Dynamic Programming Class Notes

Weighted Interval Selection, Subset Sum, and Knapsack

Algorithms & Data Structures

November 4, 2026

Today's Agenda

- **More Dynamic Programming Overview**
- **Maximal Weight Interval Selection** (Minimum Weight Activity Selection variant)
- **Subset Sum Problem**
- **0/1 Knapsack Problem**

1 Maximal Weight Interval Selection

1.1 Problem Formulation

- **Input:** A collection of n intervals, where each interval i has a start time s_i, finish time f_i, and an associated valuation/weight $v_i > 0$.
- **Output:** A non-overlapping collection of intervals that maximizes the total valuation.

1.2 Preprocessing & Notation

1. **Sorting:** Sort all n intervals by non-decreasing end time ($f_1 \leq f_2 \leq \cdots \leq f_n$). Let $v_1, v_2, \ldots, v_n$ be the valuations in this sorted order.
2. **Precomputing Compatibility:** Define $p(i)$ as the index of the last interval (in sorted order) that does not overlap with interval i. That is, $p(i)$ is the largest index $j < i$ such that $f_j \leq s_i$ (or 0 if no such interval exists).

1.3 Recurrence Relation

Let $\text{Opt}(i)$ be the optimal total value obtained using a subset of the first i intervals.

Base Cases:

$$\text{Opt}(0) = 0, \quad \text{Opt}(1) = v_1$$

Recursive Step: For $i \geq 2$, when considering interval i, we have two choices:

- **Exclude interval i:** The optimal value is Opt$(i-1)$.
- **Include interval i:** We gain value v_i and must add the optimal value of non-overlapping previous intervals, Opt$(p(i))$.

$$\text{Opt}(i) = \max \begin{cases} \text{Opt}(i-1) & \text{if we do not include the } i\text{-th interval} \\ v_i + \text{Opt}(p(i)) & \text{if we do include the } i\text{-th interval} \end{cases}$$

Runtime Complexity Note: A direct recursive implementation of this recurrence evaluates overlapping subproblems repeatedly, resulting in an exponential worst-case runtime of $\Omega(2^n)$. Using Dynamic Programming (memoization or iterative bottom-up table filling) reduces the runtime to $O(n \log n)$ total time (dominated by sorting and binary search for $p(i)$).

1.4 Iterative DP Algorithm & Solution Reconstruction

Algorithm 1 Maximal Weight Interval Selection DP

Input: Sorted intervals $1 \ldots n$, valuations $v_1 \ldots v_n$, compatibility array $p[1 \ldots n]$
Initialize array $W[0 \ldots n]$
$W[0] \leftarrow 0$
$W[1] \leftarrow v_1$
for $i = 2$ to n **do**
    $W[i] \leftarrow \max(W[i-1],\ v_i + W[p[i]])$
end for

Reconstructing Optimal Solution Set:
Output $\leftarrow []$
$i \leftarrow n$
while $i > 0$ **do**
    if $W[i] == W[i-1]$ **then**
        $i \leftarrow i - 1$ ▷ Interval i was not selected
    else
        Add i to Output ▷ Interval i was selected
        $i \leftarrow p[i]$
    end if
end while
return Output

2 Subset Sum Problem

2.1 Problem Formulation

- **Given:** A set of positive integers $x_1, x_2, \ldots, x_n$ and a target value T.
- **Output:** A collection of numbers from $\{x_1, \ldots, x_n\}$ that sums exactly to T, if such a subset exists.

2.2 Recurrence Relation

Let $\text{Opt}(i, j)$ be a boolean indicator defined as:

$$\text{Opt}(i,j) = \begin{cases} \text{True} & \text{if a subset of } \{x_1, \ldots, x_i\} \text{ can sum to } j \\ \text{False} & \text{otherwise} \end{cases}$$

Base Cases:

$$\text{Opt}(0,0) = \text{True}, \quad \text{Opt}(0,j) = \text{False} \quad \text{for all } j > 0$$

Recurrence: For $i \geq 1$ and $j \geq 0$:

$$\text{Opt}(i,j) = \text{Opt}(i-1,j) \ \vee \ \begin{cases} \text{Opt}(i-1, j-x_i) & \text{if } j \geq x_i \\ \text{False} & \text{if } j < x_i \end{cases}$$

2.3 Dynamic Programming Table & Reconstruction

We utilize an $(n+1) \times (T+1)$ array W, where entry $W[i,j]$ corresponds to $\text{Opt}(i,j)$.

Algorithm 2 Subset Sum Dynamic Programming & Backtracking

Input: Array of positive integers $x_1 \dots x_n$, target T
Create boolean matrix $W[0 \dots n,\ 0 \dots T]$
$W[0,0] \leftarrow$ True
for $j = 1$ to T **do**
    $W[0,j] \leftarrow$ False
end for
for $i = 1$ to n **do**
    for $j = 0$ to T **do**
        $W[i,j] \leftarrow W[i-1,j]$
        if $j \geq x_i$ **then**
            $W[i,j] \leftarrow W[i,j] \ \vee \ W[i-1,\ j-x_i]$
        end if
    end for
end for

if $W[n,T] ==$ False **then**
    return No valid subset sums to T
else
    Output $\leftarrow$ []
    $i \leftarrow n$, $j \leftarrow T$
    while $i > 0$ **do**
        if $W[i,j] == W[i-1,j]$ **then**
            $i \leftarrow i-1$ ▷ Element x_i was omitted
        else
            Add x_i to Output ▷ Element x_i was included
            $j \leftarrow j - x_i$
            $i \leftarrow i-1$
        end if
    end while
    return Output
end if