Waterloo ACM Programming Contest

September 17, 2005

Problem A: So you want to be a 2^n-aire?

Problem B: Ferry Loading III

Problem C: Pick-up Sticks

Problem D: Rock-Paper-Scissors Tournament

Problem E: Structural Equivalence

Problem A: So you want to be a 2^n-aire?

The player starts with a prize of $1, and is asked a sequence of *n* questions. For each question, he may

- quit and keep his prize.
- answer the question. If wrong, he quits with nothing. If correct, the prize is doubled, and he continues with the next question.

After the last question, he quits with his prize. The player wants to maximize his expected prize.

Once each question is asked, the player is able to assess the probability *p* that he will be able to answer it. For each question, we assume that *p* is a random variable uniformly distributed over the range *t .. 1*.

Input is a number of lines, each with two numbers: an integer $1 \le n \le 30$, and a real $0 \le t \le 1$. Input is terminated by a line containing `0 0`. This line should not be processed.

For each input *n* and *t*, print the player's expected prize, if he plays the best strategy. Output should be rounded to three fractional digits.

Sample input

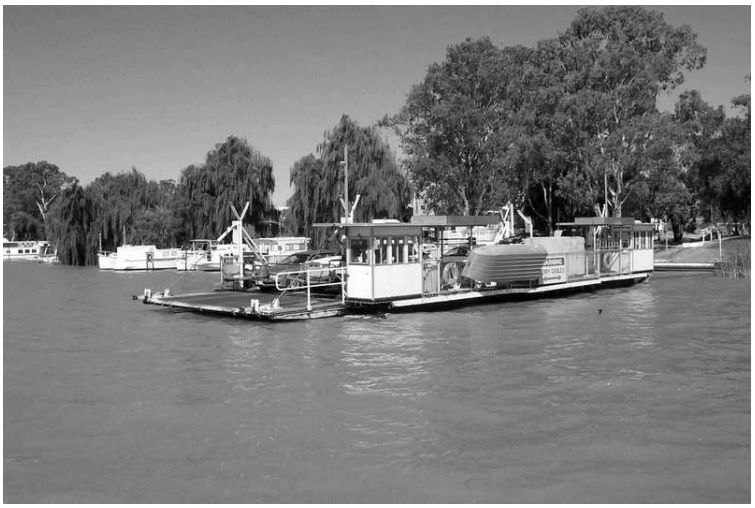
```
1 0.5
1 0.3
2 0.6
24 0.25
0 0
```

Output for sample input

```
1.500
1.357
2.560
230.138
```

Don Reble

Problem B: Ferry Loading III



Before bridges were common, ferries were used to transport cars across rivers. River ferries, unlike their larger cousins, run on a guide line and are powered by the river's current. Cars drive onto the ferry from one end, the ferry crosses the river, and the cars exit from the other end of the ferry.

There is a ferry across the river that can take *n* cars across the river in *t* minutes and return in *t* minutes. A car may arrive at either river bank to be transported by the ferry to the opposite bank. The ferry travels continuously back and forth between the banks so long it is carrying a car or there is at least one car waiting at either bank. Whenever the ferry arrives at one of the banks, it unloads its cargo and loads up to *n* cars that are waiting to cross. If there are more than *n*, those that have been waiting the longest are loaded. If there are no cars waiting on either bank, the ferry waits until one arrives, loads it (if it arrives on the same bank of the ferry), and crosses the river. At what time does each car reach the other side of the river?

The first line of input contains *c*, the number of test cases. Each test case begins with *n*, *t*, *m*. *m* lines follow, each giving the arrival time for a car (in minutes since the beginning of the day), and the bank at which the car arrives ("left" or "right"). For each test case, output one line per car, in the same order as the input, giving the time at which that car is unloaded at the opposite bank. Output an empty line between cases.

You may assume that $0 < n, t, m \leq 10000$. The arrival times for each test case are strictly increasing. The ferry is initially on the left bank. Loading and unloading time may be considered to be 0.

Sample input

```
2
2 10 10
0 left
10 left
20 left
30 left
40 left
50 left
60 left
70 left
80 left
90 left
2 10 3
10 right
25 left
40 left
```

Output for sample input

```
10
30
30
50
50
70
70
90
90
110

30
40
60
```

Gordon V. Cormack

Problem C: Pick-up sticks

Stan has *n* sticks of various length. He throws them one at a time on the floor in a random way. After finishing throwing, Stan tries to find the top sticks, that is these sticks such that there is no stick on top of them. Stan has noticed that the last thrown stick is always on top but he wants to know all the sticks that are on top. Stan sticks are very, very thin such that their thickness can be neglected.

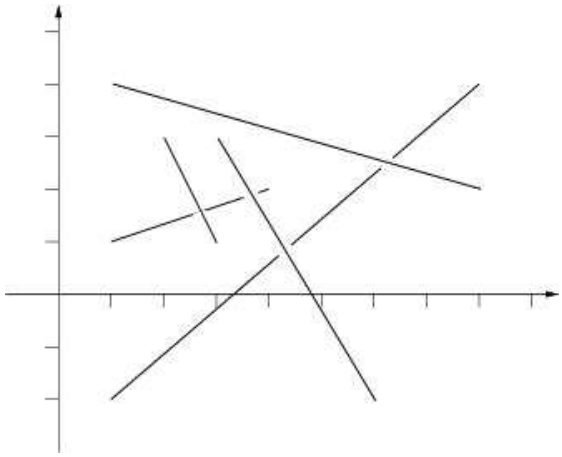
Input consists of a number of cases. The data for each case start with $1 \leq n \leq 100000$, the number of sticks for this case. The following *n* lines contain four numbers each, these numbers are the planar coordinates of the endpoints of one stick. The sticks are listed in the order in which Stan has thrown them. You may assume that there are no more than 1000 top sticks. The input is ended by the case with $n=0$. This case should not be processed.

For each input case, print one line of output listing the top sticks in the format given in the sample. The top sticks should be listed in order in which they were thrown.

The picture to the right below illustrates the first case from input.

Sample input

```
5
1 1 4 2
2 3 3 1
1 -2.0 8 4
1 4 8 2
3 3 6 -2.0
3
0 0 1 1
1 0 2 1
2 0 3 1
0
```



Output for sample input

```
Top sticks: 2, 4, 5.
Top sticks: 1, 2, 3.
```

Piotr Rudnicki

Problem D:

Rock-Paper-Scissors Tournament

Rock-Paper-Scissors is game for two players, A and B, who each choose, independently of the other, one of *rock*, *paper*, or *scissors*. A player chosing *paper* wins over a player chosing *rock*; a player chosing *scissors* wins over a player chosing *paper*; a player chosing *rock* wins over a player chosing *scissors*. A player chosing the same thing as the other player neither wins nor loses.

A tournament has been organized in which each of *n* players plays *k* rock-scissors-paper games with each of the other players - *k*n*(n-1)/2* games in total. Your job is to compute the *win average* for each player, defined as *w / (w + l)* where *w* is the number of games won, and *l* is the number of games lost, by the player.

Input consists of several test cases. The first line of input for each case contains $1 \le n \le 100$ $1 \le k \le 100$ as defined above. For each game, a line follows containing p_1, m_1, p_2, m_2. $1 \le p_1 \le n$ and $1 \le p_2 \le n$ are distinct integers identifying two players; m_1 and m_2 are their respective moves ("rock", "scissors", or "paper"). A line containing 0 follows the last test case.

Output one line each for player 1, player 2, and so on, through player *n*, giving the player's win average rounded to three decimal places. If the win average is undefined, output "-". Output an empty line between cases.

Sample Input

```
2 4
1 rock 2 paper
1 scissors 2 paper
1 rock 2 rock
2 rock 1 scissors
2 1
1 rock 2 paper
0
```

Output for Sample Input

```
0.333
0.667

0.000
1.000
```

Gordon V. Cormack

Problem E: Structural Equivalence

In programming language design circles, there has been much debate about the merits of "structural equivalence" vs. "name equivalence" for type matching. Pascal purports to have "name equivalence", but it doesn't; C purports to have structural equivalence, but it doesn't. Algol 68, the *Latin* of programming languages, has pure structural equivalence.

A simplified syntax for an Algol 68 type definition is as follows:

```
type_def -> type T = type_expr
type_expr -> T | int | real | char | struct ( field_defs )
field_defs -> T | field_defs T
```

In this syntax, *T* is a programmer-defined type name (in this problem, for simplicity, a single upper case letter). Plain text symbols appear literally in the input, and zero or more spaces may appear where there are spaces in the syntax.

Algol 68 type equivalence say that two types are equivalent if they are the same primitive type or they are both structures containing equivalent types in the same order.

Input consists of several test cases. Each test case is a sequence of Algol 68 definitions, as described above, one per line. A line containing "-" separates test cases. A line containing "--" follows the last test case. The output for each case will consist of several lines; each line should contain a list of type names, all of which represent equivalent types. Each type name should appear on exactly one line of output, and the number of output lines should be minimized. The names in each list should be in alphabetical order; the lines of output should also be in alphabetical order. Output an empty line between test cases.

Sample Input

```
type A = int
type B = A
type C = int
type X = struct(A B)
type Y = struct(B A)
type Z = struct(A Z)
type S = struct(A S)
type W = struct(B R)
type R = struct(C W)
--
```

Output for Sample Input

```
A B C
R S W Z
X Y
```

Gordon V. Cormack