<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>34</td>
</tr>
<tr>
<td>re</td>
<td>1</td>
<td>110</td>
<td>3</td>
</tr>
</tbody>
</table>

$\sum \text{ # bits} = 174 \rightarrow \text{Can we do better?}$

1. Skip ASCII step, simply encode with binary strings.
   Fixed-length encoding: codes have same length.

   For $n$ character types, need $\lceil \log(n) \rceil$ ($= \lceil \log(3) \rceil = 3$) bits per code (need $\log_2(n)$ bits to encode $n$ in binary).

2. 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.

   (i.e. 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" \\

[Now can easily decode: 011 010 00]

Notice: There will be an unique decoding for each code.

Why? All characters at leaves.

\[
\begin{align*}
&b=00 \quad a=000 \\
&00000 \quad 00000 \\
\end{align*}
\]

unambiguous encoding - unique code for each character.

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

Number of bits in terms of tree?

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

depth of c in tree

Goals (in terms of tree)

- cheapest tree that yields unambiguous encoding
Can we get a cheaper tree?

Yes!

\[
\begin{array}{c}
\text{old} \\
\text{new cost = 173.}
\end{array}
\]

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 ofChars.

Assign codes st:

- More frequent characters have longer code lengths.
- Shorter codes for less frequent characters.
Huffman's Algorithm

"Start with single-node trees (one for every char)"

"Merge the two cheapest trees"

58 (not the final tree cost!

Notice:

Less frequent chars have longer codes
More "shorter"
Huffman's Alg. (term project as grad student at MIT, 1954)

Given $A$ of size $n$, $f_c$ for $c \in A$.

1. For every char $c$ with freq $f_c$, create a
   single-node tree labeled with $c$, $f_c^2$.
   (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)

$\begin{array}{c}
\text{Optimality (Won't do formal proof).} \\
\text{(Proved day before project was due.)} \\
\text{1) unambiguous encoding - all characters at leaves.} \\
\text{2) cheapest tree -} \\
\text{'full binary tree - always merging either 2} \\
\text{leaves', 2 full binary trees, or 1 leaf and} \\
\text{1 full tree} \\
\text{'min cost - less freq chars have longer lengths} \\
\text{more than shorter'}
\end{array}$

Go to table
RunTime: For n char types.

Maintain trees in min binary heap ordered by cost.

Step 1: `Build_Heap()` : $O(n)$.

Step 2: 2(n-i) `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.
<table>
<thead>
<tr>
<th>char</th>
<th>freq</th>
<th>code</th>
<th>#bits</th>
<th>Huff code</th>
<th>#bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>10</td>
<td>000</td>
<td>30</td>
<td>001</td>
<td>30</td>
</tr>
<tr>
<td>e</td>
<td>15</td>
<td>001</td>
<td>45</td>
<td>01</td>
<td>30</td>
</tr>
<tr>
<td>i</td>
<td>12</td>
<td>010</td>
<td>36</td>
<td>10</td>
<td>24</td>
</tr>
<tr>
<td>s</td>
<td>3</td>
<td>011</td>
<td>9</td>
<td>00001</td>
<td>15</td>
</tr>
<tr>
<td>t</td>
<td>4</td>
<td>100</td>
<td>12</td>
<td>0001</td>
<td>16</td>
</tr>
<tr>
<td>sp</td>
<td>13</td>
<td>101</td>
<td>39</td>
<td>11</td>
<td>26</td>
</tr>
<tr>
<td>hi</td>
<td>1</td>
<td>110</td>
<td>3</td>
<td>00000</td>
<td>5</td>
</tr>
</tbody>
</table>

**Total**: 174  
**Total**: 146
Text Verification Problem
Possibly corrupted text file. Does it correspond to real text?

Given string $S$ of length $n$:
1. Determine if $S$ is "valid" text (i.e. made up of valid words)
2. If so, "reconstruct" $S$, split into valid words

Assumptions:
1. All lowercase
2. No punctuation
3. Given a dictionary that takes as input string $w$:

$$\text{Dict}(w) = \begin{cases} 
\text{true (1)} & \text{if } w \text{ is a valid word (including "a" and "I")} \\
\text{true (1)} & \text{if } w \text{ is empty string} \\
\text{false (0)} & \text{otherwise} 
\end{cases}$$

Note: $S = "\text{thinknot}"$ - 2 correct solutions: think not
thin knot

Ex: $S = \text{think nothing}$

Possible to solve in $O(n^2)$, $(n = 151)$
Problem exhibits optimal sub-structure property so use D.P.

1. ID array of size $n+1$, called $A$ (index 0 is for Base Case).

2. $A[i] =$ validity (true or false) of prefix of $S$ ending at $i$ (ie. $S_1...i$)

Two ways that prefix ending at $i$ can be valid:

- $S_1S_2S_i$
- or -
- $S_1S_2S_kS_{k+1}...S_i$

3. D.P. Formulation:

$$A[i] = \begin{cases} 
\text{true} & \text{if } i = 0 \text{ (Base Case)} \\
\text{true} & \text{if } \text{Dict}(S_1S_2S_i) = \text{true} \text{ and } i > 0 \\
\text{true} & \text{if } \exists k \text{ s.t. } 1 \leq k < i \text{ such that } \\
& A[S_1S_2S_k] = \text{true (valid) and } \\
& \text{Dict}(S_{k+1}...S_i) = \text{true (valid))} \text{ and } i > 0 \\
\text{false} & \text{otherwise}
\end{cases}$$

Base Case: if $i = 0 \Rightarrow \text{true (empty string)}$. 
(4) Fill A with for loop from i = 1 \ldots n.

(5) Answer (valid text?) in A[n].

(6) RunTime : \( O(n^2) \) (for each entry, search previous)

(7) Actual Solution (reconstructed text)?

ex: Øthinknothing

```
0 1 2 3 4 5 6 7 8 9 10 11 12
Øthinknothing
```

[Diagram showing a sequence with T/F values and decision paths]

Case 2 w/ k = 7 or k = 5