

ND Programming Contest, Fall 2007

ND ACM Computer Club

Sunday, October 7, 2007

#1: Are You In Your Prime?

The Problem

Professor Euclid in the Math Department has a long and productive history of beautiful number-theoretical discoveries of dazzling brilliance. Much of his recent research has dealt with some fundamental properties of prime numbers. The Professor has stumbled upon something truly great: a positive correlation between reduced carbon emissions and increased use of prime-number dimensions in automobile designs. Somehow, research on *composite* materials was all wrong and Euclid was bright enough to see the need to *refactor* his approach. Sadly, the Professor's mind is not what it used to be, and he can no longer test for primality with lightning-fast mental arithmetic. He has tasked you, his kind and obedient (and distinctly computer-adept) student, to help him find primes. Can you help Professor Euclid on his mission, one of **prime** importance?

Your program should read one number n from standard input, indicating the number of inputs to test. n numbers will follow the count, each on its own line. For each of the n inputs, your program should print either "Prime" or "Composite" (capitalization is important) depending on the input's primality. It is guaranteed that each input p will satisfy $1 < p < 10^9$.

Sample Input

```
8 // 8 integers follow
541
37
113
3
4
2
5
10
```

Sample Output

```
Prime
Prime
Prime
Prime
Composite
Prime
Prime
Composite
```

#2: Friend of a friend of a...

The Problem

John Q. Facebooker, a freshman studying Business at the University of Notre Dame, has a problem. Having far too much time on his hands, he has turned to the Internet to relieve his boredom; he has recently discovered Facebook and has not been the same since. However, John has an odd problem: his laptop's keyboard never seems to work right, and so he has to reach Facebook profiles using only mouse-clicks on friend links, starting from his own profile.

And so John has turned to you. All engineers know graph theory, he reasons. (Let's prove him right.) You have a graph of Facebook friend relationships at your disposal; starting at John's profile, your program must determine whether there is a path through Facebook friendships to person p . To simplify matters, all profiles will be identified only by a nonzero integer i ; John's profile has ID 0.

Your program will receive a number n indicating the count of friendship-edges in the graph, followed by n lines of the format " i_1 [space] i_2 "; each of these indicates that i_1 is i_2 's friend and vice-versa. Following the graph data, the program will receive a count m and then m lines, each with a queried profile-ID q . For each query, the program should determine whether John (profile-ID 0) can reach q through an arbitrary number of friend links. If he can, print "Reachable"; otherwise, print "Unreachable".

It is guaranteed that $0 \leq i < 1000$ for all profile-IDs i .

Sample Input

```
4 // four friend relationships follow
0 1 // 0, 1 are friends
1 2 // 1, 2 are friends
2 3 // etc
3 4
2 // two queries follow
4 // 4 reachable from 0?
5 // 5 reachable from 0?
```

Sample Output

```
Reachable
Unreachable
```

#3: Fractional Thoughts

The Problem

Sometimes even the smallest tasks carry the greatest importance in this world. Software to reduce fractions could be such a thing – in fact, your completion of this problem could be a vital element to the survival of the human race through the current plethora of crises. Or perhaps this problem is simply what one might, colloquially, call a “gimme”. Whatever the case may be, your task is as follows:

Your program will receive an integer n followed by n fractions, each on its own line. A fraction line contains two integers, the numerator and the denominator. Compute the reduced form of the fraction, defined as follows: $\frac{p}{q}$ is reduced if and only if $\gcd(p, q) = 1$ and $q > 0$. If the denominator given is 0, the output should be “invalid”; otherwise, output the reduced numerator, denominator, and the decimal form rounded to three decimal places, each separated by one space.

You may assume that $-2^{31} < p, q < 2^{31}$.

Sample Input

```
4 // four fractions follow
2 4
0 0
14 16
1 -2
```

Sample Output

```
1 2 0.500
invalid
7 8 0.875
-1 2 -0.500
```

#4: Follow the Sequence!

The Problem

You are pleasantly minding your own business when your roommate ¹ suddenly asks you: “Say, roommate, I have this sequence 1, 11, 21, 1211, 111221, What say you to finding the next integer?”

After beating your brain for twenty minutes in a feeble attempt to find the pattern, it is revealed to you: given element (say) $i_3 = 21$, the next element is formed from the phrase “one 2, one 1”. In general, i_{n+1} is formed by reading i_n left-to-right as a string of digits and, for each run of n instances of digit d , append n followed by d to the output digit-string. For example, 114421 contains two 1s, two 4s, one 2, and one 1; the next integer in the sequence would be 21241211.

Eager to put your programming prowess to use, you set about formalizing this transformation in a small utility program. The program should take a count n on one line, followed by n integers; for each of the n integers, it should output the integer that comes after n in the sequence.

Sample Input

```
5          // five integers follow
21
114421
555555555
123456
1
```

Sample Output

```
1211
21241211
105
111213141516
11
```

¹Thanks to my roommate Alden Golab for this sequence. To his credit, I asked for ideas; he did not blurt this out unprovoked. Also to his credit, his major has nothing to do with math.

#5: Of Pointy Things

The Problem

Humanity has recently made contact with an alien race: the Pythagoreans, of planet *Tripointy* (which happens to be a regular triangular pyramid). In a spirit of unprecedented open-market trade, the United States wishes to establish an interplanetary economy. The Pythagoreans are quite attached to their currency, and so in an act of goodwill, we have decided to conform to their system.

Unfortunately, the *Tripointy* currency is a bit hard for the average human to interpret. Pythagoreans are fond of pointy corners – the pointier, the better. As such, they value their triangular coins based on the pointiness of the sharpest corner. Terran ambassadors have been armed with very accurate Intergalactic Length/Distance Quantifiers (ie, small metric rulers) to determine the edge-lengths of each *Tripointy* coin that they handle. They need only one thing: a program to run on their handheld computers to quickly compute the value of a coin.

Formally, a *Tripointy* coin is valued as follows (in US dollars), if α is the smallest angle (in radians):

$$v(\alpha) = \frac{5}{\alpha}$$

To determine α , you may wish to make use of the Law of Cosines, which states that given any three edge lengths a , b and c , the angle θ opposite c satisfies:

$$c^2 = a^2 + b^2 - 2ab \cos(\theta)$$

The input to your program will consist of an integer n followed by n lines, each with integer side lengths a , b and c separated by whitespace. For each set of dimensions, your program should output the US-dollar equivalent according to the smallest angle in the triangle, rounded to two decimal points (integral cents).

You may assume that all triangles have non-zero area – ie, that no triangles are degenerate.

Sample Input

```
2          // two triangles follow
3 4 5
3 2 2
```

Sample Output

```
7.77      // actual value is 7.7699...; round to 7.77
6.92
```

#6: You're Either With Us or Against Us...

The Problem

A long time ago in a galaxy far, far away, a planet burst into war. Alliances quickly became important: without support of fellow nations, a nation-state could not hold its own in the hostile environs. However, the situation has an interesting quirk: every nation severely dislikes its direct neighbors, so that no nation can ally with a neighbor. Two alliances exist: the “Zeroes” and the “Ones” (the people of this planet are not terribly creative). Every nation must join one of the two alliances and must not be in alliance with any of its direct neighbors.

Your task is this: given a graph of neighbor relationships, determine whether such an arrangement is possible. For simplicity, countries are identified by number, 0 to k . The input to your program will consist of a count i followed by i test cases. Each test case will contain the country-count k , followed by another integer n , followed by n lines, each of which represents one edge in the neighbor-graph. Each neighbor line is of the format “ x y ”, indicating that x and y are direct neighbors and so cannot be allies. For each test case, the program should print “possible” or “not possible” as appropriate to indicate whether every country can join an alliance.

It is guaranteed that $k < 1000$ for every test case.

Sample Input

```
2 // two test cases
4 // first test case: four countries
3 // first test case: three neighbor-relationships
0 1 // 0, 1 are neighbors
1 2 // etc
2 3 //
3 // second test case: three countries
3 // second test case: three neighbor-relationships
0 1 // 0, 1 are neighbors
1 2 // etc
2 0
```

Sample Output

```
possible
not possible
```

#7: Sue, do Pseudo-Sudoku

The Problem

Sue is a TA for an undergraduate intro-mathematics class. In an effort to demonstrate that mathematical reasoning can be fun (and that it does not necessarily involve numbers), Sue has invented the game “Pseudo-Sudoku”². This game is identical to ordinary *Sudoku* with one exception: where *Sudoku* uses the digits 1 through 9, “Pseudo-Sudoku” uses *A, B...I*. She has assigned “Pseudo-Sudoku” puzzles to her tutorial section with extra practice, and wishes to have a program to validate solutions.

Your program should read an integer n followed by n “Pseudo-Sudoku” boards; for each board, it should check that the solution is valid. If it is, it should output “`valid`”; otherwise, it should output “`invalid`”. A “Pseudo-Sudoku” board is input as nine lines of nine letters each, separated by whitespace.

A board is valid if and only if, for every row, column, and each of nine 3×3 sub-boards, every letter appears exactly once.

Sample Input

```
2          // two boards follow
A B C D E F G H I      // board 1
D E F G H I A B C
G H I A B C D E F
B C D E F G H I A
E F G H I A B C D
H I A B C D E F G
C D E F G H I A B
F G H I A B C D E
I A B C D E F G H
A A A A A A A A A      // board 2
A A A A A A A A A
A A A A A A A A A
A A A A A A A A A
A A A A A A A A A
A A A A A A A A A
A A A A A A A A A
A A A A A A A A A
A A A A A A A A A
A A A A A A A A A
```

Sample Output

```
valid
invalid
```

²with apologies for the pun.