Problem A
Concerts

Input File: A.in
Output File: standard output
Time Limit: 0.3 seconds (C/C++)
Memory Limit: 128 megabytes

John enjoys listening to several bands, which we shall denote using **A** through **Z**. He wants to attend several concerts, so he sets out to learn their schedule for the upcoming season. He finds that in each of the following **n** days ($\mathbf{n} \le 10^4$), there is exactly one concert. He decides to show up at exactly **k** concerts ($\mathbf{k} \le 10^3$), in a given order, and he may decide to attend more than one concert of the same band.

However, some bands give more expensive concerts than others, so, after attending a concert given by band **b**, where **b** spans the letters **A** to **Z**, John decides to stay at home for at least $\mathbf{h_b}$ days before attending any other concert.

Help John figure out how many ways are there in which he can schedule his attendance, in the desired order. Since this number can be very large, the result will be given modulo $\mathbf{10^9 + 7}$.

Input
The first line contains **k** and **n**. The second line contains the **26** $\mathbf{h_b}$ values, separated by spaces. The third line contains the sequence of **k** bands whose concerts John wants to attend e.g., **AFJAZ**, meaning **A**, then **F** etc. The fourth line contains the schedule for the following **n** days, specified in an identical manner.

Output
The number of ways in which he can schedule his attendance (mod $\mathbf{10^9 + 7}$).

Sample input	Sample output
2 10 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (single line) AB ABBBBABBBB	10

Southeastern European Regional Programming Contest
Bucharest, Romania – Vinnytsya, Ukraine
October 21, 2017

Problem B
Bricks

Input File: B.in
Output File: standard output
Time Limit: 0.*2* seconds (C/C++)
Memory Limit: *512* megabytes

Let us consider a zone with **N** boxes, initially empty, numbered from **1** to **N**. We have **M** events given in chronological order. An event is described by a number "**p**": a brick falls at position **p**. If the box at that position is empty, the brick has to stay there. Otherwise, let's consider the full interval of consecutive occupied boxes which contains the box labeled **p**. We have two options: we can put the new brick either on the left side or the right side of the interval (if they exists). The left side of an interval **[a,b]** is position **a - 1**, while the right side is position **b + 1**.

Explanation:
We are given a binary string of length **N**, where **0** denotes a free box and **1** denotes an occupied box: **001110111100**. If a brick falls at position **8** (or any other position inside the interval **[7,10]**), we can place this new brick either at position **6**, or at position **11**. If it falls at position **2** (which is unoccupied), the brick must stay there.

Task:
Given **N**, **M** and a set of **M** events (in chronological order), determine the number of distinct configurations in which we can place the **M** bricks at the **N** possible positions (boxes). The answer should be computed **modulo 1.000.000.007**. The bricks are unordered, so their order does not matter. If we consider the binary interpretation (from the explanation), you are asked to find out how many distinct binary string you can form.

Constraints:
- $1 \leq \mathbf{M} \leq 100.000$
- $1 \leq \mathbf{M} \leq \mathbf{N} \leq 1.000.000$
- $1 \leq \mathbf{p} \leq \mathbf{N}$

Input
- First line: **N**, **M**
- Second line: **M** numbers denoting the **M** events (the values for **p**)

Output
- One number denoting the number of distinct configurations you can obtain **modulo 1.000.000.007**.

Sample input	Sample output
5 4 2 2 4 4	3

Southeastern European Regional Programming Contest
Bucharest, Romania – Vinnytsya, Ukraine
October 21, 2017

Problem C
Christmas Tree

Input File: C.in
Output File: standard output
Time Limit: *0.7* seconds (C/C++)
Memory Limit: *512* megabytes

Santa has a Christmas Tree. In computer science terms, a Christmas Tree is a tree with **N** nodes where each node has a colour. Initially, all nodes have colour **0**. One night, an elf comes (we call him Elf) to paint the tree because he found it very boring. **M** consecutive updates were performed on the given tree the following way: on update number **X**, Elf opens gift number **X** before Christmas (of course, it's someone else's gift) and finds a triplet (**C, A, B**) in it. After that, he paints with colour **C** all the nodes which can be found on the chain formed by vertices **A** and **B**. All colours are different from gift to gift, so we can assume that the **M** colours are **M** distinct values from **1** to **M**. While performing update **X**, some nodes may already be painted, in which case they will just be repainted with the new colour (the old one is lost forever :( )

Unfortunately, we do not know the **M** operations (the triplets found in the gifts), but we do know how the tree looks like in the end (you are given the colour of each node after performing all updates). Your task is to find a possible solution for the generated final form of the tree. More precisely, you have to find **M** triplets (in the correct order), such that if Elf would have found them in the gifts, he would obtain the Christmas Tree described in the input. Since there can be multiple solutions, you can display any of them.

Constraints

- $1 \leq$ **N, M** ≤ 100.000
- The Christmas Tree will contain colours with indexes between **1** and **M**
- It is guaranteed that there always exists at least one solution
- After all **M** operations all nodes will have been painted at least once

Input

- First line: **N**, **M**
- Second Line: **N** integers between **1** and **M**. The **X**-th value denotes the colour of the vertex number **X**
- The next **N - 1** lines describe the edges of the tree. Each such line contains a pair of numbers (**A, B**), meaning that there is an edge between node **A** and node **B**

Output

- **M** lines. Line number **X** should contain a triplet (**C, A, B**), denoting the **X**-th gift found by Elf. **Warning:** Elf performs the drawings in the order of your output, so order **DOES** matter.

Sample input	Sample output
6 3 1 3 2 1 1 2 1 4 2 4 3 4 4 5 5 6	2 6 3 1 5 1 3 2 2
5 3 3 2 1 2 1 1 2 1 3 1 4 4 5	1 3 5 2 2 4 3 1 1

Southeastern European Regional Programming Contest
Bucharest, Romania – Vinnytsya, Ukraine
October 21, 2017

Problem D

Harry Potter and The Vector Spell

Input File: D.in
Output File: standard output
Time Limit: *1* second (C/C++)
Memory Limit: *256* megabytes

Harry Potter has found another strange spell in Half-blood Prince diary, that could generate a different binary vector of size **M**. As he is not the best magician, this spell does not work perfectly so he could generate only vectors where exactly **2** elements are non zero.
Harry has used this spell **N** times and he has constructed a matrix of M rows and N columns, where all generated vectors are columns.

Now Harry has a class of Magical Matrix Theory, where the professor asked him to calculate the rank of such a matrix. You are here to help him!

Operations in Magical Matrix Theory satisfied next rules:

+	0	1
0	0	1
1	1	0

*	0	1
0	0	0
1	0	1

The rank of a matrix A corresponds to the maximal number of linearly independent columns of A. The vectors in a set $T = \{\overrightarrow{v_1}, \overrightarrow{v_2}, \ldots, \overrightarrow{v_k}\}$ are said to be linearly independent if the equation $a_1\overrightarrow{v_1} + a_2\overrightarrow{v_2} + \ldots + a_k\overrightarrow{v_k} = \vec{0}$, where $a_i = \{0,1\}$ for $i = 1, \ldots, k$ can only be satisfied by $a_i = 0$ for $i = 1, \ldots, k$.

Input

On the first line two integers - **M** (size of vectors) and **N** (number of vectors generated by Harry). Each of the next **M** lines has the format: $k_i\ c_1\ c_2 \ldots c_{k_i}$, where k_i is the number of non-zero elements in row i. The next k_i numbers are column indexes ($1 <= c_j <= N, j = 1, \ldots, k_i$), which are non-zero in this row. For more details, see examples.
$1 <= N <= 10^5$
$2 <= M <= 10^5$
$0 <= k_i <= N$

Output

Sample input 1	Sample output 1
3 3 2 1 3 2 1 2 2 2 3	2

Sample input 2	Sample output 2
4 3 3 1 2 3 1 1 1 2 1 3	3

In first example, Harry has generated 3 vectors:

$$\overrightarrow{v_1} = (1, 1, 0), \overrightarrow{v_2} = (0, 1, 1), \overrightarrow{v_3} = (1, 0, 1)$$

and the matrix is:

$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

But $\overrightarrow{v_1} + \overrightarrow{v_2} + \overrightarrow{v_3} = \vec{0}$.

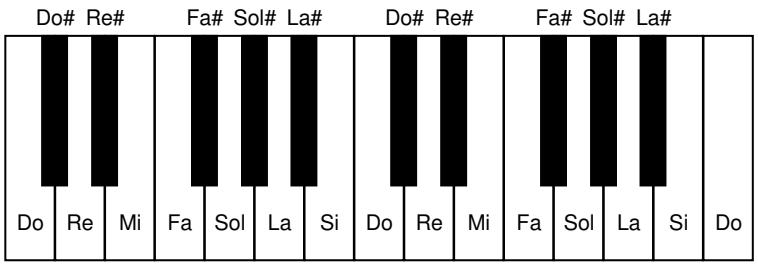
Problem E
Looping Playlist

Input File: E.in
Output File: standard output
Time Limit: *1.5* seconds (C/C++/Java)
Memory Limit: *512* megabytes

A local radio station had a hardware malfunction and is stuck playing the same playlist in a loop. Equipped with pitch detection software, you set out to determine a lower bound to the number of songs in the playlist.

You've managed to record the whole loop, starting at an arbitrary point, and ending at the same point. The pitch detection software's output consists of the N detected musical notes in the order in which they were played.

Every musical note has one of 12 names: Do, Do#, Re, Re#, Mi, Fa, Fa#, Sol, Sol#, La, La#, or Si. The interval between any two consecutive notes from the list is called a *half-step*. Notes that are 12 half-steps apart share the same name; therefore, the note that follows Si is called Do, and so on.



Each of the songs is made up of two or more notes, all belonging to a single *major scale*.

All *major scales* are defined by their *root note* and are comprised of eight notes, of which seven have distinct names. The pitch offsets from the *root note* to each note in the scale are 0, 2, 4, 5, 7, 9 , 11 and 12 half-steps, respectivley (the first and last notes have the same as the root note). It is possible to build a major scale based on any root note. For example:

	Root +0	**Root +2**	**Root +4**	**Root +5**	**Root +7**	**Root +9**	**Root +11**	**Root +12**
Do major	Do	Re	Mi	Fa	Sol	La	Si	Do
Do# major	Do#	Re#	Fa	Fa#	Sol#	La#	Do	Do#
Re major	Re	Mi	Fa#	Sol	La	Si	Do#	Re
Re# major	Re#	Fa	Sol	Sol#	La#	Do	Re	Re#
Mi major	Mi	Fa#	Sol#	La	Si	Do#	Re#	Mi
…et cetera…								

Your task is to determine M, which is the minimum number of songs that the playlist contains.

Input
The first line of the input file contains N <= 10 000 000 representing the number of musical notes detected. The following N lines each contain a musical note. All notes are capitalized as shown previously (i. e. the first letter is upper case; the rest are lower case).

Output
The output has to contain M.

Sample input	Sample output
8 La Si Do Si La Sol Fa Mi	1
9 Mi Si Sol Do Re Fa# Fa Do La	2

Problem F
Binary Transformations

Input File: F.in
Output File: standard output
Time Limit: *1* second (C/C++)
Memory Limit: *256* megabytes

There are **n** bits. Each bit **i** has a value $\mathbf{a_i}$ (0 or 1) and an associated cost $\mathbf{c_i}$. We can change the value of bit **i** with a cost computed as the sum of all the costs $\mathbf{c_j}$ of the bits **j** such that $\mathbf{a_j}$ **= 1 AFTER** bit **i** is changed. What is the minimum amount that should be paid to set each bit **i** to a specified value $\mathbf{b_i}$.

Input
The first line contains the integer **n** ($1 \le \mathbf{n} \le 5 \times 10^3$) - the number of bits
The second line contains *n* integers $\mathbf{c_i}$ ($1 \le \mathbf{c_i} \le 10^9$) - the costs associated with the bits
The third line contains the original **n** values of the bits $\mathbf{a_i}$ - the original values of the bits
The fourth line contains the required **n** values of the bits $\mathbf{b_i}$ - the required values of the bits

Output
Print one number - the minimum cost.

Sample input	Sample output
5 5 2 6 1 5 01110 10011	21

Problem G
Robots

Input File: G.in
Output File: standard output
Time Limit: *0.2* seconds (C/C++)
Memory Limit: *128* megabytes

You're the leading designer of a complex robot, that will explore human unreachable locations. Your job is to design a robot that will go as far as possible. To do this, you have **n** available energy sources. The **i**th source is capable of accelerating the robot by a rate of $\mathbf{a_i}$ (m/s^2) and can do this for a total of $\mathbf{s_i}$ seconds. The robot is initially at rest (its initial velocity is zero). You have to decide the order in which to use the sources in order to maximize the total distance traveled by the robot. You will use one source until $\mathbf{s_i}$ seconds have elapsed, then immediately switch to another unused source (the switch is instantaneous). Each source can be used only once.

Given the accelerations and durations of each source, write an efficient program to determine the optimal order of the sources, in order to maximize the total distance traveled. Your program must compute the difference between the traveled distance in the optimal case and in the default case (the order given by the input data).

Physics background: if the velocity is **v** before you start using a source whose acceleration is **a** then, after **t** seconds, the robot has traveled a total $\mathbf{vt+1/2at^2}$ meters, and the final velocity will be **v' = v+at**.

Input
The input file starts with the number **n (**$1 \le \mathbf{n} \le 10^4$**)** of sources. Starting from a different line follows the **n** space-separated acceleration and duration for each source (positive integer numbers).

Output
The output file contains the computed difference between the traveled distance in the optimal case and in the default case (the order given by the input data), with one decimal.

Sample input	Sample output
2 2 1 30 2	56.0

Southeastern European Regional Programming Contest
Bucharest, Romania – Vinnytsya, Ukraine
October 21, 2017

Problem H
Cat and Mouse

Input File: H.in
Output File: standard output
Time Limit: *10* seconds (C/C++)
Memory Limit: *512* megabytes

A cat and a mouse play a game inside an undirected tree graph with **N** vertices numbered from **1** to **N**. The cat is initially located in vertex **1**, and the mouse in vertex **M**. Each edge of the tree has a unique quantity of cheese (from **1** to **N-1**). The cat and the mouse move alternately, starting with the mouse. At its turn, the mouse moves from its current vertex to a neighboring vertex, using the edge with the largest quantity of cheese. If the cat is located in the chosen vertex, then the mouse will move to the second best neighboring vertex (i.e. the vertex connected to the current vertex with an edge having the 2nd largest quantity of cheese). If there is no second vertex to move to, then the game ends and the cat wins. At its turn, the cat can move to any neighboring vertex or stay in its current vertex.

You control the cat and you want to win the game as soon as possible. Find the minimum possible number of moves the mouse will make before the cat wins the game.

Input
The first line of the input file contains the number of test cases **T**. Then the **T** test cases follow. The 1st line of each test case contains the number **N** and **M**. The next **N-1** lines contain two numbers **u** and **v** each, denoting an edge between the vertices **u** and **v**. The $\mathbf{k}^{th}$ of these edges (**1<=k<=N-1**) has a quantity of cheese equal to **k**.

Output
For each test case, in the order given in the input, print one line containing the minimum number of moves the mouse will make before the cat wins the game (assuming the cat plays optimally). Print **-1** if the cat cannot win the game.

Constraints
- $2 \leq \mathbf{N} \leq 50000$
- $1 \leq \mathbf{T} \leq 100$
- $1 \leq \mathbf{u}, \mathbf{v} \leq N$
- The given graph is a tree in each test case.
- The cheese along the edges is not eaten by the mouse. So an edge always has the same quantity of cheese throughout the game.
- The cat can move to the same vertex as the mouse. The mouse is not harmed by such a move and the game just continues normally.

Sample input	Sample output
3 10 10 6 7 6 5 5 4 4 1 1 2 1 3 7 8 8 9 9 10 10 6 6 7 6 5 5 4 4 1 1 2 1 3 7 8 8 9 9 10 13 5 5 6 6 7 7 8 8 9 9 10 10 11 11 12 12 13 5 4 4 3 3 1 1 2	6 4 4

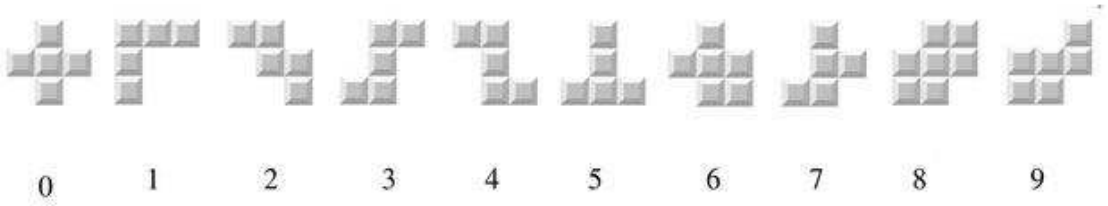
Problem I
Tetris

Input File: I.in
Output File: standard output
Time Limit: *2.5* seconds (C/C++)
Memory Limit: *512* megabytes

Sonya is looking through her old toys. Among cubes and dolls she found the old video game Tetris she loved playing with. This Tetris is quite unusual, it has pieces of **10** different shapes and a grid of width **3** and height **10**. You can imagine the game as an infinite stream of incoming pieces in which a sequence **a** is repeated continuously.

The girl decides to play Tetris. Before a piece falls down, Sonya can rotate it by any angle divisible by **90** degrees, but she can't flip it over. If all the cells in a row are covered, the row disappears, and all the rows on top of it fall down, emptying the row above them. The game is over when there is no more room for the next piece on the grid.

Sonya is smarter now, and she wants to maximize the number of incoming pieces before the game is over. The game might also never end. In this case you should tell Sonya not to start playing.



Input
The first line contains the integer **n** ($1 \le n \le 50$) - the length of the sequence **a**
The second line contains **n** integers a_i ($0 \le a_i \le 9$) - the elements of the sequence **a**

Output
Print one number - the maximum number of pieces that will fall down before game termination or -1, if it is possible to play and get an infinite number of pieces falling down.

Sample input	Sample output
1 0	4
2 3 4	12
3 5 1 1	-1

Problem J
Cunning Friends

Input File: J.in
Output File: standard output
Time Limit: *2* seconds (C/C++)
Memory Limit: *64* megabytes

Anthony and his friends Ben and Chris decided to play a game. They have **N** piles of stones such that the i^{th}-pile contains $\mathbf{A_i}$ stones. In one move a player chooses one pile and may take any non-zero number of stones from it. The players take turns. Anthony goes first then Ben and then Chris. If some player cannot make a move (no more stones exist) he loses. Ben colluded with Chris so their goal is to make Anthony lose. But Anthony doesn't want to lose. You have to find out if Anthony can avoid defeat if all players play optimally.

Input
The first line contains one integer **N** ($1 \le \mathbf{N} \le 10^5$).
The next line contains **N** integers $\mathbf{A_i}$ ($1 \le \mathbf{A_i} \le 10^9$).

Output
Print "Lose" if Anthony will lose in this game and "Win" otherwise.

Sample input	Sample output
3 2 2 1	**Win**
2 4 7	**Lose**

Problem K
Escape Room

Input File: K.in
Output File: standard output
Time Limit: *1* second (C/C++)
Memory Limit: *64* megabytes

As you know, escape rooms became very popular since they allow you to play the role of a video game hero. One such room has the following quiz. You know that the locker password is a permutation of **N** numbers. A permutation of length **N** is a sequence of distinct positive integers, whose values are at most **N**. You got the following hint regarding the password - the length of the longest increasing subsequence starting at position **i** equals $\mathbf{A_i}$. Therefore you want to find the password using these values. As there can be several possible permutations you want to find the lexicographically smallest one. Permutation **P** is lexicographically smaller than permutation **Q** if there is an index **i** such that **Pi** < **Qi** and $\mathbf{P_j} = \mathbf{Q_j}$ for all **j < i**. It is guaranteed that there is at least one possible permutation satisfying the above constraints.
Can you open the door?

Input
The first line of the input contains one integer **N** ($1 \le \mathbf{N} \le 10^5$).
The next line contains **N** space-separated integer $\mathbf{A_i}$ ($1 \le \mathbf{A_i} \le$ N).
It's guaranteed that at least one possible permutation exists.

Output
Print in one line the lexicographically smallest permutation that satisfies all the conditions.

Sample input	Sample output
4 1 2 2 1	4 2 1 3
1 1	1

Problem L
Divide and Conquer

Input File: L.in
Output File: standard output
Time Limit: *2* seconds (C/C++)
Memory Limit: *64* megabytes

Once upon a time in a far away kingdom there were **2** kings that ruled their realm together. This kingdom has **N** towns which are connected with undirected roads. Each king owns a disjoint subset of these roads, and together they own all of the roads. You know that each king owns exactly **N - 1** roads such that it is possible to reach any town from other town using these roads. There might be multiple roads between the same pair of towns.

In this story there is also an Evil Lord that one day decides to conquer the kingdom. It's a well known strategy to divide a kingdom to more easily conquer it. So the Evil Lord is interested in the minimum number of roads he has to destroy such that there exists a pair of towns **X** and **Y** with the property that it is impossible to reach **Y** from **X**. He is also interested in the number of ways to delete the minimum number of roads.

Your task is to find these 2 numbers, the minimum roads the Evil Lord has to destroy to divide the kingdom and the number of ways he can do this. As the number of ways to delete roads may be very big, output it modulo 1.000.000.007($10^9 + 7$).

Input
The first line of the input contains one integer **N** ($2 \le \mathbf{N} \le 10^5$).
Each of the next **N -1** lines contains **2** integers $\mathbf{a_i}$, $\mathbf{b_i}$, which means that first king owns the road between cities $\mathbf{a_i}$ and $\mathbf{b_i}$ ($1 \le \mathbf{a_i}, \mathbf{b_i} \le N$).
Each of the next **N -1** lines contains **2** integers $\mathbf{a_i}$, $\mathbf{b_i}$, which means that second king owns the road between cities $\mathbf{a_i}$ and $\mathbf{b_i}$ ($1 \le \mathbf{a_i}, \mathbf{b_i} \le N$).

Output
Print two integers - the minimum number of roads you need to destroy in order to disconnect the graph and number of ways to do this modulo $10^9 + 7$.

Sample input	Sample output
5 1 3 3 5 2 3 1 4 1 4 4 5 2 5 3 5	2 2
2 1 2 1 2	2 1