The 2014 ACM ICPC
Asia Regional Contest
Dhaka Site

Sponsored by IBM

Hosted by BUBT
Dhaka, Bangladesh

6th December 2014
You get 18 Pages
10 Problems
&
300 Minutes

Rules for ACM-ICPC 2014 Asia Regional Dhaka Site Onsite Contest:

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.

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 be updated** in the last one hour.

c) A contestant may submit a clarification request to judges only through the CodeMarshal clarification system. If the judges agree that an ambiguity or error exists, a clarification will be issued to all contestants. Judges may prefer not to answer a clarification at all in which case that particular clarification request will be marked as IGNORED in the CodeMarshal clarification page.

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. **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 or judges may advise contestants on system-related problems such as explaining system error messages.

e) While the contest is scheduled for a particular time length (five hours), the contest director has the authority to alter the length 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 or communicating with other teams. The **judges on the contest floor** will report to the **Judging Director** about distracting behavior of any team. **The judges can also recommend penalizing a team with additional penalty minutes for their distracting behavior.**

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 one will be solvable by a first year computer science student, another one 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 explicit permission from the judges. **The contestants are not allowed to communicate with any contestant (even contestants of his own team) or coaches when they are outside the contest arena.**

i) Teams can bring up to **200 pages of printed materials** with them but they can also bring five additional books. But they are not allowed to bring calculators or any machine-readable devices like CD, DVD, Pen-drive, IPOD, MP3/MP4 players, floppy disks etc. **Mobile phone MUST be switched off at all times and stored inside a bag or any other place that is publicly non visible during the entire contest time. Failure to adherence to this clause under any condition will very likely lead to strict disciplinary retaliation and possible disqualification.**

j) With the help of the volunteers and judges, 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/judges if they don't get reply from the judges within 10 minutes of submission. Teams should also notify the volunteers if they cannot log in into the CodeMarshal system. These sort of complains will not be entertained after the contest.

A

Decoding Baby Boos

Input: Standard Input
Output: Standard Output

Osantu has been tagged as the best Bangladeshi contestant of present time by many and now he is mostly busy with the sweetest problem of his life - a baby boy. But Osantu often struggles to decode the sound that the baby makes throughout the year and so he asks for your help. He has converted the sound that the baby makes into a huge string. He thinks that sound made by the baby is often hard to understand because he replaces one character with another character. So in his diary he has prepared a list of replacement rules. Your job is to reverse these replacements and find the actual text that the baby wanted to say although in many cases the rules are not reversible.

Input

First line of the input file contains a positive integer **T** (**$T \le 6$**) which denotes how many test cases are there in the input file. The description of each test case is given below:

First line of each test case contains a non-empty string **S** (containing only uppercase characters and underscore). The length of this string can be up to **1000000**. Next line contains a positive integer **R** (**$R \le 10000$**), which denotes how many character replacement sequences follow. Each of the next **R** lines contains two characters **a_i** and **b_i** (both **a_i** and **b_i** are uppercase letters and separated by a single space) which denotes that while pronouncing the baby replaces character **a_i** with character **b_i**. As this replacement list is prepared by Osantu (who has short term memory) so the list can contain the same replacement rules twice, there can be cyclic rules like **'A'** is replaced with **'B'**, **'B'** is replaced with **'C'** and **'C'** is replaced with **'A'** and also there can be contradicting rules like **'A'** is replaced with **'B'** and **'A'** is replaced with **'C'** etc. So what you simply need to do is apply the reverse of those rules in the order they appear in the input although it may not seem logical.

Output

For each set of input produce one line of output. This line contains the string that is found by applying all the **R** replacement rules.

Sample Input	Output for Sample Input
2 AVVU_TUMI_COLING_PARO_NAY 3 B V D L H Y AABBCCY 3 A B B C C A	ABBU_TUMI_CODING_PARO_NAH CCCCBBY

Illustration of the 2nd sample input:

First replacement rule says the baby replaces 'A' with 'B'. So to reverse that rule all 'B' s are replaced with 'A'. So the string becomes "AAAACCY". The 2nd rule says 'B' is replaced with 'C' and so to reverse this rule we replace all 'C's with 'B' and so the string becomes "AAAABBY". The 3rd rule says that 'C' is replace with 'A' and so to reverse that we now replace all 'A's with 'C' and so the string finally becomes "CCCCBBY".

Warning: The input file size is around 6 MB, so you may want to use faster IO functions. If you are not intelligent enough, you algorithm may not be fast enough to solve this problem also.

B

And Or

Input: Standard Input
Output: Standard Output

Given A and B, $1 \le A \le B \le 10^{18}$, find the result of A | (A + 1) | (A + 2) | … | B and A & (A + 1) & (A + 2) & … & B.

| operator represents bitwise OR (inclusive)

& operator represents bitwise AND

Input

The first line of the input contains an integer **T(T ≤ 100000)** denoting the number of test cases. Each of the following **T** lines has two space separated integers **A** and **B**, **$1 \le A \le B \le 10^{18}$**

Output

For each input, print the output in the format, "`Case C: X Y`" (quote for clarity). here **C** is the case number starting from **1, X** is the result of bitwise (inclusive) **OR** of numbers from **A** to **B** inclusive and **Y** is the result of bitwise **AND** of numbers from **A** to **B**, inclusive.

For the exact input/output format please check the sample input/output section.

Sample Input

```
2
1 1
1 2
```

Output for Sample Input

```
Case 1: 1 1
Case 2: 3 0
```

Warning: The input file size is around 4 MB, so you may want to use faster IO functions.

Note:

| operator represents **bitwise OR**. A **bitwise OR** takes two bit patterns of equal length and performs the logical inclusive **OR** operation on each pair of corresponding bits. The result in each position is **1** if the first bit is **1** *or* the second bit is **1** *or* both bits are **1**; otherwise, the result is **0**. [Source: Wikipedia]

& operator represents **bitwise AND**. A **bitwise AND** takes two equal-length binary representations and performs the logical **AND** operation on each pair of the corresponding bits, by multiplying them. Thus, if both bits in the compared position are **1**, the bit in the resulting binary representation is **1 (1 × 1 = 1)**; otherwise, the result is **0 (1 × 0 = 0)**. [Source: Wikipedia]

C

A game for kids

Input: Standard Input
Output: Standard Output

Do you remember your childhood? You played many meaningless games. If you don't remember, you can look at any kid and instead of thinking "how can he play such a horrible boring game?", think "Did I used to play like this as well?". Watching the kids playing, you noticed one thing- those games are endless. For example, they keep throwing balls without any specific goal, in beach they build sand houses restlessly, or they may be cooking in their small kitchen-ware endlessly throughout the day. You could not tolerate all these meaningless endless games, rather you decided to give them some educational game!

You just came to a beach and drew **N** circles on the sand, connected all the circles with **N – 1** lines so that if any kid start from any circle and follow the connecting lines he can end up at any other circle. That means, if we consider the circles as vertices and connecting lines as edges the graph will be a **tree** in graph theoretic term.

You also attached two numbers with each circle. Suppose for the **i**'th circle the numbers are $\mathbf{L_i}$ and $\mathbf{H_i}$ ($\mathbf{L_i \leq H_i}$). Kids love marbles so in each circle you put $\mathbf{H_i}$ marbles. The marbles are colorful. In a circle all the marbles will be of the same color and any two marbles of the same color will always be in the same circle. So any two marbles belonging to the same circle are considered indistinguishable and two marbles from different circles are always distinguishable. Now the game starts: you will call a boy and describe him the rules. He can start from a circle, follow some connecting lines (may be none) and end up in a circle (may be same). However, the kid is not interested to go back through the same line he used coming into the circle he is in. In graph theoretic term such sequence of circles/connecting lines are called **simple path**. While going through the **i**'th circle he can take any number of marbles between $\mathbf{L_i}$ and $\mathbf{H_i}$. He only collects marble from a circle if that circle is part of his path. Now after walking through the path and collecting marbles, his task is to divide these collected marbles into maximum number of groups so that same colored marbles appear same number of times in all the groups. Every collected marbles should belong to exactly one group. The kid will mark this maximum number in a paper. Let's name this entire round as traversal. The kid is clever and will never do the same traversal twice. That is, either the path will be different or the number of marbles taken from a particular circle will differ. Please note, path from **circle_i** to **circle_j** will be considered same as a path from **circle_j** to **circle_i**. As you can imagine if there are many circles and the difference of the numbers $\mathbf{L_i}$ and $\mathbf{H_i}$ is big then this game is kind of endless but educative! We are interested in the number of traversals that will yield the maximum number of groups to be **g**.

For example, suppose the graph contains two circles **A (1, 2)** and **B (2, 2)**. The first number in the parenthesis is $\mathbf{L_i}$ and the second number is $\mathbf{H_i}$ for the corresponding circles. Let's assume they are connected with a line. Here a kid can make **5** different traversals. All these traversals are recorded in the following table. The circle name is given in the parenthesis.

Simple path	Chosen Numbers	Maximum number of groups
A	[1(A)]	1
A	[2(A)]	2
A-B	[1(A), 2(B)]	1
A-B	[2(A), 2(B)]	2
B	[2(B)]	2

So 1 group appears 2 times, 2 groups appear 3 times.

Input

The first line of the input contains **T** **(T ≤ 50)**, number of test cases. Hence **T** cases follow. Each case starts with a positive integer **N** **(N ≤ 10000)**, number of circles in this case. Hence following **N - 1** lines describe the connecting lines. Each line will contain two integers **u** and **v** **(1 ≤ u, v ≤ N)**, denoting which pair of circles is connected by this connecting line. You can assume that the connecting lines will form a valid tree. Then there will be **2** more lines of input for the case. First line will contain the **L** values and the second line will contain the corresponding **H** **(1 ≤ L ≤ H ≤ 50)** values. That is, **i**th **L** or **H** corresponds to the **i**th circle.

Output

For each case in the first line output: "`Case C:`" where **C** is the case number (starting from 1). Then **50** lines will follow. **g**'th of these lines (**1 ≤ g ≤ 50**) will be of the format: "**g: ans**" where **ans** is the number of different traversals having **g** as the answer. Since the **ans** may be big, please output modulo **21092013**. Since a circle will contain at most **50** marbles, so it is not possible to make more than **50** groups in a traversal.

Sample Input

```
2
2
1 2
1 2
2 2
5
1 2
2 3
3 4
4 5
4 4 4 4 4
4 4 4 4 4
```

Output for Sample Input*

```
Case 1:
1: 2
2: 3
… 48 more lines with answer 0…
Case 2:
1: 0
2: 0
3: 0
4: 15
… 46 more lines with answer 0…
```

*Please note, output for the sample input is truncated.

Warning: The input file size is around 8 MB, so you may want to use faster IO functions.

D — Flood in Gridland

Input: Standard Input
Output: Standard Output

The country of Gridland is very special. It is divided into **N** rows and **M** columns; and both of them are 1 indexed. The intersection cell of the **i-th** row and the **j-th** column will be denoted as **land(i, j). land(i, j)** is either a flat surface of height H_{ij} or does not have any land rather it contains only bottomless sea water. All heights are measured from sea level; if H_{ij} is negative then it means the surface of that land is under the sea level.

Gridland is very much prone to natural calamities. Some lands of the Gridland are so low that they become flood affected most of the time in a year. Again some land are so high that there is always fog in those lands. This has been a dream of the people from Gridland for many years that the height of the country will be adjusted to reduce effect of natural calamities. Government of Gridland decided to increase or decrease heights of all the lands such that heights of all the lands reside between **L** and **U**. But people of Gridland also wants to maximize the average height of the country. So government of Gridland decided to maximize the sum of the heights of all the lands.

To increase or decrease height government can perform two operations

1. Increase the heights of all the land of a row by 1.

Example 1:

	1	2	3	4
1	12	7	3	10
2	2	X	5	3
3	8	9	-11	0
4	-1	14	X	4

Operation 1 on row 4 $\Longrightarrow$

12	7	3	10
2	X	5	3
8	9	-11	0
0	15	X	5

2. Decrease the heights of all the lands of a column by 1.

Example 2:

	1	2	3	4
1	12	7	3	10
2	2	X	5	3
3	8	9	-11	0
4	0	15	X	5

Operation 2 on Column 2 $\Downarrow$

12	6	3	10
2	X	5	3
8	8	-11	0
0	14	X	5

As mentioned above some cell contains bottomless sea water, those cells will be denoted as **X**. No operation has any effect on those lands.

Given **N, M, L, U** and the height **H** of all the lands, use the operations described above to increase or decrease height of the lands such that the new heights of the lands are between the range **L** and **U** (inclusive) and sum of the heights of all the lands is the maximum. Help government of Gridland to solve the problem.

Input

First line of the input contains a positive integer **T (T ≤ 300)**, number of test cases. First line of each test case contains four integers **N, M, L** and **U (1 ≤ N, M ≤ 75, -1000 ≤ L ≤ U ≤ 1000)**. Each of the next **N** lines contains **M** integers **H_{ij} (-500 ≤ H_{ij} ≤ 500),** height of **land(i, j) or** character **X**, where **X** means that the cell has bottomless sea water.

Note: 90% of all the test cases contain **N, M ≤ 20.**

Output

First line of each test case contains test case number and "**Impossible**", if there is no way to change the heights of all the lands with the operations described above such that the new height of each land remains between **L** and **U**. Otherwise print the sum of the new heights of all the lands (except **X** marked lands). Following line contains **N** non-negative integers **R_i (R_i ≤ 1000000)**, where **R_i** is the number of operation **1** applied on the row **i** and the next line will contain **M** non-negative integers **C_j (C_j ≤ 1000000)**, where **C_j** is the number of operation **2** applied on column **j**.

If there is any solution for a test case, then there will always be a solution with **0 ≤ R_i ≤ 1000000** and **0 ≤ C_j ≤ 1000000**.

If there are multiple solutions, print any of them.

Sample Input	Output for Sample Input
4 3 3 0 1 -1 -1 0 0 X 2 1 1 2 2 2 5 10 1 4 3 2 1 1 5 10 X 2 2 7 7 1 7 7 7	Case 1: 7 2 0 0 0 0 1 Case 2: 36 15 17 10 9 Case 3: 0 0 0 Case 4: Impossible

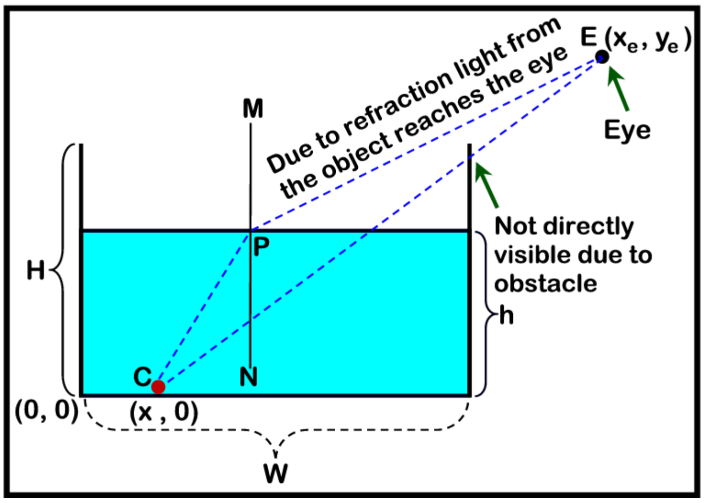
E

Refraction

Input: Standard Input
Output: Standard Output

We all know the common test of refraction of light with the visibility of a coin kept at the bottom of a bowl or cup. From a certain position the coin is not visible, but if we pour water into the bowl or cup it again becomes visible.

The empty bowl and the coin is not visible	The bowl is filled with water and a part of the coin is visible


Due to refraction light from the object reaches the eye
E (x_e, y_e)
Eye
M
H
P
Not directly visible due to obstacle
h
C
N
(0, 0)
(x , 0)
W

In this problem we will discuss A 2D version of this problem and it is shown in the figure on the left. We have a bowl made of opaque (not able to be seen through or not transparent) material. The coordinate of the lower left corner of the bowl is **(0, 0)**. At the bottom we have a point object at C whose coordinate is **(x, 0)**. The width of the bowl is **W** and the height is **H**. The refractive index of the transparent liquid (with respect to air) is **μ** (In the figure on the left $\mu = \frac{\sin\angle MPE}{\sin\angle CPN}$) and the location of the eye is **E** $(\mathbf{x_e}, \mathbf{y_e})$. Now if the bowl is empty then the object may or may not be visible (at the shown figure it is not visible) from the location of the eye, but as the bowl is filled with liquid so due to refraction, the light reflected from the object can reach the eye and it becomes visible. Now given the width and height of the bowl, abscissa of the object (ordinate is always zero), coordinate of the eye and refractive index **μ** of the liquid, you will have to find the minimum possible depth of the liquid (**h**) for which the object will be visible.

Input

First line of the input file contains an integer **T (0<T<100001)** which denotes the number of test cases. Each of the next **T** lines contains the input for one test case.

Each line contains **5** integers **W** $(100 \le W \le 1000)$, **H** $(100 \le H \le 1000)$, **x** $(1 \le x < W)$, x_e $(W < x_e \le 2000)$, y_e $(H < y_e \le 2000)$ and a floating-point number **μ** $(1.1000 \le \mu \le 5.0000)$. The meaning of these symbols are given in the problem statement.

Output

For each test case produce one line of output. This line contains a floating-point number h_{min}, which indicates the minimum depth of the liquid for which the object will be visible from the location of the eye. This number should always have four digits after the decimal point. If the object is not visible under any circumstances print the line `"Impossible"` instead. You can assume that inputs will be such so that small precision errors (absolute error of $\pm 3*10^{-6}$) will not create any difference in the output. Look at the output for sample input for details.

Sample Input

```
2
100 150 30 200 200 3.0000
100 150 30 200 200 2.0000
```

Output for Sample Input

```
136.2840
Impossible
```

Warning: The input file size is around 3 MB, so you may want to use faster IO functions.

F Reverse Polish Notation

Input: Standard Input
Output: Standard Output

It is a bit complicated to describe what is “Reverse polish notation” (in short RPN). If you already knew about RPN hopefully you can recall them after looking at some examples:

Normal Form	Reverse Polish Notation
1 + 2	1 2 +
(1 + 3) * 6	1 3 + 6 *
(1 + 3) * (7 – 2)	1 2 + 7 2 - *

If you still do not remember please go through the hint section for some detailed explanation of RPN.

For this problem we will consider only one variable **a** and only one binary operator **+**. So some valid RPN will be:

a which means: **a**
aa+ which means: **a + a**
aa+a+ which means: **(a + a) + a**
aaa++ which means: **a + (a + a)**

And some invalid RPN can be: **aaa**, **+aa**, **a+a** etc.

It might seem a bit confusing since all the variables are same, so you can not map which variable went where but for our problem it does not matter. We only care if the RPN is valid or not.

You will be given an RPN, you need to make it valid in minimum number of moves. In a move you can do one of the following operations:

1. Insert one **a** at any place you wish.
2. Insert one **+** at any place you wish.
3. Swap any two adjacent characters in the RPN.

You can apply the operations at any order you wish, that means, you can apply operation 3 with some newly added character by operation 1 or 2.

Input

First line of the input is number of test cases **T (1 ≤ T ≤ 3000)**. Hence **T** lines follow each containing **RPN** for the case. The length of the **RPN** will be at least **1** and at most **100000**. The **RPN** will be consisted of only **+** and **a**. Sum of the lengths of all the input strings will not exceed **3000000**.

Output

For each test case output “`Case C: ans`” where **C** is the case number and **ans** is the answer for the corresponding case.

Sample Input

```
4
a
a+a
+aa
aa++++a
```

Output for Sample Input

```
Case 1: 0
Case 2: 1
Case 3: 2
Case 4: 3
```

Hint:
Copied from: http://www-stone.ch.cam.ac.uk/documentation/rrf/rpn.html

Reverse Polish Notation is a way of expressing arithmetic expressions that avoids the use of brackets to define priorities for evaluation of operators. In ordinary notation, one might write

(3 + 5) * (7 – 2)

and the brackets tell us that we have to add 3 to 5, then subtract 2 from 7, and multiply the two results together. In RPN, the numbers and operators are listed one after another, and an operator always acts on the most recent numbers in the list. The numbers can be thought of as forming a stack, like a pile of plates. The most recent number goes on the top of the stack. An operator takes the appropriate number of arguments from the top of the stack and replaces them by the result of the operation.

In this notation the above expression would be

3 5 + 7 2 – *

Reading from left to right, this is interpreted as follows:

1. Push 3 onto the stack.
2. Push 5 onto the stack. Reading from the bottom, the stack now contains (3, 5).
3. Apply the + operation: take the top two numbers off the stack, add them together, and put the result back on the stack. The stack now contains just the number 8.
4. Push 7 onto the stack.
5. Push 2 onto the stack. It now contains (8, 7, 2).
6. Apply the – operation: take the top two numbers off the stack, subtract the top one from the one below, and put the result back on the stack. The stack now contains (8, 5).

Apply the * operation: take the top two numbers off the stack, multiply them together, and put the result back on the stack. The stack now contains just the number 40.

G Just Some Permutations

Input: Standard Input
Output: Standard Output

N people are invited to a dinner party and they are sitting on a round table. Each person is sitting on a chair and there are exactly **N** chairs. So each person has exactly two neighboring chairs, one on the left and the other on the right. The host decides to shuffle the sitting arrangements. A person will be happy with the new arrangement if he can sit on his initial chair or on any of his initial neighboring chairs.

Your job is to count number of different sitting arrangements such that at least **K** people are happy. Two arrangements are considered different, if there is at least one person sitting on a different chair in the arrangements.

Input

First line of the input contains **T** **(**$1 \le T \le 50$**)** which is the number of test cases. Each of the following **T** lines contain two space separated integers **N** **(**$3 \le N \le 2000$**)** and **K** **(**$0 \le K \le N$**).**

Output

Output the case number, followed by the number of different arrangements. Output the result modulo **1000000007**.

Sample Input

```
3
4 4
4 2
500 250
```

Output for Sample Input

```
Case 1: 9
Case 2: 23
Case 3: 880811602
```

For the 1st case, the 9 possible arrangements are {1, 2, 3, 4}, {1, 2, 4, 3}, {1, 3, 2, 4}, {2, 1, 3, 4}, {2, 1, 4, 3}, {2, 3, 4, 1}, {4, 1, 2, 3}, {4, 2, 3, 1} and {4, 3, 2, 1}.

H

Load Balancing

Input: Standard Input
Output: Standard Output

The infamous University of Kala Jadu (UKJ) have been operating underground for the last fourteen centuries training very select few students the dangerous art of black magic. However, with the recent trend of going digital, they too wanted to try out a bit of public exposure by enrolling students in their new distance education program.

Within the first three semesters they have got **6,789** students enrolled. The forkaclone spell allows them to clone their teachers to train at most **10,000** students in a running semester. But they do not have the server capacity to handle **10,000** students to register for their courses right at the beginning of the semester during their 4-day registration period. Sadly, their art of black magic (or kala jadu, as they say), only works on humans, it cannot be extended to their web server running on a Pentium IV machine.

UKJ server administrators realized that if they could split the load on the server and balance it somehow, then they can still handle **10,000** students per semester. Their idea is to divide the students into roughly **4** equal groups **A**, **B**, **C** and **D**. Each group would then be given one day to register; no other group can register on that same day – they will get their turn. They wanted to use total number of credits completed by a student as the deciding factor to assign students to the **4** different groups. As the students who would register can have completed any integer number of credits between **0** to **160**, one easy group assignment would be:

0 – 40 credits completed: group **A**
41 – 80 credits completed: group **B**
81 – 120 credits completed: group **C**
121 – 160 credits completed: group **D**

A bit of analysis of the number of students that may fall in these groups revealed that the number of students in each group vary greatly. So this particular idea of splitting students into **4** groups to balance the server load does not quite work out.

UKJ seeks your help in finding the credit boundaries that can create an optimal distribution of students so that each group roughly have the same number of students. You'd suggest three integers **a**, **b** and **c** to distribute the students as follows:

0 – a credits completed: group **A**
a+1 – b credits completed: group **B**
b+1 – c credits completed: group **C**
c+1 – 160 credits completed: group **D**

If the total number of students is **N**, the best possible scenario would place **N/4** students in each group. You need to minimize the sum of difference, **d** between **N/4** and the number of student you place in each group. For example, given **N = 8** students to distribute, if you divide them into a group of **3, 0, 3, 2** students then the difference with **N/4** for the groups would be **1, 2, 1, 0** respectively. This results in **1 + 2 + 1 + 0 = 4** as sum of differences. This is what you'd have to minimize. Note that, **N/4** can be a floating point number.

Input

The input description for the problem starts with **T** **(1 < T ≤ 100)** – the number of test cases, then **T** test cases follow. The first line of each case starts with the number of students **N** **(0 < N ≤ 10000)**. The next **N** line contains the number of credits (always integer), $\mathbf{C_i}$ **(0 ≤** $\mathbf{C_i}$ **≤ 160)** the $\mathbf{i}^{\text{th}}$ student have completed prior to this registration.

Output

Output for each test case will start with the test case label (starting with **1**, and formatted as shown in sample output.) The label will be followed by three integers, **a**, **b** and **c** **(0 ≤ a < b < c < 160)** denoting the group boundaries as described in the problem. If there are multiple such boundaries possible with the same **d** value, then pick the solution that has the smallest **a** value. If there is a tie, then pick the one with the smallest **b** value. If even that fails to break the tie, then pick the solution with the smallest **c** value.

Sample Input

```
2
8
0
40
41
80
85
120
150
155
9
0
40
41
80
85
120
121
150
155
```

Output for Sample Input

```
Case 1: 40 80 120
Case 2: 40 80 120
```

I Volume of Revolution

Input: Standard Input
Output: Standard Output

Revolving a profile curve around an axis can produce many real life objects. The chess pawn piece and the wine glass in the following figures are two such examples:

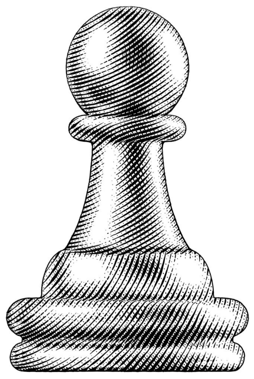
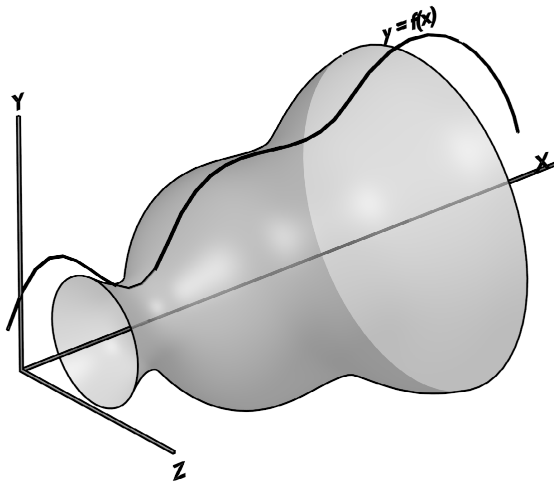
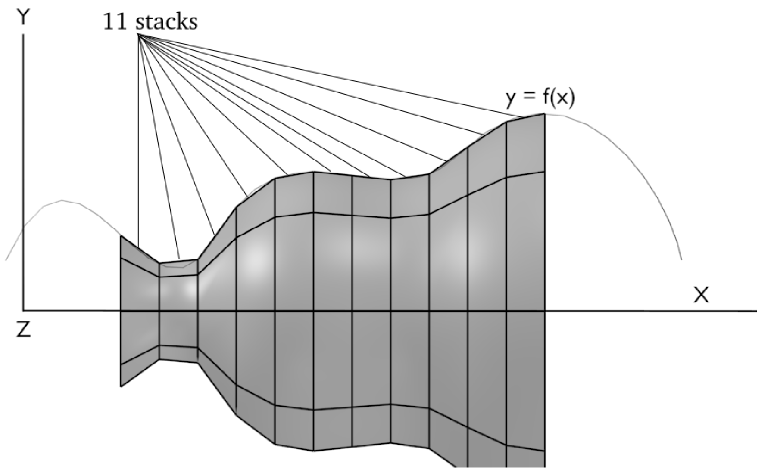
	

Figure 1a) Chess pawn	Figure 1c) Generating a 3D object by revolving a profile curve y = f(x) around the x-axis
	
Figure 1b) Wine glass	Figure 1d) Side view of a polygon mesh generated by revolving y = f(x) using 8 slices and 11 stacks

To render such objects in Computer Graphics applications (games, animations, etc.) we build polygon meshes to approximate the geometry of the object. One way to build such a mesh is to divide the object into a number of slices (segments around the **x** axis) and a number of stacks (segments along the **x** axis). While low polygon count for the mesh results in faster render time, they lack in fidelity. Figure 1d and 2b shows the effect of varying the number of stacks and slices to generate such polygon meshes:

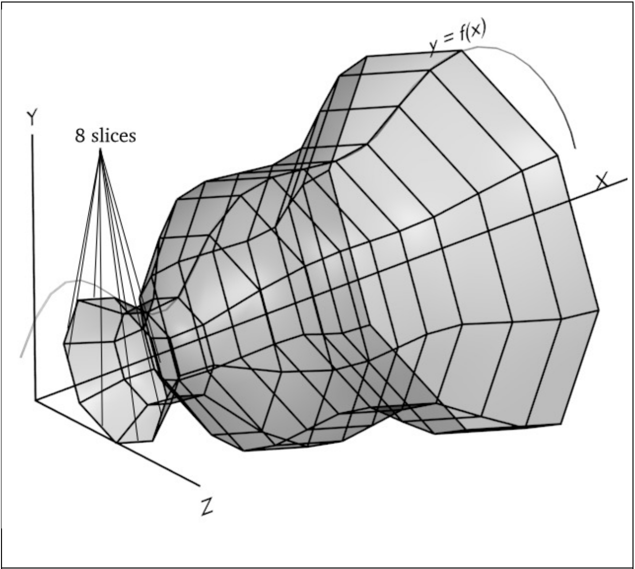


Figure 2a) Same polygon mesh viewed from a different angle

These polygon meshes are often good enough to pass the human eye for a real object as we would employ other faking techniques like per vertex normal computation, texture mapping, etc. However, none of those are relevant for this problem. Here we are only interested to know how "good" a polygon mesh is in terms of volume encapsulated when we revolve a profile curve within a given range. **Our metric of "goodness" is the volume of the revolution.**

The actual volume of the revolution can be found by using a simple integration formula:

$$V = \int_a^b A(x)\,dx$$

where **A(x)** is the area of the circle that we get by taking a cross section of the solid at a given **x** value. This formula gives us the volume of the object bounded by **x = a** and **x = b** planes.

Now the volume of the approximated polygon mesh can be computed too if we are told how many slices and how many stacks we want to divide the solid into. For the sake of simplicity, in this problem we will use polynomials as profile curves. Our polygon mesh will have equally spaced stacks and equal slices.The output you need to print is the true relative error as a percentage found by the formula $\left|\frac{True\,Value - Approximate\,Value}{True\,Value}\right| \times 100.$

Input

The input description for the problem will start with **T (1 < T ≤ 10000)** – the number of test cases. Each of the following T lines of the input will describe a test case.. The first number in each line will give you the value of **n (0 < n < 6)** – the degree of the polynomial. Then the next **n+1** values (separated by white spaces) will give you the coefficients. The coefficients will be listed in the decreasing order of power and each of these coefficients will be in the range **[-10, +10]**. Following the polynomial description you'll have two numbers giving the value of ***x*** **= a** and ***x*** **= b (-10 ≤ a < b ≤ +10)** on the same line. The last two numbers of each line will give you the number of slices **(3 ≤ slices ≤ 100)** and number of stacks **(0 < stacks ≤ 100)** to use in the polygon mesh. You can assume that the curve corresponding to the polynomial equation does not cross the **x**-axis for the range **[a, b]**.

Output

Output for each test case will start with the test case label (starting with **1**, and formatted as shown in sample output.) The label will be followed by true relative error in volume computation as a percentage. You need to round off the result to four digits after the decimal point. You can assume that inputs will be such that small precision errors will not cause difference in the output values.

Sample Input

```
2
2 1 -4 5 1 4 4 3
1 1 0 1 4 4 3
```

Output for Sample Input

```
Case 1: 27.9042
Case 2: 36.3380
```

J	Maximum Score **Input:** Standard Input **Output:** Standard Output

Maximum Score

Ron likes to play with integers. Recently he is interested in a game where some integers are given and he is allowed to permute them. His **point** will be calculated from the permutation made by him. Ron knows that he will get as many candies as his **point**, so he wants to permute the numbers to maximize his **point**.

Say, Ron has got **n** integers $\{x_1, x_2, \ldots, x_n\}$ and $(x_{i1}, x_{i2}, \ldots, x_{in})$ is the permutation made by him. His **point** will be the sum of the **score** of all integers. ***Score*** of an individual number x_{iw} in that permutation is calculated by the length of the longest subsequence (Let us consider $x_{j1}, x_{j2}, \ldots, x_{jm}$ as the subsequence where $1 \le j_1 < j_2 < \ldots < j_m \le n$) you can form with the following constraints:
1. There exists an integer **k** such that $1 \le k \le m$ and $j_k = i_w$.
2. $x_{j1} \le x_{j2} \le \ldots \le x_{jk-1} \le x_{jk} \ge x_{jk+1} \ge \ldots \ge x_{jm-1} \ge x_{jm}$.

Therefore, the ***score*** of x_{iw} in that permutation will be **m**. Say, **(1, 4, 3)** is a permutation made by Ron using the numbers **{1, 3, 4}**. For this permutation, score of **1** is **1** with subsequence **(1)**, score of **4** is **3** with subsequence **(1, 4, 3)** and score of **3** is **2** with subsequence **(1, 3).** So, Ron's point is **6** for this permutation.

Ron is not sure how to achieve the maximum **point** and he is also wondering about the number of different permutations which generate that maximum value of **point**. You need to help Ron to calculate these two values. A permutation $(x_1, x_2, \ldots, x_n)$ is different from another permutation $(y_1, y_2, \ldots, y_n)$ if there exists an integer i such that $1 \le i \le n$ and x_i is not equal to y_i.

Input

The first line of input contains a single integer **T** ($1 \le T \le 200$), which denotes the number of test cases to follow. For each test case, there will be two lines of input. The first line contains a single integer, **p** ($1 \le p \le 10^5$). The second line contains **p** pairs of integers. In each pair, there are two integers v_i and f_i ($1 \le v_i, f_i \le 10^5$) which indicate that the value v_i is present f_i times among the given numbers. Therefore, $f_1 + f_2 + \ldots + f_p = n$, where **n** is the total number of integers given to Ron. All the values of v_i will be distinct.

Output

For each case, in a separate line, print the case number and the maximum sum of scores and the number of permutations to achieve that sum of scores. As the number of permutations can be quite large, print it modulo **1000000007** (10^9+7). Follow Sample Input and Output for details. The value of the maximum sum of scores will fit in **64**-bit unsigned integer.

Sample Input

```
2
2
121 1 22 1
2
71 2 35 1
```

Output for Sample Input

```
Case 1: 3 2
Case 2: 7 2
```