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.

Activity Selection

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.
Activity Selection Algorithm

$A$ is an array of activities. $S$ is a schedule.

**Greedy-Activity-Selector($A$)**

sort $A$ by $A$.finish

$n \leftarrow A$.length

$S \leftarrow \{A[1]\}$

$last \leftarrow A[1]$

for $i \leftarrow 2$ to $n$

$\quad current \leftarrow A[i]$

$\quad$ if current.start $\geq$ last.finish

$\quad$ then $S \leftarrow S \cup \{current\}$

$\quad last \leftarrow current$

return $S$

Huffman Codes

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></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</td>
<td>11</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>13</td>
<td>2</td>
</tr>
<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>

Huffman Code Analysis

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.

Activity Selection Example

Consider these activities sorted by finish time:

<table>
<thead>
<tr>
<th>A</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>


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 $i$ from 1 to $n - 1$

$z \leftarrow$ ALLOCATE-NODE()

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

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

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

INSERT($Q$, $z$)

return EXTRACT-MIN($Q$)