Lecture 10 — Greedy Algorithms (Part 2)

COMPSCI 308 — Design and Analysis of Algorithms

Xing Shi Cai https://newptcai.gitlab.io

Duke Kunshan University, Kunshan, China

Jan-Mar 2026

Summary

ITA 15.3 Huffman codes

Correctness of Huffman's Algorithm

ASSIGNMENTS[1]

Practice makes perfect!

Introduction to Algorithms (ITA) 📖 Required Readings:

- Section 15.3.

✏️ Required Exercises:

- Exercises 15.3: 1–8.

[1] 💣 Assignments will not be collected; however, quiz problems will be selected from them. (This includes both Readings and Exercises.)

ITA 15.3 Huffman codes

It is extremely difficult to built 🚀 to take humans to traverse the interstellar space.

In the SF trilogy, *Three Bodies*, earth sent a 🚀 with a human 🧠 to intercept 👽 invaders.

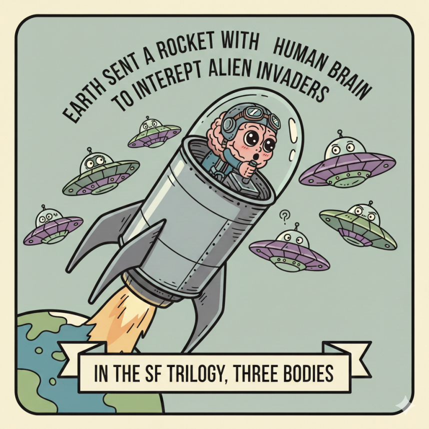


Figure 1: A 🚀 sending a 🧠 to 👽

It is extremely difficult to built 🚀 to take humans to traverse the interstellar space.

In the SF trilogy, *Three Bodies*, earth sent a 🚀 with a human 🧠 to intercept 👽 invaders.

Nowadays they could send digital copies of a human, such as ChatGPT instead instead.

🤔 How can we compress the data so we can send as many digital humans as possible?

Figure 1: Digital humans travelling through space

Consider the text:

a deaf dad bade a bee feed a babe

Encoding Digital Data

Consider the text:

a deaf dad bade a bee feed a babe

To represent the text as a binary string, we can use the following code:

Letters	a	b	d	e	f
Fixed-Len. Codewords	000	001	011	100	101

We call the binary string representation of a letter its codeword.

🍰 How can we encode the word *bad* with the above codeword?

Encoding Digital Data

Consider the text:

a deaf dad bade a bee feed a babe

To represent the text as a binary string, we can use the following code:

Letters	a	b	d	e	f
Fixed-Len. Codewords	000	001	011	100	101

The example text (with white space removed) is encoded as:

000 011 100 000 101 000 011 011
001 000 011 100 000 001 100 100
101 100 100 011 000 001 000 001 100

The result is a binary string of length 75.

Variable-length Codeword

We also can take the frequency of each letter into account —

Character	a	e	d	b	f
Frequency	0.28	0.28	0.2	0.16	0.08
Counts	7	7	5	4	2
Variable-Len. Codewords	10	11	00	011	010

🍰 How can we encode the word *bad* with the above codewords?

Variable-length Codeword

We also can take the frequency of each letter into account —

Character	a	e	d	b	f
Frequency	0.28	0.28	0.2	0.16	0.08
Counts	7	7	5	4	2
Variable-Len. Codewords	10	11	00	011	010

Encoding the text with variable-length codewords, we get:

10	00	11	10	010	10	00	11	10
10	011	10	00	11	10	011	10	00
11	10	011	10	00	11	010	11	10
00	11	10	011	10	00	11	010	11
10	00	11	10	010	10			

The result is a binary string of length 56 < 75.

The Cost of a Code

Character	a	e	d	b	f
Frequency	0.28	0.28	0.2	0.16	0.08
Counts	7	7	5	4	2
Variable-Len. Codewords	10	11	00	011	010

With this code, the average length of a code word in the encoded text is

$$0.28 \cdot 2 + 0.28 \cdot 2 + 0.2 \cdot 2 + 0.16 \cdot 3 + 0.08 \cdot 3 = 2.24$$

We call this the cost of the code.

🎯 Our aim is find the *optimal* code, i.e., the one with the minimum cost.

Caution!

Consider the following codewords:

Character	a	e	d	c	b	f
Codewords	10	11	01	001	011	010

🍰 What does the following binary string represent?

0111001

Prefix Code

To avoid ambiguity, we can will only consider prefix codes in which no codeword is prefix of another.

💬 Which ones of the following two are prefix codes?

Character	a	e	d	c	b	f
Codewords	10	11	00	001	011	010

Character	a	e	d	c	b	f
Codewords	10	11	01	001	011	010

🌲 Binary Tree

A binary tree is tree in which each node has at most two children.

A node without any children is called a leaf.

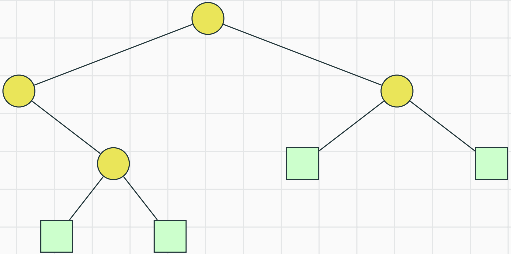


Figure 2: A Binary Tree

Binary Tree Representation

We can represent the following code

Character	a	e	d	b	f
Frequency	0.28	0.28	0.2	0.16	0.08
Counts	7	7	5	4	2
Variable-Len. Codewords	10	11	00	011	010

with a binary tree, with each leaf representing a letter.

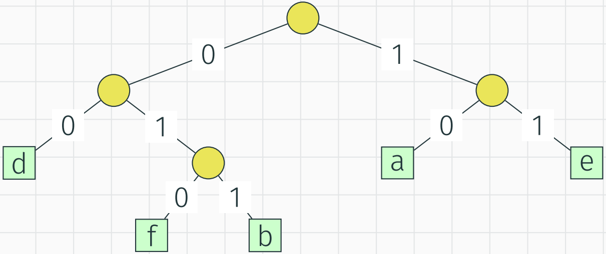


Figure 3: A Binary Tree Representation of the above codewords

🌲 Binary Tree Representation of Prefix Codes

🍰 Do you see why a code with a binary tree representation is a prefix code?

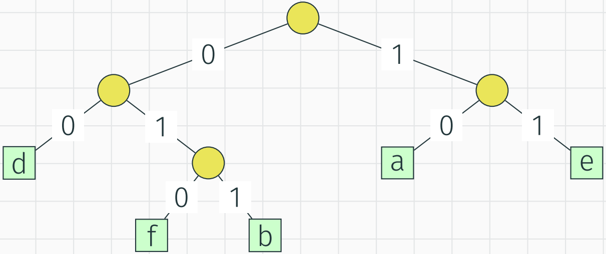


Figure 4: 📌 Example of a full binary tree T representation of a code

🍰 Do you see why a code with a binary tree representation is a prefix code?

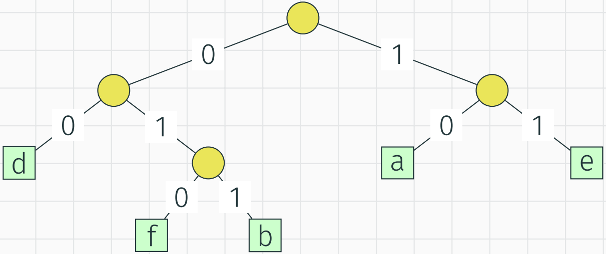


Figure 4: 📌 Example of a full binary tree T representation of a code

For a letter ℓ, let $d_T(\ell)$ be the depth of the 🍃 representing the c.

🍰 What are the $d_T(\cdot)$ in the above tree?

The Cost Revised

Let C be the set of letters in a text.

For $c \in C$, let $c.freq$ be its frequency.

The cost of a code can be written as

$$\sum_{c \in C} d_T(c) \cdot c.freq,$$

The Cost Revised

The cost of a code can be written as

$$\sum_{c \in C} d_T(c) \cdot c.freq,$$

🍰 What is the cost for the following tree/code?

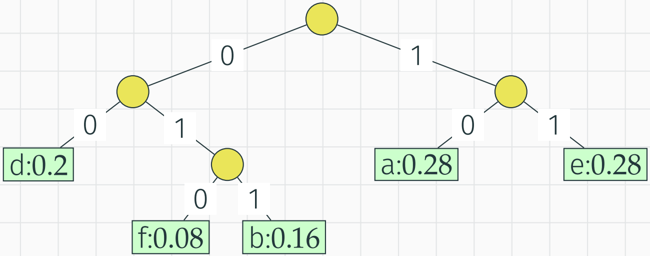


Figure 5: 📌 Example of a tree representing a code

🌲 Full Binary Tree

A full binary tree is a tree in which each node has either two or zero children.

🍰 Why does a code being optimal implies that its binary tree representation is full?

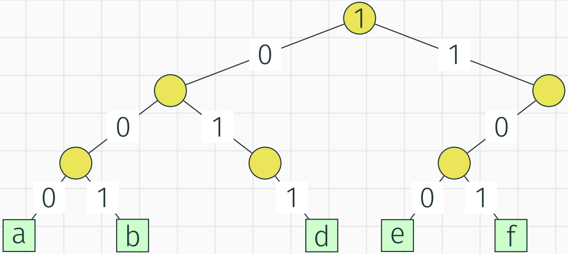


Figure 6: Not a full binary tree, not an optimal code

A full binary tree with n leaves has $n-1$ internal nodes.

Try to prove this by induction.

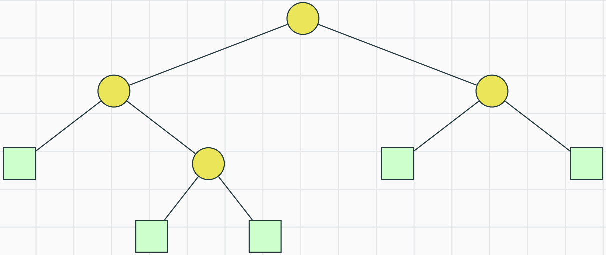


Figure 7: A full binary tree with 5 leaves and 4 internal nodes

Thus the binary tree for an optimal code for n letters contains $n-1$ internal nodes.

Huffman invented an algorithm to create an optimal code —

1. Represent each letter with a leaf node.

Huffman invented an algorithm to create an optimal code —

1. Represent each letter with a leaf node.
2. Select two nodes with the least frequencies and connect them with a parent node whose frequency is the sum of the two.

This step creates an internal node.

Huffman invented an algorithm to create an optimal code —

1. Represent each letter with a leaf node.
2. Select two nodes with the least frequencies and connect them with a parent node whose frequency is the sum of the two.
3. replace the two selected nodes with the parent node.

Huffman invented an algorithm to create an optimal code —

1. Represent each letter with a leaf node.
2. Select two nodes with the least frequencies and connect them with a parent node whose frequency is the sum of the two.
3. replace the two selected nodes with the parent node.
4. Repeat this until one node remains.

💡 The idea is that nodes with lower frequencies are merged earlier so that they are further away from the root.

🍰 How many times do we need to run step 2 if there are n letters?

🐱 Example of Huffman Code

The following code is the Huffman code for the given frequency table.

Character	a	e	d	b	f
Frequency	0.28	0.28	0.2	0.16	0.08
Counts	7	7	5	4	2
Variable-Len. Codewords	10	11	00	011	010

💬 How can construct the Huffman code from the frequency table?

💬 Try It!

The frequency table for "a cat can act" is the following —

Character	a	c	t	n
Frequency	0.40	0.30	0.20	0.10

🍰 What is the cost of the optimal code?

💡 First find the Huffman code, then compute its cost.

To implement Huffman's algorithm *efficiently*, we need a data structure called min-priority queue.

Let Q denote such a queue. Then

- Extract-Min(Q) returns an object in the queue with the minimal $freq$.
- Insert(Q, z) stores the object of z in the queue.

We assume the ⏱ complexity of each operation is $O(\log n)$.

💡 We will treat the min-priority queue as a black box.

Pseudocode of Huffman's Algorithm

Huffman(C)
$n \leftarrow |C|$
$Q \leftarrow C$
for $i \leftarrow 1$ **to** $n - 1$ **do**
    $z \leftarrow$ **new** Node
    z.left $\leftarrow x \leftarrow$ Extract-Min(Q)
    z.right $\leftarrow y \leftarrow$ Extract-Min(Q)
    z.freq $\leftarrow x$.freq $+ y$.freq
    Insert(Q, z)
return Extract-Min(Q) ▷ Return the root of the tree

🍰 What is the ⏱ complexity?

ITA 15.3 Huffman codes

Correctness of Huffman's Algorithm

For the least frequent characters $x, y \in C$, there exists an optimal prefix code where codewords for x and y have equal length and differ only in the last bit.

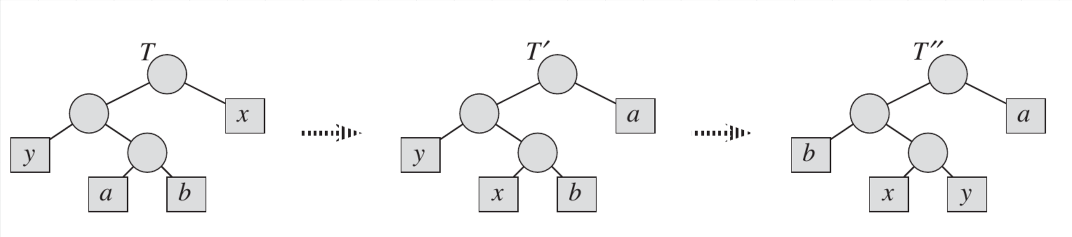


Figure 8: A proof of Lemma 15.2

💡 This lemma means Optimal prefix-free codes have the greedy-choice property.

Lemma 15.3

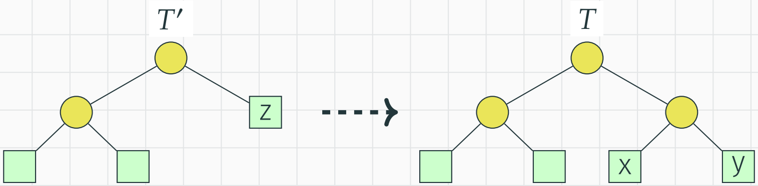
Let $x, y \in C$ be the characters with minimum frequencies.

Let $C' = C - \{x, y\} \cup \{z\}$ where $z.freq = x.freq + y.freq$.

Lemma 15.3

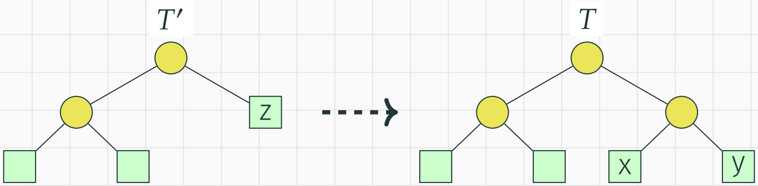
Let $x, y \in C$ be the characters with minimum frequencies.

Let $C' = C - \{x, y\} \cup \{z\}$ where $z.freq = x.freq + y.freq$.



Let $x, y \in C$ be the characters with minimum frequencies.

Let $C' = C - \{x, y\} \cup \{z\}$ where $z.freq = x.freq + y.freq$.



For a tree T' representing an optimal prefix code for C', the tree T formed by replacing T''s leaf node for z with an internal node having x and y as children, yields an optimal prefix code for C.

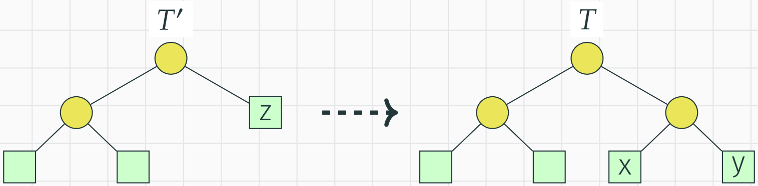
Proof of Lemma 15.3

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

$$x.freq \cdot d_T(x) + y.freq \cdot d_T(y) = z.freq \cdot d_{T'}(z) + x.freq + y.freq.$$

Thus

$$B(T) = B(T') + x.freq + y.freq.$$



Proof of Lemma 15.3

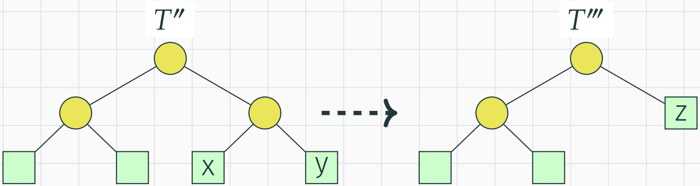
Thus

$$B(T) = B(T') + x.freq + y.freq.$$

Assume T is not optimal for C. Then there is an optimal T'' with $B(T'') < B(T)$. By Lemma 15.2, take x and y as siblings in T'' and form T''' by replacing their parent with a leaf z where $z.freq = x.freq + y.freq$. Then

$$B(T''') = B(T'') - x.freq - y.freq < B(T'),$$

a contradiction. Therefore T is optimal for C.



Theorem 15.4

Huffman's algorithm produces an optimal prefix code.

Proof.

Base Case ($n = 2$).

When the there are only two letter, the only two codes is either

- to assign one letter ______ and the other ______,
- or to swap the assignment.

Both codes thus has the same cost.

So Huffman algorithm produces an optimal prefix code. □

Theorem 15.4

Huffman's algorithm produces an optimal prefix code.

Proof.

Inductive Hypothesis — Assume that Huffman's algorithm produces an optimal prefix code for any $n-1$ letters.

□

Theorem 15.4

Huffman's algorithm produces an optimal prefix code.

Proof.

Inductive Hypothesis — Assume that Huffman's algorithm produces an optimal prefix code for any $n-1$ letters.
Inductive Step —

- **Identify the Two Least Frequent Characters**
    - Let x and y be the characters with the smallest frequencies in C.

Theorem 15.4

Huffman's algorithm produces an optimal prefix code.

Proof.

Inductive Hypothesis — Assume that Huffman's algorithm produces an optimal prefix code for any $n-1$ letters.

Inductive Step —

- **Identify the Two Least Frequent Characters**
  - Let x and y be the characters with the smallest frequencies in C.
- **Construct a Reduced Set C'**
  - Form $C' = C - \{x, y\} \cup \{z\}$, where z.freq $= x$.freq $+ y$.freq.

Theorem 15.4

Huffman's algorithm produces an optimal prefix code.

Proof.

Inductive Hypothesis — Assume that Huffman's algorithm produces an optimal prefix code for any $n-1$ letters.
Inductive Step —

- **Identify the Two Least Frequent Characters**
  - Let x and y be the characters with the smallest frequencies in C.
- **Construct a Reduced Set C'**
  - Form $C' = C - \{x, y\} \cup \{z\}$, where z.freq $=$ x.freq $+$ y.freq.
- **Apply Inductive Hypothesis to C'**
  - By induction, Huffman's algorithm produces an optimal prefix code T' for C'.

Theorem 15.4

Huffman's algorithm produces an optimal prefix code.

Proof.

Inductive Hypothesis — Assume that Huffman's algorithm produces an optimal prefix code for any $n-1$ letters.

Inductive Step —

- **Identify the Two Least Frequent Characters**
  - Let x and y be the characters with the smallest frequencies in C.
- **Construct a Reduced Set C'**
  - Form $C' = C - \{x, y\} \cup \{z\}$, where z.freq $=$ x.freq $+$ y.freq.
- **Apply Inductive Hypothesis to C'**
  - By induction, Huffman's algorithm produces an optimal prefix code T' for C'.
- **Expand T' to Obtain T for C**
  - Replace the leaf node for z in T' with an internal node having x and y as children (per Lemma 15.3).

Theorem 15.4

Huffman's algorithm produces an optimal prefix code.

Proof.

Inductive Hypothesis — Assume that Huffman's algorithm produces an optimal prefix code for any $n-1$ letters.

Inductive Step —

- **Identify the Two Least Frequent Characters**
  - Let x and y be the characters with the smallest frequencies in C.
- **Construct a Reduced Set C'**
  - Form $C' = C - \{x, y\} \cup \{z\}$, where z.freq $= x$.freq $+ y$.freq.
- **Apply Inductive Hypothesis to C'**
  - By induction, Huffman's algorithm produces an optimal prefix code T' for C'.
- **Expand T' to Obtain T for C**
  - Replace the leaf node for z in T' with an internal node having x and y as children (per Lemma 15.3).
- **Conclude Optimality** T is also exactly what Huffman's algorithm produces.

Huffman Codes with Fibonacci Frequencies

Symbol	Huffman Code	Frequency
a	1	1
b	0	1

🍰 Do you see the pattern?

Huffman Codes with Fibonacci Frequencies

Symbol	Huffman Code	Frequency
a	11	1
b	10	1
c	0	2

🍰 Do you see the pattern?

Huffman Codes with Fibonacci Frequencies

Symbol	Huffman Code	Frequency
a	111	1
b	110	1
c	10	2
d	0	3

🍰 Do you see the pattern?

Huffman Codes with Fibonacci Frequencies

Symbol	Huffman Code	Frequency
a	1111	1
b	1110	1
c	110	2
d	10	3
e	0	5

🍰 Do you see the pattern?

Huffman Codes with Fibonacci Frequencies

Symbol	Huffman Code	Frequency
a	11111	1
b	11110	1
c	1110	2
d	110	3
e	10	5
f	0	8

🍰 Do you see the pattern?

Huffman Codes with Fibonacci Frequencies

Symbol	Huffman Code	Frequency
a	111111	1
b	111110	1
c	11110	2
d	1110	3
e	110	5
f	10	8
g	0	13

🍰 Do you see the pattern?

🤪 Beyond Huffman Coding: Notable New Directions in Compression (2025–2026)

Figure 9: Silicon Valley is an American comedy television series about company trying to find new compression algorithms

🤪 Beyond Huffman Coding: Notable New Directions in Compression (2025–2026)

Learned compression. Neural codecs learn transforms and models. Entropy coding still finishes the job.

🤪 Beyond Huffman Coding: Notable New Directions in Compression (2025–2026)

Learned compression. Neural codecs learn transforms and models. Entropy coding still finishes the job.

Generative compression. The decoder guesses fine details. Smaller files, less exact images.

🤪 Beyond Huffman Coding: Notable New Directions in Compression (2025–2026)

Learned compression. Neural codecs learn transforms and models. Entropy coding still finishes the job.

Generative compression. The decoder guesses fine details. Smaller files, less exact images.

Compression for machines (VCM). Bitstreams keep features for AI tasks. Human visuals may look worse.

🤪 Beyond Huffman Coding: Notable New Directions in Compression (2025–2026)

Learned compression. Neural codecs learn transforms and models. Entropy coding still finishes the job.

Generative compression. The decoder guesses fine details. Smaller files, less exact images.

Compression for machines (VCM). Bitstreams keep features for AI tasks. Human visuals may look worse.

AV2 video standard. AV2 follows AV1. It targets smaller files.

🤪 Beyond Huffman Coding: Notable New Directions in Compression (2025–2026)

Learned compression. Neural codecs learn transforms and models. Entropy coding still finishes the job.

Generative compression. The decoder guesses fine details. Smaller files, less exact images.

Compression for machines (VCM). Bitstreams keep features for AI tasks. Human visuals may look worse.

AV2 video standard. AV2 follows AV1. It targets smaller files.

JPEG XL renaissance. JPEG XL compresses old JPEGs. It may return in browsers.

What are your main takeaways today? Any questions?