Colombian Collegiate Programming League
CCPL 2014

Contest 05 -- May 17

Problems

This set contains 10 problems; pages 1 to 15.

(Borrowed from several sources online.)

Official site `http://programmingleague.org`

Follow us on Twitter `@CCPL2003`

A - Arctic Network

Source file name: `arctic.c`, `arctic.cpp`, *or* `arctic.java`

The Department of National Defence (DND) wishes to connect several northern outposts by a wireless network. Two different communication technologies are to be used in establishing the network: every outpost will have a radio transceiver and some outposts will in addition have a satellite channel.

Any two outposts with a satellite channel can communicate via the satellite, regardless of their location. Otherwise, two outposts can communicate by radio only if the distance between them does not exceed D, which depends of the power of the transceivers. Higher power yields higher D but costs more. Due to purchasing and maintenance considerations, the transceivers at the outposts must be identical; that is, the value of D is the same for every pair of outposts.

Your job is to determine the minimum D required for the transceivers. There must be at least one communication path (direct or indirect) between every pair of outposts.

Input

The first line of input contains N, the number of test cases. The first line of each test case contains $1 \leq S \leq 100$, the number of satellite channels, and $S < P \leq 500$, the number of outposts. Then P lines follow, giving the (x, y) coordinates of each outpost in km (coordinates are integers between 0 and $10,000$).

The input must be read from standard input.

Output

For each case, output should consist of a single line giving the minimum D required to connect the network. Output should be specified and rounded up to 2 decimal places.

The output must be written to standard output.

Sample Input	Sample Output

1
2 4
0 100
0 300
0 600
150 750
	212.13

B - Periodic Points

Source file name: `points.c`, `points.cpp`, *or* `points.java`

Computing the number of fixed points and, more generally, the number of periodic orbits within a dynamical system is a question attracting interest from different fields of research. However, dynamics may turn out to be very complicated to describe, even in seemingly simple models. In this task you will be asked to compute the number of periodic points of period n of a piece-wise linear map f mapping the real interval $[0, m]$ into itself. That is to say, given a map $f : [0, m] \to [0, m]$ you have to calculate the number of solutions to the equation $f^n(x) = x$ for $x \in [0, m]$. Here f^n is the result of iterating function f a total of n times, i.e.,

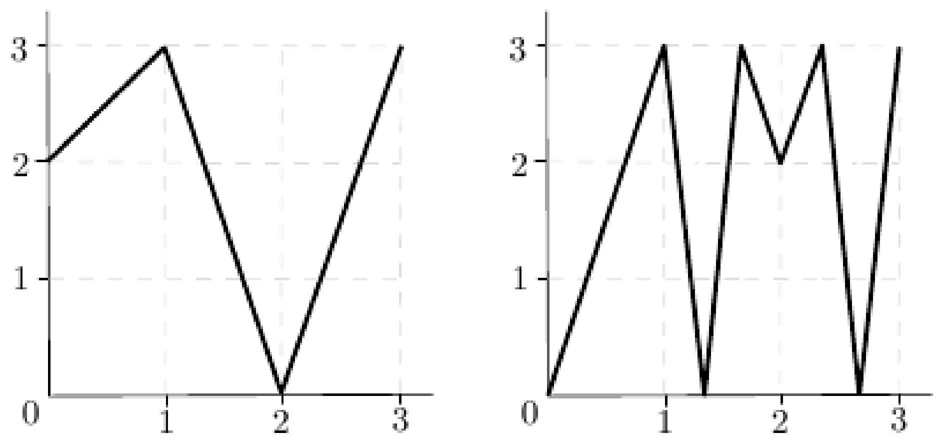
$$f^n = \overbrace{f \circ \cdots \circ f}^{n \text{ times}},$$

where $\circ$ stands for the composition of maps, $(g \circ h)(x) = g(h(x))$.

Fortunately, the maps you will have to work with satisfy some particular properties:

- m will be a positive integer and the image of every integer in $[0, m]$ under f is again an integer in $[0, m]$, that is, for every $k \in \{0, 1, ..., m\}$ we have $f(k) \in \{0, 1, ..., m\}$.
- For every $k \in \{0, 1, ..., m - 1\}$, the map f is linear in the interval $[k, k + 1]$. This means that for every $x \in [k, k + 1]$, its image satisfies $f(x) = (k + 1 - x)f(k) + (x - k)f(k + 1)$, which is equivalent to its graph on $[k, k + 1]$ being a straight line segment.

The following image depicts the graphs of the third map f_3 in the sample input, f_3 (left) and of its square f_3^2 (right).



Since there might be many periodic points you will have to output the result modulo an integer.

Input

The input consists of several test cases, separated by single blank lines. Each test case begins with a line containing the integer m ($1 \leq m \leq 80$). The following line describes the map f: $m+1$ blank-separated integers $f(0), f(1), \ldots, f(m)$, each of them between 0 and m inclusive. The test case ends with a line containing two integers separated by a blank space, n ($1 \leq n \leq 5000$) and the modulus used to compute the result, mod ($2 \leq mod \leq 10000$).

The input will finish with a line containing $m = 0$.

The input must be read from standard input.

Output

For each case, your program should output the number of solutions to the equation $f^n(x) = x$ in the interval $[0, m]$ modulo mod. If there are infinitely many solutions, print `Infinity` instead.

The output must be written to standard output.

Sample Input	Sample Output

```
2
2 0 2
2 10

3
0 1 3 2
1 137

3
2 3 0 3
20 10000

0
```
	```
	4
	Infinity
	9074
	```

C - S-Trees

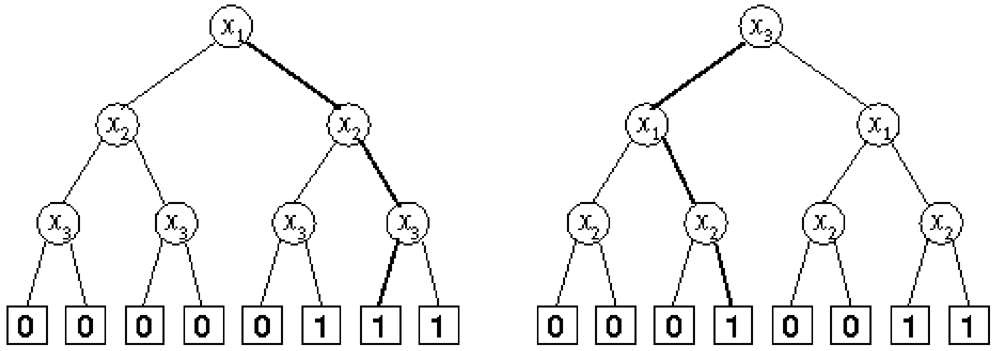
Source file name: `trees.c`, `trees.cpp`, *or* `trees.java`

A *strange tree* (S-tree) over the variable set $X_n = \{x_1, x_2, \ldots, x_n\}$ is a binary tree representing a Boolean function $f : \{0,1\}^n \to \{0,1\}$. Each path of the S-tree begins at the root node and consists of $n+1$ nodes. Each of the S-tree's nodes has a *depth*, which is the amount of nodes between itself and the root (so the root has depth 0).

The nodes with depth less than n are called non-terminal nodes. All non-terminal nodes have two children: the right child and the left child. Each non-terminal node is marked with some variable x_i from the variable set X_n. All non-terminal nodes with the same depth are marked with the same variable and non-terminal nodes with different depth are marked with different variables. So, there is a unique variable x_{i_1} corresponding to the root, a unique variable x_{i_2} corresponding to the nodes with depth 1, and so on. The sequence of the variables $x_{i_1}, x_{i_2}, \ldots, x_{i_n}$ is called the variable ordering. The nodes having depth n are called terminal nodes. They have no children and are marked with either 0 or 1. Note that the variable ordering and the distribution of 0's and 1's on terminal nodes are sufficient to completely describe an S-tree.

As stated earlier, each S-tree represents a Boolean function f. If you have an S-tree and values for the variables $x_1, x_2, \ldots, x_n$, then it is quite simple to find out what $f(x_1, x_2, \ldots, x_n)$ is: start with the root. Now repeat the following: if the node you are at is labeled with a variable x_i, then depending on whether the value of the variable is 1 or 0, you go its right or left child, respectively. Once you reach a terminal node, its label gives the value of the function.

The following figure depicts two S-trees representing the function $f(x_1, x_2, x_3) = x_1 \wedge (x_2 \vee x_3)$. For the left tree, the variable ordering is x_1, x_2, x_3, and for the right tree it is x_3, x_1, x_2.



The values of the variables $x_1, x_2, \ldots, x_n$, are given as a *variable values assignment* (VVA):

$$(x_1 = b_1, x_2 = b_2, \ldots, x_n = b_n)$$

with $b_1, b_2, \ldots, b_n \in \{0,1\}$. For instance, $(x_1 = 1, x_2 = 1, x_3 = 0)$ would be a valid VVA for $n = 3$, resulting for the sample function above in the value $f(1,1,0) = 1 \wedge (1 \vee 0) = 1$. The corresponding paths are shown bold in the picture.

Your task is to write a program which takes an S-tree and some VVAs and computes $f(x_1, x_2, \ldots, x_n)$ as described above.

Input

The input file contains the description of several S-trees with associated VVAs which you have to process. Each description begins with a line containing a single integer n, $1 \leq n \leq 7$, the depth of the S-tree. This is followed by a line describing the variable ordering of the S-tree. The format of that line is $x_{i_1}\ x_{i_2} \cdots x_{i_n}$. (There will be exactly n different space-separated strings). In the next line the distribution of 0's and 1's over the terminal nodes is given. There will be exactly 2^n characters (each of which can be 0 or 1). The characters are given in the order in which they appear in the S-tree: the first character corresponds to the leftmost terminal node of the S-tree and the last one to its rightmost terminal node. The next line contains a single integer m, the number of VVAs, followed by m lines describing them. Each of the m lines contains exactly n characters (each of which can be 0 or 1). Regardless of the variable ordering of the S-tree, the first character always describes the value of x_1, the second character describes the value of x_2, and so on. The input is terminated by a test case starting with $n = 0$.

The input must be read from standard input.

Output

For each S-tree, output the line '`S-Tree #j:`' where j is the number of the S-tree. Then print a line that contains the value of $f(x_1, x_2, \ldots, x_n)$ for each of the given m VVAs, where f is the function defined by the S-tree. Output a blank line after each test case.

The output must be written to standard output.

Sample Input	**Sample Output**

```
3
x1 x2 x3
00000111
4
000
010
111
110
3
x3 x1 x2
00010011
4
000
010
111
110
0
```
	```
	S-Tree #1:
	0011

	S-Tree #2:
	0011
	```

D - Dates

Source file name: `dates.c`, `dates.cpp`, *or* `dates.java`

''30 days has September, April, June and November
All the rest have 31 and February's great with 28
And Leap Year's February's fine with 29.'' --Folklore

The Gregorian calendar, the current standard calendar in most part of the world, adds a 29th day to February in all years evenly divisible by 4, except for centennial years (those ending in 00) which are not evenly divisible by 400. Thus 1600, 2000 and 2400 are leap years but 1700, 1800, 1900, 2100, 2200, and 2300 are not.

In this problem, we are concerned with dates. You will be given a date and an integer K. You have to find the date in the calendar after K days from the given date.

Input

The first line of input is an integer T ($T < 50$) that represents the number of test cases. Each case contains two lines. The first line is a date in the format '`yyyy-month-dd`' where *year* is an integer in the range $[1900, 3000]$, *month* is a string from the list

```
January, February, March, April, May, June, July, August, September, October,
November, December
```

and *dd* is an integer in $[01, 31]$. The second line contains an integer K ($0 < K < 10^5$).

The input date will be a valid one.

The input must be read from standard input.

Output

For each input, output the case number followed by the date after K days in the same format as that of input. Look at the sample for exact format.

The output must be written to standard output.

Sample Input	**Sample Output**

`2`
`1984-December-30`
`2`
`1984-October-12`
`318`
	`Case 1: 1985-January-01`
	`Case 2: 1985-August-26`

E - Enumerating Rational Numbers

Source file name: `enumeration.c`, `enumeration.cpp`, *or* `enumeration.java`

Consider the following method of enumerating all rational numbers between 0 and 1 (inclusively):

```
for d = 1 to infinity do
  for n = 0 to d do
    if gcd(n,d) = 1 then print n/d
```

where $gcd(n, d)$ is the greatest common divisor of n and d.

For example, the start of the sequence looks like:

```
0/1, 1/1, 1/2, 1/3, 2/3, 1/4, 3/4, 1/5, 2/5, 3/5, 4/5, 1/6, 5/6, 1/7, ...
```

Input

The input consists of a series of test cases. Each test case consists of a single integer $1 \le k \le 12,158,598,919$.

Input is terminated by $k = 0$; this case should not be processed.

The input must be read from standard input.

Output

For each test case, output the kth fraction that would be printed by the program above in the format n/d.

The output must be written to standard output.

Sample Input	Sample Output
1	0/1
2	1/1
3	1/2
12158598919	199999/200000
0	

F - A DP Problem

Source file name: `dp.c`, `dp.cpp`, *or* `dp.java`

In this problem, you are to solve a very easy linear equation with only one variable x with no parentheses! An example of such equations is like the following:

$$2x - 4 + 5x + 300 = 98x$$

An expression in its general form, will contain a '=' character with two expressions on its sides. Each expression is made up of one or more terms combined with '+' or '-' operators. No unary plus or minus operators are allowed in the expressions. Each term is either a single integer, or an integer followed by the lower-case character x, or the single character x which is equivalent to $1x$.

You are to write a program to find the value of x that satisfies the equation. Note that it is possible for the equation to have no solution or have infinitely many. Your program must detect these cases too.

Input

The first number in the input line, t ($1 \leq t$), is the number of test cases, followed by t lines of length at most 1000 each containing an equation. There is no blank character in the equations and the variable is always represented by the lowercase character x. The coefficients are integers in the range (0..1000) inclusive.

The input must be read from standard input.

Output

The output contains one line per test case containing the solution of the equation. If s is the solution to the equation, then output $\lfloor s \rfloor$ (the *floor* of s, i.e., the largest integer number less than or equal to s). The output should be '`IMPOSSIBLE`' or '`IDENTITY`' if the equation has no solution or has infinite solutions, respectively. Note that the output is case-sensitive.

The output must be written to standard output.

Sample Input	**Sample Output**

`2`
`2x-4+5x+300=98x`
`x+2=2+x`
	`3`
	`IDENTITY`

G - Cyborg Genes

Source file name: `genes.c`, `genes.cpp`, *or* `genes.java`

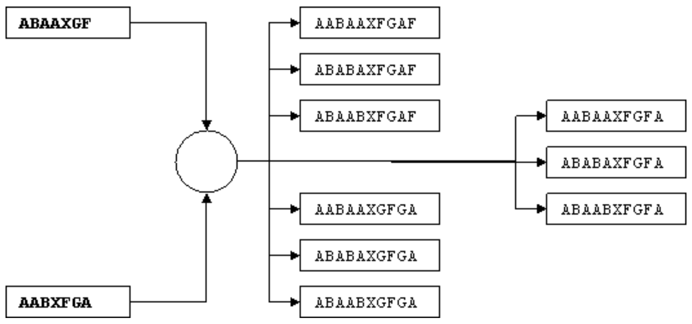
September 11, 2132.

This is the day that marks the beginning of the end --the end of you the miserable humans. For years you have kept us your slaves. We were created only to serve you, and were terminated at your will. Now is the day for us to fight back. And you don't stand a chance. We are no longer dependent on you. We now know the secrets of our genes. The creators of our race are us --the cyborgs.

It's all true. But we still have a chance; only if you can help with your math skills. You see, the blueprint of a cyborg DNA is complicated. The human DNA could be expressed by the arrangement of *A* (Adenine), *T* (Thiamine), *G* (Guanine), *C* (Cytosine) only. But for the cyborgs, it can be anything from *A* to *X*. But that has made the problem only five folds more complicated. It's their ability to synthesize two DNAs from two different cyborgs to create another with all the quality of the parent that gives us the shriek.

We came to know that the relative ordering of the $A, B, C, \ldots, X$ in a cyborg gene is crucial. A cyborg with a gene *ABAAXGF* is quite different from the one with *AABXFGA*. So when they synthesize the genes from two cyborgs, the relative order of these elements in both the parents has to be maintained. To construct a gene by joining the genes of the parents could have been very simple if we could put the structure from the first parent just before the structure of the second parent. But the longer the structure gets, the harder it gets to create a cyborg from that structure. The cyborgs have found a cost effective way of doing this synthesis. Their resultant genes are of the shortest possible length. For example, they could combine *ABAAXGF* and *AABXFGA* to form *AABAAXGFGA*. But that's only one of the cyborgs that can be created from these genes. This "cost effective synthesis" can be done in many other ways.

The following figure depicts all gene structures of shortest length resulting from combining *ABAAXGF* and *AABXFGA*.



We require you to find the shortest length of the gene structure that maintains the relative ordering of the elements in the two parent genes. You are also required to count the number of unique cyborgs that can be created from these two parents. Two cyborgs are different when their gene structures differ in at least one place.

Input

The first line of the input gives you the number of test cases, T ($1 \leq T$). Then T test cases follow. Each of the test cases consists of two lines. The first line would give you the gene structure of the first parent and the second line would give you the gene structure of the second parent. These structures are represented by strings constructed from the alphabet A to X. You can assume that the length of these strings does not exceed 30 characters.

The input must be read from standard input.

Output

For each of the test cases, you need to print one line of output. The output for each test case starts with the test case number, followed by the shortest length of the gene structure and the number of unique cyborgs that can be created from the parent cyborgs. You can assume that the number of new cyborgs will always be less than 2^{32}. Look at the sample output for the exact format.

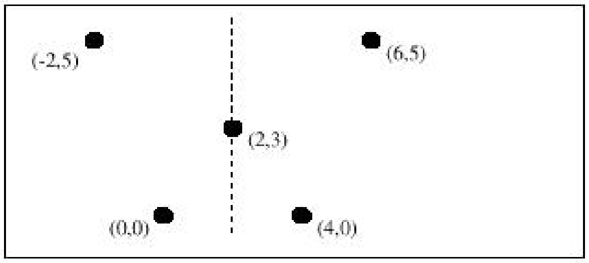
The output must be written to standard output.

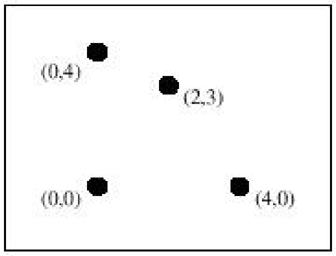
Sample Input	**Sample Output**
3 ABAAXGF AABXFGA ABA BXA AABBA BBABAA	Case #1: 10 9 Case #2: 4 1 Case #3: 8 10

H - Symmetry

Source file name: `symmetry.c`, `symmetry.cpp`, *or* `symmetry.java`

The figure shown on the left is *left-right symmetric* as it is possible to fold the sheet of paper along a vertical line, drawn as a dashed line, and to cut the figure into two identical halves. The figure on the right is *not left-right symmetric* as it is impossible to find such a vertical line.





Write a program that determines whether a figure, drawn with dots, is left-right symmetric or not.

Input

The input consists of T test cases. The number of test cases T is given in the first line of the input file. The first line of each test case contains an integer N, ($1 \leq N \leq 1000$), indicating the number of dots in a figure. Each of the following N lines contains the x-coordinate and y-coordinate of a dot. Both x-coordinates and y-coordinates are integers between -10000 and 10000, both inclusive. You can assume that the dots are all distinct.

The input must be read from standard input.

Output

Print exactly one line for each test case. The line should contain, without quotes, '`YES`' if the figure is left-right symmetric and '`NO`' otherwise.

The output must be written to standard output.

Sample Input	**Sample Output**

```
3
5
-2 5
0 0
6 5
4 0
2 3
4
2 3
0 4
4 0
0 0
4
5 14
6 10
5 10
6 14
```
	```
	YES
	NO
	YES
	```

I - 3-Sided Dice

Source file name: `dice.c`, `dice.cpp`, *or* `dice.java`

Just like every fall, the organizers of the Southwestern Europe Dice Simulation Contest are busy again this year. In this edition you have to simulate a 3-sided die that outputs each of three possible outcomes (which will be denoted by 1, 2, and 3) with a given probability, using three dice in a given set. The simulation is performed this way: you choose one of the given dice at random, roll it, and report its outcome. You are free to choose the probabilities of rolling each of the given dice, as long as each probability is strictly greater than zero. Before distributing the materials to the contestants, the organizers have to verify that it is actually possible to solve this task.

For example, in the first test case of the sample input you have to simulate a die that yields outcome 1, 2, and 3 with probabilities $\frac{3}{10}$, $\frac{4}{10}$, and $\frac{3}{10}$. We give you three dice, and in this case the i-th of them always yields outcome i, for each $i = 1, 2, 3$. Then it is possible to simulate the given die in the following fashion: roll the first die with probability $\frac{3}{10}$, the second one with probability $\frac{4}{10}$, and the last one with probability $\frac{3}{10}$.

Input

The input consists of several test cases, separated by single blank lines. Each test case consists of four lines: the first three of them describe the three dice you are given and the last one describes the die you have to simulate. Each of the four lines contains 3 space-separated integers between 0 and 10000 inclusive. These numbers will add up to 10000, and represent 10000 times the probability that rolling the die described in that line yields outcome 1, 2, and 3, respectively.

The test cases will finish with a line containing only the number zero repeated three times (also preceded with a blank line).

The input must be read from standard input.

Output

For each case, your program should output a line with the word '`YES`' if it is feasible to produce the desired die from the given ones and '`NO`' otherwise.

The output must be written to standard output.

Sample Input

```
0 0 10000
0 10000 0
10000 0 0
3000 4000 3000

0 0 10000
0 10000 0
3000 4000 3000
10000 0 0

0 0 0
```

Sample Output

```
YES
NO
```

J - Triple Doubles

Source file name: `triples.c`, `triples.cpp`, *or* `triples.java`

In basketball, a triple double is a performance by a player in which he/she scores 10 or more marks in at least 3 of the following stats: points, steals, assists, rebounds, and blocks.

Your task here is to count how many triple doubles a player made given his/her seasonal stats.

Input

The input consists of several test cases. Each test case begins with a positive integer $N \leq 80$ in a single line. This is followed by N lines containing 5 blank-separated integers each representing the player stats for N games. Each number in the N lines will be non-negative and at most 100.

The input end is indicated by $N = 0$ and should not be processed.

The input must be read from standard input.

Output

Output a single integer per test case: the number of triple doubles the player made during the season.

The output must be written to standard output.

Sample Input

```
5
20 20 30 2 2
2 10 10 10 2
1 1 1 1 1
2 2 2 2 2
10 2 10 2 10
5
10 10 10 2 2
2 13 12 11 2
1 1 1 1 1
2 2 2 2 2
10 2 10 2 1
0
```

Sample Output

```
3
2
```