SNS COLLEGE OF TECHNOLOGY

An Autonomous Institution
Coimbatore-35

DEPARTMENT OF INFORMATION TECHNOLOGY

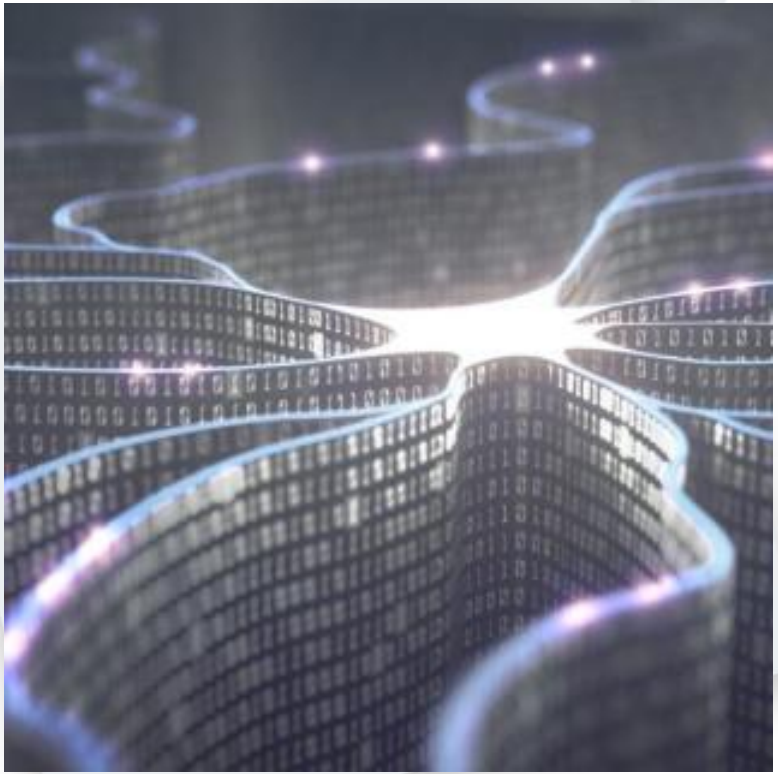
23ITT304 Information Coding Technique

Unit II

Source coding

Huffman Coding

P.Thilagarani AP/IT



Huffman Coding

Efficient Data Compression Through Smart Encoding

Recap & Icebreaker

Let's Refresh

- What is data compression? Why do we need it?
- Have you ever seen a '.zip' or '.mp3' file? What happens inside?

Quick Thought Experiment

Imagine sending a message using only '' and '1'.

If 'A' appears 50 times and 'Z' appears twice, should both use the same number of bits?

Think, pair, share in 60 seconds.

Session Objectives

By the End of This Session, You Will:

- Understand how Huffman coding reduces file size intelligently.
- Build a Huffman tree from frequency data.
- Encode and decode messages using variable-length codes.
- Apply design thinking to solve compression challenges.
- Connect Huffman coding to real-world tech like ZIP, MP3, and JPEG.

Empathize: Meet the User

Priya, a Bangalore app developer, builds a health app for rural areas with slow internet, needing small, fast downloads without losing key info.

Her users care about access, speed, and reliability — not file formats. How might we help Priya compress data without losing meaning?

Define: The Core Problem

Fixed vs. Smart Encoding

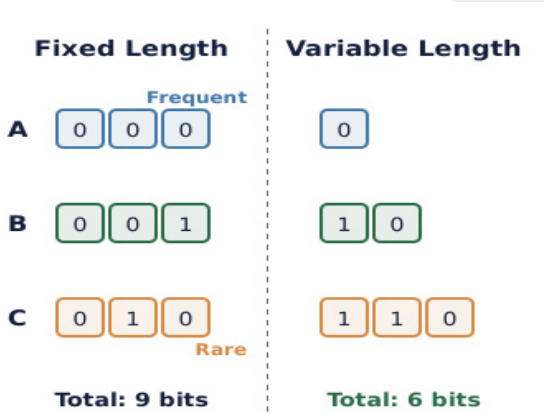
Standard ASCII uses 8 bits per character — even for rare letters.

This wastes bandwidth when some characters (like 'E') appear far more than others (like 'Q').

The Challenge

How can we assign shorter codes to frequent symbols and longer ones to rare ones — without ambiguity?

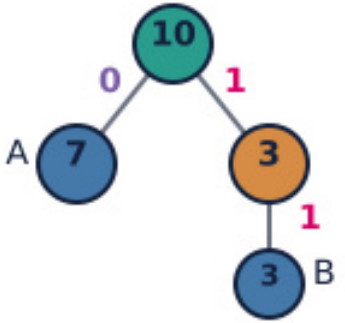
Goal: Minimize total bits, ensure unique decodability.



Ideate: Possible Approaches

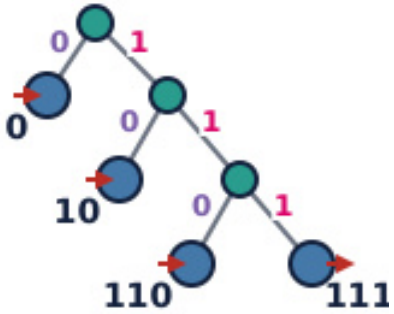
Frequency Count

Count each symbol's frequency in the message.



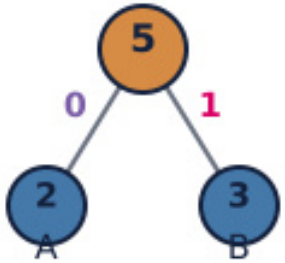
Tree Structure

Build a binary tree with frequent symbols near the root.



Prefix Codes

No code should prefix another to avoid decoding issues.



Greedy Merge

Build tree bottom-up by combining least frequent nodes.

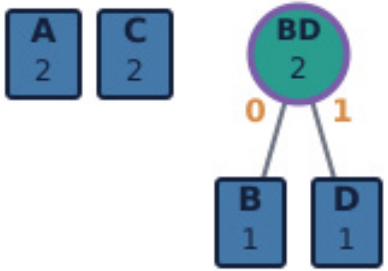
Prototype: Build a Huffman Tree

Step 1: Count

Given message: 'AABCCD'. Frequencies: A=2, B=1, C=2, D=1.

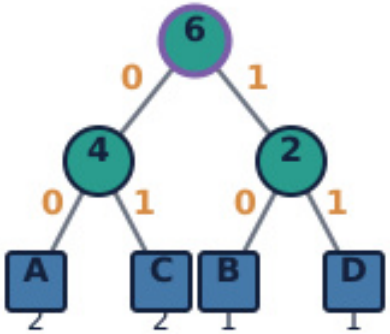
Step 2: Merge

Combine lowest: B(1) & D(1) → node BD(2). Now nodes: A(2), C(2), BD(2).



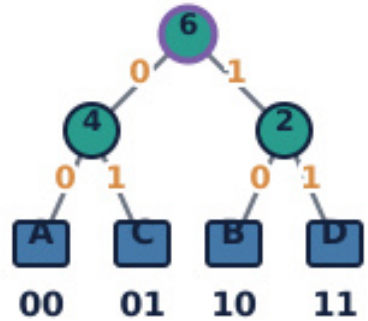
Step 3: Tree

Merge A(2) & C(2) → AC(4). Finally merge AC(4) & BD(2) → root(6).



Step 4: Assign Codes

Traverse: left=0, right=1 → A=0, C=01, B=10, D=11.



Get creative

Group Challenge: Build Your Huffman Tree

Divide into groups of 3. Each team gets a message like 'SCHOOL' and its **frequency table**. On chart paper, build a **Huffman tree**, assign **binary codes**, and calculate how many **bits are saved** compared to fixed 8-bit encoding.

What you'll need:

Chart Paper, Markers, Frequency Cards, Ruler

T-Shaped Learning: Depth & Breadth

Vertical Depth

- Huffman algorithm is greedy and optimal for symbol-by-symbol coding.
- Time complexity: O(n log n) using priority queue.
- Prefix codes prevent ambiguity in decoding.
- Shannon entropy sets the theoretical limit — Huffman approaches it.

Horizontal Breadth

- Used in ZIP, PNG, JPEG, and MP3 file formats.
- Combined with LZ77 in DEFLATE (used in gzip).
- Foundation for modern codecs and streaming tech.
- Inspires AI compression in neural networks (e.g., quantization).

Summary & Mind Map

Key Takeaways

- Huffman coding uses frequency to assign variable-length codes.
- Frequent symbols → short codes; rare → long codes.
- Built using a greedy binary tree algorithm.
- Ensures unique decodability with prefix-free codes.
- Saves space in real-world formats like ZIP and JPEG.

Design Thinking Recap: Empathize → Define → Ideate → Prototype → Test.

Discuss!

When Compression Hits Its Limit

Can **Huffman coding** compress any file? What if all characters appear equally often? How might we improve compression for such cases?

Discuss!

You might have said...

No, because if all characters are equally frequent, shorter codes offer no benefit. Fixed-length encoding may be equally or more efficient. Consider run-length encoding or context-based models to uncover hidden patterns.

Huffman Coding Checkpoint

Question 1:

What is the main idea behind Huffman coding, and how does it reduce file size?

Question 2:

Why are prefix codes essential in Huffman coding, and how do they support error-free decoding?

Question 3:

Which data structure enables efficient building of the Huffman tree, and how is it used in the algorithm?

Answers on the next slide...

Huffman Coding Checkpoint

Answer 1:

Huffman coding is a method for lossless data compression that assigns shorter binary codes to more frequent symbols and longer codes to less frequent ones, minimizing total message length using a greedy-built binary tree.

Answer 2:

Prefix codes ensure unique decodability by making sure no code is the beginning of another, preventing ambiguity when reading a bitstream sequentially.

Answer 3:

A priority queue (min-heap) is used to efficiently select and merge the two nodes with the lowest frequencies during Huffman tree construction.

Thank you