San Skulrattanakulchai
Gustavus Adolphus College

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 $1 \le i \le n$ and $0 \le v \le t$, define $m(i, v)$ to be

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

We seek $m(n, t)$.

Optimal Substructure Property Clearly $m(i, 0) = \texttt{true}$ for all $1 \le i \le n$.
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: $j_k \ne i$. 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

$$m(i,v) = \begin{cases} \texttt{true} & \text{if } v = 0 \\ \texttt{true} & \text{if } v > 0,\ i = 1,\ v = a_1 \\ \texttt{false} & \text{if } v > 0,\ i = 1,\ v \ne a_1 \\ m(i-1, v) & \text{if } v > 0,\ i > 1,\ a_i > v \\ m(i-1, v) \vee m(i-1, v - a_i) & \text{if } v > 0,\ i > 1,\ a_i \le v \end{cases}$$

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.

Running Time

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

Step 2 takes $O(1)$ time.

Notes

1. In case of positive answer, 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.

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

3. Subset Sum is an NP-complete problem. Dynamic programming solves it in *pseudopolynomial time*.

4. The 0-1 Knapsack Problem is similar to the Subset Sum Problem.