Coin Change (Total number of ways to get the change)

Given an unlimited number of coins of given denominations, find the total number of ways to get the desired change. Note : Order of coins doesn't matter.

Examples

v = {1, 2, 3} M = 4

{1, 1, 1, 1} {1, 1, 2} {1, 3} {2, 2}

v = {2, 5, 3, 6} M = 10

{2, 2, 2, 2, 2} {2, 3, 5} {2, 2, 3, 3} {2, 2, 6} {5, 5}

v= {1, 3, 5, 7} M = 8

{1, 1, 1, 1, 1, 1, 1, 1} {1, 1, 1, 1, 1, 3} {1, 1, 1, 5} {1, 7} {1, 1, 3, 3} {3, 5}

v = {1, 2, 3} M = 4

To get to the solution to the actual problem (given unlimited coins of denominations 1$, 2$ and 3$, what is the total number of ways we can make change for 4$), we will first solve the subproblems.

e.g.

• Given unlimited coins of only 1$, what is the total number of ways we can make change for 3$? (answer is 1 {1, 1, 1})

• Given unlimited coins of only 1$ and 2$, what is the total number of ways we can make change for 1$? (answer is 1 {1})

• Given unlimited coins of only 1$ and 2$, what is the total number of ways we can make change for 3$? (answer is 2 {1, 1, 1} {1, 2})

• Given unlimited coins of 1$, 2$ and 3$, what is the total number of ways we can make change for 3$? (answer is 3 {1, 1, 1} {1, 2} {3})

Coins/Amount	0	1	2	3	4
0					
1					
2					
3					

When we don't consider any coin, the total number of ways to make change for amount 0$ will be 1.

Coins/Amount	0	1	2	3	4
0	1				
1					
2					
3					

When we don't consider any coin, the total number of ways to make change for an amount greater than 0$ will be 0.

Coins/Amount	0	1	2	3	4
0	1	0	0	0	0
1					
2					
3					

No matter how many coins we consider the total number of ways to make change for amount 0$ will be 1

Coins/Amount	0	1	2	3	4
0	1	0	0	0	0
1	1				
2	1				
3	1				

Considering only the first coin (coin of 1$) **(v = {1, 2, 3} , M = 4)**

Coins/Amount	0	1	2	3	4
0	1	0	0	0	0
1	1	1	1	1	1
2	1				
3	1				

Considering first two coins (coins of 1$ and 2$) **(v = {1, 2, 3} , M = 4)**

Coins/Amount	0	1	2	3	4
0	1	0	0	0	0
1	1	1	1	1	1
2	1	1	2	2	3
3	1				

Considering all three coins (coins of 1$, 2$, 3$) **(v = {1, 2, 3} , M = 4)**

Coins/Amount	0	1	2	3	4
0	1	0	0	0	0
1	1	1	1	1	1
2	1	1	2	2	3
3	1	1	2	3	4

When we can't include the i^{th} coin for making the change j , because value of i^{th} coin is greater than j

$S[i][j] = S[i - 1] [j]$

When we can include the i^{th} coin for making the change j , because value of i^{th} coin is not greater than j

$S[i][j] = S[i - 1] [j] + S[i] [j - \text{value of the } i^{th} \text{ coin}]$

and we know that value of i^{th} coin is v[i -1]

$S[i][j] = S[i - 1] [j] + S[i] [j - v[i-1]]$

Considering all three coins (coins of 1$, 2$, 3$) **(v = {1, 2, 3} , M = 4)**

Coins/Amount	0	1	2	3	4
0	**1**	**0**	**0**	**0**	**0**
1	**1**	1	1	1	1
2	**1**	1	2	2	3
3	**1**	1	2	3	4

S[2][1] = Since value of 2^{nd} coin is greater than the change we want to make (1$), S[2][1] = S[1][1] = 1

S[2][2] = Sum (S[1][2] , S[2][2-2]) = Sum (S[1][2] , S[2][0]) = Sum (1, 1) = 2

S[2][3] = Sum (S[1][3] , S[2][3-2]) = Sum (S[1][3] , S[2][1]) = Sum (1, 1) = 2

S[2][4] = Sum (S[1][4] , S[2][4-2]) = Sum (S[1][4] , S[2][2]) = Sum (1, 2) = 3

S[3][1] = Since value of 3^{rd} coin is greater than the change we want to make (1$), S[3][1] = S[2][1] = 1

S[3][2] = Since value of 3^{rd} coin is greater than the change we want to make (2$), S[3][2] = S[2][2] = 2

S[3][3] = Sum (S[2][3] , S[3][3-3]) = Sum (S[2][3] , S[3][0]) = Sum (2, 1) = 3

S[3][4] = Sum (S[2][4] , S[3][4-3]) = Sum (S[2][4] , S[3][1]) = Sum (3, 1) = 4

Implementation

```
public class App {
   public static void main(String[] args) {
       int[] v = { 1, 2, 3 };
       int M = 4;
       int totalWays = CoinChange.coinChange(v, M);
       System.out.println("Total number of ways to make change : " + totalWays);
   }
}
```

```
public class CoinChange {

  public static int coinChange(int[] v, int M) {

       int[][] dpTable = new int[v.length + 1][M + 1];

       for (int i = 0; i <= v.length; i++) {
                   dpTable[i][0] = 1;
       }
       for (int j = 1; j <= M; j++) {
                   dpTable[0][j] = 0;
       }

       for (int i = 1; i <= v.length; i++) {
           for (int j = 1; j <= M; j++) {
               if (v[i - 1] > j) {
                   dpTable[i][j] = dpTable[i - 1][j];
               } else {
                   dpTable[i][j] = dpTable[i - 1][j] + dpTable[i][j - v[i - 1]];
               }
           }
       }
       return dpTable[v.length][M];
   }
}
```

Runtime Complexity

O(v * M)