CMSC 351: Coin Changing

Justin Wyss-Gallifent

July 4, 2023

1 Introduction ......................................................... 2
2 A Dynamic Programming Approach .............................. 2
  2.1 A Simple Observation ........................................... 2
  2.2 An Algorithmic Idea ........................................... 3
3 An Actual Algorithm .................................................. 4
4 Time Complexity ..................................................... 6
1 Introduction

Consider the following problem:

Suppose all you have are 1-cent, 5-cent and 10-cent coins but you have infinitely many of each. For any given (cent) total \( n \), how many ways are there to obtain \( n \) cents using just these denominations?

For specific values of \( n \) we can brute-force this. For example for \( n = 10 \) we can do:

- Ten 1-cent coins.
- Two 5-cent coins.
- One 10-cent coin.
- One 5-cent coin and five 1-cent coins.

Thus we have a total of 4 ways.

Suppose we wished to write an algorithm which would give the answer for any given \( n \). How might we go about this? Moreover what if we had a different set of coins rather than 1,5,10?

2 A Dynamic Programming Approach

2.1 A Simple Observation

Before creating a general approach (which will lead to an algorithm) consider the following observation:

Suppose we have two coin denominations 2 and 5. If \( n = 12 \) then it’s easy to see that there is \( T = 1 \) way to obtain \( n = 12 \) using only 2-cent coins. How about if we also allow 5-cent coins? Observe that we have a disjoint sum:

\[
\# \text{ ways to get } 12 \text{ using } 2 \text{ and/or } 5 = \# \text{ ways to get } 12 \text{ using } 2 \\
+ \# \text{ ways to get } 12 \text{ using } 2 \text{ and at least one } 5
\]

We know the first summand is 1. For the second summand once we choose to use a single 5-cent coin we have 7 cents left to obtain and we can do that however we wish. Thus:

\[
\# \text{ ways to get } 12 \text{ using } 2 \text{ and/or } 5 = \# \text{ ways to get } 12 \text{ using } 2 \\
+ \# \text{ ways to get } 7 \text{ using } 2 \text{ and/or } 5
\]

Take a moment to see what we have observed here. As a general rule for some \( n \) and two denominations \( c_1 \) and \( c_2 \) we have:
The above observation leads to the following. Suppose we have coin denominations \( [c_1, ..., c_r] \) and wanted to know the number of ways to obtain \( n \) cents using any combinations of these denominations.

Suppose we have an array \( A \) indexed 0 through \( n \) and we have some \( k < n \) such that:

- \( A[0] \), ..., \( A[k] \) contain the number of ways to obtain 0 through \( k \) cents using any denominations from \( c_1 \) through \( c_r \).
- \( A[k+1], ..., A[n] \) contain the number of ways to obtain 0 through \( k \) cents using any denominations from \( c_1 \) through \( c_{r-1} \).

Suppose we wish to update \( A[k+1] \) so that it contains the number of ways to obtain \( k + 1 \) using any denominations from \( c_1 \) through \( c_r \).

From the above rule we immediately see that:

- If \( n \geq c_r \) then we update it as follows:

\[
\]

- Otherwise we leave it alone.
3 An Actual Algorithm

Our algorithmic approach will emerge from this idea. We will start with such an array and pre-load it with the number of ways to achieve each of $0, 1, \ldots, n$ using no coins. Then we will update it with the number of ways to do so using just $c_1$, then we will update it with the number of ways to do so using $c_1$ and/or $c_2$, and so on, until we are done.

On to the algorithm!

Suppose we have coin denominations $C = \{c_1, \ldots, c_r\}$ and we wish to find the number of ways to obtain $n$ cents using any denomination from $c_1$ through $c_r$.

We first assign an array $A$ indexed 0 through $n$ as follows:

$$A = [1, 0, 0, \ldots, 0]$$

In this array, $A[i]$ tells us the number of ways to obtain $i$ through $n$ cents using no coins at all.

We then iterate over the denominations and update accordingly. Here is the pseudocode:

```python
function coincount(C, n):
    A = [1, 0, 0, \ldots, 0]
    for c in C:
        for i = 1 to n:
            if i >= c:
        end for
    end for
end function
```
Example 3.1. Let us walk through this with verb—n=10— and $C = [1,5,10]$. We assign:

$$A = [1,0,0,0,0,0,0,0,0,0]$$

Observe that $A$ now contains the number of ways to obtain 0 through 10 using no coins.

We assign $c = 1$ and pass through $i = 1,2,\ldots,10$ yielding:

$$A = [1,1,1,1,1,1,1,1,1,1]$$

Observe that $A$ now contains the number of ways to obtain 0 through 10 using just 1-cent coins.

We assign $c = 5$ and pass through $i = 1,2,\ldots,10$ yielding:

$$A = [1,1,1,1,1,2,2,2,2,3]$$

Observe that $A$ now contains the number of ways to obtain 0 through 10 using 1- and 5-cent coins.

We assign $c = 10$ and pass through $i = 1,2,\ldots,10$ yielding:

$$A = [1,1,1,1,1,2,2,2,2,4]$$

Observe that $A$ now contains the number of ways to obtain 0 through 10 using 1-, 5-, and 10-cent coins.

Example 3.2. Let us walk through this with $n=15$ and $C = [2,5,7]$. We assign:

$$A = [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]$$

Observe that $A$ now contains the number of ways to obtain 0 through 15 using no coins.

We assign $c = 2$ and pass through $i = 1,2,\ldots,15$ yielding:

$$A = [1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0]$$

Observe that $A$ now contains the number of ways to obtain 0 through 15 using just 2-cent coins.

We assign $c = 5$ and pass through $i = 1,2,\ldots,15$ yielding:

$$A = [1,0,1,0,1,1,1,1,1,1,1,1,1,1,1,1]$$

Observe that $A$ now contains the number of ways to obtain 0 through 15 using 2- and 5-cent coins.

We assign $c = 7$ and pass through $i = 1,2,\ldots,15$ yielding:

$$A = [1,0,1,0,1,1,1,2,1,2,1,2,2,2,3,2]$$

Observe that $A$ now contains the number of ways to obtain 0 through 15 using 2-, 5-, and 7-cent coins.
4 Time Complexity

We have not formally started talking about time complexity but for now it is absolutely worth simply mentioning that the algorithm involves two nested loops such that the innermost body iterates $\text{len}(C) (n + 1)$ times.

Thus intuitively the time required increases linearly as a function of either $n$ or the number of denominations available.