Huffman Encoding

Data encoding
- Pictures, songs, videos are stored as binary strings.
- Want to store using as few bits as possible.

Simplest - just store as long binary strings.
- Text: each ASCII code stored as 8-bit string.
- Picture: each pixel has 3 ASCII codes (RGB).

Better:
- Data compression - encode information using fewer bits than original.

Focus on text files, but algorithms apply to all file types.
Consider text file with following contents:

<table>
<thead>
<tr>
<th>char</th>
<th>freq</th>
<th>code</th>
<th>#bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>10</td>
<td>000</td>
<td>30</td>
</tr>
<tr>
<td>e</td>
<td>15</td>
<td>001</td>
<td>45</td>
</tr>
<tr>
<td>i</td>
<td>12</td>
<td>010</td>
<td>36</td>
</tr>
<tr>
<td>s</td>
<td>3</td>
<td>011</td>
<td>9</td>
</tr>
<tr>
<td>t</td>
<td>4</td>
<td>100</td>
<td>12</td>
</tr>
<tr>
<td>sp</td>
<td>13</td>
<td>101</td>
<td>29</td>
</tr>
<tr>
<td>re</td>
<td>1</td>
<td>110</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ 174 \rightarrow \text{Can we do better?} \]

1. Fixed-length encoding - codes have same length.
   - For n character types, need $\lceil \log(n) \rceil$ (where $\lceil \log(7) \rceil = 3$) bits per code.

Can we do better?

Goal: Given an alphabet $A$ of $n$ character types, where each char $c \in A$ has frequency $f_c$, find a binary encoding that uses fewest total number of bits:

\[ \text{minimize total \# bits: } \sum_{c \in A} f_c \cdot \lceil \log(n) \rceil \]

Helps to visualize with binary tree.
Characters at leaves:
- every left branch: 0
- right branch: 1

Now can easily decode: 011 010 100
\[ \underline{\text{s}} \underline{i} \underline{1} \underline{1} \]

Notice: There will be an unique decoding for each code.
Why? All characters at leaves.

```
00 00
b 00 000 0000
```

Not unique!

_unambiguous encoding_: unique code for each character

But is it most efficient? Uses fewest number of bits?

Number of bits in terms of tree:

\[ \text{#bits} = \text{cost of tree} = \sum_{c \in A} d_c \times f_c \]

\[ \text{cost} \times \text{depth} \]

Goals (in terms of tree):
- cheapest tree w/unambiguous encoding
Can we get a cheaper tree?

Yes!

\[ n_l = 11 \quad \text{level} \ 2 \rightarrow \text{new cost} = 173 \]

Observations:

1. Cheapest tree will be full binary tree - every node is either leaf or has exactly 2 children.
2. Variable length encoding (code lengths differ) helps.

Now that we can use variable length encoding, how can we get a cheaper tree?

Greedy idea? Hint: Frequency of chars.

Assign codes so that:

- Less frequent chars have longer code lengths.
- More frequent chars have shorter code lengths.
Huffman's Algorithm

"Start with single-node trees (one for every char)
Merge the two cheapest trees."

Notice:
Less freq chars have longer codes
More "shorter"

[decode 100001]
  1
  |
  i
Huffman's Alg. (turn project as grad student at MIT, 1954)

**Given** A of size n, \( f_c \) for \( A \in A \)

1. For every char c with freq \( f_c \), create a single-node tree labeled with \( c, f_c \).
   (cost of tree \( c = f_c \)).

2. Do \( n-1 \) times:
   Merge 2 trees with the lowest costs
   (cost of new tree is sum of costs of 2 subtrees)

\[
\begin{align*}
& \text{Optimality (Won't do formal proof)} \\
& \text{Proved day before project was due.} \\
& \text{(1) unambiguous encoding - all characters at leaves.} \\
& \text{(2) cheapest tree -} \\
& \quad \text{full binary tree - always merging either 2 leaves, 2 full binary trees, or 1 leaf and 1 full tree} \\
& \quad \text{min cost - less freq chars have longer lengths} \quad \text{shorter}
\end{align*}
\]
RunTime: For n char types:

Maintain trees in min binary heap ordered by cost.

Step 1: BuildHeap() \(\Rightarrow O(n)\).

Step 2: \(2(n-1)\) deleteMin()’s \(\Rightarrow O(n\log n)\) deleteMin is \(O(\log n)\)
\(n-1\) insert()’s \(\Rightarrow O(n\log n)\) insert is

Total: \(O(n\log n)\).

One drawback: Have to read the entire file first to get a count of chars, freqs.

How to avoid this? Use statistics of English language to get an estimate of chars, freqs.