The 2013 ACM ICPC Asia Regional Contest Dhaka Site

Sponsored by IBM

Hosted by North South University
Dhaka, Bangladesh

30th November 2013
You get 20 Pages
11 Problems
&
300 Minutes

acm International Collegiate Programming Contest

IBM event sponsor

Rules for ACM-ICPC 2013 Asia Regional Dhaka Site:

a) Solutions to problems submitted for judging are called runs. Each run is judged as accepted or rejected by the judge, and the team is notified of the results. **Submitted codes should not contain team or University name and the file name should not have any white space.**

b) Notification of accepted runs will **NOT** be suspended at the last one hour of the contest time to keep the final results secret. Notification of rejected runs will also continue until the end of the contest. But the teams will not be given any balloon and the public rank list will not be updated in the last one hour.

c) A contestant may submit a clarification request to judges. If the judges agree that an ambiguity or error exists, a clarification will be issued to all contestants.

d) Contestants are not to converse with anyone except members of their team and personnel designated by the organizing committee while seated at the team desk. **But they cannot even talk with their team members when they are walking around the contest floor to have food or any other purpose.** Systems support staff may advise contestants on system-related problems such as explaining system error messages. **Coaches should not try to enter the contest area under any circumstances.**

e) While the contest is scheduled for a particular time length (five hours), the contest director has the authority to alter the duration of the contest in the event of unforeseen difficulties. Should the contest duration be altered, every attempt will be made to notify contestants in a timely and uniform manner.

f) **A team may be disqualified by the Contest Director** for any activity that jeopardizes the contest such as dislodging extension cords, unauthorized modification of contest materials, distracting behavior of communicating with other teams.

g) Nine, ten or eleven problems will be posed. So far as possible, problems will avoid dependence on detailed knowledge of a particular applications area or particular contest language. Of these problems at least two will be solvable by a first year computer science student, another two will be solvable by a second year computer science student and rest will determine the winner.

h) Contestants will have foods available in their contest room during the contest. So they cannot leave the contest room during the contest without permission. **The contestants are not allowed to communicate with any contestant (Even contestants of his own team) or coach while are outside the contest floor.**

i) Team can bring up to **200 pages of printed materials** with them but they can also bring three additional books. But they are not allowed to bring calculators, mobile phones or any machine-readable devices like CD, DVD, Pen-drive, IPOD, MP3/MP4 players, floppy disks etc.

j) With the help of the volunteers, the contestants can have printouts of their codes for debugging purposes. **Passing of printed codes to other teams is strictly prohibited.**

k) The decision of the judges is final.

l) Teams should inform the volunteers if they don't get reply from the judges within 10 minutes of submission. Volunteers will inform the judges for further action. Teams should also notify the volunteers if they cannot log in into the PC^2 system. This sort of complains will not be entertained after the contest.

m) If you want to assume that judge data is weaker than what is stated, then do it at your own risk :).

A Falling Ants

Input: Standard Input
Output: Standard Output

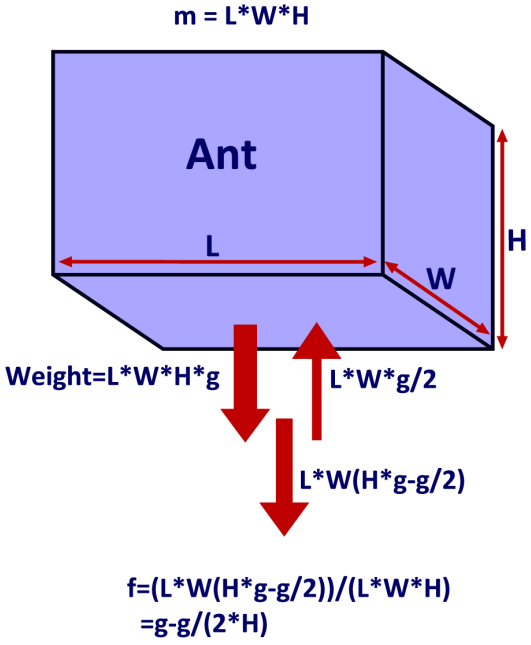
Ants make up 10% of the total world animal tissue. The total biomass of all the ants on Earth is roughly equal to the total biomass of all the people on Earth. However, unlike the people on Earth when they fall from a height they do not die for two reasons:

- They have so little mass relative to their air resistance that they fall slowly and, therefore, have little energy to dissipate when they hit the ground.

- Their bodies are tiny deformable tanks, well designed to absorb blows.

In general, small objects have less impact of gravitation on them because they have more surface area/volume compared to larger objects. For example consider a **(1x1x1)** cube. Its surface area is **6** and volume is **1**. So the ratio is **6:1** and for a **(10x10x10)** cube the surface area is **600** and volume is **1000**. So the ratio is **6:10**. Given the shape of many ants you will have to find out which ant has the highest effect of gravitation on it.

For simplicity we will assume the following things in this problem:



1. All ants are described as a box shaped object. A box shaped object is described with three integers **L**, **W**, and **H** which denotes the length, width and height of the object. So the volume of the ant is $(L \times W \times H)$.

2. The density (Mass per unit volume) is 1. So the mass of the above mentioned ant is also $(L \times W \times H)$ and so the weight is $(L \times W \times H) \times g$ (Here **g** is the acceleration caused by gravitation).

3. When an ant freely falls four sides are upright and so the faces at the top and bottom are parallel with the horizon. So the area of the plane facing bottom is always $L \times W$. For any ant the upward force put by the air is proportional to the area of the of the bottom face. To be specific it is $\frac{L \times W \times g}{2}$. After some manipulation it can be proved that the downward acceleration $f = g - \frac{g}{2H}$. So the downward acceleration actually depends solely on the value of **H** (as **g** is same for all ants).

Given the dimension of several ants, report the volume of the ant that has the highest downward acceleration. If there is a tie, report the one with the largest volume.

Input

The input file contains at most **500** test cases. The description of each test case is given below:

First line of each test case contains an integer **T (T ≤ 100)** which denotes the total number of ants to consider. Each of the next **T** lines contains three integers which denote the value of **L**, **W** and **H (1 ≤ L, W, H ≤ 50)** of an ant respectively.

Input is terminated by a line containing a single zero.

Output

For each set of input produce one line of output. This line contains the volume of the ant that has the highest downward acceleration. If there is a tie report the ant with the highest volume.

Sample Input	Output for Sample Input
3 3 4 5 12 1 5 20 10 4 3 3 4 5 20 30 5 1 2 4 0	60 3000

B

Game of MJ

Input: Standard Input
Output: Standard Output

M and **J** are playing a game. The description of the game is very simple. They have a few displays where several digits will be shown. There are buttons under each digit of every display. If someone presses the $\mathbf{i}^{th}$ button of a display once, the value of the $\mathbf{i}^{th}$ digit of that display will increase by one. Each of these displays has one fault in common. The $\mathbf{0}^{th}$ digit (least significant digit) can't be increased because its button is broken.

Each of these displays can be described using two parameters: **L** and **B**. **L** is the length of the display or the number of digits shown by the display. If **L** is **3** then the display can show **three (3)** digits side by side (we can consider it like a **3**-digit number). **B** is the base of the display. It is important in two aspects. Firstly, it limits the number of times you can press a button. Secondly, the number displayed in the display will be interpreted as a **B**-based number with **L** digits. Suppose **B** is **8** and **L** is **4**. Then the numbers shown by the display will be octal numbers and the length of the numbers will be **4**. Also you can press each of the **three** working (not broken) buttons at most **seven (7)** times. Example of such a display is shown in Figure 1.

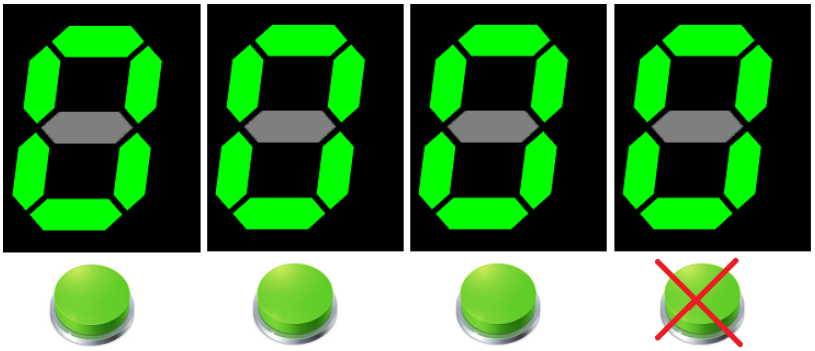


Figure 1 : A display with L = 4 and B = 8

M and **J** have **N** of these displays. Each of the display has its own **L** and **B**. Now the game **M** and **J** are playing goes like this:

1. **M** plays first, **J** plays second. After that they alternate moves.
2. In each move, a player selects a display. Then selects a digit. And presses its button. A digit can only be selected, if it hasn't reached its limit **B-1** already. A display can only be selected, if there is a digit which can be selected. However since the switch for the rightmost digit is already broken, you can not choose this button for any display.
3. After a move, if the summation of numbers (After converting it to decimal base) shown in the displays is divisible by **3**, then the player making that move loses and the other player wins.
4. If there is no move possible, then the game ends in draw.

Given description of **N** displays, you need to find out the outcome of the game. **M** and **J** both plays optimally. Initially every digit of every display is set to **0** (zero).

Input

First line of input consists of an integer **T (T ≤ 100)**, the number of test cases. Each test case starts with an integer **N (0 < N <10)**, the number of displays. Next **N** line each contains two integers, **L (0 < L < 10^9)** and **B (1 < B < 10^9)** which are the parameters that describes the **i^{th}** display.

Output

For each case print one line: **"Case X: S"**, where **X** is the case number. **S** is either **"M"**, **"J"** or **"Draw"** based on the outcome of the game in that case.

Sample Input	Output for Sample Input
3 1 5 2 2 3 2 2 6 1 2 2	Case 1: M Case 2: J Case 3: Draw

Explanation

Case 1:

00000 => 00010 => 01010 => 01110 => 11110 which is 30 in decimal and divisible by 3. So J loses and M wins. However this is one possible valid game sequence. But if two players play optimally J will always lose and thus M will always win.

Case 2:

(000,00) => (000,10) Sum = 0 + 6 = 6.

(000,00) => (100,00) => (100,10) => (110,10) Sum = 6 + 6 = 12.

(000,00) => (100,00) => (100,10) => (100,20) => (100,30) => (100,40) => (100,50) => (110,50) Sum = 6 + 30 = 36

In this way, in every game sequence M is doomed to reach such a configuration where the sum will be divisible by 3. Hence, in this scenario J will win and M will lose.

Case 3:

(00) => (10) which is 2 in decimal and not divisible by 3. There is no move left. So the game ends in a draw.

C Game of Throne

Input: Standard Input
Output: Standard Output

General Broken Arrow and Lieutenant General Shadow Coder are two great warriors of Gigaland. Every year they fought several wars together like ICPC, NCPC etc. The peoples of Gigaland were living happily until one day the king of Gigaland General Rem (RIP) died. Now both Broken Arrow and Shadow Coder wants to become the new king of Gigaland. As this is not possible for two persons to become king of a country at the same time, they leave it to the field of war. They decided to fight each other in the Great Chaos Jam (GCJ) field and the winner will become the new king.

Colonel Dragoon, friend of both Broken Arrow and Shadow Coder wants to avoid the blood shedding war and comes up with a great solution. According to his proposed solution the country will be divided into two disjoint sets of cities, called **A** and **B**. Broken Arrow will rule the cities in the set of **A** and Shadow Coder will rule the rest of the cities. But the people of Gigaland will get frustrated and angry with these divisions. To make peoples remain calm and happy Dragoon gave another proposal. They will build a sub-country of Gigaland called Megaland that consists of some cities and connecting roads of Gigaland so that peoples from set **A** and **B** can have some access to other cities. Though cities of set **A** and **B** are disjoint, cities might not be disjoint between Megaland and set **A** and between Megaland and set **B**. Broken Arrow and Shadow Coder both agreed with the idea of Dragoon.

The country of Gigaland consists of **N** (numbered from **1** to **N**) cities and **M** bidirectional roads each of which connects two different cities and each road has a cost to travel. Gigaland was built such a way that every city is connected to each other with sequence of one or more roads. Set **A** will consists of first **K (2 ≤ K ≤ min(N, 21))** cities, **1, 2, … , K** numbered cities of Gigaland and rest **N – K** cities will belong to the set **B**.

Though the warriors have agreed to build the new sub-country Megaland, they have some conditions. Each city of set **A** should be incident to **odd** number of roads of Megaland and each city of set **B** should be incident to **even** number (It can be zero also) of roads of Megaland. The cost of road system in Megaland should be minimum, that is sum of all the road cost of Megaland should be minimized. The cities of Megaland may or may not be connected to each other.

In terms of graph theory, you are given a weighted graph **G = (N, M)**, two set of nodes **A = {1, 2, …, K}** and **B = {K+1, K+2, …, N}**. You have to find the minimum cost sub-graph **S (subset of edges)**, where each node in **A** should have odd degree in **S** and each node in **B** should have even degree in **S**. Here cost of **S**, is sum of all the edge cost in **S**.

Given **N, M, K** and description of **M** roads of Gigaland, find the minimum possible cost of sub-country Megaland.

Input

Input starts with a positive integer, **T (T ≤ 250)** denoting the number of test cases. Each case starts with three integers, **N (2 ≤ N ≤ 100), M (N-1 ≤ M ≤ N*(N-1)/2)** and **K (2 ≤ K ≤ min(N, 21))**. Each of the next **M** lines contains three integers **u**, **v** and **c (1 ≤ u,v ≤ N, u ≠ v, 0 < c < 10000)** meaning that there is a bi-directional road between city **u** and **v** of cost **c**. No two roads will be same. Please also note that for **90%** of the test cases **K** is less than **20**.

Output

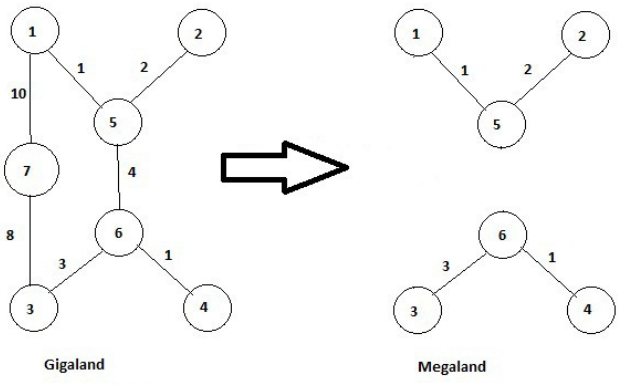
For each test case, print the test case number (starting from **1**) and the minimum possible cost of Megaland if it is possible to build the sub-country according to the conditions described above, otherwise print "Impossible" (without the quotes).

Sample Input	Output for Sample Input
2 7 7 4 1 7 10 1 5 1 2 5 2 3 6 3 4 6 1 5 6 4 3 7 8 4 4 3 1 2 1 1 3 1 2 4 1 3 4 1	Case 1: 7 Case 2: Impossible

Warning: The judge input file is around 2.5 MB. So use faster I/O functions.

Explanation

For test case 1,



In Megaland degree of each node in set A = {1,2,3,4} is 1 and degree nodes in set B = {5, 6, 7} are 2, 2 and 0 respectively.

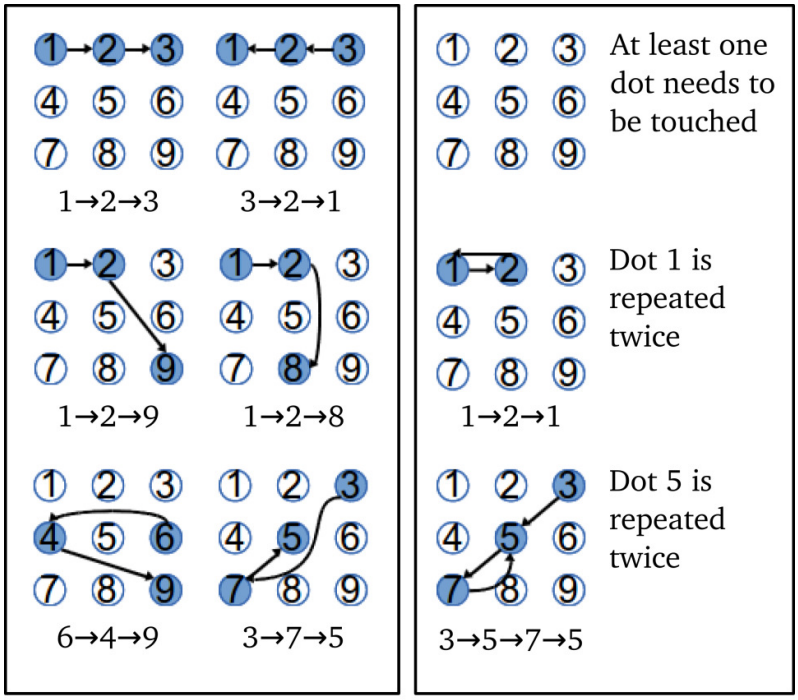
D Pattern Locker

Input: Standard Input
Output: Standard Output

Mr. Anderson is in a relationship with a very suspicious and jealous girlfriend. She is always checking Anderson's phone logs and texts to find out he is up to. Feeling that his private space is getting violated, he decided to put a pattern locker on his phone.

This pattern locker comes with a **9** dots arranged on a **3x3** square grid by default. One has to drag through several dots to record a pattern. Then to unlock the phone, one needs to replicate the same pattern recorded before. If we assume that each dot is assigned a unique number then a pattern is nothing more than a sequence of digits. The pattern locker requires that no digit appears in the sequence more than once.

Here is an example of such a pattern locker and some valid and invalid recorded pattern:



Valid patterns touching 3 dots on a 3x3 grid

Invalid patterns on a 3x3 grid

Even after recording a hard to crack pattern, Mr. Anderson doesn't feel quite comfortable. He is worried that his girlfriend might try all possible sequences to break his pattern. He wants to know how many different pattern sequences are possible for a given grid size, minimum and maximum numbers of dots recorded in patterns.

You have to help Mr. Anderson in counting the number of possible such sequences.

Input

The first line of the input will give the number of test cases, **T (1 ≤ T ≤ 10000)**. Then **T** test cases follow in separate lines. Each test case consists of three numbers **L**, **M**, **N** separated by a single space in between two numbers. The first number **L (1 < L ≤ 100)** denotes the number of rows and columns in the grid. The second number **M (1 ≤ M ≤ L*L)** denotes minimum number of dots to be included in a pattern and the third number **N (M ≤ N ≤ L*L)** denotes the maximum number of dots to be included in the pattern.

Output

For each test case, you need to print the test case number **X** in the format "Case X: ". This will be followed by the count of possible sequences for the given grid size, minimum and maximum number of dots in a sequence. Since the count can be pretty big, you need to print the value of the count modulo **10000000000007** (**1** followed by **12** zeros followed by **7**).

Sample Input	Output for Sample Input
2 3 4 9 3 1 9	Case 1: 985824 Case 2: 986409

E Pearl Chains

Input: Standard Input
Output: Standard Output

You have pearl of 3 types.

- Type **1** pearl can be of color from **1** to **X**.
- Type **2** pearl can be of color from **X+1** to **X+Y**
- Type **3** pearl can be of color from **X+Y+1** to **X+Y+Z**.

You have unlimited supply of pearls of each type and each color. You want to build some pearl chain. But you have **2** more additional constraints.

- The total number of Type **1** and Type **3** pearls will be exactly **A**.
- The total number of Type **2** and Type **3** pearls will be exactly **B**.

Given **A**, **B**, **X**, **Y** and **Z** calculate how many different pearl chains are possible. **2** chains are different if they have different length or there is a position in which they have different colored pearl.

Input

First line of the input contains **T (1 ≤ T ≤ 100)** the number of test cases. Each test case contains **5** integers **A**, **B**, **X**, **Y** and **Z**. All of these **5** integers are between **1** and 10^{17} inclusive.

Output

For each test case output an integer denoting the number of possible pearl chains. Since the result is too huge output the result modulo **1000003**.

Sample Input	Output for Sample Input
5 1 1 1 1 1 2 3 1 1 1 1 1 1 2 1 10 10 10 10 10 100 100 100 100 100	3 25 5 77069 329672

F Two Points Revisited

Input: Standard Input
Output: Standard Output

It's a common phenomenon to mix up direction when you go to a new place. Some people are so clever that they just look at sky and can tell you which one is north. And some dumb people try to remember the whole way they travelled: "First we were east facing, then we turned left, then right, then again right… and so this one is …" and thus end up with a direction. But the roads are not always in **90** degrees with each other and they do not always run from north to south or from east to west. So this kind of calculation ends up with some ridiculous answer almost all the times.

The city we are talking about is a square shaped one with vertices at **(0, 0)**, **(S, 0)**, **(S, S)** and **(0, S)**. Although the city is nice square shaped one, the road network is already messed up. But there is only one rail track and it is a straight one. So city corporation decides to build another rail track which is perpendicular to the existing one. They do not care if the new track intersects with the previous rail track or not, they just need location of two rail stations so that if a rail track is built along the straight line connecting these two stations the line would be perpendicular to the existing rail track. The problem is you have to build the stations in integer co-ordinates.

So to sum it up, you will be given co-ordinates of two stations on the already existing rail track, you have to give any two distinct integer co-ordinates within the city for building new rail track. One of the new co-ordinates may coincide with the given co-ordinates, if you want it to.

Input

First line of input will contain an integer **T**, the number of test cases (**$T \leq 15000$**). For each case there will be a line of four integers **X_1**, **Y_1**, **X_2**, **Y_2** (**$0 \leq X_1, Y_1, X_2, Y_2 \leq 10^9$**). Here **($X_1$, Y_1)** and **(X_2, Y_2)** are the co-ordinates of two stations on the already existing rail track.

You may assume that these two points are different and within the city.

Output

For each test case produce one line of output. This line contains the case number followed by four integers: where the first two integers are the x and y co-ordinate of the first station and the next two integers are the x and y co-ordinate of the second station. **They should be within the city for any possible value of S and the line passing through them must be perpendicular to the existing line (may not necessarily intersect the previous track within the city)!** Since there can be multiple answers, you may output any of them but you have to make sure that absolute value of any of these coordinates does not exceed 2×10^9. For details please go through the explanation of the sample cases.

Sample Input

```
2
4 4 5 5
9 0 10 0
```

Output for Sample Input

```
Case 1: 1 0 0 1
Case 2: 10 0 10 2
```

Explanation

First Sample:

In the first sample, the existing rail track goes through points **(4, 4)** and **(5, 5)**. If you establish two new rail stations at **(1, 0)** and **(0, 1)** the rail track passing through them will be perpendicular to the existing rail track. Note, if you output something like **(5, 5)** and **(6, 4)** you will get Wrong Answer, since if **S = 5**, point **(6, 4)** will not be within the city. But for any value of **S** such that **(4, 4)** and **(5, 5)** is within the city, points **(1, 0)** and **(0, 1)** will also be within the city.

Second Sample:

In this case, there are many correct answers. One of them is **(10, 0)** and **(10, 2)**. Note that, unlike first sample here the rail tracks intersect.

G Watching the Kangaroo

Input: Standard Input
Output: Standard Output

Day by day number of Kangaroos is decreasing just like tiger, whale or lions. So I decided to make them a sanctuary where they will live peacefully. I do not let visitors go near them. So I planted some display screen outside the sanctuary. For this problem, you may assume the sanctuary to be a long line of **1000000000** unit distance. The leftmost position is marked with **0** and the rightmost position with **1000000000**. There are at most **100000** cameras on this line. Each of the cameras is described with **(L, R)** which means that camera covers a range from position **L** to position **R** inclusive. Each of the cameras has their associated display screens outside where the visitors can see the Kangaroos.

Now for the convenience of the spectators we announce time to time: "Kangaroo appears in Screen 3", "Kangaroo appears in Screen 7" and so on. I have some employees who continuously monitor the position of Kangaroos and inform the system (here position is a marker). The system chooses the best screen to display that animal based on the coverage. The coverage of a screen for a position **x** is defined as follows:

If the position **x** is outside the range of that screen (i.e. **x < L** or **x > R**) then the coverage is **zero**. Otherwise the coverage is **minimum(x – L, R – x)**.

An example will make it clear:

Suppose there are four screens covering the range **(7, 15)**, **(14, 100)**, **(8, 10)** and **(1, 11)**. Now say one Kangaroo appears at **x = 10**.

First screen has coverage of **3** unit around **x = 10**. Because **x = 10** is within **(7, 15)** and **minimum(10 – 7, 15 – 10) = minimum(3, 5) = 3**.

Second screen has coverage of **0** unit around **x = 10**. Because **x = 10** does not belong to the range **(14, 100)**.

Third screen has coverage of **0** unit around **x = 10**. Because though **x = 10** is within **(8, 10)** but **minimum(10 – 8, 10 – 10) = 0**.

Fourth screen has coverage of **1** unit around **x = 10**. Because **x = 10** is within **(1, 11)** and **minimum(10 – 1, 11 – 10) = 1**.

So which one is better? Obviously the first one, as it has the maximum coverage.

So you are given the ranges of the screens and the positions the kangaroo appears. For each position of the Kangaroo you are to tell me the maximum coverage you can have with any of the screens.

Input

First line of the test file contains **T (T ≤ 3)**, number of test cases. Hence **T** cases follow. For each case you are given **N, M** in the first line; **N** is the number of screens and **M** is the number of Kangaroo appearance (**1 ≤ N, M ≤ 100000**). In the next **N** lines you are given the range of screens **L R** (**0 ≤ L ≤ R ≤ 1000000000**). Next **M** lines contain the position of the Kangaroo- an integer **x** (**0 ≤ x ≤ 1000000000**).

Output

For each case print the case number. Hence print **M** lines, **i**'th containing the maximum coverage you can have around **i**'th Kangaroo.

Sample Input	Output for Sample Input
1 3 2 7 15 14 100 1 11 10 120	Case 1: 3 0

Warning: The judge input file is around 6 MB. So use faster I/O functions.

H

GCD XOR

Input: Standard Input
Output: Standard Output

Given an integer **N**, find how many pairs **(A, B)** are there such that: **gcd (A, B) = A xor B** where **1 ≤ B ≤ A ≤ N**.

Here **gcd (A, B)** means the greatest common divisor of the numbers **A** and **B**. And **A xor B** is the value of the bitwise **xor** operation on the binary representation of **A** and **B**.

Input

The first line of the input contains an integer **T(T ≤ 10000)** denoting the number of test cases. The following **T** lines contain an integer **N(1 ≤ N ≤ 30000000)**.

Output

For each test case, print the case number first in the format, "**`Case X:`**" (here, **X** is the serial of the input) followed by a space and then the answer for that case.

Sample Input	Output for Sample Input
2 7 20000000	Case 1: 4 Case 2: 34866117

Explanation

Sample 1:

For **N=7**, there are four valid pairs: **(3, 2), (5, 4), (6, 4)** and **(7, 6).**

I Fiasco

Input: Standard Input
Output: Standard Output

Natasha always mixes up things in her algorithm classes. Last week Prof. Chhaya Murthy gave the class a classical problem - finding shortest path to all nodes from a specific node of a weighted graph. Before that in another class, the professor taught them Prim's and Dijkstra's algorithms which are similar looking but does very different things. Always confused, poor Natasha submitted something very similar to the Prim's algorithm (actually we should call it Natasha's algorithm), she didn't test it properly. Pseudocode of her algorithm looks as follows:

```
function Shortest(Graph, source):
   for each vertex v in Graph:              /* Initializations */
      visited[v] := false;
      dist[v]  := infinity ;                /* Distance function from source */
      previous[v]  := undefined ;           /* Previous node in optimal path */
      ans[v]  := undefined ;                /* Answer array */
   end for
    dist[source]  := 0 ;
   Q := {}                                  /* A priority queue which keeps (node, distance)
pairs. When pop is
called always gives the node with smallest distance. In case of a tie, returns the node with
smallest id. */
    Push (source, dist[source]) to Q
   while Q is not empty:                    /* The main loop */
      Pop node u from Q ;
      visited[u] := true;
      if dist[u] = infinity:
         break ;
      end if
      for each neighbor v of u:
         if visited[v] = true:
            continue;
          alt := edge_cost(u, v) ;          /* edge_cost(u, v) = cost of the edge between
vertices u and v */
         if alt < dist[v]:
            dist[v]  := alt ;
            previous[v]  := u ;
            Push (v, dist[v]) to Q;
         end if
      end for
   end while
   for each node node in the graph
      answer := 0
      u := node
      while previous[u] is defined:         /* Traverse the shortest path */
         answer := answer + edge_cost(u, previous[u]);
                                            /* edge_cost function as defined earlier */
         u := previous[u]
      end while ;
      ans[node] := answer;
   end for
   return ans;
endfunction
```

After submission, she showed the code to a friend who is good at algorithms. That friend immediately found the flaw. Natasha was really terrified and started to cry. Then there came the guys, the guys who liked her. They jumped at the opportunity and got hold of the dataset to be used to test the solutions. Your friend Rehan is one of those guys. He really wanted to impress Natasha. He asked you to change the dataset in such a way that Natasha's algorithm gives the right result. After the change, Rehan will somehow be able to put the modified data with the old timestamp. Rehan told you that:

1. The dataset has **T** test cases.
2. Each case contains a connected weighted undirected graph with **n** nodes and **m** edges.
3. Each case will contain a source node **source**.
4. Edge weights are positive and distinct.

To avoid suspicion, Rehan asked you not to change which vertices the edges connect, or the overall set of edge weights. You may only reorder which weights are assigned to which edges.

Input

The first line of the input denotes the number of datasets **T (1 ≤ T ≤ 15)**. **T** sets of case will follow. Each case will start with a triplet of numbers **n (2 ≤ n ≤ 2500)** , **m (1 ≤ m ≤ 25000)** and **source (1 ≤ source ≤ n)** - the number of nodes, the number of edges and the starting node respectively. Each of the next **m** lines will contain a triplet of numbers **(u, v, w)** meaning that there is an edge between node **u** and node **v** with weight **w (1 ≤ w ≤ m)**. Nodes are numbered from **1** to **n**. It is guaranteed that there is no duplicate or self-edges in the input and the graph is connected. In each given graph, all edge weights will be distinct.

Output

For each set of input, print one set of output. First line of a set should be of the format, "**`Case X:`**" where **X** is the case number. Then print each edge triplet - one triplet (**u, v** and **w**) in each line separated by a single space. The edges should be printed in the order given in the input. If there is more than one solution to a dataset, any one of them will do.

Sample Input

```
1
7 9 5
2 4 2
1 4 8
7 2 6
3 4 7
5 7 5
7 3 9
6 1 1
6 3 4
5 6 3
```

Output for Sample Input

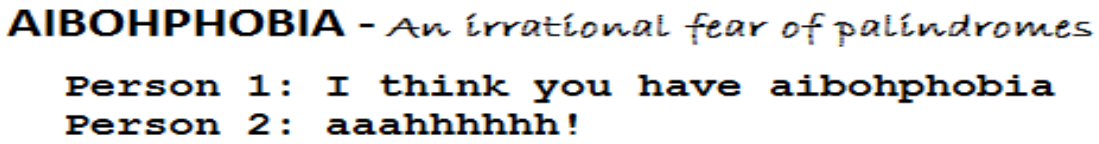
```
Case 1:
2 4 7
1 4 9
7 2 3
3 4 8
5 7 1
7 3 4
6 1 5
6 3 6
5 6 2
```

J Dromicpalin Substrings

Input: Standard Input
Output: Standard Output

Let's first define some terms:

- A string is *palindromic* if it reads the same forward and backward. Examples of palindromes are **madam** and **toot**.
- A string is a *dromicpalin* if we can rearrange its letters to make it a palindrome. An example of a *dromicpalin* string is **mmaad** because we can rearrange the letters to make it **madam**, which is a palindrome.
- A substring is any contiguous sequence of characters of a string. Some substrings of 'acmicpc' are {'a', 'c', 'i', 'icp', 'acmicpc'} but 'acpc' is not a substring. For this problem, we are not considering the empty substring, so that means there are $\frac{n(n+1)}{2}$ substrings of a string of length `n`.



Given a string, you have to figure out how many of its substrings are *dromicpalin*.

Input

The first line of input is an integer **T (T < 100)** indicating the number of test cases. Each case is a line containing a string. The strings will contain only lowercase letters **[a - z]**. The length of each string will be positive and not greater than **1000**.

Output

For each case, first output the case number followed by the number of substrings that are *dromicpalin*. Follow the samples for exact format.

Sample Input

```
4
acmicpc
aaaaa
isyoursolutionfastenough
abbababababbaba
```

Output for Sample Input

```
Case 1: 8
Case 2: 15
Case 3: 24
Case 4: 67
```

K

Fill the Cuboid

Input: Standard Input
Output: Standard Output

Can you fill an **(a*b*c)** cuboid with exactly **n** cubes? The cubes may touch, but cannot overlap and each cell inside the **(a*b*c)** cuboid should be covered by exactly one cube. You may use cubes of different dimensions and for each positive integer **p**, you can use as many **(p*p*p)** cubes as you like. For example, a **(2*2*3)** cuboid can be filled with **five** cubes: **four** **(1*1*1)** cubes and **one** **(2*2*2)** cube. It can also be filled with **twelve** **(1*1*1)** cubes.

To make this problem slightly more difficult, **m** of the cells in the cuboid are already filled and cannot be filled by another cube. Your job is to find out the possible values of **n** such that the remaining cells inside the cuboid, can be filled by exactly **n** cubes.

Input

There will be at most **300** test cases. Each test case begins with four integers **a, b, c, m (2 ≤ a, b, c ≤ 20, a*b*c ≤ 125, 0 ≤ m < a*b*c)**.

Each of the next **m** lines contains three integers **x, y, z (1 ≤ x ≤ a, 1 ≤ y ≤ b, 1 ≤ z ≤ c)**, that means the cell **(x, y, z)** is already filled.

No cell will be mentioned twice. There will be at least **one** non-filled cell.

The input is terminated by a line containing four zeroes.

Output

For each test case, print the case number and the list of possible answers in increasing order. There is a single space before each number in the output. Look at the output for sample input for details.

Sample Input

```
2 2 3 0
2 2 3 2
1 1 1
2 2 3
0 0 0 0
```

Output for Sample Input

```
Case 1: 5 12
Case 2: 10
```