**Coin Change Problem**

Problem

Given a set of coins and a value V. Find the number of ways in which we can make change of V.

Example



V = 3

Possible ways to make change are {3}, {2,1}, {1,1,1}.

Note: {1,2} is not counted as a separate way as it is same as {2,1}.

To make ways with every coin, we have 2 options

1. Take it
2. Do not take it

Recurrence relation

*cnt(S[], m, V) = cnt(S[], m, V-Sm) + cnt(S[], m-1, V)*

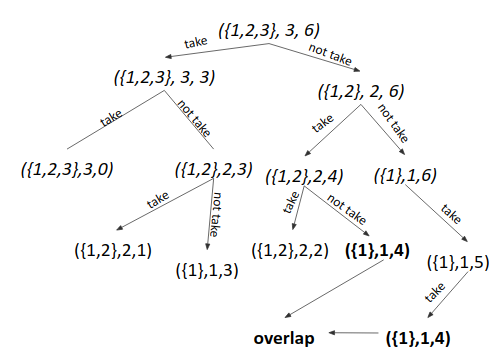
Since it can be represented as a Recurrence relation, hence it has Optimal Substructure Property.

To see overlapping subproblem property,

Let us take an example



Making recursion tree



We can see ({1},1,4) has repeated. Hence it also has Overlapping Subproblem Property.

Since it follows both optimal substructure property and overlapping subproblem property, hence it can be solved using Dynamic Programming.

Approach 1 (Using Memoization)

1. Write the recursive solution.
2. Memoize it.

Approach 2 (Tabulation - Bottom Up)

1. Take each coin one by one and fill the dp table till that coin, for all the values from 0 to V.

Example



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Coin/Value | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| # | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 2 | 2 | 3 | 3 | 4 |
| 3 | 1 | 1 | 2 | 3 | 4 | 5 | 7 |

1. For every cell, we have 2 options
   1. Take that coin *(dp[i][j-s[i-1])*
   2. Do not take that coin *(dp[i-1][j])*

Time Complexity: O(V\*n)

Space Complexity: O(V\*n).

Approach 3 (Tabulation with space efficiency)

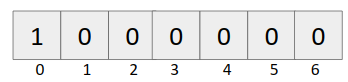
1. Just a minor change in approach 2.
2. We knew for every cell, we have 2 options
   1. Take that coin
   2. Do not take that coin. (We do not take extra row. Update on the same cell).

Time Complexity: O(V\*m)

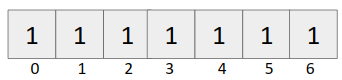
Space Complexity: O(n).

Dry Run

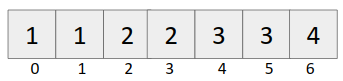
When no coin was taken



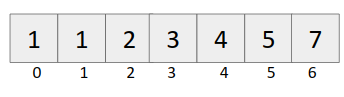
When {1} was taken



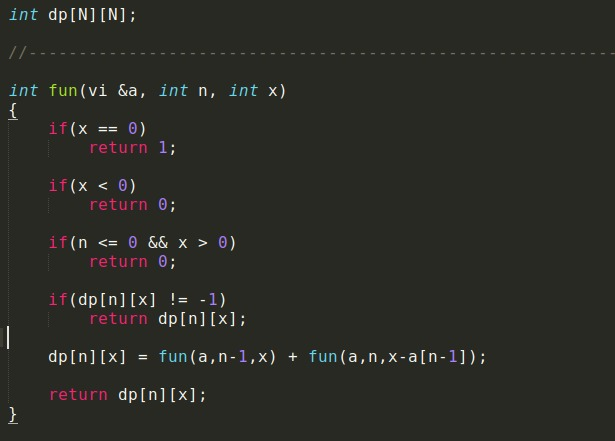
When {1,2} was taken

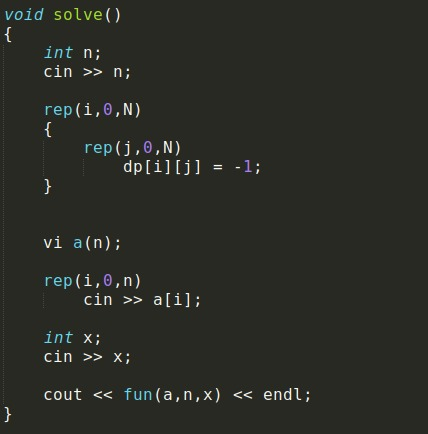


When {1,2,3} was taken

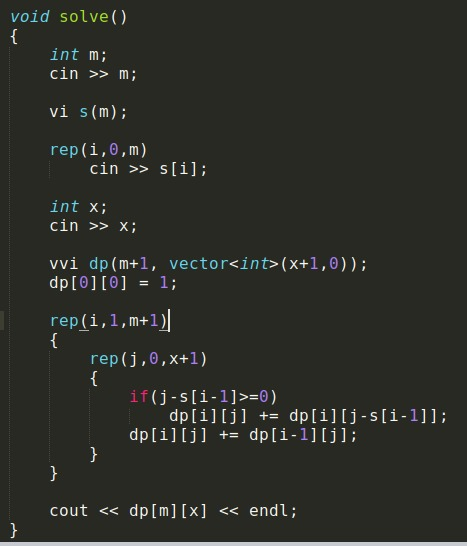


Code (Memoization)





Code (Iterative)



Code (Space optimization)

