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

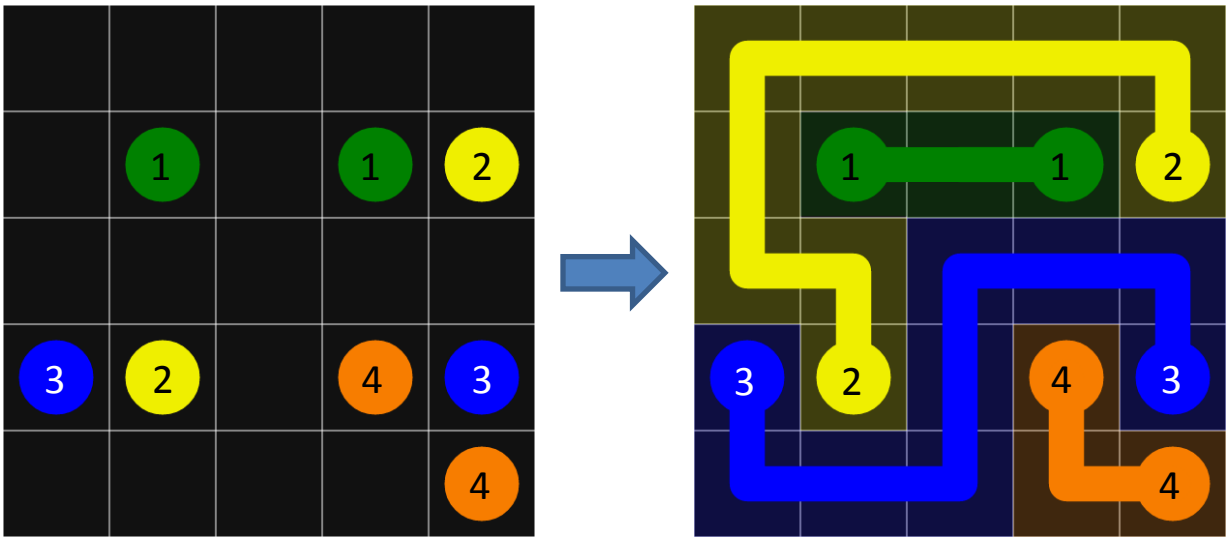
```
0001000010
0100100011
1000101010
01101010011
00000100101
01010100011
101001100110
100011001101
110011000010
1011111101110
1000110000100
1110110000110
11001001110010
01000111011010
01111011100110
110011111011010
100101111110001
110101111010110
1001111010111100
1111111100000101
1000101100001111
11011011100000110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

Sample Output

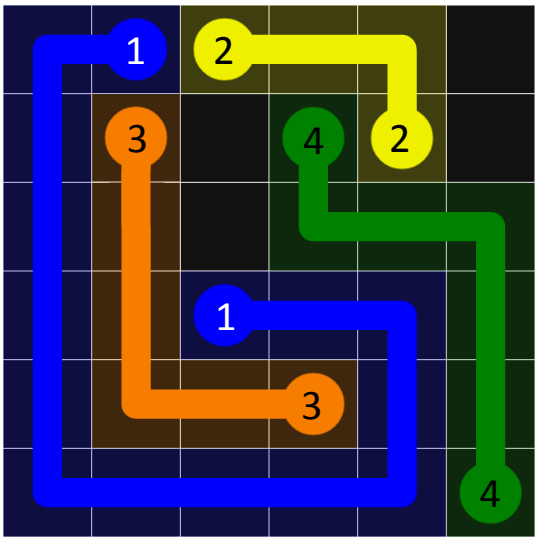
```
0.20
0.09
0.17
0.08
0.36
0.13
0.25
0.12
0.11
0.42
```

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
```