

GeeksforGeeks

A computer science portal for geeks

Log in

- [Home](#)
- [Q&A](#)
- [Interview Corner](#)
- [Ask a question](#)

- [Contribute](#)
- [GATE](#)
- [Algorithms](#)
- [C](#)
- [C++](#)
- [Books](#)
- [About us](#)

[Arrays](#)

[Bit Magic](#)

[C/C++ Puzzles](#)

[Articles](#)

[GFacts](#)

[Linked Lists](#)

[MCQ](#)

[Misc](#)

[Output](#)

[Strings](#)

[Trees](#)

Dynamic Programming | Set 7 (Coin Change)

January 29, 2012

Given a value N, if we want to make change for N cents, and we have infinite supply of each of $S = \{ S_1, S_2, \dots, S_m \}$ valued coins, how many ways can we make the change? The order of coins doesn't matter.

For example, for $N = 4$ and $S = \{1, 2, 3\}$, there are four solutions: $\{1, 1, 1, 1\}, \{1, 1, 2\}, \{2, 2\}, \{1, 3\}$. So output should be 4. For $N = 10$ and $S = \{2, 5, 3, 6\}$, there are five solutions: $\{2, 2, 2, 2, 2\}, \{2, 2, 3, 3\}, \{2, 2, 6\}, \{2, 3, 5\}$ and $\{5, 5\}$. So the output should be 5.

1) Optimal Substructure

To count total number solutions, we can divide all set solutions in two sets.

1) Solutions that do not contain m th coin (or S_m).

2) Solutions that contain at least one S_m .

Let $\text{count}(S[], m, n)$ be the function to count the number of solutions, then it can be written as sum of $\text{count}(S[], m-1, n)$ and $\text{count}(S[], m, n-S_m)$.

Therefore, the problem has optimal substructure property as the problem can be solved using solutions to subproblems.

2) Overlapping Subproblems

Following is a simple recursive implementation of the Coin Change problem. The implementation simply follows the recursive structure mentioned above.

```
#include<stdio.h>

// Returns the count of ways we can sum S[0...m-1] coins to get sum n
int count( int S[], int m, int n )
{
    // If n is 0 then there is 1 solution (do not include any coin)
    if (n == 0)
        return 1;

    // If n is less than 0 then no solution exists
    if (n < 0)
        return 0;

    // If there are no coins and n is greater than 0, then no solution ex
    if (m <= 0 && n >= 1)
        return 0;

    // count is sum of solutions (i) including S[m-1] (ii) excluding S[m-
    return count( S, m - 1, n ) + count( S, m, n-S[m-1] );
}

// Driver program to test above function
int main()
{
    int i, j;
    int arr[] = {1, 2, 3};
    int m = sizeof(arr)/sizeof(arr[0]);
    printf("%d ", count(arr, m, 4));
    getchar();
    return 0;
}
```

It should be noted that the above function computes the same subproblems again and again. See the following recursion tree for $S = \{1, 2, 3\}$ and $n = 5$.

The function $C(\{1,3\}, 1)$ is called two times. If we draw the complete tree, then we can see that there are many subproblems being called more than once.

```
C() --> count()
                        C({1,2,3}, 5)
                       /      \
```

Since same subproblems are called again, this problem has Overlapping Subproblems property. So the Coin Change problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical [Dynamic Programming\(DP\) problems](#), recomputations of same subproblems can be avoided by constructing a temporary array table[][] in bottom up manner.

Dynamic Programming Solution

```
#include<stdio.h>

int count( int S[], int m, int n )
{
    int i, j, x, y;

    // We need n+1 rows as the table is consturcted in bottom up manner u
    // the base case 0 value case (n = 0)
    int table[n+1][m];

    // Fill the enteries for 0 value case (n = 0)
    for (i=0; i<m; i++)
        table[0][i] = 1;

    // Fill rest of the table enteries in bottom up manner
    for (i = 1; i < n+1; i++)
    {
        for (j = 0; j < m; j++)
        {
            // Count of solutions including S[j]
            x = (i-S[j] >= 0)? table[i - S[j]][j]: 0;

            // Count of solutions excluding S[j]
            y = (j >= 1)? table[i][j-1]: 0;

            // total count
            table[i][j] = x + y;
        }
    }
    return table[n][m-1];
}
```

```
// Driver program to test above function
int main()
{
    int arr[] = {1, 2, 3};
    int m = sizeof(arr)/sizeof(arr[0]);
    int n = 4;
    printf(" %d ", count(arr, m, n));
    return 0;
}
```

Time Complexity: $O(mn)$

Following is a simplified version of method 2. The auxiliary space required here is $O(n)$ only.

```
int count( int S[], int m, int n )
{
    // table[i] will be storing the number of solutions for
    // value i. We need n+1 rows as the table is constructed
    // in bottom up manner using the base case (n = 0)
    int table[n+1];

    // Initialize all table values as 0
    memset(table, 0, sizeof(table));

    // Base case (If given value is 0)
    table[0] = 1;

    // Pick all coins one by one and update the table[] values
    // after the index greater than or equal to the value of the
    // picked coin
    for(int i=0; i<m; i++)
        for(int j=S[i]; j<=n; j++)
            table[j] += table[j-S[i]];

    return table[n];
}
```

Thanks to [Rohan Laishram](#) for suggesting this space optimized version.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

References:

http://www.algorithmist.com/index.php/Coin_Change

Like 13 7 Tweet 0 Share



IIT-JEE Coaching
at Rs. **0/-**



FREE Registration and access

Video Lectures

Digitized Notes

Practice Question

Register Now

You may also like following posts

1. [Dynamic Programming | Set 1 \(Overlapping Subproblems Property\)](#)
2. [Dynamic Programming | Set 3 \(Longest Increasing Subsequence\)](#)
3. [Dynamic Programming | Set 4 \(Longest Common Subsequence\)](#)
4. [Dynamic Programming | Set 5 \(Edit Distance\)](#)
5. [Dynamic Programming | Set 6 \(Min Cost Path\)](#)

Writing code in comment? Please refer [here](#) for guidelines.

0 comments

★ 0



Newest ▾ Community

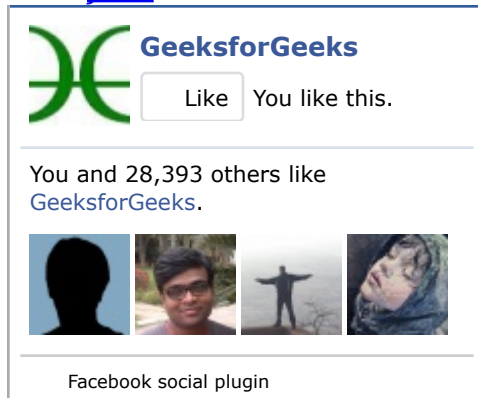
Share  

No one has commented yet.

 Comment feed  Subscribe via email

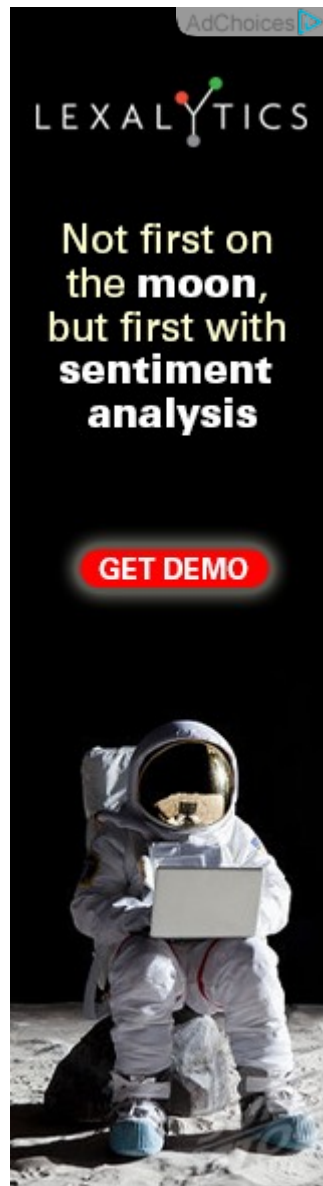
-
- - - [Interview Experiences](#)
 - [Advanced Data Structures](#)

- [Dynamic Programming](#)
- [Greedy Algorithms](#)
- [Backtracking](#)
- [Pattern Searching](#)
- [Divide & Conquer](#)
- [Graph](#)
- [Mathematical Algorithms](#)
- [Recursion](#)
- [Java](#)



• Popular Posts

- [All permutations of a given string](#)
- [Memory Layout of C Programs](#)
- [Understanding “extern” keyword in C](#)
- [Median of two sorted arrays](#)
- [Tree traversal without recursion and without stack!](#)
- [Structure Member Alignment, Padding and Data Packing](#)
- [Intersection point of two Linked Lists](#)
- [Lowest Common Ancestor in a BST.](#)
- [Check if a binary tree is BST or not](#)
- [Sorted Linked List to Balanced BST](#)



Follow @Geeksforgeeks

1,568 followers

509



[Subscribe](#)

Recent Comments

- Chandu on [Merge a linked list into another linked list at alternate positions](#)
- Sudipto on [Longest Monotonically Increasing Subsequence Size \(N log N\)](#)
- GeeksforGeeks on [Find depth of the deepest odd level leaf node](#)
- headsOn on [Morgan Stanley Interview | Set 3](#)
- headsOn on [Morgan Stanley Interview | Set 3](#)
- Coder001 on [Find depth of the deepest odd level leaf node](#)
- Ravish on [Find the repeating and the missing | Added 3 new methods](#)
- vedverma1 on [Detect and Remove Loop in a Linked List](#)

AdChoices ▶

▶ [Java Program](#)

▶ [Coin Set](#)

• ▶ [Coin Collection](#)

AdChoices ▶

▶ [Java Porgramming](#)

▶ [Array](#)

▶ [C String](#)

AdChoices ▶

▶ [Function](#)

▶ [C C++ Java](#)

▶ [Coin Values](#)

@geeksforgeeks, [Some rights reserved](#) [Contact Us](#)

Powered by [WordPress](#) & [MooTools](#), customized by geeksforgeeks team