Jade Yu Cheng
ICS 311
Homework 6
Sep 15, 2008

Question for lecture 7

Problem 16-1 on p. 402

Coin changing
Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer.

a. Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.

Answer: The algorithm for solving this real-life problem is exactly what we do in real life, which is to always use the greatest value coins for the existing amount and to use as many of those coins as possible without exceeding the existing amount. After deducting this sum from the existing amount, we use the remainder as the new existing amount and repeat the process.

This is a greedy algorithm. We apply the best solution for the current step without regard for the optimal solution. This is usually easy to understand and simple to implement. In the case of American currency, the greedy algorithm provides the best solution because locally optimal solutions lead to globally optimal solution as well.

Proof:

1. Let's consider pennies and nickels. At most, I can use 4 pennies because any number larger than 4 pennies would be replaced by at least 1 nickel. This operation would reduce the total coin number by 4. In other words, when the remainder is greater than 5 and I'm allowed to use only pennies and nickels, I would use as many nickels as possible before considering pennies.

2. Let's now consider pennies, nickels, and dimes. At most, I can use 1 nickel because any number larger than 2 nickels would be replaced by at least 1 dime. This operation would reduce the total coin number by 1. In other words, when the remainder is greater than 10 and I'm allowed to use only pennies,

nickels, and dimes, I would use as many dimes as possible before considering nickels and pennies.

3. Now, let's consider pennies, nickels, dimes, and quarters. At most, I can use 2 dimes because any number larger than 3 dimes would be replaced by 1 quarter plus 1 nickel. This operation would reduce the total coin number by 1. In other words, when the remainder is greater than 30, I would use as many combinations of quarters and nickels as possible. Then, of course, the actual amount of nickels used would fall into the second case. They would be replaced by dimes if possible. So, the solution is to use as many quarters as possible when the remainder is greater than 30.

4. For the gap between 25 and 30, based on the first two arguments, I would use 2 dimes, 1 nickel, and $n-25$ pennies. Obviously, I would use 1 quarter to replace the 2 dimes and 1 nickel. This operation would reduce the total coin number by 2.

Therefore, the greedy algorithm provides the optimal solution for this set of coin denomination.

b. Suppose that the available coins are in the denominations that are powers of c, i.e., the denominations are $c^0, c^1, c^2, ..., c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

Answer: Proof:

1. The reasoning is very similar to that of the previous question. Let's consider the first two types of coins in this set, pennies and coins that are worth c cents each, which we will call $\hat{c}$. At most, we can use $c-1$ pennies because any number larger than c pennies would be replaced by at least one $\hat{c}$ coin. This operation would reduce the total coin number by $c-1$. In other words, when the remainder is greater than c and I'm allowed to use only pennies and $\hat{c}$ coins, I would use as many $\hat{c}$ coins before considering pennies.

2. The same thoughts apply for larger coins. Let's consider $\hat{c}^{n-1}$ and $\hat{c}^n$ coins. At most, we can use $c-1$ $\hat{c}^{n-1}$ coins. Because any number larger than c^n would be replaced by at least one $\hat{c}^n$ coin. This operation would reduce the total coin number by $c-1$. In other words, when the remainder is greater than c^n and I'm allowed to use only coins that are worth less than $\hat{c}^n$, I would use as many $\hat{c}^n$ coins as possible before considering other coins.

3. Combining the first two arguments, the greedy algorithm applies to all the levels of this particular coin set denomination.

c. Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Your set should include a penny so that there is a solution for every value of n.

Answer: Let's consider the set of coins $\{1, 3, 4\}$. If we try to make change for 6 cents, the solution from the greedy algorithm would yield $1\times4+2\times1=6$ (cents). The total amount of coins used in the solution would be $1+2=3$ (coins).

A better solution, however, would be to use $2\times3=6$ (cents). The total amount of coins used in this optimal solution would be 2 (coins).

d. Give an $O(nk)$-time algorithm that makes change for any set of k different coin denominations, assuming that one of the coins is a penny.

Answer: The greedy algorithm always provides a solution but doesn't guarantee the smallest number of coins used. The greedy algorithm takes $O(nk)$ for any kind of coin set denomination, where k is the number of different coins in a particular set. The algorithm is implemented below in Java:

```
/**
 * Makes change using a recursive Greedy algorithm.
 *
 * @param amount
 *            The amount of change to make.
 * @param coins
 *            The sorted set of coins, ordered from smallest to largest.
 * @return The number of coins used to make the change.
 */
int makeChangeGreedyStyle(int amount, int[] coins) {

  // Check if there is no more change to make.
  if (amount == 0) {
    System.out.println("");
    return 0;
  }

  // Loop over the change in order of greatest to smallest.
  for (int i = coins.length; i > 0; i--) {
    int coin = coins[i - 1];

    // If the next largest coin is found, print out its value.
    if (amount >= coin) {
      System.out.print(coin + " ");
      return 1 + makeChangeGreedyStyle(amount - coin, coins);
    }
  }

  // Arriving here means it's impossible to make change
  // using this greedy algorithm, this amount of change,
  // and this set of coins.
  System.out.print("Cannot make change; ");
  System.out.println("cents remaining: " + amount);
  return 0;
}
```