Welcome to the 2017 Regional Programming Contest. Before you start the contest, please be aware of the following notes:

The Contest

1. There are twelve (12) problems in the packet, using letters A-L. These problems are NOT necessarily sorted by difficulty. As a team's solution is judged correct, the team will be awarded a balloon. The balloon colors are as follows:

Problem	Problem Name	Balloon Color
A	Proper Binary Tree	Purple
B	Enemy of my Enemy	Green
C	Pixel Virus	Blue
D	Primal Encryption	Pink
E	Stars	Brown
F	Optimization	Red
G	Stone	Yellow
H	Sum Square	Orange
I	Hyperloop	Black
J	Anti Hash II	Cyan
K	Sigma Function (II)	White
L	Squares in a Regular Pentagon	Gold

2. Solutions for problems submitted for judging are called runs. Each run will be judged. The judges will respond to your submission with one of the following responses. In the event that more than one response is applicable, the judges may respond with any of the applicable responses.

Response	Explanation
Yes	Your submission has been judged correct.
Wrong	Answer Your submission generated output that is not correct.

Output Format Error	Your submission's output is not in the correct format or is misspelled.
Incomplete Output	Your submission did not produce all of the required output.
Excessive Output	Your submission generated output in addition to or instead of what is required.
Compilation Error	Your submission failed to compile.
Run-Time Error	Your submission experienced a run-time error.
Time-Limit Exceeded	Your submission did not solve the judges' test data within 60 seconds.
Other-Contact Staff	Contact your local site judge for clarification.

3. A team's score is based on the number of problems they solve and penalty points, which reflect the amount of time and number of incorrect submissions made before the problem is solved. For each problem solved correctly, penalty points are charged equal to the time at which the problem was solved plus 20 minutes for each incorrect submission. No penalty points are added for problems that are never solved. Teams are ranked first by the number of problems solved and then by the fewest penalty points.

4. This problem set contains sample input and output for each problem. However, the judges will test your submission against longer and more complex datasets, which will not be revealed until after the contest. Your major challenge is designing other input sets for yourself so that you may fully test your program before submitting your run. Should you receive a judgment stating that your submission was incorrect, you should consider what other datasets you could design to further evaluate your program.

5. In the event that you feel a problem statement is ambiguous or incorrect, you may request a clarification. Read the problem carefully before requesting a clarification.
   If a clarification is issued during the contest, it will be broadcast to all teams.
   If the judges believe that the problem statement is sufficiently clear, you will receive the response, "No response, read problem statement."
   If you receive this response, you should read the problem description more carefully. If you still feel there is an ambiguity, you will have to be more specific or descriptive of the ambiguity you have found.
   You may not submit clarification requests asking for the correct output for inputs that you provide, e.g., "What would the correct output be for the input ...?" Determining that is your job unless the problem description is truly ambiguous. Sample inputs may be useful in explaining the nature of a perceived ambiguity, e.g., "There is no statement about the desired order of outputs. Given the input: . . . , would both this: . . . and this: . . . be valid outputs?".

6. Runs for each particular problem will be judged in the order they are received. However, it is possible that runs for different problems may be judged out of order. For example, you may submit a run for problem B followed by a run for problem C, but receive the response for C first.
   Do not request clarifications on when a response will be returned. If you have not received a response for a run within 30 minutes of submitting it, **you may have a runner ask the local site judge to determine the cause of the delay**. Under no circumstances should you ever submit a clarification request about a submission for which you have not received a judgment.
   If, due to unforeseen circumstances, judging for one or more problems begins to lag more than 30 minutes behind submissions, a clarification announcement will be issued to all teams.
   This announcement will include a change to the 30 minute time period that teams are expected to wait before consulting the site judge.

7. The submission of abusive programs or clarification requests to the judges will be considered grounds for immediate disqualification.

8. The submission of code deliberately designed to delay, crash, or otherwise negatively affect the contest itself will be considered grounds for immediate disqualification.

Your Programs

9. All solutions must read from standard input and write to standard output. In C this is ***scanf/printf***, in C++ this is ***cin/cout***, and in Java this is ***System.in/System.out***. The judges will ignore all output sent to standard error (***cerr*** in C++ or ***System.err*** in Java). You may wish to use standard error to output debugging information. From your workstation you may test your program with an input file by redirecting input from a file:

   program < file.in

10. Unless otherwise specified, all lines of program output
    - must be left justified, with no leading blank spaces prior to the first non-blank character on that line,
    - must end with the appropriate line terminator (\n, endl, or println()), and
    - must not contain any blank characters at the end of the line, between the final specified output and the line terminator.

    You must not print extra lines of output, even if empty, that are not specifically required by the problem statement.

11. Unless otherwise specified, all numbers in your output should begin with the minus sign (-) if negative, followed immediately by 1 or more decimal digits. If the number being printed is a floating point number, then the decimal point should appear, followed by the appropriate number of decimal digits. For output of real numbers, the number of digits after the decimal point will be specified in the problem description (as the "precision").
    All floating point numbers printed to a given precision should be rounded to the nearest value. For example, if 2 decimal digits of precision is requested, then 0.0152 would be printed as "0.02" but 0.0149 would be printed as "0.01".
    In simpler terms, neither scientific notation nor commas will be used for numbers, and you should ensure that you use a printing technique that rounds to the appropriate precision.

12. All input sets used by the judges will follow the input format specification found in the problem description. You do not need to test for input that violates the input format specified in the problem.

13. All lines of program input will end with the appropriate line terminator (e.g., a linefeed on Unix/Linux systems, a carriage return-linefeed pair on Windows systems).

14. If a problem specifies that an input is a floating point number, the input will be presented according to the rules stipulated above for output of real numbers, except that decimal points and the following digits may be omitted for numbers with no non-zero decimal portion. Scientific notation will not be used in input sets unless a problem explicitly allows it.

15. Every effort has been made to ensure that the compilers and run-time environments used by the judges are as similar as possible to those that you will use in developing your code. With

that said, some differences may exist. It is, in general, your responsibility to write your code in a portable manner compliant with the rules and standards of the programming language. You should not rely upon undocumented and non-standard behaviors.

(a) One place where differences are likely to arise is in the size of the various numeric types. Many problems will specify minimum and maximum values for numeric inputs and outputs. You should write your code with the understanding that, on the judges' machines:

- A C++ int, a C++ long, and a Java int are all 32-bits wide.
- A C++ long long and a Java long are 64-bits wide.
- A float in both languages is a 32-bit value capable of holding 6-7 decimal digits, though many library functions will be less accurate.
- A double in both languages is a 64-bit value capable of holding 15-16 decimal digits, though many library functions will be less accurate.

The data types on your own machines may differ in size from these, but if you follow the guidelines above in choosing the types to hold your numbers, you can be assured that they will suffice to hold those values on the judges' machines.

(b) Another common source of non-portability is in C/C++ library structures. Although the C & C++ standards are very explicit about which header file must declare certain std symbols, the standards do not prohibit other headers from duplicating or loading extra symbols.

For example, if your program uses both ***cout*** and ***ifstream***, you might find that your code compiles if you only ***#include*** <***fstream***>, because, as it happens, on your machine the ***fstream*** header ***#includes*** the ***iostream*** header where ***cout*** is properly declared. However, you cannot rely upon the judges' machines having libraries with the same structure. So it is your responsibility to ***#include*** the appropriate headers for whatever std library features you use.

Good luck, and HAVE FUN!!!

Problem A: Proper Binary Tree

Run Time Limit: 3 sec

Have you heard the name of Pirate Captain Jack Dumb? He is the most wanted blacklisted pirate who lives in Black Iceland. Recently he has learned about proper binary tree (a binary tree where every node other than leaves has two children, also known as full binary tree). With his little knowledge in C++ he wrote code that can display proper binary tree with height 1, 2 and 3.

C++ code	Output
`cout<<"o--|__"<<endl;` `cout<<"o--|"<<endl;`	`o--|__` `o--|`

`cout<<"o--|__"<<endl;`
`cout<<"o--|  |__"<<endl;`
`cout<<"o--|__|"<<endl;`
`cout<<"o--|"<<endl;`
	`o--|__`
	`o--|  |__`
	`o--|__|`
	`o--|`

`cout<<"o--|__"<<endl;`
`cout<<"o--|  |__"<<endl;`
`cout<<"o--|__|  |"<<endl;`
`cout<<"o--|  |__"<<endl;`
`cout<<"o--|__  |"<<endl;`
`cout<<"o--|  |__|"<<endl;`
`cout<<"o--|__|"<<endl;`
`cout<<"o--|"<<endl;`
	`o--|__`
	`o--|  |__`
	`o--|__|  |`
	`o--|  |__`
	`o--|__  |`
	`o--|  |__|`
	`o--|__|`
	`o--|`

Figure: Proper/full binary tree with height 1,2 and 3

It's the night before ACM ICPC Regional Contest. Festive mode is going on amongst the organizer, participants and spectators. Rumours in the air-"Pirate Captain was seen in this city few hours ago". Just at that time, Pirate Captain Jack Dumb broke into the judge's room and hijacked one of your balloons. But Pirate Captain promised that, he will give back your balloon during contest if you can draw a proper/full binary tree with height **N** for him.

Input

The input begins with a single positive integer **T(≤13)** on a line by itself indicating the number of the cases. Each of the next **T** lines contains an integer **N(1≤ N ≤13)** denoting the required height of proper binary tree.

Output

For each test case, print the test case number in a single line followed by desired proper binary tree. You should follow the exact format as sample input/output. Note that, trailing spaces are not allowed.

Sample Input	Sample Output
2 1 2	Case 1: o--|__ o--| Case 2: o--|__ o--| |__ o--|__| o--|

Note That:

C++ code for N=1 contains:

```
cout<<"o--|__"<<endl;
cout<<"o--|"<<endl;
```

- Leaf is represented by 'o' (15th smallest lowercase Latin letter) on each line.
- Two '-' (dash, ASCII value 45) holds each leaf on each line.
- One '|' (Vertical bar, ASCII value 124) on each line connects them followed by two '_' (underscore, ASCII value 95) in first line.

NB: Output Set can be large, for safety use faster I/O.

Problem B: Enemy of my Enemy

Run Time Limit: 3 sec

People typically have some others they would consider friends and some they don't consider enemies. Both the friend relation and the enemy relation are symmetric. If B is a friend of A then A is a friend of B, and if B is an enemy of A then A is an enemy of B.
However, while the friend relation is transitive, the enemy relation is antitransitive. If B is a friend of A and C is a friend of B, then C is a friend of A. However, if B is an enemy of A and C is an enemy of B, then C is a friend of A rather than an enemy. Finally, each person is an enemy of all their enemy's friends, so if B is a friend of A and C is an enemy of B, then C is an enemy of A.

Input

The first line contains integer T(1<=T<=20) which is the number of test cases.
Each test case starts with a line containing two integers separated by a single space, n (1<=n<=50) representing the number of pairs of people who are known to be friends and, m (1<=m<=50) representing the number of pairs who are known to be enemies. And this will be followed by n pairs of friends and m pairs of enemies.
Then this will be followed by a list of queries, each giving the names of two people. The pair "* *" will be the end of the queries.

Output

For each query in each test case, you have to report whether the given pairs are friends, enemies or if it is impossible to tell using the rules above. If the given pair is friends, print "FRIENDS". If they are enemies, print "ENEMIES". If they are the same person, print "SAME_PERSON". If it is impossible to tell, print "IMPOSSIBLE_TO_TELL". A blank line will be followed at the end of the output for each test case.

Sample Input	Sample Output
3	ENEMIES
1 2	FRIENDS
a b	ENEMIES
a c	IMPOSSIBLE_TO_TELL
b d	SAME_PERSON
a d	
c d	ENEMIES
b c	ENEMIES
e a	ENEMIES
c c	FRIENDS
* *	ENEMIES
3 2	
a b	FRIENDS
e d	ENEMIES
c d	ENEMIES
a e	FRIENDS
b c	FRIENDS

b e a c b d c e a d * * 5 3 a c e a e g f b d h b i e f h i e c c b d c c g c i e i h e f g g i d c g d b i * *	FRIENDS ENEMIES ENEMIES FRIENDS ENEMIES ENEMIES ENEMIES

Problem C: Pixel Virus

Run Time Limit: 3 sec

In a computer room, all the computers are placed and addressed as a two-dimensional array in column major order. They all have the Internet connectivity. An attacker created an attack message called pixel virus. One day, one of the computers firstly receives a message called pixel virus. This type of virus shuts down the light of the pixels on the computer monitor. And then, it affects to its four connected neighbor computers one by one in accessing order; firstly affect is (x-1,y), secondly affect is (x,y-1), thirdly affect is (x,y+1) and finally (x+1,y) . 4-connected neighbors are neighbors to every computer that touches one of their edges. These computers are connected horizontally and vertically. In terms of their address, every computer that has the address (x-1,y), (x,y-1), (x,y+1) or (x+1,y) is connected to the computer at (x,y). Sample case 1 is as shown in figure.

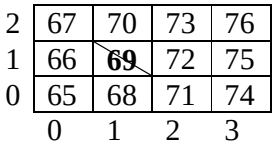


Figure :1

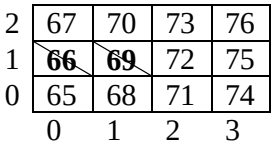


Figure : 2

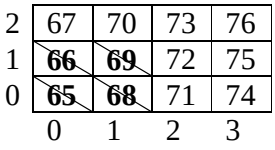


Figure : 3

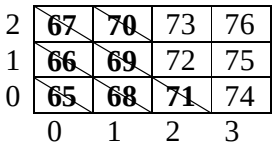


Figure :4

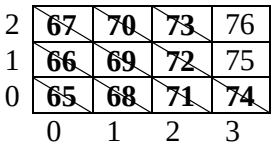


Figure : 5

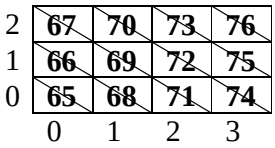


Figure : 6

Input

- First line contains **T**, the number of test cases.
- The second line shows the three separated integers such as maximum number of computer in each row, **xmax** and maximum number of computer in each column **ymax** and then the starting address number of the first computer, **S**.
- This is followed by the two separated integers in **x** and **y** coordinates such as the computer address that receives the pixel virus firstly.

Output

- Print computer addresses, which are affected by virus according to the shut down order in the same time with separated lines.

Constraint

- $1 \leq T \leq 10^5$
- $0 \leq xmax \leq 10^5$
- $0 \leq ymax \leq 10^5$
- $xmax \leq S \leq ymax$
- $xmax \leq x \leq ymax$
- $xmax \leq y \leq ymax$

Sample Input	Sample Output
2 3 2 65 1 1 3 4 5 1 2	Case 1: 69 66 65 68 67 70 71 72 73 74 75 76 Case 2: 12 7 6 11 5 8 10 13 9 14 15 16 17 18 19 20 21 22 23 24

Problem D: Primal Encryption

Run Time Limit: 1 sec

You've just learned about prime numbers and encryption and want to create your own encryption system based on primality. The encryption system is as follows: The input is a string of text. You convert the text to binary by substituting each character in the string with its ASCII code (8-bits). Then you convert the binary into a sequence of integers as follows: The i^{th} occurrence of `0` is encrypted as the i^{th} composite number and the i^{th} occurrence of `1` is encrypted as the i^{th} prime number. So, for example, the binary 10010111 would be converted into the sequence 2, 4, 6, 3, 8, 5, 7, 11. The problem would just be to take as input a string of text and convert that string into the cipher text of a sequence of integers in this manner.

Input

The first line of the input contains a single integer T, (1<=T<=100) that indicates the number of test cases. Each test case consists of a string of length M, (1<=M<=20). Each string can include only letters [a-zA-Z], numbers[0-9], and spaces.

Output

For each test case, output the cipher text of a sequence of integers. See the samples for the exact output format.

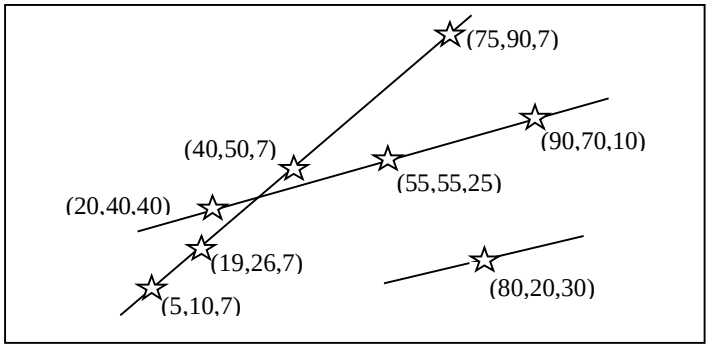
Sample Input	Sample Output
3 A B2 ab3	4,2,6,8,9,10,12,3 4,2,6,8,9,10,3,12,14,15,5,7,16,18,11,20 4,2,3,6,8,9,10,5,12,7,11,14,15,16,13,18,20,21,17,19,22,24,23,29

Problem E: Stars

Run Time Limit: 3 sec

Student X is very interested in game developing. He is a beginner and wants to develop the small shooting game where the player tries to hit the stars with bullets. In this game, a number of small stars are hanging with ropes from the ceiling at various points. These ropes are the same length or not. The player tries to shoot the stars with bullet and the bullet will pass right through and possibly hit one or more stars along its direction. The player can shoot the bullet from any position to hit the stars. The goal is being to hit all stars with as few shots as possible. The player with fewest shot will pass to next level.

In the following example, 8 stars can be hit with only 3 shots.



Three integer numbers are required to represents the star. The first two numbers represents positions at the ceiling (x and y Cartesian coordinates position) and the third number is the length of rope. Your job is to write the program to find the fewest number of shots to hit all stars hanging from the ceiling.

Input

Each test case consists of an integer n ≤ 100 that represents the number of target positions (stars positions). If the value of n is 0 the input will terminate. There will follow n integer triples (x y l), that indicate a star hanging at position (x, y) from the ceiling with rope length (l). The positions of stars should be distinct. All integers are greater than 0 and not greater than 100.

Output

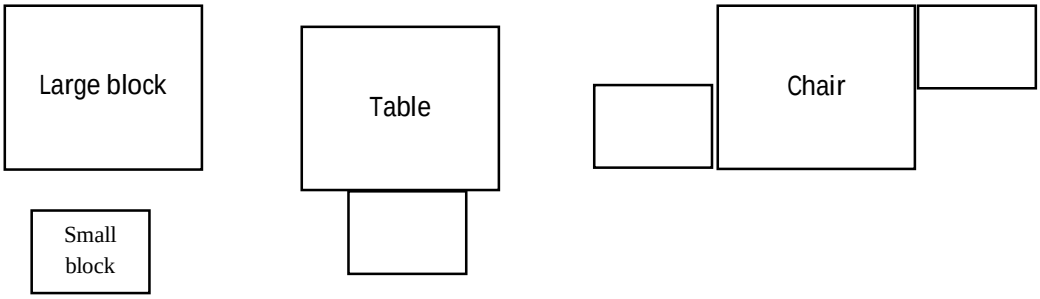
For each test case, only one integer value will output for minimum number of shots.

Sample Input	Sample Output
8 5 10 7 19 26 7 40 50 7 75 90 7 20 40 40 55 55 25 90 70 10 80 20 30 9 20 20 80 25 26 20 35 38 62 70 80 20 75 45 10 65 35 10 60 30 10 40 10 70 90 70 20 0	3 4

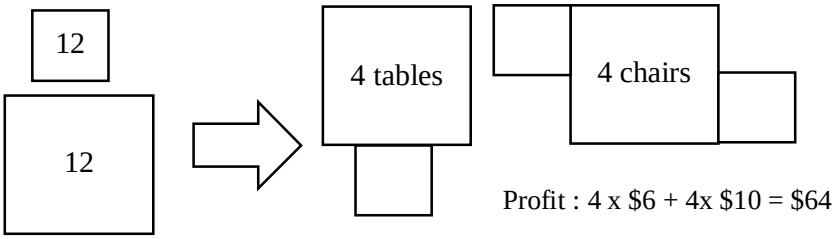
Problem F: Optimization

Run Time Limit: 1 sec

A furniture manufacturer produces two sizes of boxes (large, small) that are used to make either a table or a chair. We can make a table using 1 large block and 1 small block. A chair can be made using 1 large block and 2 small blocks. A table makes $6 profit and a chair makes $10 profit. Given **S** small blocks and **L** large blocks, we would like to know how many tables and chairs should we make to obtain the most profit.



Example 1 : Given 12 large blocks and 12 small blocks, if we make 4 tables and 4 chairs, 8 large blocks and 12 small blocks are used, we can make profit $64.



If 8 tables and 2 chairs are made using 10 large blocks and 12 small blocks, we can make profit $68. If 12 tables are made using all large and small blocks, we can make profit $72. To obtain the greatest profit, we should make 12 tables.

Example 2 : Given 12 large blocks and 20 small blocks, we can calculate and compare profit as shown below. To obtain the greatest profit, we should make 4 tables and 8 chairs.

# of tables		2	4	5	6	7	8	10	11
# of chairs		9	8	7	6	5	4	2	1
profit		102	**104**	100	96	92	88	80	76
# of blocks used	#large	11	12	12	12	12	12	12	12
	#small	20	20	19	18	17	16	14	13

Example 3 : Given 12 large blocks and 25 small blocks, we can calculate and compare profit as shown below. To obtain the greatest profit, we should make 12 chairs.

# of tables		0	1	4	5	6	7	8	10
# of chairs		12	11	8	7	6	5	4	2
profit		**120**	116	104	100	96	92	88	80
# of blocks used	#large	12	12	12	12	12	12	12	12
	#small	24	23	20	19	18	17	16	14

Given **S** small blocks and **L** large blocks, your task is to write a program that calculate the number of tables and chairs we should make to obtain the most profit?

Input

The first line contains integer T(1<=T<=10) which is the number of test cases. The input for each test case is given in a single line. This line contains two integer **L** and **S** separated by a space. Integer **L**(1<=L<=100) represents the number of large block and **S**(1<=S<=100) represents the number of small block.

Output

For each line of input, produce one line of output. It contains two integer values separated by a space. First integer value denotes the number of tables and second value denote the number of chairs. See the samples for the exact format of output.

Sample Input	Sample Output
10 1 2 2 1 50 45 60 100 100 5 70 70 10 15 12 20 1 1 2 2	0 1 1 0 45 0 20 40 5 0 70 0 5 5 4 8 1 0 2 0

Problem G: Stone

Run Time Limit: 1 sec

There are some stones lying on the road. You need to arrange them in non-decreasing order of their weight. You can move any stone to any position. You want to minimize the total weight (sum of the weight of the stones) you need to move.

Input

The first line of input contains a positive integer **T (T < 30)**, which denotes the number of test cases. The first line of each test case contains the number of stones, **n (1 <= n <= 10^5)**. The second line contains **n** positive integers $\mathbf{a_1, a_2, ..., a_n}$ **(a_i <= 10,000)** separated by a single space.

Output

For each of the cases output **"Case <x>: <y>"** in a separate line, where **x** is the case number and **y** is the minimum sum of weight you need to move. See the samples for the exact format of output.

Sample Input	Sample Output
3 4 7 1 2 3 4 7 1 2 5 11 6 4 5 3 8 2 7 2 11 2 2	Case 1: 6 Case 2: 7 Case 3: 24

Problem H: Sum Square

Run Time Limit: 3 sec

The *Sum Squared Digits* function, ***SSD(b, n)*** of a positive integer ***n***, in base ***b*** is defined by representing ***n*** in base ***b*** as in:

$$n = a_0 + a_1*b + a_2*b^2 + \ldots$$

then:

$$SSD(b, n) = a_0^2 + a_1^2 + a_2^2 + \ldots$$

is the sum of squares of the digits of the representation.

A *Sum Squared Digits Sequence*, a_n, in base ***b***, is defined by:

a_0 is arbitrary
$a_{n+1} = SSD(b, a_n)$

For example, in base 10:

4, 16, 37, 58, 89, 145, 42, 20, 4, …

Since $SSD(b, n) \le (1 + \log_b(n)) * (b-1)^2$, large starting values for a sequence decrease rapidly to 2 or 3 digit numbers and then stay in that range. This means that eventually the sequence will repeat.

Write a program that takes as input a base ***b***, the initial value a_0 and a number of terms to compute ***n*** and determines whether the sequence starting with a_0 using base ***b*** repeats in the first ***n*** terms. The sequence repeats if there are integers ***k*** and ***m*** with $k < m < n$ for which,

$$a_k = a_m$$

The *length* of the repeating sequence is $(m - k)$ if there is no integer ***j***, $k < j < m$ with $a_j = a_m$. I.e. the sequence from ***k*** to ***m*** is the shortest repeating sequence.

Input

The first line of input contains a single decimal integer ***P,*** ($1 \le P \le 10000$), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input. It contains the data set number, ***K,*** followed by the maximum number, ***n*** ($0 < n \le 1000$), of terms to compute followed by the base, ***b*** ($3 \le b \le 16$), followed by the initial value a_0 (a_0 will fit in a 32 bit *unsigned* integer).

Output

For each data set there are multiple lines of output.
If a repeating sequence is found, the first line of output contains the data set number, ***K***, followed by the index ***m*** where the sequence first repeated followed by the ***length*** of the shortest repeating

sub-sequence. The following lines of output contain the (***length*** + 1) terms of the sequence from term ***(k)*** to term ***(m),*** 20 terms to a line (except possibly for the last line). Term ***(k)*** is where $\mathbf{a_m}$ first occurred. Note that the first and last term in the displayed sub-sequence should be the same.

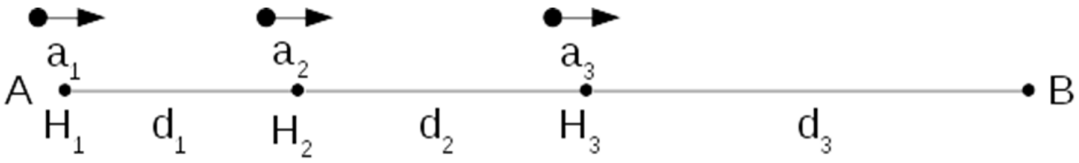
If a repeating sequence is not found in the first ***n*** terms, the first line of output contains the data set number, ***K***, followed by the number of terms ***n*** followed by the digit ***0***. The following line contains only the value $\mathbf{a_n}$ of the sequence at ***n*** (the $(n+1)^{th}$ term).

Sample Input	**Sample Output**
3 1 100 10 4 2 20 3 1234 3 100 16 289	1 9 8 4 16 37 58 89 145 42 20 4 2 5 2 2 4 2 3 10 1 1 1

Problem I: Hyperloop

Run Time Limit: 3 sec

Intergalactic Committee for Planetary Communication(ICPC) is in charge of space travel. They are implementing a new method of inter planetary travel using Hyperloop. A hyperloop is a device which will catch a vehicle with initial velocity **v** and increase the velocity of the vehicle for some time **t** (can be 0) with acceleration **a** and leave the vehicle in now new velocity **v + a*t**.



Suppose they want to make a way to go from point **A** to point **B**. They will put **N** hyperloops in the straight line **A** to **B** starting from **A**, numbered 1 to **N** with $\mathbf{a_1}$, $\mathbf{a_2}$,.., $\mathbf{a_N}$ accelerations. They are $\mathbf{d_1}$, $\mathbf{d_2}$, .., $\mathbf{d_{N-1}}$ distance away with each other and after the $\mathbf{N^{th}}$ Hyperloop **B** is $\mathbf{d_N}$ distance away.

Now ICPC wants to know what is the minimum time that a stopped vehicle with initial velocity 0 at point **A** needs to go to point **B**. They want you to find optimal time the vehicle have to spent at each Hyperloop so total time(time spent at Hyperloops and time traveling) to go to point **A** to point **B** is minimized.

Input:

First line of the input is **T**($T \leq 90$), then **T** test cases follows. First line of each case will be **N** ($1 \leq \mathbf{N} \leq 1000$). Next two line will both have **N** numbers with first line is $\mathbf{a_1}$, $\mathbf{a_2}$,,, $\mathbf{a_N}$ ($1 \leq \mathbf{a_i} \leq 10000$) and second line is $\mathbf{d_1}$, $\mathbf{d_2}$,,, $\mathbf{d_N}$ ($1 \leq \mathbf{d_i} \leq 10000$).

Output:

For each test case print a line in "Case I: T" format where **I** is case number and **T** is total time needed to go from **A** to **B**. Print **T** with exactly 4 place after decimal.

Sample Input	Sample Output
4 1 2 2 3 2 4 16 8 16 64 2 10 1 100 100 1 10 200	Case 1: 2.0000 Case 2: 10.2925 Case 3: 8.9443 Case 4: 8.9443

Problem J: Anti Hash II

Run Time Limit: 5 sec

Given a base **b** and a modulus **m**, the polynomial hash of a string consisting of only lowercase letters is defined as below.

Let $\mathbf{S = S_0S_1\ldots S_{n-1}}$ be a string of length **n** consisting of the letters **a-z**

$\mathbf{Hash(S) = \sum b^{n-i-1} * D(S_i) \% m}$

D(s) = Lexicographical position of character **s** among the letters **a-z**, indexed from 0.
D(a) = 0, D(b) = 1 ... and **D(z) = 25**.

In other words, first the letters of the string are replaced by numbers (equivalent to their position). This is then considered to be a number in base **B**, and the value of this number in base **10** modulo **m** is called the polynomial hash of the string.

Calculating the hash of a string using the above method seems easy enough. What about the opposite? You are given a base **b,** a modulus **m,** a positive integer **n,** and a hash value **h**. Calculate how many strings are there such that their hash is equal to **h**, consisting of only lowercase letters and their length not exceeding **n.** Since the answer can be rather huge, output it modulo **$10^9 + 7$ (1000000007).**

Input

The first line contains an integer **t (1≤ t≤150),** denoting the number of test cases. Each test case starts with four integers **b (26≤ b ≤ 15000), m, n(1≤ m, n≤15000) and q (1≤ q≤150).** The numbers **b, m**, and **n** denotes the base, modulus and maximum length of any string as stated above. The number **q** indicates the number of queries. Each of the next **q** lines contain a single integer, denoting **h (0≤ h<m)**, the required hash value. **For 95% of the test cases, b, m, n ≤ 150**

Output

For each case, first output a line of the format "Case X:", where X is the case number, starting from 1. And then, for each query, output the number of different strings with the given hash value modulo **$10^9 + 7$ (1000000007)** in a single line. After every test case, print a blank line.

Sample Input	Sample Output
3 26 97 2 3 0 1 96 147 147 147 3 0 10 100 100 110 120 1 35	Case 1: 8 8 6 Case 2: 944164777 944164777 0 Case 3: 110169522

Problem K: Sigma Function (II)

Run Time Limit: 3 sec

Sigma function is an interesting function in Number Theory. It is denoted by the Greek letter Sigma (σ). This function actually denotes the sum of all divisors of a number.
For example **σ(24) = 1+2+3+4+6+8+12+24=60.**

Sigma of small numbers is easy to find but for large numbers it is very difficult to find in a straight forward way. But mathematicians have discovered a formula to find sigma. If the prime power decomposition of an integer $= p_1^{e_1} \times p_2^{e_2} \times p_3^{e_3} \times \ldots \times p_{k-1}^{e_{k-1}} \times p_k^{e_k}$, then

$$\sigma(n) = \frac{p_1^{e_1+1}-1}{p_1-1} * \frac{p_2^{e_2+1}-1}{p_2-1} * \frac{p_3^{e_3+1}-1}{p_3-1} * \ldots * \frac{p_{k-1}^{e_{k-1}+1}-1}{p_{k-1}-1} * \frac{p_k^{e_k+1}-1}{p_k-1}$$

For most integer values of **n** the value of **σ(n)** is divisible by **3** but for others it is not divisible. Given two integer values $\mathbf{low_i}$ and $\mathbf{high_i}$, you will have to find for how many integer values of **n** from $\mathbf{low_i}$ to $\mathbf{high_i}$ (inclusive), **σ(n)** is divisible by **3**.

Input
The input file contains at most **20** lines of inputs.
Each line contains two integers $\mathbf{low_i}$, $\mathbf{high_i}$ $(\mathbf{0 < low_i \le high_i < 100000001})$.
Input is terminated by a line containing two zeroes. This line should not be processed.

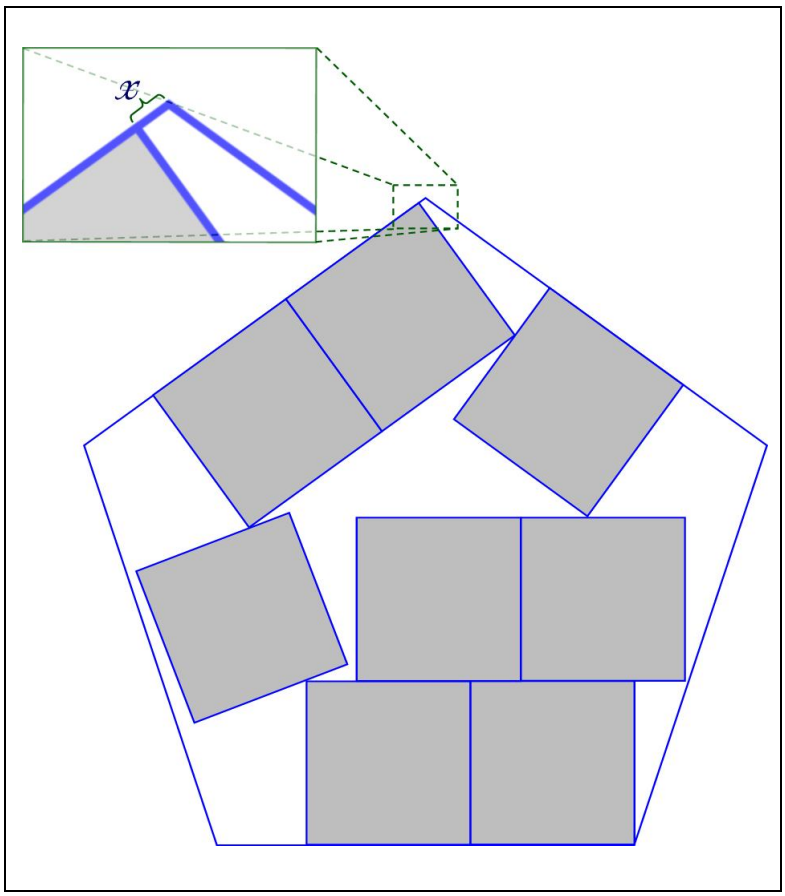
Output
For each line of input produce one line of output. This line denotes for how many integer values of **n** from **lowi** to $\mathbf{high_i}$(inclusive), **σ(n)** is divisible by **3**.

Sample Input	Sample Output
1 10 10 20 0 0	5 7

Problem L: Squares in a Regular Pentagon

Run Time Limit: 3 sec

David W. Cantrell along with Erich Friedman are two leading researchers of packing problems in geometry. In July, 2012 David proposed an optimal way of packing eight unit squares (1*1 squares) in a regular pentagon. This can be found here: http://www2.stetson.edu/~efriedma/squinpen/ . Here optimal means finding aside length s of the smallest regular pentagon which can hold eight unit squares so that none of the squares overlap. Approximate value of s found by David is `2.565+`. However, for a change in this problem you are not being asked to find the optimal value of s.



The figure on the left shows with reasonable accuracy how the eight squares were placed by David in his optimal configuration. As you can notice that the highest point of the topmost square as not overlapping with the topmost point of the pentagon. Let the distance between these two point be called x. Given the side length of the squares (All are equal) you will have to find the value of x with reasonable accuracy.

Input

Input can be at most `50001` lines of inputs. Each line contains a floating point number L `(0.0<L<100000.0)` which denotes the length of the sides of the squares. Input is terminated by a line containing a negative value. This value should not be processed.

Output

For each line of input produce one line of output. This line contains the value of x in the optimal packing situation. The printed value should have six digits after the decimal point. Inputs will be such that this value of x will not change for small precision errors like 10^{-8}. But your output should match the judge output exactly.

Sample Input	Sample Output
0.001 0.002 -1.2	0.000044 0.000088

Screenshot from the link above is shown below: