Subset Sum | Backtracking-4

Difficulty Level : Hard • Last Updated : 06 Jul, 2021

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).

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

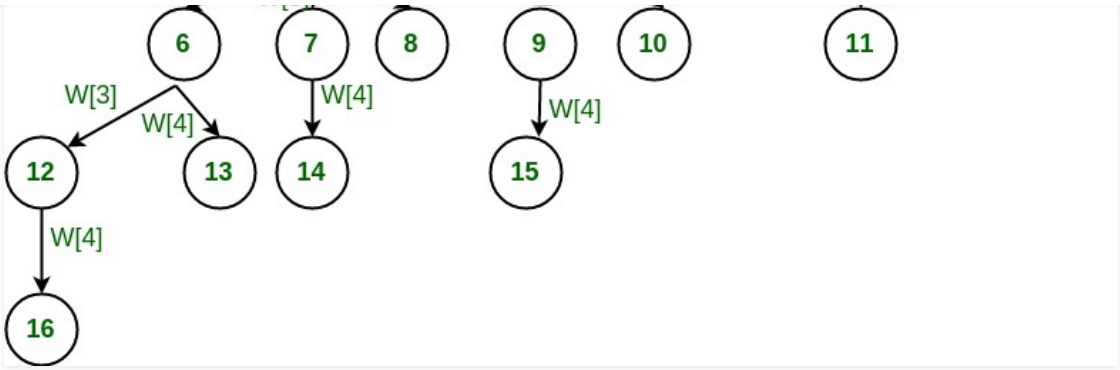
Exhaustive Search Algorithm for Subset Sum

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] ... 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

```
    generate the nodes of present level along breadth of tree and
    recur for next levels
```

Following is the 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.

Spelling mistake of

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).

C++

```
#include <bits/stdc++.h>
using namespace std;

#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++)
```

```
    int *lhs = (int *)pLhs;
    int *rhs = (int *)pRhs;
    return *lhs > *rhs;
}

// inputs
// s            - set vector
// t            - tuplet vector
// s_size       - set size
// t_size       - tuplet 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, targ
        }
        return;
    }
    else
    {

        // constraint check
        if( ite < s_size && sum + s[ite] <= target_sum )
        {
```

```
                    }
                }
            }
        }
    }

// Wrapper that prints subsets that sum to target_sum
void generateSubsets(int s[], int size, int target_sum)
{
    int *tuplet_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, tuplet_vector, size, 0, 0, 0, target_sum);
    }
    free(tuplet_vector);
}

// Driver code
int main()
{
    int weights[] = {15, 22, 14, 26, 32, 9, 16, 8};
    int target = 53;
    int size = ARRAYSIZE(weights);
    generateSubsets(weights, size, target);
    cout << "Nodes generated " << total_nodes;
    return 0;
}

//This code is contributed by shivanisinghss2110
```

```
// 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            - tuplet vector
// s_size       - set size
// t_size       - tuplet 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 < 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,
                }
            }
        }
    }
}

// Wrapper that prints subsets that sum to target_sum
void generateSubsets(int s[], int size, int target_sum)
{
    int *tuplet_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, tuplet_vector, size, 0, 0, 0, target_sum);

    }
```

```
    generateSubsets(weights, size, target);

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

    return 0;
}
```

Output:

```
 8 9 14 22n 8 14 15 16n 15 16 22nNodes generated 68
```

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.

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Find maximum subset sum formed by partitioning any subset of array into 2 partitions with equal sum**
08, Apr 20

02 **Sum of maximum and minimum of Kth subset ordered by increasing subset sum**
27, Sep 20

03 **Largest possible Subset from an Array such that no element is K times any other element in the Subset**
28, Jul 20

04 **Maximum size of subset such that product of all subset elements is a factor of N**
27, Sep 21

05 **Largest subset having with sum less than equal to sum of respective indices**
14, Jul 20

06 **Maximum Subset Sum possible by negating the entire sum after selecting the first Array element**
05, Aug 20

07 **Subset Sum Problem in O(sum) space**
17, Aug 17

08 **Subset sum problem where Array sum is at most N**
11, Apr 22

Article Contributed By :

Improved By : samrat2825, shivanisinghss2110, saintist, jay8kaku, sumitgumber28

Article Tags : Adobe, Amazon, Drishti-Soft, subset, Backtracking

Practice Tags : Amazon, Adobe, Drishti-Soft, subset, Backtracking