Greedy

Consider a set of requests for a room. Only one person can reserve the room at a time, and you want to allow the maximum number of requests. The requests for periods \((s_i, f_i)\) are:

\[(1, 4), (3, 5), (0, 6), (5, 7), (3, 8), (5, 9), (6, 10), (8, 11), (8, 12), (2, 13), (12, 14)\]

Which ones should we schedule?
Greedy

Consider a set of requests for a room. Only one person can reserve the room at a time, and you want to allow the maximum number of requests. The requests for periods \((s_i, f_i)\) are:

\[(1, 4), (3, 5), (0, 6), (5, 7), (3, 8), (5, 9), (6, 10), (8, 11), (8, 12), (2, 13), (12, 14)\]

Which ones should we schedule?

\[
\begin{array}{ccccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\
\end{array}
\]
Sort by finishing time, renumber with 1 having earliest finishing time
Output 1
last = f_1
for i = 2 to n
   if (s_i \leq last)
      Output i
      last = f_i
Proving a Greedy Algorithm is Optimal

Two components:

1. Optimal substructure

2. **Greedy Choice Property:** There exists an optimal solution that is consistent with the greedy choice made in the first step of the algorithm.
Optimal Substructure

- Let $c[i, j]$ be the number of activities scheduled from time $i$ to time $j$

$$c[i, j] = \begin{cases} 
0 & \text{if } S_{ij} = \emptyset, \\
\max_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\} & \text{if } S_{ij} \neq \emptyset
\end{cases}$$

(1)
Greedy Choice

Greedy Choice Property

1. Let $S_k$ be a nonempty subproblem containing the set of activities that finish after activity $a_k$.
2. Let $a_m$ be an activity in $S_k$ with the earliest finish time.
3. Then $a_m$ is included in some maximum-size subset of mutually compatible activities of $S_k$.

Proof

- Let $A_k$ be a maximum-size subset of mutually compatible activities in $S_k$,
- let $a_j$ be the activity in $A_k$ with the earliest finish time.
- If $a_j = a_m$, we are done, since we have shown that $a_m$ is in some maximum-size subset of mutually compatible activities of $S_k$.
- If $a_j \neq a_m$, let the set $A'_k = A_k - \{a_j\} \cup \{a_m\}$
- The activities in $A'_k$ are disjoint, because
  - the activities in $A_k$ are disjoint,
  - $a_j$ is the first activity in $A_k$ to finish,
  - $f_m \leq f_j$.
- Since $|A'_k| = |A_k|$, we conclude that $A'_k$ is a maximum-size subset of mutually compatible activities of $S_k$, and it includes $a_m$. 
Procedure for Designing a Greedy Algorithm

1. Identify optimal substructure
2. Cast the problem as a greedy algorithm with the greedy choice property
3. Write a simple iterative algorithm
Robbery

- I want to rob a house and I have a knapsack which holds $B$ pounds of stuff
- I want to fill the knapsack with the most profitable items

<table>
<thead>
<tr>
<th>item</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>weight</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>value</td>
<td>60</td>
<td>100</td>
<td>120</td>
</tr>
<tr>
<td>value/weight</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Two variants
- integral knapsack: Take an item or leave it
- fractional knapsack: Can take a fraction of an item (infinitely divisible)
Fractional vs. Integral Knapsack

- Both fractional and integral knapsack have optimal substructure.
- Only fractional knapsack has the greedy choice property.
Fractional Knapsack

**Greedy Choice Property:** Let $j$ be the item with maximum $v_i/w_i$. Then there exists an optimal solution in which you take as much of item $j$ as possible.

**Proof**

- Suppose fpoc, that there exists an optimal solution in you didn’t take as much of item $j$ as possible.

- If the knapsack is not full, add some more of item $j$, and you have a higher value solution. **Contradiction**

- We thus assume the knapsack is full.

- There must exist some item $k \neq j$ with $v_k/w_k < v_j/w_j$ that is in the knapsack.

- We also must have that not all of $j$ is in the knapsack.

- We can therefore take a piece of $k$, with $\epsilon$ weight, out of the knapsack, and put a piece of $j$ with $\epsilon$ weight in.

- This increases the knapsacks value by

$$\epsilon \frac{v_j}{w_j} - \epsilon \frac{v_k}{w_k} = \epsilon \left( \frac{v_j}{w_j} - \frac{v_k}{w_k} \right) > 0$$

**Contradition** to the original solution being optimal.
Algorithm

1. Sort items by $v_j/w_j$, renumber.
2. For $i = 1$ to $n$
   - Add as much of item $i$ as possible

**Question** Why does this fail for integer knapsack.