Sum of Subsets Problem (SSP)

Also known as "*Subset Sum*" is a classical optimization problem and NP-complete that can be thought of as a special case of the Knapsack problem.
The *sum of subsets* problem consists of finding a subset of a given set $X = \{x_1, x_2, \dots, x_n\}$ of n distinct positive integers and a positive integer S. Find all subsets of $\{x_1, x_2, \dots, x_n\}$ that sum to S. For example,
if $X = \{3, 5, 6, 7, 8, 9, 10\}$ and $S = 15$, there will be more than one subset whose sum is 15. The subsets are $\{7, 8\}, \{3, 5, 7\}$, $\{6, 9\}$, and $\{5, 10\}$. On the other hand, if $X = \{1, 5, 6, 7, 11, 12, 13\}$ and $S = 15$, there will be no subset that adds up to 15.
We will assume a binary state space tree. The nodes at depth 1 are for including ($true\ or\ yes\ =\ 1$, $false\ or\ no\ =\ 0$) item 1, the nodes at depth 2 are for item 2, etc. The left branch includes x_i, and the right branch exclude x_i. The nodes contain the sum of the numbers included so far.

Time and Space Complexity Background

The amount of workspace that is needed for an algorithm is measured either in the unit-cost RAM model, or the Turing machine model, where tapes are infinite on both sides. In 1957, both Danzig and Bellman used the unit-cost RAM model for measurement and the method of Dynamic Programming (DP) to run the SSP in time $O(mt)$ and space $O(t)$. This is pseudo-polynomial in m that is if the value of t is bounded by a polynomial in m, then it runs in polynomial time and space. In 1975, Ibarra and Kim were able to give a Fully Polynomial Time Approximation Scheme for the Sum-of-Subsets optimization problem with running time $O(m/\epsilon^2)$ and space $O(m + 1/\epsilon^3)$, where Gens and Levner had another Fully Polynomial Time Approximation scheme algorithm that runs in time $O(m/\epsilon\ + 1/\epsilon^3)$. Both Fully Polynomial Time Approximation Scheme algorithms were done by using the method of DP and measured in the unit-cost RAM model. In 2010, Elberfeld, Jakoby, and Tantau were able to be obtained logspace upper bounds by using unary encodings of the inputs in the Unary Subsets Sum problem. The Unary Subsets Sum problem is the same as the SSP except where inputs are binary instead of decimal. Later that year, Kane was able to simply the logspace algorithm. In 2012, Elberfeld, Jakoby, and Tantau were able to improve the Unary Subsets Sum under the appropriate encodings. Elberfeld, Jakorby, Tantau, and Kane's algorithms later became helpful for solving the optimization version of SSP problem which runs in time polynomial in n, t, and space bound $O(logn + logt)$ which is measured on the Turing machine model.

Backtracking Strategy

Backtracking consists of doing a *Depth First Search* of the state space tree, checking where each node is *promising* and if the node is *non-promising* backtracking to the node's parent. We call a node nod-promising if it cannot lead to a feasible (or optimal) solution, otherwise it is promising. The state space tree consisting of expanded nodes only is called the *pruned* state space tree. In a full binary of depth n, the worst-case time complexity is $O(2^n)$.

Cryptosystems using Sum of Subsets Problem in Real-Life Application

Merkle and Hellman were the first to consider the use of SSP in a public key protocol. The SSP based public key cryptosystem, a set of S of size n is calculated by utilizing a private key. The private key is distributed as a public key along with some other certain scheme parameters. The sender selects a subset in S distinctively related to a plaintext M through a binary encoding, then calculates the sum-of-subset $W = \sum_{s_i} \in S's_i$ and sends W as the ciphertext. The receiver with the private key translates the Sum of subset instance (S, W) to an efficiently solvable problem and recovers the plaintext M by solving it.
Any Sum of Subset or ciphertext W should not have two different subsets related to it, as in that case, distinct decryption would be impossible

SSP Example: $X = \{3, 5, 6, 7\}$ and $S = 15$. There are 15 nodes in the pruned state space tree shown in the figure below. The full state space tree has 31 nodes, $2^{4+1} - 1$. Trace of the Algorithm is also shown in the table below. Nodes of the are shown by circled numbers. Notice that x_i is included at level 1 of the tree, x_2 at level 2, and so on. We have only one solution with subset $\{3, 5, 7\}$, which occur at node 6.

Consider a node at depth i
$weight$ = the sum of node – that is, sum of numbers included in partial solution node
$total$ = sum of the remaining items $i + 1$ to n (for a node at depth i)
$include$ = a boolean array
x = an integer array sorted in nondecreasing order
S = target value

A node at depth i is non-promising:
if weight+total<S OR weight+x(i+1)>S

Sum-of-Subsets Algorithm

```
Procedure SumOfSubsets (i, weight, total)
        integer i, weight, total, x (1 : i), S
        boolean include (1 : n)

        if i is promising then      //may lead to solution//
                if weight = S then
                        print (include (1 : i))

        else    //expand the node when  weight is less than S//
                include (i + 1) = "true"
                SumOfSubsets (i + 1, weight + x (i + 1), total – x (i + 1))
                include (i + 1) = "false"
                SumOfSubsets (i + 1, weight, total – x (i + 1))
end SumOfSubsets
```

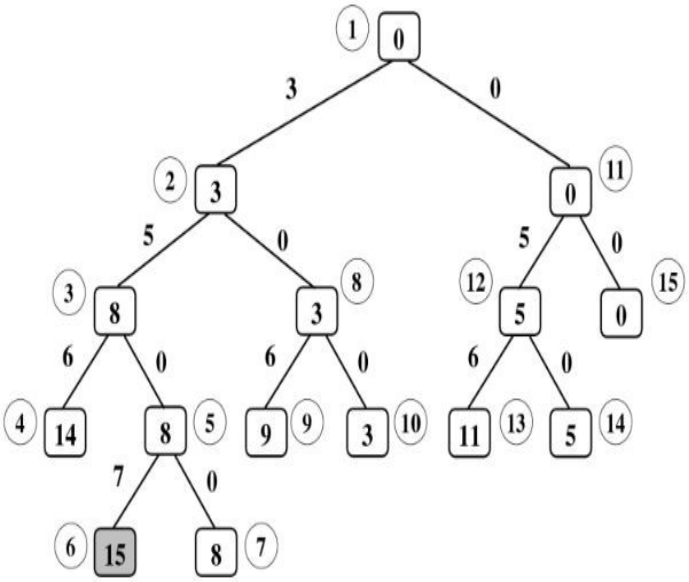
Promising Algorithm

```
Function promising (i)
        integer i, weight, total, x (1 : i), S

        if weight + total >= S AND (weight = S OR weight + x (i + 1) <= S) then
                return (true)

        else
                return (false)
end promising
```

Pruned State Space Tree



Trace of Algorithm

i	*node*	*weight*	*total*	*Comment*
0	1	0	21	Start
1	2	3	18	include 3
2	3	8	13	include 5
3	4	14	7	include 6
3	5	8	7	exclude 6
4	6	15	0	include 7
			3 5 7	← Solution
4	7	8	0	exclude 7
2	8	3	13	exclude 5
3	9	9	7	include 6
3	10	3	7	exclude 6
1	11	0	18	exclude 3
2	12	5	13	include 5
3	13	11	7	include 6
3	14	5	7	exclude 6
2	15	0	13	exclude 5