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

S			
	1	2	3

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

A. Take it
B. Do not take it

Recurrence relation

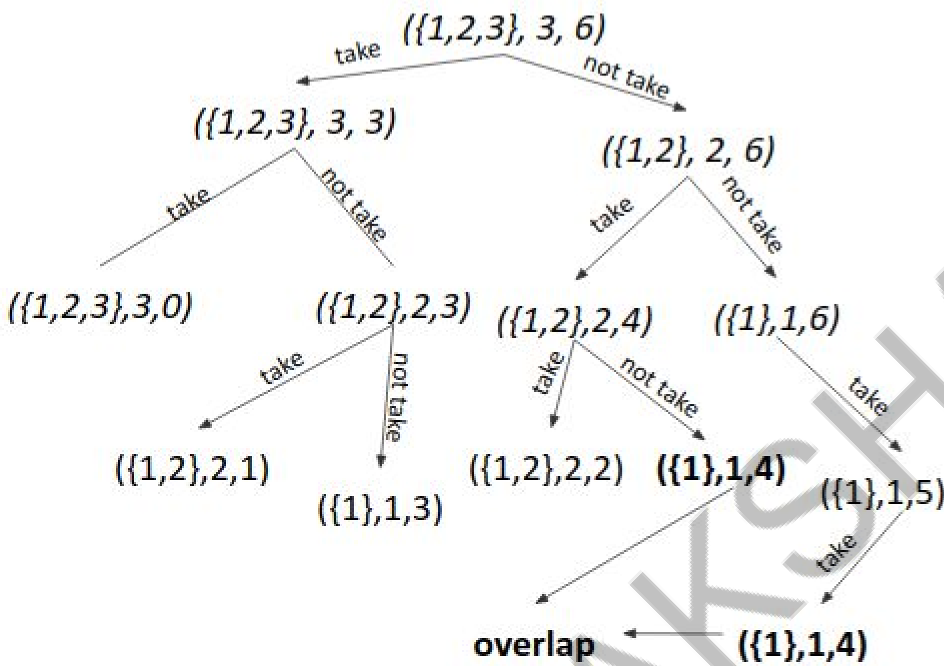
cnt(S[], m, V) = cnt(S[], m, V-Sm) + cnt(S[], m-1, V)

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

S				
	1	2	3	V = 6

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

S | 1 | 2 | 3 | V = 6

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

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

Time Complexity: O(V*n)

Space Complexity: O(V*n).

<u>Approach 3 (Tabulation with space efficiency)</u>

1. Just a minor change in approach 2.
2. We knew for every cell, we have 2 options
    a. Take that coin
    b. 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

1	0	0	0	0	0	0
0	1	2	3	4	5	6

When {1} was taken

1	1	1	1	1	1	1
0	1	2	3	4	5	6

When {1,2} was taken

1	1	2	2	3	3	4
0	1	2	3	4	5	6

When {1,2,3} was taken

1	1	2	3	4	5	7
0	1	2	3	4	5	6

Code (Memoization)

```
int dp[N][N];

//-----------------------------------------------------------

int fun(vi &a, int n, int x)
{
    if(x == 0)
        return 1;

    if(x < 0)
        return 0;

    if(n <= 0 && x > 0)
        return 0;

    if(dp[n][x] != -1)
        return dp[n][x];

    dp[n][x] = fun(a,n-1,x) + fun(a,n,x-a[n-1]);

    return dp[n][x];
}
```

```
void solve()
{
    int n;
    cin >> n;

    rep(i,0,N)
    {
        rep(j,0,N)
            dp[i][j] = -1;
    }


    vi a(n);

    rep(i,0,n)
        cin >> a[i];

    int x;
    cin >> x;

    cout << fun(a,n,x) << endl;
}
```

Code (Iterative)

```
void solve()
{
    int m;
    cin >> m;

    vi s(m);

    rep(i,0,m)
        cin >> s[i];

    int x;
    cin >> x;

    vvi dp(m+1, vector<int>(x+1,0));
    dp[0][0] = 1;

    rep(i,1,m+1)
    {
        rep(j,0,x+1)
        {
            if(j-s[i-1]>=0)
                dp[i][j] += dp[i][j-s[i-1]];
            dp[i][j] += dp[i-1][j];
        }
    }

    cout << dp[m][x] << endl;
}
```

Code (Space optimization)

```
void solve()
{
    int m;
    cin >> m;

    vi s(m);

    rep(i,0,m)
        cin >> s[i];

    int x;
    cin >> x;

    vi dp(x+1, 0);

    dp[0] = 1;

    rep(i,0,m)
    {
        rep(j,0,x+1)
        {
            if(j-s[i] >= 0)
                dp[j] += dp[j-s[i]];
        }
    }

    cout << dp[x] << endl;
```