7th Gulf Programming Contest

March 22,23 2017

Gulf University of Science and Technology

Kuwait

Duration: 10 am - 3 pm

Sponsored by

Problem Set

Problem #	***Problem Name***	***Balloon Color***
A	Squares or Circles	Red
B	The Peak Hour	Orange
C	Table Sorting	Yellow
D	The Easy League	Blue
E	Elvis the King Cobra	Silver
F	Sales Craze	Gold
G	Cairo Pentagonal Tiling	White
H	Armstrong numbers	Pink
I	Amicable Numbers	Purple
J	Number Pyramid	Navy
K	Exotic Election	Black
L	The Digits	Green

A. Squares or Circles

Program:	squares.(cpp|java)
Input:	squares.in
Balloon Color:	red

Description

You are playing a game with your friend where you have a box full of square and circle shaped plastic pieces. In each move, a player can either take two circles, two squares, or one circle and one square. A player cannot take only one piece in a move. The winner is the player who makes the last move.

Given the number of squares and circles in the box, the player who starts first (you or your friend), and assuming that both of you play with optimal strategies, your task is to find out whether you will win or lose.

Input Format

The input starts with a number T ($1 \le T \le 1{,}000$) that represents the number of test cases in the file. Each test case consists of a line that starts with two integers S ($1 \le S \le 10^6$) and C ($1 \le C \le 10^6$) representing the number of squares and number of circles, respectively, followed by a single character that represents who starts the game. The character is 'M' if you start, and it is 'F' if your friend starts the game.

Output Format

The output for each test case is in this form:

```
k. ans
```

where ***k*** represents the test case number (starting at 1), and ***ans*** is either "I Win!", if you win, or "I Lose!", if your friend wins the game.

Sample Input / Output

squares.i

```
3
3 3 M
4 2 F
5 3 M
```

OUTPUT

```
1. I Win!
2. I Lose!
3. I Lose!
```

B. The Peak Hour

Program:	`peak.(cpp|java)`
Input:	`peak.in`
Balloon Color:	`orange`

Description

Server sizing is an important factor when companies are purchasing hardware or allocating resources on a virtual hypervisor for specific services. In this problem, we will look at sizing for email gateway servers. Some of the factors that are considered are average email size, email throughput, and the number of mailboxes or users. In many cases, the major factor is email throughput, and it is usually measured by the peak hour (the maximum number of emails processed in an hour).

When administrators try to get an estimate of the peak email throughput, they usually generate reports from their mail systems and look at the maximum number of emails counted from the tip of an hour to the next (for instance, the peak hour could be from 1PM to 2PM). Estimating the peak hour with this method does not produce accurate measurements. For example, the peak hour could be from 1:48PM to 2:48PM. Administrators do not consider this as they are only looking at hourly data from the tip of an hour to the next.

In this problem, you are given the number of emails N, the time frame T which represents the frame of time in minutes, where we are looking for the peak mail throughput (e.g if T=60, then we are looking for the peak hour), and the times emails are processed (in minutes). Your task is to calculate the peak throughput over all the possible time frames of length T.

Input Format

The input starts with a number C ($1 \le C \le 100$) that represents the number of test cases in the file. Each test case starts with a line that contains two integers, N ($1 \le N \le 10^6$), and T ($1 \le T \le 10^9$), as described above. N lines follow with each containing an integer N_i ($1 \le N_i \le 10^9$) representing the time an email is processed in minutes. Multiple emails can be processed in the same minute.

Output Format

The output for each test case is in this form:

```
k. ans
```

where ***k*** represents the test case number (starting at 1), and ***ans*** is the peak throughput.

Sample Input / Output

peak.in

```
2
3 2
1
2
3
5 3
1
2
5
4
6
```

OUTPUT

```
1. 2
2. 3
```

C. Table Sorting

Program:	`sorting .(cpp|java)`
Input:	`sorting .in`
Balloon Color:	`yellow`

Description

Sorting a table is an interesting problem, which generates different views of the table based on desired criteria. For example, we may have a table of student records having student IDs, names, GPAs and so on. We may want to view this table sorted by the GPA to find the top ten students; alternatively, we may want to view the table sorted by the names to quickly search for a student by name and so on. Sometimes, when we sort using one attribute (e.g. GPA), we may find more than one students having the same GPA – in that case, we may want to sort students with same GPA based on student ID and so on.

Input

The input consists of n ($0< n <= 100$) tables. Each table starts with three integers c <space> r <space> k, where c (< 10) is the number of attributes (i.e., columns), r (< 100) is the number of rows, and k (<100) is the number of sorting instructions. Then r lines follow, where each line consists of c values; the values may be of two types: strings (lower-case alphanumeric {'a'-'z'|'0'-'9'}, max 10 characters always starting with a letter) or numbers (real double precision numbers). After the table data, k lines follow, each line containing the sorting column number u ($1<= u <= c$) in order of priority, ending with a 0. A negative column number indicates sorting must be done in descending order and positive indicates sorting must be done in ascending order. If a column number does not appear in the sorting instructions, then the order in the input should be preserved (stable sort).

Output

For each table, you should output the table number as "Table X" in one line. For each sorting instruction, you should output the instruction number as "Instruction Y" in one line, followed by the first m lines of the sorted (as per instruction) table, where $m = \min(r, 5)$. Columns should be in the same order as in the input and separated by a single space.

Sample Input/Output

sorting.in

```
1
3 4 2
hary 301 3.5
saly 201 3.5
mary 105 3.9
ali 203 4.0
-3 2 0
-3 0
```

OUTPUT

```
Table 1
Instruction 1
ali 203 4.0
mary 105 3.9
saly 201 3.5
hary 301 3.5
Instruction 2
ali 203 4.0
mary 105 3.9
hary 301 3.5
saly 201 3.5
```

D. The Easy League

Program:	`league.(cpp|java)`
Input:	`league.in`
Balloon Color:	`blue`

Description

In football leagues across many countries, each team plays two games against all other teams in the league, a home and an away game. In each game, the winner gains 3 points, and the loser does not gain any points. In case of a draw, each team gains 1 point. The team with the maximum points wins the league.

Given the number of teams in a league, your task is to calculate the maximum possible number of points a team can gain.

Input Format

The input starts with a number T ($1 \le T \le 1{,}000$) that represents the number of test cases in the file. Each test case consists of a line containing an integer N ($2 \le N \le 10^6$) that represents the number of teams in the league.

Output Format

The output for each test case is in this form:

```
k. ans
```

where ***k*** represents the test case number (starting at 1), and ***ans*** is the maximum number of points a team can gain.

`league.in`

```
2
3
4
```

OUTPUT

```
1. 12
2. 18
```

E. Elvis the King Cobra

Program:	`elvis.(cpp|java)`
Input:	`elvis.in`
Balloon Color:	`silver`

Description

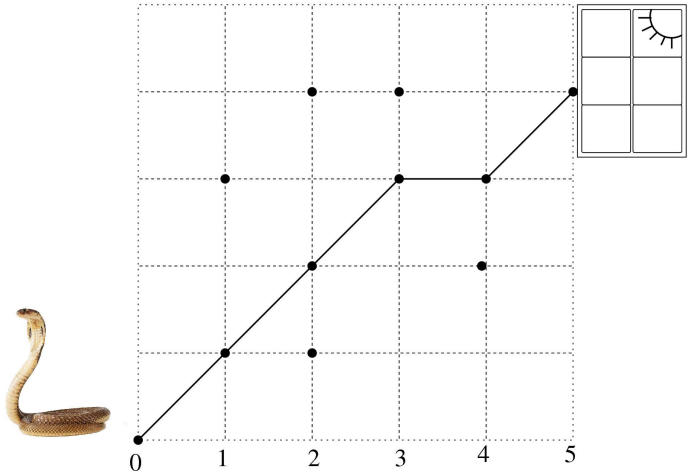
Viperkeeper is a famous Youtuber that keeps a collection of deadly venomous snakes. The most valued member of his collection is a four (4) meter long king cobra that he calls Elvis. Elvis is domesticated enough to be allowed supervised "adventure trips" through Viperkeeper's home.

Elvis is always returned to his cage, but his freedom has not left his mind! Today is a special day because it seems that Viperkeeper has left one of the house windows open. A number of anchor points are between Elvis and freedom. But which ones should he pick to hasten his escape?

Also there is the issue that Viperkeeper can be distracted for only time *T*. Elvis should not attempt an escape if his travel time to the window exceeds *T*, otherwise Viperkeeper may not let him out of his cage again. Elvis can move at a maximum velocity of *V* and so the total distance between the anchor points can be used to calculate the time to escape.

However, it should be noted that Elvis must not only make it to the window in time *T*, but he must also move all his 4 meter body out as well. There are also some restriction to Elvis's movements: (a) he cannot move vertically as he is too heavy, and (b) he cannot cross a span between two anchor points which have a distance greater than Y meters.

Your task is to calculate whether Elvis can make it to the window and escape or not.



In the example illustrated above, we have nine anchor points between Elvis and the window. If *Y*=2m, *V*=1m/sec and *T*=11sec, Elvis will be able to escape following the shown route (of length 6.657 meters), leaving also a spare of 0.343sec before Viperkeeper realizes he is gone.

Input

The input consists of several test cases. The first line contains the number of test cases. Each test case begins with a line containing four integers separated by white space: *N* the number of anchor points

(2<=*N*<=100), the velocity (*V*>0), the maximum distance *Y* (*Y*>0) that Elvis can cross and the time *T* Viperkeeper is distracted (*T*>0). *N* lines follow, each containing the x and y coordinates of each anchor point (-100<= x,y<=100) separated by whitespace. The first anchor point is the location of Elvis, and the last anchor point is the location of the window.

Output

For each test case you should output the spare time before Elvis's escape and Viperkeeper noticing he is gone, with an accuracy of three decimals. If escape is impossible, i.e. there is no way to reach the window, or the time > *T*, you should output the string IMPOSSIBLE.

Sample Input/Output

elvis.in

```
2
9 1 2 10
0 0
1 1
1 3
2 2
3 1
3 3
3 4
4 2
4 4
11 1 2 10
0 0
1 1
2 1
1 3
2 4
3 3
3 4
4 2
4 3
2 2
5 4
```

OUTPUT

```
0.343
IMPOSSIBLE
```

F: Sales Craze

Program:	sales.(cpp|java)
Input:	sales.in
Balloon Color:	gold

Description

Sales are a red cloth in the face of Mrs. K. She cannot have enough of them. Today she is in a big mall and she is planning to spend every last penny of the X money she has on her. However she would like to spend it on the items with the higher price reduction, so at the end of the day she must have saved the most amount of money possible.

The items on sale are provided in the form of a pair of numbers: sale price *P* and percent reduction *R* from the original price. The sale price is the result of reducing the original price *O* by the percent reduction *R*. The saving is obviously the difference *O-P*. Each item can be purchased only once.

Your task is to calculate the amount of money saved, to an accuracy of a penny.

Input

The input file starts with a number T ($0 < T < 100$) that represents the number of test cases in the file. Each test case starts with a line that contains a floating point number C (the total cash available) and an integer N ($0 < N <= 30$) which represents the total number of items on sale. N lines follow each containing two numbers for the corresponding item: the sale price (positive floating point number) and the sale percent (non-negative integer less than 100).

Output

For each test case you should output the maximum total amount of money saved, to an accuracy of one decimal digit.

Sample Input/Output

sales.in

```
2
100.0 3
50 10
50 15
50 25
20.5 10
21 11
200 20
11 1
20 2
30 9
40 7
50 5
60 6
7 3
8 50
```

OUTPUT

```
25.5
8.2
```

G. Cairo Pentagonal Tiling

Program:	`cairo.(cpp|java)`
Input:	`cairo.in`
Balloon Color:	`white`

Description

The Cairo pentagonal tiling is a decomposition of the plane using semi-regular pentagons. Several streets in Cairo are paved using variations of this design. See Figure 1.a below for an example. The basic unit of tiling is a pentagonal one as illustrated in Figure 1.b.

In this problem, a tiling area is obtained by fixing two tiles base-to-base in two possible orientations as illustrated in Figure 2.a. Each pair is set as horizontal or vertical, and is composed of two tiles, where each tile is either light (white color in figure 2.b.) or dark (grey color in figure 2.b). Tiling units shown in Figure 2.a are fixed beside each another in a horizontal line alternatively, in order to form a row of tiles. Then, another row is obtained by fixing alternative shapes vertically as well. This results in a tiling area similar to the one shown in Figure 2.b, which has 3 rows and 4 columns.

Figure 1.a. Cairo Pentagonal Tiling

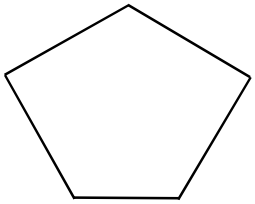
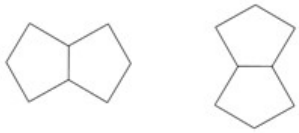


Figure1.b. Geometry of each pentagon



Horizontal pair and Vertical pair

Figure 2.a. Basic units for Tiling

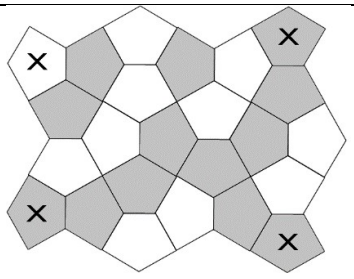


Figure 2.b. example of tiling area, and corner tiles marked with X.

In this problem, a Cairo tiling area (similar to Figure 2.b) must always start with a horizontal pair at the top left, with a minimum of two rows and two columns (i.e. 4 pairs, or 8 tiles). We define corner tiles to be the ones marked with X in the examples in Figure 2.b. Note that if the area has 3 columns, then, the top right corner will be the right tile of the last horizontal pair. A path between two corner tiles exists if there is at least one set of adjacent pieces of the same color that connects the two corners. Tiles are considered adjacent if they share an edge, not just a corner. i.e., you would like to reach one

corner from another, and you are only allowed to move from one tile into another adjacent of the same color. Figure 3 below shows a possible path between two corners for example Figure 2.b. Multiple paths between the same two corners are considered as one single path in this problem, i.e., we are not interested in unique paths, but in the existence of a path that connects the two corners.

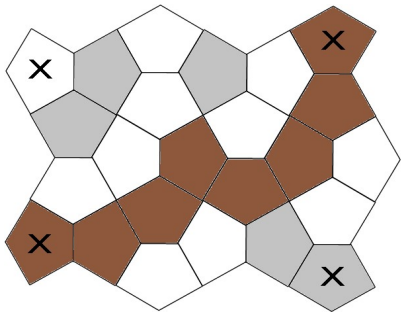


Figure 3. A possible path between two corners for example in Figure 2.b.

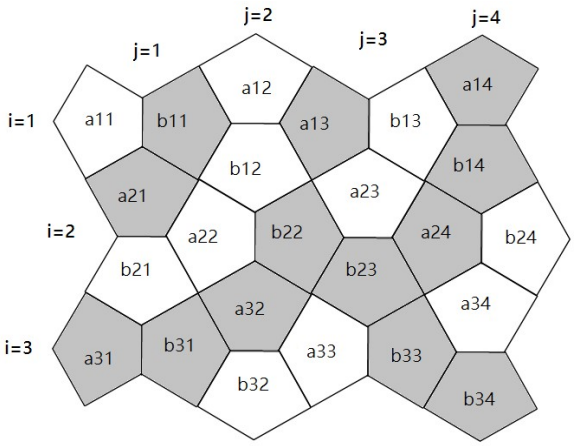


Figure 4. Tiles description using two matrices.

In this problem, you are given a description of a tiling area as matrix of pairs, with a number of columns and rows, and you are required to find how many pairs of corner tiles are connected by a path. Note that since there are 4 corner tiles, there can be a maximum of 6 pairs of connected paths. Alternatively, there can be 0 paths.

The description of the tiling area is given using two binary matrices, **a**, and **b** with values: 0 represents a white tile, and 1 represents a grey tile. For a tiling area of *N* rows by *M* columns, there are `NxM` pairs of tiles (units from figure 2.a.). Therefore, there is `2xNxM` total tiles. These are described using two `NxM` matrices. Matrix **a** defines tiles at the left side and top side of the pairs, and matrix **b** that defines tiles at the right side and bottom side of the pairs. Figure 4 below illustrates how the two matrices are used to map **a** and **b** to the example in Figure 2.b. Therefore, the example in Figure 2.b. can be described using the following matrices, (it will be used as first example in sample input below)

$$\text{Tiling area} = \left\{ \begin{matrix} (0,1)\ (0,0)\ (1,0)\ (1,1) \\ (1,0)\ (0,1)\ (0,1)\ (1,0) \\ (1,0)\ (1.0)\ (0,1)\ (0,1) \end{matrix} \right\} \quad a = \left\{ \begin{matrix} 0\,0\,1\,1 \\ 1\,0\,0\,1 \\ 1\,1\,0\,0 \end{matrix} \right\} \quad b = \left\{ \begin{matrix} 1\,0\,0\,1 \\ 0\,1\,1\,0 \\ 1\,0\,1\,1 \end{matrix} \right\}$$

Input

The input file will starts with a positive integer *k* represents the number of problem instances. Each one starts with two integers *N* and *M* followed by two NxM matrices of 0's and 1's separated by spaces, *N* rows for **a**, followed by *N* rows for **b**, one row on each line, where $2 \leq M,N \leq 50$.

Output

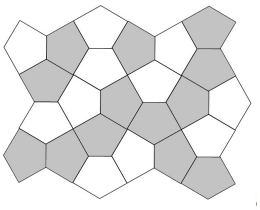
For every problem instance, print out the problem instance number (starting from 1), followed by '.', a single space and the number of pairs of connected corner tiles using a single integer, each on a new line (between 0 and 6 inclusively).

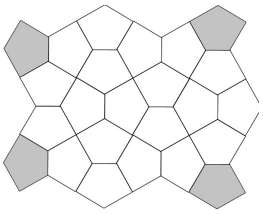
Sample Input/Output

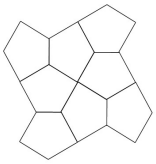
cairo.in

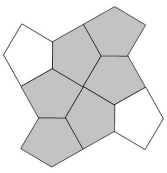
```
4
3 4
0 0 1 1
1 0 0 1
1 1 0 0
1 0 0 1
0 1 1 0
1 0 1 1

3 4
1 0 0 1
0 0 0 0
1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 1

2 2
0 0
0 0
0 0
0 0

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

Illustration of sample inputs





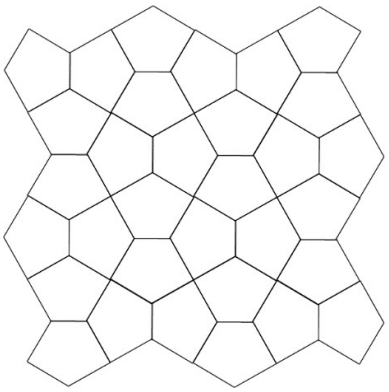


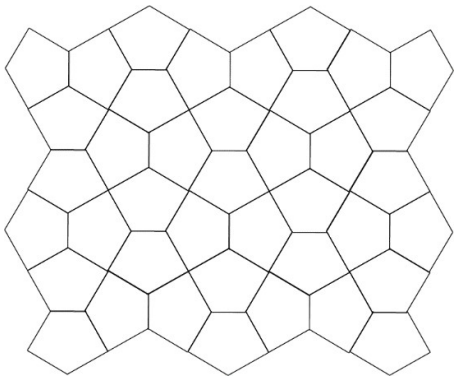


OUTPUT

```
1. 3
2. 0
3. 6
4. 1
```

4X4 and 4X5 areas that you may use for illustration





H. Armstrong numbers

Program:	armstrong .(cpp|java)
Input:	armstrong .in
Balloon Color:	pink

Description

A number that has *n* digits is considered an *Armstrong* number if the summation of every digit raised to the power *n* is equal to the same number:

The number 370: $3^3+7^3 + 0^3 = 27 + 343 + 0 = 370$.

$1634 = 1^4 + 6^4 + 3^4 + 4^4$

24 is not an Armstrong Numbers as $2^2 + 4^2 = 20$

Given a set of number, state if each one is Armstrong or not.

Input

The input file will starts with a positive integer *k* represents the number of problem instances. Followed by *n* numbers, where $0 \leq n \leq 9999999$.

Output

For every problem instance, print out the instance number starting from 1, followed by a period '.' and "yes" if the number is an Armstrong number or "no" if it is not (without quotations).

Sample Input/Output

armstrong.in

```
5
1634
150
24
407
10
```

OUTPUT

```
1. yes
2. no
3. no
4. yes
5. no
```

I. Amicable Numbers

Program:	`amicable .(cpp|java)`
Input:	`amicable .in`
Balloon Color:	`purple`

Description

Amicable numbers are two different numbers where the following holds:
- The first number is equal to the sum of the proper divisors of the second number.
- The second number is equal to the sum of the proper divisors of the first number.

Note that a proper divisor of a number *n* is a divisor of a number *n*, excluding *n* itself. For example, 220 and 284 are amicable numbers. The proper divisors of 220 are 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 and 110; their sum is 284. The proper divisors of 284 are 1, 2, 4, 71 and 142; their sum is 220. Write a program that reads pairs of numbers and decides whether each pair is amicable.

Input

The input starts with a line with one integer $1 \leq nP \leq 100{,}000$, which is the number of pairs. Each of the next *nP* lines of the input file consists of two integers *a* and *b*. All integer pairs are positive and less than 1,000,000.

Output

For each pair of integers *a* and *b* in the input, you should print whether the pair is amicable.

Display the following message for amicable pairs.

```
a b are amicable numbers
```

Display the following message for pairs that are not amicable.

```
a b are not amicable numbers
```

Sample Input/Output

amicable.in

```
2
220 284
1324 8809
```

OUTPUT

```
220 and 284 are amicable numbers
1324 and 8809 are not amicable numbers
```

J. Number Pyramid

Program:	`pyramid .(cpp|java)`
Input:	`pyramid .in`
Balloon Color:	`navy`

Description

Number Pyramid is a great game for young children to practice their addition and subtraction skills and is much liked by teachers as a pastime in class.

In this game the player is supposed to find their way out of the pyramid by finding the missing value at the top of the pyramid. Every brick (except for the bottom row) is the sum of the two bricks on which it rests. For example, take the following pyramid (Figure a) of height 4:

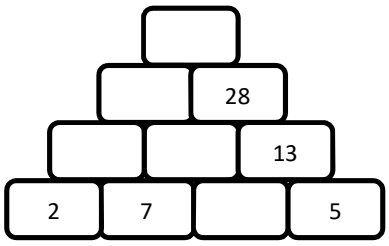


Figure **a**

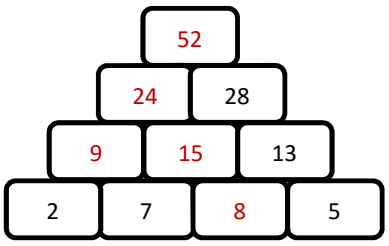


Figure **b**

To find the way out the player must compute all needed missing cells to find the top value, as shown in Figure (b).

Since you love a challenge, you decide to write a program that solves this game.

Input

Your program will be tested one or more test cases. Each test case will be specified using d+1 lines. The first line specifies an integer d (1 <= d <= 32) denoting the height of the pyramid. The following d lines describe the contents of every level of the pyramid. Values are separated by one or more spaces. For every value v at the base of the pyramid (last row) -10000 <= v <= 10000. Missing values are denoted using a dash '-'. The last line of the file contains a single -1, which is not part of the data set.

Output

For each test case print one line of output in the form:

```
k: result
```

where k is the test case number (starting at 1), and result is the sum at the top of the pyramid. If the pyramid wasn't solvable you print "`Not solvable`". Separate your result from the colon (:) by a single space.

Sample Input/Output

pyramid.in

```
4
-
-  28
-  -  13
2  7  -  5
4
-
-  7
-  -  -
2  3  -  4
-1
```

OUTPUT

```
1: 52
2: Not solvable
```

Note : The example shown in Figure (a) is the first test case in the sample I/O.

K. Exotic Election

Program:	`election.(cpp|java)`
Input:	`election.in`
Balloon Color:	`black`

Description

US Election is always creating a big fuzz. And this time it is more exciting! Because you are writing a program to predict (well, sort of) the winner. However, the program needs some initial data. To simplify the assumptions, here is some background information. There are only two candidates (*Bob* and *Alice*) and there are *K* states in the US. The president is elected through direct voting by *Electoral College* (EC) members. Each state *i* has V_i number of EC members and each member can cast one vote. The voting rule is: *winner takes all*, i.e., all the EC votes of a state goes to either Bob or Alice. So, for example, if state *M* has 20 EC votes, then either Bob will get 20 votes and Alice will get 0 vote (in this case Bob wins state M); or Bob will get 0 vote and Alice will get 20 votes (in this case Alice wins state M). A candidate wins the election if (s)he gets at least $n/2 + 1$ EC votes, where *n* is the total number of EC votes in all the states of US (i.e., sum of V_i). The input to your program will be a list of states where Bob has won and a list of states where Alice has won. You will have to find which state(s) Bob *cannot lose* if he wants to win the election and also which state(s) the Alice *cannot lose* to win the election.

Input

The first line will contain an integer $n > 0$ denoting the number of test cases to follow. Each test case will start with an integer K ($5 < K <= 100$) representing the number of states in the US, followed by a line containing the abbreviated (two letter, uppercase) names of the *K* states. The following line contains *K* integer numbers *Vi* ($1 <= i <= K$), denoting the electoral votes for state *i*. The next line will contain an integer *q* ($0 <= q < K$) denoting the number of states won by Bob, followed by *q* state names. The following line will contain an integer *r* ($0 <= r < K$) denoting the number of states won by Alice, followed by *r* state names. A state that has not been won yet by any candidate will be denoted as a *swing state*.

Output

For each test case in the input, you will have to output three lines: the first line reporting the case number as follows: Case X, where X is the case number. The next line reports only the *swing states* (in lexicographically ascending order) where Bob cannot lose if he wants to win the election (or NONE, if there is no such state), the last line reporting only the *swing states* (in lexicographically ascending order) where Alice cannot lose if she wants to win the election (or NONE, if there is no such state). By the way, a candidate may have already won the election even if there are some swing states (because of winning at least n/2 +1 votes already!). If this is the case, then there should be only two lines of output: first line – the case number and second line - "X is the winner", where X is either "Bob" or "Alice".

Sample Input/Output

flight.in

```
2
5
TX CA PA VA NC
32 30 20 9 10
1 TX
1 CA
5
TX CA PA VA NC
32 30 20 12 15
2 TX CA
1 PA
```

OUTPUT

```
Case 1
NONE
PA
Case 2
Bob is the winner
```

Explanation of the sample input and sample output

In the case 1, total votes = 101, so minimum 51 votes are needed to win.
Bob won TX, so he has 32 votes and needs 19 more to win
Alice won CA, so she has 30 votes and needs 21 more to win
If Alice loses NC, she can still win the election by winning PA and VA
But if she loses PA, then there is no way to win because VA+NC = only 19 votes but she needed 21.
So, Alice must win PA to be the winner.
Bob, on the other hand can lose any state (e.g. lose PA) but still win the election (by winning VA and NC). So, there is no state that Bob must have to win.
In case 2, Bob already won 62 votes, so he is the winner.

L. The Digits

Program:	digits.(cpp|java)
Input:	digits.in
Balloon Color:	green

Description

The 10 digits are spelled out respectively as follows: “Zero”, “One”, “Two”, “Three”, “Four”, “Five”, “Six”, “Seven”, “Eight”, and “Nine”. In this problem, you are given a number N and your task is to count the number of characters used if the digits of this number are spelled out. For instance, if N is 254, the digits representation spelled out would be “TwoFiveFour”, and the number of characters used would be 11.

Input Format

The input starts with a number T ($1 \leq T \leq 1{,}000$) that represents the number of test cases in the file. Each test case consists of a line and integer N ($0 \leq N \leq 10^{100}$).

Output Format

The output for each test case is in this form:

```
k. ans
```

where ***k*** represents the test case number (starting at 1), and ***ans*** is the number of characters required to spell out the digits of the number N.

Sample Input / Output

digits.in

```
2
254
125
```

OUTPUT

```
1. 11
2. 10
```