ACM International Collegiate Programming Contest
2012 East Central Regional Contest
Grand Valley State University
University of Cincinnati
University of Windsor
Youngstown State University
November 3, 2012

Sponsored by IBM

Rules:

1. There are **nine** problems to be completed in **five hours**.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
3. No whitespace should appear in the output except between printed fields.
4. All whitespace, either in input or output, will consist of exactly one blank character.
5. The allowed programming languages are C, C++ and Java.
6. All programs will be re-compiled prior to testing with the judges' data.
7. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contestant officials (e.g., that might generate a security violation).
8. The input to all problems will consist of multiple test cases.
9. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
10. All communication with the judges will be handled by the PC^2 environment.
11. Judges' decisions are to be considered final. No cheating will be tolerated.

Problem A: Babs' Box Boutique

Babs sells boxes and lots of them. All her boxes are rectangular but come in many different sizes. Babs wants to create a really eye-catching display by stacking, one on top of another, as many boxes as she can outside her store. To maintain neatness and stability, she will always have the sides of the boxes parallel but will never put a box on top of another if the top box sticks out over the bottom one. For example, a box with base 5-by-10 can not be placed on a box with base 12-by-4.

Of course the boxes have three dimensions and Babs can orient the boxes anyway she wishes. Thus a 5-by-10-by-12 box may be stacked so the base is 5-by-10, 5-by-12, or 10-by-12.

For example, if Babs currently has 4 boxes of dimensions 2-2-9, 6-5-5, 1-4-9, and 3-1-1, she could stack up to 3 boxes but not all four. (For example, the third box, the first box, then the last box, appropriately oriented. Alternatively, the second box could replace the third (bottom) box.)

Babs' stock rotates, so the boxes she stacks outside change frequently. It's just too much for Babs to figure out and so she has come to you for help. Your job is to find the most boxes Babs can stack up given her current inventory. Babs will have no more than 10 different sized boxes and will use at most one box of any size in her display.

Input

A positive integer n ($n \leq 10$) will be on the first input line for each test case. Each of the next n lines will contain three positive integers giving the dimensions of a box. No two boxes will have identical dimensions. None of the dimensions will exceed 20. A line with 0 will follow the last test case.

Output

For each test case, output the maximum number of boxes Babs can stack using the format given below.

Sample Input

```
4
2 2 9
6 5 5
1 4 9
3 1 1
3
2 4 2
1 5 2
3 4 1
0
```

Sample Output

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

Problem B: Flash Mob

Jumping Jack is in charge of organizing a flash mob. The members of the flash mob move around town all day and part of the appeal of this group is they come together to do their thing whenever the mood strikes Jack. When the mood does strike, Jack sends a text message to the members to meet at a particular intersection in town in exactly one hour. The streets of the town run only north-south or east-west and are evenly spaced, forming a perfect grid like a sheet of graph paper. Due to the spontaneity, Jack wants to minimize the inconvenience and so picks an intersection to minimize the total distance traveled by the flash mob's members. Fortunately Jack has the locations of all the members via the GPS on their cell phones. Your job is to find the meeting location given all the members' locations.

Each intersection will be given by a pair of non-negative integers; the first coordinate is the east-west street and the second coordinate is the north-south street. The location of each flash mob member will be an intersection. Members can travel only north-south or east-west between intersections.

For example, suppose there are 5 mob members at locations $(3,4), (0,5), (1,1), (5,5)$, and $(5,5)$. Then if Jack summons them all to location $(3,5)$, the total number of blocks traveled by the mob members would be 14. Jack could do no better – but sometimes the 'best' location may not be unique.

Input

Input for each test case will be a series of integers on one or more lines. The first integer, n ($1 \leq n \leq 1000$), indicates the number of mob members. There follow n pairs of integers indicating the location (intersection) of each member. The location coordinates are both between 0 and 10^6, inclusive. More than one member may be at the same intersection. A line containing 0 will follow the last test case.

Output

Output one line for each test case in the format given below. The ordered pair is the coordinates of the location in the city where the total distance traveled (in blocks) is minimal. If there is more than one such location, output the one with the smallest first coordinate. If there is more than one 'best' location with the smallest first coordinate, output the one of those with the smallest second coordinate. The total number of blocks traveled by all the mob members follows the location.

Sample Input

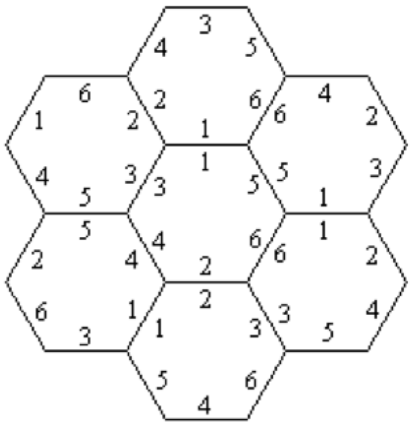
```
5 3 4 0 5 1 1 5 5 5 5
4 100 2 100 2 100 2 1 20000
0
```

Sample Output

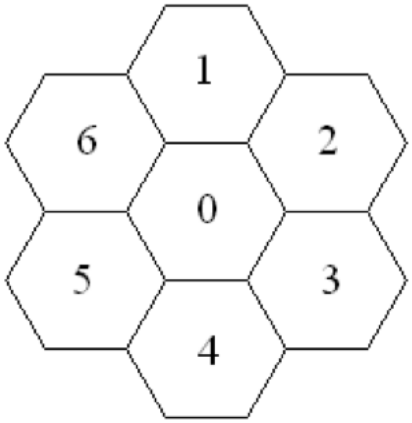
```
Case 1: (3,5) 14
Case 2: (100,2) 20097
```

Problem C: Hexagon Perplexagon

A well known puzzle consists of 7 hexagonal pieces, each with the numbers 1 through 6 printed on the sides. Each piece has a different arrangement of the numbers on its sides, and the object is to place the 7 pieces in the arrangement shown below such that the numbers on each shared edge of the arrangement are identical. Figure (a) is an example of one solution:



(a) Example Solution



(b) Position Notation for Output

Rotating any solution also gives another trivially identical solution. To avoid this redundancy, we will only deal with solutions which have a 1 on the uppermost edge of the central piece, as in the example.

Input

The first line of the input file will contain a single integer indicating the number of test cases. Each case will consist of a single line containing 42 integers. The first 6 represent the values on piece 0 listed in clockwise order; the second 6 represent the values on piece 1, and so on.

Output

For each test case, output the case number (using the format shown below) followed by either the phrase `No solution` or by a solution specification. A solution specification lists the piece numbers in the order shown in the Position Notation of Figure (b). Thus if piece 3 is in the center, a 3 is printed first; if piece 0 is at the top, 0 is printed second, and so on. Each test case is guaranteed to have at most one solution.

Sample Input

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

Sample Output

```
Case 1: 3 0 5 6 1 4 2
Case 2: No solution
```

Problem D: I've Got Your Back(gammon)

A friend of yours is working on an AI program to play backgammon, and she has a small problem. At the end of the game, each player's 15 pieces are moved onto a set of 6 board positions called *points*, numbered 1 through 6. The pieces can be distributed in any manner across these points: all 15 could be on point 3; 5 could be on point 6, 2 on point 5, 3 on point 4 and 5 on point 2; etc. Your friend wants to store all these possible configurations (of which there are 15504) into a linear array, but she needs a mapping from configuration to array location. It seems logical that the configuration with all 15 pieces on point 1 should correspond to array location 0, and the configuration of all 15 pieces on point 6 should correspond to the last array location. It's the ones in between that are giving her problems. That's why she has come to you.

You decide to specify a configuration by listing the number of pieces on each point, starting with point 6. For example, the two configurations described above could be represented by $(0, 0, 0, 15, 0, 0)$ and $(5, 2, 3, 0, 5, 0)$. Then you can order the configurations in lexicographic ordering, starting with (0,0,0,0,0,15), then $(0, 0, 0, 0, 1, 14), (0, 0, 0, 0, 2, 13), \ldots, (0, 0, 0, 0, 14, 1), (0, 0, 0, 0, 15, 0), (0, 0, 0, 1, 0, 14)$, $(0, 0, 0, 1, 1, 13)$, etc., ending with $(15, 0, 0, 0, 0, 0)$. Now all you need is a way to map these orderings to array indices. Literally, that's all you need, because that's what this problem is all about.

Input

Each test case will consist of one line, starting with a single character, either `'m'` or `'u'`. If it is an `'m'` it will be followed by a configuration and you must determine what array index it gets mapped to. If it is a `'u'` then it will be followed by an integer array index i, $0 \leq i < 15504$, and you must determine what configuration gets mapped to it. A line containing the single character `'e'` will end input.

Output

For each test case, output the requested answer – either an array index or a configuration. Follow the format in the examples below.

Sample Input

```
m 0 0 0 0 0 15
u 15503
e
```

Sample Output

```
Case 1: 0
Case 2: 15 0 0 0 0 0
```

Problem E: Parencedence!

Parencedence is a brand new two-player game that is sweeping the country (that country happens to be Liechtenstein, but no matter). The game is played as follows: a computer produces an arithmetic expression made up of integer values and the binary operators `'+'`, `'-'` and `'*'`. There are no parentheses in the expression. If Player 1 goes first he/she can put parentheses around any one operator and its two operands; the parenthesized expression is evaluated and its value is used in its place. Player 2 then does the same, and the game proceeds accordingly, Player 1 and Player 2 alternating turns. Player 1's object is to maximize the final value, while Player 2's object is to minimize it. A sample round might go as follows:

Initial expression:	3-6*4-7+12		
Player 1's move:	3-6*(4-7)+12	→	3-6*-3+12
Player 2's move:	(3-6)*-3+12	→	-3*-3+12
Player 1's move:	(-3*-3)+12	→	9+12
Player 2's move:	(9+12)	→	21

A game of Parencedence is played in two rounds, each using the same initial unparenthesized expression: in the first round, Player 1 goes first, and in the second, Player 2 goes first (Player 1 is always trying to maximize the result and Player 2 is always trying to minimize the result in both rounds, regardless of who goes first). Let r_1 be the result of the first round and r_2 the result of the second round. If $r_1 > -r_2$, then Player 1 wins; if $r_1 < -r_2$ then Player 2 wins; otherwise the game ends in a tie. Your job is to write a program to determine the final result assuming both players play as well as possible.

Input

The first line of the input file will contain an integer n indicating the number of test cases. The test cases will follow, one per line, each consisting of a positive integer $m \leq 9$ followed by an arithmetic expression. The value of m indicates the number of binary operators in the arithmetic expression. The only operators used will be `'+'`, `'-'` and `'*'`. The `'-'` operator can appear as both a unary and a binary operator. All binary operators will be surrounded by a single space on each side. There will be no space after any unary `'-'`. No combination of parentheses will ever result in an integer overflow or underflow.

Output

For each test case, output the case number followed by three lines. The first contains the first set of operands and operator to be parenthesized in round 1 (when Player 1 goes first) and r_1. The second line contains the analagous output for round 2. The third line contains either the phrase "Player 1 wins", "Player 2 wins" or "Tie" depending on the values of r_1 and r_2. In the first two output lines if there is a choice between which operator should be parenthesized first, use the one which comes earliest in the original expression. Follow the format used in the examples.

Sample Input

```
2
4 3 - 6 * 4 - 7 + 12
2 45 - -67 - 3
```

Sample Output

```
Case 1:
Player 1 (7+12) leads to -2
Player 2 (3-6) leads to -27
Player 2 wins
Case 2:
Player 1 (-67-3) leads to 115
Player 2 (45--67) leads to 109
Player 1 wins
```

Problem F: Road Series

Don and Jan spend a lot of time together on the road. To pass the time, they've invented various games they can play, many of which involve license plates and road signs. One of their favorites is Road Series. The object is to find the number 1 on some sign, then 2, then 3, and so on. When they get to two digit numbers the two digits must appear directly next to each other on the sign or license plate, and any sign or license plate can provide multiple answers. For example, if they see a sign with the characters "678-43 15", they can use the numbers 67, 78, 43 and 15 but not 84 (dash between the two digits) or 31 (space between the two digits). They could also use the individual digits 6, 7, 8, 4, 3, 1 and 5 as well as the three digit number 678 (if they got up that high)

When they first started playing the game, they had very strict rules about when you could find a number, namely you weren't allowed to find a number n until all the numbers 1 through $n-1$ were already found. They soon found that this made the game REALLY slow, so they modified the game as follows. First, they called a number n the *last complete number* if it was the highest number such that all the numbers from 1 to n had been found. (Initially, 0 is the last complete number.)

Given this, Don and Jan allowed themselves to keep track of having seen some numbers beyond the last complete number n, so long they weren't too much bigger than n. More precisely, they can keep track of which numbers they've seen in a *window* of size w beyond n. This allows them to remember any number they've seen up to $n + w$.

For example, suppose $w = 4$ and and the last complete number Don and Jan have seen is 19. When they see the following sign:

> "Show time at 8:25, no one under 21 admitted"

they can use the 21, but not the 25 (since it's not in the window). If this sign is followed by the sign:

> "The FleaBag Hotel, phone 555-2520"

they can use the 20, which now makes 21 the last complete number, and thus can also use the 25, since it's now in the window.

Input

The first line of the input file will contain an integer m indicating the number of test cases. Each test case will start with a pair of positive integers k w, where $k \leq 1000$ indicates the number of signs seen by Don and Jan, and $w \leq 100$ indicates the window size. Following that will be k lines each consisting of text from a sign. Each line of text may contain any combination of alphanumeric characters, punctuation and spaces, and will have length at most 1000. Input for each sign will end with a new line.

Output

For each test case, output the case number followed by the last complete number that can be found using the signs, followed by the highest number seen within the window.

Sample Input

```
4
2 10
13
2
2 10
2
13
1 8
Tomorrow, from 12 to 2, 4-4 basketball tournament! $3 entry fee.
1 8
Tomorrow, from 11 to 7, 4-4 basketball tournament! $3 entry fee.
```

Sample Output

```
Case 1: 3 3
Case 2: 3 13
Case 3: 4 12
Case 4: 1 7
```

Problem G: Show Me the Money

Frank Marks works at the Business Office of a large company. His company has customers all over the world and must deal with many different currencies. Employees often come to the Business Office with requisitions for certain amounts of money, such as 100 American dollars or 452 Euros. If Frank has the cash on hand, he gives the employee exactly what they need; if he does not have enough of the particular currency requested, he substitutes it with another one. This is sometimes difficult because he may have many different currencies to choose from and would like to pick the one which allows him to get as close to the original requisition as possible without going under (he must provide at least the value requested). For example, suppose Frank has six difference currencies – A, B, C, D, E and F – and he is aware of the following exchange rates:

$$\begin{aligned} 23\ A &= 17\ B \\ 16\ C &= 29\ E \\ 5\ B &= 14\ E \\ 1\ D &= 7\ F \end{aligned}$$

If a requisition for 100 A comes in but Frank has less than 100 A available, he can substitute with either 74 B (equivalent to about 100.12 A), 115 C (equivalent to about 100.72 A) or 207 E (equivalent to about 100.02 A). Thus, the best approximation available to him is 207 E. Note that Frank does not have enough information to find a relationship between currencies A and D or currencies A and F. Also, Frank only has at most 100,000 units of any one currency in stock, so he could not satisfy a request for 64,000 A using E currency and would need to use 73078 C instead.

Determining the ideal substitute currency to use when he has completely run out of the requested currency is time consuming, so Frank would like a program to do the calculations for him.

Input

Each test case begins with a positive integer n indicating the number of exchange rates. The next n lines will be of the form

$$val_1\ name_1\ =\ val_2\ name_2$$

where $name_1$ and $name_2$ are the names of two distinct currencies, and val_1 and val_2 are positive integers ≤ 30 specifying the ratio between the two currencies. There will be no more than 8 distinct currency names, and any two currencies will be paired together at most once. Currency names will consist of up to ten alphabetic characters. There will be no inconsistencies in the input (such as $1A = 2B$, $1B = 2C$ and $1C = 2A$). Following these n lines will be a single line of the form

$$val\ name$$

which specifies the amount (a positive integer not exceeding 100,000) and the name of the currency requested. A line containing 0 will follow the last test case.

Output

For each test case, print the case number and the closest approximation without going under the requested value assuming that Frank does not have any of the requested currency available but is fully in stock of all other currencies. There will be a unique answer for each problem instance.

Sample Input

```
4
23 A = 17 B
16 C = 29 E
5 B = 14 E
1 D = 7 F
100 A
1
1 shekel = 2 quatloo
40 quatloo
0
```

Sample Output

```
Case 1: 207 E
Case 2: 20 shekel
```

Problem H: Sofa, So Good

Sofia's Sofa Salon specializes in synthesizing and supplying spectacular specialty sofas. Their manufacturing process has been couched in mystery until recently, when it was revealed that the creation of a sofa is divided into two phases: framing and upholstering. When an order for a set of sofas comes in, the management team uses their knowledge of how quickly any particular worker can frame any given sofa to decide which worker will frame which sofa so as to minimize the total time spent framing. Management always ensures there are exactly enough workers on the job to handle the order being filled, and each worker can frame only one sofa per order (due to a recurring series of repetitive motion injury claims).

Once the workers have begun framing, management performs an analysis of the upholstering phase knowing the time it takes each worker to upholster any given sofa. They must assign each worker exactly one sofa to upholster (potentially different from the one they framed) when they finish their framing. A worker may have to wait until the sofa they are assigned to upholster is finished being framed by another worker, so they start work on it once it's available. If they are idle between the framing and upholstering phases they usually relax in the company's recreational facility where they play tag (this room is called the chase lounge). Management tried to stop them from playing this game, but they were stymied by a sign reading "Do not remove this tag."

Since the workers are paid by the hour whether they are working or playing tag, management would like to assign them sofas to upholster so as to minimize the total number of hours that the workers are on-site (given the framing assignment determined earlier and the fact that each worker leaves as soon as they finish upholstering). Management is not prepared to solve such a deep-seated problem, so they have requested that you furnish a program to help them.

Input

Each test case will start with a positive integer $n \leq 50$ indicating both the number of sofas and the number of workers (no workers get to sit out). The following n lines each contain n positive integers; the i^{th} value on the j^{th} line indicates the time it takes the worker j to frame sofa i (workers and sofas are numbered starting at 1). Following this are n more lines each containing n positive integers describing the upholstering times in a similar format. All times given will be no greater than 1000. A line containing a single 0 will terminate the input.

Output

For each test case, output the case number followed by n lines, one per worker (starting with worker 1). For each worker, list the sofa they framed, the sofa they upholstered, and the time at which they finished their upholstering work, assuming all workers start framing at time 0. There will be exactly one optimal pairing of workers and couches for each phase. Following these lines, output the total idle time spent by the workers in the chase lounge during that order. Use the format shown in the example.

Sample Input

```
4
8 6 12 19
13 2 18 10
9 15 16 17
5 18 4 10
2 6 3 3
8 5 9 2
5 8 4 3
4 4 5 2
0
```

Sample Output

```
Case 1:
Worker 1: 2 3 9
Worker 2: 4 4 12
Worker 3: 1 1 14
Worker 4: 3 2 10
Total idle time: 2
```

Problem I: Town Square

Felix J. Humble, a nerdy but rich resident of a small Midwest town, has erected 4 statues in his honor in a public park (which he happens to own). Felix is very proud of these statues, but is now worried about vandals, small children with bubble gum, and the occasional dog with a urinary tract infection. To solve this problem, he has decided to build a fence around the statues (Felix owns the local fencing company as well). For various aesthetic reasons he would like the following conditions met:

1. The enclosed area must be a square.
2. The distance between each statue and its nearest fence side should be 5 feet.
3. No two statues should have the same nearest fence side.

After working for a grand total of 12 seconds, Felix has realized that he has no idea of the length of fence needed, or even if the above conditions can be met. Since he is planning similar tributes in other parks he owns, he would like someone to write a program to solve this problem (which Felix will later take credit for, of course).

Input

The first line of the input file will contain an integer n indicating the number of test cases. The test cases will follow, one per line, each consisting of eight integer values giving the x and y coordinates for the first, second, third and then fourth statue. All values will be in units of feet, and will lie between -100 and 100. No two statues will be at the same location.

Output

For each test case, output the case number followed by one of two answers: 1) If there is a solution, output the side length of the enclosing square. If there are multiple solutions, output the side length of the solution of maximum size; 2) If it is not possible to build a square fence which meets Felix's conditions, output `no solution`. All numerical output should be rounded to the nearest hundredth of a foot. Follow the format in the examples below.

Sample Input

```
2
0 1 1 0 3 4 4 2
0 1 0 2 0 3 0 4
```

Sample Output

```
Case 1: 14.00
Case 2: no solution
```