**ITCS 6114/8114 - Algorithms and Data Structures Coin Changes**

**(Dynamic Programming and Greedy Algorithm)**

**1. 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 dynamic programming to make change consisting of quarters, dimes, nickels, and pennies and prove that your algorithm yields an optimal solution. Implement your algorithm and test your solution.**

**Solution:**

**Sample output:**

A screenshot of a social media post

Description automatically generated

The coin-change problem resembles the [0-1 Knapsack Problem](https://www.educative.io/edpresso/what-is-the-knapsack-problem) in [Dynamic Programming](https://www.educative.io/edpresso/what-is-dynamic-programming). It has two versions:

1. Finding the minimum number of coins, of certain denominations, required to make a given sum.
2. Finding the total number of possible ways a given sum can be made from a given set of coins.

To implement Dynamic Programming, it is essential that we divide a problem into subproblems. We can view our problem as if we were to choose the ***better*** option from the following two options:

1. **Selecting the highest possible coin:** The subproblem is about making the amount (Sum - the coin we added) with the same set of coins.
2. **Ignoring the highest possible coin:** In this case, the subproblem is making the same sum with the original set of coins, minus the highest possible coin.

Choosing the ***better*** option in this problem equates to choosing the ***smaller*** of the two options. If the highest coin does not exceed the required sum, then we take the minimum of the two. Otherwise, we choose the second option and ignore the highest coin. Since that coin cannot be used in our solution, we act as if it doesn’t exist.

The key part of a solution, when using DP, is to implement the array to keep track of the solutions to subproblems. This way, we won’t calculate what has already been done, again and again.

Each element of the 2-D array tells us the minimum number of coins required to **make the sum j, considering the first i coins only**.

This procedure obviously runs in O(nk) time. By using Dynamic approach, we can produce a runtime of O(n).

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

**Solution:**

**Sample output:**

**A screenshot of a social media post

Description automatically generated**

1. The reasoning is very similar to that of the dynamic programming question. Let’s consider the first two types of coins in this set, quarters and cents that are worth c cents each, which we will call cˆ. At most, we can use c −1 cents because any number larger than c cents would be replaced by at least one cˆ coin. This operation would reduce the total coin number by c −1. In other words, when the remainder is greater than c and we are allowed to use only cents and cˆ quarters, we would use as many cˆ coins before considering cents.

2. The same thoughts apply for larger coins. Let’s consider 1 ˆ n− c and n cˆ coins. At most, we can use c −1 1 ˆ n− c coins. Because any number larger than n c would be replaced by at least one n cˆ coin. This operation would reduce the total coin number by c −1. In other words, when the remainder is greater than n c and I’m allowed to use only coins that are worth less than n cˆ , I would use as many n cˆ 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 of quarters, dimes, cents.

Thus, we have shown that there is always an optimal solution that includes the greedy choice, and that we can combine the greedy choice with an optimal solution to the remaining sub problem to produce an optimal solution to our original problem. Therefore, the greedy algorithm produces an optimal solution.

For the algorithm that chooses one coin at a time and then recursively solves the sub-problems, the running time is Θ(k), where k is the number of coins used in an optimal solution. Since k ≤ n, the running time is O(n). For our first description of the algorithm, we perform a constant number of calculations (since there are only 4-coin types), and the running time is O (1).

**c. Suppose that the available coins are in the denominations that are powers of c, i.e., the denominations are c0, c1, ...., ck for some integers c > 1 and k >= 0. Show that the greedy algorithm always yields an optimal solution.**

**Solution:**

Let’s consider the first two types of coins in this set, dimes and coins that are worth c cents each, which we will call cˆ. At most, we can use c −1 dime because any number larger than c dimes would be replaced by at least one cˆ coin. This operation would reduce the total coin number by c - 1. In other words, when the remainder is greater than c and we are allowed to use only dimes and cˆ coins, we would use as many cˆ coins before considering dimes. The same thoughts apply for larger coins.

Let’s consider cˆ n−1 and cˆn coins. At most, we can use c−1 \* cˆn-1 coins. Because any number larger than cˆn would be replaced by at least one cˆn coin. This operation would reduce the total coin number by c - 1. In other words, when the remainder is greater than cˆn and we are allowed to use only coins that are worth less than cˆn, We would use as many cˆn coins as possible before considering other coins.

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