Stanford Local Programming Contest

Oct 4, 2003

12pm – 4pm

5 Problems

Problem 1: Sloppy Fractions

Fractions are real numbers that can be written as a/b, where a and b are both integers, and $b \neq 0$. We will restrict ourselves to fractions that lie between 0 and 1, both inclusive. This means that $0 \leq a \leq b$ when both a and b are non-negative. There are two ways of writing fractions: In the CANONICAL form, we write a fraction as a/b where a and b are positive integers having no factor in common (except 1, of course). In the DECIMAL form, brackets are placed around the smallest-sized left-most group of digits that repeats. Here are some examples:

Fraction	CANONICAL	DECIMAL	Full expansion
5/12	5/12	0.41(6)	0.41666666...
3/21	1/7	0.(142857)	0.142857142857142857...
1/5	1/5	0.2(0)	0.2000000...
7/8	7/8	0.875(0)	0.875000000...
3/9	1/3	0.(3)	0.3333333...

By convention, the CANONICAL form for fraction $0.0000\ldots$ will be 0/1.

Our protagonist is mathematician Chen, who prefers writing fractions in the DECIMAL form. Chen is quite old — he celebrated his 100^{th} birthday a fortnight ago. He's become sloppy over the years. Since his 90^{th} birthday, he hasn't necessarily been placing the brackets around the smallest-sized left-most group of digits that repeats. For example, he might write 0.52(01) as 0.5201(01) or even as 0.52010101(0101). Further, since his 100^{th} birthday, he's been forgetting to place the brackets around the repeating-groups! This means that 0.520101 might denote any of the following: 0.52010(1), 0.5201(01), 0.520(101), 0.52(0101), 0.5(20101), 0.(520101) and 0.520101. Your goal is to write a program which will take a string written down by Chen as input, and produce all possible fractions in CANONICAL form that string could correspond to.

Input: One string per line, of the form $0.a_1a_2\ldots a_\ell$, where $1 \leq \ell \leq 6$ and each of $a_1, a_2 \ldots a_\ell$ is a digit between `0` and `9`.

Output: See the format below. Each fraction should be printed in the CANONICAL form. There should be no duplicates. Fractions should be sorted by magnitude, smallest to largest. There should be exactly one blank separating the fractions, and one blank before and after the `=` symbol, as shown below.

Input	Output
`0.000`	`0.000 = 0/1`
`0.3`	`0.3 = 3/10 1/3`
`0.32`	`0.32 = 8/25 29/90 32/99`
`0.99`	`0.99 = 99/100 1/1`
`0.511`	`0.511 = 511/1000 23/45 511/999`
`0.5151`	`0.5151 = 5151/10000 1159/2250 2573/4995 17/33`

Problem 2: The Game of HEARTS

A card (belonging to the usual deck of 52 cards) will be represented by two characters. The first character will denote the face value and will belong to the set { `2, 3, ..., 9, T, J, Q, K, A` }. The second character will be `C`, `D`, `H` or `S`, denoting Clubs, Diamonds, Hearts or Spades respectively.

HEARTS is a popular four-player card-game. Each player gets 13 cards to begin with. The game proceeds in 13 rounds with each player playing one card per round. Each round has a *leader* who plays the first card in that round. The leader for the first round is that player who possesses the Two of Clubs (`2C`). In the first round, the leader must play `2C`. In other rounds, the leader can play any card (s)he holds.

The leader is followed by the other three players, going in the clockwise direction. Every player must match the suit of the leader's card if (s)he holds a card of that suit. In case a player cannot match the suit of the leader's card, (s)he can play any card (s)he currently holds.

Each round has a *winner* who goes on to become the leader for the next round. The winner of a round is decided as follows: From among the players whose card matches the suit of the leader's card, the player whose card has the highest face value, wins. The face value of a card is governed by the ordering `2 < 3 < ··· < 9 < T < J < Q < K < A`.

Input will consist of a series of HEARTS games. There will be 4 lines per game. The four lines correspond to four players in the clockwise directions. Each line represents the sequence of cards played by that player. It will contain descriptions of thirteen cards, separated by single blanks.

Output will consist of one line per game, stating `Valid` or `Invalid`, depending upon whether the sequence of cards corresponds to a valid HEARTS game or not.

You may assume that each line has exactly 13 card descriptions. Card descriptions are separated by a single blank space (see example input below). However, the cards in a line or in a game description may not be all distinct. Even if they are all distinct, the sequence may not correspond to a valid game of HEARTS.

Input	Output
`2C QD TH TS 9S 8S 4S 3C AC 9C JC TC QC`	`Invalid`
`8C 8D 6D AS 2D QH KH JH 7H 4H 2H 3H 5H`	`Valid`
`KC AD 3D JS 7S 6S 3S 6C 5C 4C 2S QS KS`	
`7C KD 4D 5S AH 9H 8H 6H JD 5D 9D 7D TD`	
`8C 9S KD 9D 6D KH 2H JH 7H 4D 9H 5C 7C`	
`JC AS 8S TD QD 5S QH 4S 6H TH 8H 3S 2S`	
`2C QS 7S 5D 2D KS 5H TS 4H 8D 3H 4C KC`	
`AC JS 6S AD JD AH QC TC 9C 7D 3D 3C 6C`	

Problem 3: Routing in CHORD

CHORD is an overlay network topology for routing messages in a peer-to-peer system. Consider a set of 2^k machines, arranged in a circle, where $1 \leq k \leq 30$. Machines are labeled 1 through 2^k, going clockwise along the circle.

The clockwise distance from node x to node y is given by the expression $(y - x + 2^k) \bmod 2^k$.

A machine x makes $k - 1$ TCP connections *with* other nodes, which are clockwise distance 1, 2, 4, 8, 16, ... away. Naturally, machines that are anti-clockwise distance 1, 2, 4, 8, ... away from x make connections *with* machine x.

For example, if $k = 7$, then there are 128 machines labeled 1 thru 128 and machine 5 makes TCP connections with machines 6, 7, 9, 13, 21, 37 and 69. Moreover, machines 69, 101, 117, 125, 1, 3 and 4 make TCP connections *with* machine 5.

If a machine wishes to send a message to another machine, the message is *routed* along the TCP connections that exist in CHORD. We will study three different routing strategies:

A) CLOCKWISE GREEDY: A node forwards a message along that TCP connection *that it established with some other node* that diminishes the clockwise distance to the destination by the maximum amount.

B) ANTI-CLOCKWISE GREEDY: A node forwards a message along that TCP connection *that other nodes established with it* that diminishes the anti-clockwise distance to the destination by the maximum amount.

C) ABSOLUTE GREEDY: A node forwards a message along that TCP connection (*irrespective of whether it established it with others, or others established it with the node*) that diminishes the absolute distance (i.e., the smaller of clockwise and anti-clockwise distances) to the destination by the maximum amount. In case there is a tie between two TCP connections, either one of them is chosen arbitrarily.

Your goal is to find out the *length* of a route from node x to node y, as per the three strategies. The length of a route is the total number of times a message is forwarded along TCP connections to reach the destination.

Input will consist of three integers per line. The first integer denotes k such that $1 \leq k \leq 30$. The next two integers denote x and y satisfying $1 \leq x, y \leq 2^k$.

Output should consist of three non-negative integers per line, separated by blank spaces. The first integer is the length of CLOCKWISE GREEDY route. The second integer is the length of the ANTI-CLOCKWISE GREEDY route. The third integer is the length of the ABSOLUTE GREEDY route.

Input	Output
1 1 1	0 0 0
5 1 32	5 1 1
5 1 16	4 2 2
10 32 128	2 4 2
20 1234 4321	6 15 4

Problem 4: Ternary Weights

Yelda is a smart Turkish shopkeeper who uses an ordinary beam balance for measurements. She owns a set of cubes that weigh 1g, 3g, 9g, 27g, 81g, 243g, ..., respectively. Yelda owns exactly one cube of a given weight, and she is allowed to place cubes in either pan.

A sack of unknown weight has been placed in the Left pan of the beam balance. Your program has to help Yelda figure out which cubes to place in which pan so that the two pans are balanced. This will help her figure out the weight of the sack.

Input will consist of a series of positive integers, one integer per line. Integers lie between 1 and $2^{31} - 1$, both inclusive. Each integer represents the weight of the sack placed in the Left pan.

Output should be in the format shown below. The string after the `=` symbol consists of `L`, `R` or `N`, depending on whether the corresponding cube was placed on the Left pan, the Right pan or not used at all. Sequences of trailing `N` symbols must be trimmed. This means that the last symbol is necessarily `R`.

For example, output `368 = LNLLLLR` means that to measure 368g, cubes weighing 1g, 9g, 27g, 81g and 243g have to be placed in the Left pan, and the cube weighing 729g has to be placed in the Right pan. Other cubes are not used at all.

Input	Output
1	1 = R
2	2 = LR
30	30 = NRNR
4	4 = RR
499	499 = RRRNNLR
368	368 = LNLLLLR

Problem 5: Rugs Galore

A number of rectangular rugs are placed on a large rectangular floor. Each side of a rug is parallel to some side of the floor. You have to compute the total area of the floor that is **not covered** by any rug. The catch lies in handling with care, portions of the floor that are covered by more than one rug.

Input will be a sequence of floor+rug descriptions:

- The first line of the description will be a pair of integers, W and H, representing the width and height of the floor. The coordinates of the lower-left, lower-right, upper-left and upper-right corners of the floor are $(0, 0)$, $(W, 0)$, $(0, H)$ and (W, H) respectively. It is guaranteed that $1 \leq W \leq 10^4$ and $1 \leq H \leq 10^4$.

- The floor description will be followed by quadruples of integers, one quadruple per line. Each quadruple represents a rug. The first pair of integers in a quadruple, say w and h, will denote the width and height of the rug. The second pair, say x and y, will denote the position of the lower-left corner of the rug. It is guaranteed that $w \geq 1$, $h \geq 1$, $x \geq 0$ and $y \geq 0$. Moreover, the input will satisfy $x + w \leq W$ and $y + h \leq H$.

- A quadruple of all 0's will terminate a floor+rug description. The number of rugs will not exceed 100.

Output should consist of as many lines as floor+rug descriptions. Each line should be of the form:

`Floor has` A `area not covered`

where A denotes the area of the floor not covered by any rug.

Input	Output
100 100 10 10 0 0 0 0 0 0 100 100 10 10 10 10 10 100 10 0 0 0 0 0 100 100 1 1 0 0 2 2 0 0 3 3 0 0 0 0 0 0	Floor has 9900 area not covered Floor has 9000 area not covered Floor has 9991 area not covered