Dynamic Programming | Set 25 (Subset Sum Problem)

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
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[
}
```

```
// 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]

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

```
// 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
bool isSubsetSum(int set[], int n, int sum)
{
    // The value of subset[i][j] will be true if there is a subset of
    //  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.

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

Recent Comments

- Generate all unique partitions of an integer
- Russian Peasant Multiplication
- Closest Pair of Points | O(nlogn) Implementation

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

55 Comments GeeksforGeeks

Sort by Newest

Guest • a month ago

bool subset(int set[], int size, int sum)

{

for(int i = 0; i<size; i++)="" {="" if="" (set[i]="" <="sum)" sum="" -="set[i];" }="" i
else="" return="" false;="" }="">

Guest • a month ago

The first problem, if replaced by:

bool subset(int set[], int size, int sum)

{

for(int i = 0; i<size; i++)="" {="" if="" (set[i]="" <="sum)" sum="" -="set[i];" }="" i
else="" return="" false;="" }="" ...works="" well="" in="" all="" the="" cases="" i'
is="" it="" i'm="" missing="" here?="">

vinnu • a month ago

Can we try this sum - ar[i] = key search the key value using binary search the
has 2 fours. Can anyone suggest me is it works with tweaking of binary searc

vrg • 2 months ago

Isn't the statement

if (set[n-1] > sum)
return isSubsetSum(set, n-1, sum);

in recursive solution redundant?
We are anyway handling both cases
(a) including the last element
(b) excluding the last element
in the statement
return isSubsetSum(set, n-1, sum) || isSubsetSum(set, n-1, sum-set[n-1]);

Can somebody explain why is it used?

guest11 ➚ vrg • 2 months ago

it will avoid the last stmt where sum will become -ve ....that is not hand

or you can make an extra base case ..... if(sum <0) return 0;

guest11 ➚ vrg • 2 months ago

dont include the element which is already greater than sum

Vinay Singh • 3 months ago

```
//this is my solution
//O(nlogn)

#include<stdio.h>

int main()

{

int array[6]={3,6,4,1,5,2};

int n=6;

int i,j,t;

int f=0,u=5,k;

int sum=5;

int no;
```

see more

prashant jha • 3 months ago

the code prints all the subsets for a given sum
http://ideone.com/DyULF2

prashant jha • 3 months ago

```
/*

void fun(int arr[],int *p,int index,int low,int high,int sum)

{
```

```
{

if((sum<0)||(low>high))

return;

if(sum==0)

{

for(int i=0;i<index;i++) {="" cout<<p[i]<<"=" ";="" }="" cout<<"\n";="" return;="" fun(arr,p,index+1,low+1,high,sum-arr[low]);="" fun(arr,p,index,low+1,high,sum arr[]="{3,1,5,2,4,6,7,8,9,12};" int="" n="sizeof(arr)/sizeof(arr[0]);" int="" *p="nev cout<<"enter="" the="" sum.\n";="" cin="">>sum;

fun(arr,p,0,0,n-1,sum);

return 0;

}
*/
```

saurabh bhatia • 6 months ago

anyone plz tell the algo if the array include negative numbers also...

Jaime saurabh bhatia • 5 months ago

I believe so:

1. Take the minimum value in the set, call it k.
2. Add each element in the set by the absolute value of k.
3. Add sum by the absolute value of k.
4. Perform the algorithm (and consider overflow).
5. Perform the inverse of steps 2 & 3.

Mike Jaime • 5 months ago

This won't work. Take the set (-5, 10) and see if any subset adc
> (0, 15) and 5->10. -5+10=5, but 0+15 != 10

Justin Domingue Mike • 3 months ago

Add n times the absolute value of k to the sum and it wc

saurabh bhatia Jaime • 5 months ago

thnx jamie but chk out this soln.....as i have used recursion 4 di
http://codingstreak.blogspot.i...

rishabh roy • 6 months ago

could any one explain this part

```
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];

}
```

}

1 ^ | v •

its_dark • 7 months ago

We can use just a 1D array to store all the possible sums.Time complexity wo complexity O(sum);

```
int possiblesum[sum+1];; //initialize it with all zeroes
for(int i=0; i< n;i++){
     for(int j = sum ; j>=a[i] ;j--){
          if(possiblesum[j-a[i]]==1)
                possiblesum[j]=1;
     }
}


If possiblesum[i]=1, sum=i is possible with a subset.
```

4 ^ | v •

Anurag ↗ its_dark • 2 months ago

It seems possiblesum[0] needs to be initialized to 1, otherwise it may r to one of array element.

^ | v •

raviteja ↗ its_dark • 6 months ago

I believe it is j-- in the second for loop

1 ^ | v •

its_dark ↗ raviteja • 6 months ago

yup..thanks

Shivam ↗ its_dark • 7 months ago

What is x?

its_dark ↗ Shivam • 7 months ago

Edited !

its_dark • 7 months ago

can't we have a dp for this that takes purely polynomial time ?

Vinodhini • 10 months 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

7

sheetal ↗ Vinodhini • 7 months ago

```
public boolean subsetSum(int[] array, int sum) {

boolean[][] d = new boolean[sum + 1][array.length + 1];

for (int i = 0; i <= array.length; i++) {

d[0][i] = true;
```

```
}

for (int i = 1; i <= sum; i++) {

d[i][0] = false;

}

for (int i = 1; i <= sum; i++) {

for (int j = 1; j <= array.length; j++) {

d[i][j] = d[i][j - 1];
```

see more

sheetal Vinodhini • 7 months ago

```
/// Sample Code here

for (int i = sum; i > 0; i--) {

for (int j = array.length; j > 0; j--) {

while (d[i][j] == false) {

if (i >= array[j]) {

i = i - array[j];

System.out.println("[" + i + "," + j + "]" + "--->" + array[j]);

}

while (i > 0 && j > 0 && d[i][j] != false) {
```

```
j--;

}

}

}

}
```

its_dark ➦ Vinodhini • 7 months ago

You can trace back down the path for a particular sum from the 2D arr

Vinodhini • 10 months 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

6

Born Actor • 10 months 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++)
                cin>>a[i];
        cout<<"enter the sum"<<endl;
```

see more

shek8034 • 11 months ago

How to find all such subsets ?

4

mani • 11 months ago

cant we first sort it and then proceed like this--O(nlogn)

First substract the lastdigit from required sum and check the nearest number do it for all possible ways and check the min.

hh • 11 months ago

In the function why we assign subset[i][j] = subset[i][j-1].
Is it necessary to do so??
Why not subset[i][j] = subset[i][j-1]||subset[i-set[j-1]][j-1].

Akshay • 11 months ago

Akshay • 11 months ago

when original set is : {3, 34, 4, 12, 5, 2} and original sum is 0 then your methoc should return FALSE.

only an original set of type: {3, 34, 4, 0, 5, 2} should return TRUE if original sun

Following is the code WITHOUT Dynamic Programming to take care of the ab

[sourcecode language="C++"]
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <Math.h>
#include <vector>

using namespace std;

bool SubsetSum(int set[], int n, int sum)
{
if(sum - set[n] == 0) return true;

see more

Akshay ➔ Akshay • 11 months ago

For DYNAMIC PROGRAMMING the solution you have given needs a m number of elements in the original set.

But in the cases where n is very small (eg. 6) and sum is very large (e will be too large. To avoid large space complexity n HASHTABLES can

Values will be stored there like this: HASH[n].find(sum)

Time complexity will be O(1) for reading the value and storing it in the H

```
/* Paste your code here (You may delete these lines if not wri
```

Rishu Kaul • 11 months ago

haha

mkamithkumar • a year ago

```
public class PartitionProblem {
	public static void main(String args[]) {
		int input[] = {1,1};
		String result = partition(input);
		System.out.println(result);

	}

	public static String partition(int[] input1) {
		int n = input1.length;
		String output = "";
		// Checking input validations
		if (n <= 0 || input1 == null) {
			output = "Invalid";
		} else {
			for (int k = 0; k < input1.length; k++) {
				if (input1[k] <= 0) {
					output = "Invalid";
```

see more

Leaner • a year ago

Can somebody pls expalin what happens in this line
subset[i][j] = subset[i][j] || subset[i - set[j-1]][j-1];
I tried putting print statements, but couldnt get the catch of it.
Thanks.

Varadh • a year ago
Why can't we build a binary tree and apply the pathSum algorithm to find the s

Varadh ➚ Varadh • a year ago
Sorry that wont work...subset need not be contigious...

Anirudh • a year ago
How to modify the code above , if we are given negative numbers also in the a

Guest ➚ Anirudh • 8 months ago
i think following will work..

take min of the given set and add |min| to every number in the set (ess
stop when sum=m*|min| (where m is size of the subset)..

Treble • a year ago

```
bool has_subset_sum(int *a, int n, int sum)
{
        bool *s = (bool *)malloc((sum+1)*sizeof(bool));
        memset(s, 0, (sum+1)*sizeof(bool));
        s[0] = true;
        for (int i = 0; i < n; ++i) {
```

```
                for (int j = 0; j <= sum; ++j) {
                        if (s[j] == true && a[i] + j <= sum) {
                                s[j + a[i]] = true;
                        }
                }
        }
        bool ret = s[sum];
        free(s);
        return ret;
}
```

Alex • a year ago

```
/*
    static boolean isSubsetSum(int array[], int n, int sum) {
        Map<Integer,Boolean> map = new HashMap<Integer, Boolean>();
        map.put(0, true);

        for(int index =0; index < n; index++){

            List<Integer> toBeAdded = new ArrayList<Integer>();
            for(Entry<Integer, Boolean> entry: map.entrySet()){
                Integer key = entry.getKey();
                if(key+array[index] <= sum) {
                    toBeAdded.add(key+array[index]);
                }
            }
            for(Integer number : toBeAdded){
                map.put(number, true);
            }
```

see more

jobin • a year ago

what you have done is similar to DP, but it won't work because you have negle

```
 int isSubsetSum(int set[], int n, int sum)
{
    int isSum[sum+1];
    int i, j;
    isSum[0] = 1;
    for(i=0;i<n;i++)
    {
      isSum[set[i]] = 1;
    }
    for (i = 0; i < n; i++) {
        for (j = sum - set[i]; j >= 0; j--) {
            if (isSum[j] == 1)
                isSum[j+set[i]] = 1;
        }
        if (isSum[sum] == 1)
            return 1;
    }

    return 0;
}
```

Kanhaiya • a year ago

I think we can apply non-dynamic solution for this particular problem. The belo
the array in with some good sort technique and then use the below.

[sourcecode language="C++"]

#include <iostream>

using namespace std;
//Please note that list is sorted
bool sum_in_list (int list[], int n, int sum)
{
int i = 0;
int j = n - 1;

while (i < n && i < j)
{
if (sum == list[i] || sum == list[j] || sum == (list[i] + list[j]))
{
return true;

see more

Kanhaiya Kanhaiya • a year ago

As pointed out by Karthik, its not correct solution for subset problem. P

Kanhaiya • a year ago

I think for this particular problem, dynamic programming may not be best solut
1. sort the array
2. Keep pointers from start and end. Check if sum of those make it to the requ
accordkingly.

Kartik ↗ Kanhaiya • a year ago

@Kanhaiya : This solution works well when we need to find two eleme
here the problem is to find a subset (there may be any number of elem

Kanhaiya ↗ Kartik • a year ago

got that :) sorry for not reading it properly.

Jason • a year ago

I was just messing around with something like this! Here's a one-liner in haske

[sourcecode language="haskell"]
import Data.List

isSubsetSum xs n = any (== n) . map sum . nub . concatMap subsequences

Prakhar Jain • a year ago

I'm indebted to bcurcio for this code.
http://www.codechef.com/viewso...

```
#include <stdio.h>

int isSubsetSum(int set[], int n, int sum)
{
        int isSum[sum+1];
        int i, j;
        isSum[0] = 1;
        for (i = 0; i < n; i++) {
                for (j = sum - set[i]; j >= 0; j--) {
```

```
                        if (isSum[j] == 1)
                                isSum[j+set[i]] = 1;
                }
                if (isSum[sum] == 1)
                        return 1;
        }
```

see more

Load more comments

Subscribe Add Disqus to your site