University of Waterloo
CS 466 — Advanced Algorithm
Spring 2013
Problem Set 1
Siwei Yang - 20258568

1. [7 marks: Greed and Approximation] As part of the process of dropping the penny as a unit of coinage, the Treasury Board observed that there would be room for a new coin in cash registers. As a consequence they have decided to introduce a new 20 cent coin (so we will have 5, 10, 20, 25, 100 and 200 cent coins, and all charges will be rounded to the nearest multiple of 5 cents). The staff of the Math Faculty C&D use a greedy algorithm for making change. They repeatedly give the largest coin that will not exceed the total. In the past this has enabled them to make change using as few coins as possible.

   a. Show that this method will no longer always give the fewest coins possible.

      Assume the cashier needs to give back a customer 40 cents in change. By using the greedy algorithm, the cashier gives the customer coins of 5, 10, 25. However, the optimal way to give back change is to use 20, 20.

   b. Show, however, that the number of coins using this heuristic will be within one of the minimum possible for any amount required. (We will not deal with other suboptimal heuristics as close to optimal as this.)

      Assume the cashier needs to give back a customer n * 5 cents in change. Let $c_0 * 5, c_1 * 10, c_2 * 20, c_3 * 25, c_4 * 100, c_5 * 200$ be one of the optimal ways to give back change while minimizing number of the 20 coins. Consider c_0, since two 5 coins can be replaced by a 10 coin to reduce number of coins used, we always have

$$c_0 < 2 \tag{1}$$

      For the same reason, we also have

$$c_1 < 2, c_2 < 5, c_3 < 4, c_4 < 2 \tag{2}$$

Note we picked a combination that minimize c_2, and three 20 coins can be replaced with two 25 coins and a 10 coin while maintaining the number of coins used, we always have

$$c_2 < 3 \tag{3}$$

Consider c, we have $c_2 \in \{0, 1, 2\}$. If $c_2 = 0$, we can create another combination $c^{''}$ where $c_i^{''} = c_i$. $c^{''}$ is a combination without 20 coin that is optimal. If $c_2 = 1$, then c is a combination with one 20 coin. We can create another combination $c^{''}$ where $c_i^{''} = c_i^{'}$, except $c_1^{''} = c_1^{'} + 2, c_2^{''} = 0$. $c^{''}$ is a combination without 20 coin that is one worse than optimal. If $c_2 = 2$, then c is a combination with two 20 coin. We can create another combination $c^{''}$ where $c_i^{''} = c_i^{'}$, except $c_0^{''} = c_0^{'} + 1, c_1^{''} = c_1^{'} + 1, c_2^{''} = 0, c_3^{''} = c_3^{'} + 1$. $c^{''}$ is a combination without 20 coin that is one worse than optimal. **there always exists $c^{''}$ without 20 coin that is at most one worse than optimal.**

Let $c_0^{'} * 5, c_1^{'} * 10, c_2^{'} * 20, c_3^{'} * 25, c_4^{'} * 100, c_5^{'} * 200$ be way to give back change using greedy algorithm.

Following the algorithm, we have

$$c_0^{'} < 2, c_1^{'} < 2, c_2^{'} < 2, c_3^{'} < 4, c_4^{'} < 2 \tag{4}$$

Given that $c_2^{'} \in \{0, 1\}$, if

$$c_2^{'} = 0 \tag{5}$$

then $c^{'}$ is a combination without 20 coin

Compare $c, c^{''}$, both use 5, 10, 25, 100 and 200 cent coins to represent the same total change. Since c is optimal when 20 coin is not used, c is at least as good as $c^{''}$. Also $c^{''}$ is at most one worse than optimal. **Thus c is at most one worse than optimal**.

If

$$c_2^{'} = 1 \tag{6}$$

Then, by the property of greedy algorithm

$$c_0^{'} = 0, c_1^{'} = 0 \tag{7}$$

Given that, we have

$$n * 5 \equiv 20 \mod 25 \tag{8}$$

Thus, create $c^{'''}$ by using greedy algorithm with 5, 10, 25, 100 and 200 cent coins. And, we know $c^{'''}$ is optimal when 20 coin is not used, $c^{'''}$ is at least as good as $c^{''}$. Also $c^{''}$ is at most one worse than optimal. **Thus, $c^{'''}$ is at most one worse than optimal**. Not only that, we have

$$c_0^{'''} = 0, c_1^{'''} = 2, c_2^{'''} = 0 \tag{9}$$

Therefore, $c^{'''}$ is one worse than c. **Thus, c is optimal**.

2. [8 marks: Avoiding Amortization] We briefly discussed the problem of counting the number of times certain events occur and maintaining them in a list by decreasing order of frequency. Assume we have n items and can reference any one of their records in constant time. Design a data structure that permits you to:

   - Increment or determine the count of any element in constant time, and
   - Access the elements in increasing or decreasing order by count, also in constant time for each element inspected.