# CS 4720/5720 Design and Analysis of Algorithms

## Homework #4

## Due: Thursday, November 15, 2018

## If written: due in class

## If typed and submitted on Canvas: 11:59 pm

**Submission requirements:**

1. You may implement the programming part in any computer language you like, but source code must be submitted digitally to the "Homework 4 Code" assignment on the Canvas site. I may ask you to demo if I have doubts about your results.
2. **5% extra credit** if you submit *digital, typed* writeups in PDF format to the "Homework 4 Writeup" assignment on the Canvas site. However, you may also turn in handwritten assignments in class – but assignments submitted in person will not receive the 5% extra credit. The due dates are the same for either submission method, but I will need handwritten assigments turned in to me in class.

**Assignment:**

1. *Edit Distance Problem:* The words *computer* and *commuter* are very similar, and a change of just one letter, $p$ to $m$ will change the first word into the second. The word *sport* can be changed into *sort* by deleting $p$, or *sort* can be changed into *short* just by inserting $h$.

   The *edit distance* of two strings $s_1$ and $s_2$ is defined as the minimum number of character mutations required to change $s_1$ into $s_2$., where a mutation is one of

   (a) change a letter,

   (b) insert a letter, or

   (c) delete a letter.

   **Develop a dynamic programming algorithm to find the edit distance between two words.** Your assignment:

   (a) Write down the algorithm clearly in pseudocode.

   (b) Analyze the best- and worst-case time complexities of the algorithm in terms of $\Theta(\cdot)$.

   (c) **Either** implement the algorithm in a language of your choice and run it on at least 5 pairs of input strings and show the output, **or** show in detail the steps your algorithm would take to process 5 pairs of input strings.

2. *Coin Change Problem:* We did a dynamic programming algorithm for the coin change problem in class.

   (a) **Implement this algorithm in a language of your choice** so that given an array of coin denominations $D[1 \ldots, m]$ and an amount to make change for $n$, it finds the minimum number of coins required to make change. As a reference, see the book's Section 8.1. Make sure your algorithm works for any $m \geq 1$, any set of denominations (you can assume that your set of denominations always contains a 1-cent coin), and any $n \geq 0$.

   (b) Now, consider a greedy algorithm to make change (given the same inputs as above). The idea of the greedy algorithm is similar to the one you implemented for the knapsack problem earlier in the semester: it should add coins one by one, and at each step the next coin it adds should be the largest one that "fits." So if you're making change for $n = 9$ with denominations $D = \{1, 4, 6\}$, the greedy algorithm chooses coins $(6, 1, 1, 1)$, even though it would have been better to do $(4, 4, 1)$.

      i. **Write pseudocode for your greedy algorithm.**

      ii. **What are the worst- and best-case complexities for your greedy algorithm in terms of $\Theta(\cdot)$?**

      iii. **Implement this algorithm in a language of your choice.**

   (c) Compare the performance of your algorithms:

      i. **Work out 3 different sets of coin denominations for which the greedy algorithm is *not* always optimal. For example, you may use $D = \{1, 4, 6\}$.**

      ii. **Run both algorithms on your sets of denominations for different $n$ from 1 to 1000 (or higher, if you wish), and record the *number* of coins each one used to make change for each value of $n$.**

      iii. **For each of your sets of coin denominations, create a plot with $n$ on the horizontal axis. The vertical axis should plot the *competitive ratio*, defined as the ratio between an optimal solution and a greedy solution:**

$$\frac{\#\text{ coins used by the DP algorithm to make change for } n}{\#\text{ coins used by the greedy algorithm to make change for } n}$$

   (d) **Graduate Students: (extra credit for undergraduates)**

      i. Prove that the greedy algorithm is optimal (that is, always finds the minimum number of coins) for the common coin denominations used in the United States $\{1, 5, 10, 25\}$.

      ii. Experiment with your algorithm and try to figure out the denominations that make the greedy algorithm perform the worst. Can you create an input that forces the greedy algorithm to use twice as many coins as the DP algorithm? Can you force it to use 10 times as many coins as the DP algorithm?