Dynamic Programming | Set 7 (Coin Change)

Given a value N, if we want to make change for N cents, and we have infinite supply of each of S = { S1, S2, .. , Sm} 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 mth coin (or Sm).
2) Solutions that contain at least one Sm.
Let count(S[], m, n) be the function to count the number of solutions, then it can be written as sum of count(S[], m-1, n) and count(S[], m, n-Sm).

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)
```

Google™ Custom Search

GeeksforGeeks
53,524 people like GeeksforGeeks.

Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion

```
    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
    if (m <=0 && n >= 1)
        return 0;

    // count is sum of solutions (i) including S[m-1] (ii) excluding S
    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) 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)
                           /                \
                         /                   \
             C({1,2,3}, 2)                 C({1,2}, 5)
            /     \                        /         \
           /        \                     /           \
C({1,2,3}, -1)  C({1,2}, 2)        C({1,2}, 3)    C({1}, 5)
               /     \            /    \            /     \
             /        \          /      \          /       \
    C({1,2},0)  C({1},2)   C({1,2},1) C({1},3)    C({1}, 4)  C({}, 5)
```

Geometric Algorithms

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

```
                 / \      / \       / \        /     \
                /   \    /   \     /   \      /       \
               .     .  .     .   .     .   C({1}, 3) C({}, 4)
                                             /  \
                                            /    \
                                           .      .
```

Since same suproblems are called again, this problem has Overlapping Subprolems 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 manne
    // 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];
```

You
make it sweet to be yourself
/RosannaPansino + 1,500,431 Fans

```
}

// 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 consturcted
    // 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:

Recent Comments

Aman Hi, Why arent we checking for conditions...

Write a C program to Delete a Tree. · 38 minutes ago

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 42 minutes ago

Sanjay Agarwal bool tree::Root_to_leaf_path_given_sum(tree...

Root to leaf path sum equal to a given number · 1 hour ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...

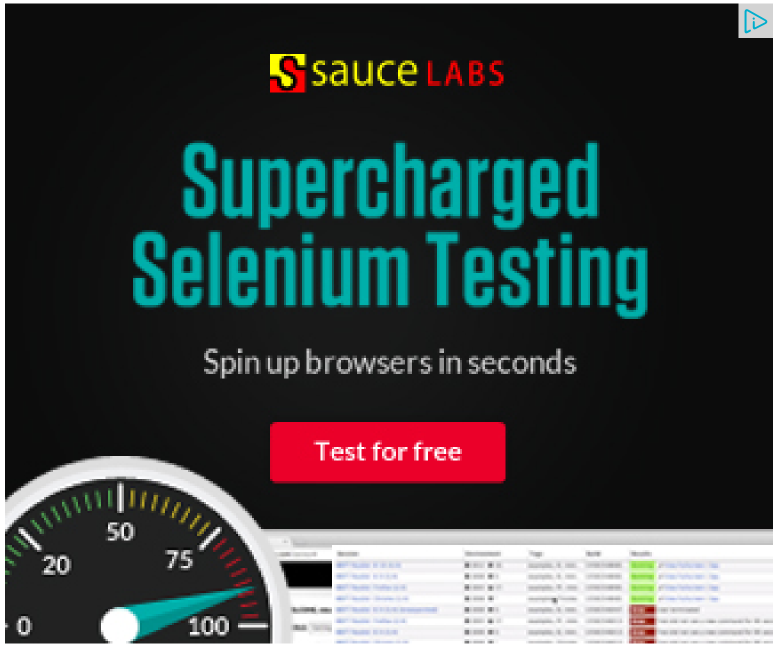
Count trailing zeroes in factorial of a number · 1 hour ago

newCoder3006 If the array contains negative numbers also. We...

Find subarray with given sum · 1 hour ago

newCoder3006 Code without using while loop. We can do it...

Find subarray with given sum · 1 hour ago http://www.algorithmist.com/index.php/Coin_Change



Related Tpoics:

- Backtracking | Set 8 (Solving Cryptarithmetic Puzzles)
- Tail Recursion
- Find if two rectangles overlap
- Analysis of Algorithm | Set 4 (Solving Recurrences)
- Print all possible paths from top left to bottom right of a mXn matrix
- Generate all unique partitions of an integer
- Russian Peasant Multiplication
- Closest Pair of Points | O(nlogn) Implementation

32 Tweet 2 8

Writing code in comment? Please use **ideone.com** and share the link here.