Example 2: is on Data Compression.

Huffman Codes (Ch 16.3) (Discussion is from the book)

Scenario: Suppose we have a file consisting of 100,000 characters $\{a, b, c, d, e, f\}$ and we know the percentage of occurrence of each character $f(a), f(b), \ldots$

Example: $a \ b \ c \ d \ e \ f$

\[
\begin{align*}
\text{freq.} & \quad 45 \quad 13 \quad 12 \quad 16 \quad 9 \quad 5
\end{align*}
\]

We want to encode these in binary. We want the code to be such that we can decipher the original characters in a unique manner (decoding).

There are two types of codes which do not lead to ambiguity in decoding:

- **Fixed Length Code**
  - Need 3-bits
  - Size of File: 300,000 fixed

- **Variable Length Code**
  - Prefix-free
  - Size of File: 300,000 fixed

\[
\begin{align*}
000\quad 001\quad 010\quad 011\quad 100\quad 101
\end{align*}
\]

Decoding:

- First position: No conflict
- Second position: No conflict
- Third position: No conflict

Decoding:

- $a/b/c$
Problem Statement

Given frequencies, find a variable length prefix-free code that minimizes file size.

Binary Trees Corresponding to prefix-free codes

Fixed length code

Variable length code

File size: proportional to

\[ B_T = \sum_{x \in C} f(x) d_T(x) \]

Find \( T \) such that Min

Binary tree with as many leaf nodes as the number of characters correspond one-to-one to prefix-free code.
Greedy algorithm:

Make min\(f(x)\) and \(s_{\text{min}}f(x)\) as siblings of an internal. Condense these into one new symbol with freq = \(f_{\text{freq}} = \min(f(x)) + s_{\text{min}}f(x)\)

And repeat. We show this with an example.

\[
\begin{array}{cccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} \\
\text{f(x)} & 45 & 13 & 12 & 16 & 9 & 5 \\
45 & 13 & 12 & 16 & 14 & \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{a} & \text{b} & \text{e} & \text{d} & \text{ef} \\
45 & 13 & 12 & 16 & 14 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{a} & \text{bc} & \text{ef} \\
45 & 25 & 16 & 14 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{a} & \text{bc} & \text{def} & \\
45 & 25 & 30 & 55 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{bc} & \text{ef} \\
13 & 12 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{bc} & \text{def} \\
25 & 14 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{def} & \\
30 & \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{bc} & \text{def} \\
25 & 14 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{bc} & \text{def} \\
13 & 12 & 55 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{b} & \text{c} & \text{ef} \\
\text{d} & \text{e} & \text{f} \\
\end{array}
\]
"Clearly" Greedy procedure
\[ \rightarrow \text{Min} + \text{Sum} \]

at each step

Pf of Correctness:
Recall: our problem is to find a binary tree with as many leaf nodes as there are characters (these can occur in any "order" in the tree) so as to minimize \( B_T = \sum_{x \in C} d(x) \), where \( d_T(x) \) is the "depth" of leaf node \( x \) in \( T \).

Special Classes of Binary Trees
A binary tree is called a "full" binary tree (Confusion between "Complete", "full", "balanced" etc: Watch out)

if every internal node has two "children".

\( \text{non-leaf} \)
Lemma: Each optimal tree is a full binary tree (as long as \( |C| \geq 2 \)).

\[ P_1: \] Suppose a tree \( T \) has as many leaf nodes as \( |C| \). Suppose \( T \) is not a full binary tree. Hence there is some internal node \( z \) which has only one child. Consider a tree \( T' \) formed by making some descendant of \( z \) that is a leaf node as the second child of \( z \).

![Diagram showing a tree and its transformation to a full binary tree]

\[ B_{T'} < B_T \] since \( d_{T'}(a) < d_T(a) \) and all others are same. (We assume all freq > 0)

Now we will show that \( T' \) is an optimal tree in which min \& max min are "siblings" at the greatest depth.
Lemma 16.2

Let \( f(x) \) : be Smallest frequency
\( f(y) \) : Second Smallest \( y \neq x \).

\exists an optimal binary tree \( T \) in which \( x \) and
\( y \) are Siblings at the greatest depth.

Pf: Let \( T' \) be an optimal tree such that the
Above statement is not true. \( T' \) Shown below.

In \( T' \), there is some leaf node, say \( a \),
Such that \( d(a) = \max d_T(p) \).

\( T' \) has a sibling \( \text{in} \ T' \) by Lemma in the
previous page. \( b \) can not be an internal
node ( Why? ) So both \( a \) and \( b \) have
max depth in \( T' \). \( \{a, b\} \cup \{x, y\} \) may be
empty, have one element or \( \{a, b\} = \{x, y\} \).

In the last case, there is nothing to prove
\( T'=T \). So we will assume the worst
possibility: \( \{a, b\} \cap \{x, y\} = \emptyset \).

The figure on next page illustrate
this case. (Others follow easily.)
Construct $T''$ by exchanging $x$ and $a$

$B_{T'} \iff B_{T''}$
\[
\frac{f(x)d(x)}{T'} + f(a)d(a)_{T'} \iff f(x)d(x)_{T''} + f(a)d(a)_{T''}
\]
\[
[f(x) - f(a)] \left[ d(x)_{T'} - d(a)_{T'} \right] \leq 0 \iff \text{Since } f(x) \text{ is min}
\]
\[
\leq 0 \iff \text{Since } a \text{ has max depth}
\]

Since $T'$ is optimal, $T''$ is also optimal.

Produce $T'''$ by switching $y$ and $b$ using a similar proof. Hence the lemma.
Lemma 16.3:

\[ \{C, f\} \] Original problem.

\( f(x) \leftarrow \min_{y} f(y) : S_{\text{min}}. \)

Let \( \{C', f'\} \) be obtained as follows.

\( C' = \{ C - \{ x, y \} + \{ z \} \} ; |C'| < |C| \)

\[ f'(p) = \begin{cases} f(p) & p \neq x, y, z \\ f(x) + f(y) & p = z \end{cases} \]

Let \( T' \) be optimal for \( \{C', f'\} \). Expand the leaf node \( z \) into two nodes, \( x \) and \( y \)

\[ \text{to create tree } T \text{ for } \{C, f\}. \]

Then \( T \) is optimal for \( \{C, f\} \).

This justifies correctness of algorithm via a proof by induction.

Proof of Lemma on next page.
Proof of Lemma 16.3:

Let $T'$ be optimal for $\{C',f'\}$. Let $T$ be obtained by extending $\mathcal{E}$ as in lemma. By way of proof by contradiction, suppose $T$ is not optimal for $\{C,f\}$. Let $\hat{T}$ be an optimal tree for $\{C,f\}$ with $x$ and $y$ as siblings at the greatest depth as per lemma 16.2. Let $\hat{T}'$ be obtained from $\hat{T}$ by the reverse of the process used to get $T$ from $T'$: i.e.

To keep things straight:

- $T'$ opt for $\{C',f'\}$: given
- $T$ obtained from $T'$ for $\{C,f\}$: Supposed to be nonopt.
- $\hat{T}$ opt for $\{C,f\}$
- $\hat{T}'$ obtained from $\hat{T}$, for $\{C',f'\}$.

Now we show $\hat{T}'$ is strictly better than $T'$ for $\{C',f'\}$, contradicting given statement $T'$ opt for $\{C',f'\}$.
This is what the algorithm does at each step:

- Implement a fixed A complexity:
- Next path
- Using induction
- Since $1 < 1$

Now we complete the proof for the algorithm:

- Contradiction to the assumption in

Contradiction $T$ not optimal for $f(x,y)$

Similarly

$B_T = B_T + f(x,y)$

$B_T = B_T + f(x,y)$

$B_T = B_T$
At each step, we need min & s min of the frequency values in the condensed system. We use min heap data structure. We illustrate its evolution now (assuming you have seen it - else read Ch 7 I think).

Key values: at the beginning

\[
\begin{array}{cccccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} \\
45 & 13 & 12 & 16 & 9 & 5 \\
\end{array}
\]

**Step 1** Create a heap with six nodes with min-heap property (\(\leq\))
Complexity (\(\mathcal{O}(n)\))

For example

**Step 2**: Extract-Min (Complexity?)

**Step 3**: Extract-Min:

\(\text{e}: 9\)
Combine \( \textbf{ef} : 14 \)

Step 4: Insert \( \textbf{ef} \) into the heap (Complexity)

All of this is implementation of one step of the algorithm.

Total Complexity = \# of steps \times \text{work per step}

\[ \text{Total Complexity} = 2 \times 3 \times 4 \]

\[ = \Theta(n \log n) \]

\[ \therefore \text{Complexity of the algorithm as a whole is } \Theta(n \log n) \]