Problem A. Freight Train

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

The chemical company NS (Nasty Substances) has three factories: one in the Netherlands, one in Belgium and one in Luxembourg. Chemicals are moved between the factories on a regular basis, which is always done by freight train. Last night, the weekly shipment of chemicals was again sent from the factory in the Netherlands to the factory in Belgium. However, something has gone wrong: some of the chemicals that arrived in Belgium were supposed to go to the factory in Luxembourg. What's more, the Luxumbourg chapter is eagerly awaiting the arrival of its chemicals, as any delay in delivery causes major issues in the production pipeline.

Picture by: Dutch Densha

In order to make sure that the error can be corrected as quickly as possible, an additional $L-1$ locomotives are sent towards the freight train, which is currently stationed at the Belgian factory. (Thus, in total there are now L locomotives available.) Each locomotive can pick up some initial segment of the train (that is, the first K wagons) and transport it either back to the Netherlands or on towards Luxembourg. There is no time to do more elaborate rearrangements of the freight train. Since time is of the essence, you want the trains going to Luxembourg to be as short as possible: shorter trains can move faster. Trains heading back to Netherlands can be as long as you want. You are given a list of the wagons that need to go to Luxembourg. All the other wagons are empty and can either be carried along to the Luxembourg plant as well, or be driven back to the Netherlands. No wagon can be left behind. Your task is to split the freight train into (at most) L consecutive trains so that the largest train heading for Luxembourg is as short as possible.

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 N, W and L, satisfying $1 \leq N \leq 10^9$ and $1 \leq W, L \leq 10000$ and $W \leq N$. These denote the number of wagons of the freight train stationed in Belgium, the number of wagons still containing freight and the number of locomotives.
- One line with W space-separated integers, denoting the numbers of the wagons that still contain freight in ascending order. Wagons are numbered 1 through N.

Output

For each test case, output one line with a single integer, denoting the number of wagons of the longest train heading for Luxembourg.

Example

Input	Output

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

In the first test case, you take the first two wagons and send them towards Luxembourg. The remaining wagons can go back to the Netherlands.

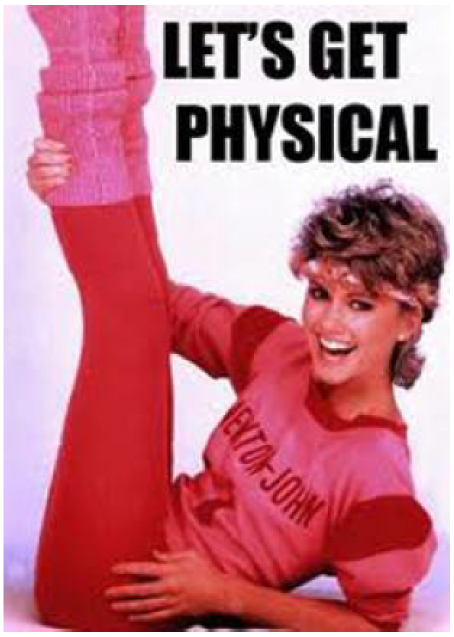
In the second test case, your best option is to simply split the train in three parts, each of which heads for Luxembourg.

In the third test case, split the train into three parts of two wagons each. Two of those are sent towards Luxembourg. Note that one of the locomotives remains unused.

Problem B. Physical Music

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

The music business is changing rapidly, and this is reflected by the single charts. Initially, the Dutch Single Top 100 was based purely on sale numbers of CD singles. In the course of time, however, these numbers dropped dramatically, in favour of legal and illegal downloads of music from the internet. Therefore, since 2006, downloads of singles from specific legal downloads sites are incorporated into the chart. Nowadays, even streams from certain streaming platforms contribute to the Single Top 100.



Between 2006 and 2013, when the Single Top 100 was based on both CD singles and downloads, there was also a chart called the Download Top 100, based purely on the downloads. Assuming that both charts used the same numbers of weekly downloads, one could tell from a comparison between the two charts, which singles were selling well physically. And in fact, since some singles did not even appear as CD singles any more, one could tell which singles were *certainly* available as CD single: the ones that were doing better (as compared to some other singles) in the Single Top 100 than in the Download Top 100.

Now, you are asked to write a program to determine these singles. To be prepared for other communities, which may also have single charts of other sizes, your program should not be limited to charts containing exactly one hundred singles.

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 N, satisfying $1 \leq N \leq 100000$, the size of both single charts.
- N lines, each containing a single integer, which together form a permutation of the numbers 1, . . ., N. The i-th integer P_i is the position in the Download Top N of the single that is ranked i in the Single Top N.

As the above specification implies, the Download Top N and the Single Top N contain exactly the same N singles, possibly in different orders.

Output

For each test case, output:

- One line containing the number of singles M that are certainly available as CD single.
- M lines, each containing a single integer, the *positions* of the singles in the Download Top N that are certainly available as CD single, in ascending order.

Example

Input	Output
2 3 1 2 3 4 4 3 1 2	0 2 3 4

Problem C. Godzilla

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

Godzilla is rampaging through Tokyo again. Luckily, the Superior Defender Mech Force (SDMF) has sent its mech units to stop Godzilla's attack. Mech units are gigantic bipedal robots, usually piloted by Japanese teenagers, that carry weapons of superior destruction. The weapons are so powerful that one hit should neutralize the Godzilla threat.

The SDMF faces two challenges. First, the mech units are so big that they cannot walk over any residential sectors, as they would then trample the people of Tokyo. Second, the weapons of the mech units are so powerful that the pilots cannot afford to fire them at Godzilla while there are any residential sectors between the firing mech and Godzilla.

The SDMF wants to run some simulations before it faces the Godzilla threat. The simulations are based on a two-dimensional map of the area of Tokyo where Godzilla is running amock. The passing of time happens between rounds, where in each round there are two turns. In the first turn, Godzilla gets to do a move. In the second turn, the mech units are allowed a move. In a single move, Godzilla moves one sector on the map in the directions North, East, South or West.

Godzilla is lacking in brain matter and has a very predictable movement scheme. First, Godzilla does not move into a sector that he already visited. Each round, on the first turn:

- It looks for any houses to destroy that are adjacent to him. If the sector adjacent to him is a residential sector, it will move to that sector and destroy the sector. When a residential sector is destroyed, it becomes a *ruined sector* instead. Godzilla looks around him in the order North, East, South, West. So, it will first look to the North to see whether there is a residential sector. If so, it will move there. Otherwise, he will look East to see if there is a residential sector to destroy, and so on.

- In the event that there are no residential sectors adjacent to Godzilla, he will try to move North. If that move would move him outside the map or into a sector he already visited, he will instead try to move East. If he cannot move East, he will move South. In the case that he cannot move South, he will try to move West instead. The terrifying presence of a mech will not stop Godzilla from moving into the sector that is occupied by a mech unit. In that case, Godzilla is considered to be in range of the mech unit's weapons of superior destruction, in the next turn.

- If Godzilla has no move options left, he will instead start roaring and wail his arms, inflicting no damage to Tokyo city.

Each round, on the second turn, each mech unit can either stay at its position, or move to an adjacent non-residential sector, or a ruined sector. It cannot move outside the map. When moving, a mech unit moves one sector in one of the directions North, East, South or West. It is allowed to move into the space of another mech unit. At the moment that a mech unit is able to fire its weapons at Godzilla, it will do so on the mech units' turn. Mech units can move and fire in the same turn. A mech can fire its weapons at Godzilla if there is a straight horizontal or vertical line between the mech and Godzilla, and the line is not obstructed by any residential sectors.

Given a map of the Tokyo area where Godzilla is rampaging and the starting location of the mechs, determine the minimum number of residential sectors that will be destroyed before Godzilla can be neutralized by a mech unit.

Input

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

- One line containing two space-separated integers L ($3 \leq L \leq 1000$) and W ($3 \leq W \leq 1000$), the length and width of the map of Tokyo in sectors, respectively.
- W lines, containing L characters, describing the map of Tokyo. The characters are:
  - '.': a non-residential sector;
  - 'R': a residential sector;
  - 'G': the starting position of Godzilla. This is always a non-residential sector;
  - 'M': the starting position of a mech unit. This is always a non-residential sector.

The sector at the top-left corner of the map represents the North-West corner of Tokyo, while the sector at the bottom-right corner of the map represents the South-East corner of Tokyo.

The simulation will contain at most 100 mech units.

It will always be possible for Godzilla to be neutralized by a mech unit.

Output

For each test case, output one line containing a single integer: the minimum number of residential sectors that will be destroyed before Godzilla can be neutralized by a mech unit.

Example

Input	Output
`2`	`1`
`3 3`	`2`
`RR.`	
`G..`	
`M.R`	
`7 5`	
`M...RR.`	
`...G...`	
`...RRR.`	
`.......`	
`..RR..M`	

Problem D. Hotels

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

The beautiful city of Leiden draws more and more tourists every year. Opposite the famous Hilbert Hotel, located near the city border, a competing chain is building a hotel in a project nicknamed *Lodgings In A Colossal Skyscraper* (LIACS). As you may know, the Hilbert Hotel has an infinite number of rooms. Guests at the Hilbert have been complaining that it takes them forever to reach their room, so the competing hotel chain decided that LIACS will only have finitely many rooms. Nevertheless, LIACS will become a colossal skyscraper consisting of many many floors, precisely F of them.

The floors will be connected by both elevators and stairways. Fire regulations demand that it must be possible to reach the ground floor using only the stairs, so there is a stairway going from the top floor all the way down to the ground floor (with a door on every floor in between). On the other hand, the elevators can only pick up people on so many floors before they are full, so the project manager decided that no elevator should stop on every floor. More precisely, the i-th elevator will only stop on floors for which the number is equivalent to R_i modulo M_i.

The guests don't mind taking the elevator or changing elevators many times in a row, but they don't like taking the stairs. The LIACS project manager has no idea whether his planned elevators provide acceptable coverage, so he asked you to calculate how many ights of stairs his guests have to take in order to get to their room, assuming that their room is situated on the worst possible floor. Guests always enter the hotel on the ground floor. Note that guests may alternately take the stairs and take the elevators in order to reach their room, as not every elevator stops at the ground floor. (See sample input below.)

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 F and E, denoting the number of floors and elevators, respectively. These satisfy $2 \le F \le 10^9$ and $0 \le E \le 100$. The floors are numbered from 0 to $F-1$, where 0 is the ground floor and $F-1$ is the top floor. (This is quite common in the Netherlands, whereas in other countries the ground floor is sometimes numbered 1 instead of 0.)
- E lines, each containing two space-separated integers R_i and M_i satisfying $2 \le M_i \le F$ and $0 \le R_i < M_i$. These denote the remainder and the modulus for every elevator. It is guaranteed that every elevator stops on at least 1 and at most 1000 floors.

Output

For each test case, output one line with two integers S and W separated by a single space, where

- S is the number of stairs a guest has to take if his room is located on the worst possible floor,
- W is the number of the worst possible floor. If multiple floors are equally bad, output the one closest to the ground floor.

Example

Input	Output
5	1 1
21 3	2 20
0 3	2 3
1 3	9 9
2 3	0 0
21 2	
0 3	
1 3	
15 2	
2 3	
1 4	
10 0	
99 3	
0 2	
1 2	
0 3	

In the second test case, the top floor is numbered 20. The closest elevator is the one on the nineteenth floor, but this one doesn't stop at the ground floor. Thus, a guest sleeping at the twentieth floor needs to walk at least two flights of stairs. (On the other hand, in the first test case one can take the elevator from the ground floor to the third floor, then walk to the second floor and take the elevator to the twentieth floor.)

Problem E. The King's Walk

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

Chess is a game in which two sides control pieces in an attempt to capture each other's king. The pieces vary in mobility. At the beginning of a game the kings are rather vulnerable. They are less mobile than most other pieces and they tend to hide behind their pawns. Like in real life, as soon as both queens have left the game it is time for the kings to come into action. Because there is little threat left for the king, he can now move safely around the board. Indeed his mobility seems to be quite strong at this stage making him one of the more dangerous pieces. Your task is to measure the mobility of the king in the endgame.

Consider a chess board of $N \times N$ squares. The king is the only piece on the board. He starts at a given position and wants to go to another given position in the minimum number of moves. The king can move to any adjacent square in any orthogonal or diagonal direction.

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 N, the size of the board, where $2 \leq N \leq 5000$.
- One line with four space-separated integers X_1, Y_1, X_2, Y_2, such that $1 \leq X_1,\ Y_1,\ X_2,\ Y_2 \leq N$, where (X_1, Y_1) is the square on which the king starts and (X_2, Y_2) is the square the king wants to go to (different from his starting position).

Output

For each test case, output one line with a single integer: the number of ways by which the king can reach the destination square in the minimum number of moves. As this number can be very large, you must reduce your answer modulo 5318008.

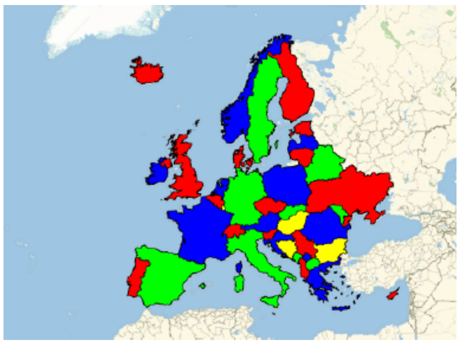
Example

Input	Output

2
3
1 2 3 2
8
2 2 7 7
	3
	1

Problem F. Map Colouring

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



A map maker wants to colour different countries or provinces on his maps using different colours. That is, he wants to colour countries that share a common border with a different colour. He has heard that any map can be coloured with four colours, but even after a lot of puzzling he is unable to colour some of his maps using four colours. Interested in the problem, and hoping that he can publish four-coloured maps, the map maker visits you for help.

After inspecting his maps you explain to the map maker that not every map can be coloured with four colours. You tell him that when countries are made up of different non- connected parts (such as Alaska and the rest of the United States), it can be the case that one needs more colours. Also, if the maps contains a quadripoint (a point where four countries touch), it can also be the case that one needs more colours. The same is true when more than four countries meet in one point.

Disappointed, the map maker asks you whether you cannot help him in quickly deciding how many colours he needs. While you try to explain that this is a quite difficult question, he shows you his maps. After a lot of persuasion by him, and the realisation that all his maps are actually very small, you agree to make a computer program for him. You agree to write a program for his small maps (at most 16 countries) that determines whether his map can be coloured with 1, 2, 3, 4 or more colours. This program should still work if there are non-connected countries, quadripoints, etc.

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 C and B, indicating the number of countries C ($1 \leq C \leq 16$) and the number of borders B ($0 \leq B \leq 120$).
- B lines with two space-separated integers i and j ($0 \leq i,\ j \leq C - 1$ and $i \neq j$), indicating that there is a border between country i and country j.

Output

For each test case, output a single line containing either an integer K, or the string ‘many’. Your program should output the integer K, where K is the minimum number of colours needed to colour the map if this number is 1, 2, 3 or 4. Your program should output ‘many’ if this number is larger.

Example

Input	Output

3
4 0
4 4
0 1
1 2
2 3
3 0
5 10
0 1
0 2
0 3
0 4
1 2
1 3
1 4
2 3
2 4
3 4
	1
	2
	many

Problem G. Mario

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

In a final attempt to complete an impossible Super Mario level, you decided to write an AI and let it play the game for you.

As a first step, we forget about badguys and only implement navigation. In this problem, you will implement navigation over a river. The river has width W and occupies $x = [0, W]$. Transportation over water happens using boats. Each boat moves in a range $[L, R]$; all boats remain at height $y = 0$. At $t = 0$, Mario is at $x = 0$ and each boat is at its left endpoint (we assume boats are infinitely thin). Each boat moves periodically between its endpoints at the speed of one unit per second. So if a boat's parameters are $[L, R]$, then it is at $x = L$ at $t = 0$, at $x = R$ at $t = R - L$, at $x = L$ again at $t = 2(R - L)$, at $x = R$ again at $t = 3(R - L)$, etc. Mario cannot jump yet, so he can move between two boats if and only if their x-coordinates are equal. Furthermore, although there could be many boats sharing the same x-coordinate at some point in time, we assume that it is possible to move to any of the boats sharing the same x-coordinate.

Find the minimal time it takes to go from $x = 0$ to $x = W$, or determine that it is impossible to reach $x = W$.

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 W ($0 \leq N \leq 100$, $1 \leq W \leq 500$), the number of boats and the width of the river, respectively.
- N lines, each containing two space-separated integers L_i and R_i ($0 \leq L_i < R_i \leq W$), representing the range $[L_i, R_i]$ covered by boat i.

Output

For each test case, output one line containing an integer, the earliest possible time to reach $x = W$, or `IMPOSSIBLE` if it is impossible to reach $x = W$.

Example

Input	Output

2
2 2
0 1
1 2
3 10
0 8
2 10
2 3
	IMPOSSIBLE
	24

In the first test case, the two boats move synchronously so it is not possible to reach one from the other.

In the second test case, we can transfer to boat 3 at time 2, then to boat 2 at time 16, so that we finish at time 24.

Problem H. Museum

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

In a new wing of the Rijksmuseum in Amsterdam, there is an exhibition of world treasures. The new wing of the museum consists of rectangular rooms. Each room contains exactly one treasure, where each treasure is placed on a square pedestal which is surrounded by glass and wood.

The curator of the Rijksmuseum is intensely worried about treasures being stolen from the new wing of the museum. He decides to place as many distinct security systems as possible in each of the rooms. A security system is composed out of three identical parts, where each part sends an infrared laser beam and has two receivers for detecting the beams sent out by the two other parts. A security system is functional when its parts are placed such that the laser beams form a triangle. In addition, none of the beams forming the triangle is allowed to be obstructed by the pedestal. If there are two security systems in a room that form the same triangle, they are not distinct.

The security system's parts can only be installed on the pillars that are placed against the walls of the room. There is a pillar placed against the wall of the room per every meter of the wall. Also, a pillar is placed in each corner of the room.

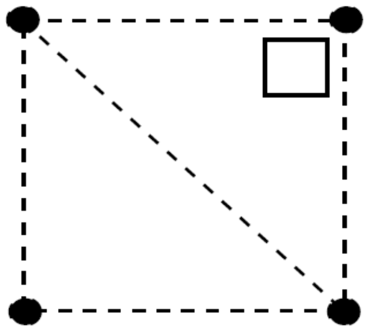


Figure 1: An example of a room with length and width of 1 meter. The pedestal is denoted by the square in the top-right corner of the room. The spots represent the pillars where the security system parts are deployed, while the dashed lines represent the infrared laser beams that are sent out by the security system parts. The pedestal blocks the beam that could be sent from the bottom-left pillar to the top-right pillar and subsequently only 2 of the 4 possible distinct security systems can be placed in this room.

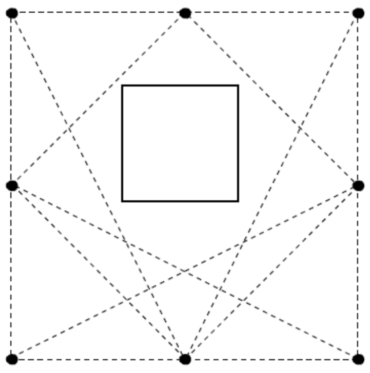


Figure 2: An example of a treasure room with length and width of 2 meters. 13 distinct security systems can be placed in this room.

The curator wishes to know the maximum number of distinct security systems that can be installed in each of the rooms of the museum's new wing.

For this problem, assume that the pillars are point-like objects, that an infinite number of parts can be placed on a single pillar and that height is no factor for this problem.

Given the length and width of a room plus the width and location of the treasure pedestal, determine the maximum number of security systems that can be installed in the given room.

Input

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

- One line containing two space-separated integers L ($1 \leq L \leq 100$) and W ($1 \leq W \leq 100$), the length and width of the room in meters, respectively.
- One line containing three space-separated floating point numbers X ($10^{-1} \leq X \leq L - 2 \cdot 10^{-1}$), Y ($10^{-1} \leq Y \leq W - 2 \cdot 10^{-1}$) and W_s ($10^{-1} \leq W_s \leq 100 - 2 \cdot 10^{-1}$), the x-coordinate, y-coordinate and width of the square pedestal all in meters. The x- and y-coordinates denote the top-left corner of the pedestal. ($x = 0$, $y = 0$) denotes the top-left corner of the room.

The square pedestal will always fit inside the room and will always be placed such that there is at least a 10^{-1} meters distance between the pedestal and the walls of the room. It is guaranteed that the distance from each corner of the pedestal to a potential laser beam (i.e. a line between two pillars) is at least 10^{-4} meters.

Output

For each test case, output one line containing a single integer: the maximum number of security systems that can be installed in the room.

Example

Input	Output

3
1 1
.81 .12 .1
2 2
.6424600 .4334300 0.6500800
2 4
.10 .19 .3
	2
	13
	158

Problem I. Six Degrees

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

For years and years, the ICT Senior Service Desk (ISSD) of the university has been confronted with a slow wired network that gives unexpected time-outs and seemingly random slow connection speeds. A new manager has been hired to solve these problems once and for all. The manager does not have any computer science or IT knowledge, but he does happen to have a strong background in sociology. He quickly finds that the network problems only affect devices of old professors with an office in some distant corner of the building.

Obsessed by the idea of six degrees of separation, the new manager proposes a rule to counter the network problems. This rule says that any two devices in the network should be connected via at most 5 intermediary devices. So, given the current lay-out of the university's wired computer network, he decides to prepare a list of all devices of peripheral professors that are currently not able to connect to all other devices within 6 steps. The manager's solution to the network problems is then to disconnect all devices on this list from the wired network at once. He explicitly ignores the fact that, possibly, then disconnecting these devices in a particular order may lead to a network structure such that some devices on the list actually no longer have to be disconnected, or that afterwards additional devices may have to be disconnected or connected to reach the actual desired result.

The board of the university, not having a background in computer science, IT or sociology is also not bothered by whether or not the proposed solution is correct, but will instead only base its decision on whether or not the prepared list of devices to be disconnected is not too long, so that not too many professors would be affected. The board will therefore only approve the plan if no more than 5% of the wired network devices is on the list.

Given the lay-out of the network, in the form of a list of pairs of IP addresses or hostnames representing directly connected devices, determine whether or not the university board will allow the new manager to execute his plan.

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 $1 \leq M \leq 30000$ denoting the number of (directly) connected pairs of devices (with at most 3000 unique devices).
- M lines, each line containing two IP addresses or hostnames of (directly) connected devices, represented by two strings of ASCII characters (of length ≤ 64) without whitespace.

Each pair of connected devices is included once in the input file. All connections are bidirectional. You may assume that all devices in the university network are in the same connected component of devices.

Output

For each test case, output one line containing either `YES` if the plan is allowed to be executed or `NO` if the plan is not allowed to be executed.

Example

Input	Output
2 5 132.229.123.1 132.229.123.2 132.229.123.2 132.229.123.3 132.229.123.3 132.229.123.4 132.229.123.4 132.229.123.5 132.229.123.5 132.229.123.6 7 a b b c c d d e e f f g g h	YES NO

Problem J. Tour de France

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

Your company has gotten a grand and prestigious contract: you are to write a program for the organisation of the Tour de France! With the large amount of money that the organisation of the Tour has, your marketing department became really fanatic, trying everything to get some work for your company that they could sell for way too much money. You, the company's best programmer, has been assigned to this job.

When you read the assignment however, both respect for the marketing department and horror at the assignment strikes you. Your marketing department has somehow managed to convince the organisation of the Tour de France that they absolutely need a program that will compute the route of the Tour through the selected French cities for them. In particular, the organisation of the Tour now believes that the best such tour would be the *shortest* one. Additionally, they have decided that the Tour should really be a tour as understood in computer science, in that the tour ends at the starting point, and that each stage (= a day of cycling from one city to the next) starts at the finish of the previous one.

You quickly look up how many stages they plan on having, see that it's a huge number (they lengthened the Tour) and you fall to your knees, burying your face in your hands in despair. Given your Computer Science background, you know that this is precisely the famous Travelling Salesman problem, famous for requiring exponential time to solve (given $\mathbb{P} \neq \mathbb{NP}$), so there is almost surely no algorithm fast enough for this many stages.

You hurry to your boss, a huge man with an equally huge temper. You start explaining how this new assignment is exactly a famous hard problem, how there are simply too many tours through these cities and that there's no way you can find the shortest one before the end of the century (skipping the dynamic programming algorithm that does better than trying all tours, because it's too complicated to explain quickly and would still not solve the problem in time). Your boss is starting to get annoyed by you, clearly convinced you must have some other motive for saying all this, such as laziness. When you say that basically the entire Computer Science community agrees with this assessment and that finding a better algorithm is a famous open problem for decades, he seems to feel that you might not be lying.

He says: "Don't worry about it. They promised us something in later negotiations: as it's hard to set out so many possible routes between cities, they decided that every city can only choose two cities as potential destinations. Furthermore, as assistance is needed from the destination city to set up a route for a stage, every destination city need only cooperate with at most two cities that want to set out a route to them. That should take care of your 'there's too many possible tours' problem. Now scram and don't bother me again."

He slams the door in your face. You feel a bit better now: apparently, the problem is now one on a directional graph, where every vertex has at most two incoming and outgoing edges... A quick calculation reveals that you still can't try all possible tours or the dynamic programming algorithm, but maybe something is known on this restriction of the Travelling Salesman problem? A quick search of the internet yields no results. The longer, more frantic search that follows also yields no results, and after cursing your luck and life in general for a minute, you resign yourself to your fate: finding some way of solving the problem fast enough by yourself in an attempt to avoid incurring the wrath of your boss.

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, the number of cities ($3 \leq N \leq 36$) and M, the number of edges ($N \leq M \leq 2 \cdot N$).
- M lines, each with three space-separated integers i, j, d, indicating that there is a directed edge from city i to city j with length d. ($0 \leq i,\ j < N,\quad i \neq j$ and $1 \leq d \leq 10000$). For any pair of cities a, b, there will be at most one line a, b, x (i.e. no duplicate edges), but there can be a line b, a, y (the backward edge).

Output

For each test case, output one line with one integer indicating the length of the shortest tour through all N cities. Note: for every test case, there will always be at least one tour possible.

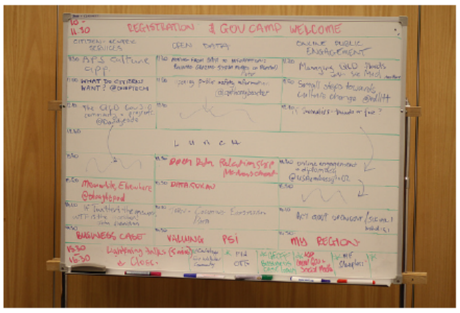
Example

Input	Output

2
3 5
0 1 2
0 2 1
1 0 1
1 2 3
2 0 4
5 10
0 2 1
0 4 5
1 0 1
1 2 2
2 4 1
2 3 3
3 1 1
3 0 4
4 3 1
4 1 6
	9
	5

Problem K. Wipe Your Whiteboards

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



You enter an empty classroom to do some homework, and you find out that someone did not clean the whiteboard properly. Apparently, the previous instruction in that room has been about the Extended Euclidean algorithm, because you see lots of intermediate results of this algorithm on the whiteboard. However, some parts of it have been wiped out, so you don't see everything they did. In particular, you're not sure what numbers they used as the inputs for the algorithm. As you didn't really feel like doing your homework in the first place, you decide to see if you can figure out the numbers they started with.

Looking at the algorithm as executed on the whiteboard, you can determine one thing for certain from their intermediate results: given that their inputs were A and B ($A,\ B \geq 1$, both integers), you see three integers R, S and Q (with $R \geq 2,\quad S \leq -2$ and $Q \geq 1$), for which you know that $A \cdot R + B \cdot S = Q$. Now, given these three numbers, you want to figure out the A and B they started out with. Unfortunately, you quickly realize that there may not be a single A and B pair that fit this, so you decide to go looking for the pair with smallest positive A and B. Finally, you decide to not actually bother trying to find an A and B such that R, S and Q are intermediate results of the Extended Euclidean algorithm applied on A and B: you decide you're happy if they just satisfy $A \cdot R + B \cdot S = Q$.

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 R, S and Q. These satisfy $2 \leq R \leq 10^8$, $-10^8 \leq S \leq -2$ and $1 \leq Q \leq 10^8$. You are given that Q is a multiple of the greatest common divisor of R and S.

Output

For each test case, output one line with two space-separated integers $A \geq 1$ and $B \geq 1$, the smallest such pair of numbers so that $A \cdot R + B \cdot S = Q$. By the smallest pair, we mean the pair such that A is minimal and if there are multiple such pairs the one of these for which B is minimal.

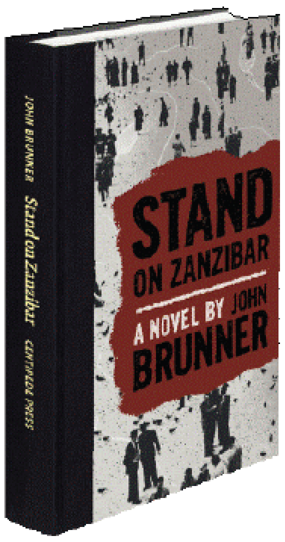
Example

Input	Output
`4`	`10 3`
`3 -5 15`	`24 91`
`110 -29 1`	`7 6`
`6 -5 12`	`6 5`
`6 -5 11`	

Problem L. Stand on Zanzibar

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

Turtles live long (and prosper). Turtles on the island Zanzibar are even immortal. Furthermore, they are asexual, and every year they give birth to at most one child. Apart from that, they do nothing. They never leave their tropical paradise.



Zanzi Bar, the first turtle on Zanzibar, has one further activity: it keeps track of the number of turtles on the island. Every New Year's Day it counts the turtles, and writes the total number in a small booklet. After many years this booklet contains a non-decreasing sequence of integers, starting with one or more ones. (After emerging from its egg on Zanzibar's beautiful beach, it took Zanzi some time to start a family on its own.)

One day Zanzi realizes that it could also be the case that turtles from abroad come to Zanzibar, by boat or plane. Now it wonders how many of the inhabitants were not born on Zanzibar. Unfortunately, it can only derive a lower bound from the sequence in the booklet. Indeed, if the number of turtles in a year is more than twice as big as the year before, the difference must be fully explained by import.

As soon as Zanzibar has 1000000 turtles, the island is totally covered with turtles, and both reproduction and import come to a halt. Please help Zanzi! Write a program that computes the lower bound of import turtles, given a sequence, as described above.

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 sequence of space-separated, positive integers ($\leq$ 1000000), non-decreasing, starting with one or more ones. For convenience, a single space and a 0 are appended to the end of the sequence.

Output

For each test case, output a line containing a single integer: the lower bound for the number of turtles not born on Zanzibar.

Example

Input	Output
`3`	`98`
`1 100 0`	`0`
`1 1 1 2 2 4 8 8 9 0`	`42`
`1 28 72 0`	

Problem M. Find the Twins

Source file name:	twins.c, twins.cpp, twins.java, twins.py
Input:	Standard
Output:	Standard

Dr. Orooji's twins (Mack and Zack) play soccer. We will assume Mack wears jersey number 18 and Zack wears 17. So, Dr. O has to look for these two numbers when trying to find the twins.

Given a list of 10 numbers, determine if the twins are there.

Input

The first input line contains a positive integer, n, indicating the number of data sets to check. The sets are on the following n input lines, one set per line. Each set consists of exactly 10 single-space-separated distinct integers (each integer between 11 and 99 inclusive) giving the jersey numbers for the players.

Output

Print each input set. Then, on the next output line, print one of four messages (mack, zack, both, none), indicating how many of the twins are in the set.

Example

Input	Output

4
11 99 88 17 19 20 12 13 33 44
11 12 13 14 15 16 66 88 19 20
20 18 55 66 77 88 17 33 44 11
12 23 34 45 56 67 78 89 91 18
	11 99 88 17 19 20 12 13 33 44
	zack
	11 12 13 14 15 16 66 88 19 20
	none
	20 18 55 66 77 88 17 33 44 11
	both
	12 23 34 45 56 67 78 89 91 18
	mack

Problem N. Medal Ranking

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

When different countries compete against each other (e.g., in the Olympics), they receive gold/silver/bronze medals. The countries can then be ranked in one of two ways: by "count" which is based on the total number of medals (regardless of the medal colors) or by "color" which is based on the number of gold medals (and silver medals if tied in gold medals, and bronze medals if tied in gold and silver).

Given the gold/silver/bronze medal counts for USA and Russia, you are to determine if USA wins in these two ranking methods.

Input

The first input line contains a positive integer, n, indicating the number of data sets to check. The sets are on the following n input lines, one set per line. Each set consists of 6 integers (each integer between 0 and 500 inclusive); the first three integers represent (respectively) the gold, silver, and bronze medal counts for USA; the last three integers provide this info for Russia (in same order).

Output

Print each input set. Then, on the next output line, print one of four messages (count, color, both, none), indicating how USA can win. USA will win by count if its total medal count is higher than the total for Russia. USA will win by color if it has more gold medals than Russia (if tied in gold, then USA must have more silver; if tied in gold and silver, then USA must have more bronze).

Example

Input	Output

```
5
10 5 15 10 1 0
10 5 15 10 6 10
12 5 10 5 20 30
10 0 15 10 5 30
10 5 15 10 5 15
```
	```
	10 5 15 10 1 0
	both
	10 5 15 10 6 10
	count
	12 5 10 5 20 30
	color
	10 0 15 10 5 30
	none
	10 5 15 10 5 15
	none
	```