

Backtracking | Set 4 (Subset Sum)

Subset sum problem is to find subset of elements that are selected from a given set whose sum adds up to a given number K . We are considering the set contains non-negative values. It is assumed that the input set is unique (no duplicates are presented).

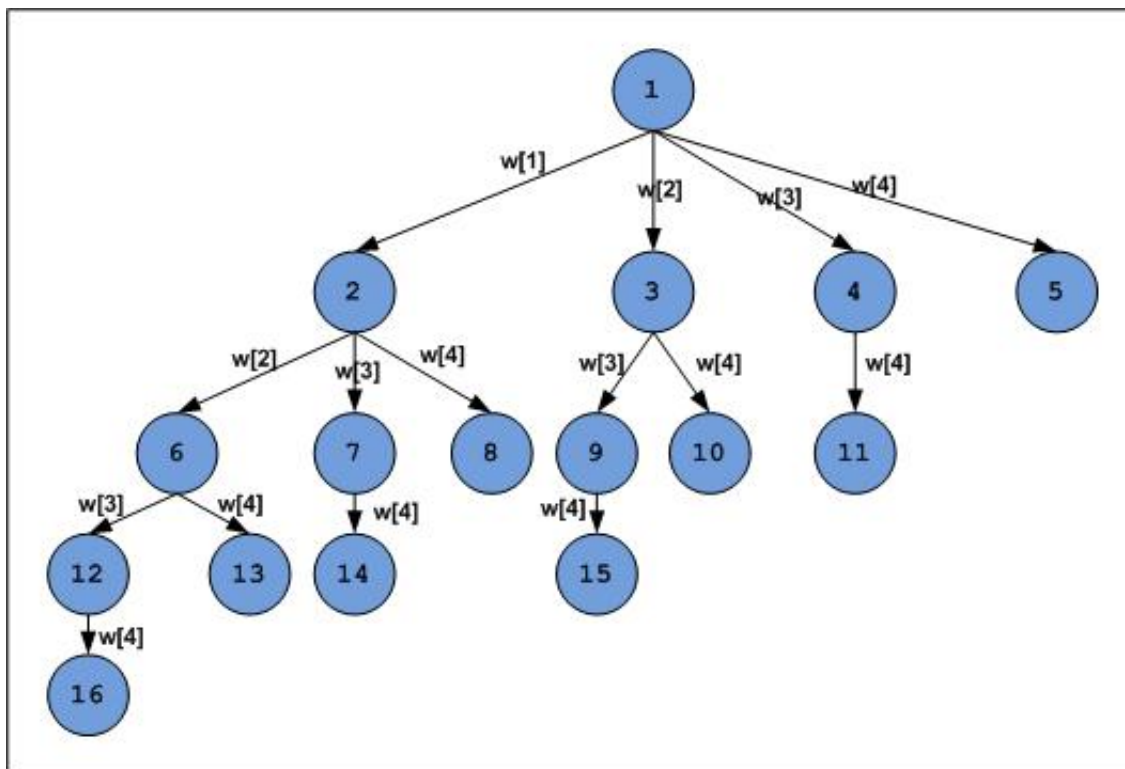
Exhaustive Search Algorithm for Subset Sum

One way to find subsets that sum to K is to consider all possible subsets. A **power set** contains all those subsets generated from a given set. The size of such a power set is 2^N .

Backtracking Algorithm for Subset Sum

Using exhaustive search we consider all subsets irrespective of whether they satisfy given constraints or not. Backtracking can be used to make a systematic consideration of the elements to be selected.

Assume given set of 4 elements, say $w[1] \dots w[4]$. Tree diagrams can be used to design backtracking algorithms. The following tree diagram depicts approach of generating variable sized tuple.



In the above tree, a node represents function call and a branch represents candidate element. The root node contains 4 children. In other words, root considers every element of the set as different branch. The next level sub-trees correspond to the subsets that includes the parent node. The branches at each level represent tuple element to be considered. For example, if we are at level 1, tuple_vector[1] can take any value of four branches generated. If we are at level 2 of left most node, tuple_vector[2] can take any value of three branches generated, and so on...

For example the left most child of root generates all those subsets that include w[1]. Similarly the second child of root generates all those subsets that includes w[2] and excludes w[1].

As we go down along depth of tree we add elements so far, and if the added sum is satisfying explicit constraints, we will continue to generate child nodes further. Whenever the constraints are not met, we stop further generation of sub-trees of that node, and backtrack to previous node to explore the nodes not yet explored. In many scenarios, it saves considerable amount of processing time.

The tree should trigger a clue to implement the backtracking algorithm (try yourself). It prints all those subsets whose sum add up to given number. We need to explore the nodes along the breadth and depth of the tree. Generating nodes along breadth is controlled by loop and nodes along the depth are generated using recursion (post order traversal). Pseudo code given below,

```
if(subset is satisfying the constraint)
    print the subset
    exclude the current element and consider next element
else
    generate the nodes of present level along breadth of tree and
    recur for next levels
```

Following is C implementation of subset sum using variable size tuple vector. Note that the following program explores all possibilities similar to exhaustive search. It is to demonstrate how backtracking can be used. See next code to verify, how we can optimize the backtracking solution.

```
#include <stdio.h>
#include <stdlib.h>

#define ARRAYSIZE(a) (sizeof(a))/(sizeof(a[0]))

static int total_nodes;
// prints subset found
void printSubset(int A[], int size)
{
    for(int i = 0; i < size; i++)
    {
        printf("%d", 5, A[i]);
    }

    printf("\n");
}

// inputs
// s          - set vector
// t          - tuple vector
// s_size     - set size
```

```

// t_size      - tuple size so far
// sum         - sum so far
// ite        - nodes count
// target_sum  - sum to be found
void subset_sum(int s[], int t[],
               int s_size, int t_size,
               int sum, int ite,
               int const target_sum)
{
    total_nodes++;
    if( target_sum == sum )
    {
        // We found subset
        printSubset(t, t_size);
        // Exclude previously added item and consider next candidate
        subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);
        return;
    }
    else
    {
        // generate nodes along the breadth
        for( int i = ite; i < s_size; i++ )
        {
            t[t_size] = s[i];
            // consider next level node (along depth)
            subset_sum(s, t, s_size, t_size + 1, sum + s[i], i + 1, target_sum);
        }
    }
}

// Wrapper to print subsets that sum to target_sum
// input is weights vector and target_sum
void generateSubsets(int s[], int size, int target_sum)
{
    int *tuple_vector = (int *)malloc(size * sizeof(int));
    subset_sum(s, tuple_vector, size, 0, 0, 0, target_sum);
    free(tuple_vector);
}

int main()
{
    int weights[] = {10, 7, 5, 18, 12, 20, 15};
    int size = ARRAYSIZE(weights);

    generateSubsets(weights, size, 35);
    printf("Nodes generated %d\n", total_nodes);
    return 0;
}

```

[Run on IDE](#)

The power of backtracking appears when we combine explicit and implicit constraints, and we stop generating nodes when these checks fail. We can improve the above algorithm by strengthening the constraint checks and presorting the data. By sorting the initial array, we need not to consider rest of the array, once the sum so far is greater than target number. We can backtrack and check other possibilities.

Similarly, assume the array is presorted and we found one subset. We can generate next node excluding the present node only when inclusion of next node satisfies the constraints. Given below is optimized implementation (it prunes the subtree if it is not satisfying constraints).

```

#include <stdio.h>
#include <stdlib.h>

#define ARRAYSIZE(a) (sizeof(a))/(sizeof(a[0]))

static int total_nodes;

// prints subset found
void printSubset(int A[], int size)
{
    for(int i = 0; i < size; i++)
    {
        printf("%d", 5, A[i]);
    }

    printf("\n");
}

// qsort compare function
int comparator(const void *pLhs, const void *pRhs)
{
    int *lhs = (int *)pLhs;
    int *rhs = (int *)pRhs;

    return *lhs > *rhs;
}

// inputs
// s          - set vector
// t          - tuple vector
// s_size     - set size
// t_size     - tuple size so far
// sum        - sum so far
// ite       - nodes count
// target_sum - sum to be found
void subset_sum(int s[], int t[],
               int s_size, int t_size,
               int sum, int ite,
               int const target_sum)
{
    total_nodes++;

    if( target_sum == sum )
    {
        // We found sum
        printSubset(t, t_size);

        // constraint check
        if( ite + 1 < s_size && sum - s[ite] + s[ite+1] <= target_sum )
        {
            // Exclude previous added item and consider next candidate
            subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);
        }
        return;
    }
    else
    {
        // constraint check
        if( ite < s_size && sum + s[ite] <= target_sum )
        {
            // generate nodes along the breadth
            for( int i = ite; i < s_size; i++ )
            {
                t[t_size] = s[i];
            }
        }
    }
}

```

```

        if( sum + s[i] <= target_sum )
        {
            // consider next level node (along depth)
            subset_sum(s, t, s_size, t_size + 1, sum + s[i], i + 1, target_sum);
        }
    }
}

// Wrapper that prints subsets that sum to target_sum
void generateSubsets(int s[], int size, int target_sum)
{
    int *tuple_vector = (int *)malloc(size * sizeof(int));

    int total = 0;

    // sort the set
    qsort(s, size, sizeof(int), &comparator);

    for( int i = 0; i < size; i++ )
    {
        total += s[i];
    }

    if( s[0] <= target_sum && total >= target_sum )
    {
        subset_sum(s, tuple_vector, size, 0, 0, 0, target_sum);
    }

    free(tuple_vector);
}

int main()
{
    int weights[] = {15, 22, 14, 26, 32, 9, 16, 8};
    int target = 53;

    int size = ARRAYSIZE(weights);

    generateSubsets(weights, size, target);

    printf("Nodes generated %d\n", total_nodes);

    return 0;
}

```

[Run on IDE](#)

As another approach, we can generate the tree in fixed size tuple analogs to binary pattern. We will kill the sub-trees when the constraints are not satisfied.

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



Learn to Program in
Python

Python for Data Science
A free online course from Microsoft

edX
www.edx.org

Enroll Now

54 Comments Category: [Backtracking](#) Tags: [Backtracking](#)

About Venki

Software Engineer

[View all posts by Venki](#) →

Related Posts:

- [Fill two instances of all numbers from 1 to n in a specific way](#)
- [Backtracking | Set 8 \(Solving Cryptarithmic Puzzles\)](#)
- [Print all possible paths from top left to bottom right of a mXn matrix](#)
- [Tug of War](#)
- [Backtracking | Set 7 \(Sudoku\)](#)
- [Backtracking | Set 5 \(m Coloring Problem\)](#)
- [Backtracking | Set 3 \(N Queen Problem\)](#)
- [Backtracking | Set 2 \(Rat in a Maze\)](#)

([Login](#) to Rate and Mark)

0

Average Difficulty : **0/5.0**
No votes yet.

☐
☐

Add to TODO List

Mark as DONE

Like Share 9 people like this.

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

54 Comments **GeeksforGeeks**

 **Login** ▾

 Recommend 1

 Share

Sort by Newest ▾



Join the discussion...



Shivam Maharshi • a day ago

```
private static void print(int[] a, int sum, int index, List<integer> list) {  
    if (sum == 0) {  
        printList(list);  
        return;  
    }  
    for (int i = index + 1; i < a.length; i++) {  
        if (a[i] <= sum) {  
            list.add(a[i]);  
            print(a, sum - a[i], i, list);  
            list.remove(list.size() - 1); // This is backtracking.  
        }  
    }  
}
```

^ | ▾ • Reply • Share ▸



pankkum2 • 19 days ago

I think, this can be simpler.

```
/*  
int a[] = {10, 7, 5, 18, 12, 20, 15};  
int i = 0;  
int S = 35;  
int n = 7;  
int b[] = {0,0,0,0,0,0,0};  
*/  
void sumSub(int a[], int i, int S, int n, int b[])  
{  
    if (i > n) return;  
    if (a[i] == S)  
    {  
        b[i] = 1;  
        print(b,n);  
    }  
    else if (S-a[i] > 0)  
    {
```

[see more](#)[^](#) | [v](#) • [Reply](#) • [Share](#) ›**fateh sidhu** • 3 months ago

much simpler and cleaner code (: no fancy stuff

<https://ideone.com/cqPIWA>

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**Klaus** • 4 months ago

Exclude previous added item and consider next candidate.

Please explain what role it is playing ?

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**Billionaire** • 4 months ago

// Exclude previous added item and consider next candidate
subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);"

this is just redundant!

1 [^](#) | [v](#) • [Reply](#) • [Share](#) ›

**Gajanan** • 5 months ago

Hi,

Can someone please explain in the first solution why do we need this recursion:

// Exclude previously added item and consider next candidate
subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);

Even if i remove that, I am getting the result right.

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**superman** • 5 months ago

When we found the sum

-->next iterator should be set to iter not iter+1

-->current sum should be sum-s[ite-1]

as we have to remove the sum obtained by previous element and next iter should be at iter as a next candidate but pos of t array will be decreased by 1.

try for this : {10, 7, 5, 18, 4,12, 20,2, 15},22

Above given code will fail.

Correct Code:

```
if( target_sum == sum ){
```



```

// target_sum = sum, ite = index

```

```

// We found subset
printSubset(t, t_size);

```

```

// Exclude previously added item and consider next candidate
subset_sum(s, t, s_size, t_size-1, sum - s[ite-1], ite, target_sum);

```

```

return;
}

```

^ | v • Reply • Share ›



Rishabh • 6 months ago

hi is this the same problem as <http://www.geeksforgeeks.org/d...> . I mean , my understanding is that both the problems evaluate the subsets which can have sum x? so we have 2 approaches , backtrack and dp ?

^ | v • Reply • Share ›



Klaus → Rishabh • 6 months ago

Yes, but DP approach will not work if the sum is very large because of table construction issues with large sum.

^ | v • Reply • Share ›



Vaibhav Bajpai → Rishabh • 6 months ago

Yes, you are right. These are two different and efficient approaches.

^ | v • Reply • Share ›



Sam_Sparks • 7 months ago

is there a better way to solve this question?????

^ | v • Reply • Share ›



Abhishek → Sam_Sparks • 7 months ago

this is NP-Complete. the best we could do is a pseudo-polynomial solution.

^ | v • Reply • Share ›



Puneet • a year ago

```

public class SubsetSum {

```

```

    public static void main(String[] args) {

```

```

        int[] a = new int[]{1, 1, 1, 1, 1, 1, 2, 3, 1, 1, 4, 5, 2, 1};

```

```

        boolean[] b = new boolean[a.length];

```

```

        subset(a, b, 0, 5);

```

```

}

private static void subset(int[] a, boolean[] b, int index, int sum) {

if (sum == 0) {

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

if (b[i]) System.out.print(a[i] + " ");

}

```

[see more](#)

^ | v • Reply • Share ›



Ankit Bhodia • a year ago

<https://ideone.com/Rdta1P>

I think once we print the subset, we can directly backtrack rather than recursing after the print. I commented the recursive call after print and I think it works fine. Do let me know if I am wrong.

^ | v • Reply • Share ›



Aditya Goel • a year ago

Can someone check why this code is not working -

<http://ideone.com/7lj8XT>

^ | v • Reply • Share ›



ell • a year ago

if the set include negative integers , cant it be implemented with backtrack algorithm ?

2 ^ | v • Reply • Share ›



Nâkshâtrâ Mâhêshwârî → ell • 7 months ago

it works...(Y)

^ | v • Reply • Share ›



Gaurav Gupta • 2 years ago

<http://ideone.com/9lXga4>

using vectors and backtracking. Generates Subsets as well

^ | v • Reply • Share ›



RACHIT SAXENA • 2 years ago

can one find all possible subsets of a big array say no. of elements = 10^5
in less tha $O(n^2)$???

^ | v • Reply • Share ›

^ | v • Reply • Share ›



sujeet singh • 2 years ago

```
void print_subarray_sum(int *arr,int arr_size,int given_sum)
```

```
{
```

```
int sum =0;
```

```
shorting_array(arr,arr_size);
```

```
for(int i =0;i<arr_size;i++) for(int j=i;j<arr_size;j++) {sum=sum+*(arr+j);
if(sum==given_sum) {print_array(arr+i,j-i+1);return;} else if(sum>
given_sum)
```

```
{
```

```
sum =0;break;
```

```
}
```

```
}
```

```
}
```

^ | v • Reply • Share ›



AlienOnEarth • 2 years ago

Another solution without backtracking (using Algorithm of string combination)

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
#include<stdbool.h>
```

```
void print(int arr[], int n)
```

```
{
```

```
int i;
```

```
printf("subset: ");
```

```
for(i=0;i<n;i++) {printf("%d ",arr[i]);printf("\n");} void
printallsubsets(int arr[],int n,int output[],int start,int index,
int sum,int k) {int i="0;" base=" case=" 1:" sum=" subset=" not="
found=" if(k>sum)
```

[see more](#)

1 ^ | v • Reply • Share ›



disqus_0z6aYV2hDC • 2 years ago

Isn't the following wrong;

```
if( target_sum == sum )  
  
{  
  
    // We found subset  
  
    printSubset(t, t_size);  
  
    // Exclude previously added item and consider next candidate  
  
    subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);  
  
    return;  
  
}
```

We called subset_sum with $i = \text{ite} + 1$, and in the loop when $\text{sum} == \text{target_sum}$, we are calling subset_sum with again $i = \text{ite} + 1$. We should have called it with $i = \text{ite}$. Also, the sum passed is $\text{sum} - s[\text{ite}]$. It should be $\text{sum} - s[\text{ite} - 1]$

3 ^ | v • Reply • Share ›



Mayank yadav → disqus_0z6aYV2hDC • a month ago

you are right

^ | v • Reply • Share ›



prashant • 2 years ago

```
int fun(int arr[],int low,int high,int s)
```

```
{  
  
    if(s==0)  
  
        return 1;  
  
    if(s<0)  
  
        return 0;  
  
    if(low==high)  
  
    {  
  
        if(arr[low]==s)
```

```
{
cout<<arr[low]<<" ";return 1;} else return 0;} if(arr[low]==s)
return fun(arr,low+1,high,s);

for(int k=low;k<high;k++) {int p=fun(arr,k+1,high,s-arr[k]); if(p) {cout<<arr[k]
<<" ";return 1;}} return 0;}>
1 ^ | v • Reply • Share ›
```



prashant jha • 2 years ago

/*

```
int fun(int arr[],int low,int high,int s)
```

```
{
```

```
if(s==0)
```

```
return 1;
```

```
if(s<0)
```

```
return 0;
```

```
if(low==high)
```

```
{
```

```
if(arr[low]==s)
```

```
{
```

[see more](#)

^ | v • Reply • Share ›



Ajp • 2 years ago

```
private static void subSetSum(Integer[] arr, Integer[] tArr, int ts,
int sum, int ite, int num) {
```

```
if (num == sum) {
```

```
printSubset(tArr, ts);
```

```
} else if (sum < num) {
```

```
for (int i = ite; i < arr.length; i++) {
```

```
tArr[ts] = arr[i];
```

```
subSetSum(arr, tArr, ts + 1, sum + arr[i], i, num);
```

```
}
```

```
} else if (sum > num) {
```

```
// Nothing do
```

```
}
```

```
}
```

^ | v • Reply • Share ›



C# Rules • 2 years ago

/// Logic is sort numbers and use this logic. Below method assumes that number are already sorted.

```
public void FindSubset(int sum, int tSum, int[] allInts, ArrayList possibbleset, int
currentIndex)
{
    if (sum == tSum) {PrintResult(possibbleset);}
    else if (sum > tSum){
        for (int i = currentIndex; i < allInts.Length; i++)
        {
            ArrayList ps = (ArrayList) possibbleset.Clone();
            if (tSum + allInts[i] <= sum)
            {
                ps.Add(allInts[i]);
                FindSubset(sum, tSum + allInts[i], allInts, ps, (i + 1));
            }
            else break;
        }
    }
}
```

[see more](#)

^ | v • Reply • Share ›



Vijay Kumar • 2 years ago

```
public static void select(int i,int n,int k,String output) {

    if(k==0 && n==0){

        count++;

        System.out.println(output);

        return;

    }

    if(i<1 || n<0 || k==0){return;}

    select(i-1,n,k,output); // not choosing i

    select(i-1,n-i,k-1,output+i + " ");
```

}

1 ^ | v • Reply • Share ›

**Guest** • 2 years ago

#include <stdio.h>

#include <stdlib.h>

void sumOfSubsets(int x[],int w[],int n,int s,int k,int r,int m)

{

int i;

x[k]=1;

if(s+w[k]==m)

{

printf("SOLUTION : ");

for(i=1;i<=n;i++)

{

[see more](#)

^ | v • Reply • Share ›

**Guest** • 2 years ago

I think this is a better solution if we are not allowed to modify the array by sorting it.

#include <stdio.h>

#include <stdlib.h>

void sumOfSubsets(int x[],int w[],int n,int s,int k,int r,int m)

{

int i;

x[k]=1;

if(s+w[k]==m)

{

printf("SOLUTION : ");

for(i=1;i<=n;i++)

{

if(i>k)

{

x[i]=0;

}

```
printf("%d ",x[i]);
```

[see more](#)

^ | v • Reply • Share ›



abhishek08aug • 3 years ago

Intelligent :D

^ | v • Reply • Share ›



rahul23 • 3 years ago

@venki--

Firstly following isnt not needed i think..And it can be modified to handle duplicates if we need this...say 2,4,4 and we need to find 6...2 4 matched..now ite is at 3rd element i.e 4..wta we so....we do sum-s[ite-1],i.e exclude previous and + s[ite]..add next..agin we get 2,4.....correct me if wrng...

// it would not be needed in case unique elements....

```
if( ite + 1 < s_size && sum - s[ite] + s[ite+1] <= target_sum )
```

```
{
```

// Exclude previous added item and consider next candidate

```
subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);
```

```
}
```

```
return;
```

```
}
```

3 ^ | v • Reply • Share ›



bugfinder • 3 years ago

isnt this code finding repeated same subsets which satisfies the constraint.

for example :

consider the tree : ..

```
1
```

```
/ \
```

```
2
```

```
/ \
```

```
6 7
```

suppose 6 leads to a subset ... Then

```
[code][code]
```

```
if( target_sum == sum )
```

```
{
```

// We found sum

```
printSubset(t, t_size);
```


[see more](#)[^](#) | [v](#) • [Reply](#) • [Share](#) ›**Balaji** • 3 years ago

Thanks for explaining how to accomplish this through backtracking. Just to be more clear and to verify my belief - i'm thinking we can model this problem as the coin change problem which essentially asks in how many ways can you make change to a given sum 'N' using 'm' coins $S_1 < S_2 < \dots < S_m$, the solution to which is simply the recurrence $C(N, m) = C(N, m - 1) + C(N - S_m, m)$

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**kalyan** • 3 years ago

```
if( ite + 1 < s_size && sum - s[ite] + s[ite+1] <= target_sum )
{
    // Exclude previous added item and consider next candidate
    subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);
}
```

i think this condition is redundant ?can some one tell me why is it required ...i ran it without that condition its running fine ..

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**Anil** • 3 years ago

I feel below check is redundant **and not** required. Could u please explain the necessity

My point is

if we got subset whose sum is equal to target sum, **then** there is no question of being

```
if(ite + 1 < s_size && sum - s[ite] + s[ite+1] <= target_sum)
{
    subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);
}
```

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**Anil** ➔ Anil • 3 years ago

I am talking about optimized version.

Also in unoptimized version the return statement is redundant.

```
if( target_sum == sum )
{
    // We found subset
    printSubset(t, t_size);
    // Exclude previously added item and consider next candidate
    subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);
    return; // AKT the return is not required redundant code
}
```

^ | v • Reply • Share ›



dreamer → Anil • 3 years ago

exactly my point.. this statement is not required.. the exclusion statement... it doesnt make any difference .. plus we should exclude "s[ite-1]" here.. coz it is the previously added item.. wasted hours figuring out why we are excluding s[ite] when we haven't included it yet

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



^ | v • Reply • Share ›



Mayautobot • 3 years ago

//this program prints all subsets in the array that sum to the given sum

```
#include<iostream>
```

```
using namespace std;
```

```
void Tour(int arr[],int size,int reSum)
```

```
{
```

```
    int *stack=new int[size];
```

```
    int sp=-1;
```

```
    int sum=0;
```

```
    int arrip;
```

```
    for(int i=0;i<size;i++)
```

```
    {
```

```
        sp=-1;
```

```
        stack[++sp]=i;
```

```
        sum+=arr[stack[sp]];
```

[see more](#)

^ | v • Reply • Share ›



spark9 • 4 years ago

your code will not work properly when the set will contain 0.

like

[1,2,0] target sum = 3.

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

^ | v • Reply • Share ›



kartik → spark9 • 4 years ago

@spark9: I tried the code for your input. It seems to be working. It produced two subsets as output:

1 2 0

1 2

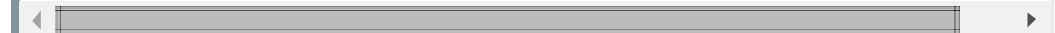
^ | v • Reply • Share ›



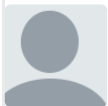
spark9 → kartik • 4 years ago

Did you try with the first version of the code (the one without sorting) ?

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



^ | v • Reply • Share ›



sachin • 4 years ago

in the first code,

what if sum becomes greater than the target sum, then in that case also, it should exclude current element and consider next?

^ | v • Reply • Share ›



Venki • 4 years ago

@Arvind, thanks for pointing this. You have provided a set with duplicated entries. It was assumed that the elements are unique.

With the duplicated entries, during branch generation upon step back by excluding the current element it is assumed that the current element is excluded to consider next elements. In case of duplicate elements it is not true.

I have updated the problem description.

^ | v • Reply • Share ›



Arvind • 4 years ago

The code doesn't work Example:

For

```
int weights[] = {10, 7, 5, 18, 12, 12, 15};
```

Output

18 7 10

18 7 12

18 5 12

18 5 12

Nodes generated 55

Which is not right

^ | v • Reply • Share ›



Arvind → Arvind • 4 years ago

```
if( target_sum == sum )
{
    // We found sum
    printSubset(t, t_size);

    // constraint check
    if( ite + 1 < s_size && sum - s[ite] + s[ite+1] <= target_sum )
    {
        // Exclude previous added item and consider next candidate
        subset_sum(s, t, s_size, t_size-1, sum - s[ite], ite + 1, target_sum);
    }
    return;
}
```

This is **not** right. why **do** we **do** Exclude previous added item **and** consider next

Output for {10, 7, 5, 18, 12, 12, 15};

10 7 18 (sum = 35)

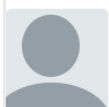
10 7 12 (sum = 29)

5 18 12 (sum = 35)

5 18 12

Nodes generated 124

^ | v • Reply • Share ›



Arvind • 4 years ago

The code doesn't work

^ | v • Reply • Share ›

  • Reply • Share ›**kamlesh** • 4 years ago

what is the time complexity of Backtracking Algorithm for Subset Sum

  • Reply • Share ›**Sambasiva** • 4 years ago<http://effprog.blogspot.com/20...>  • Reply • Share ›**Venki** → Sambasiva • 4 years ago

@Sambasiva, thanks for the comment. We are missing your comments on GFG :). Welcome back.

  • Reply • Share ›

Load more comments



Subscribe



Add Disqus to your site Add Disqus Add



Privacy

@geeksforgeeks, Some rights reserved

[Contact Us!](#)[About Us!](#)[Advertise with us!](#)