

Huffman Trees are Optimal

In section 16.3 the authors give a proof that code trees produced by the Huffman algorithm are optimal. Their argument is difficult to follow, so here is an alternative proof that may be easier to understand.

Recasting the Huffman Algorithm as a Recursion

In the listing on page 388 the authors provide pseudocode for the Huffman algorithm. The problem with this pseudocode is that it represents an optimized version of the algorithm. In particular, the optimized version has optimized away a recursion that is essential for understanding the proof of optimality. The first step in constructing a better proof is to bring back the sub-optimal, recursive version of the algorithm.

```
Huffman(C,F,T)
  if |C| > 1 then
    x,y <- Min2(C,F)
    C <- C - {x,y} + {z}
    F[z] <- F[x] + F[y]
    T[z] <- MakeNode(z,F[z])
    C <- Huffman(C,F,T)
    T[z] <- MakeTree(T[x],T[y])
  return C
```

This recursive function does its work by first reducing the character set to a smaller character set by removing the characters x and y with the smallest frequencies and replacing them with a character z whose frequency is the sum of the frequencies for x and y . It then uses recursion to construct an optimal code tree for the smaller character set. Finally, it modifies the tree returned by the recursive call by replacing the node for z in that tree with a subtree with children x and y .

The recursive function comes with a wrapper that constructs a node for each character in the original character set and then calls the recursive function to do the work of constructing the tree.

```
Huffman(C,F)
  for n <- 1 to |C|
    do T[C[n]] <- MakeNode(C[n],F[n])
  C <- Huffman(C,F,T)
  return T[C[1]]
```

A Proof of Optimality for the Recursive Version

The method I am going to use to prove the optimality of the algorithm above is based on the method typically used to construct proofs of correctness for recursive algorithms, induction on the size of the input C . In this case we will use an induction argument to prove that the algorithm above constructs optimal code trees for all character sets.

The measure that we are trying to optimize for any code tree T is a measure of the average code length for characters in C :

$$B(T) = \sum_{c \in C} d_T(c) F(c)$$

We will prove that the tree T returned by the Huffman algorithm minimizes $B(T)$. To do this proof, we use versions of lemmas 16.2 and 16.3 in the text. Please see the text for proofs of these two lemmas.

Lemma 16.2 Let x and y be two characters with the smallest frequencies in a character set C . Let T be any code tree for C . Find an internal node in T with maximal depth, and let a and b be the leaves that are children of that node. If T' is the tree formed from T by swapping x and y with a and b , then we have that $B(T') \leq B(T)$. (Another way to state this lemma is to say that there is an optimal code tree for C in which x and y are siblings.)

Lemma 16.3 Let T be any tree in which x and y are siblings. If we construct a tree T' by removing the subtree containing x and y and replacing it with a leaf z with frequency $F(z) = F(x) + F(y)$, then $B(T) = B(T') + F(x) + F(y)$.

With these two lemmas we are ready to do the induction proof.

The base case for the induction proof is a character set C with $|C| = 1$. For this case, the code above returns a tree consisting of a single node, which is optimal for that character set.

For the induction step, we assume that the algorithm returns an optimal code tree for character sets of size n or less and use that to prove that the algorithm returns an optimal code tree for a character set C with $|C| = n + 1$. Let T' be the tree returned by the recursive call. By the induction assumption this tree is optimal for its character set.

The tree T that we will return is essentially the tree T' with the node for z replaced by a sub-tree with children x and y . Let us now suppose by way of contradiction that there is a tree S for C which is better than T .

$$B(S) < B(T)$$

Using Lemma 16.2 we can claim that x and y are siblings in S . By construction, x and y are also siblings in T . Now replace the subtree containing x and y with a single node z with frequency $F(z) = F(x) + F(y)$ in both S and T . By construction, this transforms T to T' . It also transforms S into a new tree S' . Lemma 16.3 now tells us that

$$B(T) = B(T') + F(x) + F(y)$$

$$B(S) = B(S') + F(x) + F(y)$$

Combining the last three equations now gives us

$$B(S') + F(x) + F(y) = B(S) < B(T) = B(T') + F(x) + F(y)$$

or

$$B(S') < B(T')$$

This contradicts the induction assumption, which says that T' is optimal for its character set. The only way out of this contradiction is to recognize that T must have been optimal for C in the first place.