Dynamic Programming | Set 18 (Partition problem)

Partition problem is to determine whether a given set can be partitioned into two subsets such that the sum of elements in both subsets is same.

Examples

```
arr[] = {1, 5, 11, 5}
Output: true
The array can be partitioned as {1, 5, 5} and {11}

arr[] = {1, 5, 3}
Output: false
The array cannot be partitioned into equal sum sets.
```

Following are the two main steps to solve this problem:
1) Calculate sum of the array. If sum is odd, there can not be two subsets with equal sum, so return false.
2) If sum of array elements is even, calculate sum/2 and find a subset of array with sum equal to sum/2.

The first step is simple. The second step is crucial, it can be solved either using recursion or Dynamic Programming.

Recursive Solution
Following is the recursive property of the second step mentioned above.

```
Let isSubsetSum(arr, n, sum/2) be the function that returns true if
there is a subset of arr[0..n-1] with sum equal to sum/2
```


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

```
The isSubsetSum problem can be divided into two subproblems
 a) isSubsetSum() without considering last element
    (reducing n to n-1)
 b) isSubsetSum considering the last element
    (reducing sum/2 by arr[n-1] and n to n-1)
If any of the above the above subproblems return true, then return true.
isSubsetSum (arr, n, sum/2) = isSubsetSum (arr, n-1, sum/2) ||
                              isSubsetSum (arr, n-1, sum/2 - arr[n-1])
```

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

// A utility function that returns true if there is a subset of arr[]
// with sun equal to given sum
bool isSubsetSum (int arr[], 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 (arr[n-1] > sum)
     return isSubsetSum (arr, 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 (arr, n-1, sum) || isSubsetSum (arr, n-1, sum-ar

}

// Returns true if arr[] can be partitioned in two subsets of
// equal sum, otherwise false
bool findPartiion (int arr[], int n)
{
    // Calculate sum of the elements in array
    int sum = 0;
    for (int i = 0; i < n; i++)
       sum += arr[i];

    // If sum is odd, there cannot be two subsets with equal sum
```


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

```
    if (sum%2 != 0)
       return false;

    // Find if there is subset with sum equal to half of total sum
    return isSubsetSum (arr, n, sum/2);
}
```

```
// Driver program to test above function
int main()
{
  int arr[] = {3, 1, 5, 9, 12};
  int n = sizeof(arr)/sizeof(arr[0]);
  if (findPartiion(arr, n) == true)
     printf("Can be divided into two subsets of equal sum");
  else
     printf("Can not be divided into two subsets of equal sum");
  getchar();
  return 0;
}
```

Output:

```
Can be divided into two subsets of equal sum
```

Time Complexity: O(2^n) In worst case, this solution tries two possibilities (whether to include or exclude) for every element.

Dynamic Programming Solution

The problem can be solved using dynamic programming when the sum of the elements is not too big. We can create a 2D array part[][] of size (sum/2)*(n+1). And we can construct the solution in bottom up manner such that every filled entry has following property

```
part[i][j] = true if a subset of {arr[0], arr[1], ..arr[j-1]} has sum
             equal to i, otherwise false
```

```
// A Dynamic Programming solution to partition problem
#include <stdio.h>
```

```
// Returns true if arr[] can be partitioned in two subsets of
// equal sum, otherwise false
bool findPartiion (int arr[], int n)
{
    int sum = 0;
```

```
    int i, j;

    // Caculcate sun of all elements
    for (i = 0; i < n; i++)
      sum += arr[i];

    if (sum%2 != 0)
       return false;

    bool part[sum/2+1][n+1];

    // initialize top row as true
    for (i = 0; i <= n; i++)
      part[0][i] = true;

    // initialize leftmost column, except part[0][0], as 0
    for (i = 1; i <= sum/2; i++)
      part[i][0] = false;

     // Fill the partition table in botton up manner
     for (i = 1; i <= sum/2; i++)
     {
       for (j = 1; j <= n; j++)
       {
         part[i][j] = part[i][j-1];
         if (i >= arr[j-1])
           part[i][j] = part[i][j] || part[i - arr[j-1]][j-1];
       }
     }

    /* // uncomment this part to print table
     for (i = 0; i <= sum/2; i++)
     {
       for (j = 0; j <= n; j++)
          printf ("%4d", part[i][j]);
       printf("\n");
     } */

     return part[sum/2][n];
}

// Driver program to test above funtion
int main()
{
  int arr[] = {3, 1, 1, 2, 2, 1};
  int n = sizeof(arr)/sizeof(arr[0]);
  if (findPartiion(arr, n) == true)
```


Recent Comments

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

Write a C program to Delete a Tree. · 37 minutes ago

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 41 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

```
    printf("Can be divided into two subsets of equal sum");
  else
    printf("Can not be divided into two subsets of equal sum");
  getchar();
  return 0;
}
```

Output:

```
Can be divided into two subsets of equal sum
```

Following diagram shows the values in partition table. The diagram is taken form the wiki page of partition problem.

The entry part[i][j] indicates whether there is a subset of {arr[0], arr[1], .. arr[j-1]} that sums to i

	{}	{3}	{3,1}	{3,1,1}	{3,1,1,2}	{3,1,1,2,2}	{3,1,1,2,2,1}
0	True	True	True	True	True	True	True
1	False	False	True	True	True	True	True
2	False	False	False	True	True	True	True
3	False	True	True	True	True	True	True
4	False	False	True	True	True	True	True
5	False	False	False	True	True	True	True

Dynamic Programming table for

arr[] ={3,1,1,2,2,1}

Time Complexity: O(sum*n)
Auxiliary Space: O(sum*n)
Please note that this solution will not be feasible for arrays with big sum.

References:

http://en.wikipedia.org/wiki/Partition_problem

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

Related Tpoics:

- Remove minimum elements from either side such that 2*min becomes more than max
- Divide and Conquer | Set 6 (Search in a Row-wise and Column-wise Sorted 2D Array)
- Bucket Sort
- Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1
- Find the number of zeroes
- Find if there is a subarray with 0 sum
- Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)
- Count all possible groups of size 2 or 3 that have sum as multiple of 3

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

48 Comments GeeksforGeeks

48 Comments GeeksforGeeks

Sort by Newest

Join the discussion…

Rohit Sharma • a month ago

here is the code with time complexity-0(n^2)

==========

algorithm---

1-find sum of array.

if odd return false;

sum=sum/2;

2.sort array by using any method.

3.apply dynamic algorithm similar to find sub array given sum.

here is code !!

==========

#include<stdio.h>

see more

Dante Fan → Rohit Sharma • 5 days ago

It's an NP-complete problem, no polynomial solution yet...

prashant • 4 months ago

here is the naive recursive approach which returns the min differnece detweer

```
/*

int min(int a,int b)

{

return a>b?b:a;

}

int fun(int arr[],int low,int high,int s1,int s2)

{

if(low>high)

return s1>s2?(s1-s2):(s2-s1);

return min(fun(arr,low+1,high,s1+arr[low],s2),fun(arr,low+1,high,s1,s2+arr[low
```

see more

Aja Huang • 7 months ago

There is a much faster and simpler solution using STL bitset.
https://github.com/swem/UVa-On...

2

sumit dey • 9 months ago

Here is the java version of the same problem, it will also print the solution of th
solution. No need of sorted input and it works for negative solution. The printed

solution. No need of sorted input and it works for negative solution. The printed subset, other subset will be the excluded element.

```
/**
*
*/


import java.util.LinkedHashMap;
import java.util.Map;

public class PartitionSumSubsetProblem {

        public static class CachedDataAttr {
                int sum;
                int indexOfArray;
                int bfrSumDiff;
```

see more

1

Soumak Datta • 9 months ago

will not work for all cases.....consider the case where nos after sorted are {21, approach, the sets will be {21,5,2}and {20,6}...while the solution is {21,6} and {

1

Sunil Singhal • 9 months ago

Will Not. Consider a case {5,9,9,13}

Kuldeep Tiwari • 10 months ago

Will work. You have not sorted the array here.

Aman Jain • 10 months ago

if array contain -ve elements too, then this code can work for it too..
sum can be any value..

```
[sourcecode language="C++"]
#include<iostream>
using namespace std;
int func(int *arr,int n,int sum)
{
int x=0;
int y=0;

int i,j;
for(i=1;i<=n;i++)
{
if(arr[i]>=0)
{
x+=arr[i];

}
```

see more

Sandeep Jain • a year ago

This doesn&#039t work for all cases. For example, it won&#039t work for {5, 5
3, 3}.

GeeksforGeeks • a year ago

Siddhartha: Please take a closer look at the problem statement. You need to d
element must be in one of the two parts. In your example, you have ignored 15

element must be in one of the two parts. In your example, you have ignored re

Puneet Garg • a year ago

another solution.
at first, sort array in decreasing order. then take two variable as sum1 & sum2
1st element to sum1 & add 2nd element to sum2. if sum1 < sum2 then add 3r
then add 3rd element to sum2.continue up to last element. if sum1==sum2 the
not.

anonymous ➦ Puneet Garg • 4 months ago

for this {5, 5, 4, 3, 3} .. ??

anonymous ➦ Puneet Garg • 4 months ago

if sum1 = sum2, then what to do? add to which sum?

Susheel Pandey • a year ago

superb man :P

abhishek08aug • a year ago

Intelligent :D

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

googlybhai • a year ago

Solution will work only for +ve numbers. So either we should change the probl

Siddhartha Sharma • a year ago

As mentioned above:-

Following are the two main steps to solve this problem:
1) Calculate sum of the array. If sum is odd, there can not be two subsets with
2) If sum of array elements is even, calculate sum/2 and find a subset of array

what if we have an array {15,10,10} which has sum odd (35) but can be broke
equal sum contradicting point 1.

Siddhartha's Father → Siddhartha Sharma • 6 months ago

What are you son? Blind! Your Array contains 3 elements.

3

anand • a year ago

```
public class PartitionProblem {

        /**
         * @param args
         */
        public static void main(String[] args) {
                int A[] =  {5, 5, 4, 3, 3};
                System.out.println(isSubSet(A, A.length));
        }

        private static boolean isSubSet(int[] A, int length) {
                int i,partialSum,sum = 0;

                for(i=0;i<length;i++){
```

```
            sum +=A[i];
        }
        if(sum%2 == 1)return false;
```

see more

kT → anand • a year ago

@ anand,

I think your solution will fail for i/p : 3,4,8,9.
Please check and sorry if my observation is incorrect.

Thanks.

Patrick → kT • a year ago

@KT:
I modified anand's code and now it's working for all cases.

```
bool subPartition(int *A, int len){
        int sum=0,partitionSum;
        for(int i=0; i<len;i++){
                sum=sum+A[i];
        }
        if(sum%2==1)
                return false;
        else{
                partitionSum=sum/2;
                qsort(A,len); //sort in decreasing order
                for(int i=0; i<len;i++){
                        while(partitionSum<A[i] && i<ler
                                i++;
```

```
            }
                        partitionSum=partitionSum-A[i];
```

see more

Rahul Singh → Patrick • a year ago

Patrick, what you are suggesting is a greedy approach. {2,3,4,5,7,9}.

Patrick → anand • a year ago

nice solution.... thanks

Raja • 2 years ago

Hi,

I would like to know In what cases sum is less than an array element(it might l numbers).

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

Nikhil • 2 years ago

// If last element is greater than sum, then ignore it
if (arr[n-1] > sum)
return isSubsetSum (arr, n-1, sum);
Regarding this code,I think we should return false as question says we need t ignoring any element,the we will create more than 2 subsets.

Please correct me if I am wrong!!!

Nikhil • 2 years ago

// If last element is greater than sum, then ignore it
if (arr[n-1] > sum)
return isSubsetSum (arr, n-1, sum);
Regarding this code,I think we should return false as question says we need t
ignoring any element,the we will create more than 2 subsets.

Please correct me if I am wrong!!!

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

Nishant Mittal • 2 years ago

*can be done by recursion with O(n^2).
int32_t check_sum (int32_t array[] , int32_t i , int32_t sum) {
if (i<max) {
check_sum (array , i+1 , sum);
sum=sum-array[i];
if (sum ==0) return ans=1;
check_sum (array , i+1 , sum);
}
}

Anonymous • 2 years ago

im wondering if the condition should be:

```
if (i >= arr[j-1])
            part[i][j] = part[i][j-1] || part[i - arr[j-1]][j-1];
```

Mayautobot • 2 years ago

"Please note that this solution will not be feasible for arrays with big sum."

--->What can be the approach for finding solution with large sum?

Simpson • 2 years ago

```
#include
#include
#include
int compare(const void * a, const void * b)
{
return ( *(int*)a - *(int*)b );
}
int main()
{
int a[7];
int sum_even ,sum,m,n,i,flag;
sum_even=sum=m=flag=0;
n=6;
for( i=0;i<7;i++)
{
scanf("%d",&a[i]);
}
```

see more

keshav → Simpson • 2 years ago

Plz clearify ur method.

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

Abhinav Priyadarshi • 2 years ago

i think there is a error in explanation of part[i][j], it should be
part[i][j] = true if a subset of {arr[0], arr[1], ..arr[j-1]} has sum equal to **** i ****

Abhinav Priyadarshi → Abhinav Priyadarshi • 2 years ago

***** are used to highlight.

GeeksforGeeks → Abhinav Priyadarshi • 2 years ago

@Abhinav Priyadarshi: Thanks for pointing this out. We have c

Ankit Gupta • 2 years ago

Hi, I am using a top-down recursive memoization to populate the cache array

I want to know :
1. Whether following solution is correctly implemented.
2. I don't see the sub-problems being reused in the cache array so a bottom u
entries unused (and populated nonethless).

```
 int arr[] = {3, 1, 1, 2, 2, 1};
int N, S;
int cache[MAXN][MAXSUM];
```

```
int cache[MAXN][MAXSUM];

bool partition(int pos, int sum)
{
    if (sum == S>>1) {
        return true;
    }

    if (pos == N || sum > S>>1) {
        return false;
```

see more

Ankit Gupta Ankit Gupta • 2 years ago

I have initialized cache to -1 and S to total Sum.

atul • 2 years ago

space optimized DP approach :-

```
part[0]=1;
for(i=0;i<n;i++)
{
   for(j=sum;j>=arr[i];j--)
   {
        part[j]=part[j] | part[j-arr[i]]
   }
}
if(part[sum])
   printf("\nsubset exists\n");
```

penhaunt ↪ atul • 9 months ago

aap to chaa gye sir :)

penhaunt ↪ atul • 9 months ago

aap god hain :)

atul ↪ atul • 2 years ago

here sum=Total_Sum/2;

Wayne • 2 years ago

Condn- sum should be divisible by 2;

We know the "sum" right !!! sum = sum of all the elements in array

Cant we use sum of subset problem (knapsack) to find out the subset which h
easy because we know that the rest of elements sum is "sum/2",

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

sonal ↪ Wayne • 2 years ago

I think that is what the given DP code is doing :)

Wayne ↪ sonal • 2 years ago

Oops .. My Bad ...

But why are they using 2D array .... I guess 1D array of SUM/2

```
/* Paste your code here (You may delete these lines if
```

sindabad • 2 years ago

```
 Sort the data in descending order
for(i=1 to n)
{
if(sum(set1) > sum(set2))
include the number in set2
else
if(sum(set2) > sum(set1))
include the number in set1
else
return 1;
}
retunr 0;
```

kartik → sindabad • 2 years ago

this will not work for {5, 5, 4, 3, 3}

Venki • 2 years ago

How is different from fair work load problem?

http://topcoder.bgcoder.com/pr...

Reference: The Algorithm Design Manual by Skiena.

Ankit Gupta → Venki • 2 years ago

Hi, thanks for sharing the problem.
I can think of a recursive backtracking solution (do not know if it's right)

```
 If workers == 0
    return 0
else
    calculate all possible partition(folders, N, sum/N) say nev
    // sum is total count of the elements in folders and N is ≠
    foreach newfolders
        max = maximum of (sum(folders)-sum(newfolders), getMost
                and max
    return max
```

Can you hint at a solution ?