CSCI 241
Scott Wehrwein
Huffman Codes:
Building Optimal Coding Trees
Goals

Understand the intuition behind Huffman Coding Trees

Be able to execute on paper and implement construction of Huffman Coding Trees.
Tree Construction: Intuition

Suppose we want to encode the following strings:

aaaaaaa 0000000
aaaabbb 0000101101101
fffffff 1100110011001100110011001100

Different trees are good for different inputs.

Idea:
- Build a tree that's **optimal** for your input.
- Store the tree plus the encoded string.
Tree Construction: Intuition

Suppose we want to encode the following strings:

- `aaaaaaa` 0000000
- `aaaabbb` 000101101101
- `fffffff` 11001100110011001100

Different trees are good for different inputs.

What makes a tree good for a given input? Key idea:
- Characters that appear **more often** should live **closer to the root**.
Tree Construction: Intuition

Characters that appear **more often** should live **closer to the root**.

What are those numbers I hid?

**Frequencies!**

**Goal:** build a tree with the most-frequent symbols as close to the root as possible.
Tree Construction: Algorithm

Overview:

1. Count the frequency of each character in the input.
2. Create a forest of single-node trees, one for each character.
3. Build the tree bottom-up by repeatedly connecting the two lowest-frequency trees.
Tree Construction: Algorithm

Example: 

<table>
<thead>
<tr>
<th>Frequency (in thousands)</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>45</td>
<td>13</td>
<td>12</td>
<td>16</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>

1. Count the frequency of each character in the input.
Tree Construction: Algorithm

Example:

<table>
<thead>
<tr>
<th>Character</th>
<th>Frequency (in thousands)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>45</td>
</tr>
<tr>
<td>b</td>
<td>13</td>
</tr>
<tr>
<td>c</td>
<td>12</td>
</tr>
<tr>
<td>d</td>
<td>16</td>
</tr>
<tr>
<td>e</td>
<td>9</td>
</tr>
<tr>
<td>f</td>
<td>5</td>
</tr>
</tbody>
</table>

1. Count the frequency of each character in the input.
2. Create a **forest** of single-node trees, one for each character.
Tree Construction: Algorithm

3. Greedily connect the two lowest-frequency trees.
   
   A. Remove the two trees with smallest root frequencies from the forest.
   
   B. Join them with a parent whose frequency is the sum of the child frequencies.
   
   C. Put the newly created tree back into the forest.
Tree Construction: Algorithm

3. Greedily connect the two lowest-frequency trees.
   
   A. Remove the two trees with smallest root frequencies from the forest.
   
   B. Join them with a parent whose frequency is the sum of the child frequencies.
   
   C. Put the newly created tree back into the forest.
Tree Construction: Algorithm

3. Greedily connect the two lowest-frequency trees.

A. Remove the two trees with smallest root frequencies from the forest.

B. Join them with a parent whose frequency is the sum of the child frequencies.

C. Put the newly created tree back into the forest.
Tree Construction: Algorithm

3. Greedily connect the two lowest-frequency trees.
   
   A. Remove the two trees with smallest root frequencies from the forest.
   
   B. Join them with a parent whose frequency is the sum of the child frequencies.
   
   C. Put the newly created tree back into the forest.
Tree Construction: Algorithm

3. Greedily connect the two lowest-frequency trees.
   
   A. Remove the two trees with smallest root frequencies from the forest.
   
   B. Join them with a parent whose frequency is the sum of the child frequencies.
   
   C. Put the newly created tree back into the forest.
Tree Construction: Algorithm

3. Greedily connect the two lowest-frequency trees.
   
   A. Remove the two trees with smallest root frequencies from the forest.
   
   B. Join them with a parent whose frequency is the sum of the child frequencies.
   
   C. Put the newly created tree back into the forest.
Tree Construction: Algorithm

3. Greedily connect the two lowest-frequency trees.
   
   A. Remove the two trees with smallest root frequencies from the forest.
   
   B. Join them with a parent whose frequency is the sum of the child frequencies.
   
   C. Put the newly created tree back into the forest.
Tree Construction: Algorithm

3. Greedily connect the two lowest-frequency trees.
   A. Remove the two trees with smallest root frequencies from the forest.
   B. Join them with a parent whose frequency is the sum of the child frequencies.
   C. Put the newly created tree back into the forest.
Tree Construction: Algorithm

3. Greedily connect the two lowest-frequency trees.
   A. Remove the two trees with smallest root frequencies from the forest.
   B. Join them with a parent whose frequency is the sum of the child frequencies.
   C. Put the newly created tree back into the forest.
Tree Construction: Algorithm

3. Greedily connect the two lowest-frequency trees.
   A. Remove the two trees with smallest root frequencies from the forest.
   B. Join them with a parent whose frequency is the sum of the child frequencies.
   C. Put the newly created tree back into the forest.
Tree Construction: Algorithm

3. Greedily connect the two lowest-frequency trees.
   
   A. Remove the two trees with smallest root frequencies from the forest.
   
   B. Join them with a parent whose frequency is the sum of the child frequencies.
   
   C. Put the newly created tree back into the forest.
Tree Construction: Algorithm

3. Greedily connect the two lowest-frequency trees.
   
   A. Remove the two trees with smallest root frequencies from the forest.
   
   B. Join them with a parent whose frequency is the sum of the child frequencies.
   
   C. Put the newly created tree back into the forest.
Tree Construction: Algorithm

3. Greedily connect the two lowest-frequency trees.

   A. Remove the two trees with smallest root frequencies from the forest.

   B. Join them with a parent whose frequency is the sum of the child frequencies.

   C. Put the newly created tree back into the forest.