Making Change for Melman: Solution

Problem: You are given a currency system with n different values:

$$C = [c_0, c_1, \ldots, c_{n-1}]$$

and a desired total value total. How many ways are there to make total using these coins?

Example: C = [2, 10, 11], total = 22, then there are 4 combinations:

1. (2+2+2+…+2) (repeated 11 times)
2. 10+(2+2+…+2) (repeated 6 times)
3. 10+10+2
4. 11+11

Solution Structure

Table: Let n be the number of coins. Create a 2-dimensional array

nCombs[n+1][total+1],

where nCombs[i][t] is the number of ways to obtain t using first i coins.

Final result: nCombs[n][total].

Computing nCombs[i][t]: for i = 0, 1, …, n and t = 0, 1, …, total.

For i = 0: There are no coins. The only sum is 0 and one way to do it. Thus

nCombs[0][0] = 1 and **nCombs[0][t] = 0** for t > 0.

For i > 0: Let j be the number of times we use coin c. Clearly $0 \le j \le t/c$. This this leaves t - j·c remaining to be made up by the previous i-1 coins. We have already computed this as nCombs[i-1][t - j·c]. Thus:

```
nCombs[i][t] = nCombs[i-1][t]
             + nCombs[i-1][t - c]
             + nCombs[i-1][t - 2·c] + ...
             + nCombs[i-1][t - m·c], where m =  t/c.
```

We just need to set up loops to compute this table.

Pseudo-code

```
nCombs ← new int[n+1][total+1]
nCombs[0][0] ← 1                            // basis case (no coins)
for (t ← 1 up to total) nCombs[0][t] ← 0
for (i ← 1 up to n) {                       // consider the ith coin
    c ← coins[i-1]                          // current coin value
    for (t ← 0 up to total) {               // compute count for all totals
        sum ← 0
        for (j ← 0 up to t/c) {             // sum up prior combinations
            sum ← sum + nCombs[i-1][t-j·c]
        }
        nCombs[i][t] ← sum                  // store final sum
    }
}
return nCombs[n][total]                     // return final total
```