Etan Soclof

DPMakingChange

Approach
The optimal combination of denominations that makes up an amount of change C can be broken down into a sub problem of the optimal combination of denominations for (C - a particular denomination) + 1. Therefore, we can iterate through available denominations and determine the optimal combination of denominations by finding the minimum of (C - a particular denomination) and adding 1.
We can construct a matrix in which column i represents amounts of change and row j represents using denominations 1…j to make those amounts of change. By iterating column by column, we can determine the optimal combination of denominations that can make up a given amount of change i. These then act as subproblems for amounts in later columns.

Definition of Notation
Given an array of denominations D, if a certain combination of denominations $D_1 \dots D_i$ is the optimal combination of denominations that makes up an amount of change C, denoted as $OPT(D_{1\dots i}, C)$, then there is an optimal combination of denominations $D_1 \dots D_{i+1}$ that makes up that same amount of change C, denoted as $OPT(D_{1\dots i+1}, C)$. There are two scenarios to this affect:

1. The optimal combination of denominations $D_1 \dots D_{i+1}$ is the same as the optimal combination of denominations $D_1 \dots D_i$. In this case, $OPT(D_{1\dots i+1}, C) = OPT(D_{1\dots i}, C)$.
2. The optimal combination of denominations $D_1 \dots D_{i+1}$ is better than the optimal combination of denominations $D_1 \dots D_i$. In such a case, the optimal combination of denominations $D_1 \dots D_{i+1}$ is 1 greater than the optimal combination of denominations that makes up amount $(C-D_{i+1})$. In this case, $OPT(D_{1\dots i+1}, C) = OPT(D_{i+1}, C-D_{i+1}) + 1$.

Given the above, we can define the following recurrence relation:

$$OPT(D_{1\dots i}, C) = \begin{cases} \infty & \text{if } C \text{ or } i \text{ is 0} \\ min \begin{cases} OPT(D_{1\dots i-1},\ C) \\ OPT(D_{1\dots i},\ C - D_j) + 1 \end{cases} & \text{otherwise} \end{cases}$$

where j is incremented by 1 from 1 to i.

Payout Implementation
I took the "hack" approach in which I kept track of the denomination added to $OPT(D_i,\ C - D_i)$ to get $OPT(D_i,\ C)$ (in the case that this was the minimum value) as I constructed the matrix. By the end of the matrix construction, I had an array that held all of the values such that array[C] = i where $OPT(D_i,\ C) = OPT(D_i,\ C - D_i) + 1$. Once this array was constructed, one can determine the combination of denominations that make up $OPT(D_i,\ C)$ by starting at array[C] and recursively traversing backwards through the array to array[C - $D_{array[C]}$] and recording the values of each element.

"Big- O" space and computation requirements
The algorithm constructs a n x N matrix where n is the number of denominations N is the amount for which you have to make change. The calculation for each cell of the matrix is O(1) and therefore the runtime is O(n x N). The space requirement is, of course, n x N as well. There is an additional array used for the Payout calculation that is size N. Thus, the total spaced used is (n x N) + N.