COMPUTER AND INFORMATION SCIENCE

UNIVERSITY OF OREGON

Fourteenth Annual University of Oregon Programming Competition

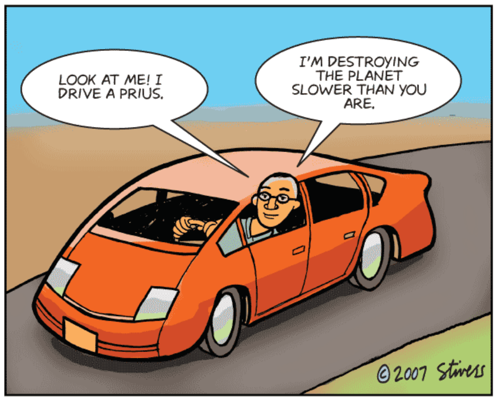
Saturday, April 10, 2010

Problem Contributors

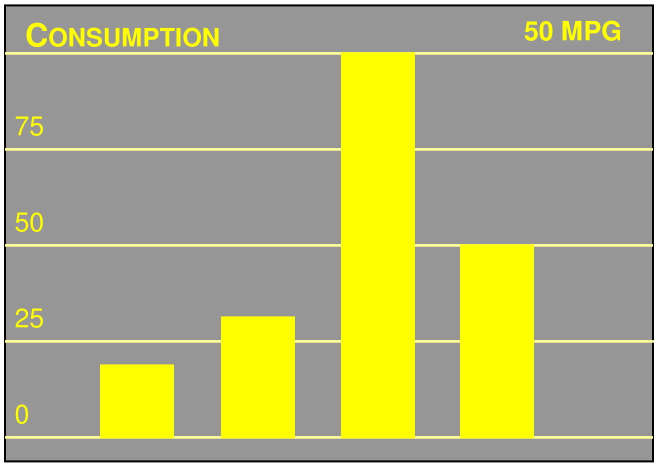
Jim Allen
David Atkins
Gene Luks

Supported in part by Google

RAISING THE BAR


LOOK AT ME! I DRIVE A PRIUS.
I'M DESTROYING THE PLANET SLOWER THAN YOU ARE.
©2007 Stiness

Like many new hybrid cars, a Prius has an energy consumption display that shows the driver the fuel efficiency in real time. The display shows the average miles per gallon for the tank of gas, but also has a bar chart that shows the mpg over five mile intervals. Since driving up hill or accelerating uses more fuel, if one of the intervals is mostly uphill, the mpg would be fairly low for that interval. Likewise, if one of the intervals consisted mostly of going downhill and coasting, the mileage would be very high. The mpg figures in the display are accurate to one decimal place, and limited to two significant figures. Thus, the highest value that can be shown is 99.9 mpg. This applies to the bar chart as well, where the height of each bar is limited to 99.9. One result of this limitation is that the bar chart may have a bar of maximum height (e.g., when coasting down hill), but the actual consumption during that interval could be higher than 99.9. We would like to know what the real fuel consumption is for such an interval.



For this problem, you must write a program that determines the actual miles per gallon consumed during an interval that displays as a 99.9 mpg bar on the bar chart. For example, suppose there are two bars showing: one is 25 mpg, the second is 99.9 mpg. The average for the ten miles traveled is 70 mpg. So, the actual consumption for the second bar is 115 mpg, since it must exceed the average of 70 by the 45 that the first bar falls short in order to average out to 70. The figure to the left is an example display with four bars of heights 20, 30, 99.9, and 50 with an overall mpg of 50. In this case, the actual mileage for the 99.9 bar is 100 mpg.

Input to the program consists of a number of problems to solve. The first line indicates how many problems follow. Each problem consists of one line. The line begins with a decimal number that is the average miles per gallon over all intervals. Next is an integer from 1 to 20 that indicates the number of bars, followed by the mpg for each bar as a positive decimal number. Exactly one of the bar values is 99.9. Your program must output the actual miles per gallon for that bar to one decimal place in the form shown. If the actual mpg value is less than 99.9, your program must output “Impossible display!”.

Sample Input

```
3
70 2 25 99.9
50 4 20 30 99.9 50
50 4 25 99.9 75 50
```

Sample Output

```
Actual mpg on interval 2 is 115.0
Actual mpg on interval 3 is 100.0
Impossible display!
```

CHECKMATE

The N-Queens problem involves counting the number of ways to place n queens on an n by n chessboard in a way that the queens cannot capture each other. The classic problem has O(n!) complexity and is the textbook example of a backtracking algorithm used to show the usefulness of functional programming paradigms. For values of $n < 27$, there are several groups who have compiled tables of all possible solutions. In particular, they are interested in "unique" solutions. That is, solutions that cannot be obtained by simply rotating the chessboard of another solution or holding it to a mirror (or some combination of both). For example, for an 8×8 chessboard, there are 92 different configurations, but only 12 of them are unique in this sense.

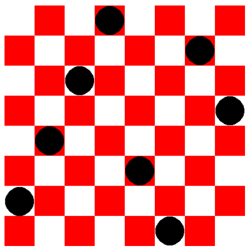
In this problem you must decide whether two solutions are unique. Your program will take as input two different solutions A and B and determines whether there is some combination of rotation and/or reflection of the board that would make A the same as B. In all cases, B is a previously verified solution.

The first input line is the number of problems, and each problem consists of three lines. The first problem line is a single number $0 < n < 26$, and each of the next two lines is a configuration on an $n \times n$ board. The configurations are given as an n-length, space delimited list of numbers. The first number tells which column the queen occupies in the first row; the second number tells which column the queen occupies in the second row; etc. The columns are numbered 1 through n. First, you must test whether the first proposed solution is correct, i.e., that no queen in the solution is on the same row, column or diagonal as any other queen on that board. Your output should be either "`unique`" if the first solution cannot be generated from the second by rotating or reflecting the board, "`not unique`" if it can, or "`illegal configuration`" if the first line is not a solution at all.

Sample Input
```
5
4
2 4 3 1
2 4 1 3
4
2 4 1 3
3 1 4 2
5
2 4 1 3 5
2 4 1 5 3
5
1 3 5 2 4
3 5 2 4 1
5
3 5 2 4 3
3 5 2 4 1
```

Sample Output
```
illegal configuration
not unique
unique
not unique
illegal configuration
```

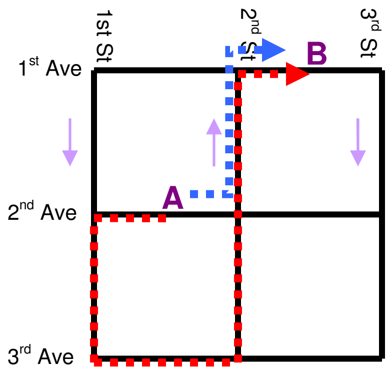
Example of a configuration:



The first row contains a queen at position 4; the second row contains a queen at position 7; the third row contains a queen at position 3, etc. So, the encoding for this solution is `4 7 3 8 2 5 1 6`.
.

NO LEFT TURN

Programmer's Pizza Delivery analyzed its driving patterns and found that left turns at unprotected intersections have a higher rate of accidents as well as slowing down the delivery. When there is no left turn signal arrow or four way stop sign, there is no control for on-coming traffic and the drivers have to wait to turn safely. So, PPD is paying some programmers (in pizza, of course) to route the drivers by the shortest way that avoids left turns at the unprotected intersections.



For example, in the pictured street grid of three North-South streets and three East-West avenues, where odd numbered streets are one way south and even numbered streets are one way north, the shortest route from A to B is two blocks (shown in blue). However, if no left turn is allowed at the intersection of 2nd Avenue and 2nd Street, then the shortest route is five blocks (shown in red). If there was also no left turn allowed at 2nd Avenue and 1st Street, then the shortest route would have to use 3rd Street and take six blocks.

In this problem, you are given a list of map descriptions. For each map, your program must compute the number of blocks in the shortest route that avoids the given left turns. If there is no route possible, your program must print "You can't get there from here!".

The first line of input will be a problem count that specifies how many maps you are to process. Each map will consist of three lines. The first line consists of two numbers (between 1 and 15) specifying the number of avenues and the number of streets. The second line consists of four pairs of numbers. The first pair and second pair are the intersections surrounding the start point, and the third and fourth pairs are the intersections around the ending point. Assume that the starting and ending points are halfway down the block. The third line is a list of the no-left-turn intersections, where the first number specifies how many pairs are on the line. Odd numbered streets are one way south and even numbered streets are one way north. East-west avenues are both directions, but you may not make U-turns.

Sample Input

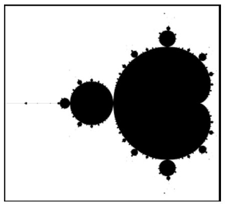
```
4
3 3
2 1 2 2 1 2 1 3
0
3 3
2 1 2 2 1 2 1 3
1 2 2
3 3
2 1 2 2 1 2 1 3
2 2 2 2 1
3 3
3 1 3 2 1 2 2 2
3 2 1 2 2 3 2
```

Sample Output

```
2 blocks
5 blocks
6 blocks
You can't get there from here!
```

A RECURRING PROBLEM

D



The physics students are collaborating with the biology students to study the effects of climate change on species extinction. This involves recurrence modeling techniques, the discrete nature of which confuses and enrages the physics undergrads. They want a simple computer program that takes as input a recurrence relation and outputs a particular element of the sequence (modulo some prime). For example, starting with initial terms $a_0 = a_1 = a_2 = a_3 = 1$, they need to know the ten-millionth term (mod 23) in the sequence:

$$\boxed{a_n = a_{n-1} + 3a_{n-2} + 2a_{n-3} - 5a_{n-4}}$$

To solve the problem, the physics students wrote this FORTRAN program. However, they found that it took too long, even though FORTRAN is known to be faster than Java and Python. They were all out of ideas on how to make their code faster, so they turned to the computer science students to help them out.

```
      INTEGER A(10000000), RESULT
      DO 10, N = 1, 4
10    A(N) = 1
      DO 20, N = 5, 10000000
20    A(N) = A(N-1) + 3*A(N-2) + 2*A(N-3) - 5*A(N-4)
      RESULT = MODULO(A(10000000), 23)
      WRITE(*,1000) RESULT
1000  FORMAT (I2)
```

For this problem, you are given a recurrence relation and must produce a specified term in the sequence (modulo a given prime). The first line of input will contain the number of problems to solve. Each problem consists of three lines of space-separated integers. The first of the three lines contains the integer coefficients of the recurrence relation. The second line consists of the first elements in the sequence, starting with a_0, and the third line contains two numbers, the prime number modulus followed by the index of the term in the sequence to find. (Since the sequence is zero-based, 0 indexes the first term in the sequence.) The first and second lines will always have the same number of entries, and the third line will have exactly two entries. The problem described above would be coded as shown to the right.

```
1 3 2 -5
1 1 1 1
23 10000000
```

The first number on the third line will be a prime number less than 10,000. The second number will be less than 1,576,800,000 (representing 50 years in seconds).

Sample Input

```
4
1 1
1 1
5 1000
1 2 0 -5 3
1 0 0 0 0
7907 7919
5 -2 3 -2 1 -2 3 -2 5
1 2 3 4 5 6 7 8 9
997 1400000000
1 3 2 -5
1 1 1 1
23 10000000
```

Sample Output

```
1
5825
39
1
```

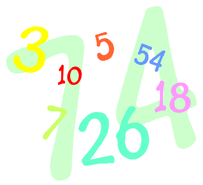
FACTORIAL QUOTIENTS

The most solved problem on last December's Putnam Mathematical Competition was the following

Show that every positive rational number can be written as a quotient of products of factorials of (not necessarily distinct) primes. For example,

$$\frac{10}{9} = \frac{2!\cdot 5!}{3!\cdot 3!\cdot 3!}$$

Invariably, solutions described (not necessarily efficient) algorithms for obtaining such representations.



In this problem, you are to compute a quotient of products of factorials of given rational numbers with numerators and denominators between 1 and 10,000. (To save you some trouble in testing primality, the primes less than 10,000 are in the file `/cs/contest/primes`.)

The first line in the input will give an integer *m*, indicating that *m* problem instances are to follow, one to a line. Each problem instance will consist of two integers *n* and *d*, separated by a single space, indicating the numerator and denominator of a fraction.

The output for each problem instance, should be of the form

{ set of primes **}** **{** set of primes **}**

where the primes in each set are listed in order and separated by commas, and the lists should have no primes in common. The first set should correspond to the numerator in the required representation, the second to the denominator. For example, given the input **`10 9`**, your program should produce the output

`{2,5}{3,3,3}`

since $\frac{2!\cdot 5!}{3!\cdot 3!\cdot 3!} = \frac{2\cdot 120}{6\cdot 6\cdot 6} = \frac{10}{9}$. (Beware that the output for an instance may be an extremely long line.)

There should be blank lines in the output between problem instances.

Sample Input

```
5
10 9
1 8
5 5
8 10
17 1
```

Sample Output

```
{2,5}{3,3,3}

{}{2,2,2}

{}{}

{2,2,2,2,3}{5}

{3,17}{2,2,7,13}
```

NOTE: The product of an empty set (**{ }**) is understood to be 1.