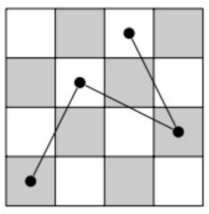
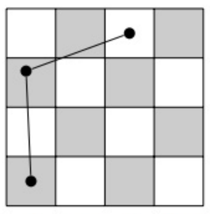
Problem A. Cheating Knight

Source file name:	cheating.c, cheating.cpp, cheating.java
Input:	`Standard`
Output:	`Standard`

The land of the Black King is an infinitely stretching flat surface neatly divided into black and white squares, much like an infinite chessboard. The area of every square is exactly one square metre, and the squares are neatly layed out on a perfect grid. Sir Jumpsalot is a knight who lives in the land of the Black King. By law, knights may only move around by jumping from the centre of a square to the centre of another square, as long as the distance between those centres is exactly $\sqrt{D}$ metres. (In the game of chess, the value of D is fixed at 5, but we will consider other values of D as well.) The value of D fixed in the law can be written as a sum of two squares, for otherwise knights would be unable to move around.

Sir Jumpsalot doesn't like to play by the rules. While he still moves around by jumping $\sqrt{D}$ metres, he doesn't bother landing on the centre of a square every time. In other words, he sometimes lands in a corner of a square or even on the border between adjacent squares, whichever is most convenient. Reaching his destination is usually easier this way, as can be seen from the following example (travelling two squares horizontally and three vertically with the familiar value of $D = 5$). Given that Sir Jumpsalot


a) Route for any normal, law abiding knight.


b) Route for the cheating Sir Jumpsalot.

starts at the centre of the square with coordinates (0, 0) and has to travel to the centre of the square with coordinates (X, Y), your task is to calculate the minimum number of jumps Sir Jumpsalot needs to reach his destination. You may safely assume that there are no obstructions in his way.

Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- One line with three space-separated integers D, X and Y , denoting (the square of) the distance jumped in a single jump and the coordinates of the final destination. These satisfy $1 \leq D \leq 10^8$ and $-10^4 \leq X, Y \leq 10^4$. It is guaranteed that D can be written as the sum of two squares.

Output

For each test case, output one line with a single integer J, the minimum number of jumps needed to reach the destination.

Example

`Input`	`Output`
`5`	`0`
`5 0 0`	`2`
`5 2 3`	`1`
`25 -3 -4`	`10`
`100 0 -100`	`7183`
`1 2345 6789`	

Problem B. Chess Tournament

Source file name:	chess.c, chess.cpp, chess.java
Input:	`Standard`
Output:	`Standard`

There is a large chess tournament being held in your town, and you decide to go and have a look. Two teams A and B each consisting of n players face each other. Every player from team A will play a match against every player from team B. In the end, the team with the most wins will be declared the winner. (In case of a draw, both players receive 1/2 point.)

You don't know which players are on which team, but you have meticulously crafted a list of all matches that have been played during the tournament. The only problem is that some players have played matches outside the tournament schedule, facing a player from their own team or possibly even a player from the opposing team, either for practice or for fun. Can you still tell which players belong to which team?

Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- One line with two space-separated integers N and M, denoting the number of players per team and the total number of matches that have been played. These satisfy $1 \leq N \leq 250$ and $N^2 \leq M \leq 10^6$.
- After that, M lines follow with two different space-separated integers A and B per line, indicating that players A and B have played a match against one another. These satisfy $1 \leq A < B \leq 2N$.

Note that a pair of players may be listed more than once.

Output

For every test case, output two lines:

- One line with the number of solutions. As the number of solutions can be very large, you must reduce your answer modulo 10^8. No answer < 0 or $\geq 10^8$ will be accepted.
- One line with N space separated integers, denoting the team members of the first team in ascending order. The first team is defined to be the team that contains player 1. If this cannot be uniquely

determined, output one possible configuration. There will always be at least one solution (though the number of solutions may nevertheless be 0 modulo 10^8).

Example

Input	Output
3 2 4 1 3 2 4 3 4 1 2 3 21 1 2 1 3 1 3 1 4 1 5 1 6 2 3 2 3 2 4 2 5 2 6 2 6 3 4 3 5 3 5 3 6 4 5 4 5 4 6 5 6 5 6 25 1225 1 2 1 3 ⋮ 48 50 49 50	1 1 4 10 1 2 3 3218876 1 2 3 4 5 [...] 22 23 24 25

For the third testcase, both the input and the output have been abbreviated due to space constraints that are beyond our control. In the input, every player plays exactly one match against every other player. In the output, we chose players 1 through 25 to form the first team, but the solution is far from unique. In fact, any list of 25 different players is a valid answer, as long as you make sure that

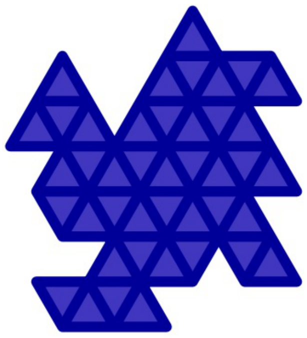
- the list contains player 1, and
- the players are listed in ascending order.

You will find the test case in its entirety in your home directory.

Problem C. Colby's Costly Collectibles

Source file name:	colby.c, colby.cpp, colby.java
Input:	`Standard`
Output:	`Standard`

At the Colby's Costly Collectibles workshop, special jewels are manufactured. In his production process Colby first creates a large grid of stones, each of which has the shape of an equilateral triangle. From this grid he cuts out all kinds of shapes which he sells as pieces of jewellery. Customers can choose their preferred shape according to the following basic rules:



1. Cuts can only be made along the edges of the triangles.
2. The jewel must consist of one piece. A connection at a vertex is too weak, so the triangles must be connected by their edges.
3. The jewel cannot contain any holes.

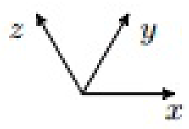
To illustrate the production process, a typical jewel is depicted in the figure on the right. This corresponds with the third test case in the samples below.

Since the customer has to pay per triangle, Colby has asked you to help him calculate the number of triangles used. You are given a description of the jewel's outer boundary. Note that it follows from rules 2 and 3 that the boundary will never intersect or touch itself.

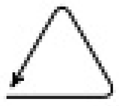
Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- One line with a single integer C, the number of cuts made. This number satisfies $3 \leq C \leq 100$.
- Then C lines follow, describing the boundary of the jewel. Each line starts with a single letter x, y or z, denoting the direction in which to move, followed by an integer denoting the number of steps. The directions correspond with the axes depicted below:



To further illustrate this, the first test case below describes a single triangle, starting from the lower left corner:



(First take one step in the x direction, then one step in the z direction and finally one step in the negative y direction.)

The boundary will never touch itself. It will always end up where it started and the total length of the boundary will not exceed 1000. The path starts at a vertex and the endpoint of every segment will again be a vertex. (In other words, no two consecutive edges in the input will be in the same direction, not even the first and the last edge.)

Output

For each test case, output one line with a single integer N, the number of triangles.

Example

Input	Output
`3`	`1`
`3`	`110889`
`x 1`	`42`
`z 1`	
`y -1`	
`3`	
`y 333`	
`x -333`	
`z -333`	
`23`	
`y -3`	
`z 2`	
`y -2`	
`x 1`	
`y -1`	
`z -1`	
`x 1`	
`y -1`	
`x -1`	
`z -1`	
`x 2`	
`z 1`	
`x 1`	
`y 1`	
`z -1`	
`x 1`	
`z 2`	
`x 1`	
`z 2`	
`x 1`	
`z 1`	
`x -1`	
`z 1`	

Problem D. Ga

Source file name: ga.c, ga.cpp, ga.java
Input: `Standard`
Output: `Standard`

The Dutch version of the game "Go" is called "Ga". It is played on an $N \times N$ board; the N^2 squares are initially empty. Two players, White and Black, take turns; White begins. During a move, the player places one or more stones of his own colour (one by one) on previously empty squares, one per square. The first player that cannot move, loses.

We say that two squares are adjacent if they directly touch each other in a horizontal, vertical or diagonal way. A player must always place his stones on squares that are adjacent to squares that are already occupied by a stone from his own colour. (The very first stone of the first move of a player may be placed at will.)

The number of stones a player is allowed to put on the board during his move is usually determined by local game rules. E.g., in Volendam this number is determined by throwing dice. Specialized shops sell beautifully carved sets of dice (called "gabbers") with different numbers of sides. Anyway, it is important to have an upper bound on the number of stones a player can put on the board during his move.

We want to have a program that determines how many stones the White player at most can put on the board during his move.

Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- A line containing one integer N (with $2 \leq N \leq 100$), the height (equal to the width) of the board.
- Then N lines, each containing N characters, describing the state of the gameboard. The characters are: -, w and b, representing an empty square, a White stone and a Black stone, respectively.

Every gameboard contains at least one White stone and at least one Black stone. The White stones so far have been placed according to the rules, and so have the Black ones.

Output

For each test case, output one line containing a single integer: the maximum number of stones that White can put on the given board in one move, according to the rules of "Ga".

Example

Input	Output
2 3 bbb bwb bbb 6 --b--- bbww-- -bbbw- ---bbb ---b-- ------	0 8

Problem E. Icelandic Motorclubs

Source file name:	icelandic.c, icelandic.cpp, icelandic.java
Input:	`Standard`
Output:	`Standard`

The country of Iceland has a huge number of beautiful sights, including waterfalls, lava fields, cliffs, geysirs, valleys, volcanoes, glaciers, whales, seals, sheep and much more. Recently, the country has attracted the attention of motor clubs. Expelled from their home countries as a result of revealing reality TV programs, they now seek to bike elsewhere. Members of motor clubs have a lot of money (how else would they be able to pay for all those fancy motor bikes), and thus also own private planes that can drop them with a parachute together with their motor cycle anywhere they want.

The country of Iceland has many rough roads that are not suitable for motor driving, but luckily Iceland has one accessible paved ring road (Road 1) that makes a circle, passing by all the beautiful sights an adventurous motor driver may be interested in. However, Iceland is sparsely populated, and there are not so many gas stations. In fact, in rough winter times, there are barely enough gas stations supplied with gas to make a full round trip with one motor cycle. In the winter, only the inner clockwise lane (Iceland drives right) is free of snow.

Nevertheless, a biker is brave and will try to make a round trip. Luckily, bikers know how much gas is at each gas station and know the distance between each subsequent pair of gas stations on the ring road of Iceland. Also, bikers do not have to worry about the size of the gas tank on the motor bike: it is assumed to be of infinite size. Also, filled gas tanks collapse under air pressure, so the biker jumps out of the plane with an empty tank.

Given the distance between each subsequent pair of gas stations and the amount of gas available at each gas station, at which gas station should the biker start in order to complete a full clockwise circle without running out of gas? Assume that motor bikes are not the most eco-friendly vehicles: they use one liter of gas to drive one mile.

Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- One line containing an integer $3 \leq N \leq 1000000$ denoting the number of gas stations on the circular road (assume stations are numbered from 1 to N).
- N lines, each describing a gas station using two space-separated integers G and D (with $0 \leq G \leq 512$ and $1 \leq D \leq 512$), representing the amount of gas available at this gas station and the distance to the next gas station in miles. Because it is a circular road, the final gas station N lists the distance to gas station 1.

Output

For each test case, output one line containing an integer representing the number of a gas station where the biker could start his round trip. If there are multiple solutions, output one. If the round trip cannot be made, output one line containing the string `IMPOSSIBLE` instead.

Example

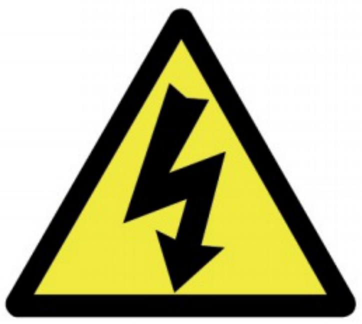
Input	Output

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

Problem F. Manhattan Power Failure

Source file name:	powerfailure.c, powerfailure.cpp, powerfailure.java
Input:	`Standard`
Output:	`Standard`

There has been a huge storm ravaging through Manhattan destroying many power-lines and leaving entire blocks without power. The first damage assessment came up with a report showing which blocks are still connected by power lines and which are not. When a block is connected to another block with power lines, this means that if one block has power, then the other will also have power. Only blocks that are adjacent (either horizontally, or vertically) may be connected by a power line. Also, there is a quickly made list with all blocks that have power generators, or that are connected to external power sources.



Your task is to quickly identify where to put up emergency power lines, so that all block of this grid-like city have power again.

Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- One line with four integers, n, m, p, c, where n, m $(1 \leq n, m \leq 100)$ are the number of blocks that the city is wide respectively long (the city has $n \times m$ blocks), p $(1 \leq p \leq n \cdot m)$ is the number of power generators, and c $(1 \leq c \leq 2 \cdot n \cdot m$, or in fact: $1 \leq c \leq 2 \cdot n \cdot m - n - m)$ is the number of intact power lines between adjacent city blocks.
- p lines, each with two integers x $(0 \leq x < n)$, y $(0 \leq y < m)$ indicating that the block with coordinates (x, y) in the grid has its own power source or is connected to an external power source.
- c lines, each with two integers x $(0 \leq x < n)$, y $(0 \leq y < m)$, and a character d $(d =$ 'R' or $d =$ 'U') indicating that there is an intact power line either between block (x, y) and block $(x + 1, y)$ if $d =$ 'R', or between block (x, y) and block $(x, y + 1)$ if $d =$ 'U'. Of course, if $x = n - 1$, then d cannot be 'R'. Likewise, if $y = m - 1$, then d cannot be 'U'.

Output

Per test case, output one line with one integer indicating the number of emergency power lines needed to connect all blocks to a power source.

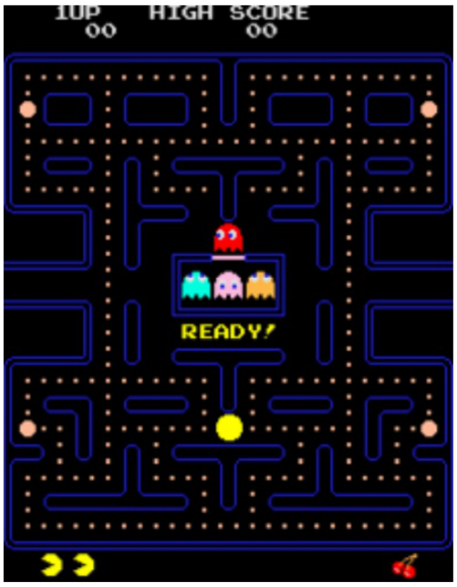
Example

Input	Output
2 2 3 2 3 0 2 1 1 0 0 U 0 2 R 1 1 U 2 4 2 6 0 3 1 0 0 0 U 0 1 U 0 1 R 0 2 R 0 3 R 1 1 U	2 1

Problem G. Pac-Man

Source file name:	pacman.c, pacman.cpp, pacman.java
Input:	`Standard`
Output:	`Standard`

While playing some old-school Pac-Man game, you suddenly discover that Pac-Man has magically doubled itself. There are two Pac-Man characters in the maze, at different positions, and both respond to your single joystick. When you move the joystick north, they both go north, when you move it east, they both go east, et cetera. However, the characters cannot move into (or through) a wall. If there is a wall to the north of a Pac-Man character, and you move the joystick north, then this particular character does not move.



On the one hand, it is beneficial to have two PacMan characters: they can eat two pac-dots in one move. On the other hand, you get terribly confused from looking at two characters all the time. And it is much harder to stay away from the ghosts (the enemies in the game) with two characters than with one. If any character bumps into a ghost (or the other way round), you lose a life. Because you have only five lives, and the second character did not get you an additional life, you decide it would be best if you could move the two Pac-Man characters onto the same position. But is that possible?

Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- A line containing two space-separated integers M and N satisfying $2 \leq M, N \leq 50$: the dimensions of the maze.
- M lines describing the maze, each containing N characters from $\{P, X, G, .\}$, where
  - P indicates a Pac-Man character,
  - X indicates a wall,
  - G indicates a ghost,
  - **.** indicates an empty position.

  There are exactly two occurrences of P in the maze.

Output

For each test case, output a single line containing a positive integer D, a space and a sequence of D characters from $\{N, E, S, W\}$ (North, East, South, West), indicating a shortest sequence of joystick moves that brings the two Pac-Man characters at the same position, without bumping into a ghost with either of the characters. If there are multiple solutions, output one. If there is no such sequence of moves at all, then output the string `IMPOSSIBLE`.

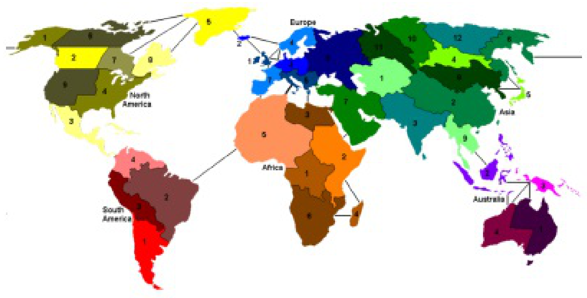
For simplicity, we assume that the ghosts in the maze do not move. We further assume that a Pac-Man character can step in one move from one edge of the maze to the corresponding position at the opposite edge of the maze.

Example

Input	Output
3	7 WSEESEE
2 5	10 EEESSWWWSS
.P...	IMPOSSIBLE
XG.P.	
8 8	
X...X.X.	
X.......	
.XXP...X	
..X..X..	
.PXXXX..	
.......X	
....	
XXXXXXX.	
2 2	
P.	
GP	

Problem H. Risk

Source file name:	risk.c, risk.cpp, risk.java
Input:	`Standard`
Output:	`Standard`



Good old friends Vladimir and Barack like to play games together. One of their favourites is the strategy board game Risk, where players occupy certain regions of the world, and are to conquer all regions of specific continents. Once in a while, Vladimir challenges the little boys in the neighbourhood for a game. This time, he asked Mark, who immediately said yes, excited that he was invited to play with one of the big boys.

Only later, Mark realized that Risk is a serious game and that Vladimir is a much more experienced player than he is. In order to have at least a little chance, Mark ask you for your advice on a specific aspect of the game. In particular, can you tell him what the probability is that Mark, occupying a certain region with M armies, wins the battle if he decides to attack an adjacent region occupied by Vladimir with N armies?

Rules for a battle

We consider the standard rules of Risk, with one exception: the dice used do not necessarily have six sides. The dice may have any number of sides $D \geq 1$, all sides having different values and equal probability.

During a battle, the two players take turns in rolling the dice, until one of them has won. Suppose that the attacking player has A armies left in the region from which he is attacking, and that the defender has B armies in his region. Then first, the attacker rolls $\min\{3, A-1\}$ dice. That is, in principle, he uses three dice. However, if $A < 4$, he uses $A-1$ dice, one die for each army he has minus 1, because one army must remain in the region and is not involved in the attack. Now, if $B = 1$, the defender rolls one die. If $B \geq 2$, the defender chooses to roll one or two dice, so as to optimize his probability to win the battle.

Each player's highest die is compared, as is their second-highest die (if both players roll more than one). In each comparison, the highest number wins. The defender wins in the event of a tie. With each dice comparison, the loser removes one army from his region from the game board. Any extra dice are disregarded and do not affect the results.

For example, suppose that the attackers rolls three dice, yielding 4, 2 and 1, and that the defender rolls two dice, yielding 3 and 2. Then the first comparison (between 4 and 3) is won by the attacker, the second comparison (between 2 and 2) is won by the defender. As a result, both players lose one army. If, as another example, the attacker has two dice showing value 6, then it may be wise for the defender to roll only one die, so he loses at most one army.

The battle is over, when either the attacker has only one army left (the defender has won) or the defender has no more army left (the attacker has won).

Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- One line containing an integer D satisfying $1 \leq D \leq 20$: the number of sides of the dice in the game.
- One line containing two space-separated integers M and N satisfying $2 \leq M \leq 100$ and $1 \leq N \leq 100$: the number of armies at Mark's region, and the number of armies at Vladimir's region, respectively.

Output

For each test case, output one line with a single real value: the probability that Mark wins the battle. An absolute precision error of 0.0001 is allowed. Scientific notation is also allowed.

Example

Input	Output

3
6
10 10
2
7 3
15
85 98
	0.363205
	0.322393
	0.734746

Problem I. Rummikub

Source file name:	rummikub.c, rummikub.cpp, rummikub.java
Input:	`Standard`
Output:	`Standard`

Rummikub is a simple game, often played by elderly people versus their grandchildren. The tile set consists of four suits, represented by the colors blue, green, red and yellow. Each suit consists of $N > 0$ different tiles, numbered from 1 to N. The number of a tile represents the *value* of that tile. There are two tiles of each (value, color) combination.

At the start of the game, the referee deals each player a hand of tiles. These are only visible to them. All other tiles that are not dealt to any player are placed face down on the table, and form the pool. The players take turns in playing tiles. Any tile that is played must be an element of either a group or a run. A *group* is defined as a set of at least three tiles of the same value but a different suit. A *run* is defined as a set of at least three tiles of the same suit but with consecutive values. If a player cannot play any tiles, the player must take a tile from the pool.

The elderly people like progress a lot. They love to play lots of tiles each turn, and they hate it when they have to pick a tile from the pool, as this retains progress. Sometimes, the elderly people accidentally take a tile from the pool, because they overlook a run or group in their hand. You have been asked to solve this problem for them. Given a hand of tiles, write a program that determines whether they can create a new run or group.

Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- One line with an integer M, satisfying $1 \leq M \leq 800$: the number of tiles the player has in his hand.
- One line with M space-separated strings, representing the tiles in the hand. Each tile string consists of an integer V satisfying $1 \leq V \leq 100$ (the value), and a character which is either b, g, r or y (representing the suit).

Output

For each test case, output a single line containing the string YES if the player can form a run or a group; NO otherwise.

Example

Input	Output

`2`
`3`
`1r 2r 3r`
`4`
`1r 1y 2g 3g`
	`YES`
	`NO`

Problem J. Song Titles

Source file name:	songtitles.c, songtitles.cpp, songtitles.java
Input:	`Standard`
Output:	`Standard`

Somewhere on the outskirts of Leiden's music scene, there is an obscure instrumental rock group consisting of students from Leiden University. They call themselves *naaagrm*, which is an anagram of an existing word in the Swedish language. All of their song titles are anagrams of existings words as well. However, many people seem to struggle with the pronunciation of the band name, as they don't know how to properly pronunciate the triple a (in case you were wondering, it is pronounced as what linguists like to call an *open front unrounded vowel*, and a very long one at that). Therefore the band members have decided to modify all their song titles by changing the order of the letters in such a way that no two consecutive letters are the same. Since this is a tedious task, they ask for your help.

Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- One line containing a single string, the current name of the song. The string consists of lowercase letters only, and has length between 1 and 10^5 inclusive.

Output

For each test case, output an anagram of the input string such that no two consecutive letters are the same. If there are multiple ways to do this, output the lexicographically smallest one. If it is impossible to do so, output `IMPOSSIBLE` instead.

Example

Input	Output
5 naaagrm hiking snaaan banana aaaaaaargh	agamanr ghikin ananas abanan IMPOSSIBLE

Problem K. Tanks

Source file name:	tanks.c, tanks.cpp, tanks.java
Input:	`Standard`
Output:	`Standard`

The army has given you an assignment: given a model of a tank hull and a hit by a shot on that tank, which crucial components of that tank are pierced by the shot?

The tank hull is modelled by a convex three-dimensional polygon. All crucial components are modelled by axis-aligned boxes. Crucial components are disjoint and all fully inside the tank. The components and the tank hull also do not touch each other anywhere. The tank shot is specified by an origin vector, a direction vector and an initial speed. The origin vector always lies strictly outside of the tank hull.

All faces of the tank and crucial components have a resistance and a deflection threshold. The tank shot does not lose speed except when it hits a face, at which point the speed is reduced by the resistance of that face. After reduction, if the speed is at most the deflection threshold, the shot ricochets, deflecting from the face. We assume these ricochets are perfect, in that the angle of reflection is identical to the angle of incidence. A crucial component has the same resistance and deflection threshold for all six of its faces. Note that it is possible for a shot to ricochet more than once. If the shot does not ricochet, it continues in the same direction, potentially hitting another face after.

The shot disintegrates if its speed has been reduced to 0, or after it has pierced or deflected off of D faces. When the shot disintegrates, it no longer moves to pierce or deflect off of any other face. If a shot disintegrates when it reaches the limit of D face hits, it disintegrates at the position it hits that last face. A crucial component only counts as pierced if the tank shot comes strictly inside of it (so deflections off crucial components don't count).

The army has told you that they will be happy with your software if it correctly deals with all cases in which the tank shot never comes within 10 centimeters of any edge or vertex of a face of the tank or a crucial component - your software can return any answer it wants if this does happen to be the case. You can therefore also assume it never happens that a tank shot goes through any face while moving parallel to that face (as you'd need different modelling for that anyway).

Given the tank hull, the crucial components, D, and the specification of the tank shot, can you list which crucial components end up being hit by the shot, and give the position where the tank shot ends up disintegrating?

Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- A line containing two integers F and C, the number of faces that the tank hull consists of, and the number of critical components ($4 \leq F \leq 1000$, $0 \leq C \leq 100$).
- Then a line containing two integers D and S, the maximum number of faces the tank shot will pierce of deflect off from, and the initial speed of the tank shot ($1 \leq D \leq 100$, $1 \leq S \leq 1000000$).
- Then a line containing six integers (a, b, c) and (d, e, f), with (a, b, c) the origin position of the tank shot and (d, e, f) the direction vector of the tank shot ($-1000000 \leq a, b, c, d, e, f \leq 1000000$).
- Then F times the following:

  - A line containing three integers v, r and d, the number of vertices v on that face, the amount that the face reduces speed r and the deflection threshold of that face d ($0 \leq r, d \leq 1000000$, $3 \leq v \leq F$).
  - Then v lines containing three integers (x, y, z) each, the coordinates for every vertex making up that face in order ($-1000000 \leq x, y, z \leq 1000000$ - you are given that no three successive vertices of a face lie on the same line).

- Then C lines, containing eight integers (x, y, z), (x_s, y_s, z_s) and r, d each, with (x, y, z) the center of the crucial component, (x_s, y_s, z_s) sizes of the crucial components, r the speed reduction of every face of this component and d the deflection threshold of every face of this component ($-1000000 \leq x, y, z \leq 1000000$, $1 \leq x_s, y_s, z_s \leq 1000000$, $0 \leq r, d \leq 1000000$).

Integers on the same input line are separated by single spaces.

Output

For each test case, output:

- A line with the number C_p of crucial components pierced by the tank shot.
- Then C_p lines each with a single integer: the (0-based) index of the crucial components that have been hit, in increasing order.
- Then another line, with three space-separated integers (x, y, z): the position at which the tank shot ended up disintegrating, coordinates rounded down to the nearest integer (assume that these coordinates are either exactly integers because they appeared in the input, or are not close enough to an integer to cause rouding issues). If the tank shot never disintegrated, output a line with `SHOT NEVER DISINTEGRATED` instead of this final line.

Example

Input	Output
1	1
6 2	1
10 100	2 0 0
-20 0 0 1 0 0	
4 50 30	
10 10 10	
10 10 -10	
10 -10 -10	
10 -10 10	
4 50 30	
-10 10 10	
-10 10 -10	
-10 -10 -10	
-10 -10 10	
4 50 30	
10 10 10	
10 10 -10	
-10 10 -10	
-10 10 10	
4 50 30	
10 -10 10	
10 -10 -10	
-10 -10 -10	
-10 -10 10	
4 50 30	
10 10 10	
-10 10 10	
-10 -10 10	
10 -10 10	
4 50 30	
10 10 -10	
-10 10 -10	
-10 -10 -10	
10 -10 -10	
3 0 0 2 2 2 10 20	
-3 0 0 2 2 2 10 20	