Dynamic Programming | Set 25 (Subset Sum Problem)

December 24, 2012

Given a set of non-negative integers, and a value *sum*, determine if there is a subset of the given set with sum equal to given *sum*.

```
Examples: set[] = {3, 34, 4, 12, 5, 2}, sum = 9
Output:  True  //There is a subset (4, 5) with sum 9.
```

Let isSubSetSum(int set[], int n, int sum) be the function to find whether there is a subset of set[] with sum equal to *sum*. n is the number of elements in set[].

The isSubsetSum problem can be divided into two subproblems

…a) Include the last element, recur for n = n-1, sum = sum – set[n-1]
…b) Exclude the last element, recur for n = n-1.
If any of the above the above subproblems return true, then return true.

Following is the recursive formula for isSubsetSum() problem.

```
isSubsetSum(set, n, sum) = isSubsetSum(set, n-1, sum) ||
                           isSubsetSum(arr, n-1, sum-set[n-1])
Base Cases:
isSubsetSum(set, n, sum) = false, if sum > 0 and n == 0
isSubsetSum(set, n, sum) = true, if sum == 0
```

Following is naive recursive implementation that simply follows the recursive structure mentioned above.

```
// A recursive solution for subset sum problem
#include <stdio.h>

// Returns true if there is a subset of set[] with sun equal to given sum
bool isSubsetSum(int set[], int n, int sum)
{
   // Base Cases
   if (sum == 0)
     return true;
   if (n == 0 && sum != 0)
     return false;

   // If last element is greater than sum, then ignore it
   if (set[n-1] > sum)
     return isSubsetSum(set, n-1, sum);

   /* else, check if sum can be obtained by any of the following
      (a) including the last element
      (b) excluding the last element   */
   return isSubsetSum(set, n-1, sum) || isSubsetSum(set, n-1, sum-set[n-1
}

// Driver program to test above function
int main()
{
  int set[] = {3, 34, 4, 12, 5, 2};
  int sum = 9;
  int n = sizeof(set)/sizeof(set[0]);
  if (isSubsetSum(set, n, sum) == true)
     printf("Found a subset with given sum");
  else
     printf("No subset with given sum");
  return 0;
}
```

Output:

```
Found a subset with given sum
```

The above solution may try all subsets of given set in worst case. Therefore time complexity of the above solution is exponential. The problem is in-fact NP-Complete (There is no known polynomial time solution for this problem).

We can solve the problem in Pseudo-polynomial time using Dynamic programming. We create a boolean 2D table subset[][] and fill it in bottom up manner. The value of subset[i][j] will be true if there is a subset of set[0..j-1] with sum equal to i., otherwise false. Finally, we return subset[sum][n]

```
// A Dynamic Programming solution for subset sum problem
#include <stdio.h>

// Returns true if there is a subset of set[] with sun equal to given sum
bool isSubsetSum(int set[], int n, int sum)
{
    // The value of subset[i][j] will be true if there is a subset of set
    //  with sum equal to i
    bool subset[sum+1][n+1];

    // If sum is 0, then answer is true
    for (int i = 0; i <= n; i++)
      subset[0][i] = true;

    // If sum is not 0 and set is empty, then answer is false
    for (int i = 1; i <= sum; i++)
      subset[i][0] = false;

     // Fill the subset table in botton up manner
     for (int i = 1; i <= sum; i++)
     {
       for (int j = 1; j <= n; j++)
       {
         subset[i][j] = subset[i][j-1];
         if (i >= set[j-1])
           subset[i][j] = subset[i][j] || subset[i - set[j-1]][j-1];
       }
     }

    /* // uncomment this code to print table
     for (int i = 0; i <= sum; i++)
     {
       for (int j = 0; j <= n; j++)
          printf ("%4d", subset[i][j]);
       printf("\n");
     } */

     return subset[sum][n];
}

// Driver program to test above function
int main()
{
  int set[] = {3, 34, 4, 12, 5, 2};
  int sum = 9;
  int n = sizeof(set)/sizeof(set[0]);
  if (isSubsetSum(set, n, sum) == true)
     printf("Found a subset with given sum");
  else
     printf("No subset with given sum");
  return 0;
}
```

Output:

```
Found a subset with given sum
```

Time complexity of the above solution is O(sum*n).

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

Like 14 0 Tweet 4 Share

You may also like following posts

1. Dynamic Programming | Set 3 (Longest Increasing Subsequence)
2. Dynamic Programming | Set 7 (Coin Change)
3. Dynamic Programming | Set 10 (0-1 Knapsack Problem)
4. Dynamic Programming | Set 13 (Cutting a Rod)
5. Dynamic Programming | Set 18 (Partition problem)

Writing code in comment? Please refer here for guidelines.

29 comments

★ 0

Leave a message...

Newest Community Share ⚙

Vinodhini • a month ago

Can we extend this DP logic to
1) print all the subsets of sum X
2) Find the number of subsets of sum X

If anyone could write a post on it, it would be very helpful

Thanks in advance

Reply Share ›

Vinodhini • a month ago

Can we extend this DP logic to
1) print the subsets of sum X
2) Find the number of subsets of sum X

If anyone could write a post on it, it would be very helpful

Thanks in advance

Reply Share ›

Born Actor • a month ago

```
 #include <iostream>
#include <stdio.h>
#include <stdlib.h>
using namespace std;
int a[50];
int n;
int sum_final;
int lut[100][1000];
int function(int end, int sum);
int main()
{
        cout<<"enter the size"<<endl;
        cin>>n;
        cout<<"enter the values"<<endl;
        int i,j;
        for(i=0;i<n;i++)
```

- Search
-
  - Interview Experiences
  - Advanced Data Structures
  - Dynamic Programming
  - Greedy Algorithms
  - Backtracking
  - Pattern Searching
  - Divide & Conquer
  - Graph
  - Mathematical Algorithms
  - Recursion
  - Java
- GeeksforGeeks

  Like You like this.

  You and 28,399 others like GeeksforGeeks.

  Facebook social plugin
-

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

AdChoices

LEXALYTICS

Not first on the **moon**, but first with **sentiment analysis**

GET DEMO

- 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