Correctness and running time of Huffman’s algorithm

Vassos Hadzilacos

We prove the correctness of Huffman’s algorithm by induction on the number of symbols $n$ in the alphabet.

The base case, $n = 2$ is obvious because the only possibility (that is not obviously suboptimal) is a code where both codewords are one bit long, which is what Huffman’s algorithm produces in this case.

Suppose that the algorithm produces an optimal tree for alphabets with $n - 1 \geq 2$ symbols and their associated frequencies. We will prove that it produces an optimal tree for alphabets with $n$ symbols and their associated frequencies.

Let $\Gamma$ be an alphabet with $n$ symbols, and $f(a)$ be the frequency for each $a \in \Gamma$. Let $H$ be the tree produced by Huffman’s algorithm for $\Gamma, f$. We must prove that $H$ is optimal for this input.

By the algorithm, there are two symbols of minimum frequency (according to $f$) that are siblings in $H$; let these symbols be $x$ and $y$. Let $z$ be a new symbol (that is not in $\Gamma$); and let $\Gamma' = (\Gamma - \{x, y\}) \cup \{z\}$ and $f'$ be frequencies of the symbols in $\Gamma'$ defined by

$$f'(a) = \begin{cases} f(a), & \text{if } a \neq z \\ f(x) + f(y), & \text{if } a = z. \end{cases}$$

(Intuitively, we are replacing the symbols $x$ and $y$ with a new symbol $z$, whose frequency is the sum of the frequencies of $x$ and $y$.) Finally, let $H'$ be the tree obtained from $H$ by removing $x$ and $y$ and replacing their parent by $z$. From the definition of weighted average depth, we have

$$\text{ad}(H) = \text{ad}(H') + (f(x) + f(y)). \quad (1)$$

Note that $H'$ is a tree produced by Huffman’s algorithm on input $\Gamma', f'$. $\Gamma'$ has $n - 1$ symbols so, by induction hypothesis,

$$H' \text{ is optimal for } \Gamma', f'. \quad (2)$$

Now, let $T$ be an optimal tree for $\Gamma, f$. Without loss of generality, we can assume that $x$ and $y$ are siblings and are at maximum depth of $T$. (If not, we can move them so that they are siblings at the maximum depth of $T$ without increasing the weighted average depth of the tree, by swapping them with symbols that are siblings at the maximum depth.) Let $T'$ be obtained from $T$ as $H'$ was obtained from $H$. Thus, $T'$ is a tree for $\Gamma', f'$. We have:

$$\text{ad}(T) = \text{ad}(T') + (f(x) + f(y)) \quad \text{[by definition of ad]}$$

$$\geq \text{ad}(H') + (f(x) + f(y)) \quad \text{[by (2)]}$$

$$= \text{ad}(H) \quad \text{[by (1)]}$$

Since $T$ is optimal for $\Gamma, f$, so is $H$. So, Huffman’s algorithm produces optimal trees for alphabets with $n$ symbols and their associated frequencies.

We can implement this algorithm to run in time $O(n \log n)$ using heaps. Let $n$ be the number of symbols in the alphabet, and $f(i)$ be the frequency of the $i$-th symbol, $1 \leq i \leq n$. The algorithm constructs a full
binary tree with \(2n-1\) nodes, each labeled with a positive integer \(i\), \(1 \leq i \leq 2n-1\). Nodes labeled \(1, 2, \ldots, n\) are leaves, where the leaf node labeled \(i\) corresponds to the \(i\)-th symbol. Nodes \(n+1, n+2, \ldots, 2n-1\) are internal nodes, i.e., nodes that are not leaves. (Note that a full binary tree with \(n\) leaves has \(n-1\) internal nodes, and therefore a total of \(2n-1\) nodes. This is easy to prove by induction.)

The algorithm uses a heap \(H\) that stores pairs of the form \(x = (i, p)\) where \(1 \leq i \leq 2n-1\) and \(0 \leq p \leq 1\). The first component of the pair \(x\), denoted \(x.\text{label}\), is the label of a node in the tree that the algorithm constructs. The second component, denoted \(x.\text{freq}\), is the label of a node in the tree that the algorithm stored in the leaves of the subtree rooted at the node labeled \(x.\text{label}\); \(x.\text{freq}\) is used as the priority for ordering the pairs in the heap \(H\). The algorithm expressed in pseudocode is shown below.

\[
\text{Huffman}(n, f) \\
1 \quad \text{for } i := 1 \text{ to } n \text{ do} \\
2 \quad \quad H[i] := (i, f(i)) \\
3 \quad \quad \text{create a leaf node labeled } i \text{ (both children are Nil)} \\
4 \quad \text{BUILDHEAP}(H) \\
5 \quad \text{for } i := n+1 \text{ to } 2n-1 \text{ do} \\
6 \quad \quad x := \text{ExtractMin}(H); y := \text{ExtractMin}(H) \\
7 \quad \quad \text{create a node labeled } i \text{ with children the nodes labeled } x.\text{label} \text{ and } y.\text{label} \\
8 \quad \quad \text{INSERT}(H, (i, x.\text{freq} + y.\text{freq}))
\]

This algorithm runs in \(O(n \log n)\) time: Putting the first \(n\) pairs into \(H\) and creating the \(n\) leaves takes \(O(n)\) time (lines 1–3), and turning \(H\) into a heap using \text{BUILDHEAP} also takes \(O(n)\) time (line 4). The \textbf{for} loop in lines 5–8 is repeated \(n-1\) times. In each iteration we perform two \text{ExtractMin} operations and one \text{Insert} operation, each of which takes \(O(\log n)\) time. So the loop takes \(O(n \log n)\) time, and the entire algorithm takes \(O(n) + O(n) + O(n \log n) = O(n \log n)\) time.