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

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-arr[n-1])
}

// 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
    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)
     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 Topics:

- Find Union and Intersection of two unsorted arrays
- Pythagorean Triplet in an array
- Maximum profit by buying and selling a share at most twice
- Design a data structure that supports insert, delete, search and getRandom in constant time
- Print missing elements that lie in range 0 – 99
- Iterative Merge Sort
- Group multiple occurrence of array elements ordered by first occurrence
- Given a sorted and rotated array, find if there is a pair with a given sum

Tags: Dynamic Programming

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

Recommend Share Sort by Newest

Join the discussion…

sanki • 13 days ago

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

i think this should be changed to

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

• Reply • Share ›

amit singh ➚ sanki • 9 days ago

already did it before the if block

• Reply • Share ›

Victor • 23 days ago

What will be the result if Input is {1, 3}

• Reply • Share ›

MiniMe • 2 months ago

Do we really need a 2-D array for this?

I mean we can just have an array part[0..SUM], with all elements initialized to -1. For every element in the array arr[], we iterate over part. Not only do we make part [arr[i]] to ONE, but all part[i + part[j]] ONE such that part[j] is also ONE.

• Reply • Share ›

Vinay Pai • 3 months ago

Why cant we sort the given array and maintain 2 pointers. one at the beginning of the array and the other at the end of the array and increment and decrement each of the pointers respectively till the sum become s equal?

• Reply • Share ›

Balaji ➚ Vinay Pai • 3 months ago

What if the required subset is {first, last} and {remaining} ? The problem is proven NP complete, no polynomial solution exists. The DP approach is just Pseudo polynomial as sum increases complexity increases.

2 • Reply • Share ›

gg • 4 months ago

why is size (sum/2)*(n+1)? why (sum/2) and not sum?

• Reply • Share ›

Anurag Singh gg • 4 months ago

Here we are looking for two subsets with equal sum, i.e. two subsets with sum = sum/2.

• Reply • Share ›

Sunil Sharma • 5 months ago

Why are we initializing the top row as true and leftmost colmun as false ?

• Reply • Share ›

Anurag Singh Sunil Sharma • 5 months ago

part[i][j] = true if a subset of {arr[0], arr[1], ..arr[j-1]} has sum
equal to i, otherwise false
part[0][i] will be true if there is a subset in index range 0 to i-1 with sum equal to zero. And this is true for all i and the subset would be the empty subset. So part[0][i] is always true and same is initialized in code in 1st row.
part[i][0] (where i > 0) will be true when there is a subset in index range 0 to 0-1 with sum equal to i (where i > 0). There is no subset for this and so part[i][0] (where i > 0) will always be false.

• Reply • Share ›

Charanjeet Singh • 5 months ago

i made this:-
using System;

public class CandidateCode

{

public static string partition(int[] input1)

{

int totalSum=0;

if(input1.Length==0){return "Invalid";}

for (int i=0;i<input1.length;i++) {="" if(input1[i]<="0)" {="" return="" "invalid";="" }="" totalsum+="input1[i];" }="" int="" cell="input1.Length;" for(int="" i="0;i<cell;i++)" {="" for(int="" j="0;j<cell-1;j++)" {="" if(input1[j]="">input1[j+1])

{

see more

• Reply • Share ›

Shubhang Sati • 5 months ago

Can we extend this for any number of partitions? I mean, we have an array and we want to check whether we can divide it into "k" subsets with equal sum or not?

3 • Reply • Share ›

Anurag Singh ➚ Shubhang Sati • 5 months ago

The k-partition problem has no pseudo-polynomial time solution.
One way to do this could be using Subset Sum Problem approach.
Given the array, 1st find if there is a subset with sum equal to "sum/k".
If it is found, remove those subset element from original array and repeat the same process on modifying array. If we find K subset with sum "sum/k" and array becomes empty, you have got the K subsets. At anytime in the process, if you don't find subset with sum "sum/k", stop the process as K subset with equal sum "sum/k" are not possible. Note that here sum is the sum of original array elements, not the modified one in later processes.

1 • Reply • Share ›

Counter-rationalsit ➚ Anurag Singh • 4 months ago

Such a greedy approach wont always work.

Assume sum=30, k=3 with numbers as 4, 3, 3, 7, 7, 6.

You will remove (4, 3, 3) and be left with 7, 7, 6 and then declare its not possible while the solution should be (4, 6), (3, 7), (3, 7).

• Reply • Share ›

Anurag Singh ➚ Counter-rationalsit • 4 months ago

You are right. If there are multiple subset for a given SUM, then we need to try all subsets one by one.

• Reply • Share ›

Noman ➚ Shubhang Sati • 5 months ago

i have also the same question. can anyone help please.

• Reply • Share ›

Mandeep • 5 months ago

Here's a simpler way (albeit not very efficient) way of thinking about the recursive solution -

```
boolean canSubsetParition(int i, int[] arr, int sum1, int sum2) {
if (i >= arr.length) {
```

```
return sum1 == sum2;
}

return canSubsetParition(i+1, arr, sum1+arr[i], sum2) ||
canSubsetParition(i+1, arr, sum1, sum2+arr[i]);
```

1 • Reply • Share ›

ashish100 • 5 months ago

working solution.....one can fill upper half matrix to arrive to conclusion:

int fun(int arr[],int size,int sum)

{

int brr[size][size],i,j,d=0;

int flag = 1;

d=0;

while(flag == 1)

{

//d=0;

for(i=0;i<size;i++) {="" if(j+d="" <="" size)="" {="" if(i="j+d)" {="" brr[i][j+d]="arr[i];" }="" else="" {="" brr[i][j+d]="brr[i][j+d-1]+arr[j+d];" }="" }="" if(brr[i][j+d]="=" sum)="" {="" return="" 1;="" }="

see more

• Reply • Share ›

Hello_world • 6 months ago

What if sum is very large .... Is there any other method to get that other than naive and recursive approch ?

• Reply • Share ›

Anurag Singh ➚ Hello_world • 6 months ago

This problem is a variation of subset sum problem which is NP-complete problem. The dynamic programming approach gives Pseudo-polynomial time result.
More details can be found at wiki reference.

• Reply • Share ›

Hello_world • 6 months ago

this problem is same as coin cost algo. We need to change for sum/2.

1 • Reply • Share ›

Shruti Saxena • 6 months ago

can anyone explain this part

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];

2 • Reply • Share ›

Sneha ➚ Shruti Saxena • 6 months ago

statement1: First we will check that excluding current item can we achieve sum i.

statement 2: We are checking i>=arr[j-1] because in next statement we are accessing index i-arr[j-1]. This checking is necessary because there is possibility that array index may become negative.

statement 3: Just check that excluding current item OR including current item gives us required sum.

3 • Reply • Share ›

Itachi Uchiha ➚ Sneha • 3 months ago

Thanks! It Helped! :)

• Reply • Share ›

RK- An Unproven Theorem ➚ Sneha • 5 months ago

@Sneha Can you please elaborate? I still can't understand...

• Reply • Share ›

Zaid ➚ RK- An Unproven Theorem • 2 months ago

There are two options:
1. Pick current element, OR
2. Do not pick it
If we choose one, we have to check whether the Sum - CurrentElement can be partitioned.
What if CurrentElement > Sum
=>We avoid it and choose option 2.
That is all

• Reply • Share ›

ryan • 7 months ago

what is need of
if (arr[n-1] > sum)
return isSubsetSum (arr, n-1, sum);in recursion

• Reply • Share ›

Anurag Singh ↗ ryan • 7 months ago

If all elements of array add to "sum", which is even, then
We are trying to find a subset of array which adds to sum/2.
For this, we do following for each elements:
1. if we DO NOT consider current element as part of subset, can we find a possible subset (which adds to sum/2) finally ?
2. if we consider current element as part of subset, can we find a possible subset (which adds to sum/2) finally ?

If current element is already greater than expected total sum, then we definitely know that this element should not be considered. And this is what we are doing using if condition.

• Reply • Share ›

ryan ↗ Anurag Singh • 7 months ago

thanks anurag

• Reply • Share ›

Priya • 7 months ago

Can anyone explain why array part[][] has size (sum/2)*(n+1)

• Reply • Share ›

Siya ↗ Priya • 7 months ago

Read this before solving this question

http://www.geeksforgeeks.org/d...

• Reply • Share ›

Guest • 7 months ago

why not just sort the array and take two pointers. One pointing to the start of the array and other to the end. Decrement end pointer if the element pointer by end pointer is greater than sum/2; else keep end pointer steady and keep incrementing start pointer and calculating the sum of elements. If at any time your sum exceeds the required sum move start pointer back to starting point and decrement end pointer.

• Reply • Share ›

Rohit Adhav ↗ Guest • 6 months ago

hey thats really not bad idea...!! great bro. who are you.?

• Reply • Share ›

Anurag Singh ↗ Guest • 7 months ago

Consider array after sorting as {1, 2, 3, 4, 5, 6, 7}

Here partition is: {3, 5, 6} and {1, 2, 4, 7}
which can't be obtained by the logic you mentioned.

• Reply • Share ›

praveen • 7 months ago

it can be done by taking a 1-d array instead of 2d array....

• Reply • Share ›

Siya praveen • 7 months ago

Can u plz explain how will you do it with 1 d array?

• Reply • Share ›

Guest • 7 months ago

I think that the recursive solution fails when the input is {2,5,11,5, 16}. 2 subsets can be formed from this set that meet the requirement: {11,5} and {16}. But the sum of all array items amounts to an odd number, and the function will return false;

• Reply • Share ›

Anurag Singh Guest • 7 months ago

You MUST consider all the array elements while partionining.
We are not doing partial partitioning here.

• Reply • Share ›

Som Bose • 7 months ago

A simpler solution:

1.sort the array

2. sum = 0;

3. for(i =1..n)

4. sum = sum + arr[i];

5. if(sum == desired sum(i.e original sum/2))

6. break;

7 if(sum > desired sum(i.e original sum/2))

8. break;

9 end for

10. if(sum == desired sum(i.e original sum/2))

11. print(is possible)

11. print(is possible)
12 else
13. print(not possible)

• Reply • Share ›

Jon Snow ➚ Som Bose • 7 months ago
your solution will not work on following input
{1,5,7,11}
Even though there is subset{1,11} and {5,7} your solution will print not possible.

1 • Reply • Share ›

Vikas Malviya • 9 months ago
please explain the dp solution a little bit

• Reply • Share ›

guest ➚ Vikas Malviya • 9 months ago
yes please

• Reply • Share ›

kapil jain ➚ guest • 9 months ago
In this problem our task is to collect sum/2 number. that is nothing but a variant of knapsack problem .. here sum/2 is your bag capacity. and number is weight .... value is equal for all .... that's why we are using true and false here...

3 • Reply • Share ›

Karan Gupta • 10 months ago
Is it correct one..?
1. take sum of array and if sum is not even partition is not possible.
2. sort it O(nlgn)
3. find subarray with given s=sum/2, in O(n) time. if we get then partition is possible

4 • Reply • Share ›

Jon Snow ➚ Karan Gupta • 7 months ago
your solution will not work on following input
{1,5,7,11}.
Even though there is subset{1,11} and {5,7} your solution will print not possible.

1 • Reply • Share ›

Rohit Sharma ➚ Karan Gupta • 8 months ago
greedy approach after sorting....it is similar to knapsack problem!!

• Reply • Share ›

Palak Jain ➚ Karan Gupta • 9 months ago
i don't think this will work!

try ur soln for the test case used in code for recursive soln!

• Reply • Share ›

hsg92 • 10 months ago

What would be an algorithm for numbers which are negative as well ? this works for only positive numbers ?

2 • Reply • Share ›

krishna • 10 months ago

algorithm
1.calculate the sum of elements of array(sum)
2.check that number is divisible by 2 or not [if(sum/2==0)p=sum/2;]
3.and check sum of any 'k' elements equal to p
3.if there exits any k elements then we can divide into two sub sets
correct me if i am wrong

1 • Reply • Share ›

Rohit Sharma • a year ago

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

==========

algorithm---

1-find sum of array.

if odd return false;

sum=sum/2;

2.sort array by using any method(quick_sort).

3.apply knapsack or greedy algorithm similar to find sub array with given sum.

here is code !!

==========

#include<stdio.h>

see more

• Reply • Share ›

Load more comments

Popular Posts

- 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

Recent Comments

- lt_k

  i need help for coding this function in java...

  Java Programming Language · 1 hour ago

- Piyush

  What is the purpose of else if (recStack[*i])...

  Detect Cycle in a Directed Graph · 1 hour ago

- Andy Toh

  My compile-time solution, which agrees with the...

  Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour ago

- lucy

  because we first fill zero in first col and...

  Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

- lucy

  @GeeksforGeeks i don't n know what is this long...

  Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 3 hours ago

- manish

  Because TAN is not a subsequence of RANT. ANT...

  Given two strings, find if first string is a subsequence of second · 3 hours ago