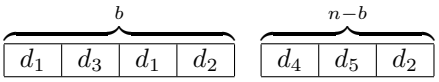
Dynamic Programming Solution to the Coin Changing Problem

(1) Characterize the Structure of an Optimal Solution. The Coin Changing problem exhibits optimal substructure in the following manner. Consider any optimal solution to making change for n cents using coins of denominations $d_1, d_2, \ldots, d_k$. Now consider breaking that solution into two different pieces along any coin boundary. Suppose that the "left-half" of the solution amounts to b cents and the "right-half" of the solution amounts to $n - b$ cents, as shown below.



Claim 1 *The left-half of the solution must be an optimal way to make change for b cents using coins of denominations $d_1, d_2, \ldots, d_k$, and the right-half of the solution must be an optimal way to make change for $n - b$ cents using coins of denominations $d_1, d_2, \ldots, d_k$.*

Proof: By contradiction, suppose that there was a better solution to making change for b cents than the "left-half" of the optimal solution shown. Then the left-half of the optimal solution could be replaced with this better solution, yielding a valid solution to making change for n cents with fewer coins than the solution being considered. But this contradicts the supposed optimality of the given solution, $\rightarrow\leftarrow$. An identical argument applies to the "right-half" of the solution. □

Thus, the optimal solution to the coin changing problem is composed of optimal solutions to smaller subproblems.

(2) Recursively Define the Value of the Optimal Solution. First, we define in English the quantity we shall later define recursively. Let $C[p]$ be the minimum number of coins of denominations $d_1, d_2, \ldots, d_k$ needed to make change for p cents. In the optimal solution to making change for p cents, there must exist some first coin d_i, where $d_i \leq p$. Furthermore, the remaining coins in the optimal solution must themselves be the optimal solution to making change for $p - d_i$ cents, since coin changing exhibits optimal substructure as proven above. Thus, if d_i is the first coin in the optimal solution to making change for p cents, then $C[p] = 1 + C[p - d_i]$; i.e., one d_i coin plus $C[p - d_i]$ coins to optimally make change for $p - d_i$ cents. We don't know which coin d_i is the first coin in the optimal solution to making change for p cents; however, we may check all k such possibilities (subject to the constraint that $d_i \leq p$), and the value of the optimal solution must correspond to the minimum value of $1 + C[p - d_i]$, by definition. Furthermore, when making change for 0 cents, the value of the optimal solution is clearly 0 coins. We thus have the following recurrence.

Claim 2 $C[p] = \begin{cases} 0 & \text{if } p = 0 \\ \min_{i:d_i \leq p}\{1 + C[p - d_i]\} & \text{if } p > 0 \end{cases}$

Proof: The correctness of this recursive definition is embodied in the paragraph which precedes it. □

(3) Compute the Value of the Optimal Solution Bottom-up. Consider the following piece of pseudocode, where d is the array of denomination values, k is the number of denominations, and n is the amount for which change is to be made.

```
CHANGE(d, k, n)
1  C[0] ← 0
2  for p ← 1 to n
3          min ← ∞
4          for i ← 1 to k
5              if d[i] ≤ p then
6                  if 1 + C[p − d[i]] < min then
7                      min ← 1 + C[p − d[i]]
8                      coin ← i
9          C[p] ← min
10         S[p] ← coin
11 return C and S
```

Claim 3 *When the above procedure terminates, for all $0 \leq p \leq n$, $C[p]$ will contain the correct minimum number of coins needed to make change for p cents, and $S[p]$ will contain (the index of) the first coin in an optimal solution to making change for p cents.*

Proof: The correctness of the above procedure is based on the fact that it correctly implements the recursive definition given above. The base case is properly handled in Line 1, and the recursive case is properly handled in Lines 2 to 10, as shown below. Note that since the loop defined in Line 2 goes from 1 to n and since $d_i \geq 1$ for all i, no array element $C[\cdot]$ is accessed in either Line 6 or 7 before it has been computed. Lines 3 to 7 correctly compute $\min_{i:d_i \leq p}\{1 + C[p - d_i]\}$, and $C[p]$ is set to this value in Line 9. Similarly, Lines 3 to 8 correctly compute $\arg\min_{i:d_i \leq p}\{1 + C[p - d_i]\}$, and $S[p]$ is set to this value in Line 10. □

(4) Construct the Optimal Solution from the Computed Information. Consider the following piece of pseudocode, where S is the array computed above, d is the array of denomination values, and n is the amount for which change is to be made.

```
MAKE-CHANGE(S, d, n)
1  while n > 0
2      Print S[n]
3      n ← n − d[S[n]]
```

Claim 4 *The above procedure correctly outputs an optimal set of coins for making change for n cents.*

Proof: By Claim 3, $S[n]$ will contain (the index of) the first coin in an optimal solution to making change for n cents, and this coin in printed in Line 2 during the first pass through the **while** loop. Since our problem exhibits optimal substructure by Claim 1, it must be the case that the solution to the remaining $n - d_{S[n]}$ cents be optimal as well. By setting $n \leftarrow n - d[S[n]]$ in Line 3, the first coin in such a solution will be printed in the next pass through the **while** loop, and so on. (Properly, one would prove that the sequence of coins output by this procedure is correct by induction, but the above suffices as far as I'm concerned.) □

(5) Running Time and Space Requirements. The CHANGE procedure runs in $\Theta(nk)$ due to the nested loops (Lines 2 and 4), and it uses $\Theta(n)$ additional space in the form of the $C[\cdot]$ and $S[\cdot]$ arrays. The MAKE-CHANGE procedure runs in time $O(n)$ since the parameter n is reduced by at least 1 (the minimum coin denomination value) in each pass through the **while** loop. It uses no additional space beyond the inputs given. Thus, the total running time is $\Theta(nk)$ and the total space requirement is $\Theta(n)$.