Tutorial notes for the Change Making Problem

This question concerns a dynamic programming algorithm for the Change Making Problem (CMP). The input to CMP is a sequence of positive integers $c_1, \ldots, c_n, T$, where $c_1, \ldots, c_n$ represent coin denominations and T is a target amount. Assuming an unlimited supply of coins of each denomination, we are to find the minimum number M of coins needed to form T exactly, or output ∞ is no combination of coins of the given denominations sums to T.

For example, if $n =$, the denominations are $c_1 = 5$, $c_2 = 9$ and $c_3 = 13$, and $T = 19$, then the answer is $M = 3$, since three coins suffice ($19 = 5 + 5 + 9$). With the same input denominations, if $T = 6$, then the answer is $M = \infty$.

Solution

To simplify presentation, we will use a two-dimensional array $A(i, t)$, $0 \leq i \leq n$, $0 \leq t \leq T$.

Step 1: $A(i, t)$ is the minimum number of coins needed to form t using only coins $c_1, \ldots, c_i$. If no combination of coins sums to t, then $A(i, t) = \infty$.

Once the array is filled, the value of M is $A(n, T)$.

Step 2: Now we are ready to give the recurrence for filling the array. The initialization is:

$A(0, 0) = 0$ and for all $0 \lneq t \leq T$, $A(0, t) = \infty$. That is, no coins are needed to get $T = 0$, and at least some coins are necessary to obtain any sum $t > 0$.

The body of the recurrence is

$A(i, t) = \min_{0 \leq k \leq \lfloor t/c_i \rfloor} A(i - 1, t - k * c_i) + k$.

That is, for every new denomination c_i under consideration, we try to see if we can make the number of coins to form t smaller by adding some coins of denomination c_i. Then we take the minimum over all multiplicities of c_i that we can add. That is, we see if the number of coins is minimized if we add $0, 1, 2, \ldots$ up to $\lfloor t/c_i \rfloor$ coins. For that, we look at the number of coins needed to form $t - k * c_i$, and add to it the number of coins k of denomination c_i. Clearly, we cannot add more copies of c_i than "fits" into t; this is why the maximal number of c_i is limited by $\lfloor t/c_i \rfloor$.

Note that this recurrence takes care of the special case $c_i > t$: if $c_i > t$ then the only possible value for k is 0, which amounts to choosing $A(i, t) = A(i - 1, t)$. Also, for $k = 0$ it considers the number of coins necessary to make t without any coins of denomination c_i.

Example: Let $n = 3$, $c_1 = 2$, $c_2 = 3$ and $c_3 = 5$ with $T = 7$. Then the array becomes:

$i \backslash t$	0	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞	∞
1	0	∞	1	∞	2	∞	3	∞
2	0	∞	1	1	2	2	2	3
3	0	∞	1	1	2	1	2	2

Step 3: The following program fills in the array $B[i, t]$, which corresponds to $A(i, t)$ in the recurrence. It has a different name to make it easier to prove that it contains the same values. Assume that the array $C[i]$ contains denomination of i^{th} coin, Also, assume that we have a constant $INF > T$ to represent ∞.

$B[0,0] \leftarrow 0$
For every $t \in \{1, \ldots, T\}$
    $B[0,t] \leftarrow \infty$
end for
For i from 1 to n
    for every $t \in \{0, \ldots, T\}$
        $B[i,t] \leftarrow INF$
        for k from 0 to $floor(t/C[i])$
            if $B[i,t] > B[i-1, t-k*C[i]] + k$ then
                $B[i,t] \leftarrow B[i-1, t-k*C[i]] + k$
            end if
        end for
    end for
end for
Output $B[n,T]$

Step 4: In order to reconstruct the solution, we go backwards through our array and check how many coins of each denomination we used.