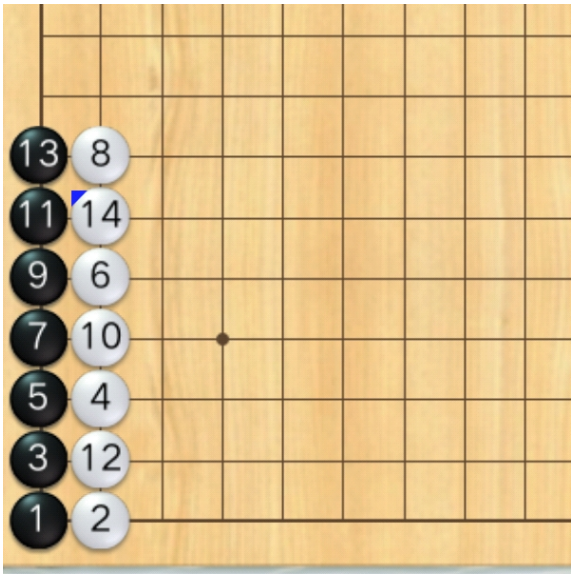
# Problem A

## New Gomoku

Time limit: 1 second




Traditional Gomoku, or *five in a row*, is played on a small board and ends immediately when five pieces of the same color are aligned. However, in "New Gomoku," the rules are different. The rules of New Gomoku are as follows:

- The board is larger, with a length and width of $1\,000$.
- Black and white players take turns placing their pieces, starting with the black player. A coordinate that has already been placed with a piece cannot be placed again.
- If exactly $5$ pieces of the same color are arranged to form an unbroken line of five pieces (horizontally, vertically, or diagonally at $45°/135°$ to the horizontal line), it is called a set of *five in a row*. The game does not end when one side gets a set of five in a row.
- The game ends when the predetermined number of moves is reached.

Your task is to output the total number of *five in a row* for the player who just placed their piece after each of the $n$ moves in the given sequence.

### Input

There are multiple test cases in a single test file. The first line of the input contains a single integer $T$ ($1 \le T \le 10^4$), indicating the number of test cases.

For each test case:

- The first line contains an integer $n$ ($1 \le n \le 10^4$), indicating the number of moves.
- The next $n$ lines each contain two integers $x_i, y_i$ ($1 \le x_i, y_i \le 1\,000$) representing the coordinates of the $i$-th move, ensuring that all coordinates in the same data set are unique.

It is guaranteed that the total sum of $n$ over all test cases does not exceed $10^4$, i.e., $\sum n \le 10^4$.

## Output

For each test case, output a single line with $n$ numbers, where the $i$-th number indicates the total number of *five in a row* for the player who just placed their piece after the $i$-th move

### Sample Input 1

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

### Sample Output 1

```
0 0 0 0 0 0 0 0 1 0 2 1 3 3
```

# Problem B
## Abstract Portal
### Time limit: 1 second

There are $n+1$ nodes, numbered from 0 to $n$. Initially, you are at node 0.

Each node has a **portal** and a **counter**, where the counter for node $p$ (with $0 \le p \le n$) is denoted as $c_p$ (initially all are 0). Each counter's range is between 0 and $k-1$, representing $k$ types of teleportation methods. These teleportation methods are controlled by the array $t_1, t_2, \ldots, t_{k-1}$, as described below.

When you are at any node $p$, you will perform the following operations:

1. Press the portal button once, updating the counter $c_p$ to $(c_p + 1) \mod k$.
2. If the updated $c_p = 0$, teleport to node $p+1$.
3. Otherwise, if $p - t_{c_p} \ge 0$, teleport to node $p - t_{c_p}$ (specifically, when $t_{c_p} = 0$, teleporting to yourself also counts as one teleport).
4. Otherwise, return to step one and repeat the above steps.

You stop immediately when you first reach node $n$ and do not continue any operations.

Calculate the number of times you used the portal from the start until you stop (i.e., the number of successful teleports), and return the result modulo $10^9 + 7$.

### Input

The first line of the input contains two integers $n$ ($1 \le n \le 10^9$) and $k$ ($2 \le k \le 1\,000$).

The second line contains $k-1$ integers, representing $t_1, t_2, \ldots, t_{k-1}$. For any $i \in [1, k-1]$, it holds that $0 \le t_i \le \min\{100, n-1\}$.

### Output

Output a single integer representing the number of times the portal was used, with the result taken modulo $10^9 + 7$.

**Sample Input 1**

```
2 2
1
```

**Sample Output 1**

```
4
```

**Sample Input 2**

```
6 3
0 0
```

**Sample Output 2**

```
18
```

**Sample Input 3**

```
114514 5
28 26 70 27
```

**Sample Output 3**

```
611437835
```

For the first example:

- The state of all node counters is $[0, 0, 0]$, and the position is at node 0. Pressing the portal button once changes the counter state to $[1, 0, 0]$. Since $t_1 = 1$ and $0 - 1 < 0$, no teleport occurs, so pressing the portal button again changes the counter state to $[0, 0, 0]$, and you teleport to node 1.

- The position is now at node $1$. Pressing the portal button once changes the counter state to $[0, 1, 0]$. Since $t_1 = 1$ and $1 - 1 = 0$, you teleport to node $0$.
- The position is now at node $0$. Pressing the portal button once changes the counter state to $[1, 1, 0]$. Since $t_1 = 1$ and $0 - 1 < 0$, no teleport occurs, so pressing the portal button again changes the counter state to $[0, 1, 0]$, and you teleport to node $1$.
- The position is now at node $1$. Pressing the portal button once changes the counter state to $[0, 0, 0]$, and you teleport to node $2$. At this point, you stop immediately, having made a total of $4$ teleports.

# Problem C
## GPA Optimization Plan
### Time limit: 2 seconds

Xiao Bo is a diligent college student who is very concerned about his academic performance. In each semester, he earns credits and corresponding grade points for several courses. Grade points are a way to measure course performance, while the average grade point (GPA) is an important indicator of a student's overall academic level. Xiao Bo hopes to achieve the highest possible GPA.

The school offers a special policy: each semester, students can choose to convert the grade of **at most** one course to a "Pass/No Pass" (PNP) mode (of course, they can also choose not to convert any course to PNP mode in certain semesters). In PNP mode, the grade points of this course will not be included in the GPA calculation.

Xiao Bo knows that he has taken a total of $n$ courses over $m$ semesters. He is aware of the credits, grade points, and semester for each course on his transcript, denoted as $s_i, g_i, t_i$. It is guaranteed that the first $2m$ courses on the transcript satisfy: the semester for the $i$-th course is $\lfloor \frac{i+1}{2} \rfloor$ (this ensures that at least two courses are taken each semester), while the semester for the remaining courses is not guaranteed. Xiao Bo hopes to make reasonable decisions each semester to maximize his final GPA.

However, Xiao Bo thinks that this is a bit too easy and wants to know: for any $k$ satisfying $2m \le k \le n$, if he only takes the courses from the 1st to the $k$-th on his transcript, considering the PNP decisions for each semester, what is the maximum GPA he can achieve?

The GPA calculation formula is as follows: assuming there are currently $n$ courses that have not chosen PNP mode, with the $i$-th course having credits and grade points of $s_i, g_i$, respectively, then

$$\text{GPA} = \frac{\sum_{i=1}^{n} s_i \cdot g_i}{\sum_{i=1}^{n} s_i}$$

## Input

There are multiple test cases in a single test file. The first line of the input contains a single integer $T$ ($1 \le T \le 10^4$), indicating the number of test cases.

For each test case:

- The first line contains two positive integers $n, m$ ($1 \le m \le 8,\ 2m \le n \le 10^5$), separated by a space, representing the total number of courses and the number of semesters.
- The next $n$ lines each contain three integers $s_i, g_i, t_i$ ($1 \le s_i, g_i \le 10^5, 1 \le t_i \le m$), separated by spaces, representing the credits, grade points, and the semester in which the course was taken. It is guaranteed that $\forall 1 \le i \le 2m, t_i = \lfloor \frac{i+1}{2} \rfloor$.

It is guaranteed that the sum of $n$ over all test cases does not exceed $3 \times 10^5$, i.e., $\sum n \le 3 \times 10^5$.

## Output

For each test case, output $n - 2m + 1$ lines, with each line corresponding to $k = 2m, k = 2m + 1, \ldots, k = n$, representing the maximum GPA Xiao Bo can achieve if he only takes the courses from the 1st to the $k$-th on his transcript.

**Answer Precision Requirement**: Your output will be compared with the standard answer. If the contestant's output value is $x$ and the standard answer is $y$, it will be judged as correct if $\frac{|x-y|}{\max(1,|y|)} \le 10^{-6}$

**Sample Input 1**

```
2
5 1
9 7 1
1 2 1
9 6 1
6 4 1
4 7 1
9 2
9 1 1
9 9 1
1 9 2
4 4 2
6 10 1
8 6 2
2 7 1
8 6 2
6 5 1
```

**Sample Output 1**

```
7.0000000000
6.5000000000
6.2631578947
6.3913043478
9.0000000000
9.3750000000
8.3000000000
8.1818181818
7.6470588235
7.2500000000
```

# Problem D
## Route Investigation
### Time limit: 6 seconds

The main road of Mazhen is oriented north-south, starting at the southernmost point and ending at the northernmost point. Along this road, there are $n-1$ intersections, dividing the entire road into $n$ segments. The $i$-th intersection from south to north connects the $i$-th segment of the road and the $(i+1)$-th segment of the road. Due to potential ambiguity in direction, the $i$-th intersection and segment are defined as the $i$-th intersection and segment from south to north.

Each intersection has a traffic light, and due to the directional nature of the streets connected to the intersection, the time spent passing through the same intersection with the same color light can differ when going north and south. The researcher believes that when randomly passing through the $i$-th intersection from south to north, the probabilities of encountering a red light, yellow light, and green light are $r_i, y_i, g_i$, respectively; when passing through the $i$-th intersection from north to south, the probabilities of encountering a red light, yellow light, and green light are $r'_i, y'_i, g'_i$, respectively. It is assumed that upon reaching the intersection, one will encounter one of the three situations: red light, yellow light, or green light, thus $r_i + y_i + g_i = r'_i + y'_i + g'_i = 1$.

To simplify the problem, the researcher assumes that regardless of the direction and time, when encountering a red light, yellow light, or green light, the waiting times are always 1 second, 2 seconds, and 0 seconds, respectively.

As a competitive programming participant, the researcher quickly calculated the expected time to traverse the entire road segment in both directions, under modulo $998\,244\,353$. However, now the researcher wants to know some more specific information. To this end, the researcher conducted a test: starting from the beginning to the end, he recorded the total waiting time for reaching the $i$-th segment, denoted as $a_i$; then, returning from the end to the beginning, he recorded the total waiting time for reaching the $i$-th segment, denoted as $b_i$. Clearly, $a_1 = b_n = 0$ and $a_i \le a_{i+1} \le a_i + 2$ and $b_{i+1} \le b_i \le b_{i+1} + 2$ $(1 \le i < n)$. Let $c_i = a_i + b_i$. The researcher recorded all $c_i$ $(1 \le i \le n)$ from this test.

The researcher wants to know the practical value of this test, so he hopes to find the probability for each $i = 1, 2, \cdots, n$ that the "total waiting time from the starting point to the $i$-th segment" + "total waiting time from the end point to the $i$-th segment" is exactly equal to $c_i$, under modulo $998\,244\,353$. This has him stumped, so he comes to you for help.

## Input

The first line of the input contains a single integer $n$ $(2 \le n \le 2 \times 10^5)$.

The next $n-1$ lines each contain 4 positive integers. The $i$-th line's 4 positive integers represent $100 \cdot r_i$, $100 \cdot y_i$, $100 \cdot r'_i$, $100 \cdot y'_i$. It is guaranteed that these $4(n-1)$ positive integers do not exceed 100, and that $r_i + y_i$ and $r'_i + y'_i$ are both $< 1$.

The next line of the input contains $n$ non-negative integers, where the $i$-th integer represents $c_i$ $(0 \le c_i \le 2n)$. It is guaranteed that there exist $a_1, a_2, \cdots, a_n$ and $b_1, b_2, \cdots, b_n$ satisfying $a_1 = b_n = 0$ and $a_i \le a_{i+1} \le a_i + 2$ and $b_{i+1} \le b_i \le b_{i+1} + 2$ $(1 \le i < n)$ and $a_i + b_i = c_i$.

## Output

Output a line with $n$ non-negative integers. The $i$-th integer represents the probability that the "total waiting time from the starting point to the $i$-th segment" + "total waiting time from the end point to the $i$-th segment" is exactly equal to $c_i$, under modulo $998\,244\,353$.

| Sample Input 1 | Sample Output 1 |
| --- | --- |
| 2<br>25 25 25 25<br>0 1 | 499122177 748683265 |

In the example, the probabilities of encountering red/yellow/green lights at the 1-st intersection in both directions are $\frac{1}{4}, \frac{1}{4}, \frac{1}{2}$.

At this point, the probability of reaching the 1-st segment from south to north with waiting time $a_1 = 0$ is 1 (since no intersections were crossed), and the probability of reaching the 1-st segment from north to south with waiting time $b_1 = 0$ is $\frac{1}{2}$ (i.e., waiting for a green light at the 1-st intersection), thus the probability for $c_1 = 0$ is $1 \times \frac{1}{2} = \frac{1}{2}$.

Similarly, the probabilities of reaching the 2-nd segment from south to north with waiting times $a_2 = 0, 1$ are $\frac{1}{2}, \frac{1}{4}$, and the probabilities of reaching the 2-nd segment from north to south with waiting times $b_2 = 0, 1$ are $1, 0$, thus the probability for $c_2 = 1$ is $1 \times \frac{1}{4} + 0 \times \frac{1}{2} = \frac{1}{4}$

# Problem E
## Dinner Invitation
### Time limit: 1 second

You are the leader of a school programming competition team in Chongqing for the CCPC. To enhance team bonding after the contest, you wish to organize a group dinner. However, since the budget was exhausted on hotpot before the competition, the cost of the meal must be **split equally**[*] among all participants.

There are $n$ classmates in the team. Each classmate who attends the dinner will order exactly one dish. The $i$-th classmate orders a dish costing $w_i$. Each classmate also has a budget constraint $r_i$: if the final shared cost of the meal exceeds $r_i$, the $i$-th classmate will stay at the hotel and order takeout instead.

As the leader, you want to maximize the number of classmates attending the dinner. Your goal is to select a subset of $m$ classmates such that if they each order their respective dishes, the shared cost (the sum of their $m$ dish prices divided by $m$) does not exceed the budget $r_i$ of any of the $m$ selected classmates. Find the maximum possible value of $m$.

If no such subset exists for $m \geq 1$, output $0$.

### Input

There are multiple test cases in a single test file. The first line of the input contains a single integer $T$ ($1 \leq T \leq 10^5$), indicating the number of test cases.

For each test case:

- The first line of the input contains a single integer $n$ ($1 \leq n \leq 10^5$), representing the number of classmates.
- Next, input a line with $n$ integers, where the $i$-th integer $w_i$ ($1 \leq w_i \leq 10^9$) represents the price of the dish ordered by the $i$-th classmate.
- Then, input a line with $n$ integers, where the $i$-th integer $r_i$ ($1 \leq r_i \leq 10^9$) represents the maximum price the $i$-th classmate is willing to pay.

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

### Output

For each test case, output a line with a single integer, representing the answer.

| Sample Input 1 | Sample Output 1 |
|---|---|
| 2<br>5<br>1 2 3 4 5<br>5 4 3 2 1<br>10<br>3 1 4 1 5 9 2 6 5 3<br>5 8 9 7 9 3 2 3 8 4 | 3<br>7 |

[*] That is, if $k$ people attend a meal with a total cost of $W$, each person pays $\frac{W}{k}$.

This page is intentionally left blank.

# Problem F
## Turtle Soup
### Time limit: 1 second

A company consists of $n$ members with employee IDs $1, 2, \ldots, n$. Member 1 is the boss, and every other member $i$ ($2 \le i \le n$) has a direct supervisor $f_i$ ($1 \le f_i < i$). The company structure forms a rooted tree with member 1 at the root.

In this company, member $a$ **knows** member $b$ if and only if at least one of the following conditions is satisfied:

- $a = b$.
- $b$ is an **ancestor** of $a$. In other words, there exists a sequence $u_1, u_2, \cdots, u_k$ ($2 \le k$) such that $u_1 = a$, $u_k = b$, and $u_i = f_{u_{i-1}}$ for all $2 \le i \le k$.
- $b$ is a **direct** child of $a$ ($b$ is a **direct** subordinate of $a$). In other words, $f_b = a$.

Note that the acquaintance relationship is not necessarily symmetric; $a$ knowing $b$ does not imply that $b$ knows $a$.

Now, members are playing a game. In each round, a specific member $m$ acts as the **guesser**, and another member $p$ is the **target**. It is guaranteed that $p$ is a member whom $m$ **does not know**.

To identify $p$, $m$ asks a sequence of questions. For each question, $m$ chooses an ID $x$ and asks: "Does $p$ know member $x$?" Member $p$ always answers truthfully. Additionally, if $x = p$, $p$ will specifically reveal that they are member $x$, at which point the game ends.

Each question carries a travel cost based on the company tree. If the $i$-th question concerns member $x_i$, the cost of that question is $dis(x_{i-1}, x_i)$, where $dis(a, b)$ is the number of edges on the unique path between $a$ and $b$ in the tree. For the first question ($i = 1$), the cost is $dis(m, x_1)$.

Member $m$ is extremely clever and uses an optimal strategy to minimize the total cost required to guarantee the completion of the game in the **worst-case scenario**. Your task is to calculate this minimum worst-case cost for every possible guesser $m \in \{1, 2, \ldots, n\}$.

If a member $m$ already knows everyone in the company, the answer for that $m$ is 0.

### Input

The first line of the input contains an integer $n$ ($1 \le n \le 10^5$), representing the total number of members in the company.

The second line contains $n - 1$ integers, representing $f_2, f_3, \cdots, f_n$. It satisfies $1 \le f_i < i$.

### Output

Output a single line containing $n$ integers, representing the answers for the cases where $m = 1, m = 2, \cdots, m = n$. The integers should be separated by spaces.

**Sample Input 1**

```
3
1 1
```

**Sample Output 1**

```
0 2 2
```

**Sample Input 2**

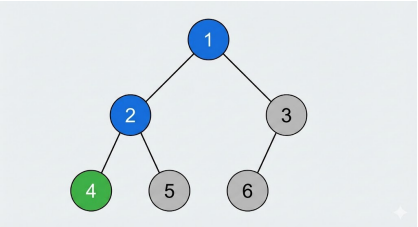
```
5
1 1 3 2
```

**Sample Output 2**

```
4 3 3 4 4
```

| Sample Input 3 | Sample Output 3 |
| --- | --- |
| 6<br>1 1 2 2 3 | 4 3 5 4 4 6 |

To explain the case of $m = 4$ in sample 3: First, as shown in the diagram, member $m$ does not know membes $\{3, 5, 6\}$. One possible optimal strategy is to first ask if member 2 is known by $p$, costing 1. If so, it indicates that $p = 5$, and the next step is to ask about member 5 to end the game, with a total cost of 2. Otherwise, continue to ask if member 3 is known by $p$, costing 2. If $p = 3$, the game ends at a total cost of 3. Otherwise, it indicates that $p = 6$, and the next step costs 1 to ask about member 6, ending the game with a total cost of 4. Therefore, this strategy requires a worst-case cost of 4. It can be proven that no other strategy can reduce the worst-case inquiry cost further.



# Problem G
## The Messenger
### Time limit: 1 second

Given $n$ integers $a_1, a_2, \ldots, a_n$. For any $k > n$, $a_k = \max\{a_i - a_j : k - n \le i, j < k\}$.

For a given integer $m > n$, output the value of $a_m$.

### Input

There are multiple test cases in a single test file. The first line of the input contains a single integer $T$ ($1 \le T \le 10^4$), indicating the number of test cases.

For each test case:

- The first line of the input contains two integers $n$ and $m$, ($2 \le n \le 500$, $1 \le m \le 10^{18}$).
- The second line of the input contains $n$ non-negative integers $a_1, a_2, \cdots, a_n$ ($0 \le a_i \le 10^{18}$ for all $1 \le i \le n$).

It is guaranteed that the sum of $n$ over all test cases does not exceed $500$, i.e., $\sum n \le 500$.

### Output

For each test case, output a line with a single integer, indicating the answer.

| Sample Input 1 | Sample Output 1 |
|---|---|
| 5<br>3 5<br>43 47 41<br>3 6<br>43 47 41<br>3 7<br>43 47 41<br>3 8<br>43 47 41<br>4 200<br>104857601998244353 1145141919810 3141592653589793 27182818284590452 | 41<br>35<br>35<br>6<br>377005672062684 |

This page is intentionally left blank.

# Problem H
## Hot Pot
### Time limit: 1 second

You are the captain of the school programming competition team, leading your classmates to Chongqing to participate in the CCPC. According to the team's tradition, as the captain, you need to treat your classmates to a meal, and you plan to eat the local specialty hot pot.

According to statistics, there are $x$ students in the team who only accept spicy food, $y$ students who only accept non-spicy food, and $z$ students who accept both spicy and non-spicy food. The hot pot restaurant you visit offers three types of pots:

- Spicy pot, which can serve at most **two** students who accept spicy food, costing $a$ yuan per serving.
- Mixed pot, which can serve at most **one** student who accepts spicy food and at most **one** student who accepts non-spicy food, costing $b$ yuan per serving.
- Non-spicy pot, which can serve at most **two** students who accept non-spicy food, costing $c$ yuan per serving.

You want to know the minimum amount of money needed to ensure that all students can eat hot pot according to their preferences.

### Input

There are multiple test cases in a single test file. The first line of the input contains a single integer $T$ ($1 \le T \le 2\,025$), indicating the number of test cases.

For each test case:

- Each line contains 6 positive integers $x, y, z, a, b, c$ ($1 \le x, y, z, a, b, c \le 2\,025$), representing the number of students in each category and the prices of the three types of hot pots.

### Output

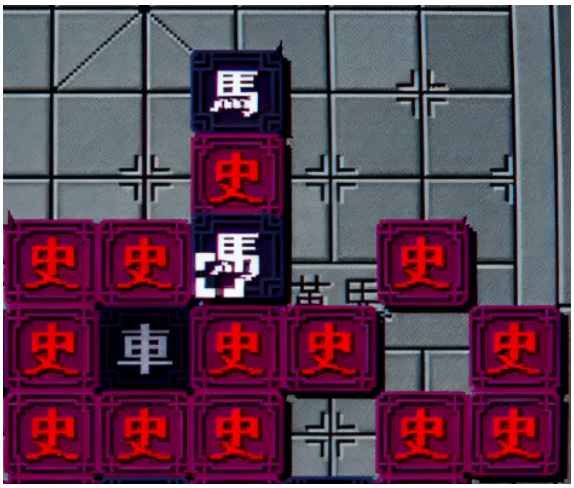
For each test case, output a single integer on a single line, representing the minimum possible cost to ensure that all students can eat hot pot according to their preferences.

| Sample Input 1 | Sample Output 1 |
|---|---|
| 6<br>3 1 4 1 5 9<br>1 1 4 5 1 4<br>1 5 2 8 2 3<br>9 6 2 3 5 10<br>1 5 8 2 8 7<br>1 9 5 10 3 5 | 8<br>3<br>9<br>39<br>29<br>26 |

- In the first example, you can let all mixed preference students eat spicy food, then order three spicy pots and one mixed pot, with a total cost of $1 \times 3 + 5 \times 1 = 8$.
- In the second example, you can let the mixed preference students have two eat spicy and two eat non-spicy, then order three mixed pots, with a total cost of $1 \times 3 = 3$.

This page is intentionally left blank.

# Problem I
## Shit
Time limit: 1 second



On a 10 row by 9 column chessboard, a game called "Qi Men Xiang Qi" is being played. Each side has several pieces, and different pieces have different characteristics. We use the $(x, y)$ coordinates to represent the position in the $x$-th row and the $y$-th column of the chessboard.

Our side has several Rooks and several Knights, as well as the skill Shape-shifting. The movement methods of these two types of pieces and the skill are described as follows:

- Suppose the Rook is located at coordinates $(a, b)$. If it moves to coordinates $(c, d)$ on the chessboard in one action, the following conditions must be met:
  - There are no friendly pieces at $(c, d)$. That is, it cannot capture friendly pieces, nor can it overlap with friendly pieces.
  - $c = a$ or $d = b$, but $(a, b) \neq (c, d)$. That is, it can move along the row or column it is in, but cannot remain in the same position.
  - There cannot be any other pieces between $(a, b)$ and $(c, d)$. That is, it cannot jump over other pieces while moving.
- Suppose the Knight is located at coordinates $(a, b)$. If it moves to coordinates $(c, d)$ on the chessboard in one action, the following conditions must be met:
  - There are no friendly pieces at $(c, d)$. That is, it cannot capture friendly pieces, nor can it overlap with friendly pieces.
  - $\{|a - c|, |b - d|\} = \{1, 2\}$. Here, $\{\dots\}$ denotes an unordered set. This corresponds to the traditional chess movement of the "L" shape (note that there are no "knight traps" in this problem).
- Shape-shifting: This can be used as one action to swap the positions of any two friendly pieces. There is no limit on the number of uses.

The enemy has several Slimes. Slimes cannot move but have the following characteristics:

- Suppose the Slime is located at coordinates $(a, b)$. When it is captured and removed, it will immediately split, generating one Slime in each of the four adjacent positions (up, down, left, right). Specifically, in these four adjacent positions, no new slimes will be generated in positions that already have any other pieces or are out of bounds of the chessboard.

- The Slimes generated by splitting have the same splitting characteristics and there is no limit on the number of splits.

After moving the pieces, if they overlap with enemy pieces, it triggers a capture, removing the enemy pieces. The victory condition of the game is to capture all enemy Slimes, ensuring that there are no enemy pieces left on the chessboard.

While playing the game, Xiao Biao found that although the enemy cannot move their pieces, it is still difficult for our side to cleanly capture the Slimes, as they are numerous and split very efficiently, often leading to more slimes being generated, making it hard for Xiao Biao to determine if victory is possible in this game. He has come to you, who are skilled in strategy and analysis, hoping you can help him determine if there exists a strategy to achieve victory within $10^4$ moves. If such a winning strategy exists, please output the operations for each step.

## Input

The input consists of 10 lines, each containing a string of length 9, representing the chessboard situation.

Where the character `.` represents an empty position; the character `C` represents our Rook; the character `M` represents our Knight; and the character `S` represents the enemy Slime.

Each type of character may appear multiple times. The data guarantees that there is at least one Slime on the chessboard and at least one friendly piece.

## Output

If there is no strategy to win within $10^4$ moves:

- Please output a single line string "WuJie".

If there exists a strategy to win within $10^4$ moves:

- First, output a single line string "YouJie".
- Next, output an integer $s$, representing the number of actions.
- Then, output $s$ lines, each containing four integers $a, b, c, d$, indicating that the friendly piece currently at $(a, b)$ is moved to position $(c, d)$. Specifically, if both positions $(a, b)$ and $(c, d)$ already have friendly pieces, this action is considered Shape-shifting, indicating that these two pieces swap positions.
- Your strategy must ensure that after all actions are completed, there are no Slimes remaining on the chessboard.

If there are multiple different valid solutions, any one of them can be output.

Solutions that do not meet the format requirements, do not comply with the piece movement rules, or indicate that there is no piece at position $(a, b)$ will be considered incorrect.

| Sample Input 1 | Sample Output 1 |
|---|---|
| ...CSC...<br>...CCCM..<br>.........<br>.........<br>.........<br>.........<br>.........<br>.........<br>.........<br>......... | YouJie<br>6<br>1 4 1 3<br>1 3 1 2<br>1 2 1 1<br>1 1 2 4<br>2 4 1 4<br>2 7 1 5 |

| Sample Input 2 | Sample Output 2 |
| --- | --- |
| SSSSSSSSS<br>SSSSSSSSS<br>SSSSSSSSS<br>SSSSSSSSS<br>SSSSSSSSS<br>SSSSSSSSS<br>SSSSSSSSS<br>SSSSSSSSS<br>SSSSSSSSS<br>SSSSCSSSS | WuJie |

The output for sample 1 demonstrates multiple valid operations, and ultimately captures all slimes.

- Steps 1 to 3 show the Rook at $(1, 4)$ moving left, choosing to move one square at a time, and finally landing at $(1, 1)$.
- Step 4 demonstrates the Shape-shifting operation, swapping the Rook at $(1, 1)$ with the Rook at $(2, 4)$.
- Step 5 shows the Rook at $(2, 4)$ moving up to $(1, 4)$.
- The final step shows the Knight jumping from $(2, 7)$ to $(1, 5)$ to capture the only Slime. Since there are no valid empty positions adjacent to the Slime after this, it cannot split. The game is won.

This page is intentionally left blank.

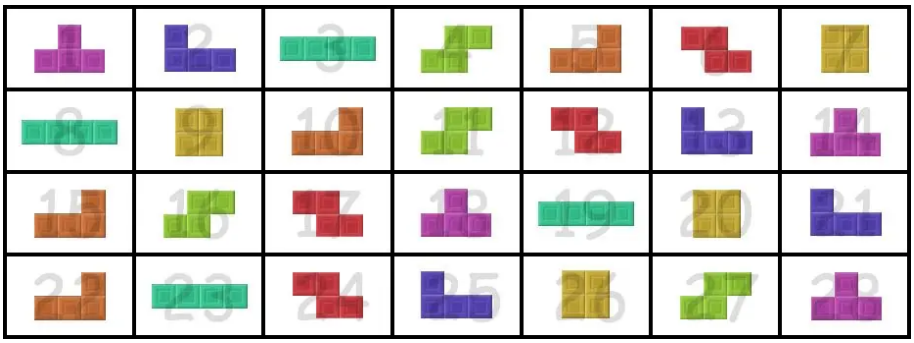
# Problem J

## Tetris

Time limit: 3 seconds

Little C is obsessed with Tetris. In a typical Tetris game, there are 7 types of tetrominoes, each represented by a letter according to its shape: I, T, O, J, L, S, and Z. Players receive a tetromino each time and need to place it in a certain position.

However, the tetrominoes that players receive are not completely random. Instead, every 7 distinct tetrominoes are grouped into a bag, and at the start of the game, players will receive each tetromino in the first bag in sequence, followed by each tetromino in the second bag, and so on. The order of receiving tetrominoes within each bag is completely random, and the order between bags is completely independent. This method of receiving tetrominoes is referred to as the 7-BAG.



**In this problem**, we are not limited to the 7 different tetrominoes, but rather consider a more general case where there are $n$ different tetrominoes, and the method of receiving tetrominoes is $n$-BAG (i.e., every $n$ distinct tetrominoes are grouped into a bag, with the order of receiving tetrominoes within each bag being completely independent). Each tetromino is numbered from 1 to $n$, and we have the following problem:

Little C has received $x$ tetrominoes from the start of the game until now. Starting from the $(x+1)$-th tetromino, Little C begins to remember the types of tetrominoes he received, up to the $(x+m)$-th tetromino. Therefore, we can represent the types of these tetrominoes as $f_1, f_2, \cdots, f_m$.

Since Little C is not a memory master, he may forget the type numbers of some (or even all) of the $m$ tetrominoes he remembers, and he uses 0 to represent the forgotten type numbers.

Unfortunately, Little C has also forgotten the exact value of $x$, but this value is very important for his future game planning. Therefore, Little C hopes you can infer which non-negative integers $x$ could possibly be based on the information he remembers.

Since there could be infinitely many possible values for $x$, Little C is only interested in the possible values of $x \bmod n$. To reduce the output, you only need to provide the **count** of all values of $x \bmod n$ that satisfy the known conditions, as well as their **binary XOR sum**.

Since Little C is not a memory master, he cannot guarantee that the types of the tetrominoes he has not forgotten are all correct. In this case, contradictions may arise, leading to the absence of any $x$ that satisfies the known conditions. In this case, simply output two 0s to indicate the answer.

Little C will also correct his remembered information $q$ times. Specifically, he will modify the type number of the $(x+a)$-th tetromino he remembers, changing it from $f_a$ to $b$ (in particular, if $b = 0$, it means Little C has

temporarily forgotten the type of the $(x+a)$-th tetromino). After each modification, you need to output the **count** of all values of $x \bmod n$ that satisfy the known conditions, as well as the **binary XOR sum**.

Note: Each modification is not independent; its effects will persist until the end.

## Input

The first line contains three integers $n, m, q$ $(1 \le n \le 10^{18}, 1 \le m \le 5 \times 10^5, 0 \le q \le 5 \times 10^5)$.

The second line contains $m$ integers, where the $i$-th integer $f_i$ $(0 \le f_i \le n)$ represents the type number of the $(x+i)$-th tetromino that Little C remembers. In particular, if $f_i = 0$, it means Little C has forgotten the type of the $(x+i)$-th tetromino.

The next $q$ lines each contain two integers $a, b$ $(1 \le a \le m, 0 \le b \le n)$, representing the modification of the type number of the $(x+a)$-th tetromino that Little C remembers to $b$. In particular, if $b = 0$, it means Little C has temporarily forgotten the type of the $(x+a)$-th tetromino.

## Output

Output a total of $q+1$ lines. The $i$-th line should contain two integers representing the **count** of all values of $x \bmod n$ that satisfy the known conditions after the first $i-1$ modifications, as well as the **binary XOR sum**.

### Sample Input 1

```
3 17 0
1 0 0 3 0 1 0 1 2 0 2 0 1 1 0 0 1
```

### Sample Output 1

```
1 2
```

### Sample Input 2

```
4 10 4
0 4 0 0 1 0 0 3 1 0
5 0
5 3
10 4
9 3
```

### Sample Output 2

```
4 0
4 0
3 0
3 0
0 0
```

### Sample Input 3

```
999999999999999999 10 10
0 0 0 0 314 15 0 0 9 0
1 0
9 287665588162745942
10 0
8 953846340212508779
10 275685051906270282
10 0
5 288735940850628909
9 14189307401575225
2 0
1 0
```

### Sample Output 3

```
999999999999999999 999999999999999999
999999999999999999 999999999999999999
999999999999999999 999999999999999999
999999999999999999 999999999999999999
999999999999999999 999999999999999999
999999999999999999 999999999999999999
999999999999999999 999999999999999999
999999999999999999 999999999999999999
999999999999999999 999999999999999999
999999999999999999 999999999999999999
999999999999999999 999999999999999999
```

**[Sample #1 Explanation]**

This memory sequence originally came from the Tetris sequence: [3,2,1],[3,2,**1**],[2,1,**3**],[2,**1**,3],[**1**,**2**,3],[**2**,3,**1**],[**1**,2,3],[**1**,2,3],[1,
(The bolded numbers are the information Little C remembers.)

In this case, $x = 5$. Of course, there are also valid cases for $x = 2, 8, 11, \ldots$, but when $x \bmod 3 \neq 2$, there are no valid cases. Therefore, the unique set of possible values for $x \bmod 3$ is $\{2\}$, so the count is 1, and the binary XOR sum is 2.

**[Sample #2 Explanation]**

Initially, the unique set of possible values for $x \bmod n$ is $\{0, 1, 2, 3\}$.

The first modification changes $f_5$ to 0, resulting in the memory sequence $[0, 4, 0, 0, 0, 0, 0, 3, 1, 0]$, and the possible values for $x \bmod n$ remain $\{0, 1, 2, 3\}$.

After the second modification, the unique set of possible values for $x \bmod n$ is $\{1, 2, 3\}$.

After the third modification, the unique set of possible values for $x \bmod n$ remains $\{1, 2, 3\}$.

**[Sample #3 Hint]**

Note: $n$ can be very large

This page is intentionally left blank.

# Problem K
## Reverse KMP
### Time limit: 1 second

In the KMP algorithm, the next array (also known as the LPS array, for Longest Prefix Suffix, or the failure function) is a core concept. Given a string $S = s_1 s_2 ... s_n$ of length $n$, its next array next[1..n] is defined such that next[i] represents the length of the longest equal proper prefix and proper suffix of the substring $S[1..i]$, specifically, next[1] = 0. Formally, it is defined as:

$$next[i] = \max_{k=0,\ldots,i-1} \{k : S[1 \ldots k] = S[i-k+1 \ldots i]\}$$

Here, $S[l..r]$ is defined as the substring formed by the $l$-th character to the $r$-th character of the string $S$. It is considered that $\forall 1 \le i \le n,\ S[1...0] = S[i+1...i]$, both of which are empty strings.

Given an array $A[1..n]$ of length $n$. You need to count how many different strings $S$ of length $n$ defined over the character set $\Sigma$ have a next array that is exactly equal to the given array $A$. The size of the character set is $m$. Output the number of such strings modulo 998 244 353.

## Input

There are multiple test cases in a single test file. The first line of the input contains a single integer $T$ ($1 \le T \le 10^5$), indicating the number of test cases.

For each test case:

- The first line contains two integers $n$ ($2 \le n \le 10^5$) and $m$ ($1 \le m \le 10^6$) representing the length of the given array and the size of the character set.
- The second line contains $n$ integers, separated by spaces, representing the given array $A$. It is guaranteed that $\forall 1 \le i \le n, 0 \le A_i \le i-1$.

It is guaranteed that the total sum of $n$ does not exceed $5 \times 10^5$, i.e., $\sum n \le 5 \times 10^5$.

## Output

For each test case, output a single line with a single integer, representing the number of strings that satisfy the conditions, with the answer taken modulo 998 244 353.

| Sample Input 1 | Sample Output 1 |
|---|---|
| 3<br>3 3<br>0 1 2<br>7 3<br>0 0 0 1 2 3 0<br>7 3<br>0 0 0 1 2 3 1 | 3<br>24<br>0 |

For the second group of data in the sample, let us assume the character set $\Sigma = \{'a','b','c'\}$. The 24 strings that meet the requirements are:

| | | | | | |
|---|---|---|---|---|---|
| abbabbb | abbabbc | abcabcb | abcabcc | acbacbb | acbacbc |
| accaccb | accaccc | baabaaa | baabaac | bacbaca | bacbacc |
| bcabcaa | bcabcac | bccbcca | bccbccc | caacaaa | caacaab |
| cabcaba | cabcabb | cbacbaa | cbacbab | cbbcbba | cbbcbbb |

This page is intentionally left blank.

# Problem L
## Tiling the Floor
### Time limit: 1 second

On an infinitely large floor, you need to lay down an infinite number of tiles.

Each tile has a pattern of square cells of size $n \times n$. Each cell of the tile has an LED light strip fixed along its bottom and left borders, with different colors that create a colorful pattern.

The pattern on each tile is exactly the same. When laying the tiles, we will place an infinite number of tiles seamlessly next to each other, forming an infinitely large light strip pattern.

There are $k$ different colors available, and you need to choose corresponding colors for a total of $2n^2$ light strips on one tile, after which the other tiles will automatically replicate this coloring scheme. When laying the tiles, all tiles will be laid in the same direction, meaning they can be viewed as directly translating one tile and then stitching them together without rotation.

As you walk on the floor, you want to know how many essentially different schemes there are for coloring the tiles. Since the floor is infinite, and a person may face different directions (north, south, east, west) when observing the tiles, two schemes are defined as essentially the same if:

When observing the infinite floor plane created by the two schemes, if there exists a series of translations or rotations (excluding reflections) that can make the colors of the light strips at all positions on the floor completely identical, then these two schemes are considered essentially the same.

### Input

There are multiple test cases in a single test file. The first line of the input contains a single integer $T$ ($1 \le T \le 10$), indicating the number of test cases.

For each test case:

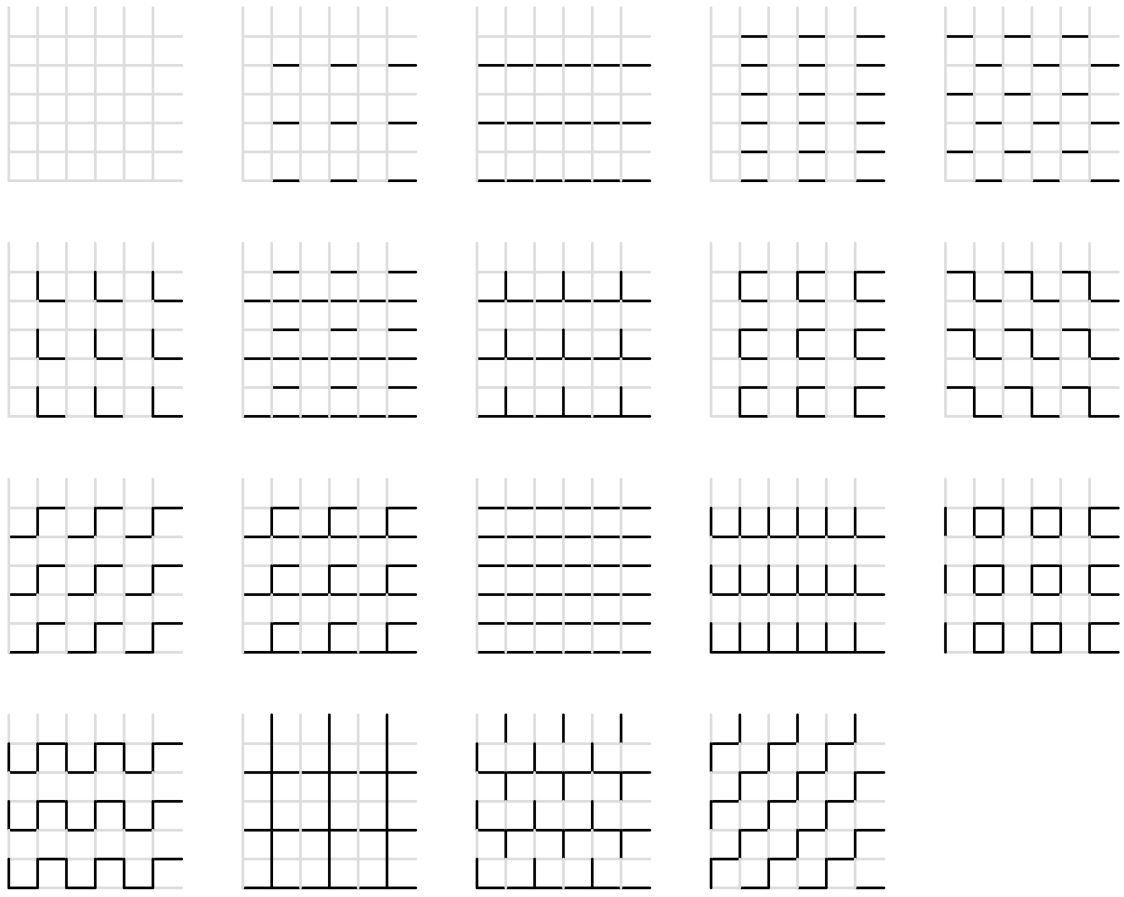
- The first line of the input contains two integers $n$ and $k$ ($1 \le n \le 10^9$, $1 \le k \le 10^9$), representing the size of the pattern and the number of colors, respectively.

### Output

For each test case, output a single line with a single integer, representing the number of essentially different schemes, modulo $10^9 + 7$.

| Sample Input 1 | Sample Output 1 |
|---|---|
| 3<br>1 2<br>2 2<br>114514 5201314 | 3<br>31<br>377726028 |

The following image shows 19 construction schemes for $n = k = 2$ (to better illustrate the tiling effect, each image shows 9 tiles arranged in a 3 by 3 pattern, with LEDs available in dark and light colors). By swapping the dark and light colors of the first 12 schemes, another 12 schemes can be obtained, resulting in a total of 31 schemes.



# Problem M
## Reduction and Growth
### Time limit: 2 seconds

Consider two positive integer variables $a$ and $b$. We define a "reduction" operation: $D(a, b)$ means changing the value of variable $a$ to $\frac{a}{\gcd(a,b)}$, while the value of $b$ remains unchanged. Here, $\gcd(a, b)$ denotes the greatest common divisor of $a$ and $b$. For example, if $a = 6$ and $b = 10$, then after calling $D(a, b)$ once, the value of $a$ becomes 3, while $b$ remains 10.

Consider a tree with positive integer vertex weights, where the vertex weight of node $i$ is $c_i$. We define a "chain reduction" operation on the tree: $T_{u,v}(a)$. Here, $u$ and $v$ are two nodes in the tree, and $a$ is a positive integer variable. Let's assume that the simple path from $u$ to $v$ in the tree visits the nodes $x_1, x_2, \ldots, x_k$ (where $1 \le k$, $x_1 = u$, and $x_k = v$). The operation $T_{u,v}(a)$ means performing the $D(a, c_{x_i})$ operation in order from $i = 1$ to $k$, that is, using the vertex weights along the path to reduce the value of variable $a$ sequentially.

Consider a tree $R$ that initially has only one vertex, labeled 1, with a vertex weight of $c_1$. To grow it into a tree with $n$ nodes, you need to perform $n-1$ growth operations. In the $i$-th operation, three integers $a_i, u_i, v_i$ are given, where $u_i$ and $v_i$ are guaranteed to be nodes currently in the tree, and $a_i$ is an integer variable. You need to first perform a chain reduction operation $T_{u_i,v_i}(a_i)$, and then add a new node to the tree: this new node is labeled $i+1$, its weight $c_{i+1}$ is the value of $a_i$ after the chain reduction, and it is connected to node $v_i$.

Finally, please output the vertex weights of each node as the answer.

### Input

The first line contains two positive integers $n$ and $c_1$ ($1 \le n \le 2 \times 10^5$, $1 \le c_1 \le 10^6$), representing the need to grow a tree with $n$ nodes, where the weight of the first node is currently $c_1$.

From the second to the $n$-th line, the $i$-th line contains three positive integers $a_i, u_i, v_i$ ($1 \le u_i, v_i < i$, $1 \le a_i \le 10^6$), indicating that the $i$-th node is grown, connected to node $v_i$, and its weight $c_i$ is the result of performing the $T_{u_i,v_i}(a_i)$ operation on $a_i$.

### Output

Output one line containing $n$ positive integers, separated by spaces, representing the weights $c_1, c_2, \ldots, c_n$

| Sample Input 1 | Sample Output 1 |
|---|---|
| 5 10<br>6 1 1<br>2 2 2<br>35 1 2<br>84 3 4 | 10 3 2 7 2 |

This page is intentionally left blank.