Activity Selection

Given a set of activities represented by intervals \((s_i, f_i), i = 1, \ldots, n\).

To select a maximum number of compatible intervals (activities).

Two activities \((s_i, f_i)\) and \((s_j, f_j)\) are compatible if the two intervals overlap at most at end points; that is, \(s_i \geq f_j\) or \(s_j \geq f_i\).

By sorting \(f_i\) (in \(O(n \log n)\) time) we may assume that

\[ f_1 \leq f_2 \ldots \leq f_n \]

Locally optimal (greedy) strategy: maximize remaining time
Among the remaining unscheduled activities, choose one which is

- compatible with the chosen activities
- with earliest possible finish time $f_i$

Method:

- First activity to choose is $(s_1, f_1)$.
- Suppose inductively $(s_j, f_j)$ was the most recent addition, then the next addition is the least $i > j$ so that $s_i \geq f_j$.

Time: $O(n)$.
Correctness:

First choice can’t be wrong: that is, there is an optimal solution which contains our first choice \((s_1, f_1)\).

Consider a solution \(B\) that does not contain 1.

Suppose \(i\) is the activity in \(B\) having the least finishing time.

Then for all other \(j \in B\), \(i, j\) compatible \(\Rightarrow\)

\[ s_j \geq f_i \geq f_1. \]

Hence \(B - i \cup \{1\}\) is a compatible set, a solution no worse than \(B\).
Suboptimal structure: once a locally optimal choice is made, we are left with a similar but smaller problem, hence we can recurse on the same locally optimal strategy to solve the smaller problem.

Claim: Suppose $A$ is an optimal solution to $S = \{1, \ldots, n\}$ and $1 \in A$. Then $A - \{1\}$ is an optimal solution for $S' = \{i \mid s_i \geq f_1\}$.

(Proof) Otherwise there is a solution $B$ to $S'$ with $|B| > |A| - 1$. But then $B \cup \{1\}$ is a better solution for the original problem.
Greedy vs Dynamic programming – 0-1 vs Fractional Knapsack Problem.

Given

• A (knap)sack can carry no greater than $W$ pounds of good.

• $n$ items, $i$-th item worth $v_i$ dollars and has weight $w_i$ pounds.

To take as a valuable load as possible.
0-1 version: each item is either taken or left untaken.

\[ \max \left\{ \sum_{i=1}^{n} x_i v_i \mid \sum_{i=1}^{n} x_i w_i \leq W, x_i \in \{0, 1\} \right\} \]

fractional: a fractional amount of an item can be taken.

\[ \max \left\{ \sum_{i=1}^{n} x_i v_i \mid \sum_{i=1}^{n} x_i w_i \leq W, 0 \leq x_i \leq 1 \right\} \]
Greedy method for fractional knapsack

Take as much as possible of the item with the greatest value per pound \( \frac{v_i}{w_i} \).

By sorting \( \frac{v_i}{w_i} \) we may assume

\[
\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \ldots \geq \frac{v_n}{w_n}
\]

First choice: If \( w_1 \geq q W \), then \( x_1 := \frac{W}{w_1} \) and we are done. If \( w_1 < W \), then \( x_1 = 1 \) and we are left with a similar problem on the remaining items of total weight bound \( W - w_1 \).

Why it works?
Suppose $\alpha_1$ of item 1 is picked under the greedy strategy above.

Claim: This is at least as valuable as any other combination of choices with the same weight $\alpha_1 w_1$.

So assume $\sum_i y_i w_i = \alpha_1 w_1$.

\[
\alpha_1 v_1 - \sum_i y_i v_i = (\alpha_1 w_1) v_1 / w_1 - \sum_i y_i v_i
\]

\[
= (\sum_i y_i w_i) v_1 / w_1 - \sum_i y_i v_i
\]

\[
= \sum_i y_i w_i (v_1 / w_1 - v_i / w_i) \geq 0.
\]
0-1 Knapsack

If an optimal sol contains item $n$, the remaining choices must constitute an opt. sol. to similar problem on items 1, 2, ..., $n-1$ with wt bound $W - w_n$.

If an optimal sol does not contain item $n$, the sol. must also be an opt. sol. to similar problem on items 1, 2, ..., $n-1$ with wt bound $W$.

So

$$Opt(n, W) = \max\{ opt(n-1, W - w_n) + v_n, opt(n-1, W) \}$$
Huffman Code

Given an alphabet, a file over the alphabet, and the frequency $f(C)$ of the char. $C$ in the file.

To construct a variable-length prefix-free code for each char. so that the total length of the stored file

$$\sum_{C} f(c) l(C')$$

is minimized, where $l(C)$ denotes the code length of $C'$. 
Prefix-free code: no codeword is a prefix of another codeword. (To ensure unique decoding.)

Binary tree representation for prefix-free code: label each edge by 0 or 1; codewords appear as leaves (hence prefix-free since no ancestors of a leaf are codewords).

An easy obs: may assume full binary tree representation.
Greedy first choice: two char with lowest freq. chosen as siblings of max. depth.

Justification: Suppose $b$ and $c$ have lowest and second-lowest freq.

Consider any other sol. represented as a binary tree $T$.

Suppose $x, y$ appear on $T$ as two siblings with max depth with $f(x) \leq f(y)$.

Swap: $x \leftrightarrow b; \ y \leftrightarrow c$ on $T$. Call the resulting tree (code) $T'$.

$$w(T') - w(T) =$$

$$f(x)d(b) + f(b)d(x) + f(y)d(c) + f(c)d(y) -$$

$$f(x)d(x) + f(b)d(b) + f(y)d(y) + f(c)d(c)$$

$$= (f(x) - f(b))(d(b) - d(x)) + (f(y) - f(c))(d(c) - d(y)) \leq 0.$$
Reduction to smaller problem:

Let $T$ be a prefix-free code tree with $b$ and $c$ as siblings.

"Merge" $b$ and $c$ into their parent node labelled by a new pseudo-char $v$, with $f(v) = f(b) + f(c)$.

Let $T'$ be the new prefix-code tree on $\Sigma' = \Sigma - \{b, c\} \cup \{v\}$.

$$w(T') = w(T) + f(v)d_{T'}(v) - (f(b) + f(c))d_T(b)$$

$$= w(T) - (f(b) + f(c))$$

So $T$ is opt. for $\Sigma$ iff $T'$ is opt. for $\Sigma'$. 