16. Greedy algorithms
16.1 An activity-selection problem

- Suppose we have a set $S = \{a_1, a_2, ..., a_n\}$ of $n$ proposed activities that with to use a resource. Each activity $a_i$ has a start time $s_i$ and a finish time $f_i$, where $0 \leq s_i < f_i < \infty$.

- If selected, activity $a_i$ take place during the half-open time interval $[s_i, f_i)$. Activities $a_i$ and $a_j$ are compatible if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap (i.e., $a_i$ and $a_j$ are compatible if $s_i \geq f_j$ or $s_j \geq f_i$).
The activity-selection problem is to select a maximum-size subset of mutually compatible activities.

Example:

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
</tbody>
</table>

Activities in each line are compatible.
We shall solve this problem in several steps. We start by formulating a dynamic programming solution to this program in which we combine optimal solutions to two subproblems to form an optimal solution to the original problem.

We shall then observe that we need only consider one choice – the greedy choice – and that when we make the greedy choice, one of the subproblems is guaranteed to be empty, so that only one nonempty subproblem remains.
The optimal substructure of the activity-selection problem

- \[ S_{ij} = \{ a_k \in S : f_i \leq s_k < f_k \leq s_j \} \],
  So that \( S_{ij} \) is the subset of activities in \( S \) that can start after activity \( a_i \) finishes and finish before activity \( a_j \) starts.

- \( f_0 = 0 \) and \( s_{n+1} = \infty \). Then \( S = S_{0,n+1} \), and the ranges for \( i \) and \( j \) are given by \( 0 \leq i, j \leq n+1 \).

- \( A_{ij} = A_{ik} \cup \{ a_k \} \cup A_{kj} \)
A recursive solution

\[
c[i, j] = c[i, k] + c[k, j] + 1
\]

\[
c[i, j] = \begin{cases} 
0 & \text{if } S_{ij} = 0 \\
\max_{i < k < j} \{c[i, k] + c[k, j] + 1\} & \text{if } S_{ij} \neq 0 
\end{cases}
\]
Converting a dynamic-programming solution to a greedy solution

**Lemma 16.1**

Consider any nonempty subproblem $S_{ij}$, and let $a_m$ be the activity in $S_{ij}$ with the earliest finish time:

$$f_m = \min \left\{ f_k : a_k \in S_{ij} \right\}.$$

then

1. Activity $a_m$ is used in some maximum-size subset of mutually compatible activities of $S_{ij}$.

2. The subproblem $S_{im}$ is empty, so that choosing $a_m$ leaves the subproblem $S_{mj}$ as the only one that may be nonempty.
A recursive greedy algorithm

**RECURSIVE-ACTIVITY-SELECTION**(*s*, *f*, *i*, *j*)

1. \( m \leftarrow i + 1 \)
2. while \( m < j \) and \( s_m < f_i \) do \( m \leftarrow m + 1 \)
3. if \( m < j \) then return \( \{ a_m \} \cup \text{RECURSIVE-ACTIVITY-SELECTION} \ (s, f, i, j) \)
4. else return 0

\( \triangleright \) Find the first activity in \( S_{ij} \)
The operation of \texttt{RECURSIVE-ACTIVITY-SELECTOR} on the 11 activities given earlier.
An iterative greedy algorithm

**Greedy-Activity-Selector**\((s, f)\)

1. \(n \leftarrow \text{length}[s]\)
2. \(A \leftarrow \{a_1\}\)
3. \(i \leftarrow 1\)
4. **for** \(m \leftarrow 2\) **to** \(n\)
5. **do if** \(s_m \geq f_i\)
6. **then** \(A \leftarrow A \cup \{a_m\}\)
7. \(i \leftarrow m\)
8. **return** \(A\)
16.2 Elements of the greedy strategy

1. Determine the optimal substructure of the problem.
2. Develop a recursive solution.
3. Prove that at any stage of the recursion, one of the optimal choices is the greedy choice. Thus, it is always safe to make the greedy choice.
4. Show that all but one of the subproblems induced by having make the greedy choice are empty.

5. Develop a recursive algorithm that implements the greedy strategy.

6. Convert the recursive algorithm to an iterative algorithm.

- We could have started out with a greedy algorithm in mind
Greedy choice proof

1. Cast the optimization problem as one in which we make a choice and are left with one subproblem to solve.

2. Prove that there is always an optimal solution to the original problem that makes the greedy choice, so that the greedy choice is always safe.

3. Demonstrate that, having made the greedy choice, what remains is a subproblem with the property that if we combine an optimal solution to the subproblem with the greedy choice we have made, we arrive at an optimal solution to the original problem.
Greedy-choice property: A global optimal solution can be achieved by making a local optimal (optimal) choice.

Optimal substructure: An optimal solution to the problem within its optimal solution to subproblem.
Greedy versus dynamic programming

- 0-1 knapsack problem
- Fractional knapsack problem
- (Exercise 16.2.2)
- **0-1 knapsack problem:**
  - $n$ items.
  - Item $i$ is worth $v_i$, weighs $w_i$ pounds.
  - Find a most valuable subset of items with total weight $\leq W$.
  - Have to either take an item or not take it—can’t take part of it.

- **Fractional knapsack problem:** Like the 0-1 knapsack problem, but can take fraction of an item.
  - Both have optimal substructure.
  - But the fractional knapsack problem has the greedy-choice property, and the 0-1 knapsack problem does not.
  - To solve the fractional problem, rank items by value/weight: $v_i/w_i$.
  - Let $v_i/w_i \geq v_{i+1}/w_{i+1}$ for all $i$. 
The greedy strategy does not work for the 0-1 knapsack
16.3 Huffman codes

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency (in hundred)</td>
<td>45</td>
<td>13</td>
<td>12</td>
<td>16</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>Fixed length codeword</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>101</td>
</tr>
<tr>
<td>Variable length codeword</td>
<td>0</td>
<td>101</td>
<td>100</td>
<td>111</td>
<td>1101</td>
<td>1100</td>
</tr>
</tbody>
</table>

- In a 100,000 characters file, FLC and VLC need 300,000 and 224,000 bits, respectively.

- **Prefix code**: no codeword is also a prefix of some other codeword.
Can be shown that the optimal data compression achievable by a character code can always be achieved with prefix codes.

Simple encoding and decoding.

An optimal code for a file is always represented by a binary tree.
Tree correspond to the coding schemes

\[ B(T) = \sum_{c \in C} f(c) d_T(c) \]

cost of tree \( T \)
Constructing a Huffman code

HUFFMAN( C )

1  \( n \leftarrow |C| \)
2  \( Q \leftarrow C \)
3  \textbf{for} \ i \leftarrow 1 \textbf{ to } n - 1 \textbf{ do} \quad \text{allocate a new node } z
4     \( \text{left}[z] \leftarrow x \leftarrow \text{EXTRACT-MIN}(Q) \)
5     \( \text{right}[z] \leftarrow y \leftarrow \text{EXTRACT-MIN}(Q) \)
6     \( f[z] \leftarrow f[x] + f[y] \)
7  \text{INSERT}(Q,Z)
8 \textbf{return} \ \text{EXTRACT-MIN}(Q)

\textbf{Complexity:} \ O(n \log n)
The steps of Huffman’s algorithm

(a) $f:5$  $e:9$  $c:12$  $b:13$  $d:16$  $a:45$  

(b) $c:12$  $b:13$  

(c) $14$  

(d) $25$  

$0$  $1$  

$14$  

$25$  

$0$  $1$  

$0$  $1$  

$14$  

$30$  

$a:45$  

$c:12$  $b:13$  

$f:5$  $e:9$  

$d:16$  

$a:45$  

$c:12$  $b:13$  

$f:5$  $e:9$
Correction of Huffman’s algorithm

(Greedy-choice property)

Lemma 16.2.

Let \( C \) be an alphabet in which \( c \in C \) has frequency \( f[c] \). Let \( x \) and \( y \) be the two characters in \( C \) having the lowest frequencies. Then there exists an optimal prefix code in \( C \) in which the codeword for \( x \) and \( y \) having the same length and differ only in the last bit.
Proof:

\[
B(T) - B(T') = \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c)
\]

\[
= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a)
\]

\[
= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x)
\]

\[
= (f[a] - f[x])(d_T(a) - d_T(x)) \geq 0,
\]

\[
B(T) - B(T'') \geq 0
\]
Lemma 16.3
(Optimal substructure)

Let $C$ be a given alphabet with frequency $f[c]$ defined for each character $c \in C$. Let $x$ and $y$ be two characters in $C$ with minimum frequency. Let $C'$ be the alphabet $C$ with characters $x$, $y$ removed and character $z$ added, so that $C' = C - \{x, y\} \cup \{z\}$; define $f$ for $C'$ as for $C$, except that $f[z] = f[x] + f[y]$. 
Lemma 16.3 (cont)

Let $T'$ be any tree representing an optimal prefix code for the alphabet $C'$. Then the tree $T$, obtained from $T'$ by replacing the leaf node for $z$ with an internal node having $x$ and $y$ as children, represents an optimal prefix code for the alphabet $C$.

Proof

$$B(T) = B(T') + f[x] + f[y]$$
Theorem 16.4

Procedure HUFFMAN produces an optimal prefix code.