Greedy Algorithms

Definition
Activity Selection
Huffman Codes
Greedy algorithms is a technique for solving some optimization problems.

A greedy algorithm solution to a problem usually involves:

1. Repeatedly identify a decision to be made.
2. Make a locally optimal choice for each decision.

To reach a globally optimal solution, the problem must have an appropriate recursive structure, i.e., an optimal solution equals a locally optimal choice plus an optimal solution for the remainder of the problem.
A set of activities are to be scheduled in a room. An activity \( a \) has start time \( a.\text{start} \) and a finish time \( a.\text{finish} \). Maximize the number of scheduled activities.

Locally optimal choice: Among activities with no conflicts with previously scheduled activities, choose the activity with the earliest finish time.

Recursive structure: This choice maximizes the amount of time afterwards, so no other choice can allow more activities to be scheduled.
A is an array of activities. $S$ is a schedule.

**GREEDY-ACTIVITY-SELECTOR($A$)**

1. sort $A$ by $A.\text{finish}$
2. $n \leftarrow A.\text{length}$
3. $S \leftarrow \{A[1]\}$
4. $\text{last} \leftarrow A[1]$
5. for $i \leftarrow 2$ to $n$
   1. $\text{current} \leftarrow A[i]$
   2. if $\text{current}.\text{start} \geq \text{last}.\text{finish}$
      1. then $S \leftarrow S \cup \{\text{current}\}$
      2. $\text{last} \leftarrow \text{current}$
6. return $S$
Consider these activities sorted by finish time:

<table>
<thead>
<tr>
<th></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>start</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>finish</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>


A binary character code assigns a bit string to each character.

In a prefix code, no codeword is a prefix of another codeword.

A Huffman code is an optimal prefix code for a sequence of characters.

Example prefix codes:

<table>
<thead>
<tr>
<th>Frequency</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code 1</td>
<td>00</td>
<td>100</td>
<td>101</td>
<td>110</td>
<td>01</td>
<td>111</td>
</tr>
<tr>
<td>Code 2</td>
<td>00</td>
<td>1110</td>
<td>10</td>
<td>110</td>
<td>01</td>
<td>1111</td>
</tr>
</tbody>
</table>
Locally optimal choice:

Combine the two least frequent characters into a single pseudo-character, i.e., they will have the same prefix, followed by a 0 for one character and 1 for the other.

Recursive structure:

An optimal prefix code corresponds to a full binary tree with characters as leaves.

Less frequent characters (including pseudo-characters) should have equal or greater depth in the tree.
Huffman Code Algorithm

Let $C$ be a set of characters with a $freq$ field.

**HUFFMAN($C$)**

$n \leftarrow C.length$

create priority queue $Q$ from $C$ using $freq$ values

for $i \leftarrow 1$ to $n - 1$

$z \leftarrow \text{ALLOCATE-NODE}()$

$z.left \leftarrow \text{EXTRACT-MIN}(Q)$

$z.right \leftarrow \text{EXTRACT-MIN}(Q)$

$z.freq \leftarrow z.left.freq + z.right.freq$

INSERT($Q, z$)

return EXTRACT-MIN($Q$)
Huffman Code Example

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

(b) c:12  b:13  14  d:16  a:45

(c) 14
    /   \
   0     1
f:5    e:9

(d) 25
    /   \
   0     1
f:5    e:9

(c) 25
    /   \
   0     1
f:5    e:9

(d) 30
    /   \
   0     1
f:5    e:9

(e) a:45
    /   \
   0     1
f:5    e:9

(f) 100
    /   \
   0     1
a:45  55
    /   \
   0     1
f:5    e:9

(c) 25
    /   \
   0     1
f:5    e:9

(f) 30
    /   \
   0     1
f:5    e:9

CS 5633 Analysis of Algorithms