lecture 14

- Subset sum
- Knapsack

Resources for this lecture

- I used Kleinberg & Tardos
  Ch. 6.4
- Also see Roughgarden
  Week 3

We have a machine (resource) that can do only one task at a time. Task $i$ takes time $w_i$, which tasks should we do?

\textbf{Problem 1.} Maximize the number of tasks that can be completed in time $W$.

This is similar to interval scheduling but now we only have durations, not start and finish times.

\begin{align*}
\text{task index} & \quad 1 \\
& \quad 2 \\
& \quad 3 \\
& \quad 4 \\
& \quad 5 \\
& \quad 6 \\
\text{duration is } w_i & \quad \{ \text{total available time is } W \}
\end{align*}

\textbf{Problem 1 (restated)}

Given a set of $N$ items with weights $w_i \geq 0$ and given a bound $W$, find the largest subset $S \subseteq \{1, 2, \ldots, N\}$ of items such that $\sum_{i \in S} w_i \leq W$. 

Page dimensions: 612.0x792.0
Greedy approach

- order the intervals by increasing $W_i$
- Find the largest $k$ such that
  \[
  \sum_{j=1}^{k} W_i \leq W
  \]

Proof that greedy finds optimal solution:

By contradiction:

Let greedy choose items $\{i_1, i_2, \ldots, i_k\}$

Assume there exists a subset with $k+1$ items $S = \{i_1, i_2, i_3, \ldots, i_k, i_{k+1}\}$ in increasing sequence such that

\[
\sum_{j=1}^{k+1} W_{i_j} \leq W
\]

But then the greedy solution would not have stopped after $k$, since

\[
\sum_{j=1}^{k} W_{i_j} + W_{i_{k+1}} \leq \sum_{j=1}^{k+1} W_{i_j} \leq W
\]

assumed

Thus, the assumed sequence $\{i_j\}$ cannot exist. (contradiction)

Q: Why does greedy work?

A:

Intuitively, choosing the smallest $w_i$ leaves the most remainder.

How to prove it mathematically?

Problem 2 ("subset sum")

Find the subset $S \subseteq \{1, 2, \ldots, N\}$ that maximizes

\[
\sum_{i \in S} W_i
\]

such that

\[
\sum_{i \in S} W_i \leq W
\]
Example:
\[ w_1 = 1, \ w_2 = 1, \ w_3 = 9, \ w_4 = 9 \]
\[ W = 18 \]

Problem 1 \[ \Rightarrow S = \{ w_1, w_2, w_3 \} \]

Problem 2 \[ \Rightarrow S = \{ w_3, w_4 \} \]

Question: how many subsets of \( \{ 1, 2, 3, 4, \ldots, N \} \) are there?

Answer:
\[ 2^N = 2 \times 2 \times \cdots \times 2 \]

In each element is either in or out.

To solve Problem 2 efficiently, we must avoid the exponential number of subsets.

Define:
\[
\text{Opt}(N, W) \equiv \max \left\{ \sum_{i \in S} w_i : \sum_{i \in S} w_i \leq W \right\}
\]

Dynamic programming: how to break this problem into smaller problems?

"Smaller"? \[ \Rightarrow \text{reduce } N \text{ or } W \]

If \( w_i > W \) \ [then we can't use i]

\[
\text{Opt}(i, W) = \text{Opt}(i-1, W)
\]

Else
\[
\text{Opt}(i, W) = \max \{ \text{Opt}(i-1, W), \ w_i + \text{Opt}(i-1, W - w_i) \}
\]
Exercise

\( w_1 = 2, w_2 = 3, w_3 = 3, W = 6 \)

Find \( \text{Opt}[i][\omega] \)

\[
\begin{array}{c|cccccc}
\omega & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
w_i & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\
 & 0 & 0 & 2 & 4 & 4 & 4 & 4 \\
 & 0 & 0 & 2 & 3 & 4 & ? & ? \\
\end{array}
\]

Given the table \( \text{Opt}[i][\omega] \)

find \( S = \{1, 2, \ldots, N^3\} \) such that

\[ \sum_{j \in S} w_j = \text{Opt}[N][W] \]

\[
\begin{array}{c|cccccc}
\omega & 0 & 1 & 2 & 3 & 4 & 5 & 6 = W \\
\hline
w_i & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\
 & 0 & 0 & 2 & 4 & 4 & 4 & 4 \\
 & 0 & 0 & 2 & 3 & 4 & 5 & ? \\
\end{array}
\]

\[
\begin{array}{c|cccccc}
\omega & 0 & 1 & 2 & 3 & 4 & 5 & 6 = W \\
\hline
w_i & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & 0 & 0 & 2 & 2 & 2 & 2 & 2 \\
 & 0 & 0 & 2 & 4 & 4 & 4 & 4 \\
 & 0 & 0 & 2 & 3 & 4 & 5 & ? \\
\end{array}
\]

\[
S = 2 + w_3 = 5 \neq 4 \therefore w_3 \notin S
\]
2 = 0 + \text{W}_2, \quad 2 = \text{W}_2

Thus,

either solution works.

\textbf{Claim:} running time and space required is \( O(NW) \).

\textbf{Exercise:} what if we used a recursive approach instead?

\underline{Knapsack}

Given a set of \( N \) items with weights \( \text{w}_i \) and values \( \text{v}_i \), and given a bound \( W \) on the total weight (as before), find a subset \( S \) of the items such that \( \sum_{i \in S} \text{w}_i \leq W \) (as before) and \( \sum_{i \in S} \text{v}_i \) is maximized.

\textbf{e.g.}

\begin{align*}
\text{bar of gold} & \quad \text{large} & \quad \text{large} \\
\text{brick} & \quad \text{large} & \quad \text{small} \\
\text{stack of \$10,000 bills} & \quad \text{small} & \quad \text{large} \\
\text{stack of Canadian Tire bills} & \quad \text{small} & \quad \text{small}
\end{align*}
Algorithm for knapsack is identical to that of subset sum, except for that minor change in the recurrence.

Time and space are again $O(NW)$.

**Subtlety:**
- $N$ is the number of elements in $\{w_1, w_2, \ldots, w_N\}$.
- $W$ is a number (always one number).

In theoretical computer science, one expresses the time or space used in a computation (algorithm) in terms of the "size" (memory used) of the input.

*eg.* sorting $N$ numbers

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$\vdots$</th>
<th>$x_N$</th>
</tr>
</thead>
</table>

But for subset sum (or knapsack), we have

The $W$ in $O(NW)$ doesn't refer to the size of the input.

Exercise (Advanced): how to reconcile this?