Subset Sum Problem Revisited

Problem
We are given a positive integer \( t \) and a sequence \( A = \langle a_1, a_2, \ldots, a_n \rangle \) of (not necessarily distinct) \( n \) positive integers. We want to find out whether some subsequence of \( A \) sums to \( t \).

Dynamic Programming Solution
For \( 1 \leq i \leq n \) and \( 0 \leq v \leq t \), define \( m(i, v) \) to be

\[
m(i, v) = \begin{cases} 
  \text{true} & \text{if some subsequence of } \langle a_1, a_2, \ldots, a_i \rangle \text{ sums to } v \\
  \text{false} & \text{otherwise.}
\end{cases}
\]

We seek \( m(n, t) \).

Optimal Substructure Property
Clearly \( m(i, 0) = \text{true} \) for all \( 1 \leq i \leq n \).

Now let \( i, v \) be positive integers. Suppose the sequence \( \langle a_1, a_2, \ldots, a_i \rangle \) contains a subsequence \( \langle a_{j_1}, a_{j_2}, \ldots, a_{j_k} \rangle \) that sums to \( v \).

Case 1: \( j_k = i \). Then the sequence \( \langle a_1, a_2, \ldots, a_{i-1} \rangle \) contains the subsequence \( \langle a_{j_1}, a_{j_2}, \ldots, a_{j_{k-1}} \rangle \) that sums to \( v - a_i \).

Case 2: \( j_k \neq i \). Then the sequence \( \langle a_1, a_2, \ldots, a_{i-1} \rangle \) contains the subsequence \( \langle a_{j_1}, a_{j_2}, \ldots, a_{j_k} \rangle \) that sums to \( v \).

This gives us the following recurrence.

Recurrence

\[
m(i, v) = \begin{cases} 
  \text{false} & \text{if } i = 0 \text{ or } v < 0 \\
  \text{true} & \text{if } i \geq 1, v = 0 \\
  m(i - 1, v) \lor m(i - 1, v - a_i) & \text{if } i \geq 1, v > 0
\end{cases}
\]

In this recurrence, \( i \) is allowed to be 0, and \( v \) is allowed to be negative, i.e., \( 0 \leq i \leq n \) and \( v \leq t \). The base case of “\( i = 0 \) or \( v < 0 \)” is artificial, introduced to simplify the recurrence.
Subset Sum Algorithm

Step 1. Fill in a table of $m(\cdot, \cdot)$ values. We can fill the table row-by-row or column-by-column, with both the row and column indices increasing.

Step 2. Return the value $m(n, t)$ as answer.

Remark on Implementation
Even though the recurrence seems to require an infinite space to implement (because $v$ can be any negative integer), we only need a table of size $n(t + 1)$ to store values of $m(i, v)$ for all $1 \leq i \leq n$ and $0 \leq v \leq t$. We implement a function for accessing $m(i, v)$. When an access$(i, v)$ is requested, we check whether $v < 0$ or not. If it is, we return false; otherwise we access the table as usual.

Running Time
Step 1 fills out each table entry in $O(1)$ time, $O(nt)$ time total.
Step 2 takes $O(1)$ time.

Remarks

1. Artificial base cases always help simplify recurrences. For example, without it the Subset Sum Problem recurrence will have to be written like this:

$$m(i, v) = \begin{cases} 
\text{true} & \text{if } v = 0 \\
\text{true} & \text{if } i = 1, v > 0, v = a_1 \\
\text{false} & \text{if } i = 1, v > 0, v \neq a_1 \\
m(i - 1, v) & \text{if } i > 1, v > 0, a_i > v \\
m(i - 1, v) \lor m(i - 1, v - a_i) & \text{if } i > 1, v > 0, a_i \leq v 
\end{cases}$$

In this recurrence, $1 \leq i \leq n$ and $0 \leq v \leq t$.

2. If the set of numbers that adds up to the target $t$ is desired, we can get them from the filled-out $M[\cdot, \cdot]$ table directly without having to use an optimizer table.

3. Instead of asking whether some subset of $A$ summing to $t$ exists, we can instead ask how many subsets sum to $t$.

4. Subset Sum is an NP-complete problem. Dynamic programming solves it in pseudopolynomial time.
5. The 0-1 Knapsack Problem is similar to the Subset Sum Problem. (See CLRS p.425–426.)