# CS340 Analysis of Algorithms

| | | | |
|---|---|---|---|
| **Handout:** | **4** | **Professor:** | **Dianna Xu** |
| **Title:** | **Huffman Coding** | **E-mail:** | `dxu@cs.brynmawr.edu` |
| **Date:** | | **URL:** | **http://cs.brynmawr.edu/cs340** |

Sample description, pseudo code and proof of correctness for the Huffman coding algorithm. Please refer to lecture notes for details of the algorithm design and time analysis. Recall that given an alphabet $S$ and the probabilities $p(x)$ of occurence for each character $x \in S$, we want to compute a prefix code $T$ that minimizes the total expected length of the encoded string $B(T)$, where

$$B(T) = \sum_{x \in S} p(x) d_T(x)$$

$d_T(x)$ denotes the depth of the leaf representing $x$ on $T$.

## 1 Description

Huffman's encodes the characters via building a binary tree bottom up. All characters start as leaves. Merge the two characters $x$ and $y$ with the lowest probabilites into a new parent node and assign it the probability $p(x)+p(y)$. Repeat until tree is complete. Codes are assigned by performing a DFS on the tree. During the traversal, append 0 to the left branch and 1 to the right branch. Whenever a leaf (representing an input character) is reached, the accumulated binary string is its Huffman code.

## 2 Pseudocode

```
// C is a list of chars modeled as objects storing the associated probabilities
Function Huffman((Node[] C)
    //put the chars into a priority queue Q keyed by frequencies
    for each $x \in C$ do
        add x to Q sorted by x.prob
    end
    n = C.length
    for i = 1 to n-1 do
        z = new Node
        z.left = x = extractMin from Q
        z.right = y = extractMin from Q
        z.prob = x.prob + y.prob
        insert z into Q
    end
    return last element in Q as root
```

## 3 Correctness Proof

Termination is easily argued because $Q$ is finite and we shorten it by one each iteration (2 `extractMin` and 1 `insert`).

We prove the optimality of `Huffman` by showing that any other optimal tree can be converted to a Huffman tree without increasing cost.
We will start with some observations/lemmas.

**Lemma 3.1.** Huffman tree is full binary.

Recall that full binary means that every internal node has either exactly two or zero children. This is clearly true for any Huffman tree, by construction.

**Lemma 3.2.** Any optimal tree is full binary.

We will argue by contradiction. Suppose to the contrary there is some optimal tree with at least one internal node $u$ with a single child $v$. Recall that characters are represented by leaves of the tree and depths of the leaves determine their encoding lengths. Therefore in this case we should simply delete $u$ and replace it with its single child $v$. Nothing else is affected and $v$'s depth is reduced by 1, resulting in a more optimal tree. Contradction. ■.

**Lemma 3.3.** The two characters with the lowest frequencies will be siblings at the max depth of any optimal tree.

**Proof**: by contradiction. Let $x$ and $y$ be the two chars with the lowest probabilities in alphabet $S$, $T$ be any optimal prefix code tree for $S$, and $b$ and $c$ be two siblings at the max depth of $T$, which are distinct from $x$ and $y$. The relative order of the probabilities (or depths) within each specific pair ($b$ verus $c$ or $x$ versus $y$) is not important, because we can always rename the variables, so we will assign an abitrary order. Suppose without loss of generality that $p(b) \leq p(c)$ and $p(y) \leq p(x)$. Now we have

$$p(c) \geq p(b) > p(x) \geq p(y) \tag{1}$$

because $x$ and $y$ have the lowest probabilities. And similarly,

$$d_T(c) \geq d_T(b) > d_T(y) \geq d_T(x) \tag{2}$$

because $b$ and $c$ have max depths.

**Corollary 3.4.** From (1), we have $p(b) > p(x)$ and from (2), we have $d_T(b) > d_T(x)$. Therefore, $p(b) - p(x) > 0$ and $d_T(b) - d_T(x) > 0$.

Now, consider $T$ as shown on the left in Fig 1, which conforms to all of our existing suppositions. We now propose to swap $x$ with $b$ in the tree, resulting in $T'$ as shown on the right.

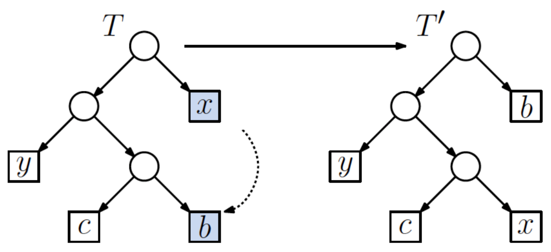

Figure 1: $x$, $y$ have the lowest probabilities, but $b$ and $c$ are at max depth of $T$

How does the cost change as we go from $T$ to $T'$? The only nodes that changed are $b$ and $x$. As shown below, we will compute $B(T')$ by substracting their old costs in $T$ and add in the new costs in $T'$ (see (1)). Notice that because of the swap, $d_T(x) = d_{T'}(b)$ and similiarly, $d_T(b) = d_{T'}(x)$. Therefore we have (2). Steps (3) and (4) are just rearrangements of (2), where we factor out $p(x)$ and $p(b)$ and then $d_T(b) - d_T(x)$, ending in (5). Recall corollary 3.4 stated that $p(b) - p(x) > 0$ and $d_T(b) - d_T(x) > 0$, which we are substracting from $B(T)$, therefore $B(T') < B(T)$.

*Proof.*

$$\begin{aligned}
B(T') &= B(T) - p(x)d_T(x) + p(x)d_{T'}(x) - p(b)d_T(b) + p(b)d_{T'}(b) && (1)\\
&= B(T) - p(x)d_T(x) + p(x)d_T(b) - p(b)d_T(b) + p(b)d_T(x) && (2)\\
&= B(T) + p(x)(d_T(b) - d_T(x)) - p(b)(d_T(b) - d_T(x)) && (3)\\
&= B(T) + (p(x) - p(b))(d_T(b) - d_T(x)) && (4)\\
&= B(T) - (p(b) - p(x))(d_T(b) - d_T(x)) && (5)\\
&< B(T) && \square
\end{aligned}$$

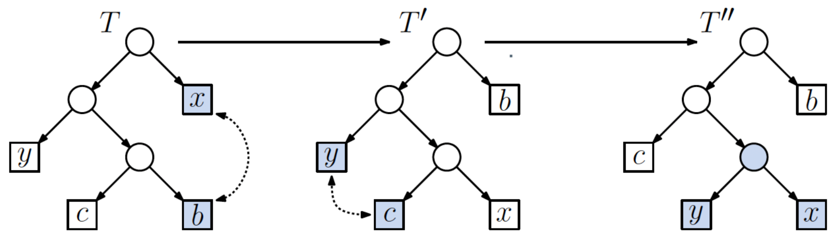

Figure 2: $x$, $y$ have the lowest probabilities, but $b$ and $c$ are at max depth of $T$

By a similiar argument, we can swap $y$ and $c$ to achieve an even more optimal $T''$, as shown in Fig 2. We have now reached a contradiction that $T$ was optimal with the two lowest probabilities nodes NOT at max depth. ■

Finally, we will show the optimality of Huffman coding by induction on the size of the alphabet $S$.

**Proof**: by induction.

- $|S| = 1$, Huffman's tree has a single leaf node and it is optimal.
- Assume that given $n-1$ characters, Huffman's constructs an optimal tree.
- $|S| = n$. Let $x$ and $y$ be the two characters in $S$ with the lowest probability. By Lemma 3.3, they must be at the max depth of some Huffman tree of size $n$. Remove $x$ and $y$ and replace with a new node $z$, where $p(z) = p(x) + p(y)$. Consider the new tree $T$ generated with this new set of $n-1$ characters. By the IH, this Huffman tree is optimal. Now replace $z$ with the $xy$ subtree again, resulting in $T'$. See Fig 3.

  We now compute the cost of $T'$ as shown below. Note that by construction, the depth of $x$ and $y$ in $T'$ is one level lower than the depth of $z$ in $T$, and therefore $d_{T'}(x) = d_T(z) + 1$ and $d_{T'}(y) = d_T(z) + 1$, resulting in (2). (3) and (4) are again algebraic rearrangements, leading to the conclusion in (5), which states $B(T') = B(T) + (p(x) + p(y))$. Observe that $p(x)$ and $p(y)$ are given by the input and are not affected any algorithm, therefore the optimality of $B(T')$ is only dependent on $B(T)$, which is already optimal by the IH. ■

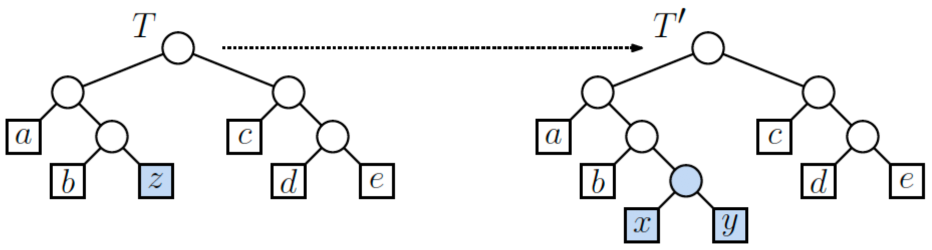

Figure 3: $T$ is optimal. Replace $z$ with $xy$ subtree

*Proof.*

$$\begin{aligned}
B(T') &= B(T) - p(z)d_T(z) + p(x)d_{T'}(x) + p(y)d_{T'}(y) && (1)\\
&= B(T) - (p(x)+p(y))d_T(z) + p(x)(d_T(z)+1) + p(y)(d_T(z)+1) && (2)\\
&= B(T) - (p(x)+p(y))d_T(z) + (p(x)+p(y))(d_T(z)+1) && (3)\\
&= B(T) + (p(x)+p(y))(d_T(z)+1-d_T(z)) && (4)\\
&= B(T) + (p(x)+p(y)) && \square
\end{aligned}$$