Design and Analysis of Algorithms

Mohammad GANJTABESH

mgtabesh@ut.ac.ir

School of Mathematics, Statistics and Computer Science,
University of Tehran,
Tehran, Iran.

Techniques for the design of Algorithms

The classical techniques are as follows:

1. Divide and Conquer
2. Dynamic Programming
3. Greedy Algorithms
4. Backtracking Algorithms
5. Branch and Bound Algorithms

Huffman Codes

- Each character appears with some frequency in a text.

Huffman Codes

- Each character appears with some frequency in a text.
- A binary code assigns a string of 0s and 1s to each character in the alphabet.

Huffman Codes

- Each character appears with some frequency in a text.
- A binary code assigns a string of 0s and 1s to each character in the alphabet.
- A binary code is prefix-free if no code is a prefix of any other.

Huffman Codes

- Each character appears with some frequency in a text.
- A binary code assigns a string of 0s and 1s to each character in the alphabet.
- A binary code is prefix-free if no code is a prefix of any other.
- Any prefix-free binary code can be visualized as a binary tree with the encoded characters stored at the leaves.

Huffman Codes

- Each character appears with some frequency in a text.
- A binary code assigns a string of 0s and 1s to each character in the alphabet.
- A binary code is prefix-free if no code is a prefix of any other.
- Any prefix-free binary code can be visualized as a binary tree with the encoded characters stored at the leaves.
- The code word for any symbol is given by the path from the root to the corresponding leaf; 0 for left, 1 for right.

Huffman Codes

- Each character appears with some frequency in a text.
- A binary code assigns a string of 0s and 1s to each character in the alphabet.
- A binary code is prefix-free if no code is a prefix of any other.
- Any prefix-free binary code can be visualized as a binary tree with the encoded characters stored at the leaves.
- The code word for any symbol is given by the path from the root to the corresponding leaf; 0 for left, 1 for right.
- The length of a codeword for a symbol is the depth of the corresponding leaf.

Huffman Codes

- Each character appears with some frequency in a text.
- A binary code assigns a string of 0s and 1s to each character in the alphabet.
- A binary code is prefix-free if no code is a prefix of any other.
- Any prefix-free binary code can be visualized as a binary tree with the encoded characters stored at the leaves.
- The code word for any symbol is given by the path from the root to the corresponding leaf; 0 for left, 1 for right.
- The length of a codeword for a symbol is the depth of the corresponding leaf.
- Fixed-length code: where a fixed-length code is assigned to each character.
- Variable-length code: can do considerably better than a fixed-length code, by giving frequent characters short codewords and infrequent characters long codewords.

Huffman Codes: Example

Example

Suppose that there is a text of length 100 over the alphabet $\{a, b, c, d, e, f\}$ with frequency of each character. Two different codes are as follows:

Character	a	b	c	d	e	f
Frequenct	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Huffman Codes: Example

Example

Suppose that there is a text of length 100 over the alphabet $\{a, b, c, d, e, f\}$ with frequency of each character. Two different codes are as follows:

Character	a	b	c	d	e	f
Frequenct	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

The total required bits to store this text is as follows:

- 300 bits for Fixed-length codeword
- 224 bits for Variable-length codeword

Huffman Codes: Example

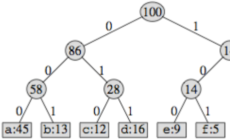
Example

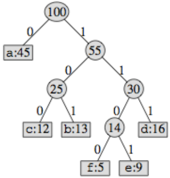
Suppose that there is a text of length 100 over the alphabet $\{a, b, c, d, e, f\}$ with frequency of each character. Two different codes are as follows:

Character	a	b	c	d	e	f
Frequenct	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

The total required bits to store this text is as follows:

- 300 bits for Fixed-length codeword
- 224 bits for Variable-length codeword





Huffman Codes

Definition (Optimal Code Tree)

Suppose that a text is given over the alphabet C with the frequency function $f : C \mapsto \mathbb{N}^+$. The optimal code tree is a binary tree T, where the characters in C are assigned to leaves of T and minimize the following term:

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

(called optimal code tree)

Huffman Codes

Definition (Optimal Code Tree)

Suppose that a text is given over the alphabet C with the frequency function $f : C \mapsto \mathbb{N}^+$. The optimal code tree is a binary tree T, where the characters in C are assigned to leaves of T and minimize the following term:

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

(called optimal code tree)

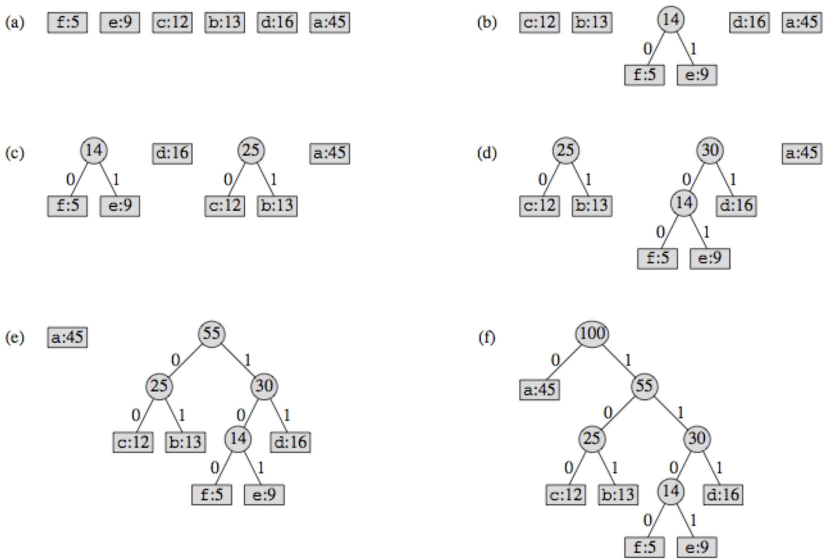
In 1952, David Huffman developed a greedy algorithm to produce such an optimal code:

Huffman: Merge the two least frequent letters and recurse.

Huffman Codes: Greedy Algorithm

```
HUFFMAN(C)
n ← |C|
Q ← C
for i ← 1 to n − 1
    do allocate a new node z
       left[z] ← x ← EXTRACT-MIN(Q)
       right[z] ← y ← EXTRACT-MIN(Q)
       f[z] ← f[x] + f[y]
       INSERT(Q, z)
return EXTRACT-MIN(Q)          ▷ Return the root of the tree.
```

Huffman Codes: Construction



Huffman Code: Correctness

Lemma

Let x and y be the two least frequent characters. There is an optimal code tree in which x and y are siblings and have the largest depth of any leaf.

Huffman Code: Correctness

Lemma

Let x and y be the two least frequent characters. There is an optimal code tree in which x and y are siblings and have the largest depth of any leaf.

Proof.

- Let T be an optimal code tree with depth d.

Huffman Code: Correctness

Lemma

Let x and y be the two least frequent characters. There is an optimal code tree in which x and y are siblings and have the largest depth of any leaf.

Proof.

- Let T be an optimal code tree with depth d.
- Since T is a full binary tree, it has at least two leaves a and b at depth d that are siblings ($\{a,b\} \neq \{x,y\}$).

Huffman Code: Correctness

Lemma

Let x and y be the two least frequent characters. There is an optimal code tree in which x and y are siblings and have the largest depth of any leaf.

Proof.

- Let T be an optimal code tree with depth d.
- Since T is a full binary tree, it has at least two leaves a and b at depth d that are siblings ($\{a, b\} \neq \{x, y\}$).
- Let T' be the code tree obtained by swapping x and a.

Huffman Code: Correctness

Lemma

Let x and y be the two least frequent characters. There is an optimal code tree in which x and y are siblings and have the largest depth of any leaf.

Proof.

- Let T be an optimal code tree with depth d.
- Since T is a full binary tree, it has at least two leaves a and b at depth d that are siblings ($\{a, b\} \neq \{x, y\}$).
- Let T' be the code tree obtained by swapping x and a.
- The depth of x (a) increases (decreases) by some amount α, thus $B(T') = B(T) - \alpha[f(a) - f(x)]$.

Huffman Code: Correctness

Lemma

Let x and y be the two least frequent characters. There is an optimal code tree in which x and y are siblings and have the largest depth of any leaf.

Proof.

- Let T be an optimal code tree with depth d.
- Since T is a full binary tree, it has at least two leaves a and b at depth d that are siblings ($\{a, b\} \neq \{x, y\}$).
- Let T' be the code tree obtained by swapping x and a.
- The depth of x (a) increases (decreases) by some amount α, thus $B(T') = B(T) - \alpha[f(a) - f(x)]$.
- By assumption, $f(a) \geq f(x)$ which implies that $B(T') \leq B(T)$.

Huffman Code: Correctness

Lemma

Let x and y be the two least frequent characters. There is an optimal code tree in which x and y are siblings and have the largest depth of any leaf.

Proof.

- Let T be an optimal code tree with depth d.
- Since T is a full binary tree, it has at least two leaves a and b at depth d that are siblings ($\{a, b\} \neq \{x, y\}$).
- Let T' be the code tree obtained by swapping x and a.
- The depth of x (a) increases (decreases) by some amount α, thus $B(T') = B(T) - \alpha[f(a) - f(x)]$.
- By assumption, $f(a) \geq f(x)$ which implies that $B(T') \leq B(T)$.
- Since T is optimal, therefore $B(T') = B(T)$ and T' is also optimal.

Huffman Code: Correctness

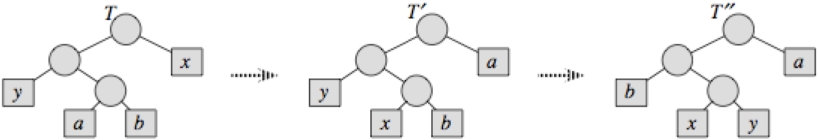
Lemma

Let x and y be the two least frequent characters. There is an optimal code tree in which x and y are siblings and have the largest depth of any leaf.

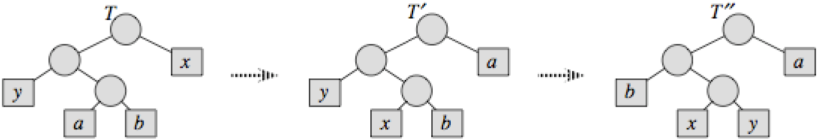
Proof.

- Let T be an optimal code tree with depth d.
- Since T is a full binary tree, it has at least two leaves a and b at depth d that are siblings ($\{a, b\} \neq \{x, y\}$).
- Let T' be the code tree obtained by swapping x and a.
- The depth of x (a) increases (decreases) by some amount α, thus $B(T') = B(T) - \alpha[f(a) - f(x)]$.
- By assumption, $f(a) \geq f(x)$ which implies that $B(T') \leq B(T)$.
- Since T is optimal, therefore $B(T') = B(T)$ and T' is also optimal.
- Swapping y and b yields another optimal code tree T'', where x and y becomes sibling and have the largest depth. $\square$

Huffman Code: Correctness



Huffman Code: Correctness



Lemma

Let x and y be two characters in C with minimum frequency. Let C' be the alphabet C with characters x, y removed and (new) character z added, so that $C' = C - \{x, y\} \cup \{z\}$. Define f for C' as for C,except that $f(z) = f(x) + f(y)$. Let T' be any tree representing an optimal prefix-free code for the alphabet C'. Then the tree T, obtained from T' by replacing the leaf node for z with an internal node having x and y as children, represents an optimal prefix-free code for the alphabet C.

Huffman Code: Correctness

Proof.

- For each $c \in C - \{x, y\}$, we have $d_T(c) = d_{T'}(c)$.

Huffman Code: Correctness

Proof.

- For each $c \in C - \{x, y\}$, we have $d_T(c) = d_{T'}(c)$.
- $d_T(x) = d_T(y) = d_{T'}(z) + 1$. So we have:

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + f(x) + f(y) \end{aligned}$$

Huffman Code: Correctness

Proof.

- For each $c \in C - \{x, y\}$, we have $d_T(c) = d_{T'}(c)$.
- $d_T(x) = d_T(y) = d_{T'}(z) + 1$. So we have:

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + f(x) + f(y) \end{aligned}$$

- This implies that $B(T) = B(T') + f(x) + f(y)$, or equivalently $B(T') = B(T) - f(x) - f(y)$.

Huffman Code: Correctness

Proof.

- For each $c \in C - \{x, y\}$, we have $d_T(c) = d_{T'}(c)$.
- $d_T(x) = d_T(y) = d_{T'}(z) + 1$. So we have:

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + f(x) + f(y) \end{aligned}$$

- This implies that $B(T) = B(T') + f(x) + f(y)$, or equivalently $B(T') = B(T) - f(x) - f(y)$.
- Now, suppose that T does not represent an optimal prefix-free code for C (Contradiction). Then there exists a tree T'' such that $B(T'') < B(T)$, where x and y are siblings in T''.

Huffman Code: Correctness

Proof.

- For each $c \in C - \{x, y\}$, we have $d_T(c) = d_{T'}(c)$.
- $d_T(x) = d_T(y) = d_{T'}(z) + 1$. So we have:

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + f(x) + f(y) \end{aligned}$$

- This implies that $B(T) = B(T') + f(x) + f(y)$, or equivalently $B(T') = B(T) - f(x) - f(y)$.
- Now, suppose that T does not represent an optimal prefix-free code for C (Contradiction). Then there exists a tree T'' such that $B(T'') < B(T)$, where x and y are siblings in T''.
- Let T''' be the tree T'' with the common parent of x and y replaced by a leaf z with frequency $f(z) = f(x) + f(y)$. Therefore we have:

$$B(T''') = B(T'') - f(x) - f(y) < B(T) - f(x) - f(y) = B(T')$$

Huffman Code: Correctness

Proof.

- For each $c \in C - \{x, y\}$, we have $d_T(c) = d_{T'}(c)$.
- $d_T(x) = d_T(y) = d_{T'}(z) + 1$. So we have:

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &= f(z)d_{T'}(z) + f(x) + f(y) \end{aligned}$$

- This implies that $B(T) = B(T') + f(x) + f(y)$, or equivalently $B(T') = B(T) - f(x) - f(y)$.
- Now, suppose that T does not represent an optimal prefix-free code for C (Contradiction). Then there exists a tree T'' such that $B(T'') < B(T)$, where x and y are siblings in T''.
- Let T''' be the tree T'' with the common parent of x and y replaced by a leaf z with frequency $f(z) = f(x) + f(y)$. Therefore we have:

$$B(T''') = B(T'') - f(x) - f(y) < B(T) - f(x) - f(y) = B(T')$$

- This yields a contradiction to the assumption that T' represents an optimal prefix-free code for C'. Thus, T must be an optimal prefix-free code for C. $\square$

Excercises

1. Implement the Huffman's algorithm for compressing and decompressing a file.
2. Generalize Huffman's algorithm to ternary codewords (i.e., codewords using the symbols 0, 1, and 2), and prove that it yields optimal prefix-free ternary codes.
3. Prove that a binary tree that is not full cannot correspond to an optimal prefix code.
4. What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers?

Character	a	b	c	d	e	f	g	h
Frequenct	1	1	2	3	5	8	13	21

   Can you generalize your answer to find the optimal code when the frequencies are the first n Fibonacci numbers?
5. Suppose we have an optimal prefix code on a set $C = \{0, 1, \cdots, n-1\}$ of characters and we wish to transmit this code using as few bits as possible. Show how to represent any optimal prefix code on C using only $2n - 1 + n\lceil \log n \rceil$ bits.