More on Huffman's Algorithm

We provide additional details on the analysis of Huffman's algorithm given in the textbook.

Given an "alphabet" of symbols V and a positive integer frequency $f(v)$ for every $v \in V$, we want to find a prefix-free way of mapping each symbol $v \in V$ to a bit string $C(v)$ of length $\ell(v)$, in such a way that we minimize the total encoding length

$$\sum_{v \in V} f(v) \cdot \ell(v)$$

A bit string s is a prefix of a bit string s' if s' is obtained by adding one or more bits at the end of s. For example 010 is a prefix of 01001. A mapping is prefix-free if all the encodings $C(v)$ are different and none of them is a prefix of any other.

For example if $V = \{a, b, c, d\}$, then

$$\begin{aligned} C(a) &= 0 \\ C(b) &= 10 \\ C(c) &= 110 \\ C(d) &= 111 \end{aligned}$$

is a prefix-free encoding, while

$$\begin{aligned} C(a) &= 0 \\ C(b) &= 10 \\ C(c) &= 101 \\ C(d) &= 111 \end{aligned}$$

is not prefix free because the encoding of b is a prefix of the encoding of c.

If all the encodings have the same length, then the mapping is always prefix-free. In a fixed-length mapping, the length of the encoding of any element of V must be at least $\log_2 V$, because if we use encodings of length ℓ we can represent at most 2^ℓ symbols.

This means that the total encoding length achieved with fixed-length mappings is at least

$$\lceil \log_2 |V| \rceil \cdot \sum_v f(v)$$

With variable-length prefix-free mappings it is often possible to do better. As said above, we are interested in finding a prefix-free mapping that minimizes the total encoding length.

A prefix-free encoding of V can be represented as a binary tree in which each leaf is associated with an element of V. Each node of the binary tree is labeled by a binary string: the root is labeled with the empty string, and if s is the label of a node, then its left child, if it exists, is labeled $s0$, and the right child, if it exists, is labeled $s1$. The mapping defined by such a binary tree is to map each element $v \in V$ to the label of the leaf associated to v. Note that labelling is such that if a binary string s is a prefix of a binary string s', then the node labeled s' is a descendant of the node labeled s, so by associating elements of V to leaves we are guaranteed the prefix-free property, because leaves have no descendants.

We first note that, in an optimal solution represented as a binary tree, every non-leaf node has precisely two children. This is because if we had a node x with exactly one child y, we could "merge" x and y, and all the symbols that are descendants of y would have encodings one bit shorter than before, meaning that the original tree could not be an optimal solution.

Our starting point in reasoning about optimality is the following lemma.

Lemma 1 *Let V be a set of symbols, $f(v)$ the frequency of symbol $f(v)$, and let $v_1, \ldots, v_n$ be a sorting of V such that $f(v_1) \leq f(v_2) \leq f(v_3) \leq \cdots \leq f(v_n)$, where $n = |V|$.*

Then there is an optimal solution to the prefix-free encoding problem in which v_1 has an encoding at least as long as that of any other symbol.

PROOF: Consider any optimal solution C^*, in which each symbol v has length $\ell(v)$. If v_1 is associated to a deepest leaf of C^* there is nothing to prove. Otherwise, let x be a symbol associated to a deepest leaf of C^*, and consider a new mapping C' that is identical to C^* except that we switched the encodings of x and of v_1, and let us call $\ell'(v)$ the length of the encoding of each symbol v according to this new encoding.

Calling $cost(C)$ the total encoding length of a mapping C, the difference between the

total encoding length in C^* and the total encoding length in C' is

$$\begin{aligned} cost(C^*) - cost(C') &= \left(\sum_{v \in V} \ell(v) f(v)\right) - \left(\sum_{v \in V} \ell(v)' f(v)\right) \\ &= \ell(v_1) f(v_1) + \ell(x) f(x) - \ell'(v_1) f(v_1) - \ell'(x) f(x) \\ &= \ell(v_1) f(v_1) + \ell(x) f(x) - \ell(x) f(v_1) - \ell(v_1) f(x) \\ &= (\ell(x) - \ell(v_1)) \cdot (f(x) - f(v_1)) \\ &\geq 0 \end{aligned}$$

where we used the fact that the old and new encodings of all symbols are the same, except for v_1 and x, then we used the fact that the encoding lengths of x and v_1 are switched in the new encodings, then we used the distributivity property, and finally we used that $\ell(x) \geq \ell(v_1)$ and $f(v_1) \leq f(x)$ because x was a symbol with longest encoding and v_1 was defined as a symbol of smallest frequency.

This means that $cost(C') \leq cost(C^*)$, but C^* was assumed to be optimal so C' is also optimal and we conclude that there is an optimal solution in which v_1 is a symbol with longest encoding. □

Lemma 2 *Let V be a set of symbols, $f(v)$ the frequency of symbol $f(v)$, and let $v_1, \ldots, v_n$ be a sorting of V such that $f(v_1) \leq f(v_2) \leq f(v_3) \leq \cdots \leq f(v_n)$, where $n = |V|$.*

Then there is an optimal solution to the prefix-free encoding problem in which v_1 and v_2 are siblings and have an encoding at least as long as that of any other symbol.

PROOF: From the previous lemma we know that there is an optimal solution C' such that v_1 is associated to a deepest leaf in the binary tree representation of the encoding C'. Since all internal nodes must have exactly two children, the leaf associated to v_1 must have a sibling, and that sibling must also be a leaf (otherwise there would be a symbol with a longer encoding than v_1). If the symbol associated to that sibling leaf is v_2 then we are done. Suppose, otherwise, that it is some other symbol x. Then we have $f(v_2) \leq f(x)$ because of the definition of the sorted order and $\ell(x) \geq \ell(v_2)$ because $\ell(x) = \ell(v_1)$ and v_1 is a symbol with longest encoding in C'. So we can switch x and v_2 and, via the same reasoning of the previous lemma, observe that the solution C'' that we obtain has total encoding length smaller than or equal to the total encoding length of C', so that C'' is also an optimal solution, and v_1 and v_2 are siblings in C'', and are symbols of biggest encoding length. □

Now comes the main observation: if we are given an instance $V, f(\cdot)$ of the minimum-total-length prefix-free encoding problem, and $v_1, v_2, \ldots, v_n$ is a sorted order of V according to f, we can, without loss of generality, optimize among solutions in which

v_1 and v_2 are siblings. Now, the problem of finding the optimal encoding in which v_1 and v_2 are siblings can be transformed into an equivalent encoding problem over $n-1$ symbols. Once we explain how this works, it will become clear that we can repeat this transformation several times until we are left with a problem involving only two symbols, which is always optimally solved by encoding one symbol as 0 and the other as 1.

The transformation is as follows: we take out v_1 and v_2 from V, and replace them with a new symbol z of frequency $f(z) = f(v_1) + f(v_2)$. We do not make any change to the other symbols. The claim is that if C' is an optimal mapping for this new instance over the symbols $z, v_3, \ldots, v_m$, and if we define a mapping C such that $C(v_1) = C'(z)0$, $C(v_2) = C'(z)1$, $C(v_i) = C(v_i)$ for $i = 3, \ldots, n$, then C is optimal for our original problem.

First of all, note that C is a valid prefix-free encoding of our original symbol set, and if let $cost(\cdot)$ denote the total encoding length of a mapping we have

$$cost(C) = cost(C') + f(v_1) + f(v_2)$$

Now consider an optimal solution C^* for our original symbol set, such that v_1 and v_2 are siblings in the binary tree defined by C^* (by our previous lemma, we can assume that such a C^* exists). Define a possible solution C'' for the new problem, by letting $C''(z)$ be equal to the string associated to the parent of v_1 and v_2, and letting C'' be equal to C on $v_3, \ldots, v_n$. Then

$$cost(C'') = cost(C^*) - f(v_1) - f(v_2)$$

Putting things together we have

$$cost(C) = cost(C') + f(v_1) + f(v_2) \leq cost(C'') + f(v_1) + f(v_2) = cost(C^*)$$

where we used the fact that C' is optimal for the new problem. This means that

$$cost(C) \leq cost(C^*)$$

where C^* is optimal for the original problem, and so C is also optimal for the original problem, as claimed.

The algorithm at the end of section 5.2 of the book iteratively applies this transformation, and finds an optimal solution in time $O(n \log n)$.