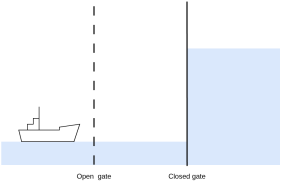
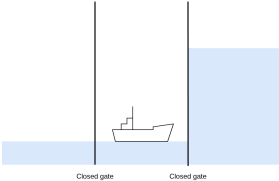
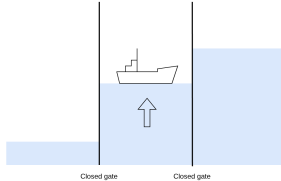
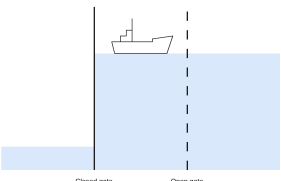
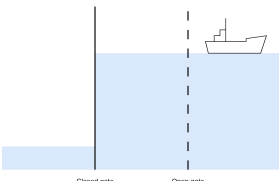


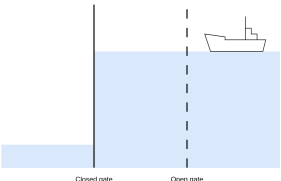
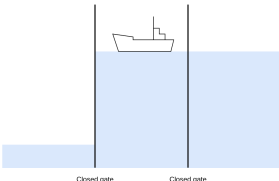
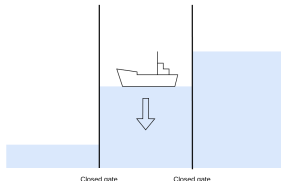
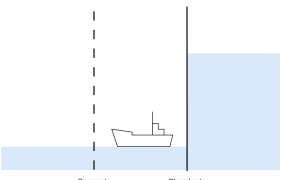
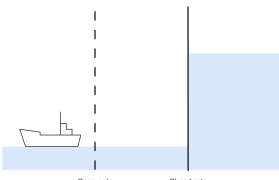
Problem A: Aquatic Elevator

An **aquatic elevator** is a gated chamber used on canals and rivers to move boats between sections of water at different heights. When a boat enters the chamber, the gates are closed, and the water level inside is raised or lowered to match the next section, allowing the boat to continue its journey smoothly. Thanks to the elevators, long waterways can climb hills and cross varied terrain without needing a perfectly flat route.

The following set of images demonstrates how an aquatic elevator chamber helps bring a boat uphill in a side view.

 <p>1. The boat is at the lower altitude. The left gate is open for the boat to enter.</p>	 <p>2. The boat enters the chamber. Both gates are now closed.</p>	 <p>3. Water is pumped into the chamber, raising the water level.</p>
 <p>4. The boat is now at a higher altitude. The right gate is open for the boat to exit.</p>	 <p>5. The boat exits the chamber.</p>	

The chamber can also help bring a boat downhill in a similar manner:

 <p>1. The boat is at the higher altitude. The right gate is open for the boat to enter.</p>	 <p>2. The boat enters the chamber. Both gates are now closed.</p>	 <p>3. Water is removed from the chamber, lowering the water level.</p>
 <p>4. The boat is now at a lower altitude. The left gate is open for the boat to exit.</p>	 <p>5. The boat exits the chamber.</p>	

You are in charge of the most complex aquatic elevator system in the world, consisting of n chambers. The i -th chamber is placed right at the height h_i ($0 = h_0 < h_1 < h_2 < \dots < h_n$), and can bring a boat from the height h_{i-1} to height h_i and vice versa. The i -th chamber also has a base area s_i . This means the volume of water required to maintain the i -th chamber is $s_i \cdot (h_i - h_{i-1})$. The volume of water required to maintain the whole system is defined as the **maximum** volume of water required by any chamber in the system.

To cut the operational cost, you are tasked to reduce the number of chambers down to k . To do so, you are allowed to do the following operation $n - k$ times:

- Suppose that the current number of chambers is n' . Choose two consecutive chambers i and $i + 1$ ($i \leq n' - 1$). They are replaced by a single chamber that
 - connects height h_{i-1} directly to height h_{i+1} (height h_i disappears from the system);
 - has base area $s_i + s_{i+1}$.

After the merge, the number of chambers decreases by 1, and the chambers are renumbered from 1 to $n' - 1$ in order of increasing height.

The operational cost is important, but saving water is important as well. Among all possible ways to reduce the number of chambers to k , determine the minimum possible volume of water required to maintain the whole system.

Input

Each test contains multiple test cases. The first line contains the number of test cases τ ($1 \leq \tau \leq 10^4$). The description of the test cases follows.

- The first line contains two integers n and k ($1 \leq k < n \leq 2 \cdot 10^5$) – the initial number of chambers in the system and the target number of chambers.
- The second line contains n integers h_1, h_2, \dots, h_n ($0 < h_1 < h_2 < \dots < h_n \leq 10^6$) – the heights at which the chambers are placed at.
- The third line contains n integers s_1, s_2, \dots, s_n ($1 \leq s_i \leq 10^6$) – the base area of the chambers.

Output

For each test case, print the minimum possible volume of water required to maintain the whole system among all possible ways to reduce the number of chambers to k .

Sample Input 1

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

Sample Output 1

```
48
4
```

Sample Explanation

In the first test case, there are $n = 3$ chambers that are placed at heights $[2, 3, 5]$ with the base area of $[7, 9, 8]$, respectively. There are 2 ways to reduce them into $k = 2$ chambers:

- The first way is to merge the first and the second chambers. After merging, the new chamber can connect height 0 to 3, with base area $7 + 9 = 16$. Its maintenance water volume is $16 \cdot (3 - 0) = 48$, and the other chamber's maintenance water volume is $8 \cdot (5 - 3) = 16$. Therefore the volume of water required to maintain the whole system is $\max\{48, 16\} = 48$.
- The second way is to merge the second and the third chambers. The first chamber's maintenance water volume is 14, and the new chamber's maintenance water volume is $(5 - 2) \cdot (9 + 8) = 51$. Therefore the volume of water required to maintain the whole system is 51.

The first way requires the lower maintenance water volume, so the answer is 48.

In the second test case, the optimal merging is as follows:

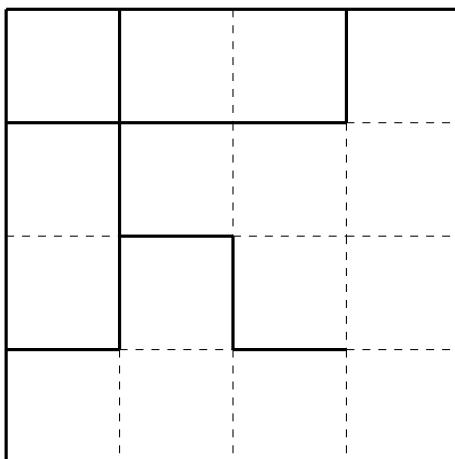
- Merge the two first chambers into a new one with the maintenance water volume of 4.
- Merge the two last chambers into a new one with the maintenance water volume of 4.

This page is intentionally left blank.

Problem B: Board Bicoloring

You are given a board with r rows and c columns, divided into $r \cdot c$ cells. The rows are numbered from 1 to r and the columns are numbered from 1 to c . We denote by (i, j) the cell at the i -th row and j -th column. Two cells are neighbors if and only if they share an edge.

We paint some edges between neighboring cells.



We define a path from a cell (i_1, j_1) to cell (i_k, j_k) as a sequence of cells $(i_1, j_1), (i_2, j_2), \dots, (i_k, j_k)$ where two consecutive cells are neighbors and their common edge is not painted.

We define a connected component as a set of cells where there is a path between any pair of cells in the set, and it is not possible to add any cell to this set without losing this property.

Now we want to color the board using two colors: Red and Green, so that two adjacent cells on two different connected components have different colors. We call such coloring **Bicoloring**. Note that a **Bicoloring** may not exist.

You need to process q queries of the following three types:

1. H i j if the common edge between cells (i, j) and $(i + 1, j)$ is not painted, we paint it. Otherwise, we erase the paint.
2. V i j if the common edge between cells (i, j) and $(i, j + 1)$ is not painted, we paint it. Otherwise, we erase the paint.
3. ? i j we consider three cases:
 - if there exists a pair of neighboring cells anywhere in the board that are in the same connected component, where their common edge is painted, print one line containing the string **Oh no!**,
 - otherwise, if there is no **Bicoloring** for the board, print one line containing the string **Cannot!**,
 - otherwise, consider all possible **Bicolorings** where the cell $(1, 1)$ is Red, find all possible colors of cell (i, j) . If this cell can be painted by either color, print **GR**. If this cell can only be painted red, print **R**, and if this cell can only be painted green, print **G**.

Input

The first line of the input contains two positive integers r and c ($1 \leq r \cdot c \leq 3 \cdot 10^5$). The next $r + 1$ lines describe the initial board according to the following rules:

- Each line contains $2 \cdot c + 1$ characters, numbered starting from 1. In each line:
 - All even-indexed characters can be either a space (' ') or an underscore ('_').
 - All odd-indexed characters can be either a space (' ') or a pipe ('|').
- All even-indexed characters in the first line and the last line are underscores, which demonstrate the top and the bottom borders of the board.
- Every underscore in the input, except the ones mentioned above, represents a horizontal painted edge between two consecutive cells in the same column.
- The first and the last characters of all lines (except the first one) are pipes, which demonstrate the left and the right borders of the board.
- Every pipe in the input, except the ones mentioned above, represents a vertical painted edge between two consecutive cells in the same row.

The next line contains a single integer q ($1 \leq q \leq 3 \cdot 10^5$) — the number of queries. The next q lines describe the queries, each line can be one of the three types, as described above:

- H i j ($1 \leq i < r$, $1 \leq j \leq c$),
- V i j ($1 \leq i \leq r$, $1 \leq j < c$),
- ? i j ($1 \leq i \leq r$, $1 \leq j \leq c$).

Output

For each query of type 3, print a single line as described above.

Sample Input 1

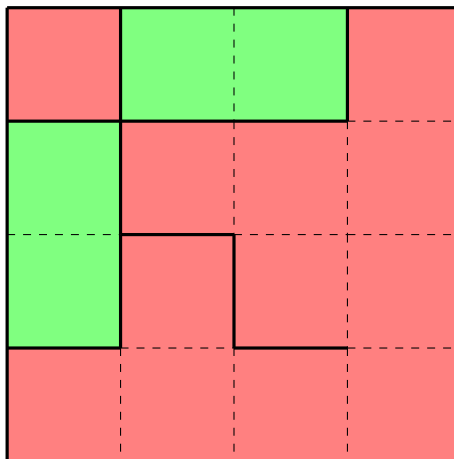
```
4 4
|_|_|_|
|_|_|_|
|_|_|_|
|_|_|_|
8
? 4 4
H 3 4
? 4 4
H 2 2
V 3 2
H 3 3
H 3 4
? 4 4
```

Sample Output 1

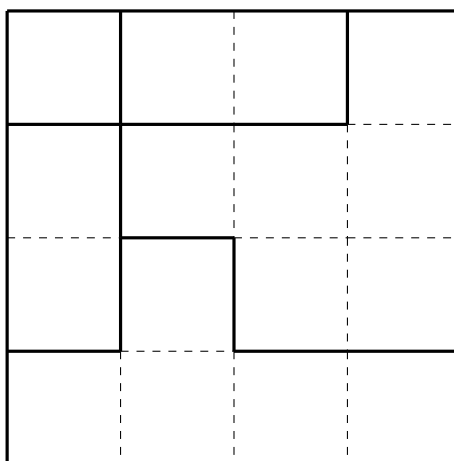
```
Oh no!
Cannot!
R
```

Sample Explanation

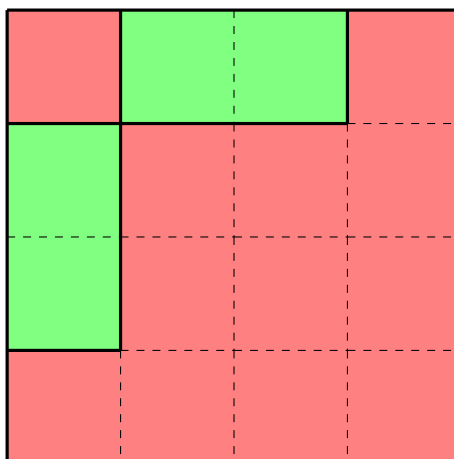
For the first query of type 3 (the 1-st query), the cells (3,3) and (4,3) belong to the same connected component, but their common edge is painted.



Before the second query of type 3 (the 3-rd query), the board is as follows. It can be shown that there exists no **Bicoloring**.



For the third query of type 3 (the 8-th query), the only possible **Bicoloring** is as follows:



This page is intentionally left blank.

Problem C: Cake Cutting

It is your birthday! You have prepared a rectangular cake and invited q friends over to celebrate. Your cake can be seen as an **axis-aligned** rectangle on the Oxy plane, with one corner located at point $(0, 0)$ and another at (w, h) .

Each of your friends, one by one, goes to you to give you a present and ask for some cake. The i -th friend points to a point (x_i, y_i) and instructs you to make two straight cuts from that point:

- one cut goes either **left** or **right**, parallel to the Ox axis;
- one cut goes either **upward** or **downward**, parallel to the Oy axis.

After you make the cuts, the friend gets all the cake inside the quadrant formed by the two cuts. Note that the point may **not** lie inside the cake. It is also possible that there is no cake inside the cut quadrant at all, as your friend may be full at the moment.

After each pair of cuts, your cake becomes smaller and smaller. You wonder if each of your friends has received a fair share of cake. For each friend, find the area of the cake that they receive.

Input

Each test contains multiple test cases. The first line contains the number of test cases τ ($1 \leq \tau \leq 10^4$). The description of the test cases follows.

- The first line contains three integers w , h , and q ($1 \leq w, h \leq 10^9$; $1 \leq q \leq 10^5$) — the cake's dimensions and the number of friends you invited.
- The i -th line of the next q lines contains two integers x_i , y_i and two characters hor and ver ($0 \leq x_i \leq w$; $0 \leq y_i \leq h$; $hor \in \{L, R\}$; $ver \in \{U, D\}$) — the coordinates of the i -th friend's point, and the cutting directions:
 - hor denotes the direction of the first cut: $hor = L$ for a **left** cut, $hor = R$ for a **right** cut;
 - ver denotes the direction of the second cut: $ver = U$ for an **upward** cut, $ver = D$ for a **downward** cut.

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

Output

For each friend, print the area of the cake that they receive.

Sample Input 1

```
3
8 6 2
4 3 R U
7 4 R D
8 6 2
4 3 R U
1 1 L D
8 6 2
4 3 R U
5 4 R U
```

Sample Output 1

```
12
3
12
1
12
0
```

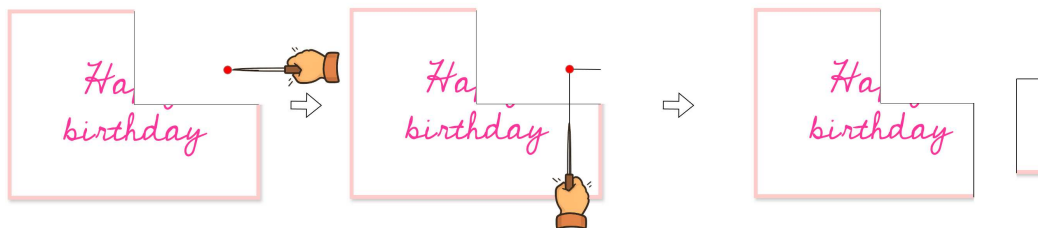
Sample Explanation

In the first test case, the cake has a width of 8 and a height of 6.

- The first friend gets a piece of cake with an area of $4 \cdot 3 = 12$ by cutting the piece from the cake center as follows:

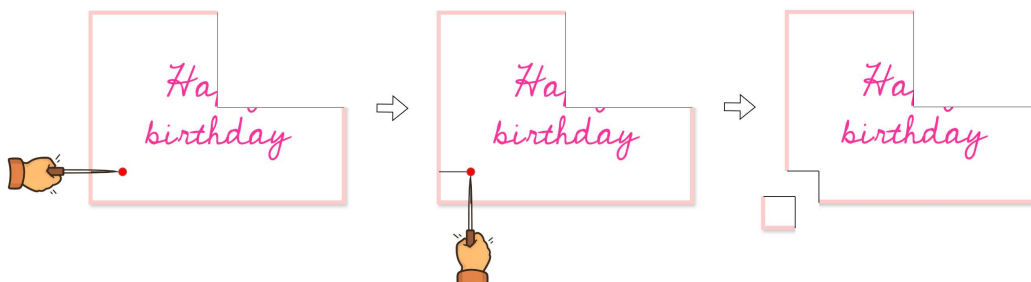


- The second friend gets a piece of cake with an area of 3 by cutting the cake as follows:



In the second test case, the cake has the same dimensions as the cake in the first test case.

- The first friend gets the same piece as in the first test case.
- The second friend gets a piece with an area of 1 by cutting the cake as follows:



Problem D: Divisibility Grid

You are given two integers r, c and an **odd** integer k . Consider a grid with r rows and c columns. You want to fill this grid with integers.

We call a filling of the grid *valid* if it satisfies both of the following:

- every cell contains an integer from 1 to $r \cdot c$;
- each integer from 1 to $r \cdot c$ appears in the grid exactly once.

A *divisibility* in the grid is a sequence of k consecutive cells in the same row or in the same column such that the sum of the numbers in these cells is divisible by k .

Among all valid fillings of the grid, find any one that maximizes the number of *divisibilities* in the grid.

Input

The first line contains an integer τ ($1 \leq \tau \leq 10^4$) – the number of test cases. τ test cases follow, each is presented by a single line with three integers r, c , and k ($1 \leq r, c \leq 50$; $1 \leq k \leq \min\{r, c\}$; k is odd).

It is guaranteed that the sum of $r \cdot c$ over all test cases does not exceed 10^4 .

Output

For each test case, print r lines, each containing c numbers, representing an optimal filling.

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

Sample Input 1

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

Sample Output 1

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

Sample Explanation

In the first test case, since $k = 1$, every valid filling of the grid should have the same number of *divisibilities*.

In the second test case, the maximum number of *divisibilities* is 6.

This page is intentionally left blank.

Problem E: Electronic Toll Collection

In a land far, far away, where administrative reforms to abolish district-level subdivisions have not been implemented, lies the vibrant city of *Nogias*, divided into four districts, each with its own unique beauty. District 1 serves as the “heart” of *Nogias*, where administrative buildings are built. District 2 is home to a myriad of gorgeous, affluent neighborhoods. District 3 hosts the majority of corporate buildings in the city. And district 4 has earned the reputation of a food paradise by both *Nogias*’ residents and foreign tourists. The districts are further divided into n wards numbered from 1 to n , such that each district has at least one ward.

The city’s road network consists of m bidirectional roads, each connecting two wards. There may exist multiple roads that connect the same pair of wards, and there may even exist roads that connect a ward to itself – after all, more roads is good, maintenance cost be damned. It is guaranteed that for every pair of wards (u, v) , there exists at least one path connecting them.

If there is one thing to be critical of *Nogias*, that would be its countless traffic jams. Every day, hundreds of thousands of motorbikes flood the roads, creating a truly nightmarish experience. As one of the best performing teams in the International City Planning Competition (ICPC), you have been hired by the city council to address the issue. After analyzing, you have found that most of the traffic comes from people who reside at one district and work in another. In particular, the following pairs of districts are very problematic in terms of traffic load:

- District 1 and district 2
- District 2 and district 3
- District 3 and district 4
- District 4 and district 1

Since relocating the people is out of the question and your suggestion of building public transport infrastructure got promptly ignored by the city, you decided on the only option left: Electronic Toll Collections (ETC)! The plan is to install ETC along the roads, such that for every pair of problematic districts (x, y) , **every** path that starts in a ward belonging to district x and ends in a ward belonging to district y (or vice versa) must go through **at least** one ETC road.

To the surprise of no one, your genius plan was immediately approved by the city, with a small caveat: people really *don’t* like paying money. Therefore, your plan should also minimize the number of roads with ETC installed.

Input

Each test contains multiple test cases. The first line contains the number of test cases τ ($1 \leq \tau \leq 10^5$). The description of the test cases follows.

- The first line contains two integers n and m ($4 \leq n \leq 10^5$; $n - 1 \leq m \leq 10^5$) — the number of wards and the number of roads.
- The second line contains n integers a_1, a_2, \dots, a_n ($1 \leq a_i \leq 4$), where a_i is the district that ward i belongs to.
- In the next m lines, each contains two integers u and v ($1 \leq u, v \leq n$), representing a road between two wards u and v .

It is guaranteed that:

- Each of the four districts has at least one ward and for every pair of wards (u, v) , there exists at least one path connecting them.
- The sum of n over all test cases does not exceed 10^5 .
- The sum of m over all test cases does not exceed 10^5 .

Output

For each test case, print the minimum number of roads with ETC installed.

Sample Input 1

```

1
16 16
1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4
1 4
2 4
3 4
4 8
5 8
6 8
7 8
8 12
9 12
10 12
11 12
12 16
13 16
14 16
15 16
16 4

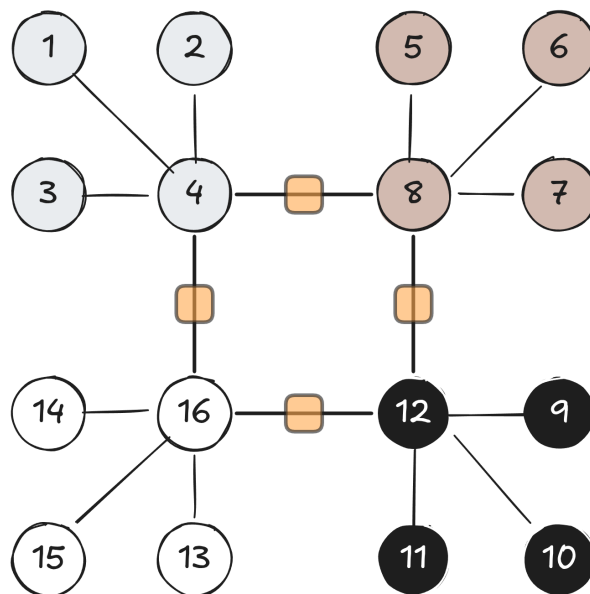
```

Sample Output 1

4

Sample Explanation

Below is one optimal plan for the first test case.



Here, wards with the same colors belong to the same district. The edges with small squares on it represent roads with ETC installed.

This page is intentionally left blank.

Problem F: Finest Final String

FWMC receives a string s of length n , consisting of only characters A and B, as a birthday gift. The identity of the person who keeps gifting FWMC strings as birthday gifts remains a mystery...

FWMC then starts playing around with the string. In each step, FWMC can perform the following compression operation on the string any number of times:

- Choose a position i such that $1 \leq i < n$ and $s_i = s_{i+1}$. Then delete two characters s_i and s_{i+1} , and insert one character of the opposite type at the deleted position (B if $s_i = A$, and vice versa).

FWMC thinks that the smaller the lexicographical order¹ of a string, the **finer** it is. Your task is to help them find the **finest** string s' that can be obtained from string s , and one possible sequence of compression operations to obtain this string.

Input

Each test contains multiple test cases. The first line contains the number of test cases τ ($1 \leq \tau \leq 5000$). The description of the test cases follows.

- The first line contains an integer n ($1 \leq n \leq 5000$) — the length of the string s .
- The second line contains the string s . This string only contains characters A and B.

It is guaranteed that the sum of n over all test cases does not exceed 5000.

Output

For each test case, print the following:

- The first line contains a string s' — the **finest** string that can be obtained from the string s .
- The second line contains an integer k ($0 \leq k < n$) — the number of compression operations.
- The third line contains k integers p_1, p_2, \dots, p_k representing k operations. The value p_i means that the i -th operation is to remove two characters at positions p_i and $p_i + 1$, then insert a new character at position p_i . The value p_i must be less than the length of s before this operation.

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

¹A string a is lexicographically smaller than string b if and only if one of the following holds:

- a is a prefix of b , but $a \neq b$; or
- in the first position where a and b differ, the string a has a letter that appears earlier in the alphabet than the corresponding letter in b .

Sample Input 1

Sample Output 1

3	B
1	0
B	
3	A
BAA	2
5	2 1
ABBBA	AA
	3
	3 3 2

Sample Explanation

In the first test case, it is impossible to perform any compression operation on the given string. Therefore, the **finest** string obtainable is 'B'.

In the second test case, one possible way to obtain the string 'A' (which can be shown to be the **finest** string obtainable) is as follows:

$$BAA \rightarrow BB$$

$$BB \rightarrow A$$

In the third test case, one possible way to obtain the string 'AA' (which can be shown to be the **finest** obtainable string) is as follows:

$$ABBBA \rightarrow ABAA$$

$$ABAA \rightarrow ABB$$

$$ABB \rightarrow AA$$

Problem G: Group Raiding

World of Warcraft (UwU) is one of the largest MMORPG in the world. One prominent activity in UwU, as with every other MMORPG, is *group raiding*, where players form parties to take on one or several powerful bosses. Today, RR and Ming were supposed to lead a party of 69 players in a raid against one of the most powerful bosses in the game, but since the other 67 did not show up, the raid was called off. To not waste their preparation, RR and Ming decided to raid Harmony — one of the easier bosses instead.

Since Harmony is way beneath RR and Ming's level, it inflicts negligible damage to both of them. On the other hand, it is quite formidable defensively. At the start of the raid, Harmony has n HP and a *damage limiter value* x . Any skill, no matter how powerful, can only deal at most x damage to Harmony. After Harmony takes a hit losing y HP ($1 \leq y \leq x$), it will update its *damage limiter value* to $L(y) = \sum_{k=1}^y \lfloor \frac{y}{k} \rfloor$, where $\lfloor m \rfloor$ is the largest integer not exceeding m .

Note that all HP and damage values are **positive integers**.

Now, this does not trouble RR and Ming at all. Both are strong enough to deal any amount of damage to Harmony (up to its *damage limiter value*) as they please. However, the one who deals the killing blow (i.e. one that reduces Harmony's HP to 0) will get most of the loot. Both have agreed to take turns hitting Harmony, with Ming getting to choose who goes first. Ming wonders whether he should go first or second to guarantee having the killing blow, given perfect play by both players. Please help him!

Input

The first line contains an integer τ ($1 \leq \tau \leq 10^5$) – the number of test cases. τ test cases follow, each is presented by a single line with two integers n and x ($1 \leq x \leq n \leq 10^{14}$) — Harmony’s initial HP and *damage limiter value*.

Output

For each test case, if the first player can guarantee having the killing blow, output First. Otherwise, output Second.

Sample Input 1

```
4
2 1
8 3
10 3
1000000000000000 1000000000000000
```

Sample Output 1

```
Second
Second
First
First
```

Sample Explanation

In the first test case, the first player must deal 1 damage to Harmony, yielding the killing blow to the second player.

In the second test case, the second player can secure the killing blow as follows:

- If the first player deals 1 damage on the first turn, since $L(1) = 1$, both players are forced to deal 1 damage until Harmony is defeated, with the killing blow going to the second player.
- If the first player deals 2 damage on the first turn, since $L(2) = 3$, the second player can deal 2 damage. Then:
 - If the first player deals 1 damage on the next turn, both players are forced to deal 1 damage until Harmony is defeated, with the killing blow going to the second player.
 - If the first player deals 2 or 3 damage on the next turn, the second player can deal 2 or 1 damage, respectively, killing Harmony.
- If the first player deals 3 damage on the first turn, since $L(3) = 5$, the second player can deal 5 damage, killing Harmony.

In the third test case, the first player can secure the killing blow by dealing 2 damage on the first turn, then using the same strategy as the second player in the second test case.

In the fourth test case, the first player can deal 10^{14} damage, killing Harmony immediately.

Problem H: Hamiltonian Path Remix

The *Mafia* team is reviewing the problem proposals for a certain ICPC contest. One of the proposed problem statement goes as follows:

Hamiltonian Path

You are given a complete weighted undirected graph consisting of n vertices, numbered from 1 to n . The distance between vertex i and vertex j is denoted by $w_{i,j}$.

A *Hamiltonian path* in this graph is a path that visits every vertex exactly once. The **length** of such a path is the sum of weights of all edges used along the path.

Your task is to find the shortest Hamiltonian path (i.e. one with minimum possible total length).

Formally, find a permutation p_1, p_2, \dots, p_n of $\{1, 2, \dots, n\}$ that minimizes $\sum_{i=1}^{n-1} w_{p_i, p_{i+1}}$.

”But wait, isn’t this problem too classical?”—thinks Quang, a member of *The Mafia* team.

After playing around with the idea of Hamiltonian path, he came up with the following variant of the problem. In this variant, the locations of the vertices are the vertices of a 2D convex polygon listed in counter-clockwise order, and the distance between vertex i and vertex j equals the Euclidean distance between the two vertices.

This variant turns out to be interesting enough (and hopefully, has not appeared in any other programming contest) and is accepted by the team, which goes on to be the problem H of that contest. As talented participants, can you solve it?

Input

Each test contains multiple test cases. The first line contains the number of test cases τ ($1 \leq \tau \leq 1000$). The description of the test cases follows.

- The first line contains an integer n ($3 \leq n \leq 3000$) — the number of vertices.
- The i -th of the next n lines contains two integers x_i and y_i ($0 \leq x_i, y_i \leq 10^6$) — the coordinates of the i -th vertex.

It is guaranteed that

- The vertex coordinates correspond to the vertices of a convex polygon listed in counter-clockwise order.
- No three consecutive vertices lie on the same line.
- The sum of n over all test cases does not exceed 3000.

Output

For each test case, print n integers p_1, p_2, \dots, p_n — the indices of vertices on the shortest Hamiltonian path. The sequence p must be a permutation of $\{1, 2, \dots, n\}$.

Your answer is considered correct if the absolute or relative error between the total length of your path and jury’s path does not exceed 10^{-6} .

Formally, let the total length of your path be P , and the total length of jury’s path be J . Your answer is accepted if and only if $\frac{|P-J|}{\max(1, |J|)} \leq 10^{-6}$.

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

Sample Input 1

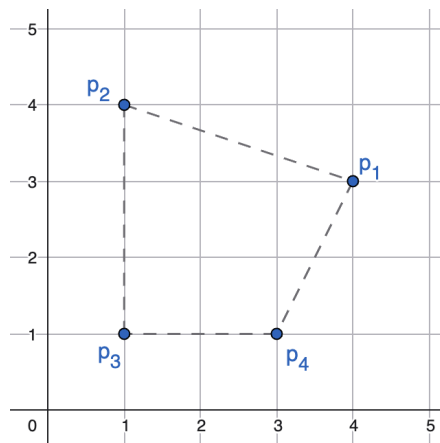
```
2
4
4 3
1 4
1 1
3 1
8
2 4
1 3
1 2
2 1
3 1
4 2
4 3
3 4
```

Sample Output 1

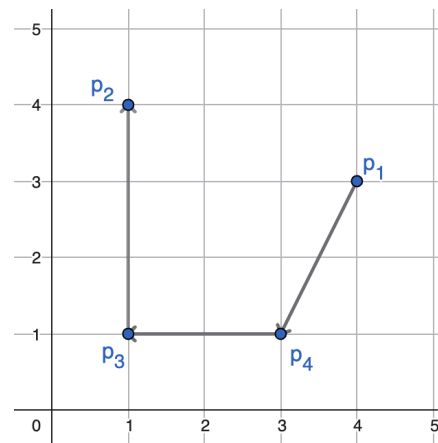
```
1 4 3 2
8 1 2 3 4 5 6 7
```

Sample Explanation

Illustration of the first test case:

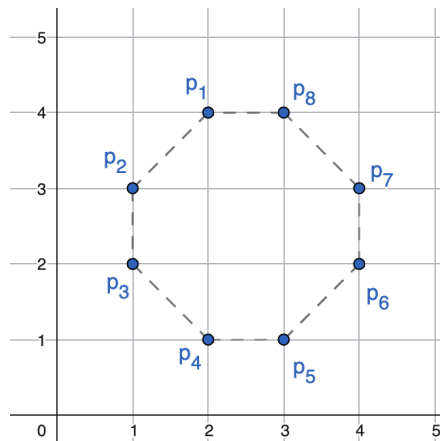


Location of vertices

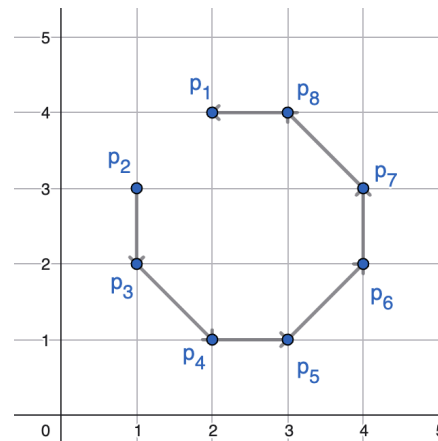


A possible shortest Hamiltonian path
(total length: $\sqrt{5} + 2 + 3 \approx 7.236$)

Illustration of the second test case:



Location of vertices



A possible shortest Hamiltonian path
(total length $4 \times 1 + 3 \times \sqrt{2} \approx 8.243$)

Problem I: Impossible Inversion Counting

Mr. Onion loves numbers. Today, during his algorithm lectures, he writes on the board a sequence of integers, denoted as $c_1^{(0)}, c_2^{(0)}, \dots, c_n^{(0)}$. Since Mr. Onion hates repetition, his sequence has a special property: **No value appears in the sequence more than ten times.**

As today's lectures is about lexicographical order, Mr. Onion sequentially asks k students to go to the board. Each student is asked to create a new sequence of integers based on the one created by the previous student (the first student has to use Mr. Onion's sequence). More precisely, the i -th student ($1 \leq i \leq k$) is asked to create a sequence of n integers, denoted as $c_1^{(i)}, c_2^{(i)}, \dots, c_n^{(i)}$, satisfying the following conditions:

- $c_1^{(i)}, c_2^{(i)}, \dots, c_n^{(i)}$ must be a permutation² of $c_1^{(i-1)}, c_2^{(i-1)}, \dots, c_n^{(i-1)}$.
- $c_1^{(i)}, c_2^{(i)}, \dots, c_n^{(i)}$ must be lexicographically larger³ than $c_1^{(i-1)}, c_2^{(i-1)}, \dots, c_n^{(i-1)}$.
- $c_1^{(i)}, c_2^{(i)}, \dots, c_n^{(i)}$ must be the lexicographically smallest sequence among all those that satisfy the two above conditions.

All k students go to the board and manage to find all required sequences in a matter of seconds, which makes Mr. Onion angry. As you may know, in Vietnamese, *onion* and *bully* are exactly the same word. Therefore, Mr. Onion tries to bully one more student. He asks him to calculate the total number of inversions⁴ of all $k + 1$ sequences written on the board.

As the board is full of numbers now, the unlucky student quickly feels dizzy. Could you help him find the result?

Input

Each test contains multiple test cases. The first line contains the number of test cases τ ($1 \leq \tau \leq 2 \cdot 10^5$). The description of the test cases follows.

- The first line contains two integers n and k ($1 \leq n \leq 2 \cdot 10^5$; $0 \leq k \leq 2 \cdot 10^7$).
- The second line contains n integers $c_1^{(0)}, c_2^{(0)}, \dots, c_n^{(0)}$ ($1 \leq c_i^{(0)} \leq n$).

It is guaranteed that:

- No value appears in the sequence $c_1^{(0)}, c_2^{(0)}, \dots, c_n^{(0)}$ more than ten times.
- k is less than or equal to the number of permutations of $c_1^{(0)}, c_2^{(0)}, \dots, c_n^{(0)}$ which is lexicographically larger than $c_1^{(0)}, c_2^{(0)}, \dots, c_n^{(0)}$.
- The sum of n over all test cases does not exceed $2 \cdot 10^5$.

²A sequence $\beta_1, \beta_2, \dots, \beta_\eta$ is a permutation of the sequence $\alpha_1, \alpha_2, \dots, \alpha_\eta$ if and only if there exists η distinct integers $\rho_1, \rho_2, \dots, \rho_\eta$ so that $1 \leq \rho_\iota \leq \eta$ and $\beta_{\rho_\iota} = \alpha_\iota$ for all $1 \leq \iota \leq \eta$.

³A sequence α is lexicographically smaller than a sequence β if and only if one of the following holds:

- α is a prefix of β , but $\alpha \neq \beta$; or
- in the first position where α and β differ, the sequence α has a smaller element than the corresponding element in β .

⁴An inversion of a sequence $\alpha_1, \alpha_2, \dots, \alpha_\eta$ is a pair of integers (ι, κ) so that $1 \leq \iota < \kappa \leq \eta$ and $\alpha_\iota > \alpha_\kappa$.

- The sum of k over all test cases does not exceed $2 \cdot 10^7$.

Output

For each test case, print an integer on a single line denoting the total number of inversions of all $k + 1$ created sequences of integers.

Sample Input 1

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

Sample Output 1

```
31
12
```

Sample Explanation

In the first test case:

- $c_1^{(0)}, c_2^{(0)}, \dots, c_7^{(0)} = (5, 6, 2, 1, 3, 7, 4)$. This sequence has 10 inversions.
- $c_1^{(1)}, c_2^{(1)}, \dots, c_7^{(1)} = (5, 6, 2, 1, 4, 3, 7)$. This sequence has 10 inversions.
- $c_1^{(2)}, c_2^{(2)}, \dots, c_7^{(2)} = (5, 6, 2, 1, 4, 7, 3)$. This sequence has 11 inversions.
- The total number of inversions is $10 + 10 + 11 = 31$.

In the second test case:

- $c_1^{(0)}, c_2^{(0)}, \dots, c_6^{(0)} = (2, 2, 1, 3, 3, 1)$. This sequence has 6 inversions.
- $c_1^{(1)}, c_2^{(1)}, \dots, c_6^{(1)} = (2, 2, 3, 1, 1, 3)$. This sequence has 6 inversions.
- The total number of inversions is $6 + 6 = 12$.

Problem J: Joint Farm

An agricultural cooperative consists of m households, numbered from 1 to m , that is planning their activities for the next year to maximize their profit. This has become an annual challenge for the cooperative, as there are multiple elements that can affect the production output, such as climate, types of crops, transportation, and storage.

During the farming year, there are three farming seasons: spring, summer-fall, and winter, that are numbered from 1 to 3 in their respective order. In each season, the cooperative can assign each of the households **at most** one type of crop to grow. Based on past records, there are k viable crop types, numbered from 1 to k . During the s -th season, the i -th household can be assigned to grow the j -th crop type to produce an **agricultural product unit** that yields the profit $c_{s,i,j}$. An unassigned household does not farm during the season.

The local climate also comes with a set of constraints on transportation and storage for the cooperative. During the summer-fall season, people go directly to the farm to buy the products, so there is no cost for transportation and long-term storage. However, in the spring and winter seasons, all agricultural products are gathered and sent to the warehouse at the end of the season.

- Before sending to the warehouse, the agricultural product units are grouped into their corresponding crop types. At the end of the s -th season ($s \neq 2$), the cost to send the group of the j -th crop type is $x_{s,j}^2 \cdot w_{s,j}$, where $x_{s,j}$ is the number of the gathered units of the j -th crop type, and $w_{s,j}$ is the weight of one such unit farmed during the s -th season.
- Because of the warehouse's capacity, there is a limit on $\sum_{j=1}^k x_{s,j}$ — the total number of produced agricultural product units over all k types:
 - In the spring season, $\sum_{j=1}^k x_{1,j} \leq a$.
 - In the winter season, $\sum_{j=1}^k x_{3,j} \leq b$.

The final constraint for the cooperative is the farmers themselves. A household can be assigned in multiple seasons. But each of the households disagrees to farm in two consecutive seasons. Because of the obvious advantage of the summer-fall season, the farmers say they would only farm in the summer-fall season and be happy with the yield.

As the number of households grows, the planning becomes more complicated for the cooperative. Therefore, the cooperative asks for your help with the calculation, so the cooperative can have a better picture before the execution. For every p from 1 to $2 \cdot m$, help the cooperative find the maximum profit and an optimal assignment if the total number of produced agricultural product units across three seasons is p (formally, $p = \sum_{s=1}^3 \sum_{j=1}^k x_{s,j}$).

Input

The first line contains four integers m, k, a, b ($1 \leq m, k, a, b \leq 200$) — the number of households in the cooperative, the number of viable crop types, the warehouse's capacity during spring, and the warehouse's capacity during winter, respectively.

Then follows 3 groups of m lines. The s -th group of lines describes the profit during the s -th season:

- The i -th line contains k integers $c_{s,i,1}, c_{s,i,2}, \dots, c_{s,i,k}$ ($1 \leq c_{s,i,j} \leq 10^9$) — the profit if the i -th household farms the j -th crop during the s -th season.

The next line contains k integers $w_{1,1}, w_{1,2}, \dots, w_{1,k}$ ($1 \leq w_{1,j} \leq 10^9$) — the weight of one agricultural product unit of the j -th crop type farmed during the spring season.

The last line contains k integers $w_{3,1}, w_{3,2}, \dots, w_{3,k}$ ($1 \leq w_{3,j} \leq 10^9$) — the weight of one agricultural product unit of the j -th crop type farmed during the winter season.

Output

Print $2 \cdot m$ groups of lines; the p -th of them describes the optimal assignment if the total produced agricultural product units in the year is p :

- If there is no way to produce exactly p agricultural product units and still follow all requirements, print **impossible**.
- Otherwise:
 - On the first line, print an integer b_p ($|b_p| \leq 10^{18}$) — the maximum profit.
 - On the i -th line of the next m lines, output three integers $t_{i,1}, t_{i,2}$, and $t_{i,3}$ ($0 \leq t_{i,s} \leq k$) — the assigned crop type of the i -th household during the spring season, the summer-fall season, and the winter season, respectively; $t_{i,s} = 0$ means the household is not assigned any crop type during the s -th season.

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

Sample Input 1

```
2 2 2 1
5 10
10 15
2 1
3 2
6 9
24 17
18 22
23 17
```

Sample Output 1

```
3
0 0 0
0 1 0
5
0 1 0
0 1 0
-4
0 1 0
2 0 1
impossible
```

Problem K: KayTee vs. TeaOne

After long journeys with lots of sweet and crazy victories, KayTee and TeaOne – the top two *Leaf of Lemon* teams in the world, have advanced to the Final. Here they are going to fight against each other for the prestigious *Summer Cup*. As all members of the champion will also be awarded fruity lemon skins, both teams are trying their best for the victory!

For the final preparation before the final, the two teams decide to play a practice series. This series will be a *best of k* , where k is an odd integer. The series looks as below:

- Two teams will play at most k games.
- In *Leaf of Lemon*, a game always ends with a victory for exactly one team. In other words, no draw can happen.
- The team with more winning games wins the whole series.
- At some point in time, if one team is guaranteed to have more winning games no matter how future games go, the series stops immediately.

This is just a practice match and it has nothing to do with deciding the champion. History has proven that several teams (especially teams with names starting with G) used to have dominating performance against TeaOne in practice series, but then lost to TeaOne quickly in important matches. However, Hanh, an eternal fan of TeaOne, is very eager to know the result of the practice series between KayTee and TeaOne.

While surfing through the internet, Hanh gets the result of some of the first games of the series. Hanh would like to know, according to the information he gets, which team will win the series, or whether the series is still going on. It is possible that the information he gets is incorrect and impossible to be true.

Please tell Hanh the result of the series or tell him the information cannot be true.

Input

The first line of the input contains an integer τ ($1 \leq \tau \leq 3 \cdot 10^5$) – the number of test cases. τ test cases follow, each is presented by a single line with an **odd** integer k ($1 \leq k \leq 3 \cdot 10^5$), followed by a string s representing the result of the games. s contains between 1 and k characters. The i -th character of s is K if KayTee wins the i -th game, or 1 if TeaOne wins this game.

It is guaranteed that the sum of k over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, print:

- KayTee if KayTee wins the whole series,
- TeaOne if TeaOne wins the whole series,
- MoveOn if the series is still going on,
- CantBe if the result Hanh gets cannot be true.

Sample Input 1

```
6
5 1KK11
5 K1
5 1111
5 1K1K1
5 11KK1
5 K11K1
```

Sample Output 1

```
TeaOne
MoveOn
CantBe
TeaOne
TeaOne
TeaOne
```

Sample Explanation

In all test cases, the series is *best of 5*.

In the first test case, TeaOne beats KayTee 3 – 2.

In the second test case, both teams are currently tied 1 – 1 and still have a chance to win.

In the third test case, the result shows that TeaOne has won 4 games against KayTee. However, after TeaOne's third winning game, they are leading KayTee 3 – 0 and win the series for sure, as there are at most 2 games left. Therefore, the series should have stopped immediately at this point.

Problem L: Linked List Loop

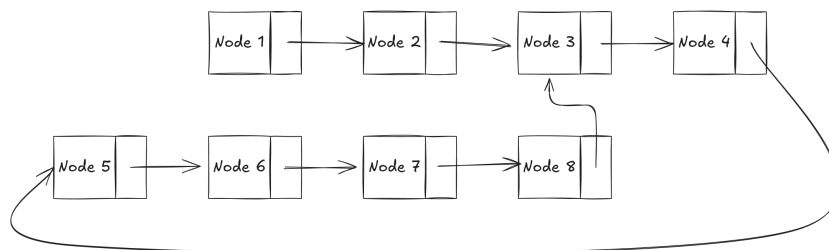
This is an interactive problem.

In computer science, a **linked list** is a linear data structure where elements, or nodes, are connected through links. Each node contains data and a reference (or pointer) to the next node in the sequence. This structure allows the list to grow or shrink dynamically, as nodes are not required to be in contiguous memory locations like an array.

Below is a C++ definition of a linked list:

```
struct Node {
    int value;
    Node* next; // pointer to the next node
}
struct LinkedList {
    Node* head; // pointer to the first node of the linked list
}
```

When a node in a linked list points to a previous node in the linked list, it creates a **loop**. Note that it's possible for a node to point to itself, in which case the loop has a length of 1. The following illustration shows a linked list with 8 nodes and a **loop** of length 6.



In this problem, the judge has prepared an interactor, initialized with a head node of a secret linked list of length n which **has a loop**. You do not have any information about this linked list, even its length n . Your goal is to find the length of the loop.

To do this, you can **mark** some nodes, and when visiting a node, you will be informed by the interactor whether a node has been **marked**. More precisely:

- Initially, all nodes are unmarked.
- The interactor has a pointer variable p . At any point in time, p points to some node of the linked list. Initially, p points to the head node.
- You will give instructions to the interactor in at most $3 \cdot n$ rounds. In each round:
 - You tell the interactor whether you want to **permanently mark** the node that p is currently pointing to, or do nothing with this node.
 - The interactor then set p to its next node. In other words, the interactor performs the following assignment: $p \leftarrow p.next$.
 - The interactor then tell you whether the new node p is pointing to has been previously marked.
- After that, you need to determine the length of the loop.

Interaction Protocol

Each test contains multiple test cases. First, your program reads the number of test cases τ ($1 \leq \tau \leq 10^5$). For each test case:

- First, your program interacts with the interactor in **at most** $3 \cdot n$ rounds. In each round:
 - Your program prints either 1 if you want to permanently mark the node that p is currently pointing to, or 0 otherwise.
 - Your program then reads a character c which can be either Y if the node currently pointed by p (after the interactor the assignment $p \leftarrow p.next$) has been previously marked, or N otherwise.
- Then your program prints -1ℓ to answer that the length of the loop is ℓ . Your program should continue with the next test case or terminate immediately if this is the last one.

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

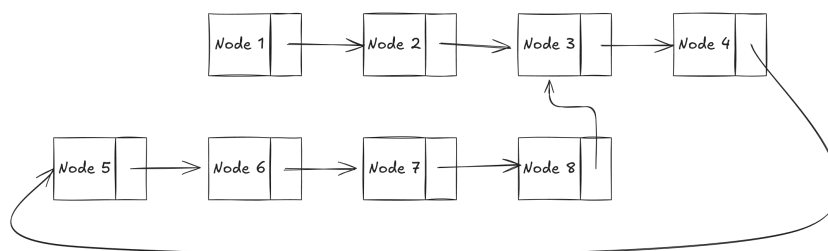
After printing each query do not forget to output the end of line and flush⁵ the output.

Sample Interaction

Stdin	Stdout	Comment
1		There is one test case.
		p points to node 1.
0		You don't mark node 1.
N		p points to node 2. It's not marked.
0		You don't mark node 2.
N		p points to node 3. It's not marked.
1		You mark node 3.
N		p points to node 4. It's not marked.
0		You don't mark node 4.
N		p points to node 5. It's not marked.
0		You don't mark node 5.
N		p points to node 6. It's not marked.
0		You don't mark node 6.
N		p points to node 7. It's not marked.
0		You don't mark node 7.
N		p points to node 8. It's not marked.
0		You don't mark node 8.
Y		p points to node 3. It's marked .
-1 6		You answer that the length of the loop is $\ell = 6$.

The example interaction only shows how your program interacts with the interactor, and does not show how the answer is deduced.

The image below illustrates the linked list used in the sample interaction.



⁵To flush, use: `fflush(stdout)` or `cout.flush()` in C++; `sys.stdout.flush()` in Python; or see the documentation for other languages.

Problem M: Matrix Multiplication

Ming is a former rhythm game player. After realizing that his world rank of 384 is mostly due to farming overrated maps and that he actually sucks (while also failing to qualify for the IOI in the process), Ming decided to go back to competitive programming where he sucks less. However, due to being away from the scene for too long, he now has to relearn most things from scratch!

Today, Ming is practicing matrix multiplications. As a learning aid, he draws himself n matrices, where the i -th matrix has a_i rows and b_i columns. Every minute, Ming chooses two matrices A and B of sizes $p \times q$ and $q \times r$, respectively, then replace them with their product $A \times B$, which is a matrix of size $p \times r$. Note that the number of columns of matrix A must match the number of rows of matrix B , otherwise Ming cannot perform the multiplication. He can stop at any point, or when there is no satisfying pair of matrices to choose.

Unsurprisingly, the “learning” session gets boring quick. Just as he is about to wrap up, Ming realizes a rare property. After some of his operations, the total size across all matrices **strictly increases** compared to the beginning! Formally, let m be the number of operations Ming has done, and c_i, d_i ($1 \leq i \leq n - m$) be the number of rows and columns of the i -th matrix after all operations, then:

$$\sum_{i=1}^n a_i b_i < \sum_{i=1}^{n-m} c_i d_i$$

Ming is very proud of his random invention and challenges you to find **any** sequence of operations that achieves the same property. Of course, since Ming has only relearned how to multiply matrices recently, he could have made a mistake and no such sequence actually exists, in which case you should inform him as well.

Input

Each test contains multiple test cases. The first line contains the number of test cases τ ($1 \leq \tau \leq 10$). The description of the test cases follows.

- The first line contains an integer n ($2 \leq n \leq 1000$) — the number of matrices Ming originally has.
- In the next n lines, the i -th line contains two integers a_i and b_i ($1 \leq a_i, b_i \leq 10^6$) — the number of rows and columns of the i -th matrix, respectively.

Output

For each test case:

- If there exists no sequence of operations that increases the total size of the matrices, print -1 .
- Otherwise, print an integer m ($1 \leq m < n$). Then, print m lines, where the j -th line contains three integers p_j , q_j , and r_j ($1 \leq p_j, q_j, r_j \leq 10^6$), denoting that, for the j -th operation, Ming multiplies a matrix of size $p_j \times q_j$ with a matrix of size $q_j \times r_j$ to produce a new matrix of size $p_j \times r_j$.

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

Sample Input 1

```
3
4
1 4
1 5
2 5
3 6
3
36 69
18 36
67 18
2
2 1000000
1000000 1
```

Sample Output 1

```
-1
2
18 36 69
67 18 69
-1
```

Sample Explanation

In the first test case, there is no valid matrix multiplication using the initial matrices.

In the second test case, the total size of the matrices before all operations is $36 \cdot 69 + 18 \cdot 36 + 67 \cdot 18 = 4338$. After the operations, the size of the only matrix left is $67 \cdot 69 = 4623$.

In the third test case, the total size of the matrices before all operations is $2 \cdot 10^6 + 10^6 \cdot 1 = 3 \cdot 10^6$. The only valid operation is to multiply the first matrix with the second, resulting in a matrix of size $2 \cdot 1 = 2$.