Fantasy Baseball

Filename: `baseball`

Each year the state of Florida picks a Most Valuable Player (MVP) amongst all of its baseball players. Last year however, many were outraged when the winner became the child of one of the committee selection members. Shortly thereafter, the committee was disbanded and it was decreed that it would be replaced by a more objective selection process: a computer program! Luckily, you've learned to program and are vying for the job, which pays much more than your current gig at the local grocery store. You have been told that the MVP should be the player with the highest batting average. If there are two such players, ties should be broken by on-base percentage. If two players have the same maximal batting average and on base percentage, then those ties should be broken by slugging percentage. You are guaranteed that no two players will have identical statistics for all three measures.

All of the statistics described above are based on "at bats" and "hits". Each player has a certain number of plate appearances, of which a subset of them are "at bats". A subset of the "at bats" are hits. The rest of the "at bats" are "outs." Each plate appearance that is NOT an "at bat" can not result in a hit.

Whenever a player has a "plate appearance" one of the following outcomes is possible. These abbreviations will be the same codes used in the input format.

The following four outcomes represent hits:
1B: single, a 1 base hit
2B: double, a 2 base hit
3B: triple, a 3 base hit
HR: home run, a 4 base hit

The following outcomes are neither a hit nor an "at bat":

BB: walk, which counts as getting on base
SAC: sacrifice, which does not count as getting on base

The following three outcomes count as "at bats" but are not hits:

K: Strikeout
GO: Ground Out
FO: Fly Out

A player's batting average is simply their number of hits divided by the number of at bats. A player's on base percentage is the number of times they get on base (hits plus walks) divided by the number of plate appearances. A player's slugging percentage is the number of bases they earn on hits divided by the number of at bats. (Thus, walks don't affect one's slugging percentage.)

The Problem:

Given the batting histories of the best players in the state, created a sorted list of the players based on the statistics described above, ranking the players for the title of MVP.

The Input:

The first line of input will contain a single positive integer, *n (n ≤ 100)*, representing the number of seasons for which to compute the MVP. Data for each season follows. The first line of input for each season will be a positive integer, *P (P ≤ 1000),* representing the number of players vying for MVP for that season. The following *P* lines will contain data for one player each. All the data on each of these lines is space separated. The first piece of data on each of these lines is the last name of the player, a string of 20 or fewer uppercase letters. This is followed by the first name of the player, which is also a string of 20 or fewer uppercase letters. The next piece of data is the number of plate appearances, *A (A ≤ 200)*. This is followed by *A* character strings indicating the outcome of each plate appearance. (Each character string is as described previously.) All players are guaranteed to have at least one at bat.

The Output:

For each season, output a single header as follows:

```
Season #k:
```

where k represents the number of the season, starting with 1. This should be followed by the names of each player (last name, followed by a comma, a space and then the first name), one per line in the order of their finishing for state MVP. Separate the output for each season with a blank line.

Sample Input:

```
2
4
JONES JIMMY 10 1B 2B K K GO FO SAC BB HR K
ADAMS SAMUEL 9 K K K K K 1B 1B 1B K
KINGSTON MAURICE 6 1B K 1B BB GO GO
RUDWELL TOMMY 5 1B 2B BB GO FO
10
RICHARDSON CHRISTIAN 9 1B 2B 1B K K GO HR 2B 1B
SALIBA MATT 8 1B K 2B K 3B GO HR FO
TARRATS CARLOS 9 BB BB BB K FO BB BB BB 1B
SMITH TIMOTHY 10 1B 2B K 3B GO FO 1B 2B 3B HR
SMITH DERRICK 9 BB BB BB K 2B FO BB BB BB
REINOSO RYAN 8 SAC K 2B K 3B GO HR FO
HIERS JOHN 10 1B 2B K 3B GO FO 1B 2B HR HR
GUILLORME LUIS 7 BB BB BB BB BB 1B GO
BROADERICK RYAN 10 1B 2B K 3B GO K BB 2B 3B HR
```

```
HAYDEN WILSON 9 1B 2B 1B K K GO 2B 2B 2B
```

Sample Output:

```
Season #1:
RUDWELL, TOMMY
KINGSTON, MAURICE
JONES, JIMMY
ADAMS, SAMUEL

Season #2:
HIERS, JOHN
SMITH, TIMOTHY
BROADERICK, RYAN
RICHARDSON, CHRISTIAN
HAYDEN, WILSON
GUILLORME, LUIS
SALIBA, MATT
REINOSO, RYAN
SMITH, DERRICK
TARRATS, CARLOS
```

Note: *The names for the second case come from maxpreps.com. The statisics for these players in the sample are completely fictitious.*

Piece of Cake

Filename: `cake`

You have noticed that many of your school's clubs frequently order cake for their various functions. You've decided that you can get rich by selling the cakes to all the clubs at your school. Whenever you get an order for cake, you are told how many servings you must provide. A typical serving is a 1" x 1" piece. However, due to the fancy frosting you use at the edges of the cake, you would like to minimize the perimeter of the cake. Furthermore, you know that the only shape that everyone likes is rectangular, thus, you must make your cake rectangular. Finally, to make matters simple, you have decided that lengths of the sides of the cake must be a positive integer number of inches. Given these constraints, your goal is to write a program that minimizes the perimeter of a cake.

The Problem:

Given a positive integer, *A*, representing the area of a rectangle in square inches, determine the minimum perimeter the cake must have if its length and width must both be an integer number of inches.

The Input:

The first line of the input file will contain a single positive integer, *n (n ≤ 100)*, representing the number of cakes to create. The following *n* lines will each contain one positive integer, *A (A ≤ 500000000)*, representing the desired area for that particular cake in square inches.

The Output:

For each cake, simply output the desired minimum perimeter on a line by itself.

Sample Input:

```
4
1
121
96
35
```

Sample Output:

```
4
44
40
24
```

Coupon-Palooza

Filename: `coupon`

Your mother collects many coupons. She has a dilemma however. She collects SO many coupons, that she gets coupons for the same item. Luckily, since many stores are on hard times these days, they accept multiple coupons applied to the same purchase. For example, if she has a coupon for $5 off for a dinner set and 20% off and the dinner set costs $100, she can apply the $5 off first, reducing the price to 95% and then apply the 20% discount to get a final purchase cost of $76. All of her coupons are of two varieties: a fixed dollar amount taken off, or a percent discount. Given a list of all of her coupons for a particular item as well as the price of the item in dollars, determine the minimum cost she'll pay for the item if she applies her coupons in the appropriate order.

The Problem:

Given a list of discounts for an item and the original price of the item, determine the minimum price that can be achieved by applying the discounts in any order. You'll never be given a case where the ordering of the discounts could bring the cost down to $0 or below.

The Input:

The first line of the input contains a single positive integer, n $(n \leq 100)$, representing the number of items for which your mother has coupons for. The following $3n$ lines will contain the data for each item. The first line (of three) specifying each item will contain a positive real number to two decimal places, D $(D \leq 10000.00)$, representing the original cost of the item in dollars. The second line specifying each item will start with a positive integer, s $(s \leq 5)$, denoting the number of subtractive discounts. The following s items on this line, each separated by spaces, will be positive real numbers to two decimal places indicating the dollar amounts of each of these subtractive discounts. The third line specifying each item will start with a positive integer, p $(p \leq 5)$, denoting the number of percentage discounts. The following p items on this line, each separated by spaces, will be positive integers strictly in between 0 and 100, indicating the percentage off for each of these discounts. Note that the values will be such that no ordering of the discounts will bring the price to $0 or below.

The Output:

For each item, simply output the minimum cost for the item with the optimal application of your mother's coupons, in dollars. Please print this amount rounded to two decimal places.

Sample Input:

```
2
100.00
1 5.00
1 20
10000.00
1 300.00
0
```

Sample Output:

```
75.00
9700.00
```

Five Dollar Deal

Filename: `five`

You've recently taken a job at the local movie theater. Your friends are estatic to find out that you can get them tickets for five dollars! Since you're a nice person, you've decided to accept all the ticket requests from your friends, so long as they pay you for the tickets they want up front.

The Problem:

Given a number of tickets to buy (at $5 a piece), calculate the price of the tickets in dolars.

The Input:

The first line of the input will contain a single positive integer, *n (n ≤ 1000)*, representing the number of movie ticket orders you'll process. The following *n* lines will have a positive integer *T (T ≤ 1000)*, representing the number of tickets in that order.

The Output:

For each order, output a single integer representing the cost of that order in dollars on a line by itself.

Sample Input:

```
6
2
5
9
10
137
1000
```

Sample Output:

```
10
25
45
50
685
5000
```

Late Night Wake Up

Filename: `night`

One of UCF's Programming Team coaches, Arup, is a new father. While having a new child is wonderful, as everyone knows, infants wake up in the middle of the night to be fed and this drastically reduces the amount of sleep that both parents get. Arup and his wife have worked out a system where they alternate shifts. One of the two chooses to wake up the first time their daughter wakes up in the middle of the night and from that point on, they alternate. Like everyone else, Arup likes his sleep. He wants you to write a program that will help him determine whether or not he should take the first shift, given that his goal is to maximize the amount of time that he sleeps. If both strategies give him an equal amount of sleep, he prefers the strategy that gives him the longest continuity of sleep. You will not be given any scenarios where both of these values are the same for both options.

For the purposes of this problem, Arup and his wife sleep from 11pm to 8am the next morning. All of their daughter's wake up times for a single night will be given within this time span, in order. Thus, the earliest any span would start is 11pm and the latest any span will go through is 8am. Each span will be an positive integer number of minutes.

The Problem:

Given Arup's daughter's wake up schedule for a night, determine whether or not he should wake up for the odd numbered shifts (first, third, etc.) or the even numbered shifts (second, fourth, etc.)

The Input:

The first line of the input file has a single positive integer, n $(n \leq 100)$, representing the number of evenings to determine Arup's wake up schedule. The first line describing each evening will have a single positive integer, w $(w \leq 10)$, representing the number of times Arup's daughter wakes up that evening. The following w lines will contain the time his daughter wakes up and the time his daughter falls back to sleep for each of her late night feedings, separated by a space. All times will be in the form X:Ypm or X:Yam, where X is the hour given without a leading 0 and Y is the two digit representation of the minute. Within each line, the first time is guaranteed to be before the second time, the lines themselves are ordered in chronological order and the end of a wake up period will be strictly before the beginning of the next wake up period.

The Output:

For each night output a line with either the string "FIRST" or the string "SECOND", corresponding to the strategy that maximizes Arup's sleep for the input case. In the case of ties, choose the strategy that maximizes the length of the longest stretch of sleep. No case will include an option where this measure is also a tie.

Sample Input:

```
2
2
11:30pm 1:15am
3:30am 5:30am
2
2:00am 3:00am
6:30am 7:30am
```

Sample Output:

```
FIRST
SECOND
```

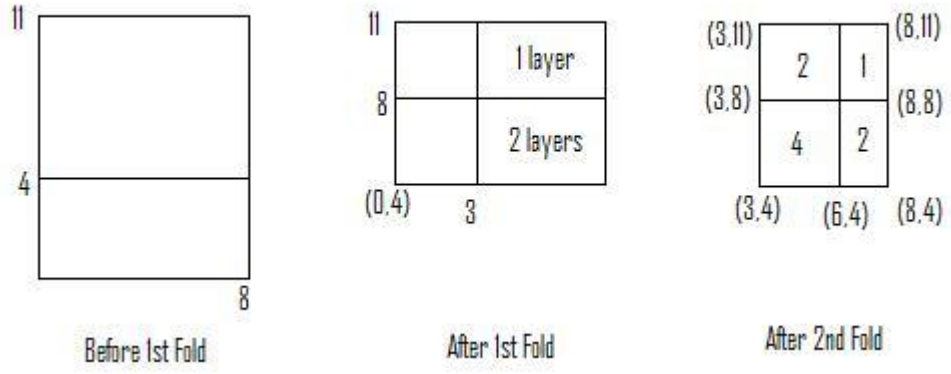
Origami

Filename: `origami`

You've recently gotten into Origami and want to write a computer program to help you out a bit. However, you've realized that the geometry of folding paper is very difficult, so you've scaled back your ambitions somewhat. Your goal is now to write a program that determines the layout of a sheet of paper that has been folded twice: once in the horizontal direction and once in the vertical directions. In particular, given a piece of paper with dimensions W x L, where the paper is initially located with its bottom left corner at (0, 0) and its top right corner at (W, L) on the Cartesian plane, along with two folding directions of the form y = w, x = l, where $0 < w < L$ and $0 < l < W$, you need to determine the areas of the resulting paper that have 1 layer of paper, 2 layers of paper and 4 layers of paper, respectively, underneath them. The first fold will be from the bottom of the paper while the second fold will be from the left side of the paper.

Consider the following example:

Let the initial paper be of size 8" x 11". Then, make the first fold across the line y = 4. This will result in one portion of the paper having two layers (with area 32 sq. in.) and another area of the paper having one layer (with area 24 sq. in.) Then, fold this resulting paper along the line x = 3. This will result in four distinct areas, one with 1 layer, two with 2 layers, and 1 with 4 layers, as illustrated below:



At the end of the sequence of moves, we see that the area of the section with 4 layers is 12, the area with the sections with 2 layers is 17, and the are with the second with 1 layer is 6.

The Problem:

Given the dimensions of a rectangular sheet of paper positioned with its lower left corner at the origin of the Cartesian plane, the horizontal line across which the first fold is made, and the vertical line across which the second fold is made, determine the area of the portions of the resulting paper that contain 1 layer, 2 layers and 4 layers, respectively.

The Input:

The first line of input will contain a single positive integer, *n (n ≤ 10000)*, representing the number of pieces of paper to fold. The input for each case follows, with one case per line. The first two values on each of these lines will be positive integers specifying the width, W and length, L, respectively of the original sheet of paper. $1 < W \leq 100$ and $1 < L \leq 100$. This is followed by two positive integers, a and b, representing the horizontal and vertical folds, respectively. These folds will be along the lines $y = a$ and $x = b$, respectively, with $0 < a < L$ and $0 < b < W$. Each of the four values on each of these lines will be separated by spaces.

The Output:

For each piece of paper, output three integers separated by spaces on a line: The area of the resulting paper with 1 layer, 2 layers and 4 layers, respectively.

Sample Input:

```
2
8 11 4 3
4 4 2 2
```

Sample Output:

```
6 17 12
0 0 4
```

Maximum Stock Return

Filename: `stock`

In your Economics class, you calculated the profit of buying and selling a stock once, over the course of a semester. However, real day traders tend to make more than one trade every season. In fact, as the term suggests, they make trades every day!!! In this problem, you want to build a better model that might earn you more money than making just one trade. To keep things simple however, at any given point in time, you're only allowed to own one stock out of two possible stocks. (You may also keep all of your money out of the market for a particular day.) If you own a particular stock, you will buy as many shares of that stock as possible.

In this problem you'll be given the daily stock history of two stocks, the initial amount of money you have to invest, as well as the cost of a transaction, known as a transaction fee. A transaction is the buying or selling of any quantity of shares of a stock.

As an example, consider the following three day data of Google stock and Apple stock:

Google: 759.68, 765.74, 770.17
Apple: 443.00, 457.82, 457.04

Imagine that we started with $10,000 and the transaction fee was $50. On day one, we buy Apple stock. The maximum we can buy is 22 shares, which costs us 22 x $443 + $50 = $9796. Thus we have $204 cash leftover. In one day, our 22 shares we own are worth 22 x $457.82 = $10072.04. Thus our current value, if we choose to cash out would be $10072.04 - $50 + $204 = $10226.04. If we then turn around at the end of that day and buy Google stock, we can get 13 shares for 13 x $765.74 + $50 = $10004.62, leaving us with $221.42 in cash. At the end of the last day, the value of our Google stock is 13 x $770.17 = $10012.21. At the end of our trading period (three days in this example) we are forced to sell the stock and our total value at the end of the three days for these set of actions is $221.42 (old cash) + $10012.21 (selling stock) - $50 (selling fee) = $10183.63. Alternatively, we could have simply chosen to not buy any stock on day three. In this scenario, we simply keep the $10226.04 that we had right after selling the Apple stock. (For this scenario, this is the optimal outcome.)

The Problem:

Based on this the daily stock history of two stocks for a fixed trading period, the initial amount of money you have to invest and the transaction fee, you need to calculate the maximum money you can have at the end of the last day of the trading period given, assuming that you cash out at the end of that day.

The Input:

The first line of the input file has a single positive integer, n ($n \leq 50$), representing the number of stock scenarios to evaluate. The first line of each stock scenario has the number of days for that scenario, d ($1 < d \leq 15$), followed by a space, then a positive real number given to two decimal places, T ($T \leq 100.00$), representing the transaction fee for the scenario, followed by a space and a positive real number given to two decimal places, M ($M < 10000000$), representing the amount of money available to invest at the beginning of the stock scenario in dollars. The second line of

each stock scenario will contain *d* positive real numbers given to two decimal places separated by spaces representing the value of stock #1 on days 1 through *d*, respectively. The third line of each stock scenario will contain *d* positive real numbers given to two decimal places separated by spaces representing the value of stock #2 on days 1 through *d*, respectively. All of the costs for each stock per share will be less than 1000.00 (in dollars).

The Output:

For each stock scenario, output a single line with the maximum value in dollars to decimal places that can be obtained by trading under the given restrictions of the problem.

Sample Input:

```
2
3 50.00 10000.00
759.68 765.74 770.17
443.00 457.82 457.04
2 5.99 29999.99
59.99 58.99
22.67 20.73
```

Sample Output:

```
10226.04
29999.99
```

Up-wards

Filename: `upwards`

You love finding random properties in words and recently, you've been fascinated with "Up Words", words that already have their letters in sorted order, with no repeated letters. In order to aid your search for these words, write a program that can automatically discern if a word is an "Up Word" or not.

The Problem:

Given a string with all lower case letters, determine whether or not the letters are distinct and in alphabetical order.

The Input:

The first line of the input will contain a single positive integer, *n (n ≤ 1000)*, representing the number of words you'll evaluate. The following *n* lines will have a non-empty string of lowercase letters of length 26 or less, each.

The Output:

For each string, either output "YES" on a single line or "NO" on a single line, depending on whether or not the corresponding string is an "Up Word" as previously defined.

Sample Input:

```
5
act
cat
cry
cell
tux
```

Sample Output:

```
YES
NO
YES
NO
YES
```