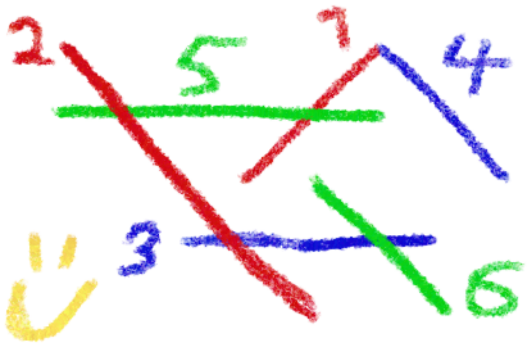
Problem A. Building Marble Tracks

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	1024 megabytes

Jonathan wants you to attach some marble tracks to the wall to play with. Each of them is a straight piece of plastic with a little indent on one side for the marble to roll in. After asking Jonathan how he wants them attached, he spends 15 minutes with some crayons to draw you a blueprint. After he hands it to you, you inform him that you can't build two tracks that intersect at inner points. Jonathan ponders this insight for a moment and then modifies his blueprint. But instead of properly fixing it, he decides to number all tracks from 1 (his most favorite) to n (his least favorite).



The first sample (original blueprint).

It is now your task to figure out which tracks to build. You decide to go through all tracks from his most to least favorite and build them if they do not intersect a track that you have already attached to the wall.

Input

The first line contains one integer n ($1 \le n \le 6 \cdot 10^4$) — the number of marble tracks in the blueprint.

Each of the following n lines contains four integers x_1, y_1, x_2, y_2 ($-10^5 \le x_1, y_1, x_2, y_2 \le 10^5$ and $x_1 \ne x_2$). These represent a marble track from (x_1, y_1) to (x_2, y_2) in the blueprint.

They are given in order from most to least favorite. Two different tracks intersect in at most one point.

Output

Output one integer k in one line, the number of tracks to build. In the next line, output k integers, the indices of the tracks to build in increasing order.

Examples

standard input	*standard output*

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

2
0 0 2 2
1 1 2 0
	2
	1 2

10
-30428 50667 -18028 82591
-19828 85240 -16439 54314
-16439 54314 -13911 33099
-18028 82591 -13739 13271
-13911 33099 -11978 -46941
-13739 13271 14783 13050
-11978 -46941 35511 -79125
35511 -79125 35524 -89845
35524 -89845 67295 -41898
14783 13050 72720 -67171
	6
	1 3 5 7 8 9

Note

As you might have noticed, the input format specifies $x_1 \neq x_2$ for each possible marble track. This is because you can't roll a marble down a vertical track.

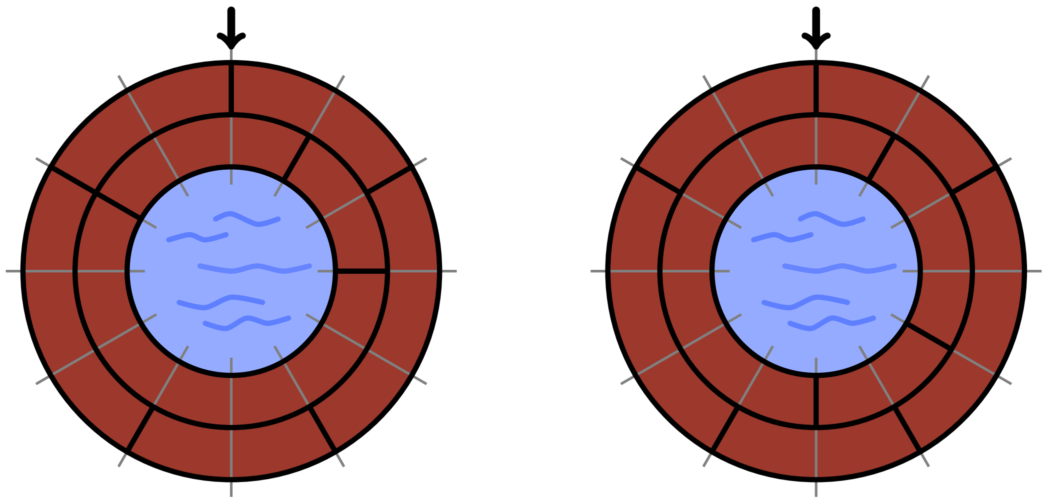
Problem B. Build Well

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

Bob the Builder is tired of building tiny houses and paving narrow roads, and he strives for something bigger. The new job given to him by a very eccentric client is exactly what he needs: He is tasked with building a round well of a certain circumference that is infinitely deep! His client assured him that he does not need to worry about the building material, and that an infinite supply of various kinds of bricks, only differing by their arc lengths, has already been ordered for him.

Of course, building a stable well takes very careful planning, especially if it is supposed to be infinitely deep. In particular, a well is only stable if no two gaps between bricks in consecutive rows end up directly above each other, as shown in the figure below. The bricks are all of integer length and can only be offset by an integer length. Note that even a brick that covers the complete circumference has a start and an end and therefore, a gap.

Bob knows from his long-time experience that if it is possible to build such a well, then it can be done by alternating just two row configurations.



On the left, we see an unstable well using the brick types of Sample Input 1. On the right, we see a stable well using the same brick types. For visual reasons, bricks on the outer rows appear larger even though they have the same total arc length. Note that even though only two rows of the well are shown, it is possible to build an infinitely deep well by repeating these two row configurations. The arrow points to the zero offset of the sample.

Bob is terribly excited about the new job and quickly goes to work. Given the types of arc bricks available, is it possible to build a stable well of circumference exactly w and infinite height? If yes, how should Bob build it using only two alternating row configurations?

Input

The input consists of one line with two integers n and w ($1 \le n, w \le 3 \cdot 10^5$) — the number of brick types and the circumference of the well. The following line contains n integers b_i ($1 \le b_i \le w$) — the arc lengths of the brick types.

Note that Bob has an infinite supply of all brick types.

Output

If it is possible to build a well, output `possible`. Otherwise, output `impossible`.

If a well can be built, provide two row configurations that can be used in an alternating fashion. For both row configurations, first output the number of bricks needed for that row and the clockwise offset where to place the first brick in a single line, followed by a line containing the arc lengths of the bricks in the clockwise order you want to use them. Note that the offset has to be a non negative integer smaller than the circumference of the well. Your solution is considered valid if alternating the two rows infinitely would result in a stable well.

If there are multiple valid solutions, you may output any one of them.

Examples

standard input	*standard output*

`4 12`
`3 2 7 2`
	`possible`
	`5 0`
	`2 3 2 3 2`
	`3 1`
	`3 2 7`

`3 11`
`6 7 8`
	`impossible`

Problem C. Centrifuge

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

On your latest intergalactic scavenger trip, you discovered an abandoned futuristic robot. Mysteriously, the robot consists of n ball-shaped joints with some kind of fluid inside. These joints are all connected together via $n-1$ flexible pipes that allow the fluid to flow bidirectionally from one joint to another. To analyze this strange fluid inside the robot, you decide to use a centrifuge.

Since you have no clue how to actually use a centrifuge, you secure one of the joints to the center of it and turn the machine on. The machine then rapidly spins around the center, and all the fluids are pushed away from the center towards the outside. Whenever the fluid has more than one pipe to flow through, the fluid splits evenly between all of these pipes.

You wonder how much fluid will be in the outermost joints at the end of this process. After thinking so much, you forgot which joint you secured to the center. Therefore, you have to calculate the expected amount of fluid in each joint as if you chose a joint at random.

Input

The first line contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the number of joints of the robot.

The next line contains n integers a_i ($0 \le a_i \le 10^9$) — the amount of fluid in the i-th joint.

The next $n-1$ lines contain two integers u and v ($1 \le u, v \le n$) — indicating a pipe between joint u and v. These pipes form a tree.

Output

Output n lines with one integer per line, the i-th representing the expected amount of fluid in the i-th joint after the process modulo 998 244 353.

Formally, let $M = 998\,244\,353$. It can be shown that the answer can be expressed as an irreducible fraction $\frac{p}{q}$, where p and q are integers and $q \not\equiv 0 \pmod{M}$. Output the integer equal to $p \cdot q^{-1} \bmod M$. In other words, output such an integer x that $0 \le x < M$ and $x \cdot q \equiv p \pmod{M}$.

Examples

standard input	*standard output*

3
1 2 4
1 2
2 3
	3
	0
	4

6
4 1 3 11 7 2
1 4
2 3
6 2
2 4
4 5
	928921837
	0
	818005794
	0
	679360751
	568444705

Note

In the first sample, if the first joint is fixed at the center, all fluid will flow from joint 1 to 2, and the combined fluid will flow to joint 3. In total, joint 1 and 2 will be empty, and all 7 units of fluid will be at joint 3.

If the second joint is fixed to the center, half of its fluid will flow to 1 and half will flow to 3. There will be 2, 0, and 5 units of fluid at each joint respectively. If joint 3 is fixed, all 7 units of fluid will be at joint 1.

In total, the expected amount of fluid in each joint is

$$\frac{1}{3} \cdot (0 + 2 + 7) = 3$$
$$\frac{1}{3} \cdot (0 + 0 + 0) = 0$$
$$\frac{1}{3} \cdot (7 + 5 + 0) = 4$$

Problem D. Infinity Triples

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

Consider numbers in base b where all digits are equal to a with $1 \le a < b$. We call a triple (n, a, b) an *infinity triple* if infinitely many of those numbers are divisible by n.

For example, $(3, 9, 10)$ is an infinity triple because infinitely many of the numbers 9, 99, 999, ... are divisible by 3. The triple $(7, 9, 10)$ is also an infinity triple, but $(5, 9, 10)$ is not.

Given m, count the number of infinity triples with $1 \le n \le m$ and $1 \le a < b \le m$.

Input

The input contains one integer m $(2 \le m \le 10^5)$.

Output

Output one integer, the number of infinity triples with $1 \le n \le m$ and $1 \le a < b \le m$.

Examples

standard input	*standard output*
2	1
3	6
42	25055

Note

In the first sample, $(1, 1, 2)$ is the only infinity triple.

In the second sample, the infinity triples are $(1, 1, 2), (1, 1, 3), (1, 2, 3), (2, 1, 3), (2, 2, 3)$, and $(3, 1, 2)$.

Problem E. Taxi

Input file:	*standard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	1024 megabytes

There are n cities that are connected by $n-1$ roads, forming a tree. Note that each road has a given length.

When you are at city v, you can take a taxi of the local taxi company to any other city w. For this, you have to pay $a_v + d \cdot b_v$ cookies, where d is the distance from v to w. In other words, you have to pay the base cost a_v and additionally b_v for each unit of distance traveled.

You are currently at city 1, and for each other city v, you want to know the minimum cost to get there.

Input

The first line contains one integer n ($2 \le n \le 10^5$) — the number of cities.

The second line contains n integers a_i ($0 \le a_i \le 10^{12}$) — the base costs of the taxis.

The third line contains n integers b_i ($1 \le b_i \le 10^6$) — the cost per distance.

Then $n-1$ lines follow, describing the roads between the cities. Every line contains three integers u, v, and ℓ ($1 \le u, v \le n$, $u \ne v$, $1 \le \ell \le 10^6$) describing a bidirectional road between cities u and v of length ℓ.

Output

Output a single line containing $n-1$ integers. The i-th of them should be the minimum cost to get to city $i+1$.

Examples

standard input	*standard output*

3
0 1 2
8 4 4
1 2 1
1 3 7
	8 41

2
353 313
928248 475634
2 1 898027
	833591767049

Note

Consider the cost to get to city 3 in the first sample: Driving directly from 1 to 3 would cost $0 + 7 \cdot 8 = 56$. It is better to drive from 1 to 2 with a cost of 8 and take a second taxi from 2 to 3 with a cost of $1 + 8 \cdot 4 = 33$. While the distance traveled is larger, the cost is still smaller.

Problem F. Periodic Sequence

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

As you may know, some fractions of integers $\frac{A}{B}$ result in an infinite periodic decimal representation. For example, $\frac{4}{7}$ results in $0.57\overline{142857}$, which means that the 142857 part is repeating itself. Note that we could also write $0.5714\overline{285714}$ or $0.571\overline{428571428571}$. As you see, it is not trivial to check if two such sequences are equal.

Therefore, you have to help us. You are given only the periodic part of two sequences and need to check if they are equal. Note that the periodic parts are considered equal if they can be made equal by repetition and cyclic shifting.

Input

The first line contains two integers n and m ($1 \le n, m \le 5 \cdot 10^5$) — the length of the first and second sequence.

The second line contains n integers a_i ($0 \le a_i < 10$) — the first sequence.

The third line contains m integers b_i ($0 \le b_i < 10$) — the second sequence.

Output

Print `YES` if the two sequences are equal and `NO` if they are not.

Examples

standard input	*standard output*

`6 3`
`1 5 6 1 5 6`
`6 1 5`
	`YES`

`7 3`
`1 5 6 1 5 6 7`
`5 6 7`
	`NO`

Problem G. Scheduling

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	1024 megabytes

Your boss made you find a valid schedule for his meetings. He has n meetings, where the i-th meeting has to start and end in the time interval $[\ell_i, r_i]$, where time is measured in hours. Each meeting takes exactly one hour, and your boss obviously cannot attend two meetings at the same time.

As a competitive programmer, you have already solved this classical scheduling problem countless times, and this time, you finally notice that it is completely unrealistic. How can one person attend over a hundred thousand meetings at different places without any time to travel between them and without taking any breaks either? There is no way every meeting takes exactly one hour – speaking from experience – and there will surely be meetings that start late because someone is running late.

These deep-rooted philosophical questions may be a big problem for humanity, but since you do not have to attend these meetings yourself, you simply do not care enough to solve all of them. However, impressing your boss will certainly be worth it for you, so you try to at least improve the schedule a little bit.

Find any schedule where there is at least one hour between every pair of meetings or report that there is none. Surely this is just one insignificant change to the classical problem, right?

Input

Each test contains multiple test cases. The first line of input contains a single integer t ($1 \le t \le 10^5$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the number of meetings.

The next n lines each contain two integers ℓ_i and r_i ($1 \le \ell_i < r_i \le 10^6$) — the time interval in hours the i-th meeting has to start and end in.

The sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

If there is no valid schedule, output -1 in a single line. Otherwise, output n integers a_i in a single line where a_i is the start time of the i-th meeting.

Example

standard input	*standard output*

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

Note

In the first test case, the first meeting starts at the start of hour 1 and goes on for one hour until the start of hour 2. After that, your boss takes a break of one hour until he attends the third meeting from the start of hour 3 to 4. Then, he takes another one hour break until he finally attends the second meeting, which happens from the start of hour 5 to 6.

Note that it is **not** possible to start the third meeting at the start of hour 4 as this meeting would end at 5.

Problem H. Mod Graph

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

You are given an undirected, connected graph with n vertices. Every vertex v has a counter that operates modulo b_v. The initial state of that counter is a_v. Every time you visit that vertex, it is increased by one (i.e. $a_v \leftarrow (a_v + 1) \bmod b_v$).

You have to process q queries of the following two types:

- "`1` s": Suppose that you start at vertex s, you have to answer whether it is possible to make $a_v = 0$ for all v. You are allowed to take an arbitrary walk through G, i.e. you can use edges and vertices multiple times. Note that starting the walk at s already counts as visiting s, i.e. its counter is increased by one initially.
- "`2` v x": Update $a_v \leftarrow x$.

Input

The first line contains three integers n, m, and q ($1 \le n, q \le 5 \cdot 10^4$, $0 \le m \le 10^5$) — the number of vertices, edges, and queries, respectively.

The second line contains n integers $a_1, \ldots, a_n$.

The third line contains n integers $b_1, \ldots, b_n$ ($0 \le a_v < b_v \le 10^9$).

The following m lines contain the edges of the graph. Each of those m lines contains two integers u and v ($1 \le u, v \le n$, $u \ne v$) describing an edge connecting vertices u and v. The given graph is connected.

Finally, there are q lines describing the queries. They can be in the two formats described above, i.e.

- "`1` s": ($1 \le s \le n$)
- "`2` v x": ($1 \le v \le n$, $0 \le x < b_v$)

Output

For each query of type `1`, output `YES` in one line if it is possible to make $a_v = 0$ for all v and `NO` otherwise.

Example

standard input	*standard output*

2 1 4
0 0
3 3
1 2
1 1
2 1 1
1 1
1 2
	YES
	NO
	YES

Note

In the first query, we start at vertex 1 with counter states $[1, 0]$.

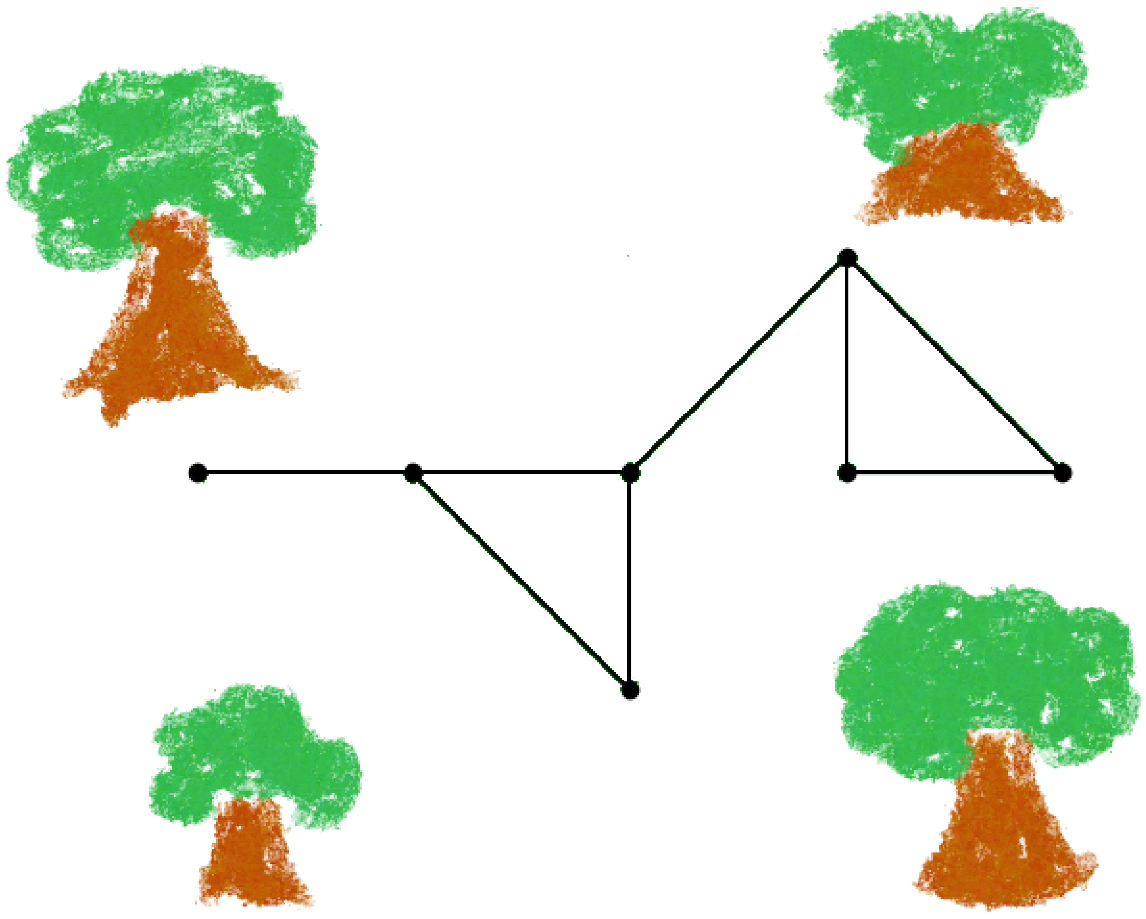
We move along the walk $1 \to 2 \to 1 \to 2 \to 1 \to 2$.

The counter states change as follows: $[1, 0] \to [1, 1] \to [2, 1] \to [2, 2] \to [0, 2] \to [0, 0]$.

Problem I. In the Treetops

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

Kindergarten Timelimit is taking a trip to a treetop park. A treetop park has platforms attached to some trees and rope bridges connecting these platforms. Due to the laws of physics, you can assume that all bridges are straight when seen from above. You can walk from any platform to any other by just using these bridges. For safety reasons, no two bridges cross over each other. Additionally, it is possible to reach every tree with a platform from outside the park without passing under any bridge.


The layout of the park in sample 1. The trees surrounding the park layout are decorative.

The children of the Kindergarten want to go to every platform on their visit to the park. You know that you have barely enough time to visit all of them, so you need to find a route that doesn't visit any platform twice. However, the park offers a nice shuttle service: They drop you off at whatever tree with a platform you want to start your journey, and when you're done with your visit, they pick you up again. Every tree with a platform has a ladder to climb up or down, so you can freely choose where to start and end your visit.

Input

The first line contains two integers n and m ($1 \le n \le 10^5$, $0 \le m \le 2 \cdot 10^5$) — the number of platforms and the number of bridges.

The following n lines contain two integers x and y ($-10^9 \le x, y \le 10^9$) — the location of the trees with platforms.

The following m lines contain two integers u and v ($1 \le u, v \le n$, $u \ne v$) describing a bridge between platform u and v. Any pair of platforms has at most one bridge between them.

Output

Output `YES` in a single line if there is a path visiting each platform once, and `NO` otherwise.

Examples

standard input	*standard output*
`7 8`	`YES`
`0 0`	
`1 0`	
`2 -1`	
`2 0`	
`3 1`	
`3 0`	
`4 0`	
`1 2`	
`2 3`	
`2 4`	
`3 4`	
`4 5`	
`5 6`	
`5 7`	
`6 7`	
`6 6`	`NO`
`0 0`	
`1 0`	
`2 1`	
`2 -1`	
`3 0`	
`4 0`	
`1 2`	
`2 3`	
`2 4`	
`3 5`	
`4 5`	
`5 6`	
`1 0`	`YES`
`1000000000 -1000000000`	

Problem J. Permutation Recovery

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

Initially, we had k permutations of the integers from 1 to n. We created a $2k \times n$ matrix by writing each permutation as well as its inverse in its own row. However, we forgot the permutations, and someone shuffled every column. Given this matrix, can you determine any set of permutations which we could have started with?

Input

The first line contains two integers n and k ($1 \le n \le 4 \cdot 10^4$, $1 \le k \le 7$).

The i-th of the following $2k$ lines contains n integers a_{ij}, the i-th row of the matrix ($1 \le a_{ij} \le n$).

It is guaranteed that the matrix could have been obtained as described above.

Output

Output k lines. Each of them should contain a permutation of the integers from 1 to n. After writing these permutations as well as their inverses in the rows of a $2k \times n$ matrix, it must be possible to obtain the input matrix by reordering the values in every column.

If there are multiple solutions, output any of them.

Examples

standard input	*standard output*

3 1
1 2 3
1 2 3
	1 2 3

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