Problem A. Edit Distance Yet Again

Input file:	*standard input*
Output file:	*standard output*
Time limit:	20 seconds
Memory limit:	512 mebibytes

Have you ever heard of the *edit distance* problem? Given two strings of lowercase English letters, you must determine the minimum number of operations needed to transform the first one into the second one. A single operation can be either:

- inserting a character into the sequence, at any spot,
- deleting any character from the sequence,
- substituting a character with another one.

Everyone at our university loves this problem a lot – maybe a little bit too much – so we decided to create a problem which is easier! You are given two strings $s = s_1 \dots s_n$, $t = t_1 \dots t_m$ and an integer k. Find out whether the edit distance between the strings is less than or equal to k. If so, you are also asked to provide any sequence of minimum possible number of operations to transform the first string into the second one.

Input

The first line of input contains the number of test cases z $(1 \le z \le 100)$. The descriptions of the test cases follow.

The first line of each test case contains three integers n, m, k $(1 \le n, m \le 1\,000\,000,\ 0 \le k \le 1000)$ – the lengths of the strings and the parameter from the problem description.

The second line contains a string of length n consisting of lowercase English letters – the string s from the problem description.

The third line contains a string of length m consisting of lowercase English letters – the string t from the problem description.

The total length of all strings in all test cases will not exceed 10^7.

Output

For each test case, if the edit distance is greater than k, output a single line containing the word "`NO`". Otherwise, the first line should contain the word "`YES`", and the next lines should describe the answer as follows:

In the second line output the minimum number r of operations required to transform s into t. In the next r lines output the operations, one per line.

- To insert a lowercase English character c into a sequence of size w at the position p $(1 \le p \le w + 1)$, print `INSERT p c`.
- To delete a character from a sequence of size w from the position p $(1 \le p \le w)$, print `DELETE p`.
- To substitute a character in a sequence of size w from the position p $(1 \le p \le w)$ with a lowercase English character c, print `REPLACE p c`.

Example

standard input	standard output
2 3 4 3 kot plot 5 7 3 zycie porazka	YES 2 REPLACE 1 l INSERT 1 p NO

Problem B. (Almost) Fair Cake-Cutting

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

You are surely familiar with the *fair cake-cutting* scheme, where one person cuts the cake in two and the other person gets to choose which part they would prefer to eat. This solution is supposed to be fair as neither of the participants can claim to have received the smaller part as the result.

Well, at Alice's, it is her who dictates the rules – and they are most certainly not supposed to be fair. She orders her younger brother, Bob, to make n cuts rather than one. Now, for *every* cut, Alice chooses one of the sides and eats all the cake at this side. After she finishes going through all the cuts, Bob gets to eat the rest.

The cake is represented as a square on the Cartesian plane (it is actually a cuboid, of course, but we assume all the cuts to be perpendicular to the surface) with side length M. Bob has just made n cuts and now it is time for Alice to make her choices. Determine how much cake will she be able to eat if she chooses wisely.

Input

The first line of input contains the number of test cases z $(1 \le z \le 500)$. The descriptions of the test cases follow.

The first line of every test case contains two integers n $(1 \le n \le 4\,000)$ and M $(1 \le M \le 1000)$ – the number of cuts and the cake's side length. The cake is a square with its opposing vertices located in points $(0, 0)$ and (M, M).

Then follow n lines, the i-th of them containing three integers A_i, B_i and C_i $(-1000 \le A_i, B_i \le 1000, -10^6 \le C_i \le 10^6, A_i^2 + B_i^2 > 0)$, which define the line equation $A_i x + B_i y + C_i = 0$ of the i-th cut.

More precisely, Alice is given a set of n line equations. For each equation, she needs to replace the $=$ operator with either $\le$ or $\ge$, obtaining a half-plane equation. The intersection of the cake with the sum of n such half-planes is what Alice will be allowed to eat.

Each cut splits the cake into two parts of non-zero area each.

The total number of cuts in all test cases does not exceed 10 000.

Output

For each test case output a single line containing a real number P $(0 \le P \le 100)$ with 6 decimal digits, followed by the '%' sign – the percentage of the cake which Alice will be able to eat if she chooses all sides of the cuts optimally. Your solution will be accepted if P differs from the correct percentage by no more than 0.000002%.

Example

standard input	standard output

1
2 1000
0 1 -750
1 0 -750
	93.750000%

Problem C. Jellyfish

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Everyone knows that at Jagiellonian University we do love plants a lot. We created hundreds of problems about trees, forests and even cacti! Unfortunately, problems about animals are not that popular. Today we want to prove that we love animals as well.

We say that a graph is a *jellyfish*, if it is a simple connected undirected graph with equal number of vertices and edges. You are given a jellyfish J with n vertices. For an arbitrary subset of vertices $S \subseteq J$, we say that S is an *awesome* subset if for every $T \subseteq S$ there exists a **connected** subgraph of the jellyfish which contains every vertex from T and does not contain any other vertex from S.

What is the maximum possible size of an awesome subset of J?

Input

The first line of input contains the number of test cases z. The descriptions of the test cases follow.

The first line of each test case contains one integer n ($3 \leq n \leq 100\,000$) – the number of vertices of the jellyfish.

The next n lines contain two integers u_i, v_i ($1 \leq u_i \neq v_i \leq n$) each, corresponding to the jellyfish edges. It is guaranteed that the given graph is a jellyfish, and every two vertices are connected by at most one edge.

The total number of vertices in all test cases does not exceed 10^6.

Output

For each test case, output a single line which contains a single integer – the maximum possible size of an awesome subset of the jellyfish.

Example

standard input	standard output

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

Problem D. Flat Organization

Input file:	*standard input*
Output file:	*standard output*
Time limit:	12 seconds
Memory limit:	512 mebibytes

The company you are currently working for has decided to push the idea of flat organizational structure to its limits: for every pair of employees A and B, either A has been assigned to directly supervise B's work or B has been assigned to directly supervise A's work. Of course, it means that one might now have quite a lot of direct supervisors... which is great, because it makes an employee feel that their work truly is important to so many people rather than just to a single manager, the executives say.

There is always room for improvement, though. As the corporate goal for this year, the hierarchy will be revised to ensure that whenever a person A is directly supervised by a person B, then B is also indirectly supervised by A at the same time (we say that B is indirectly supervised by A if there exists $n > 2$ and a sequence $(c_1, \ldots, c_n)$ such that $c_1 = A, c_n = B$ and for each $i < n$, c_i is a direct supervisor of c_{i+1}).

It will ensure that any employee would think twice before deciding to abuse their position of power over anyone else, the executives say.

It should not come off as a surprise, though, that one might get somewhat annoyed if they learn that their supervisee has suddenly been appointed as their supervisor. And some such decisions might cause more resentment than others. Your task is to fulfill the corporate goal by reversing some of the dependencies between employees in such a way that the sum of resentments over these changes is as small as possible.

Input

The first line of input contains the number of test cases z $(1 \le z \le 100)$. The descriptions of the test cases follow.

The first line of every test case contains a single integer n $(1 \le n \le 2000)$ – the number of employees. The employees are numbered from 1 to n.

Then follow n lines, containing n integers $d_{i,j}$ each $(0 \le d_{i,j} \le 2 \cdot 10^9)$. If the employee i is a direct supervisor of employee j, then $d_{i,j} > 0$ describes the resentment that i would feel if this dependency got reversed. Otherwise (that is, if j is a direct supervisor of i or if $i = j$), $d_{i,j} = 0$.

The total number of employees in all test cases does not exceed 10000.

Output

For each test case, produce a solution which ensures that, for any pair of employees i, j $(1 \le i, j \le n, i \ne j)$, either i will be a direct supervisor of j and j will be an indirect supervisor of i, or vice versa. Your solution should minimize the sum of resentments the employees will feel. If more than one such solution exists, you can print any of them.

If no solution exists, you should output a single line containing the word "`NO`".

Otherwise, in the first line output a single word "`YES`". In the second line, print two integers k and r – the number of dependencies between employees which you intend to reverse and the achieved sum of resentments, respectively. Note that you do not need to minimize k.

Then output k lines, each containing two integers – the identifiers of employees a, b $(1 \le a, b \le n, a \ne b)$ such that a is currently a direct supervisor of b and their relationship should get reversed. You should never output the same pair of employees more than once.

Example

standard input	standard output

1
5
0 1 0 6 11
0 0 1 6 12
2 0 0 7 12
0 0 0 0 4
0 0 0 0 0
	YES
	2 10
	4 5
	2 4

Problem E. Archer Vlad

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	512 mebibytes

Vlad was an exemplary student known for his exceptional adventures, many of which have been preserved as tasks for programming contests. But such a restless life had exhausted Vlad a bit too much. "Wherever I go, there are only problems! I'm done!" – he announced, right before he left the university, headed toward Bieszczady mountains.

Vlad rented a small hut, in which he spent the first few months of his vacations. But soon boredom started having its grasp on him, so Vlad decided to find himself a hobby: he bought a bow, a few arrows and started his daily practice of archery. And after a few more months under solid training, Vlad reached some very satisfying results, as he was able to shoot an arrow with an astonishing speed of C meters per second. But it was hard to enjoy such accomplishments with no one around.

"Check this out! I'm gonna stand right here and shoot an arrow so quick, that it's gonna fly over each and every single one of all those trees!" — Vlad exclaimed to you, a young programmer who decided to pay him a visit. Vlad tightened the bow and shot the first arrow. Its feathers were swaying in the air, its arrowhead was shining in the sky... but it hit a tree. "Hold on, let me try again!"

His second attempt was even more spectacular than the first one. But this arrow could not find its way out of the forest either. "One last time!" Vlad shouted, reaching his hand to the sack once again. Then you stopped him. Afraid that Vlad would run out of arrows, you decided to find an optimal angle at which he should aim. And so you reached to the computer in your backpack, ready to solve this problem in the UJ TCS style.

Vlad stands on a Cartesian plane at the point $(0, 0)$. Both points $(0, 1)$ and $(1, 0)$ are precisely 1 meter away from Vlad. There are N trees numbered from 1 to N, and the tree number i is represented by a vertical segment connecting the points $(x_i, 0)$ and (x_i, y_i) for some positive integers x_i and y_i. When Vlad shoots at an angle α it gives his arrow an initial horizontal speed v_x equal to $C \cdot \cos(\alpha)$ and an initial vertical speed $v_y = C \cdot \sin(\alpha)$. The arrow is not affected by air resistance and its trajectory is a parabola (to be precise, its horizontal speed v_x stays constant throughout the whole flight, while v_y decreases linearly with per-second loss equal to g), containing the point $(0, 0)$. We assume that the acceleration due to gravity is $g = 10m/s^2$. Vlad's objective will be reached if the trajectory of the arrow he shot does not intersect any of the trees (or more specifically intervals representing them) at any point. Furthermore, the trajectory of the arrow must intersect the x-axis at the point which has a greater x-coordinate than any tree.

Output a possible value of $\tan(\alpha)$ which allows Vlad to meet these conditions.

Input

The first line of input contains the number of test cases z. The descriptions of the test cases follow.

The first line of each case consists of an integer $1 \leq C \leq 10^9$ which is the speed of Vlad's arrow in meters/second.

The second line of each case contains a single integer $1 \leq N \leq 100\,000$ – the number of trees.

For each case the next N lines contain two integers x_i, y_i ($1 \leq x_i, y_i \leq 10^9$) each. The i-th tree is represented by a vertical segment between points $(x_i, 0)$ and (x_i, y_i).

The sum of N in all the test cases does not exceed $300\,000$.

Output

For each case output a single number with exactly 3 digits after the decimal point. It must approximate one of the correct values of $\tan(\alpha)$ with error no greater than 10^{-3}. You may assume that the solutions always exist, and that any correct value of $\tan(\alpha)$ is contained in an interval of solutions of length at least 10^{-2}.

Example

standard input	standard output

3
5
1
1 1
5
1
1 1
13
1
7 7
	2.000
	3.000
	2.429

Problem F. A Very Different Word

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Working as a freelancer has never been easier, you are thinking to yourself, laying in a hammock and having a drink, scrolling lazily over the next page of requests for work to do. Suddenly, you notice an unusual request. I must even say, a strange one. One writer is looking for a... word. No, not a usual word, he is in desperate need of an unusual one. You decide to take this job. After all, who is more experienced in programming some strange stuff than you?

The next day, you get all the details. The request is from a renowned author who is currently stuck on writing his next novel. Like, really stuck... to the point that the last season of the TV series based on his work has already aired. After signing a non-disclosure agreement, you learn that the truth is more complicated than it seemed. The book has actually been almost complete for several years already, but since then the author has kept rewriting a single chapter which he can never get quite right. The chapter revolves around a crucial prophecy, which is intended as a very intricate wordplay on three words of exactly the same length.

You know that the first word s is lexicographically earlier than the last word t are they have the same number of characters. Your client wants to find a word x of the same length, which is lexicographically strictly between s and t and at the same time contains the first letter of the promised hero's name: the character K. It is possible that such a word x does not exist (which would fully explain all the delays), but... who knows?

Input

The first line of input contains the number of test cases z ($1 \le z \le 100\,000$). The descriptions of the test cases follow.

The first line of a test case contains one integer n – the length of s and t ($1 \le n \le 25\,000$) – and a lowercase letter K. The next two lines contain words s and t, composed of lowercase letters of the English alphabet.

The sum of n among all the test cases does not exceed $100\,000$.

Output

For each test case output a single line with one string: any word x of length n, composed of lowercase letters of the English alphabet, which meets the requirements or "NO" in case no such word exists.

Example

standard input	standard output
4 10 m christmasa christmasx 6 m spring winter 21 a ithinkthereforeisleep ithinkthereforeithink 3 z tcs tcz	christmass summer ithinkthereforeistand NO

Problem G. Cactus

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

Let me be upfront about this: things are not going well. What was supposed to be a relaxing evening with friends has taken a turn for the worse: you got assaulted by a walking advertisement of cheap cologne, and your Priceless Argentinian Cactus – the only thing you hold dear – was thrown out of the window.

Right after the deed – or, shall I say, as soon as was physically possible – you ran down the stairs to assess losses. And there it was, your priceless cactus, alive! With a few scratches here and there, but alive nonetheless. How did this happen? Did it land on something soft? Overwhelmed with joy, you decide not to seek answers. Did I say things are not going well? Scratch that, everything is great – and it's time to celebrate! Of course, at the heart of this celebration will be your green stinging friend.

Those less acquainted with botany may now appreciate a refresher: a cactus is a connected graph, where each vertex lies on at most one cycle. To add to the festive mood, you decide to color every vertex of your cactus with one of k colors. You would like to give yourself lots of freedom here, but you do want to adhere to the golden rule of cactus coloring: no two adjacent vertices should be assigned the same color.

One coloring seems not enough, so you decide that after the colors fade, you will recolor the cactus again and again, using a different coloring each time. But how long will you be able to keep this going? Given the description of your cactus and the number k, count the number of correct k-colorings of the cactus' vertices. Since the answer may be very large, it's enough if you compute its remainder modulo $10^9 + 7$.

Input

The first line of input contains the number of test cases z ($1 \le z \le 50\,000$). The descriptions of the test cases follow.

The first line of a test case contains three integers n, m and k ($1 \le n \le 300\,000$, $0 \le m \le 400\,000$, $2 \le k \le 10^9$) – the number of vertices and edges of your cactus, and the number of colors.

The next m lines contain two integers u_i, v_i ($1 \le u_i \ne v_i \le n$) each, corresponding to the cactus edges. It is guaranteed that the given graph is a cactus and that every two vertices are connected by at most one edge.

The total number of vertices and edges in all test cases does not exceed $3 \cdot 10^6$ and $4 \cdot 10^6$, respectively.

Output

For each test case output a single integer: the number of proper colorings of the vertices of the cactus with k colors, taken modulo $10^9 + 7$.

Example

standard input	standard output

2
2 1 100
1 2
6 7 3
1 2
2 3
3 1
4 5
5 6
6 4
1 4
	9900
	24

Problem H. Social Distancing

Input file:	*standard input*
Output file:	*standard output*
Time limit:	10 seconds
Memory limit:	512 mebibytes

Two things ought to be said about students: they hate to do more work than necessary, and love distancing themselves from others. The former is probably why the department forms a tree (building a corridor between two indirectly connected rooms would be a waste of time); the latter is why they thrive during the ongoing pandemic. Now social distancing is no longer a luxury - it's the norm!

However, tree-structured buildings and distancing yourself from others don't exactly go hand-in-hand. Currently there are k students in some of the rooms, and due to a distancing policy there is at most one student per room. What is more, no two students reside in rooms directly connected by a corridor.

The ICPC competition is starting soon, and students rush to take seats at the computers scattered around the department. There are k computers – as many as there are students – located in some of the rooms; moreover, to make distancing possible, no two computers are located in the same room and no two directly connected rooms both have a computer. The students can assign themselves to computers arbitrarily, but they have to maintain social distancing at all times – so getting them to where they should be can be tricky, if not impossible.

You are a ruthless ICPC organizer, and the creator of the ultimate killer problemset. Watching students run around frantically, you realize a horrible truth: if the students don't reach their rooms in time, they will not be able to take part in the competition, and thus all the hard work on preparing unsolvable problems will go to waste! Surely you cannot allow this.

Given the current positions of students and the positions of computers, design a sequence of operations that moves every student to a room with a computer. Every such operation should move a student to an adjacent room; after every operation no two students should be in the same room or in two adjacent rooms. The remaining time before the competition starts permits you to perform at most $4n^2$ moves, where n is the number of rooms. It may as well be that your task is impossible, but there's only one way to find out...

Input

The first line of input contains the number of test cases z $(1 \leq z \leq 100\,000)$. The descriptions of the test cases follow.

The first line of a test case contains a single integer n $(2 \leq n \leq 2\,000)$ – the number of rooms at the department.

The next $n-1$ lines contain two integers u_i, v_i each $(1 \leq u_i \neq v_i \leq n)$ – two rooms connected by a corridor. It is guaranteed that the described corridors form a tree (a connected graph without cycles).

The next line contains a single integer k $(1 \leq k < n)$ – the number of students (and computers).

The next line contains integers $s_1, ..., s_k$ $(1 \leq s_1 < s_2 < ... < s_k \leq n)$ – the initial locations of the students.

The next line contains integers $c_1, ..., c_k$ in a similar format, denoting rooms with computers.

It is guaranteed that there is at least one student located in a room without a computer.

The sum of n^2 over all test cases does not exceed $4 \cdot 10^7$.

Output

For each test case, output "`YES`" (without quotes) if it's possible to move students to rooms with computers while maintaining social distancing, and "`NO`" otherwise. In the former case, in the following lines print any valid solution. The solution description should start with a single integer m $(1 \leq m \leq 4 \cdot n^2)$ denoting the number of moves. Then m lines should follow, each describing a single move with two integers a_i, b_i $(1 \leq a_i \neq b_i \leq n)$, with the meaning that a student who is currently in room a_i should move to room b_i, which is connected with a_i by a corridor.

You don't need to minimize solution length.

Example

standard input	standard output

2
5
1 2
1 3
2 4
2 5
2
1 4
1 5
7
1 2
2 3
2 4
4 6
6 5
6 7
3
1 4 5
3 4 7
	YES
	4
	1 3
	4 2
	2 5
	3 1
	NO

Problem I. GCD vs. XOR

Input file:	*standard input*
Output file:	*standard output*
Time limit:	20 seconds
Memory limit:	512 mebibytes

Optimizing is fun! Especially when it's not exactly required.

Everyone knows that bit operations (e.g. bitwise XOR) are faster than recursive functions (such as the greatest common divisor, GCD). To impress your internship supervisors you replaced, in the company's flagship project, all instances of $gcd(x, y)$ with much quicker $xor(x, y)$.

That was yesterday, on Friday. Now you start thinking whether you should have tested your new code before deploying to production... Well, better late than never. Given a sequence of numbers $a_1, \ldots, a_n$, determine how many pairs (i, j) $(1 \leq i < j \leq n)$ actually satisfy $gcd(a_i, a_j) = xor(a_i, a_j)$. Recall that $gcd(x, y)$ denotes the greatest common divisor of x and y, while $xor(x, y)$ is the bitwise-XOR operation on x and y.

Input

The first line of input contains the number of test cases z $(1 \leq z \leq 20)$. The descriptions of the test cases follow.

The first line of a test case contains an integer n $(1 \leq n \leq 2\,000\,000)$. The second line contains integers $a_1, a_2, \ldots, a_n$, all positive and not exceeding $1\,000\,000$.

The total length of all sequences over all test cases does not exceed $3 \cdot 10^7$.

Output

For each test case output a single integer: the number of pairs (a_i, a_j) with $i < j$ satisfying $gcd(a_i, a_j) = xor(a_i, a_j)$.

Example

standard input	standard output

1
4
2 3 4 3
	2

Problem J. Civilizations

Input file:	*standard input*
Output file:	*standard output*
Time limit:	15 seconds
Memory limit:	512 mebibytes

There is a new hot game in development: *Civilizations* (not to be confused with *Civilization*). As one of the *Senior Developers* on the team, it is your job to write the main game engine.

The world is divided into n rows and n columns of unit fields. The unit field in the i-th row and j-th column is initially owned by civilization $p_{i,j}$ (you can assume that for every integer between 0 and n^2-1 inclusive, there is a civilization corresponding to that integer. Of course, at any given time many of them may not own any fields), and has value $v_{i,j}$, which corresponds to precious resources (or financial liabilities) associated with it.

For a given civilization p, we define two important measures: its *wealth* (w_p) and *length of borders* (l_p). The wealth of a civilization is the total value of the fields it owns, while the length of borders is the number of unordered pairs of fields $\{a, b\}$, such that a and b share a side, and exactly one of them is owned by p.

The game engine will have to handle a sequence of events; in each, the owner of one of the fields changes, as a result of a war between two civilizations. The change of the owner is permanent, at least until next war. After each such event, the engine should determine how powerful is the current *most powerful civilization* (only counting civilizations that own at least one field).

The game design team has already decided that the *power* of civilization p will be computed as $Aw_p + Bl_p + Cw_pl_p$. This is where things get tricky though: the definition of power changes as the situation in the game world develops! After each event, your engine will be supplied with new values for the coefficients A, B and C.

Of course, your engine also has to be fast – otherwise *Civilizations* players will get bored!

Input

The first line of input contains the number of test cases z ($1 \le z \le 5000$). The descriptions of the test cases follow.

The first line of every test case consists of a single integer n ($2 \le n \le 500$) – the size of the world.

The next n lines contain n integers each, and describe field values $v_{i,j}$ ($|v_{i,j}| \le 100$).

The next n lines contain n integers each, and describe initial field owners $p_{i,j}$ ($0 \le p_{i,j} < n^2$).

The next line consists of a single integer q ($1 \le q \le 10^5$) – the number of events.

The next q lines describe the events. Each of them contains six integers: i, j, p, A, B, C ($1 \le i, j \le n$; $0 \le p < n^2$; $|A| \le 10^{10}$; $|B| \le 10^{12}$; $|C| \le 10^4$), corresponding to: the row and column of the field that changes owners, the new owner civilization, and the new coefficients to compute the powers of civilizations, respectively. It is guaranteed that before the event civilization p did not own the field (i, j).

The total number of unit fields in all test cases does not exceed 500 000.

The total number of queries in all test cases does not exceed 200 000.

Output

For each test case output a single line containing q integers: the power value of the most powerful civilization after each of the events.

Example

standard input	standard output

1
2
1 2
3 4
1 1
2 2
6
2 2 1 1 -1 0
1 2 2 1 2 -1
2 1 3 0 1 -1
1 2 3 2 0 0
1 1 3 1 1 1
2 2 3 -1 -1 -1
	5 -7 -2 10 20 -10

Note

After the first event, civilization 2 owns only the $(2, 1)$ field, while civilization 1 owns the rest. Both civilizations have borders of length 2, and their wealth is 7 and 3, respectively. The civilization 1 with power of 5 is the most powerful.

After the second event, civilization 1 owns fields on one diagonal, while civilization 2 on the other. Both civilizations have borders of length 4, and wealth of 5, so they are equally powerful with power of -7.

After the third event, there are now three civilizations on the board: 1, 2 and 3. The civilization 6 is now the most powerful.

Finally, in the last three events, civilization 3 takes over the remaining fields. Note that now 3 is the most powerful civilization for any A, B and C, since we only take into account civilizations controlling at least one field. The power of civilization 3 at the end of the game is -10, since it has borders of length 0, and wealth of 10.

Problem K. We apologize for any inconvenience

Input file:	*standard input*
Output file:	*standard output*
Time limit:	10 seconds
Memory limit:	512 mebibytes

Studying at the Jagiellonian University in Krakow has its pros and cons. Pros: Jagiellonian University. Cons: Krakow... Or, more precisely, a constant necessity to deal with tram track reconstructions.

Initially, the public transport network consists of some number of tram lines. Then one of them gets suspended, then another one, and then another... As you may know yourself, the inviolable rule in Krakow is to always suspend a line before any of the previously suspended lines resumes its operation. Right now not all of the lines have been suspended (yet), and you sit in a tram, annoyed that your direct connection to the university has just disappeared. You look out of the window and you ask yourself: "Am I actually the most unlucky passenger in this city? Or is there someone out there somewhere who needs to change lines even more times in order to get where they want?"

More precisely, we say that stop B is reachable from stop A with c changes if there exist lines $l_0, \ldots, l_c$ such that l_0 serves stop A, l_c serves stop B, and for each $0 \le i < c$ there exists some stop served by l_i and l_{i+1}. At each point in time, you want to know the largest value of c such that there exists a pair of stops (A, B) where B is reachable from A with c changes and B is not reachable from A with c' changes for any $c' < c$.

Note that sometimes it might not be possible to travel between a pair of stops at all. As follows from the definition above, you decide not to take such pairs into consideration in your analysis – you conclude that if someone wishes to travel between those stops, they will take an Uber anyway.

Input

The first line of input contains the number of test cases z ($1 \le z \le 35$). The descriptions of the test cases follow.

The first line of each test case contains the number of stops n and the number of tram lines k ($2 \le n, k \le 750$). The stops are numbered from 1 to n and the lines are numbered from 1 to k.

Then, k lines follow. The i-th of those lines describes the route of the tram line number i. Each line starts with an integer r_i ($2 \le r_i \le n$) followed by r_i distinct integers $a_{i,j}$ ($1 \le a_{i,j} \le n$) – the identifiers of stops served by the i-th tram line. Any tram line runs in both directions.

The next line contains a single integer s ($1 \le s \le k - 1$).

Then, s lines follow, describing the order in which the tram lines get suspended. Each of those lines contains a single integer s_i ($1 \le s_i \le k$) – the identifier of the suspended tram line. Any line can get suspended at most once.

The sums of values of n and k in all test cases do not exceed 1000 each.

Output

For each test case, output $s + 1$ lines, each containing a single integer. The $i + 1$-th line should denote the largest number of line changes necessary after the i-th suspension event (the first line denoting the answer before any suspensions).

Example

standard input	standard output

1
5 4
3 1 3 5
2 1 4
2 2 3
2 2 4
3
1
4
3
	1
	2
	0
	0

Note

Initially, one line change is required to travel, for example, from stop 4 to stop 5 (or vice versa). Such travel is possible by taking line 2, then line 1. There are no pairs of stops requiring 2 or more changes.

After line 1 gets removed, two line changes are required to travel from stop 1 to stop 3 (or vice versa).

When lines 1 and 4 get removed, the only pairs of stops still reachable one from another are (1, 4) and (2, 3), and in both cases no line changes are required to travel between them.

When lines 1, 4 and 3 get removed, the only pair of stops still reachable one from another is (1, 4). No line changes are required to travel between these stops.

Problem L. Patrol Drone

Input file:	*standard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	512 mebibytes

You are trying to steal a very valuable diamond tiara from the museum. What makes it tempting is that (due to budget cuts) all the guards have been replaced by a single automatic drone that patrols the main hall of museum. What makes it less tempting, though, is that this drone is equipped with some very modern weaponry and you are likely to be disintegrated if you attract its attention.

Fortunately, you have done your homework and have a pretty good grasp of how the drone works. Imagine that the hall is an Euclidean plane, divided into unit square cells. The central cell, $(0, 0)$, contains the tiara. The drone has a simple processing unit that stores two things: its current position (x, y) and the sequence of instructions, denoted by letters 'N', 'E', 'S', 'W'. Every minute, the drone moves to an adjacent spot according to the first letter of the sequence (North, South, West or East), changes the stored position accordingly, and then moves this first letter to the end of the sequence to access the next one in the next minute. If the instruction sequence is empty, the drone simply does nothing. It is guaranteed that these instructions describe a closed loop, and that the drone never enters the $(0, 0)$ cell.

Right now, the drone is at position (x_0, y_0) and has a string T of instructions. You can modify the drone's memory by some clever hacking – your goal is to reach a situation in which the drone is at the same position (x_0, y_0), but with different string T'. Your hacking strategy, unfortunately, is somewhat limited – each minute, you can only access the two front letters of the string and perform any number of the following operations:

- remove two front letters from the string, but only if they are "NS", "SN", "EW" or "WE";
- add two letters "NS", "SN", "EW" or "WE" to the front of the string;
- swap two front letters of the string (any combination of letters may be swapped).

Also, there is a collision detection system implemented on the drone, which detects if the current set of instructions would possibly bring the drone to $(0, 0)$. An alarm is raised in this case – this is the situation you want to avoid at all costs.

Find a sequence of hacking operations which will bring the drone to (x_0, y_0) with the desired sequence T' in its memory.

Input

The first line of input consists of a single number z ($1 \leq z \leq 100$), denoting the number of test cases. The descriptions of the test cases follow.

The first line of each test case consists of two integers x_0, y_0 ($-1000 \leq x_0, y_0 \leq 1000$), denoting the initial position of the drone. At least one of numbers x_0, y_0 is not zero.

The second line contains two numbers n, m ($2 \leq n, m \leq 2000$) – the length of the current (T) and target (T') string, respectively.

The next two lines contain one string each, of length n and m respectively, denoting T and T', consisting only of letters N, S, E, and W.

It is guaranteed that the current and target sequences are different. Moreover, both describe some closed loops and neither of these loops cross $(0, 0)$ at any moment of the route.

The total number of letters in all test cases does not exceed 20 000.

Output

For each test case, if it's impossible to meet the task's requirements, output "NO" in a single line. Otherwise, output "YES", and then a solution description on the next line. The solution must consist only of symbols $\{C, N, S, E, W, -, R\}$, where each character indicates a single hacking operation:

- Symbol 'N' means adding "NS" at the front of the string.
- Similarly, symbols 'S', 'E' and 'W' mean adding "SN", "EW" and "WE" at the front.

- Symbol ‘R’ means removing two first letters from the string – this is only allowed if these letters are “NS”, “SN”, “EW” or “WE”.
- Symbol ‘C’ means swapping two first letters.
- Symbol ‘-’ means that you wait for the rest of the minute (until the drone moves to the next instruction).

Note that multiple hacks can be made in a single minute. **You do not need to minimize solution length**, but the actions description must have no more than $2 \cdot 10^7$ characters. By the end of the last minute of your output, the string and the position of the drone must match the desired ones. Removing or swapping elements of a string with at most one letter is not allowed. At no point, the loop described by the drone's sequence can go through $(0, 0)$.

Example

standard input	standard output
2 1 0 10 10 NNWWSSSEEN NWWSSSEENN -1 0 8 8 NEESSWWN SEENNWWS	YES -C-C-R--S-C-C--- NO

Problem M. Social Justice

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

The local election is over. Your town has got a new mayor and you are his most trusted advisor! During the campaign, you have built his popularity on the promise to *bring social justice to the town.* Initially having intended this as a slogan not to be dwelled too much into, you were forced by all those pesky journalists to define its precise meaning in the end. You came up with a constant $K > 1$ and stated that social justice will be achieved when nobody earns more than K times the average pay of the town's residents.

Now the time has come to fulfill that promise. The mayor doesn't really have any reasonable plan on how to enforce social justice without collapsing the economy but, thankfully, he has come up with a much simpler idea. It will suffice to choose a group of citizens whose salaries fit the definition... and banish everyone else. A flawless plan indeed! Those who remain in the town will get to live in a pure, socially just society. Those who get banished... well, they won't get a chance to vote in the next election anyway. Simple and effective – what could possibly go wrong?

Nothing can go wrong, of course, but, for you, things can go even better! The mayor is determined to banish as few people as possible to achieve the goal, but if there is more than one possible way to do it, you will surely be able to influence the choice. Clearly, it won't hurt to talk to the citizens beforehand and find out if some of them have anything interesting to offer in exchange for your protection when the decisions are being made.

Here's the catch, though: if there is no possibility that a given person could be allowed to stay, discussing this with them would be an unnecessary and pointless risk as you couldn't offer them your protection no matter what. A more pragmatic choice will be to make a list of all such citizens – and talk with everyone else.

Input

The first line of input contains the number of test cases z ($1 \le z \le 1000$). The descriptions of the test cases follow.

The first line of every test case contains a single integer n ($1 \le n \le 200\,000$) – the number of the citizens. The citizens are numbered from 1 to n.

The next line contains n integers a_i ($0 \le a_i \le 10^9$) – the citizens' salaries.

The last line contains two integers p and q ($1 \le q < p \le 1000$) which define the constant $K := \frac{p}{q}$.

The total number of citizens in all test cases does not exceed $1\,000\,000$.

Output

For each test case output a line containing an integer c ($0 \le c < n$): the number of people who definitely cannot stay in the town. Then output a single line containing c integers: the identifiers of those citizens in an ascending order.

Example

standard input	standard output

3
4
1 2 3 4
3 2
5
1 15 2 5 1
2 1
5
1 2 3 1000 10000
4 3
	0
	
	1
	2
	2
	4 5

Note

In the first test case, the whole set is not socially just. One can see that for each citizen there exists a socially just set of size 3 containing this citizen. Therefore, someone must get banished, but anyone has a chance not to be this person.

In the second test case, two people must be banished. There are three possibilities: the citizens number 1 and 2 might get banished, or 2 and 4, or 2 and 5. Therefore, it is not possible to build justice with person 2 on board, while every other citizen has a chance to stay.

In the third case, citizens 4 and 5 must clearly get banished – just look at their outrageous salaries!