ECOO 2013

Programming Contest Questions

Local Competition (Round 1)

March 20-27, 2013

Problem 1: Take a Number

Due to overwhelming demand, the principal has installed one of those “take a number” dispensers to help the attendance secretary manage the line for late slips. The dispenser is filled with slips of paper numbered in order from 1 to 999. The principal has made sure to order lots of refills! The attendance desk opens at 8:00 am every morning and closes at 3:00 pm. When a late student arrives they take the next number from the machine, and when the attendance secretary is ready, he calls the next number in order. When a student takes the last number, the secretary immediately refills the machine with a new set of numbers from 1 to 999. At 3:00 pm, he removes the dispenser and stores it for the next day, then serves any students who are still waiting with numbers in their hands before closing for the day.

DATA11.txt (DATA12.txt for the second try) will contain detailed data for a number of days in the late slip lineup. The first line of the file contains an integer N (0 < N < 1000) representing the next number in the take a number machine. This will be followed by some number of lines (up to 1 000 000) representing the activity at the attendance desk. If a line contains the word “TAKE”, it means a student has arrived and taken the next number (when a student takes the last number available, the machine is immediately refilled.) If a line contains the word “SERVE” it means that the attendance secretary has served the next student in line (this word will only appear in the file when there is at least one student waiting). If a line contains the word “CLOSE” it means that the desk has closed for the day and the attendance secretary will serve the students remaining in line and then go home. The very last line of the file will contain the string “EOF”. At no time will there be more than 999 students waiting in line to be served.

Your job is to keep track of the line. Each time you encounter the word “CLOSE”, you must print three integers on a single line, each separated by a single space. The first integer represents the number of students who were late that day, the second integer represents the number of students who remained in line after the desk was closed, and the third integer represents the next number in the take a number machine for the next day.

Sample Input

```
23
TAKE
TAKE
SERVE
TAKE
SERVE
SERVE
CLOSE
TAKE
```

```
TAKE
TAKE
SERVE
CLOSE
TAKE
SERVE
TAKE
SERVE
TAKE
```

```
TAKE
TAKE
TAKE
TAKE
TAKE
SERVE
CLOSE
EOF
```

Sample Output

```
3 0 26
3 2 29
8 5 37
```

Problem 2: The Luhn Algorithm

In the 1950's, Hans Peter Luhn invented a method for checking the validity of ID numbers. This method (known as the Luhn Algorithm or the Luhn Formula) is still used today for a number of different purposes, including all major credit card numbers and Social Insurance Numbers.

Here's how the Luhn Algorithm works when checking for a valid ID number:

1. Starting from the right, double every second digit, add up the digits of the result, and total up all the resulting numbers.
2. Add to this total the sum of all the remaining digits.
3. If the result is divisible by 10, the id number is valid.

Example 1: Validate 42395

Step 1

9 * 2 = 18, 1 + 8 = 9.

2 * 2 = 4.

4 + 9 = 13

Step 2

13 + 4 + 3 + 5= 25

Step 3

25 is not divisible by 10.

Not valid.

Example 2: Validate 35436

Step 1

3 * 2 = 6.

5 * 2 = 10. 1 + 0 = 1.

1+6 = 7

Step 2

7 + 3 + 4 + 6 = 20.

Step 3

20 is divisible by 10.

Valid.

The last digit of every number is the "check digit" and the rest of it is the base number. So in the first example above, 4239 is the base number and 5 is the check digit. When generating card numbers or other ID numbers, you first generate the base number without the final digit, then you figure out what the check digit has to be to make the whole ID number valid.

DATA21.txt (DATA22.txt for the second try) will contain 5 test cases. Each test case consists of a batch of 5 base numbers (1 to 100 digits each) on one line, each separated by a single space character. Your job is to compute the check digit for each base number in the batch and then output the result as a single 5-digit number.

Sample Input

```
389796 4565280784 8451692334 46 465949539
97699 7392253 54011409 8073542288 303142477
334 349839 12593962 02497993 9468
53173 2901524 2493367526 39094 83530
08080532 5023002 57849 9853641952 027179
```

Sample Output

```
48336
36757
31920
15686
88201
```

Problem 3: Hexudoku

Hexudoku is a game of logic in which the goal is to fill in a grid with hexadecimal digits from 0 to F. The grid has 16 rows, 16 columns, and 16 4x4 quadrants. The game starts with a partially filled in board, like the one shown below. The goal is to fill up the rest of the board with hexadecimal digits from 0 to F so that no row, column, or quadrant contains a repeated digit.

D15-	--0-	-8--	---3
----	7-C-	-4EB	----
6--B	-E-2	--9-	----
--C-	--48	7--0	----
A--8	5--6	4---	B--0
-3D-	---E	----	-7--
----	----	---4	1---
-7--	--D-	----	----
C---	----	B---	----
---E	3---	----	-4--
---9	----	--A-	--0-
-5--	---1	----	----
----	---C	----	9---
B---	----	0--A	3---
----	4---	----	----
-D--	8---	-C--	5---

For reference, the hexadecimal digits (in order from least to greatest) are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.

Lazy Larry almost never completes a Hexudoku game, but his simple strategy does make quite a bit of progress. He starts in the top left corner and scans to the right until he finds a blank cell. Then he looks for the smallest hexadecimal digit that can go there without creating a conflict in its row, column, or quadrant. If he finds one, he fills it in. Then he moves to the right until he finds another blank cell and fills it in the same way. He continues until the first row is finished, then moves to the first cell in the next row down and continues until he reaches the bottom right corner. Then he gives up.

Applying this strategy to the board above, the first cell Larry can fill in is the 4th from the left in the top row. Digits 0 and 1 both create conflicts but 2 is safe. In the next cell over 0 through 5 are no good, but 6 works. Eventually, he ends up with the board on the right.

D152	690A	C8F-	4B73
083A	71C5	24EB	69DF
647B	DE32	159-	08AC
9ECF	B-48	7360	215-
A218	5376	49CD	BEF0
43D0	128E	56BF	A79-
569C	0ABF	3274	1D8-
E7B-	94D-	801-	C235
C021	A564	B738	DFE9
7A6E	3029	D15C	84B-
3B49	C7ED	6FA2	-501
85FD	-B-1	9E0-	7326
1F03	265C	EB47	9A-8
B984	ED17	0-2A	36C-
2CA5	4F90	-D81	E--7
-DE6	8-A3	FC-9	5012

DATA21.txt (DATA22.txt for the second try) contains 10 Hexudoku boards. Each board consists of 16 lines of 16 characters each. A blank cell is represented with a '-' character (ASCII code 45). Your job is to report the progress Lazy Larry will make by following the simple strategy described above. The output should be one line for each test case reporting the number of blanks Larry manages to fill in.

The sample input on the next page contains only 3 test cases, and is laid out in 3 columns for easier reading.

Sample Input

```
--A---------5---
-CF--B-9--------
----------5----0
--------------9-
---7---3-----BF-
-B------0-7-----
-F-----C-----D--
---------4-D----
-----------4----
--7-------------
---E---------5--
----D-2-8---A---
-------------F--
---------A-9----
------------6---
------70--8-----
```

```
D15---0--8-----3
----7-C--4EB----
6--B-E-2--9-----
--C---487--0----
A--85--64---B--0
-3D----E-----7--
-----------41---
-7----D---------
C-------B-------
---E3--------4--
---9------A---0-
-5-----1--------
-------C----9---
B-------0--A3---
----4-----------
-D--8----C--5---
```

```
----B-----C1F--8
---C--4-F-5-D---
--E----1----4--9
-0--172--D------
----4---3----A--
--7----------8--
-------C53------
-D--0-----------
---A62---4---3--
--0F-9-----B---D
----7--8A--6----
E----8------3-9-
4------EB-F-----
7-2-81------A---
--9---8--13-E---
--------1---C---
```

Sample Output

```
189
176
164
```

Problem 4: Coupon Day

It's coupon day at Panther Redirect. Customers have been collecting coupons all year, and today is the day they get to use them. During this special sale day, there's a limit of 10 purchases per customer and they are allowed to bring up to 10 coupons with them to the counter. Each coupon can be applied to a maximum of 1 item, and each item can have a maximum of 1 coupon applied to it. The cashier will scan the price codes and the coupons and will help the customers decide how to use their coupons to maximize their savings.

There are 7 coupon types available. The $5, $10, and $50 coupons entitle the customers to a flat discount (if the item is worth less than the coupon, they get it for free). The 10% and 20% coupons entitle the customer to a percentage discount before tax. The TAX coupon entitles the customer to have the item without paying any HST. Finally, the BOGO coupon (maximum of 1 per customer) allows the user to buy one item at full price and get a second item of equal or lesser price for free. Note that neither of the items involved in the BOGO can have other coupons applied to them. The 13% HST is calculated separately on the unrounded price of each item after the coupon is applied. The after tax price for each item is rounded to the nearest cent after tax has been applied. These final prices are added together to get the total purchase price.

DATA41.txt (DATA42.txt for the second try) will contain 5 test cases. The first line of each test case contains an integer N indicating the number of purchase items ($1 \le N \le 10$). This is followed by the N prices P_i in dollars and cents, each on a separate line ($0.0 < P_i \le 100.0$, $1 \le i \le N$). The next line contains an integer M, indicating the number of coupons ($1 \le M \le 10$). This is followed by the M coupon names, each on a separate line.

Write a program that finds the best way to apply the coupons for each customer (the best way being the way that yields the lowest total price according to the rules and restrictions applied above) and then states the final price exactly as shown in the sample output below, always showing two decimal places. The program must terminate within the time limit set out in the general contest rules.

Sample Input

```
3               BOGO            93.43           0.11            BOGO            2.51
74.54           20%             13.69           10              BOGO            5
19.8            TAX             17.02           $5              TAX             20%
69.99           20%             1.94            $10             BOGO            20%
10              $5              6.52            $10             4               $50
BOGO            $5              65.55           TAX             88.17           TAX
20%             10%             8.36            $5              43.18           $50
$50             9               83.2            20%             67.14
```

Sample Output

```
The best price is $84.23
The best price is $184.51
The best price is $101.35
```

ECOO 2013

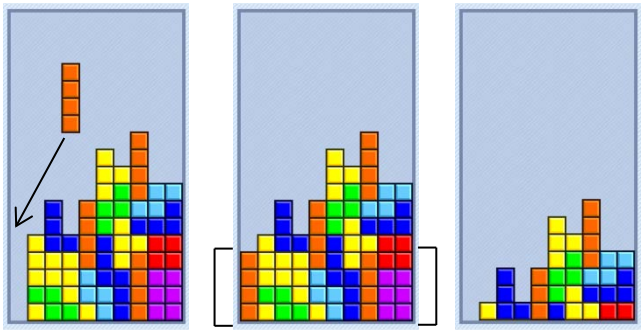
Programming Contest Questions

Regional Competition (Round 2)

April 27, 2013

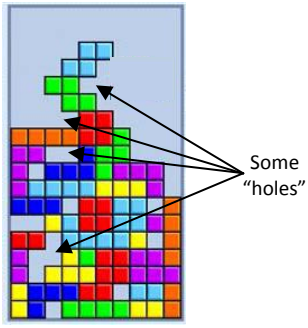
Problem 1: Upsetris

The game of Upsetris is played just like the classic game of Tetris, but with a twist. Just like in Tetris, puzzle pieces, each made of four square bricks, fall from above and the user has to decide where they will land to try and make full rows of blocks. The game ends when the pile reaches the top of the screen.



1 2 3

In the game shown at left, a straight vertical piece is falling. If the user drops it in the left most column (picture 1), she will make the bottom four rows full (picture 2), and the full rows will disappear allowing the pile to move down (picture 3).



If the player makes bad decisions about where to put pieces, they can end up with boards with "holes" in them as shown at right. These holes will eventually cost them the game.

In Upsetris, the "twist" (pun intended) is that the user has a one-time-only chance to "upset" the board by rotating it 180 degrees. When this happens, all the individual bricks separate and fall to the floor. This eliminates the holes and sometimes makes new rows.

Here's an example of the "upset" process in action:

```
|          |    |==========|    |          |    |          |    |          |
|          |    |0 00000 00|    |0 00000 00|    |          |    |          |
|          |    |00000     |    |00000     |    |          |    |          |
|          |    | 00000 00 |    | 00000 00 |    |          |    |          |
| 0   000 0|    |0 00 0 0  |    |0 00 0 0  |    |  00      |    |          |
|0 0  0000 |    | 0000  0 0|    | 0000  0 0|    |  000     |    |  00      |
|  0 0 00 0|    |0 000   0 |    |0 000   0 |    |0 000     |    |  000     |
| 00 00000 |    |          |    |          |    |000000 00 |    |0 000     |
|      00000|    |          |    |          |    |000000 000|    |000000 00 |
|00 00000 0|    |          |    |          |    |**0000000000**|    |000000 000|
|==========|    |          |    |==========|    |==========|    |==========|
      1              2              3              4              5
```

In the example above, the bricks are represented by capital O's and the floor is a row of equal signs. The original board (1) is rotated 180 degrees (2), then the floor of the board is moved to the bottom(3), then the pieces all settle to the new floor (4) making a single complete row which gets removed (5).

DATA11.txt (DATA12.txt for the second try) contains five different Upsetris boards, each with from 5 to 20 rows (not including the “floor”) and from 5 to 20 columns (not including the walls). You must simulate the “upset” operation and show the final result for each board. You must show the complete board in your output, including the “floor” row. You can display the boards under input control if you like (e.g. “hit a key for the next board”) but during judging, you will be allowed to scroll the output screen horizontally or vertically if necessary. The output must be in a fixed-width font (e.g. Courier). If you are unable to configure your IDE to use a fixed width font, you may copy and paste the output of your program into a text editor that uses a fixed-width font during judging. Note that all the boards in the sample input are the same size and have the same number of rows as columns, but this might not be true of the data sets you will be judged on. The side characters for the board are ASCII code 124.

Sample Input

```
|            |
|            |
|0           |
|0  00   0  0|
|      0 0   |
| 0    0  00 |
|   0       00|
|0  0        |
|    0   0  0|
|0         0 |
|==========|
|            |
|            |
|            |
|      0     |
|    0   00  0|
|    0       0|
|     0   0  |
|         00 |
```

```
|0           |
|    0       0|
|==========|
|            |
|            |
|            |
|            |
| 0    000  0|
|0 0   0000  |
|   0  0  00  0|
| 00  00000  |
|      00000|
|00  00000  0|
|==========|
|            |
|            |
|            |
```

```
|            |
|            |
|            |
| 0       0  |
|       0    0|
|==========|
|            |
|            |
|            |
|            |
|00  00  0000|
| 0  0000000|
|0  0000  000|
| 00000000   |
|0000  0  0  |
|0   000  000|
|==========|
```

Sample Output

```
|            |
|            |
|            |
|            |
|            |
|            |
|            |
| 0          0|
| 0  0     0  0|
| 000   00  0|
|==========|
|            |
|            |
|            |
|            |
|            |
|            |
|            |
|            |
|0  0     0  |
```

```
|0  0   00    |
|0000  00   0|
|==========|
|            |
|            |
|            |
|            |
|            |
|            |
|   00       |
|   000      |
|0  000      |
|000000  00  |
|000000  000|
|==========|
|            |
|            |
|            |
|            |
```

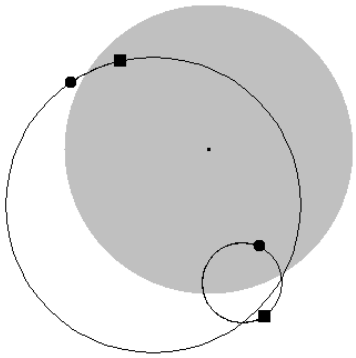
```
|            |
|            |
|            |
|            |
|0  0  0     0  |
|==========|
|            |
|            |
|            |
|            |
|            |
|            |
|            |
|   0     0    |
|  00  000     |
|000  000  00|
|==========|
```

Problem 2: The Walking Dead

The zombie apocalypse has finally arrived. You were at school when it happened and now you are trapped in a portable classroom on the edge of a schoolyard that is swarming with the re-animated corpses of your former teachers and classmates. Fortunately, you saw this coming weeks ago and cunningly buried explosive devices in key locations. One of them is under the schoolyard. You can start the bomb's countdown timer from your smart phone. Right now the zombies are just milling around aimlessly. When the bomb detonates it will destroy some of them and the rest will start moving towards the sound, giving you the space you need to make it across the field.

Zombies always walk with a limp, so when they have nowhere in particular to go they end up walking in circles. When you trigger your explosive device, the zombies will continue to walk during the countdown and then when the bomb goes off, any zombie inside the circular blast area will be destroyed.

In the picture at right, the grey circle shows the area covered by the bomb blast and the outlined circles show the paths of two zombies. One is moving quickly (for a zombie) in a clockwise direction and is limping quite badly so it walks in a tight circle. The other is limping less and moving more slowly in a counter-clockwise direction. The black squares show their positions when the timer was started and the black circles show where they were when the bomb went off. As you can see one zombie was destroyed in the blast and the other escaped.



DATA21.txt (DATA22.txt for the second try) contains 10 different cases. Each case starts with a line containing four numbers Bx, By, R, and T separated by spaces. The location of the bomb (in metres relative to the center of the schoolyard) is given by the floating point numbers Bx and By ($-500.0 \le Bx, By \le 500.0$ – it's a big school, ok?). The blast radius of the bomb (in metres) is given by the floating point number R ($10.0 \le R \le 500.0$). T is an integer representing the number of seconds until the bomb goes off. This is followed by an integer N on its own line representing the number of zombies on the field ($1 \le N \le 1000$). This is followed by N lines, each representing a zombie using 5 floating point numbers separated by spaces: X, Y, Cx, Cy, and S. The position of the zombie is given by X and Y and the centre point of the zombie's circular path is given by Cx and Cy ($-500.0 \le X, Y, Cx, Cy \le 500.0$). S is the speed of the zombie in metres per second ($-1.0 \le S \le 1.0, s \ne 0$) where a positive speed indicates clockwise motion and a negative speed indicates counter-clockwise motion. All floating point numbers in the file have at most two digits after the decimal point.

Your job is to output an integer representing the number of zombies that will be destroyed by the bomb when it goes off. When determining whether a zombie is within the bomb blast area, you should round its position to two decimal places and consider the zombie destroyed if it is within the bomb blast or if it is exactly on the border. Don't worry about zombies bumping into each other as they walk, just assume that won't happen. Note that the sample data below consists of only 5 cases with boldfacing used to separate them visually. The real data files have 10 cases each.

Sample Input

```
0.0 0.0 130.0 500
2
50.0 -150.0 30.0 -120.0 0.3
-80.0 80.0 -50.0 -50.0 -0.1
-17.11 16.81 21.99 79
4
-7.91 22.79 -18.07 13.85 -0.13
-19.53 -19.43 -19.37 13.35 -0.21
10.65 -1.34 0.23 -16.52 0.2
-3.25 -24.61 16.21 18.87 0.17
1.48 21.27 20.24 13
12
-3.6 -20.42 23.02 -15.12 -0.16
19.95 -14.18 -6.0 -23.23 -0.18
5.15 13.06 20.26 11.81 0.26
4.16 -18.42 -13.43 16.23 -0.07
-24.79 12.28 4.11 9.55 -0.13
12.31 11.1 11.87 20.86 0.05
11.78 12.93 -7.82 -24.35 0.03
-14.71 -15.49 16.43 -3.9 0.15
14.31 5.03 -22.68 16.29 -0.05
-22.1 4.86 13.96 -7.16 -0.2
-21.16 -11.18 4.71 -7.64 -0.04
12.74 14.08 -1.88 -5.92 0.16
-13.83 4.62 21.45 61
16
-7.66 -3.24 1.33 -6.53 0.03
-19.0 -0.56 -13.26 24.93 0.03
-7.5 -20.3 -23.88 -5.15 -0.01
-12.09 -4.41 14.17 -0.82 0.08
-11.48 18.29 -1.8 7.82 0.15
20.57 13.01 -17.16 -3.08 -0.15
8.39 0.29 -8.25 -23.22 0.04
6.07 -16.38 -0.48 22.44 0.11
16.92 20.4 0.64 -16.19 0.02
-14.67 -1.2 -18.32 3.41 0.25
16.27 1.26 -2.92 18.78 -0.12
-9.25 1.74 -8.01 2.07 -0.07
10.88 17.68 6.2 3.94 0.23
21.08 -9.37 10.94 1.29 0.09
11.31 8.68 -17.95 9.54 -0.1
-1.46 21.11 13.39 9.04 -0.12
4.57 8.57 18.83 169
20
22.2 5.22 18.09 18.98 0.04
-24.75 -14.48 -24.14 -16.0 0.01
-5.22 -16.06 5.75 -3.2 -0.05
9.5 22.34 12.87 -24.34 0.06
-22.32 19.67 16.79 8.64 -0.02
4.46 10.22 14.43 22.25 -0.21
-15.7 -18.05 1.37 -5.16 0.19
-8.34 -6.63 -16.61 16.73 -0.07
22.78 -17.16 19.91 12.26 0.23
-14.35 18.54 1.5 -11.14 0.14
-5.22 9.95 21.32 5.24 0.08
1.61 9.98 -22.21 -2.42 0.22
-12.72 15.29 -3.63 -19.68 0.05
-3.74 9.51 -7.84 -3.44 -0.08
-20.14 -17.28 -0.82 2.43 -0.24
-11.01 -12.15 -0.81 11.95 -0.17
10.91 1.02 -13.54 18.59 -0.02
-6.2 7.71 18.18 -20.29 -0.08
-22.82 -24.88 23.3 -7.44 -0.13
20.95 -16.7 7.61 -23.1 -0.06
```

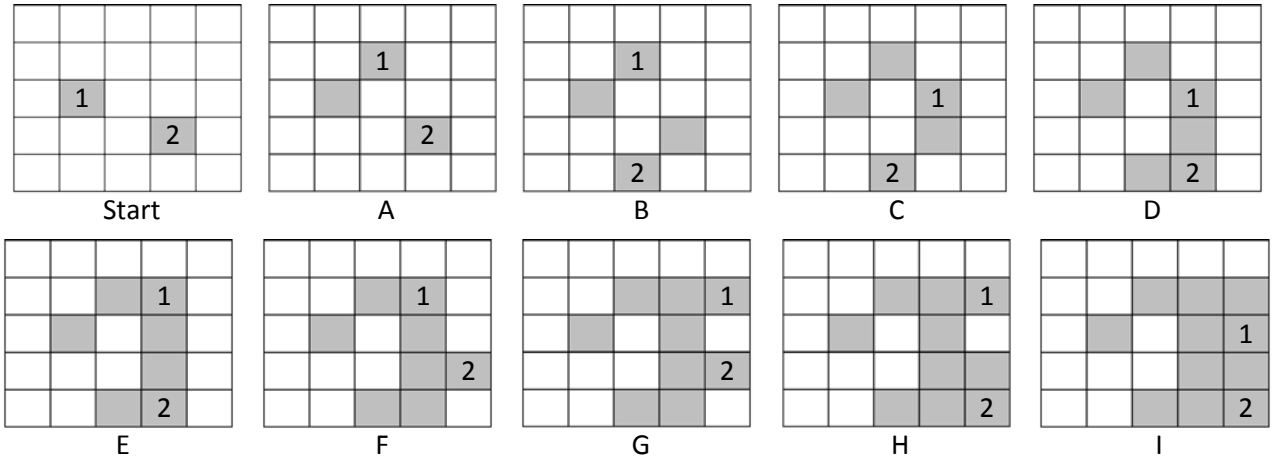
Sample Output

```
1
1
4
7
8
```

Problem 3: Last Robot Standing

You are entering your robot into a simple game of strategy. The game is played on a grid and the robots take it in turns to move one space at a time (including diagonally) around the grid. When a robot moves into a square, that square is taken and nobody else can move into it for the duration of the game. When a robot is boxed in and cannot move on its turn, it is out. The winner is the Last Robot Standing.

Here's an example of the game in action on a small grid with 2 robots. The positions of the robots are marked 1 and 2 and the shaded squares are taken and cannot be used again by either robot.



In this example, Robot 1 is the Last Robot Standing. Note that Robot 2 was boxed in on Move H, but was not removed from the game at that point. Any robot that is able to move on its turn is safe for that turn. It is only on its next move (Move J) that it is removed from the game. Note that if Robot 2 had started out just one square to the left and used the same set of moves, it would have won.

8	1	2
7		3
6	5	4

The robots in the above game are choosing their moves from a hardwired "strategy" that tells them the order of moves to try. There are 8 possible moves, numbered clockwise from 1 to 8 as shown at right. A robot's strategy can be represented as a list of numbers. Robot 1's strategy is 2481377746543, and Robot 2's strategy is 62437428513.

On Robot 1's first turn (A), it uses move 2, which is up and right. Then on its next turn (C) it uses move 4 (down and right). Then on its next turn (E) it tries move 8 (up and left) but is blocked, so it uses the next move in its strategy, which is 1 (up). If a Robot ever runs out of moves in its hardwired strategy, it simply starts again from the beginning. Note that this can only work if each of the 8 moves appears in each strategy at least once.

Using your evil hacking skills, you have found out the strategies of the other robots. Since you get to pick your starting position and you're moving last, you can figure out a winning start position.

DATA31.txt (DATA32.txt for the second try) will contain 10 cases. Each case starts with a line of 3 integers R, C, and N. R and C are the number of rows and columns on the playing surface ($4 \le R,C \le 10$) and N is the number of robots in the competition ($2 \le N \le 16$). Your robot is the N^{th} one. Following this are N-1 lines containing the integer starting position for each of the other robots (row then column, numbered from 1 upwards), then N lines containing the strategy for each robot (including your own). During the game, the robots take turns in the order they appear in the file, with your robot moving last.

Your job is to output a list of starting positions that will lead to a guaranteed win for your robot. If there is no such starting position, you should simply output the word "LOSE". Your output must be formatted exactly as shown below with no extra spaces or punctuation, but the order you output the starting positions for each case can be different from that shown. Note that the sample input below only contains 4 cases, but the data files will contain 10 cases each.

Sample Input

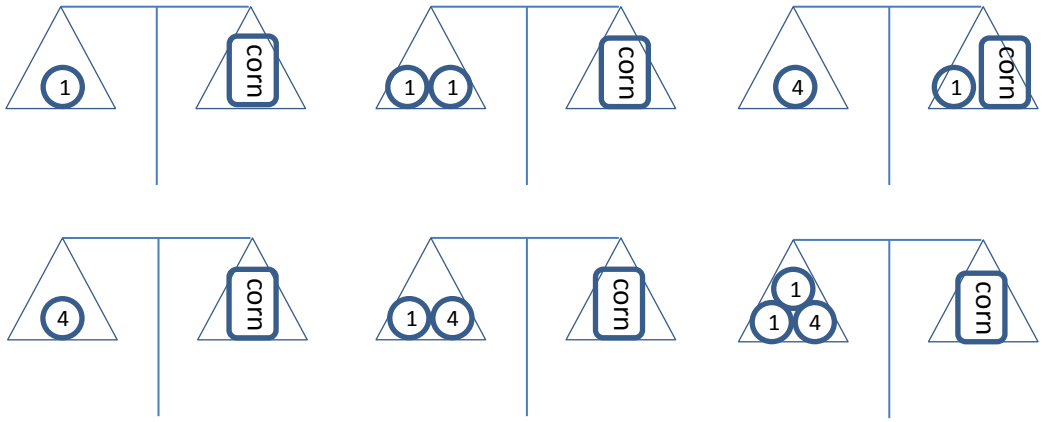
```
5 5 2
3 2
2481377746543
62437428513
3 3 3
1 1
2 2
45362718
45362718
87654321
5 5 3
1 1
3 3
81726354
45362718
12345678
10 10 8
1 3
1 8
3 1
8 1
3 10
8 10
10 3
12345678
87654321
18273645
54637281
21436587
78563412
16372485
73265841
```

Sample Output

```
(1,2) (1,3) (2,3) (2,4) (3,1) (4,5) (5,2) (5,5)
(3,3)
(3,2) (4,2)
(1,9) (4,9) (4,10) (5,7) (7,1) (7,2) (7,6) (7,7) (9,4)
```

Problem 4: Breaking Rocks

Farmer Jane needs to be able to measure out corn to feed her cows, but all she has to help her is a primitive balance scale and a 6 kilogram rock. With this rock she could use the balance scale to measure out 6 kg of corn, but she often needs to measure out smaller quantities. She figures out that if she breaks the rock into three pieces, where two of them are 1 kg and the third is 4 kg, then she can measure out all integer quantities of corn from 1 to 6, as shown below.



Farmer Jane is happy now, but the situation gets her thinking. She knows she could have broken the rock into 1, 2, and 3 kg pieces and this would also have worked. But things are not so simple for other numbers. For example, there are 15 ways to break a 12 kg rock into 4 integer pieces but only 9 of them let you measure all integer weights from 1 to 12. She wonders if there could be some kind of algorithm to determine how many combinations work for a given size of rock and a given number of pieces…

DATA41.txt (DATA42.txt for the second try) contains 10 test cases. Each test case consists of two integers (**P** and **R**) on a single line separated by a space. The integer **P** gives the number of pieces to break the rock into and the integer **R** gives the original size of the rock. For all test cases, $1 \leq R \leq 100$. For the first 5 test cases $3 \leq P \leq 5$ and for the next 5 test cases $6 \leq P \leq 10$. Your job is to output a single line for each test case indicating the number of ways you can break up the rock into **P** integer-sized pieces so that all possible integer weights from 1 to **R** can be measured on a balance scale.

Sample Input

```
3 6
4 12
4 30
5 40
5 5
6 25
7 55
8 65
9 75
10 85
```

Sample Output

```
2
9
5
137
1
154
5749
28051
121108
474402
```

ECOO 2013

Programming Contest Questions

Final Competition (Round 3)

May 11, 2013

Problem 1: Irregular Expressions

You might have heard of Regular Expressions, but this question is about Irregular Expressions, a concept we just invented. An irregular expression is a pattern string that can be used to match target strings. Here's an example pattern that can be used to "accept" or "reject" target strings:

```
adbsk[kkjsvf]bbb
```

There are two rules for matching a target string to a pattern like the one above:

1. Any lower case letter that is not inside square brackets in the pattern must match to the same letter in the target string, in the correct position.
2. If a set of lower case letters appears in square brackets in the pattern, the target string must contain, at that point in the string, exactly half of these letters (rounded up) in any order.

There are lots of target strings that match the above pattern (60 to be precise). Here are a few of them. The parts that match the bracketed portion are highlighted:

adbsk**kkj**bbb, adbsk**jkk**bbb, adbsk**skf**bbb, adbsk**fjk**bbb

Of course there are also an infinite number of targets that do not match. Here are a few of them:

adbskkkjsbbb, adbskjkbbb, adbsjsvbbb, adbsksfjbb, fhfghj tomato pesto

An irregular expression can be any length, can use any plain lower case letter, and can contain zero or more non-nested square bracket sections.

DATA11.txt (DATA12.txt for the second try) will contain 10 lines. Each line will consist of an irregular expression pattern string followed by 5 target strings to match, all separated by spaces. Your program must output either "true" or "false" for each target string depending on whether it matches the pattern. The words "true" and "false" must be lower case and separated by a single space. You should produce a separate line of text for each line in the input. The maximum length for each pattern and each target string is 256 characters. Note that the test data below has only 5 lines, but the real data file will have 10.

Sample Input

```
adbsk[kkjsvf]bbb adbskkkjbbb adbskkkjsbbb adbskskfbbb adbsjsvbbb fhfghj
fkqm[qzqyledbqq]c fkqmdqqqlc fkqmqlbyq fkqmqedqc fkqmblqqzc fkqmlqbdzc
dj[mocn]dd djnmdd djocdd djmodd djmodd djcndd
wsvahfskbs[uucysmlfrcnhu]mjgphbd this problem is ridiculous dude
a[aauzujbtipmca]bsk aiamzuausk acuaaujabsk aaaauucbsk aazpubtubsk aiaujcambsk
```

Sample Output

```
true false true false false
true false false true true
true true true true true
false false false false false
false true false true true
```

Problem 2: Mutant Children

The field of Genetic Algorithms was inspired by what we know about evolutionary theory and the chemistry of sexual reproduction. Computer scientists use genetic algorithms when they have a problem they don't know how to solve, but can represent a possible solution as a "chromosome" – that is, a string of binary "genes".

Here's an example chromosome with 25 genes:

```
1101101100011010100011111
```

The Genetic Algorithm starts with a population of randomly generated chromosomes like the one above, all the same length, then selects the best ones to "breed" to produce "children" and repeats the process until an acceptable solution is found. The hope is that after hundreds or thousands of generations, you will find at least one good solution to your problem.

When two "parents" produce children, it goes like this:

1. Pair up the parents

   ```
   1101101100011010100011111   ← first parent
   0010110001111000101001100   ← second parent
   ```

2. Pick two crossover points (A and B)

   ```
   1101101100011010100011111   Note: A and B must be different and can't
   0010110001111000101001100   be the first or last gene location.
        ↑                  ↑
        A                  B
   ```

3. Exchange genes to produce two children. To do this, the parents swap their middle sections using the crossover points from step 2.

   ```
   11011000111100010100 1111   ← first child
   00101 0110001101010001 1100   ← second child
   ```

4. Mutate the children by randomly changing a few genes.

   ```
   0 1011100011000010100 1111
   00 11 0110001101010 1 011 000
   ```

Every Genetic Algorithm has a "mutation rate" parameter that represents the probability that each gene will mutate when children are produced. For example if the mutation rate is 0.05 then every gene in every child produced has a 5% chance of mutation.

Your job in this question is to estimate the mutation rate given a pair of parents and one of their two children. To make your estimate, find the minimum possible number of mutations in the child assuming that it was produced through the process described above, then divide this minimum number of mutations by the number of genes to get the estimated mutation rate.

DATA21.txt (DATA22.txt for the second try) contains 10 test cases. Each test case consists of 3 chromosomes of equal length, each on a separate line. Chromosome length can range from 10 to 10 000 genes. The first two chromosomes in each test case are the parents and the third is one of the two children produced from these parents. Your job is to output a floating point number rounded to two decimal places that represents your estimate of the mutation rate. You must include a leading 0 in your answer and you must output two decimal places. For example, if the estimated mutation rate is 30% you should output "0.30".

Sample Input

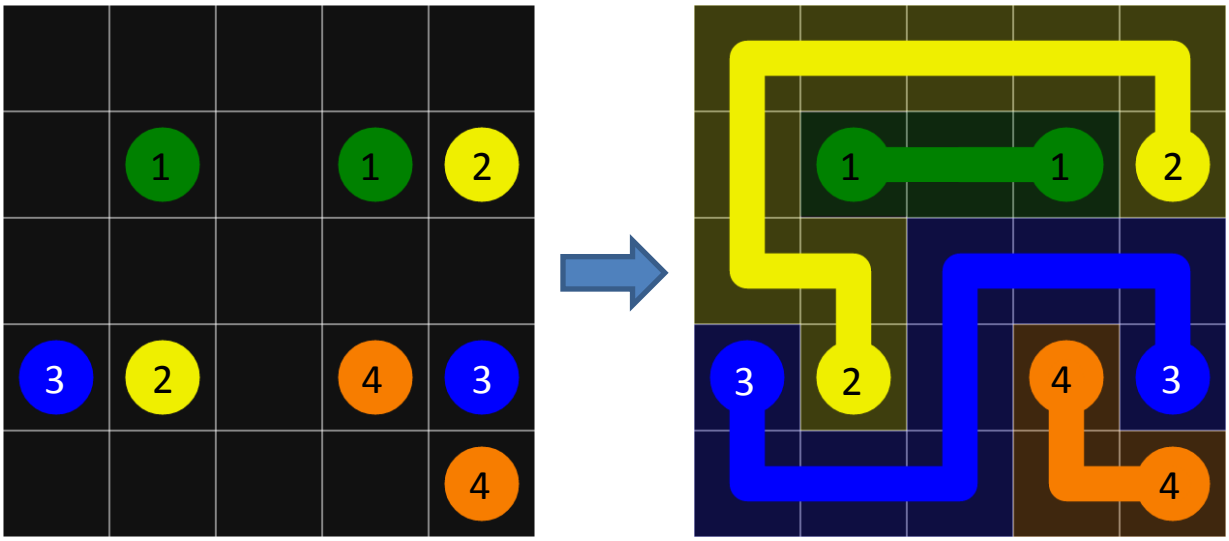
```
0001000010
0100100011
1000101010
01101010011
00000100101
01010100011
101001100110
100011001101
110011000010
1011111101110
1000110000100
1110110000110
11001001110010
01000111011010
01111011100110
110011111011010
100101111110001
110101111010110
1001111010111100
1111111100000101
1000101100001111
11011011100000110
00010001100010100
11011001001010110
1000101100101000000
1100101011111110000
1000101011111101100
00101001101111100101
10000111111101010011
10011000000010111100
```

Sample Output

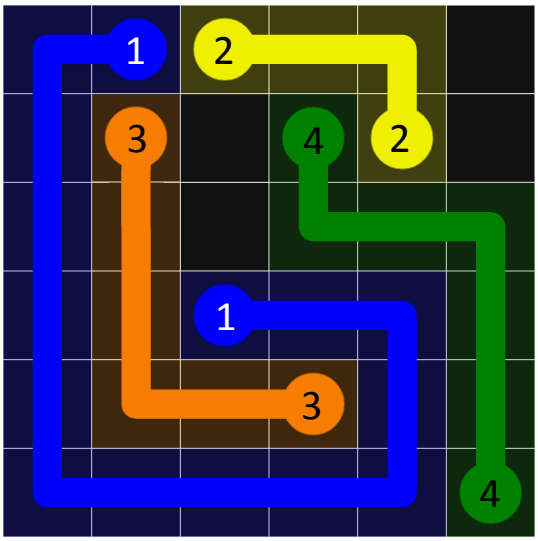
```
0.20
0.09
0.17
0.08
0.36
0.13
0.25
0.12
0.11
0.42
```

Problem 3: Go With the Flow

There's a popular mobile game app in which the goal is to join coloured dots on a grid. You start with a board like the one below left. Your job is to draw a path from each dot to its partner of the same color, making sure you use all the available grid spaces, as shown in the picture below right. (Since you may be viewing this in black and white, the dots are numbered to show which ones should be joined.)



The two pictures above represent a win. The picture on the right represents a loss. In this case, all the dots are joined correctly, but not all of the grid spaces have been filled. This board actually has three correct solutions.



DATA31.txt (DATA32.txt for the second try) will contain 10 boards for the game described above. The boards will always be square with a size ranging from 5x5 to 7x7. Each board has only one possible winning solution. The different coloured dots will be represented as sequential digits starting at 1 up to a maximum of 9, and the empty grid spaces will be represented with a dot character (ASCII value 46). There will be no spacing between board squares and no blank lines between boards.

Your job is to output the winning solution for each board. Each path should be represented using the number of the two dots the path is joining, as shown in the sample solutions on the next page. You should have a single blank line separating each board in the output, and you must have a way to show the judges all 10 boards, either by scrolling the output, or by displaying the boards under input control (e.g. press the space bar to make each board appear). You will have 10 seconds to compute the solution

for each board. If at any point a solution to a test case takes longer than 10 seconds to appear (after the previous one, or after pressing a key) scoring will stop immediately and you will be awarded the points you have accumulated so far.

Note that in the sample data, there are only 5 boards but the judging files will contain 10 boards each.

Sample Input

```
.....
.1.12
.....
32.43
....4
1....
.....
..2..
324.1
4...3
123.45
....6.
..3...
..4...
1.6...
2.5...
.12..3
......
..45.3
...6.2
.54.61
......
......1
.....23
.2.....
...45..
..4.6..
....36.
.....15
```

Sample Output

```
22222
21112
22333
32343
33344

11111
33331
32231
32431
44433

123445
123465
123465
124465
126665
225555

112223
155523
154523
154622
154661
111111

1111111
1222223
1233333
1334555
1344665
1333365
1111115
```

Problem 4: Tour de Force

A few years after Trivial Pursuit became a hit, Canadian authors Pierre Berton and Charles Templeton created a rival trivia game called Tour De Force. Unfortunately, it never caught on because it wasn't a very good game.

Like Trivial Pursuit, Tour de Force has question cards but there are only two questions per card and each is worth a different number of points. On your turn, another player asks you the first question on your first card. If you get it right, you get the points and you have the option of trying the second follow-up question on the card (usually on the same subject). If you get that right, you get those points as well, and you have the option of taking another card and continuing. If at any point you get a question wrong, your turn is over, the card you are working on is discarded, and you lose a point. There is no penalty if you stop voluntarily. If you manage to answer 10 questions in a row (5 cards), you score a "Tour de Force" and you win immediately.

For example, suppose the first card has questions worth 1 and 3 points. You get the first right and choose to go for the second one. You get that one right too and choose to take another card. Then on this card, the questions are worth 2 and 4 points. You answer the first correctly, and then try the second, but mess it up. Your total score is 5 points for the turn (1 + 3 + 2 – 1). If you had chosen not to move on after the third question, you would have scored 6 points. If you had answered the first question on the second card wrong, you would have scored 3 points and the second card would have been discarded (meaning the 4 point question would not be used on subsequent turns).

A smart Tour De Force player quits when they're ahead to avoid losing a point for the last question. But Bert is not a smart player. He always goes for a Tour De Force on every turn even though he has never once made it. Your task is to figure out the maximum possible score Bert can get (without scoring a Tour De Force) with any given set of cards.

For example, suppose there are 6 question cards in the deck with pairs of point values [8 3], [4 5], [3 1], [2 2], [6 7], and [2 3] in that order. Then the maximum score Bert can get is 44. He achieves this by going on a streak until the second question on the third card, then going on another streak to the end of the cards. This gets him 8 + 3 + 4 + 5 + 3 – 1 + 2 + 2 + 6 + 7 + 2 + 3 = 44 points. Note that we are ignoring other players and assuming all 6 cards are just for Bert.

DATA41.txt (DATA42.txt for the second try) contains 10 test cases. Each test case starts with an integer N on a single line representing the number of cards in the case ($6 \le N \le 1000$). This is followed by N lines for the cards in the order that they will appear in the game. Each line contains two integers representing the point values of the first and second question on the card, respectively (each question is worth between 1 and 10 points). Your program should output the maximum possible score Bert can get by following his strategy of never turning down a card or a question, assuming that he will never make it all the way to a Tour De Force. He can take as many turns as he likes to get through the cards, and can be in the middle of a turn when the cards run out.

The sample input on the below is shown in columns to save space (two test cases per column) and the test cases are separated visually using boldface.

Sample Input

```
10
7 1
4 1
3 8
4 6
2 2
4 3
4 5
10 6
8 6
7 2
6
4 4
7 5
2 4
8 1
9 3
5 7
```

```
9
2 6
2 8
3 1
6 3
7 4
2 6
1 10
9 9
7 1
8
2 9
1 6
10 3
9 6
6 2
9 9
2 5
4 1
```

```
7
4 8
7 1
1 1
3 3
7 3
6 3
3 8
8
6 8
1 2
10 4
7 7
10 8
5 8
8 5
3 10
```

```
6
4 9
4 5
8 4
9 3
8 10
10 10
9
6 2
9 9
5 4
10 8
3 7
4 1
2 1
8 6
4 8
```

```
8
1 4
10 1
1 10
7 2
8 5
3 3
6 6
5 10
6
1 8
7 6
9 10
1 3
10 7
7 8
```

Sample Output

```
87
57
82
81
56
94
80
92
79
73
```