ECOO 2013
Programming Contest Solutions and Notes

Local Competition (Round 1)

March 20-27, 2013

Sam Scott, Sheridan College (sam.scott@sheridancollege.ca)

Problem 1: Take a Number

Recommended Approach

This is a fairly straightforward simulation. Here is one way to do it:

For each "day" of the simulation, get the first number (number of late students) by counting the occurrences of the word "TAKE", the second (number left in line at end of day) by subtracting the number of occurrences of "SERVE", and the third (next number in machine) by keeping track of the next number, adding 1 after each "TAKE" event, and resetting it to 1 if it goes over 999.

Solution to DATA11.txt

```
1724 42 834
1110 0 945
1454 166 401
2263 135 666
282 26 948
497 85 446
0 0 446
1110 90 557
1626 93 185
342 3 527
```

Solution to DATA12.txt

```
908 54 646
5 1 651
588 38 240
449 28 689
324 36 14
2170 143 186
2117 162 305
1620 75 926
1339 138 267
874 17 142
1458 111 601
1080 67 682
2041 127 725
1898 107 625
555 43 181
1907 85 90
1977 127 69
2091 138 162
1232 49 395
1873 151 270
```

Problem 2: The Luhn Algorithm

Recommended Approach

Perform the Luhn Algorithm on the given number, but modify it to take account of the fact that the last digit is not there. After the sum step, isolate the last digit of the sum using the integer modulus (remainder) operator, then subtract this number from 10 to get the check digit.

Another Approach

From each credit card number, generate 10 candidates for all the possible check digits. Then run the Luhn algorithm on each candidate and find the one that comes out valid using the Luhn Algorithm.

Test Cases

In each batch of test cases, the first 2 contain numbers between 2 and 10 digits. The final 3 contain numbers between 1 and 100 digits. One test case in each batch contains a single digit number.

Solution to DATA21.txt

```
65437
60714
00909
99258
92214
```

Solution to DATA22.txt

```
18048
69854
48516
43680
49351
```

Problem 3: Hexudoku

Recommended Approach

There is not much to say here except that you have to simulate the strategy exactly as described, preferably using a two dimensional array or an array of strings to represent the board.

The Test Data

Each set contains a few boards that are highly filled in and a few that are fairly sparse.

Solution to DATA31.txt

```
157
81
256
189
59
191
184
198
58
44
```

Solution to DATA32.txt

```
165
78
189
59
229
193
190
193
58
54
```

Problem 4: Coupon Day

Recommended Approach

I think the best approach is to treat the BOGO coupon separately from the other coupons. If there is no BOGO, use a backtracking search, trying each coupon on each unused price. But if there is a BOGO, first try the backtracking search without that coupon, then sort the prices in ascending order, apply the BOGO to each adjacent pair of prices, then use the backtracker to fill in the rest. Keep track of the best score from all these searches and print it out. If done correctly, this approach is guaranteed to find the optimal solution. It is also possible to solve this a bit more quickly with a "best first" search, but this is more complicated and involves the use of a priority queue structure.

Random Approach

Because the problem size is so small, a random search works quite well too. In this approach you simply randomly pair items to coupons and compute the resulting price. Do this a bunch of times (10,000 or 100,000) and you are almost guaranteed to come up with the correct answer, making this a great strategy for the purposes of this contest. When the random search was coded by one of the problem testers, even iterating 100 times came up with correct answers 9 times out of 10.

Greedy Approach

You can get a long way substituting a greedy algorithm for the backtracker in the recommended approach above. The greedy algorithm repeatedly looks for the best price-coupon pairing and applies it. If there is a tie among the flat coupons, it chooses the one that is least wasteful (e.g. if the best is $10 coupon applied to either $53 or $14, apply it to $14.) This works for about 90% of test cases, but there are still a few where it does not. For example, prices 88.17, 43.18, 67.14, 2.51 and coupons 20%, 20%, $50, $50. Greedy wants to apply the $50 coupons to the two highest prices, but the better solution is to apply it to the middle two and use a 20% coupon on the other two. It is also possible to do the greedy but mess up the BOGO (e.g. include it in the regular greedy search instead of exhaustively trying all BOGOs). This is a problem when there is a two-coupon combo that beats the BOGO.

Pitfalls

There is a catch in the TAX coupon. When you calculate the savings on the $5, $10, and $50 coupons, you must take the savings in HST into account as well. Otherwise there is a small range of dollar values where you will mistakenly think the TAX coupon saves you more than the other one. For $5, the range is $39-$43. For $10, it's $77-$86. And for $50, it's $385 to $434. You also have to be very careful with rounding in this problem, and when reading the numbers from the file you should apply rounding to the numbers you get because of possible floating point precision errors.

The Test Cases

The contest team coded 6 solvers. Greedy, Greedy with the BOGO mistake above, Greedy with the TAX mistake above, a backtracker, best first, and a random search. Each set of test cases has one case that work with all four approaches, and then at least two that fail on the Greedy version and then one that fails on the versions that have bad BOGO and bad TAX calculations. Backtrackers or random search algorithms with the TAX mistake might also fail on some cases.

Solution to DATA41.txt

```
The best price is $431.02
The best price is $337.19
The best price is $279.92
The best price is $288.15
The best price is $106.32
```

Solution to DATA42.txt

```
The best price is $247.09
The best price is $217.20
The best price is $149.01
The best price is $173.08
The best price is $132.41
```