The Greedy Method
Outline and Reading

- The Greedy Method Technique (§5.1)
- Fractional Knapsack Problem (§5.1.1)
- Task Scheduling (§5.1.2)
The greedy method is a general algorithm design paradigm, built on the following elements:

- **configurations**: different choices, collections, or values to find
- **objective function**: a score assigned to configurations, which we want to either maximize or minimize

This does not always find the optimal solution. Problems for which this method works are said to have the **greedy-choice** property:

- A globally-optimal solution can always be found by a series of local improvements from a starting configuration.

Just because it doesn’t find an optimal solution, doesn’t mean that it isn’t useful...

- Approximation algorithms
Making Change

Problem: A dollar amount to reach and a collection of coin amounts to use to get there.

Configuration: A dollar amount yet to return to a customer plus the coins already returned

Objective function: Minimize number of coins returned.

Greedy solution: Always return the largest coin you can

Example 1: Coins are valued $.32, $.08, $.01
  - Has the greedy-choice property, since no amount over $.32 can be made with a minimum number of coins by omitting a $.32 coin (similarly for amounts over $.08, but under $.32).

Example 2: Coins are valued $.30, $.20, $.05, $.01
  - Does not have greedy-choice property, since $.40 is best made with two $.20’s, but the greedy solution will pick three coins (which ones?)
The Fractional Knapsack Problem

Given: A set $S$ of $n$ items, with each item $i$ having
- $b_i$ - a positive benefit
- $w_i$ - a positive weight

Goal: Choose items with maximum total benefit but with weight at most $W$.

If we are allowed to take fractional amounts, then this is the fractional knapsack problem.
- In this case, we let $x_i$ denote the amount we take of item $i$

Objective: maximize

$$\sum_{i \in S} b_i \left( \frac{x_i}{w_i} \right)$$

Constraint:

$$\sum_{i \in S} x_i \leq W$$
Example

Given: A set $S$ of $n$ items, with each item $i$ having
- $b_i$ - a positive benefit
- $w_i$ - a positive weight

Goal: Choose items with maximum total benefit but with weight at most $W$.

<table>
<thead>
<tr>
<th>Items</th>
<th>Weight</th>
<th>Benefit</th>
<th>Value ($/ml)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4 ml</td>
<td>$12</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>8 ml</td>
<td>$32</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2 ml</td>
<td>$40</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>6 ml</td>
<td>$30</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>1 ml</td>
<td>$50</td>
<td>50</td>
</tr>
</tbody>
</table>

Solution:
- 1 ml of 5
- 2 ml of 3
- 6 ml of 4
- 1 ml of 2

"knapsack"
The Fractional Knapsack Algorithm

- **Greedy choice:** Keep taking item with highest value (benefit to weight ratio)
  - Since $\sum_{i \in S} b_i (x_i / w_i) = \sum_{i \in S} (b_i / w_i) x_i$
  - Run time: $O(n \log n)$. Why?

- **Correctness:** Suppose there is a better solution
  - there is an item $i$ with higher value than a chosen item $j$, but $x_i < w_i$, $x_j > 0$ and $v_i > v_j$
  - If we replace some item $j$ with $i$, we get a better solution
  - How much of $i$: $\min\{w_i - x_i, x_j\}$
  - Thus, there is no better solution than the greedy one

---

**Algorithm** `fractionalKnapsack(S, W)`

- **Input:** set $S$ of items w/ benefit $b_i$ and weight $w_i$; max. weight $W$
- **Output:** amount $x_i$ of each item $i$ to maximize benefit w/ weight at most $W$

```plaintext
for each item i in S
    $x_i \leftarrow 0$
    $v_i \leftarrow b_i / w_i$  \{value\}
    $w \leftarrow 0$  \{total weight\}
while $w < W$
    remove item i w/ highest $v_i$
    $x_i \leftarrow \min\{w_i, W - w\}$
    $w \leftarrow w + \min\{w_i, W - w\}$
```
The 0-1 Knapsack Problem

- Same as before but now we are not allowed to take fractions of an item. We take whole item or none of it.
- Much more difficult than the fractional problem (as we’ll see at a later date)