Problem A. Avoiding the Apocalypse

Source file name:	apocalypse.c, apocalypse.cpp, apocalypse.java
Input:	`standard`
Output:	`standard`

You and the rest of your team are stuck in a town during the zombie apocalypse of 2020. You all might be infected with the virus and hence you will have to find your way to one of the medical facilities to get a cure before you also become zombies. Because you are scientists you quickly realize that it is safer to try and sneak your way past the zombies than to recklessly start fighting them. Obviously the zombies are everywhere so some streets might take more time to sneak through than others. It is also obvious that by splitting up into smaller groups it is easier to move around undetected.

Furthermore, since these zombies have not mutated to the extent that they actually have eyes in the back of their heads, it might be easy to cross some streets in one direction while hard or impossible to cross them in the other direction.

How many of you can sneak past all the zombies and get to a medical facility in time?

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with a single integer n ($1 \leq n \leq 1000$): the number of locations in the town.
- one line with three space-separated integers i, g and s ($1 \leq i \leq n$ and $1 \leq g \leq 100$ and $1 \leq s \leq 100$): the starting location of your group, the number of people in it, and the number of time steps you have to get to the safety of a medical facility, respectively.
- one line with a single integer m ($1 \leq m \leq n$): the number of medical facilities in the town.
- lines, each with a single integer x ($1 \leq x \leq n$): the location of each of the medical facilities.
- one line with a single integer r ($0 \leq r \leq 1000$): the number of roads in the town.
- r lines, each with four space-separated integers a, b, p and t ($1 \leq a, b \leq n$ and $a \neq b$ and $1 \leq p \leq 100$ and $1 \leq t \leq 100$), indicating that there is a road from a to b, which p people can enter at every time step and takes t timesteps to traverse.

There are at most two roads - one in each direction - between any pair of locations. The locations are safe enough to wait at for any amount of time and do not have a limit on the number of people that can be there.

Output

Per test case:

- one line with a single integer: the largest number of people that can get to a medical facility in time.

Example

Input	Output
2 4 3 8 5 2 2 4 5 1 2 1 3 3 2 1 4 3 1 2 1 1 4 1 3 3 4 1 3 4 3 10 5 2 2 4 5 1 2 1 3 3 2 1 4 3 1 2 1 1 4 1 3 3 4 1 3	8 9

Problem B. Button Bashing

Source file name:	bashing.c, bashing.cpp, bashing.java
Input:	`standard`
Output:	`standard`

You recently acquired a new microwave, and noticed that it provides a large number of buttons to be able to quickly specify the time that the microwave should be running for. There are buttons both for adding time, and for subtracting time. You wonder how efficient you can be when entering cooking times: you want to minimize the number of required button presses.

The microwave can be running for at least 0 seconds, and at most 1 hour. If a button press would result in a cooking time of less than 0 seconds, the microwave will set the cooking time to 0 seconds. If a button press would result in a cooking time of more than 1 hour, the microwave will set the cooking time to 1 hour. Initially, the microwave will run for 0 seconds. There will always be a button adding at least 1 second to the cooking time.

Given the buttons that the microwave provides for entering cooking times, determine the least amount of button presses required to let the microwave run for a certain amount of time. If it is not possible to enter the desired cooking time precisely, determine the smallest achievable cooking time above the target, and the minimum number of button presses required for that cooking time, instead. The microwave does not allow to adjust the cooking time once it has started cooking.

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with two space-separated integers n and t ($1 \leq n \leq 16$ and $0 \leq t \leq 3600$): the number of buttons available to change the cooking time, and the desired cooking time in seconds, respectively.
- one line with n space-separated integers b_i ($-3600 \leq b_i \leq 3600$): the number of seconds added to the cooking time when button i is pressed.

Output

Per test case:

- one line with two space-separated integers: the minimum number of button presses required to reach the required cooking time, and the minimum number of extra seconds that the microwave must be running for, respectively.

Example

Input	Output
2 3 50 -10 10 60 1 50 20	2 0 3 10

Problem C. Citadel Construction

Source file name:	citadel.c, citadel.cpp, citadel.java
Input:	`standard`
Output:	`standard`

The army wants to put up a new base in dangerous territory. The base will be surrounded by a citadel consisting of a number of straight walls. At every corner where two walls meet, there will be a watchtower. In order to enable efficient coordination in case of attack, the number of watchtowers should not exceed four.

After a detailed survey of the area, the army has concluded that not all locations are equally suitable for a watchtower: the ground needs to be firm enough to support the weight of the tower, and there should be enough visibility. They have selected a number of locations that meet the requirements.

Within these constraints, the army would like to build a base that is as large as possible. For this, they have come to you for help. Since the results of the survey are of course classified, you will not be given the possible locations directly; rather, you should write a program that can take them as input. Furthermore, they want the program to be able to handle multiple test cases in one go, so that they can hide the real data among lots of fake data.

For the computation of the area, you may assume that the watchtowers are infinitesimal in size and the walls infinitesimal in width.

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with a single integer n ($3 \leq n \leq 1000$): the number of locations that are suitable for a watchtower.
- n lines, each with two space-separated integers x and y ($-10\,000 \leq x, y \leq 10\,000$): the coordinates of each location.

All locations are distinct.

Output

Per test case:

- one line with a single number: the largest possible area that the base can have. This number will be either an integer or a half-integer. If it is an integer, print that integer; if it is a half-integer, print the integer part followed by ".5". Trailing zeros are not allowed.

Example

Input	Output
3 6 0 0 3 7 10 0 11 6 0 10 10 10 5 0 0 -2 -2 3 -2 0 1 0 3 10 3 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3 2 3 8 4	100 12.5 31

Problem D. Dropping Directions

Source file name:	dropping.c, dropping.cpp, dropping.java
Input:	`standard`
Output:	`standard`

Your sports club has a rivaling sports club in the same city. They did some awful things to you and you want to get back at them. You have learned that they are planning on doing a 'drop': they drive people blindfolded to a city none of the participants know and tell them to find a specific place, the goal. They then have fun randomly walking through the city trying to find the goal.

You intend to spoil their fun thoroughly: you know that they promise a prize for whoever reaches the goal first, so the participants will use all available means to get to the goal. Indeed, you are fairly sure that if you set up official-looking signposts in that city in advance, they will probably follow them. You therefore decide to place signposts throughout the city so that no matter where the participants get dropped, they can follow the signposts to the goal; this takes out the element of 'randomly walking around' and therefore all the fun.

However, official-looking signposts are not cheap and attract a lot of attention, particularly from police officers. So you wish to minimize the number of signposts you have to place in the city. This may lead the participants to use a very slow detour, but they don't know the city anyway, so they won't find out.

You get yourself a map of the city and start planning. You notice one nice aspect of the city: all intersections are cross-shaped, so it is easy to predict where participants will go to: they will just go to the opposite side of the intersection they arrive at. If a participant gets dropped at an intersection with a signpost, he or she will follow that sign; otherwise they go in an arbitrary direction until they hit a sign post. You know that participants never get dropped at the goal (that would be silly).

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with two space-separated integers n and g ($5 \leq n \leq 100\,000$ and $1 \leq g \leq n$): the number of intersections and the goal, respectively.
- n lines, each with four space-separated integers a, b, c and d ($1 \leq a, b, c, d \leq n$). The i-th line gives the intersections you end up at if you follow one of the roads adjacent to the i-th intersection; more precisely, participants who approach the i-th intersection coming from intersection a will continue towards intersection c and vice versa, while participants coming from intersection b will continue towards intersection d and vice versa

Each intersection connects to four different other intersections.

Output

Per test case:

- one line with a single integer: the smallest number of signposts needed to ruin the fun.

Example

Input	Output
4 5 5 2 3 4 5 3 4 5 1 4 5 1 2 5 1 2 3 1 2 3 4 5 5 5 3 2 4 4 5 3 1 1 5 2 4 1 3 5 2 1 3 2 4 5 4 5 3 2 4 4 5 3 1 1 5 2 4 1 3 5 2 1 3 2 4 5 5 5 3 2 4 4 5 3 1 1 5 2 4 1 5 2 3 1 3 2 4	0 0 1 1

Problem E. Excellent Engineers

Source file name:	engineers.c, engineers.cpp, engineers.java
Input:	`standard`
Output:	`standard`

You are working for an agency that selects the best software engineers from Belgium, the Netherlands and Luxembourg for employment at various international companies. Given the very large number of excellent software engineers these countries have to offer, the agency has charged you to develop a tool that quickly selects the best candidates available from the agency's files.

Before a software engineer is included in the agency's files, he has to undergo extensive testing. Based on these tests, all software engineers are ranked on three essential skills: communication skills, programming skills, and algorithmic knowledge. The software engineer with rank one in the category algorithmic knowledge is the best algorithmic expert in the files, with rank two the second best, etcetera.

For potential customers, your tool needs to process the agency's files and produce a shortlist of the potentially most interesting candidates. A software engineer is a potential candidate that is to be put on this shortlist if there is no other software engineer in the files that scores better on all three skills. That is, an engineer is to be put on the shortlist if there is no other software engineer that has better communication skills, better programming skills, and more algorithmic knowledge.

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with a single integer n ($1 \leq n \leq 100\ 000$): the number of software engineers in the agency's files.
- n lines, each with three space-separated integers r_1 , r_2 and r_3 ($1 \leq r_1, r_2, r_3 \leq n$): the rank of each software engineer from the files with respect to his communication skills, programming skills and algorithmic knowledge, respectively.

For each skill s and each rank x between 1 and n, there is exactly one engineer with $r_s = x$.

Output

Per test case:

- one line with a single integer: the number of candidates on the shortlist.

Example

Input	Output
3 3 2 3 2 3 2 3 1 1 1 3 1 2 3 2 3 1 3 1 2 10 1 7 10 3 9 7 2 2 9 5 10 8 4 3 5 7 5 2 6 1 3 9 6 6 8 4 4 10 8 1	1 3 7

Problem F. Floating Formation

Source file name:	formation.c, formation.cpp, formation.java
Input:	`standard`
Output:	`standard`

Industrial design students have come up with an amazing idea: they have made some awesome pieces of design and they want to showcase them floating on the water of the IJsselmeer! Unfortunately, their designs do not stay afloat by themselves, so they took a few boats and used them to tie designs together.

The left and right sides of a boat have rope connection points that can be tied to designs. A boat can therefore be attached to at most two designs. A design that is connected to two or more boats will stay afloat indefinitely. Unfortunately, designs that are connected to only one boat will not float properly: the part of the design furthest away from the boat will sink a bit and be partially in the water, so it will soak, making the design heavier, which will eventually sink the design, making it too heavy for the boat to keep afloat. A boat that is attached to a design that has sunk will detach itself from the other design First sample input it is attached to, if applicable, and will carry the soaked design back to shore. Designs that are soaked are placed on the beach, but the students want the ship that brought the design to shore to stay attached to the soaked design, as they feel this composition has artistic value. The boat will therefore not go back to the designs that are still floating, which means that if the boat was attached to another design, that design loses a supporting boat and will also start to sink!

Keeping the above in mind, there are simple ways to string the designs together with boats so that no design sinks, such as forming a circle. This is not the approach that the students used: their approach involved much creativity. The result, although a single connected component, is a configuration that leaves many designs connected by only one boat and hence in danger of sinking. You also note that the students have always attached boats to two designs: there are no boats that are attached to a single design. Also, boats can never be attached to a single design twice, and there are no two boats that are attached to the same pair of designs.

Reluctantly, the students have admitted that they may have made mistakes, so they have asked you to use the few boats they have left to keep as many designs afloat as possible. As the current configuration has much artistic value, you are not allowed to change it. In fact, you are only allowed to use the connection points on the left sides of the spare boats (you are not allowed to use the right side), so that the geometric ordering of the boats is not changed.

Given this situation, how can you use the boats so that the number of designs that end up sunk is minimized? Note that we want your program to be able to handle non-planar configurations as well, in case the students ever turn to air balloons. For all testcases, you may assume that at least one design would stay afloat, even if none of the spare boats are used.

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with three space-separated integers n, m and k ($1 \leq n, m, k \leq 10\,000$): the number of designs, used boats and spare boats respectively.
- m lines, each with two space-separated integers a and b ($1 \leq a, b \leq n$ and $a \neq b$): a boat is attached to design a on its left side and design b on its right side.

Output

Per test case:

- one line with a single integer: the smallest number of designs that are eventually sunk after employing

the spare ships.

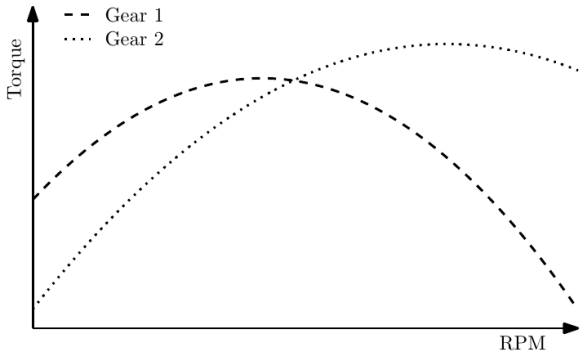
Example

Input	Output
2 6 6 1 1 2 2 3 1 3 1 4 4 5 1 6 11 11 3 1 2 2 3 1 3 1 4 4 5 2 6 3 7 7 8 7 9 8 10 9 11	1 1

Problem G. Growling Gears

Source file name:	growling.c, growling.cpp, growling.java
Input:	`standard`
Output:	`standard`

The *Best Acceleration Production Company* specializes in multi-gear engines. The performance of an engine in a certain gear, measured in the amount of torque produced, is not constant: the amount of torque depends on the RPM of the engine. This relationship can be described using a *torque-RPM curve*.



The torque-RPM curve of the gears given in the second sample input. The second gear can produce the highest torque.

For the latest line of engines, the torque-RPM curve of all gears in the engine is a parabola of the form $T = -aR^2 + bR + c$, where R is the RPM of the engine, and T is the resulting torque.

Given the parabolas describing all gears in an engine, determine the gear in which the highest torque is produced. The first gear is gear 1, the second gear is gear 2, etc. There will be only one gear that produces the highest torque: all test cases are such that the maximum torque is at least 1 higher than the maximum torque in all the other gears.

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with a single integer n ($1 \leq n \leq 10$): the number of gears in the engine.
- n lines, each with three space-separated integers a, b and c ($1 \leq a, b, c \leq 10\,000$): the parameters of the parabola $T = -aR^2 + bR + c$ describing the torque-RPM curve of each engine.

Output

Per test case:

- one line with a single integer: the gear in which the maximum torque is generated.

Example

Input	Output
3 1 1 4 2 2 3 126 1400 2 152 208 2 3 127 1400 2 154 208	1 2 2

Problem H. Highway Hassle

Source file name:	highway.c, highway.cpp, highway.java
Input:	`standard`
Output:	`standard`

A transport company has come to you for help. One of the biggest expenses that they have is the petrol for the trucks. They would like to minimize the money they need to spend on petrol.

Because of the long trips, a truck driver typically needs to stop multiple times at a petrol station to tank up. What complicates matters is that the price of petrol is not the same at every station. The differences could in fact be so significant that it pays to take a detour in order to visit a station with a low price. Yet another complication is that the price is not the same every day (but it does not change during the day).

The good news is that they can find out, every morning, what the price of petrol is at every station for that day. They also have, for every destination, a simple graph representing the relevant part of the road network, containing only the major intersections and petrol stations as nodes. Furthermore, they know for every road exactly how much petrol is needed to go from one node to the other, down to the milliliter; it does not depend on the direction or the amount of petrol in the tank. The truck drivers also have the ability to tank with milliliter precision.

It is perfectly fine for a truck to run out of petrol at the exact moment it arrives at a petrol station or at the destination; there is in fact a spare tank to allow for small fluctuations in fuel consumption, but that petrol is not supposed to be used. You may therefore ignore its existence.

A final thing to take into consideration is that the trucks have a fuel tank of limited size. With all that information, can you work out what the optimal path to the destination is, along with the optimal tanking strategy?

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with three space-separated integers n, m and s ($2 \leq n \leq 1000$ and $1 \leq m \leq 10\,000$ and $1 \leq s \leq 120$): the number of nodes, edges and petrol stations, respectively.
- one line with a single integer t ($1 \leq t \leq 100\,000$): the maximum amount of petrol that the fuel tank can hold in milliliters.
- m lines, each with three space-separated integers a, b and f ($1 \leq a, b \leq n$ and $a \neq b$ and $1 \leq f \leq 100\,000$), indicating that there is a road between nodes a and b which takes f milliliters of petrol (fuel) to traverse.
- s lines, each with two space-separated integers x and p ($1 \leq x \leq n$ and $1 \leq p \leq 100$): the node x where each petrol station is located and the price p per milliliter of petrol at that station, respectively.
- one line with two space-separated integers c and d ($1 \leq c, d \leq n$ and $c \neq d$): the nodes where the company and the destination are located, respectively.

Every road is bidirectional. There is at most one road between any pair of nodes. There is always a petrol station at node c (it is right next to the company). The truck starts with an empty tank. The destination is guaranteed to be reachable.

Output

Per test case:

- one line with a single integer: the minimum amount of money that needs to be spent on petrol.

Example

Input	Output
3 3 3 2 2000 1 3 800 1 2 500 2 3 500 1 70 2 40 1 3 5 5 3 1000 1 2 800 2 5 800 1 3 400 3 4 600 4 5 600 1 80 2 90 3 20 1 5 4 3 3 1000 1 2 200 2 3 600 3 4 300 1 40 2 70 3 90 2 4	55000 134000 61000

Problem I. Interesting Integers

Source file name:	interest.c, interest.cpp, interest.java
Input:	`standard`
Output:	`standard`

Undoubtedly you know of the Fibonacci numbers. Starting with $F_1 = 1$ and $F_2 = 1$, every next number is the sum of the two previous ones. This results in the sequence $1, 1, 2, 3, 5, 8, 13, \ldots$.

Now let us consider more generally sequences that obey the same recursion relation

$$G_i = G_{i-1} + G_{i-2} \qquad \text{for } i > 2$$

but start with two numbers $G_1 \leq G_2$ of our own choice. We shall call these Gabonacci sequences. For example, if one uses $G_1 = 1$ and $G_2 = 3$, one gets what are known as the Lucas numbers: $1, 3, 4, 7, 11, 18, 29, \ldots$. These numbers are - apart from 1 and 3 - different from the Fibonacci numbers.

By choosing the first two numbers appropriately, you can get any number you like to appear in the Gabonacci sequence. For example, the number n appears in the sequence that starts with 1 and $n - 1$, but that is a bit lame. It would be more fun to start with numbers that are as small as possible, would you not agree?

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with a single integer n ($2 \leq n \leq 10^9$): the number to appear in the sequence.

Output

Per test case:

- one line with two integers a and b ($0 < a \leq b$), such that, for $G_1 = a$ and $G_2 = b, G_k = n$ for some k. These numbers should be the smallest possible, i.e., there should be no numbers a' and b' with the same property, for which $b' < b$, or for which $b' = b$ and $a' < a$.

Example

Input	Output
5 89 123 1000 1573655 842831057	1 1 1 3 2 10 985 1971 2 7

Problem J. Jury Jeopardy

Source file name:	jeopardy.c, jeopardy.cpp, jeopardy.java
Input:	`standard`
Output:	`standard`

What would a programming contest be without a problem featuring an ASCII-maze? Do not despair: one of the judges has designed such a problem.

The problem is about a maze that has exactly one entrance/exit, contains no cycles and has no empty space that is completely enclosed by walls. A robot is sent in to explore the entire maze. The robot always faces the direction it travels in. At every step, the robot will try to turn right. If there is a wall there, it will attempt to go forward instead. If that is not possible, it will try to turn left. If all three directions are unavailable, it will turn back.

The challenge for the contestants is to write a program that describes the path of the robot, starting from the entrance/exit square until it finally comes back to it. The movements are described by a single letter: 'F' means forward, 'L' is left, 'R' is right and 'B' stands for backward.

Each of 'L', 'R' and 'B' does not only describe the change in orientation of the robot, but also the advancement of one square in that direction. The robot's initial direction is East. In addition, the path of the robot always ends at the entrance/exit square.

The judge responsible for the problem had completed all the samples and testdata, when disaster struck: the input file got deleted and there is no way to recover it! Fortunately the output and the samples are still there. Can you reconstruct the input from the output? For your convenience, he has manually added the number of test cases to both the sample output and the testdata output.

Input

On the first line one positive number: the number of test cases. After that per test case:

- one line with a single string: the movements of the robot through the maze.

Output

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with two space-separated integers h and w ($3 \leq h, w \leq 100$): the height and width of the maze, respectively.
- h lines, each with w characters, describing the maze: a '#' indicates a wall and a '.' represents an empty square.

The entire contour of the maze consists of walls, with the exception of one square on the left: this is the entrance. The maze contains no cycles (i.e. paths that would lead the robot back to a square it had left in another direction) and no empty squares that cannot be reached from the entrance. Every row or column - with the exception of the top row, bottom row and right column - contains at least one empty square.

Example

Input	Output
3 FFRBLF FFRFRBRFBFRBRFLF FRLFFFLBRFFFRFFFRFRFBRFLBRFRLFLFFR	3 4 4 #### ...# ##.# #### 7 5 ##### ...## ##.## #...# ##.## ##.## ##### 7 7 ####### #...#.# #.#...# #.#.### ..###.# #.....# #######

Problem K. Key to Knowledge

Source file name:	knowledge.c, knowledge.cpp, knowledge.java
Input:	`standard`
Output:	`standard`

A while ago a class of students took an exam. It consisted of true-or-false questions only, but the questions were really tough! Even afterward, with the help of the book, lecture notes and each other, the students are not sure what all the correct answers were. They could ask the professor of course, but he is not the kind of guy you would dare to disturb with questions that betray your ignorance. Anyway, the students at least have an idea of how they performed.

The professor has just returned the exams, but the results are not quite as expected. Unfortunately, he did not indicate on their exams which answers were right or wrong, he has simply put the number of correct answers at the top. So what were the correct answers according to the professor then? Given how far off some of the grades are from expected, there is a suspicion among some of the students that the professor did not count the correct answers properly.

Given, for each student, the answers he/she gave and the number of correct answers, can you work out what set of correct answers matches the results?

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with two space-separated integers n and m ($1 \leq n \leq 12$ and $1 \leq m \leq 30$): the number of students and the number of questions in the exam, respectively.
- n lines, each with m digits, each digit either 0 or 1: the answers given by each student, with 0 representing "false" and 1 representing "true". This is followed by a space and a single integer c ($0 \leq c \leq m$): the number of correct answers for that student.

Output

Per test case:

- one line with m digits, each one either 0 or 1: the unique set of correct answers that could account for all the results. If there is not exactly one such set, the line should read "# solutions" instead, with the number of solutions in place of '#'.

Example

Input	Output
3 3 5 01101 4 10100 3 00011 3 3 5 01101 0 10100 3 00011 2 4 4 0000 2 1010 2 0101 2 1111 2	00101 0 solutions 4 solutions

Problem L. Vowel Count

Source file name:	vowel.c, vowel.cpp, vowel.java
Input:	`standard`
Output:	`standard`

Dr. Orooji noticed that his name has more vowels than consonants. Since he likes meeting people like himself, he has asked you to write a program to help him identify such names.

Given a name, determine whether or not it has more vowels than consonants. Assume the vowels are "aeiou".

Input

The first input line contains a positive integer, n, indicating the number of names to check. The names are on the following n input lines, one name per line. Each name starts in column 1 and consists of 1-20 lowercase letters (assume the name will not contain any other characters).

Output

Output each name on a separate line as it appears in the input followed by a 1 (one) or 0 (zero) on the next line indicating whether or not it has more vowels than consonants. Follow the format illustrated in Example Output.

Example

Input	Output
4 ali arup travis orooji	ali 1 arup 0 travis 0 orooji 1

Problem M. Soccer Standings

Source file name:	soccer.c, soccer.cpp, soccer.java
Input:	`standard`
Output:	`standard`

In a soccer match, a team either earns a win, tie or loss. A win is worth 3 points, a tie is worth 1 point, and a loss is worth 0 points. Unfortunately, due to poor record-keeping, some leagues have only saved the number of total matches played and the number of points each team has earned. One of these leagues has asked you to write a program to recreate the possible combinations of wins, ties and losses for certain teams in the league.

Given the number of games played by a soccer team in a season and the number of points earned by the team, list each possible combination of wins, ties and losses that the team could have gotten to achieve the given total points.

Input

The first input line contains a positive integer, n, indicating the number of teams for which you are to reconstruct possible records. The teams' information are on the following n input lines, one team per line. Each of these lines contains two space separated integers: g ($0 < g \leq 100$), and p ($0 \leq p \leq 300$), representing the number of games played and the total points earned by the team, respectively. It is guaranteed that there is at least one possible combination of wins, ties and losses that is consistent with the given information for each team.

Output

For each team, first output header info with the following format:

```
Team #k
Games:  g
Points:  p
Possible records:
```

where k is the team number (starting with 1), g is the number of games, and p is the total points earned. Following the above header info, output the possible records, each on a separate line with the format:

```
 w-t-l
```

where w is the number of wins, t is the number of ties and l is the number of losses. Print these by descending order of wins.

Leave a blank line after the output for each team. Follow the format illustrated in Example Output.

Example

Input	Output
3 6 10 1 3 4 4	Team #1 Games: 6 Points: 10 Possible records: 3-1-2 2-4-0 Team #2 Games: 1 Points: 3 Possible records: 1-0-0 Team #3 Games: 4 Points: 4 Possible records: 1-1-2 0-4-0