10th Annual Manitoba Programming Contest
September 24, 2016

General Instructions:

1. This document is printed double-sided. Please read both sides of each page.
2. Submit solutions using the PC^2 software.
3. All input should be read from standard input.
4. All output should be written to standard output.
5. Each problem has a ten (10) second time limit for CPU time when executed on the judging data.

UNIVERSITY OF MANITOBA

Sponsors

Google

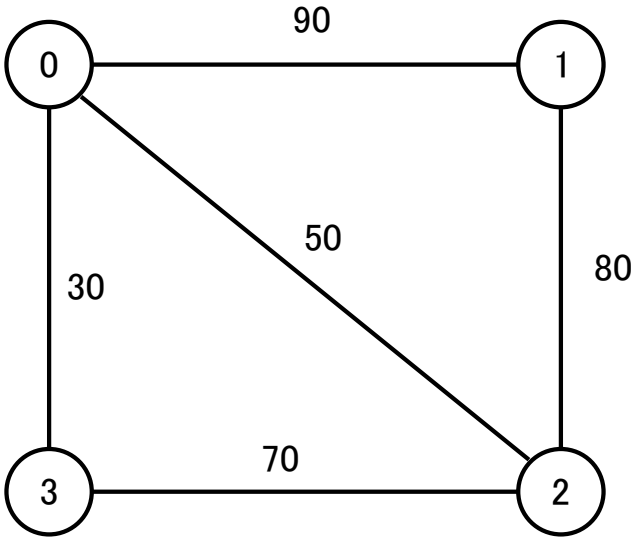
UManitoba Computer Science Students Association

Problem 1 – Slowing Down

You know what sucks while driving? Going at highway speed and coming into a town and having to slow to a low speed limit. To solve this, you'd like to find ways to drive into town that slow down gradually, moving you from one speed limit to another by decreasing by 10 km/h at each step, so you can get used to each speed limit before slowing down to the next.

Write a program that reads in a map of town with speed limits for all roads, each with their speed limits, and a starting position, and finds the lowest speed you can get to in that town by decreasing your speed by 10 km/h each time the speed limit changes. Each road will be given by the connection between two intersections and the speed limit between those intersections. Roads can be travelled in either direction and there is only one road between any pair of intersections. There is no road between an intersection and itself. Each intersection is given by a number, with the first intersection numbered by zero, and using each intersection number up to a predefined limit n. Your starting point is an intersection at the edge of town, and your starting speed is provided. Intersections may not connect more than one road: they may also be dead-ends at the edge of town.

For instance, in the following town map, intersections are given as circles with their number inside them, while roads are given by lines connecting the intersections. In this town, if you start at intersection 0 driving 100 km/h, you can get to intersection 3 by going through intersections 1 and 2, and when you arrive at intersection 3, you are driving 70 km/h. This is the lowest speed you can get to, so the answer in this case is 70 km/h.



Note that when you are moving from road to road (by passing through intersections) you must either decrease speed by 10 km/h or maintain the same speed as the previous road.

Input

The first line of input contains a single integer, which is the number of towns you should examine. Each town starts with a line of input with four integers: the number of intersections (n), the number of roads (r), the starting intersection (s) and the starting speed (v). The next r lines contain three integers, i j k, which represents that the speed limit between intersection i and j is k km/h. The integers i,j,k all satisfy $0 \le i < j < n$, $0 < k \le v$. All integers are separated from each other on the same line by a single space.

Output

For each town, output "TOWN X: y" where X is a number starting from 0 and y is the lowest possible speed that you can drive following the rules above. There should be one space following the colon.

Sample Input	Sample Output
2 4 5 0 100 0 1 90 0 2 50 0 3 30 1 2 80 2 3 70 5 5 0 100 0 1 100 1 2 90 2 3 90 3 1 90 3 4 80	Town 0: 70 Town 1: 80

Problem 2 – Morton Numbers

Morton numbers (aka interleaved bit patterns) are useful for linearizing 2D integer coordinates. In particular, the Morton order maps an n-dimensional (in this question 2D) space onto a list of numbers. This mapping has some nice properties. For example, coordinates that are close to each in the 2D space have Morton numbers that are also close to each other.

Converting a pair of integer coordinates to a Morton code involves converting the decimal values to binary and interleaving the bits of each coordinate, starting with the y coordinate. For example, if we want to interleave the coordinate (5,9), we get

(x,y) = (5,9) = (**0101,***1001*) in binary.

Interleaving gives *1***0***0***1***0***0***1***1** = 10010011. Converting this number to decimal gives 147. So the Morton number for (5,9) is 147.

Write a program that reads a series of 2D (x,y) coordinates and converts them to the corresponding Morton code.

Input

Input will consist of the number of total coordinates immediately followed by the coordinate pairs (one pair per line), where each coordinate is an unsigned 16-bit integer.

Output

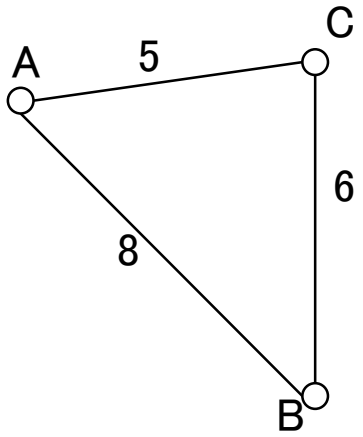
Output the Morton codes, one per line, where each Morton code is an unsigned 32-bit integer.

Sample Input	Sample Output
2 5 9 8037 45813	147 2338176563

Problem 3 – Dog Walk Penalty

You walk your dog every day, and she always wants to visit places you don't want to go. This includes places where there are fire hydrants, other dogs, squirrels, rabbits, imaginary rabbits, lights, sounds, absence of sounds, or other stimuli or lack of stimuli. You, on the other hand, would like to walk in a straight line to your destination. You would like to know how much more you walk by going to visit a certain location and then going to your destination over taking the shortest path to your destination, as you want to.

You are given three points: your current location, your goal, and your dog's goal. Calculate the percentage penalty to visit your dog's goal and then go to your goal over simply having travelled to your goal. For instance, in the diagram below, your current position is point A, your goal is point B and your dog's goal is point C.



The minimum distance to travel to your goal is 8 units away and the minimum distance to travel to your dog's goal and then your goal is 11 units. Then the ratio of the longer path is 11/8 = 1.375 and the penalty in this case is 37.5% extra distance walked.

Input

The first line of input gives the number of test cases n. Each of the next n lines represents one test case. Each test case is given by four integers `x1 y1 x2 y2`. The test case represents a starting point at the origin `(0, 0)`, your dog's goal point is `(x1,y1)` and your goal point is `(x2,y2)`.

Output

Calculate the percentage of extra time you would spend by visiting your dog's goal point before going to your goal point. For each case, output "`Case X: YYY%`" where X is the case number (starting at 0) and x is the percent extra time spent visiting your dog's goal point. The percent should be expressed to the nearest whole number.

Sample Input	Sample Output
2 3 0 3 4 50 0 53 60	Case 0: 40% Case 1: 37%

Problem 4 – Reporting Round Robin Ranking Results

In a sports tournament, a *round robin* is a preliminary set of matches that helps rank teams for a later playoff stage. In a round robin, each team plays every other team exactly once. The number of wins and losses for each team is recorded to rank teams. Unfortunately, a round robin can result in several teams having the same win-loss record. This means that tie-breaking procedures are usually necessary to determine which teams advance to the playoffs. Because of this, knowing the possible outcomes of the round robin is important.

For four teams, the four possible outcomes of a round robin (given by the win-loss record for the four teams) are

Outcome 1	Outcome 2	Outcome 3	Outcome 4
3-0	3-0	2-1	2-1
2-1	1-2	2-1	2-1
1-2	1-2	2-1	1-2
0-2	1-2	0-3	1-2

Notice that which team has which record is not crucial. We are only interested in the possible combinations of win-loss records of all teams.

If a tournament has n teams, how many different round robin outcomes are there?

Input

The input consists of several tournament sizes (the number of teams in the tournament), each on their own lines. The size of the tournaments is greater than 1 and at most 10. The final line of input is a zero. It should not be processed as input.

Output

Start each case with a line consisting of "`Case`", one space, the case number, a colon, one space, and then the number of possible outcomes for the round robin.

Sample Input	Sample Output
4 0	Case 1: 4

Problem 5 – Race Winner

Some cycling races for children are run on a small circuit for a preset amount of time. The race goes like this: all racers are given a unique number, and start the race. The racers do as many loops as possible on the circuit. Each time they go around one lap, their race number is recorded. At the end of a preset amount of time (e.g., 30 minutes), all riders are allowed to finish the lap they are on, and their final time is recorded. It does not matter how late a racer finishes (e.g., it doesn't matter if how much over the preset time they finish). When the rider finishes, that lap is counted as a finished lap for the list of laps.

So the race ends with two lists of information:

1. The list of laps: this is a list of race numbers. Each time a race number appears, that racer did one lap.
2. The list of finishing times: this is a list of race numbers and associated times. This is the time that each racer finished the race.

The winner of the race is the racer that completed the most number of laps. If two racers completed the same number of laps, the racer with the lower finishing time is the winner.

Given the two lists for a race, find the winner of the race.

Input

The first line of input gives the number of test cases.

Each test case consists of three lines of input. The first line of input is the number of racers n. You may assume that the race numbers are 0 to n-1.

The second line of input for a race is a list of race numbers, corresponding to laps of the circuit. These are given in order that they occur (so the first racer listed is the first racer to complete a lap).

The third line of input for a race is the finishing times. Each racer will have a finishing time (no one abandons the race). The finishing times are given in increasing order, so the lowest finishing times are given first and the last racer to get off the circuit is given last in the input. The times are given as MM:SS, representing a time in minutes and seconds. The number of minutes is less than 100 and the number of seconds is less than 60.

Output

For each race, output the number of the racer who won the race, using the format "`Race X: y`" where `X` is the race number (starting from 1) and `y` is the unique number of the winner of the race.

Sample Input	Sample Output
2 3 0 1 2 0 2 1 0 2 10:00 1 11:00 0 12:00 4 3 3 3 0 3 1 2 3 3 0 0 0 3 0 0 0 3 45:13 0 45:14 2 75:32 1 99:59	Race 1: 0 Race 2: 3

Problem 6 – pidgeycalc

In Pokemon Go, you are required to catch all of them. In particular, the game encourages you to catch pokemon, and then evolve those pokemon in your possession into other pokemon. Each time you catch a pokemon, you collect it, as well as a certain amount of candy, a resource (specific to that pokemon) required to evolve the pokemon. Each pokemon has a cost (in candy) to evolve it. However, each pokemon can also be sold to get one additional piece of candy. Finally, whenever you evolve one pokemon, you receive one piece of candy. Pokemon that have been evolved cannot be evolved again.

In this question, you are asked to calculate the maximum number of pokemon of a given type you can evolve, given the number of pokemon you have and the number of candy you have.

As an example, consider the Pidgey pokemon. For Pidgeys, it costs 12 Pidgey candy to evolve one Pidgey. So if you have 3 Pidgeys and 30 candies, you can evolve two Pidgeys (cost is 24 candy). For evolving, you receive two more Pidgey candies, and you end up with 1 Pidgey remaining, 2 evolutions and 8 Pidgey candies. On the other hand, if you had 10 Pidgeys and 30 candies, you could evolve 2 Pidgeys, giving you 8 Pidgeys and 8 Pidgy candies. You could then sell 4 Pidgeys (result: 4 Pidgeys remaining and 12 candies) and finally evolve one more Pidgey. Your final count in this case would be 3 Pidgeys, no candies and 3 evolutions.

Given a certain number of pokemon, candy and an evolution cost, what is the maximum number of evolutions you can perform?

Input

The input consists of lines of input. Each line of input consists of three integers: the number of pokemon that you have, the number of candies and the evolution cost for the pokemon. All of the integers are greater than zero.

The last line of input has three zeroes. You should not process this line as input; it signals the end of the input.

Output

For each case, output “Pokemon X: Y evolutions, Z pokemon and W candies.” where X is the case number (starting at 0) and Y,Z, and W are integers, as described in the question. Do not worry about pluralization of words.

Sample Input	Sample Output
3 30 12 10 30 12 0 0 0	Pokemon 0: 2 evolutions, 1 pokemon and 8 candies. Pokemon 1: 3 evolutions, 3 pokemon and 1 candies.