Dynamic Programming: the Coin Changing problem

Suppose we need to make change for 67£. We want to do this using the fewest number of coins possible among: 1, 5, 10, 25.

If the following coins are available: 1, 5, 10, 25. It is easy to see the optimal solution 67=2*25+10+5+2*1. By repeatedly choosing the largest coin less than or equal to the remaining sum, the desired sum is obtained by a greedy algorithm.

The formal description of the Coin Changing problem is as follows: Let D = {d₁, ..., dₖ} be a finite set of distinct coin denominations. We assume each dᵢ is an integer and d₁ > d₂ > ... > dₖ = 1. Each denomination is available in unlimited quantity. The problem is to make change for n£ using a minimum total number of coins, dₖ=1 so there is always a solution.

The greedy method does not work in any case (it does not give an optimal solution). For example, D = {25,10,1} and n = 30. The greedy method would produce a solution: 25+5*1, which is not as good as 3*10.

Step 1: Characterize the sub-structure. Define C[j] to be the minimum number of coins we need to make change for j£. If we knew that an optimal solution for the problem of making change for j£ used a coin of denomination dᵢ we would have C[j] = 1 + C[j − dᵢ].

Step 2: Define the value of an optimal solution. We can recursively define the value of an optimal solution from the equation found in step 1.

\[
C[j] = \min \begin{cases} 
\infty & \text{if } j < 0, \\
0 & \text{if } j = 0 \\
1 + \min_{1 ≤ i ≤ k} \{C[j − dᵢ]\} & \text{if } j ≥ 1
\end{cases}
\]

Step 3: Algorithm.

Coin-Changing(n,d,k)
C[0]=0;
For j from 1 to n
   C[j]=inf;
For i from 1 to k
   If j ≥ dᵢ and 1 + C[j − dᵢ] < C[j] then
      C[j] = 1 + C[j − dᵢ].
      Denom[j] = dᵢ

Return C
We use an additional array \( \text{Denom} \), where \( \text{Denom}[j] \) is the denomination of a coin used in an optimal solution to the problem of making change for \( j \). If the following coins are available: 1, 5, 10, 25:

<table>
<thead>
<tr>
<th>( j )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C[j] )</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>( \text{Denom}[j] )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

Given some \( n \), find all the combinations of coins that make up the value, the following coins are available: 1, 5, 10, 25. For example, for \( N = 4 \), \( D = \{1,2,3\} \), there are four solutions: \{1,1,1,1\}, \{1,1,2\}, \{2,2\}, \{1,3\}.

**Step 1:** The set of solutions for this problem, \( C(N, m) \), can be partitioned into two sets:

1. There are those sets that do not contain any \( d_m \)
2. Those sets that contain at least 1 \( d_m \)

If a solution does not contain \( d_m \), then we can solve the subproblem of \( N \) with \( D = \{d_1, d_2, ..., d_{m-1}\} \), or the solutions of \( C(N, m - 1) \).

If a solution does contain \( d_m \), then we are using at least one \( d_m \), thus we are now solving the subproblem of \( N - d_m \), with \( D = \{d_1, d_2, ..., d_m\} \). This is \( C(N - d_m, m) \).

**Step 2:** Thus, we can formulate the following: \( C(N-m) = C(N, m - 1) + C(N - d_m, m) \) with the base cases:

1. \( C(N, m) = 1, N = 0 \),
2. \( C(N, m) = 0, N < 0 \) (negative sum of money),
3. \( C(N, m) = 0, N \geq 1, m \leq 0 \) (no change available).

**Step 3:** The algorithm is as follows:

```plaintext
count(n, m)
    for i from 0 to n
        for j from 0 to m
            if i equals 0
                table[i, j] = 1
            else if j equals 0
                table[i, j] = 1 if i%S[j] equals 0 else 0
            else if d[j] greater than i
                table[i, j] = table[i, j - 1]
            else
                table[i, j] = table[i - d[j], j] + table[i, j-1]
return table[n, m]
```

2
<table>
<thead>
<tr>
<th>n/d</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>