Greedy 3: Huffman Codes

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

Given an alphabet Σ (like the English alphabet), an **encoding** is a function that maps each letter in Σ to a binary string, called a **codeword**.

For example, suppose $\Sigma = \{a, b, c, d, e, f\}$ and consider the encoding where $a = 000, b = 001, c = 010, d = 011, e = 100$, and $f = 101$. The word "bed" can be encoded as 001100011.

We can reduce the length of encoding if letters' usage frequencies are known.

Suppose that, in a document, 10% of the letters are a, namely, the letter has **frequency** 10%. Similarly, suppose that letters b, c, d, e, and f have frequencies $20\%, 13\%, 9\%, 40\%$, and 8%, respectively.

If we use the encoding $a = 100$, $b = 111$, $c = 101$, $d = 1101$, $e = 0$, $f = 1100$, the average number of bits per letter is:

$$3 \cdot 0.1 + 3 \cdot 0.2 + 3 \cdot 0.13 + 4 \cdot 0.09 + 1 \cdot 0.4 + 4 \cdot 0.08 = 2.37.$$

This is better than using 3 bits per letter.

What is wrong with the encoding $e = 0, b = 1, c = 00, a = 01, d = 10, f = 11$? **Ambiguity in decoding!** For example, does the string 10 mean "be" or "d"?

To allow decoding, we enforce the following constraint:

> No letter's codeword should be a prefix of another letter's codeword.

An encoding satisfying the constraint is said to be a **prefix code**.

> **Example:** The encoding $a = 100$, $b = 111$, $c = 101$, $d = 1101$, $e = 0$, $f = 1100$ is a prefix code. Just for fun, trying decoding the following binary string.
>
> 1001101010011001101100110 1

The Prefix Coding Problem

For each letter $\sigma \in \Sigma$, let $freq(\sigma)$ denote the frequency of σ. Also, denote by $len(\sigma)$ the number of bits in the codeword of σ.

Given an encoding, its **average length** is

$$\sum_{\sigma \in \Sigma} freq(\sigma) \cdot len(\sigma).$$

The objective of the **prefix coding problem** is to find a prefix code for Σ with the shortest average length.

A **code tree** on Σ as a binary tree T satisfying:

- Every leaf node of T corresponds to a unique letter in Σ; every letter in Σ corresponds to a unique leaf node in T.
- For every internal node of T, its left edge (if exists) is labeled 0, and its right edge (if exists) is labeled 1.

T generates a prefix code as follows:

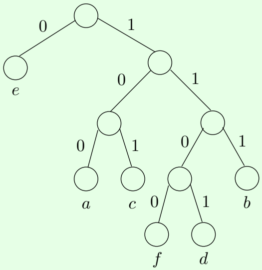
- For each letter $\sigma \in \Sigma$, generate its codeword by concatenating the bit labels of the edges on the path from the root of T to σ.

Think: Why must the encoding be a prefix code?

Lemma: Every prefix code is generated by a code tree.

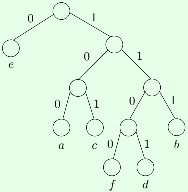
The proof will be left as a regular exercise.

Example: For our encoding $a = 100$, $b = 111$, $c = 101$, $d = 1101$, $e = 0$, and $f = 1100$, the code tree is:



Let T be the code tree generating a prefix code. Given a letter σ of Σ, its code word length $len(\sigma)$ is the **level** of its leaf node $level(\sigma)$ in T (i.e., the number edges from the root to node σ).

Example:



The levels of e, a, c, f, d, and b are 1, 3, 3, 4, 4, and 3, respectively.

Hence:

$$\text{avg length} = \sum_{\sigma \in \Sigma} freq(\sigma) \cdot len(\sigma) = \sum_{\sigma \in \Sigma} freq(\sigma) \cdot level(\sigma) = \text{avg height of } T$$

Goal (restated): Find a code tree on Σ with the smallest average height.

Huffman's Algorithm

Next, we will see a simple algorithm for solving the prefix coding problem.

Let $n = |\Sigma|$. In the beginning, create a set S of n stand-alone leaves, each corresponding to a distinct letter in Σ. If leaf z is for letter σ, define the **frequency** of z to be $freq(\sigma)$.

Huffman's Algorithm

Then, repeat until $|S| = 1$:

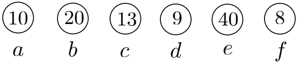
1. Remove from S two nodes u_1 and u_2 with the smallest frequencies.
2. Create a node v with u_1 and u_2 as the children. Set the **frequency** of v to be the frequency sum of u_1 and u_2.
3. Add v to S.

When $|S| = 1$, we have obtained a code tree. The prefix code derived from this tree is a Huffman code.

Example

Consider our earlier example where a, b, c, d, e, and f have frequencies $0.1, 0.2, 0.13, 0.09, 0.4$, and 0.08, respectively.

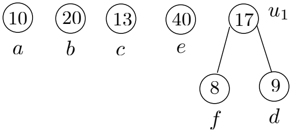
Initially, S has 6 nodes:



The number in each circle represents frequency (e.g., 10 means 10%).

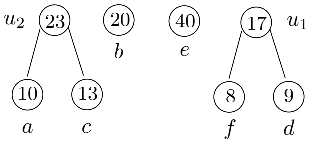
Example

Merge the two nodes with the smallest frequencies 8 and 9. Now S has 5 nodes $\{a, b, c, e, u_1\}$:



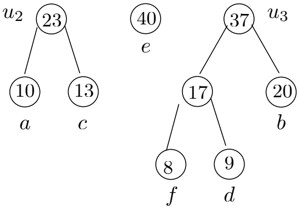
Example

Merge the two nodes with the smallest frequencies 10 and 13. Now S has 4 nodes $\{b, e, u_1, u_2\}$:



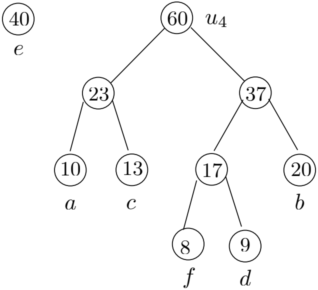
Example

Merge the two nodes with the smallest frequencies 17 and 20. Now S has 3 nodes $\{e, u_2, u_3\}$:



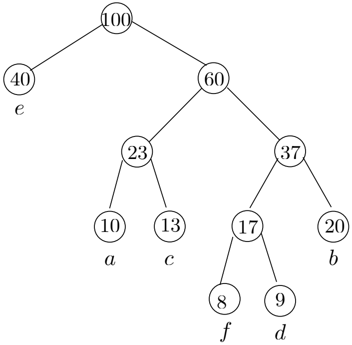
Example

Merge the two nodes with the smallest frequencies 23 and 37. Now S has 2 nodes $\{e, u_4\}$:



Example

Merge the two remaining nodes. Now S has a single node left.



This is the final code tree.

It is easy to implement the algorithm in $O(n \log n)$ time (exercise).

Next, we prove that the algorithm gives an optimal code tree, i.e., one that minimizes the average height.

Property 1

Lemma: In an optimal code tree, every internal node of T must have two children.

The proof is left as a regular exercise.

Property 2

Lemma: Let σ_1 and σ_2 be two letters in Σ with the lowest frequencies. There exists an optimal code tree where σ_1 and σ_2 have the same parent.

Proof: W.l.o.g., assume $freq(\sigma_1) \leq freq(\sigma_2)$. Let T be any optimal code tree. Let p be an arbitrary internal node with the largest level in T. By Property 1, p must have two leaves. Let x and y be letters corresponding to those leaves such that $freq(x) \leq freq(y)$. Swap σ_1 with x and σ_2 with y, which gives a new code tree T'. Note that both σ_1 and σ_2 are children of p in T'.

Convince yourself that the average length of T' is at most that of T. Hence, T' is optimal as well. □

Theorem: Huffman's algorithm produces an optimal prefix code.

Proof: We will prove by induction on the size n of the alphabet Σ.

Base Case: $n = 2$. In this case, the algorithm encodes one letter with 0, and the other with 1, which is clearly optimal.

General Case: Assuming the theorem's correctness for $n = k - 1$ where $k \geq 3$, next we show that it also holds for $n = k$.

Proof (cont.): Let σ_1 and σ_2 be two letters in Σ with the lowest frequencies.

By Property 2, there is an optimal code tree T on Σ where leaves σ_1 and σ_2 are the children of the same parent p.

Let T_{huff} be the code tree returned by Huffman's algorithm on Σ. Convince yourself that σ_1 and σ_2 have the same parent q in T_{huff}.

Proof (cont.): Construct a new alphabet Σ' from Σ by removing σ_1 and σ_2, and adding a letter σ^* with frequency $freq(\sigma_1) + freq(\sigma_2)$.

Let T' be the tree obtained by removing leaves σ_1 and σ_2 from T (thus making p a leaf). T' is a code tree on Σ' where p corresponds to σ^*. Observe:

$$\text{avg height of } T = \text{avg height of } T' + freq(\sigma_1) + freq(\sigma_2).$$

Let T'_{huff} be the tree obtained by removing leaves σ_1 and σ_2 from T_{huff} (thus making q a leaf). T'_{huff} is a code tree on Σ' where q corresponds to σ^*.

$$\text{avg height of } T_{huff} = \text{avg height of } T'_{huff} + freq(\sigma_1) + freq(\sigma_2).$$

Proof (cont.): T'_{huff} is the output of Huffman's algorithm on Σ'.

By our inductive assumption, T'_{huff} is optimal on Σ'. Thus:

$$\text{avg height of } T'_{huff} \leq \text{avg height of } T'$$

Hence:

$$\text{avg height of } T_{huff} \leq \text{avg height of } T.$$

$\square$