Greedy 3: Huffman Codes

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

Coding

Suppose that we have an alphabet Σ (like the English alphabet). The goal of coding is to map each alphabet to a binary string—called a codeword—so that they can be transmitted electronically.

For example, suppose $\Sigma = \{a, b, c, d, e, f\}$. Assume that we agree on $a = 000$, $b = 001$, $c = 010$, $d = 011$, $e = 100$, and $f = 101$. Then, a letter such as "bed" will be encoded as 001100011.

We can, however, achieve better coding efficiency (i.e., producing shorter digital documents) if the frequencies of the letters are known. In general, more frequent letters should be encoded with less bits. The next slide shows an example.

Example

Suppose we know that the frequencies of a, b, c, d, e, f are $0.1, 0.2, 0.13, 0.09, 0.4, 0.08$, respectively.

If we encode each letter with 3 digits, then the average number of digits per letter is obviously 3.

However, if we adopt the encoding of $a = 100$, $b = 111$, $c = 101$, $d = 1101$, $e = 0$, $f = 1100$, the average number of digits per letter is:

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

So in the long run, the new encoding is expected to save $1 - (2.37/3) = 21\%$ of bits!

Example

You probably would ask: why not just encode the letters as: $e = 0, b = 1, c = 00, a = 01, d = 10, f = 11$—namely, encode the next frequent letter using as few bits as possible?

The answer is: you cannot decode a document unambiguously! For example, consider the string 10: how do you know whether this is two letters "be", or just one letter "d"?

This issue arises because the codeword of a letter happens to be a prefix of the codeword of another letter. We, therefore, should prevent this, which has led to an important class of codes in coding theory: the prefix codes (actually "prefix-free" codes would have been more appropriate, but the name "prefix codes" has become a standard).

Example

Consider once again our earlier encoding: $a = 100$, $b = 111$, $c = 101$, $d = 1101$, $e = 0$, $f = 1100$. Observe that the encoding is "prefix free", and hence, allows unambiguous decoding.

For example, what does the following binary string say?

$$1001101010011001101100\,1101$$

The Prefix Coding Problem

An encoding of the letters in an alphabet Σ is a prefix code if no codeword is a prefix of another codeword.

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

Given an encoding, its average length is calculated as

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

The objective of the prefix coding problem is to find a prefix code for Σ that has the smallest average length.

A Binary Tree View

Let us start to attack the prefix coding problem. A key observation is that every prefix code can be represented as a binary tree.

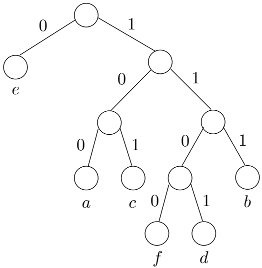
A **code tree** on Σ as a binary tree T satisfying:

- Every leaf node of T is labeled with a distinct letter in Σ; conversely, every letter in Σ is the label of a distinct leaf node in T.
- For every internal node of T, its left edge (if exists) is labeled with 0, and its right edge (if exists) with 1.

The codeword of a letter $\sigma \in \Sigma$ can be obtained by concatenating the bit labels of the edges on the path from the root of T to the leaf σ.

Example

Consider once again our earlier encoding: $a = 100$, $b = 111$, $c = 101$, $d = 1101$, $e = 0$, $f = 1100$. The following is the corresponding code tree:



Think: Why must every letter be at the leaf? (Hint: prefix free)

Average Length from the Binary Tree

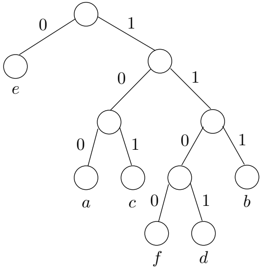
Let T be the code tree corresponding to a prefix code.

Given a letter σ of Σ, denote by $d(\sigma)$ its **depth**, which is the **level** of its leaf node in T (i.e., the number of edges on the path from the root to the leaf).

Clearly, the average length of the prefix code equals

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

Example



The depths of e, a, c, f, d, b are 1, 3, 3, 4, 4, 3, respectively. The average length of the encoding equals

$$freq(e) \cdot 1 + freq(a) \cdot 3 + freq(c) \cdot 3 + freq(f) \cdot 4 + freq(d) \cdot 4 + freq(b) \cdot 3.$$

Huffman's Algorithm

Next, we will present a surprisingly simple algorithm for solving the prefix coding problem. The algorithm constructs a code tree in a bottom-up manner.

Let $n = |\Sigma|$. At the beginning, there are n stand-alone nodes, each corresponding to a different letter in Σ. If letter σ corresponds to a node z, set the frequency of z to $freq(\sigma)$.

Let S be the set of these n nodes.

Huffman's Algorithm

Then, the algorithm repeats the following until S has a single node left:

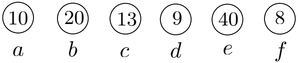
1. Remove from S two nodes u_1, u_2 with the smallest frequencies.
2. Create a node v with u_1, u_2 as its children. Set the frequency of v to be the frequency sum of u_1 and u_2.
3. Insert v into S.

When S has only one node left, we have already obtained the target code tree. The prefix code derived from this code tree is known as a Huffman code.

Example

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

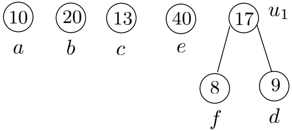
At the beginning, S has 6 nodes:



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

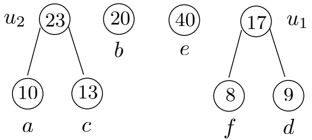
Example

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



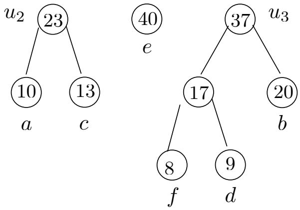
Example

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



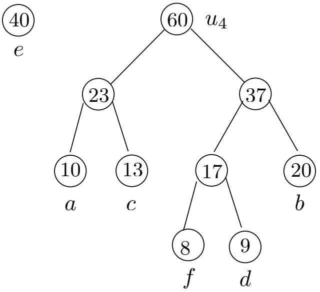
Example

Merge the two nodes with the smallest frequencies 17 and 20. Now S has 3 nodes $\{e, u_1, u_3\}$:



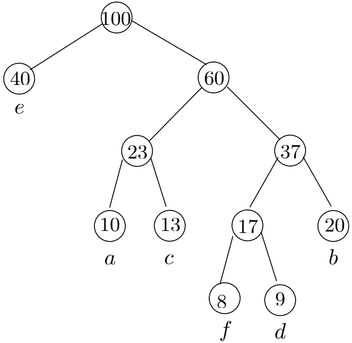
Example

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 should be fairly straightforward for you to implement the algorithm in $O(n \log n)$ time, where $n = |\Sigma|$.

Think: Why do we say the algorithm is greedy?

Next, we prove that the algorithm indeed gives an optimal prefix code, i.e., one that has the smallest average length among all the possible prefix codes.

Crucial Property 1

Lemma: Let T be a code tree corresponds to an optimal prefix code. Then, every internal node of T must have two children.

Proof: Suppose that the lemma is not true. Then, there is an internal node u with only one child node v. Imagine removing u as follows:

- If u is the root, simply make v the new root.
- Otherwise, make v a child node of the parent of u.

The above removal generates a new binary tree whose average length is smaller than that of T, which contradicts the fact that T is optimal. $\square$.

Crucial Property 2

Lemma: Let σ_1 and σ_2 be two letters in Σ with the lowest frequencies. There exists an optimal prefix code whose code tree has σ_1 and σ_2 as two sibling leaves at the deepest level.

Proof: Take an arbitrary prefix code with binary tree T. If σ_1 and σ_2 are indeed sibling leaves at the deepest level, then the claim already holds. Next, we assume that this is not the case.

Suppose T has height h. Recall that the **level** of a node is the number of edges on the path from the root to the node. In other words, the deepest leaves are at $h - 1$. Take an arbitrary internal node p at level $h - 2$. By the previous lemma, p must have two leaves (at level $h - 1$). Let σ_1' and σ_2' be the letters corresponding to those leaves.

Crucial Property 2

Proof (cont.): Now swap σ_1 with σ'_1, and σ_2 with σ'_2, which gives a new binary tree T'. Note that T' has σ_1 and σ_2 as sibling leaves at the deepest level.

How does the average length of T' compare with that of T? As the frequency of σ_1 is no higher than that of σ'_1, swapping the two letters can only decrease the average length of the tree (i.e., as we are assigning a shorter codeword to a more frequent letter). Similarly, the other swap can only decrease the average length.

It follows that the average length of T' is no larger than that of T, meaning that T' is optimal as well. $\square$

Optimality of Huffman Coding

We are now ready to prove:

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 that the theorem holds for $n = k - 1$ ($k \geq 3$), next we show that it also holds for $n = k$.

Optimality of Huffman Coding

Proof (cont.): Let σ_1 and σ_2 be two letters with the lowest frequencies. From Property 2, we know that there is an optimal prefix code whose code tree T has σ_1 and σ_2 as two sibling leaves at the deepest level. Let p be the parent of σ_1 and σ_2.

Construct a new alphabet Σ' that includes all letters in Σ, except σ_1 and σ_2, but a letter p whose frequency equals $f(\sigma_1) + f(\sigma_2)$. Let T' be the tree obtained by removing leaf nodes σ_1 and σ_2 from T (thus making p a leaf).

Let $\mathcal{T}'$ be the binary tree obtained by Huffman's algorithm on Σ'. Since $|\Sigma'| = k - 1$, we know from the inductive assumption that $\mathcal{T}'$ is optimal, meaning that

$$\text{avg length of } \mathcal{T}' \quad \leq \quad \text{avg length of } T'$$

Optimality of Huffman Coding

Proof (cont.): Now consider the binary tree $\mathcal{T}$ produced by Huffman's algorithm on Σ. Clearly, $\mathcal{T}$ extends $\mathcal{T}'$ by simply putting σ_1 and σ_2 as child nodes of p. Hence:

$$\begin{aligned}\text{avg length of } \mathcal{T} &= \text{avg length of } \mathcal{T}' + f(\sigma_1) + f(\sigma_2) \\ &\leq \text{avg length of } T' + f(\sigma_1) + f(\sigma_2) \\ &= \text{avg length of } T.\end{aligned}$$

This indicates that $\mathcal{T}$ also gives an optimal prefix code. □