N, M).

Input

The first line contains three integers N, M, and K ($1 \le N, M \le 50, 1 \le K \le 2000$).

The following K lines contain six integers each. The i-th line contains six integers $x_{i,1}, y_{i,1}, x_{i,2}, y_{i,2}, a_i, b_i$, describing the position and costs of the i-th rectangle. It is guaranteed that $0 \le x_{i,1} \le x_{i,2} \le N$, $0 \le y_{i,1} \le y_{i,2} \le M$, and $0 \le a_i, b_i \le 10^4$.

Output

Output a single integer denoting the minimum total cost required to travel from $(0, 0)$ to (N, M).

Examples

standard input	standard output

`1 2 1`
`0 1 1 1 1 2`
	`2`

`4 2 6`
`0 1 2 1 3 3`
`0 1 4 1 2 9`
`1 0 1 2 4 6`
`3 0 3 2 6 2`
`2 0 3 2 5 1`
`1 0 2 1 6 0`
	`27`

Problem I. Knapsack Problem

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	1024 megabytes

Given an undirected graph with n vertices and m edges, each edge holds an item with weight w_i. Xiao S has a compulsion: every time he traverses an edge, he must put the item on that edge into his backpack. The backpack has a capacity V. Initially, he starts with a new backpack. If the remaining capacity of the current backpack is insufficient to carry w_i, he switches to a new backpack (discarding the old one). For each starting vertex from 1 to n, Xiao S chooses a path to reach a designated vertex T. Your task is to determine the minimum number of backpacks needed for each starting vertex to reach T. If it is impossible to reach T, output -1.

Input

The first line contains four integers n, m, V, and T $(1 \leq n, m, V \leq 10^5, 1 \leq T \leq n)$.

The following m lines contain three integers each; the i-th line contains three integers x_i, y_i, and w_i, representing an edge between x_i and y_i with an item of weight w_i. It is guaranteed that $1 \leq x_i, y_i \leq n$ and $1 \leq w_i \leq V$.

Output

Output a single line with n integers; the i-th integer is the minimum number of backpacks needed when starting from vertex i. If T is unreachable, output -1.

Example

standard input	standard output

8 10 7 4
1 2 4
2 3 4
3 4 4
1 5 2
5 6 5
6 7 3
7 4 4
2 6 3
3 6 1
2 5 3
	2 2 1 1 2 1 1 -1

Problem J. Moving on the Plane

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

There are N points on a plane. The initial coordinates of the i-th point are (x_i, y_i).

Then, there will be M rounds of walks. In each round, every point must move from its current position (x, y) to one of the following four adjacent positions: $(x+1, y), (x-1, y), (x, y+1), (x, y-1)$.

Two plans are considered different if and only if there exists at least one point whose walk path is different in these two plans.

After exactly M rounds of walks, how many different plans are there such that the Manhattan distance between every pair of points does not exceed K?

Output the answer modulo 998244353.

Input

The first line contains three integers: N, M, and $K (1 \le N \le 50, 1 \le M \le 10^5, 0 \le K \le 10)$.

The $i+1$-th line contains the coordinates of the i-th point: x_i and y_i $(0 \le |x_i|, |y_i| \le 10^5)$.

Output

Only one line, the answer.

Examples

standard input	standard output

3 2 2
2 2
2 1
1 1
	672

3 3 1
3 3
0 2
1 2
	2340

2 3 0
3 0
2 1
	300

Problem K. Counting

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

Given three integers N, M, and K, consider all the unlabeled rooted trees with N vertices satisfying that exactly M of these vertices have K children. Let $A(N, M, K)$ be the number of such trees considering the order of each vertex's children.

Formally, two trees are considered different if the degrees of their roots are unequal or there exists a number t such that the subtrees of the t-th child of the two trees are different.

Now you are given an integer N and Q pairs $(M_1, K_1), \cdots, (M_Q, K_Q)$. Find the values of $A(N, M_1, K_1), \cdots, A(N, M_Q, K_Q)$.

Since the result may be large, output the answer modulo 998 244 353.

Input

The first line contains two integers N and Q ($1 \le N \le 10^5$, $1 \le Q \le 10^6$).

The following Q lines contain two integers each; the i-th line contains two integers M_i and K_i. It is guaranteed that $0 \le M_i, K_i \le N$.

Output

Output Q lines; the i-th line contains a single integer — $A(N, M_i, K_i)$ modulo 998 244 353.

Examples

standard input	standard output
1 1 1 0	1

5 10
4 1
0 0
1 2
4 0
1 3
1 4
3 0
3 1
0 5
0 1
	1
	0
	6
	1
	4
	1
	6
	0
	14
	3

Problem L. cover

Input file:	`standard input`
Output file:	`standard output`
Time limit:	8 seconds
Memory limit:	512 megabytes

You are given an array $a_{1,2,\cdots,n}$ of length n and m modification operations. The i-th operation is described by three parameters l_i, r_i, c_i, indicating that you may choose to set all elements in the subarray $a_{l_i}, a_{l_i+1}, \cdots, a_{r_i}$ to c_i. For each operation, you can choose to either apply it or skip it.

Your goal is to maximize the number of contiguous segments in the final array by optimally deciding whether to apply each operation (processed in the given order). The number of contiguous segments is defined as $1 + \sum_{i=2}^{n}[a_i \neq a_{i-1}]$, which counts the number of adjacent distinct elements plus one.

Additionally, all operations satisfy the following special property: for any two operations $i < j$, the interval $[l_j, r_j]$ is not a subinterval of $[l_i, r_i]$. In other words, there are no $i < j$ such that both $l_i \leq l_j$ and $r_j \leq r_i$ hold simultaneously.

Input

The input consists of multiple test cases. The first line contains an integer $T(1 \leq T \leq 10000)$, denoting the number of test cases.

Each test case consists of the following parts:

- The first line contains two positive integers $n(1 \leq n \leq 3 \times 10^5)$ and $m(1 \leq m \leq 3 \times 10^5)$, denoting the length of the array and the number of operations, respectively.
- The second line contains n positive integers $a_1, a_2, \cdots, a_n$ $(1 \leq a_i \leq n)$, representing the initial array.
- The next m lines each contain three integers l_k, r_k, c_k, describing the k-th operation $(1 \leq l_k \leq r_k \leq n$, $1 \leq c_k \leq n)$.

It is guaranteed that the sum of n across all test cases does not exceed 3×10^5, and the sum of m does not exceed 3×10^5.

Output

Output a single integer, representing the maximum number of contiguous segments achievable after optimally applying the operations.

Example

standard input	standard output

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

Problem M. Teleporter

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

A country has n cities, with $n-1$ roads connecting them. The i-th road connects city u_i and v_i, costing w_i units of time to go through the road. Every city is accessible from another city via these roads, forming a tree. Since technology is very developed in the country, there is a teleporter in each city. There are m teleport paths between these teleporters. The i-th path connects the teleporter in city p_i and q_i. If a person is in city u, he can travel to city v without costing any time if there is a teleport path directly connecting the teleporters in these two cities. As teleporting is very expensive, people can only use the teleporter no more than k times during their trip.

Small N is the minister of transportation of the country. He wants to assess the efficiency of this transportation network. Precisely speaking, he wants to calculate $\sum_{u=1}^{n} d(u,k)$ for each $k = 0, 1, \cdots, n$ where $d(u,k)$ is the minimum time cost to travel from city u to 1 using teleporters no more than k times. As the calculation is too complex for him, he wants you to design a program to calculate for him.

Input

The first line contains two integers n and m ($1 \le n \le 5000, 0 \le m \le 10000$), indicating the number of cities and teleport paths.

The following $n-1$ lines contain three integers each; the i-th line contains three integers u_i, v_i and w_i ($1 \le u_i, v_i \le n, 1 \le w_i \le 10^9$), indicating road i connects city u_i and v_i, costing w_i units of time to go through the road.

The following m lines contain two integers each; the i-th line contains two integers p_i and q_i ($1 \le p_i, q_i \le n$), indicating teleport path i connects the teleporter in city p_i and q_i.

Output

Your output should contain $n+1$ lines; the i-th line should contain one integer indicating the result when $k = i-1$.

Example

standard input	standard output

5 4
2 5 6
1 2 7
1 3 6
1 4 4
4 5
3 1
3 5
1 2
	30
	8
	4
	0
	0
	0