Abstract Huffman Coding and PIFO Tree Embeddings

Keri D’Angelo*    Dexter Kozen†
Cornell University
Computer Science Department
Ithaca, New York 14853-7501, USA
*kd349@cornell.edu    †kozen@cs.cornell.edu

November 18, 2022

Abstract

Algorithms for deriving Huffman codes and the recently developed algorithm for compiling PIFO trees to trees of fixed shape [1] are similar, but work with different underlying algebraic operations. In this paper, we exploit the monadic structure of prefix codes to create a generalized Huffman algorithm that has these two applications as special cases.

1 Introduction

Huffman codes translate letters from a fixed alphabet to $d$-ary codewords, achieving optimal compression for a given frequency distribution of letters. There is a well-known greedy algorithm for producing Huffman codes from a given distribution (see [2]).

A new data structure called a PIFO tree (priority-in first-out) has recently been proposed for implementing a wide range of packet scheduling algorithms in programmable network routers [3, 4]. A PIFO tree is a tree of priority queues. Currently, most routers support just a few scheduling algorithms such as strict priority or weighted fair queueing, which are baked into the hardware. The schedulers can be configured to some extent, but it is generally not possible to implement more sophisticated scheduling algorithms that require reordering of already queued packets. This is exactly what PIFO trees permit. It seems likely that PIFOs will be supported on network devices in the near future.

Some researchers have already begun to explore how the PIFO abstraction can be emulated on conventional routers [4]. In very recent work [1], it was shown how to compile an algorithm designed for a PIFO tree of arbitrary shape to a PIFO tree of fixed shape, perhaps a complete $d$-ary tree, that might be implemented in hardware, with negligible performance degradation.
The embedding algorithm is greedy and very similar to the Huffman algorithm, except that it is based on different algebraic operations. For Huffman coding, one wishes to choose a d-ary prefix code C so as to minimize the value of \( \sum_{x \in C} |x| \cdot r(x) \), where \( r(x) \) is the frequency of the letter assigned to the codeword \( x \). This minimizes the entropy of the resulting code. For PIFO trees, one wishes to minimize \( \max_{x \in C} |x| + r(x) \), where \( r(x) \) is the height of a subtree. This minimizes the height of the resulting d-ary tree and determines whether an embedding is at all possible.

This similarity leads us to seek a unified axiomatic treatment that is parametric in the algebraic operations and that can be instantiated to produce both applications as special cases. Our treatment exploits the monadic structure of prefix codes to obtain an abstract formulation of the problem and its solution. We identify sufficient conditions for our abstract algorithm to produce optimal solutions, where the meaning of optimal is also parametric in the instantiation.

We state axioms that are sufficient for optimality in §2. The algorithm is presented in §3 and its correctness proved in §4. The two applications of Huffman codes and PIFO trees are derived in §5.

2 Axioms

Let \( \mathcal{C} : \text{Set} \to \text{Set} \) be an endofunctor in which

- \( \mathcal{C}X \) is the set of pairs \((C, r)\) such that \( C \) is a prefix code over a d-ary alphabet for some arbitrary but fixed \( d \geq 2 \) and \( r : C \to X \), and
- for \( h : X \to Y \), \( \mathcal{C}h : \mathcal{C}X \to \mathcal{C}Y \) with \( \mathcal{C}h(C, r) = (C, h \circ r) \).

Recall that a monad is a triple \((\mathcal{T}, \eta, \mu)\) where \( \mathcal{T} : C \to C \) is an endofunctor on a category \( C \), \( \eta_X : X \to \mathcal{T}X \) and \( \mu_X : \mathcal{T}\mathcal{T}X \to \mathcal{T}X \) are natural transformations called the unit and multiplication respectively, that satisfy \( \mu_X \circ \mathcal{T}\eta_X = \mu_X \circ \eta_X \mu_X = \mu_X \circ \mathcal{T}\mu_X = \mu_X \circ \mathcal{T}\eta_X \) for all objects \( X \in C \).

In our case, the functor \( \mathcal{C} \) carries a natural monad structure with unit \( \eta_X : X \to \mathcal{C}X \) and multiplication \( \mu_X : \mathcal{C}^2X \to \mathcal{C}X \) given by: for \( a \in X \) and \((C, r) \in \mathcal{C}^2X \) with \( r(x) = (C_x, r_x) \),

\[
\eta_X(a) = (\{\varepsilon\}, \varepsilon \mapsto a) \quad \mu_X(C, r) = (\{xy \mid x \in C, y \in C_x\}, xy \mapsto r_x(y)).
\]

The map \( xy \mapsto r_x(y) \) is well defined, as the string \( xy \) is uniquely splittable into \( x \in C \) and \( y \in C_x \) because \( C \) is a prefix code.

Given a monad \((\mathcal{T}, \eta, \mu)\) on a category \( C \), a \( \mathcal{T} \)-algebra is a pair \((X, \gamma)\), where \( \gamma : \mathcal{T}X \to X \) is a morphism of \( C \) and satisfies \( \gamma \circ \eta_X = \mathcal{T}\gamma \) and \( \gamma \circ \mathcal{T}\gamma = \gamma \circ \mu_X \). These objects form the so-called Eilenberg-Moore category over the monad \( \mathcal{T} \), where morphisms are structure-preserving maps; that is, a morphism \( f : (X, \gamma) \to (Y, \delta) \) of \( \mathcal{T} \)-algebras is a morphism \( f : X \to Y \) of the underlying category \( C \) such that \( f \circ \gamma = \delta \circ \mathcal{T}f \).

Suppose there is a fixed (Eilenberg-Moore) algebra \((W, w)\) with \( w : \mathcal{C}W \to W \). We call the elements of \( W \) weights and \((W, w)\) a weighting. The map \( w \) tells how to assign a weight
to the object \((C, r) \in \mathcal{CW}\), given the weights of its leaves as specified by \(r\). We assume that \(W\) is totally preordered by \(\leq\); that is, \(\leq\) is reflexive and transitive, and for all \(x, y \in W\), either \(x \leq y\) or \(y \leq x\) (or both). We write \(x \equiv y\) if both \(x \leq y\) and \(y \leq x\).

Suppose further that we have total preorder on \(\mathcal{CW}\) and \(\mathcal{C}^2\mathcal{W}\), also denoted \(\leq\), satisfying the following properties.

(i) If \(f : C \to D\) is injective and length-nondecreasing, and if \(r \leq s \circ f\) pointwise, then \((C, r) \leq (D, s)\). This says that more or longer codewords or larger leaf values cannot cause a decrease in the order \(\leq\).

(ii) (Exchange property) If \(r(x) \leq r(y), |x| \leq |y|,\) and

\[
    s(z) = \begin{cases} 
    r(x), & \text{if } z = y, \\
    r(y), & \text{if } z = x, \\
    r(z), & \text{if } z \in C \setminus \{x, y\},
    \end{cases}
\]

then \((C, s) \leq (C, r)\). That is, it never hurts to swap a larger element deeper in the tree with a smaller element higher in the tree.

(iii) The monad structure maps \(\eta_W : W \to \mathcal{CW}\) and \(\mu_W : \mathcal{C}^2\mathcal{W} \to \mathcal{CW}\) are monotone with respect to \(\leq\).

Some consequences of (i) are

- If \(f : C \to D\) is injective and length nondecreasing, then \((C, s \circ f) \leq (D, s)\). Thus lengthening codewords cannot cause \(\leq\) to decrease.

- If \(f : C \to D\) is bijective and length-preserving, then \((C, s \circ f) \equiv (D, s)\). This says that the order \(\leq\) on trees depends only on the lengths of the codewords in \(C\), not on the actual codewords themselves.

- If \(r, s : C \to W\) and \(r \leq s\) pointwise, then \((C, r) \leq (C, s)\). Thus larger leaves cannot cause \(\leq\) to decrease.

We assume these properties hold for the algorithm described in the next section.

For \((C, r), (D, s) \in \mathcal{CW}\), let us write \((C, r) \sim (D, s)\) if the multisets of weights represented by the two objects are the same; that is, there is a bijective function \(f : C \to D\) such that \(r = s \circ f\). A tree \((C, r) \in \mathcal{CW}\) is defined to be optimal (for its multiset of weights) if \((C, r)\) is \(\leq\)-minimum in its \(\sim\)-class; that is, \((C, r) \leq (D, s)\) for all \((D, s)\) such that \((C, r) \sim (D, s)\).

3 Algorithm

Suppose we are given a multiset \(M\) of weights in \(W\), \(|M| \geq 2\). We would like to find an optimal tree for this multiset of weights. The following is a recursive algorithm to find such an optimal tree.
1. Say there are \( n \geq 2 \) elements in \( M \). Let \( k \in \{2, \ldots, d\} \) such that \( n \equiv k \mod (d - 1) \). Let \( a_0, \ldots, a_{k-1} \) be the \( k \) elements of least weight. Form the object

\[
(\{0,1, \ldots, k-1\}, i \mapsto a_i) \in \mathcal{CW}.
\]

If there are no other elements of \( M \), return that object.

2. Otherwise, let

\[
M' = \{(\{0,1, \ldots, k-1\}, i \mapsto a_i)\} \cup \{\eta_W(a) \mid a \in M \setminus \{a_0, \ldots, a_{k-1}\}\},
\]

a multiset of \( n - k + 1 \) \( < n \) elements of \( \mathcal{CW} \).

3. Recursively call the algorithm at step 1 with \( M'' = \{\mathcal{CW}(E,t) \mid (E,t) \in M'\} \), a multiset of elements of \( W \). This returns a tree \((D,s)\) of type \( \mathcal{CW} \) that is optimal for \( M'' \). The bijective map \( s : D \to M'' \) factors as \( \mathcal{CW} \circ s' \) for some bijective \( s' : D \to M' \), and \((D,s') \in \mathcal{C}^2W\).

Flatten this to \( \mu_W(D,s') \in \mathcal{CW} \) and return that value.

Note that the number of items combined in step 1 will be \( d \) in all recursive calls except possibly the first. This is because in every step, if \( k \in \{2,3, \ldots, d\} \), then after that step the number of remaining elements will be \( (c(d-1) + k) - k + 1 = c(d-1) + 1 \), which is congruent to \( d \mod d - 1 \), so \( d \) elements will be taken in the next step. But from that point on, it is an invariant of the loop that the number of elements remaining is \( 1 \) mod \( d - 1 \), since in each step we remove \( d \) elements and add one back, decreasing the number by \( d - 1 \).

4 Correctness

In this section, we prove the correctness of the algorithm, making use of the following lemma.

**Lemma 1.** Let \( k \in \{2,3, \ldots, d\} \) and \( k \equiv |M| \mod (d - 1) \). Let \( a_0, \ldots, a_{k-1} \) be the \( k \) elements of \( M \) of least weight. There is an optimal tree in \( \mathcal{CW} \) in which \( a_0, \ldots, a_{k-1} \) are sibling leaves at the deepest level and have no other siblings.

**Proof.** Let \((C,r) \in \mathcal{CW}\) be optimal. Axiom (i) allows us to transform \((C,r)\) so that there are no deficient nodes (nodes with fewer than \( d \) children) at any level except the deepest, and only one deficient node at the deepest level. Thus we can assume without loss of generality that there are \( k \) elements \( x_0, \ldots, x_{k-1} \in C \) of maximum length \( n \) in \( C \) with a common prefix of length \( n - 1 \), and no other \( y \in C \) has that prefix. Say the \( x_0, \ldots, x_{k-1} \) are listed in nondecreasing order of \( r(x_i) \); that is, \( r(x_i) \leq r(x_j) \) for all \( 0 \leq i \leq j \leq k-1 \). Let
\( y_0, \ldots, y_{k-1} \in C \) such that \( r(y_i) = a_i \). Since the \( a_i \) are minimal, \( r(y_i) \leq r(x_i) \). Because the \( |x_i| \) are of maximum length, \( |y_i| \leq |x_i| \). Now we can swap using axiom (ii). Let

\[
s(z) = \begin{cases} 
  r(x_i), & \text{if } z = y_i, \\
  r(y_i), & \text{if } z = x_i, \\
  r(z), & \text{otherwise}.
\end{cases}
\]

Then \((C, s) \leq (C, r)\). But since \((C, r)\) was optimal, \((C, r) \equiv (C, s)\) and \((C, s)\) is also optimal. \( \square \)

**Theorem 1.** The algorithm of §3 produces an optimal tree.

**Proof.** By induction on \( n \). The basis is \( n \leq d \), in which case the result is straightforward.

Suppose that we have a multiset \( M \) of \( n > d \) elements of \( W \). Let \((C, r)\) be an optimal tree for \( M \). Let \( k \in \{2, 3, \ldots, d\} \) be congruent mod \( d - 1 \) to \(|M|\). Let \( a_0, \ldots, a_{k-1} \) be the \( k \) smallest elements of \( M \). By Lemma 1, we can assume without loss of generality that \( a_0, \ldots, a_{k-1} \) are siblings and occur at maximum depth in \((C, r)\), so there exist strings \( x_0, x_1, \ldots, x(k - 1) \in C \) of maximum length and \( r(x_i) = a_i \). Remove the strings \( x \) from \( C \) and replace them with \( x \). Call the resulting set \( C' \). Let

\[
r'(z) = \begin{cases} 
  \{0, 1, \ldots, k - 1\}, i \mapsto a_i \in C W, & \text{if } z = x, \\
  \eta_W(r(z)) \in C W, & \text{otherwise}.
\end{cases}
\]

Then \((C', r') \in C^2 W \) and \((C, r) = \mu_W(C', r')\). The multiset of values of \( r' \) is just the \( M' \) of step 2 of the algorithm.

The algorithm will form the multiset \( M'' = \{\eta_W(E, t) \mid (E, t) \in M'\} \) and recursively call with these weights. By the induction hypothesis, the return value will be a tree \((D, s) \in C W\) that is optimal for \( M'' \), thus \((D, s) \leq \eta_W(C', r') \equiv (C', w \circ r')\), thus there exists a bijective map \( f : D \to C' \) such that \( s = w \circ r' \circ f \). Let \( s' = r' \circ f \). Then \((D, s') = (D, r' \circ f) \leq (C', r')\), and

\[
\mu_W(D, s') \leq \mu_W(C', r') = (C, r),
\]

since \( \mu_W \) is monotone. As \((C, r)\) was optimal, so is \( \mu_W(D, s') \), and this is the value returned by the algorithm. \( \square \)

### 5 Applications

By choosing two specific Eilenberg-Moore algebras \((W, w)\) and defining the ordering relations \( \leq \) appropriately, we can recover two special cases of this algorithm.
5.1 Huffman coding

Our first application is Huffman codes. Here we wish to minimize the expected length of variable-length codewords, given frequencies of the letters to be coded. For this application, we take \( W = \mathbb{R}_+ = \{ a \in \mathbb{R} \mid a \geq 0 \} \) with weighting

\[
w(C, r) = \sum_{x \in C} r(x).
\]

Recall that for \( a \in W \) and \((C, r) \in \mathcal{C}^2W\) with \( r(x) = (C_x, r_x) \),

\[
\eta_W(a) = (\{ \varepsilon \}, \varepsilon \mapsto a) \quad \mu_W(C, r) = (\{ xy \mid x \in C, y \in C_x \}, xy \mapsto r_x(y)).
\]

Then \((W, w)\) is an Eilenberg-Moore algebra for the monad \((\mathcal{C}, \mu, \eta)\), as

\[
w(\eta_W(a)) = w(\{ \varepsilon \}, \varepsilon \mapsto a) = \sum_{x \in \{ \varepsilon \}} (\varepsilon \mapsto a)(x) = a,
\]

\[
w(\mu_W(C, r)) = \sum_{x \in C} \sum_{y \in C_x} r_x(y) = \sum_{x \in C} w(C_x, r_x)
= \sum_{x \in C} r(x) = w(C, w \circ r) = w(\mathcal{C}w(C, r)).
\]

In addition, let us define \( \alpha : \mathcal{C}W \to W \) by

\[
\alpha(C, r) = \sum_{x \in C} |x| \cdot r(x).
\]

Lemma 2.

\[
\alpha(\eta_W(a)) = 0 \quad \alpha(\mu_W(C, r)) = \alpha(C, w \circ r) + w(C, \alpha \circ r).
\]

Proof.

\[
\alpha(\eta_W(a)) = \alpha(\{ \varepsilon \}, \varepsilon \mapsto a) = \sum_{x \in \{ \varepsilon \}} |x| \cdot (\varepsilon \mapsto a)(x) = |\varepsilon| \cdot a = 0,
\]

\[
\alpha(\mu_W(C, r)) = \alpha(\{ xy \mid x \in C, y \in C_x \}, xy \mapsto r_x(y))
= \sum_{x \in C} \sum_{y \in C_x} |xy| \cdot r_x(y) = \sum_{x \in C} |x| \sum_{y \in C_x} r_x(y) + \sum_{x \in C} \sum_{y \in C_x} |y| \cdot r_x(y)
= \sum_{x \in C} |x| \cdot w(C_x, r_x) + \sum_{x \in C} \alpha(C_x, r_x)
= \sum_{x \in C} |x| \cdot w(r(x)) + \sum_{x \in C} \alpha(r(x))
= \alpha(C, w \circ r) + w(C, \alpha \circ r). \quad \square
\]

Note that \( \alpha \) and \( w \) agree on trees of depth one:

\[
w(\{0, 1, \ldots, k - 1\}, i \mapsto a_i) = \sum_{i=0}^{k-1} a_i,
\]

\[
\alpha(\{0, 1, \ldots, k - 1\}, i \mapsto a_i) = \sum_{i=0}^{k-1} |i| \cdot a_i = \sum_{i=0}^{k-1} a_i.
\]
where \(|i|\) refers to the length of \(i\) as a string, which in this case is 1.

The map \(\alpha\) is related to Shannon entropy \(H\). If \(r(x) = d^{-|x|}\), the probability of a \(d\)-ary codeword \(x\) under the uniform distribution on a \(d\)-ary alphabet, then

\[
H(C, r) = \sum_{x \in \mathcal{C}} d^{-|x|} \log d^{-|x|} = \sum_{x \in \mathcal{C}} |x| \cdot d^{-|x|} \log d = \alpha(C, r) \log d,
\]

so \(\alpha(C, r) = H(C, r) / \log d\).

To use the algorithm in §3, we need an order \(\leq\) on \(\mathcal{C}W\). Define \((C, r) \leq (D, s)\) if \((C, r) \sim (D, s)\), that is, there is a bijective map \(f : C \to D\) such that \(r = s \circ f\), and

\[
\alpha(C, r) \leq \alpha(D, s).
\]

Note that if \((C, r) \leq (D, s)\), then

\[
w(C, r) = \sum_{x \in \mathcal{C}} r(x) = \sum_{x \in \mathcal{C}} s(f(x)) = \sum_{y \in \mathcal{D}} s(y) = w(D, s).
\]

For \((C, r), (D, s) \in \mathcal{C}^2W\), let

\[
(C, r) \leq (D, s) \Leftrightarrow \alpha(\mathcal{C}w(C, r)) \leq \alpha(\mathcal{C}w(D, s))
\]

\[
\Leftrightarrow \alpha(C, w \circ r) \leq (D, w \circ s).
\]

If \((C, r) \leq (D, s)\) in \(\mathcal{C}^2W\), then

\[
w(C, \alpha \circ r) = \sum_{x \in \mathcal{C}} \alpha(r(x)) = \sum_{x \in \mathcal{C}} \alpha(s(f(x))) = \sum_{y \in \mathcal{D}} \alpha(s(y)) = w(D, \alpha \circ s). \quad (1)
\]

Trivially, \(\eta_\mu : W \to \mathcal{C}W\) is monotone with respect to \(\leq\).

**Lemma 3.** \(\mu_W : \mathcal{C}^2W \to \mathcal{C}W\) is monotone with respect to \(\leq\).

**Proof.** Suppose \((C, r), (D, s) \in \mathcal{C}^2W\) and \((C, r) \leq (D, s)\). By Lemma 2 and (1),

\[
\alpha(\mu_W(C, r)) = \alpha(C, w \circ r) + w(C, \alpha \circ r)
\]

\[
\leq \alpha(D, w \circ s) + w(D, \alpha \circ s) = \alpha(\mu_W(D, s)).
\]

**Theorem 2.** The algorithm in §3 for the algebra \((\mathbb{R}_+, \mu)\) and ordering relation \(\leq\) defined by \(\alpha\) is equivalent to Huffman’s algorithm and produces an optimal Huffman code for a given multiset of weights.

**Proof.** Take \(X \subset \mathbb{R}_+\) to be a finite multiset and sort the set \(X\) in increasing order. For the binary case of Huffman codes (the \(d\)-ary version follows the same way), we always choose \(k = 2\). For the first step, let \(a_0, a_1 \in X\) be the two smallest elements in the list. Form the object \((\{0, 1\}, i \mapsto a_i) \in \mathcal{C}X\). In the case \(n = 2\), this is the only remaining object in the list. Otherwise, we combined them into one element with the sum of the weights of \(a_0\) and \(a_1\) as the weight of the new element, exactly as the Huffman coding does.
For the case \( n > 2 \), there are remaining elements in the set \( X \). Take all remaining \( a \in X \setminus \{a_0, a_1\} \) and replace \( a \) by \( \eta_X(a) \). We are left with \( n - 1 \) elements of type \( C^X \). If we recursively call the algorithm in step 1, we are continually combining the least two elements in the remaining set with the elements weighted by \( w \). Note by the weighting \( w, w(\eta_X(a)) = a \) and on elements in \( C^X, w \) takes the sum of \( r(x)'s \), exactly as Huffman coding does. Finally, this leaves us with a tree in \( C \) of \( \eta \). This gives an Eilenberg-Moore algebra \( d \) the height of the target \( w \) algorithm of \( \S 3 \), where we choose \( d \) an arbitrary PIFO tree and embeds it into a \( \epsilon \). Denote this tree by \( (D, s) \). Taking \( \mu_X(D, S) \) gives our desired tree in \( C^X \).

\[ \square \]

### 5.2 PIFO trees

PIFO trees were introduced in [3] as a model for programmable packet schedulers. In the recent work of [1], further work was done on PIFO trees giving a semantics that allows for certain embedding algorithms. The definition of a homomorphic embedding was given for the purpose finding when a PIFO tree could be represented by another PIFO tree, and what such an embedding would look like. The embedding algorithm we consider takes an arbitrary PIFO tree and embeds it into a \( d \)-ary tree. This becomes a special case of the algorithm of \( \S 3 \), where we choose \( w \) in the Eilenberg-Moore algebra \( (W, w) \) to minimize the height of the target \( d \)-ary tree into which the source tree can embed.

For this application, we take \( W = \mathbb{N} \) with weighting

\[ w(C, r) = \max_{x \in C} |x| + r(x). \]

This gives an Eilenberg-Moore algebra \( (W, w) \) for the monad \( (\epsilon, \mu, \eta) \). For \( a \in W \) and \( (C, r) \in \epsilon^2W \) with \( r(x) = (C, r_x) \), as before we have

\[ \eta_W(a) = (\{\epsilon\}, \epsilon \mapsto a) \]

\[ \mu_W(C, r) = (\{xy \mid x \in C, y \in C_x\}, xy \mapsto r_x(y)), \]

so

\[ w(\eta_W(a)) = w(\{\epsilon\}, \epsilon \mapsto a) = \max_{x \in \{\epsilon\}} |x| + (\epsilon \mapsto a)(x) = |\epsilon| + a = a, \]

\[ w(\mu_W(C, r)) = w(\{xy \mid x \in C, y \in C_x\}, xy \mapsto r_x(y)) = \max_{x \in C} \max_{y \in C_x} |xy| + r_x(y) \]

\[ = \max_{x \in C} \max_{y \in C_x} |x| + |y| + r_x(y) = \max_{x \in C} |x| + \max_{y \in C_x} |y| + r_x(y) \]

\[ = \max_{x \in C} |x| + w(C, r_x) = \max_{x \in C} |x| + w(r(x)) \]

\[ = w(C, w \circ r) = w(\epsilon w(C, r)). \]

For \( (C, r), (D, s) \in \epsilon W \), let us define \( (C, r) \leq (D, s) \) if there is a bijective function \( f : C \to D \) such that \( r = s \circ f \) and

\[ w(C, r) \leq w(D, s). \]

For \( (C, r), (D, s) \in \epsilon^2W \), let

\[ (C, r) \leq (D, s) \iff w(\epsilon w(C, r)) \leq w(\epsilon w(D, s)). \]
Lemma 4. $\mu_W : \mathcal{C}^2 W \to \mathcal{C} W$ is monotone with respect to $\leq$.

Proof. Suppose $(C, r), (D, s) \in \mathcal{C}^2 W$ and $(C, r) \leq (D, s)$. By (2),

$$w(\mu_W(C, r)) = w(\varepsilon w(C, r)) \leq w(\varepsilon w(D, s)) = w(\mu_W(D, s)).$$

Theorem 3. The algorithm in §3 for the algebra $(\mathbb{N}, w)$ and ordering relation $\leq$ defined by $w$ and is equivalent to determining whether an embedding of a PIFO tree into a bounded $d$-ary tree exists and finding the embedding if so.

6 Conclusion

We have presented a generalized Huffman algorithm and shown that two known algorithms, Huffman codes and embedding of PIFOs trees, can be derived as special cases. The PIFO embedding algorithm was introduced in [1] and observed to be very similar to the usual combinatorial algorithm for optimal Huffman codes, albeit based on a different algebraic structure. This suggested the common generalization presented in this paper.

Our generalized algorithm exploits the monadic structure of prefix codes, which allows a more algebraic treatment of the Huffman algorithm than the usual combinatorial approaches. The two applications fit naturally in the categorical setting by choosing specific Eilenberg-Moore algebras for each one. It is possible that other greedy algorithms might fit into this framework as well.

References


