The Sum of Subset Problem

In this problem, we are given a vector of N values, called weights. The weights are usually given in ascending order of magnitude and are unique. For example, \( W = (2, 4, 6, 8, 10) \) is a weight vector. We are also given a value \( M \), for example 20. The problem is to find all combinations of the weights that exactly add to \( M \). In this example, the weights that add to 20 are: \( (2, 4, 6, 8) \); \( (2, 8, 10) \); and \( (4, 6, 10) \). Solutions to this problem are often expressed by an N-bit binary solution vector, \( X \), where a 1 in position \( i \) indicates that \( W_i \) is part of the solution and a 0 indicates it is not. In this manner the three solutions above could be expressed as: \( (1,1,1,0) \); \( (1,0,1,1) \); \( (0,1,1,0) \).

One way to solve this problem is by backtracking. We create a binary tree where level \( i \) refers to weight \( W_i \). Each node has two branches out of it, one labeled with a 0, the other labeled with a 1. The 0 branch means that the weight represented by the child is NOT included in the solution, and the 1 branch means that the weight represented by the child IS included. That is, if the right branch is the 0 branch and the left branch is the 1 branch, then the vector \( (1, 0, 1) \) means from the root, take the left branch, then the right branch, and then the left branch. Thus, the process of finding all the solutions is to traverse this tree and to print out the path from the root for all nodes for which the accumulated sum is equal to \( M \).

A backtracking solution is to traverse this tree in a depth-first manner, but, when it is clear that the current branch cannot possibly reach a solution, abandon the branch, backup to the parent and try the next branch.

While there are many ways to test if a branch should be abandoned, a simple test is the following. Let \( S \) be the accumulated sum thus far and suppose that we are at level \( k \). Because the weights are given in ascending order of magnitude, if \( S + W_k > M \), then no solution is possible along this branch. The test used to determine if a solution is still possible is often called a bounding function. The inputs to a bounding function always include the nodes on the current path: the path from the root to the current node.

Recursive version

In the recursive form of the algorithm, to backtrack means to not make a recursive call.

Let \( W = (w_1, w_2, \ldots, w_n) \) be the weight vector where \( w_i < w_{i+1} \)

Let \( X = (x_1, x_2, \ldots, x_n) \) be an n-bit binary vector, the solution vector. The vector \( X \) is initialized to \( (0, 0, \ldots, 0) \), the 0 vector.

Let \( M \) be an integer where \( M < w_1 + w_2 + \ldots + w_n \), the target sum.

Let \( k \) be an integer index variable.

Let \( S \) be an integer that represents the accumulated sum.
\[ S = w_1x_1 + w_2x_2 + \ldots + w_kx_k, \] where \( k \leq n \).
Algorithm \texttt{recursiveSubsetSum}(\text{integer } S, \text{integer } k) \\

set $X_k = 1$ //Try one branch of the tree \\

if( $S + W_k = M$ ) \\
    print vector $X$ //We have a solution \\
else if( $(k + 1 \leq n) \text{ AND } (s + W_k \leq M)$) \\
    call recursiveSubsetSum( $s + W_k$, $k + 1$ ) \\
    //A solution is still possible, so continue along this branch \\

if( $(k + 1 \leq n) \text{ AND } (s + W_{k+1} \leq M)$) \\
    set $X_k = 0$ //Now try the other branch \\
    call recursiveSubsetSum( $s$, $k + 1$ )

The initial call of the algorithm is: \texttt{recursiveSubsetSum}( 0, 1 )

Iterative version

In the iterative version, to backtrack mean to decrement $k$ until $X_k = 1$. Then $X_k$ is set to 0. If $k$ is decremented to 0, the algorithm is finished.

Algorithm \texttt{subsetSum}(\text{vector } W, \text{integer } M) \\

local vector $X$ initialized to 0 \\
local integer $i$ \\
local integer $s$ initialized to 0 \\
local integer $k$ initialized to 0 \\
local integer $n$ initialized to the length of $W$ \\
local Boolean done initialized to false \\

while( not done ) \\
    set $k = k + 1$ \\
    if( $k > n$ ) \\
        call algorithm Backtrack \\
    else \\
        set $x_k = 1$ \\
        set $s = 0$ \\
        for $i = 1$ to $k$ do \\
            set $s = s + w_i \times x_i$ //calculate the current value of $s$ \\
        if( $s = M$ ) \\
            print vector $X$ //we have a solution! \\
        if( $x > M$ ) \\
            call algorithm Backtrack
Algorithm Backtrack

//Backtracking means to set the component in the solution vector to 0 for the current value of k. Then decrement k until we find a component of the solution vector that is non-zero. If k is less than 1, we have traversed the entire tree, so the Boolean value done is set to true. Otherwise, the non-zero component of the solution vector is set to zero.

//Uses integer variables k and n, vector X, and sets Boolean variable done from Algorithm subsetSum

if( k ≤ n )
  set x_k = 0

set k = k - 1

if( x_k = 1 )
  set x_k = 0
else
  while( (k > 0) AND (x_k = 0 )
    set k = k - 1

if( k > 0 )
  set x_k = 0
else
  set done = true