CPSC 320 Notes: Memoization and Dynamic Programming, Part 1

March 10, 2018

You want to make change in the world, but to get started, you're just... making change. Assuming an unlimited supply of quarters (25 cents), dimes (10 cents), nickels (5 cents), and pennies (1 cent, once upon a time), you want to make change for $n \geq 0$ cents using the smallest total number of coins.

Here is an optimal greedy algorithm to solve this problem:

While $n \geq 0$:

- If $n \geq 25$, give a quarter and reduce n by 25
- If $n \geq 10$, give a dime and reduce n by 10
- If $n \geq 5$, give a nickel and reduce n by 5
- Otherwise, give a penny and reduce n by 1

Try it on at least one instance.

A few years back, the Canadian government eliminated the penny. Imagine the Canadian government accidentally eliminated the nickel rather than the penny. (That is, assume you have an unlimited supply of quarters, dimes, and pennies, but no nickels.)

1. Adapt your greedy algorithm to this problem and then **challenge your approach** by attempting to find a counterexample to its correctness.

2. We can solve this problem with something like a divide-and-conquer algorithm. (In this case, using a brute-force, recursive approach.)

   (a) To make the change, you must start by handing the customer their first coin. What are your options?

   (b) Imagine that in order to make 81 cents of change using the fewest coins possible, you have to start by handing the customer a quarter. Clearly describe the problem you are left with (but **don't** solve it). It **will** help to give **names** to quantities and concepts in the problem if you haven't already!

   (c) Write down descriptions of the subproblems for each of your other "first coin" options (besides a quarter).

   (d) There are three choices of coin to give first. Given an optimal solution to each subproblem, how will you determine which is the best to choose?

3. It's hard to describe a recursive algorithm without naming it. We'll name the algorithm `CCC(n)` (for `CountCoinsChange(n)`). `CCC(n)` returns the **minimum number of coins** required to make n cents of change using only pennies, dimes, and quarters. (So, it returns just the count of coins, not the coins themselves.) Finish `CCC`'s implementation below:

```
CCC(n):
  If n < 0:

    Return infinity     // that is: "cannot be done"

  Else, If n = 0:

    Return _______________

  Else, n > 0:

    Return the _______________ of these possibilities:

      ----------------------------------------

      ----------------------------------------

      ----------------------------------------
```

4. Finish this recurrence for the runtime of `CCC`:

```
T(n) = 1                                                  for n ________

T(n) = ________________________________________________ otherwise
```

5. Give a disappointing Ω-bound on the runtime of `CCC` by following these steps:

   (a) $T(n)$ is hard to deal with because it has very different-looking recursive terms. To lower-bound it, we can make them all look the same **as long as the resulting function is smaller than the original** (or stays the same size). Now, try to fill in the lower-bound on $T(n)$ below so the recursive terms all match:

   For the recursive case, $T(n) \geq$

   (b) Now, draw a recurrence tree for $T(n)$ and figure out its number of levels, work per level, and total work.

6. Why is the performance so **bad**? (Hint: What subproblem do you get to if you try to give change with five dimes?)