Shannon-Fano Coding

A Foundation of Data Compression

Mahesh C
FISAT

Federal Institute of Science and Technology (FISAT)
Multimedia Technology Class

January 9, 2026

Outline

XFMDPNF UP GJTBU

What does this say?

Take 30 seconds to guess...

X	F	M	D	P	N	F		U	P		G	J	T	B	U
↓	↓	↓	↓	↓	↓	↓		↓	↓		↓	↓	↓	↓	↓
?	?	?	?	?	?	?		?	?		?	?	?	?	?

Here's a Hint...

XFMDPNF UP GJTBU

Hint

Each letter has been **shifted forward by 1 position** in the alphabet.

A → B,   B → C,   C → D,   ...   Z → A

Now try again!

X	F	M	D	P	N	F		U	P		G	J	T	B	U
↓	↓	↓	↓	↓	↓	↓		↓	↓		↓	↓	↓	↓	↓
W	E	L	C	O	M	E		T	O		F	I	S	A	T

Here's a Hint...

XFMDPNF UP GJTBU

Hint

Each letter has been **shifted forward by 1 position** in the alphabet.

A → B, B → C, C → D, ... Z → A

Now try again!

X	F	M	D	P	N	F		U	P		G	J	T	B	U
↓	↓	↓	↓	↓	↓	↓		↓	↓		↓	↓	↓	↓	↓
W	E	L	C	O	M	E		T	O		F	I	S	A	T

The Answer is...

Surprise — That Was Coding!

What You Just Did

You performed **DECODING** — converting coded information back to its original form!

Encoding (Sender):

WELCOME TO FISAT
$\downarrow$ *Shift +1*
XFMDPNF UP GJTBU

Decoding (Receiver):

XFMDPNF UP GJTBU
$\downarrow$ *Shift -1*
WELCOME TO FISAT

This is called Caesar Cipher

Used by Julius Caesar 2000+ years ago to send secret military messages!
Code = Rule to transform information

Today we'll learn a different type of coding — not for secrecy, but for ***efficiency****!*

What is “Coding” in Information Theory?

Important Clarification

Coding here does NOT mean programming or writing software!

Definition

Coding = Converting information from one representation to another

You already use coding every day!

- **Language:** Thoughts $\rightarrow$ Words $\rightarrow$ Speech sounds
- **Writing:** Words $\rightarrow$ Letters on paper
- **Emojis:** Emotions $\rightarrow$ Smiley, Party, Heart symbols
- **Traffic lights:** Instructions $\rightarrow$ Red/Yellow/Green
- **Music:** Sound $\rightarrow$ Notes on a sheet (Do Re Mi...)

Everyday Examples of Codes

1. Morse Code (1840s)

Letter	Code
A	$\cdot -$
B	$- \cdot \cdot \cdot$
E	$\cdot$
S	$\cdot \cdot \cdot$
O	$- - -$

SOS $= \cdot \cdot \cdot - - - \cdot \cdot \cdot$

Notice: 'E' (common) is short!

2. Braille (for visually impaired)

- Each letter $=$ pattern of 6 dots
- Converts visual text to touch

3. Binary Code (Computers)

- A $=$ 01000001
- B $=$ 01000010
- Everything is 0s and 1s!

Why Do We Need Different Codes?

Different situations need different codes:

Telegram (pay per character):

- "ARRIVING TOMORROW MORNING"
- Shorter = Cheaper!
- People invented abbreviations

SMS (160 character limit):

- "c u l8r" instead of "see you later"
- Shorter codes for common phrases

PIN Codes in India:

- 6 digits represent location
- 682030 = Specific area in Kochi
- Compact way to encode address

Vehicle Registration:

- KL-07-AB-1234
- State + District + Series + Number
- Structured code for identification

The Key Question: What Makes a Good Code?

If you could design your own code, how would you do it?

Properties of a good code:

1. **Efficient:** Uses minimum symbols/bits
2. **Unambiguous:** Each message has only one meaning
3. **Decodable:** Can recover original message perfectly

The Smart Idea

Frequently used items → Short codes
Rarely used items → Long codes

This is exactly what **Shannon-Fano Coding** does!

From Codes to Computers: Binary World

Computers only understand 0 and 1 (Binary)

Why binary?

- Electronic switches: ON (1) or OFF (0)
- Simple and reliable
- Easy to store and transmit

Everything becomes binary:

- Text $\rightarrow$ Binary
- Images $\rightarrow$ Binary
- Audio $\rightarrow$ Binary
- Video $\rightarrow$ Binary

Standard ASCII Code:

Character	Binary (8 bits)
A	01000001
B	01000010
a	01100001
0	00110000
Space	00100000

Every character = 8 bits (fixed)
Is this efficient? (Think about it!)

The Problem with Fixed-Length Codes

ASCII uses 8 bits for EVERY character:

"HELLO" in ASCII:

- H = 01001000
- E = 01000101
- L = 01001100
- L = 01001100
- O = 01001111

Total: $5 \times 8 = 40$ bits

But wait...

- 'E' is the most common letter
- 'Z' is very rare
- Both use 8 bits! (Wasteful!)

Better idea:

- Give 'E' a short code (2-3 bits)
- Give 'Z' a longer code
- Average bits per letter drops!

This is Variable-Length Coding!

Shannon-Fano coding assigns **different length codes** based on **how often** each

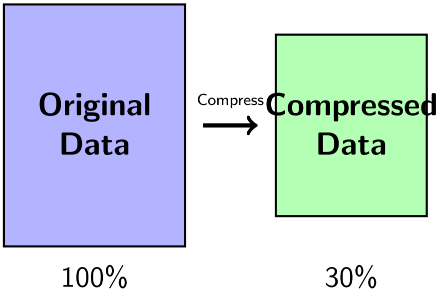
Why Data Compression?

The Need for Compression:

- Limited storage capacity
- Limited bandwidth for transmission
- Cost reduction
- Faster data transfer

Types of Compression:

- Lossless - Perfect reconstruction
- Lossy - Approximate reconstruction



Real-World Example: Why Compression Matters

Scenario: Sending a 4K Movie over the Internet

Without Compression:

- Raw 4K video: ∼500 GB for 2-hour movie
- On 50 Mbps connection: ∼22 hours to download!
- Netflix monthly data: ∼15 TB per user

With Compression (H.265):

- Compressed: ∼8-15 GB
- Download time: ∼30-45 minutes
- 97% storage saved!

Daily Life Examples

- **WhatsApp:** Compresses photos from 5MB to 100KB before sending
- **Spotify:** Streams 320kbps instead of 1411kbps (CD quality)
- **ZIP files:** Reduces document folder by 70-90%

Information Theory Basics

Claude Shannon (1948) - Father of Information Theory

Key Insight

The amount of information in a message is related to its **probability**. Rare events carry more information than common events.

Self-Information of an event with probability p:

$$I(x) = -\log_2 p(x) = \log_2 \frac{1}{p(x)} \quad \text{(bits)} \tag{1}$$

Example:

- If $p = 0.5$: $I = -\log_2(0.5) = 1$ bit
- If $p = 0.25$: $I = -\log_2(0.25) = 2$ bits
- If $p = 0.125$: $I = -\log_2(0.125) = 3$ bits

Entropy - Average Information

Shannon Entropy

The **entropy** $H(X)$ of a discrete random variable X with possible values $\{x_1, x_2, \ldots, x_n\}$ and probability mass function $P(X)$:

$$\boxed{H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) = \sum_{i=1}^{n} p(x_i) \log_2 \frac{1}{p(x_i)}} \tag{2}$$

Properties of Entropy:

- $H(X) \geq 0$ (always non-negative)
- $H(X) = 0$ if and only if one outcome has probability 1
- $H(X)$ is maximum when all outcomes are equally likely
- Maximum entropy: $H_{max} = \log_2 n$

Entropy Calculation Example

Example: A source emits symbols $\{A, B, C, D\}$ with probabilities:

Symbol	A	B	C	D
Probability	0.5	0.25	0.125	0.125

Entropy Calculation:

$$\begin{aligned} H(X) &= -[0.5\log_2(0.5) + 0.25\log_2(0.25) \\ &\quad + 0.125\log_2(0.125) + 0.125\log_2(0.125)] \\ &= -[0.5(-1) + 0.25(-2) + 0.125(-3) + 0.125(-3)] \\ &= 0.5 + 0.5 + 0.375 + 0.375 \\ &= \boxed{1.75 \text{ bits/symbol}} \end{aligned}$$

Real-World Entropy Example: English Text

Letter Frequencies in English:

Letter	Frequency
E	12.7%
T	9.1%
A	8.2%
O	7.5%
I	7.0%
N	6.7%
S	6.3%
...	...
Z	0.07%

Key Insight:

- 'E' appears 180x more than 'Z'
- Fixed 8-bit ASCII wastes bits!
- Entropy of English $\approx$ 4.2 bits/letter
- Potential savings: 48%!

Shannon-Fano Idea:

- Give 'E' a short code (2-3 bits)
- Give 'Z' a long code (8+ bits)
- Average code length drops!

Simple Analogy: Weather Reporting

Imagine you're a weather reporter in Kerala:

Weather Probabilities:

Weather	Probability
Sunny	50%
Cloudy	25%
Rainy	15%
Stormy	10%

Efficient Codes:

Weather	Code
Sunny	0 (1 bit)
Cloudy	10 (2 bits)
Rainy	110 (3 bits)
Stormy	111 (3 bits)

The Core Idea

Common events → Short codes — Rare events → Long codes

Just like in Morse code: 'E' $= \cdot$ (1 symbol), 'Q' $= - - \cdot -$ (4 symbols)

What is Shannon-Fano Coding?

Definition

Shannon-Fano coding is a **prefix-free**, **variable-length** coding technique that assigns shorter codes to more frequent symbols.

Historical Background:

- Developed independently by **Claude Shannon** and **Robert Fano** in 1948-1949
- One of the first practical entropy coding methods
- Precursor to Huffman coding

Key Properties:

- Prefix-free code - No codeword is a prefix of another
- Variable-length encoding based on symbol probability
- Instantaneously decodable
- Near-optimal but not always optimal

Prefix-Free Codes

Why Prefix-Free?

- Allows **instantaneous decoding**
- No need for separators
- Unambiguous decoding

Example - NOT Prefix-Free:

$$A \to 0 \quad B \to 01$$

Problem: Is "01" = "AB" or "B"?

Prefix-Free Example:

$$A \to 0 \qquad B \to 10$$
$$C \to 110 \quad D \to 111$$

Decode "011010":

- $0 \to A$
- $110 \to C$
- $10 \to B$

Result: **ACB**

The Shannon-Fano Algorithm

Algorithm Steps

1. **Sort** symbols in decreasing order of probability
2. **Divide** the list into two groups with approximately equal total probabilities
3. **Assign** '0' to the first group and '1' to the second group
4. **Recursively** apply steps 2-3 to each group until each group contains only one symbol

Key Principle: At each step, try to balance the probabilities on each side as equally as possible.

Algorithm Pseudocode

Algorithm 1 Shannon-Fano Coding

```
Input: List of symbols with probabilities
Output: Binary codes for each symbol
Sort symbols by probability (descending)
Call ShannonFano(symbols, code = "")

Procedure ShannonFano(symbols, code)
if length(symbols) == 1 then
    Assign code to the symbol
else
    Divide symbols into two groups (balanced probabilities)
    ShannonFano(group1, code + "0")
    ShannonFano(group2, code + "1")
end if
```

Shannon-Fano: Detailed Example

Given: Source symbols with the following probabilities:

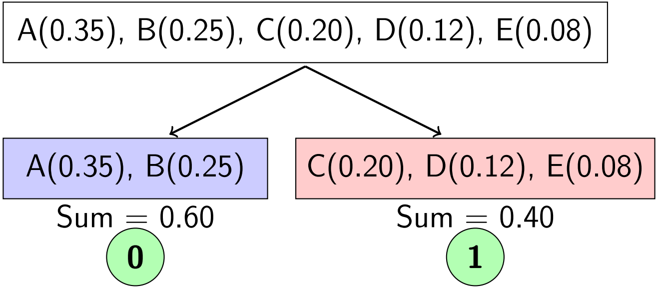
Symbol	Probability
A	0.35
B	0.25
C	0.20
D	0.12
E	0.08

Step 1: Already sorted in decreasing order of probability.

Total probability $= 1.0$ (verified)

Example: First Division

Step 2: Divide into two groups with balanced probabilities

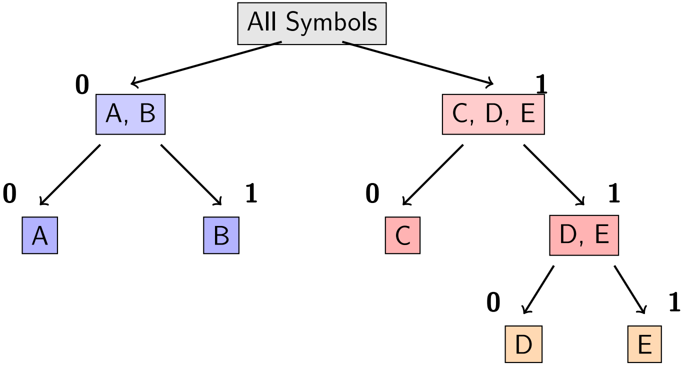


Division Options:

- {A} vs {B,C,D,E}: 0.35 vs 0.65 $\rightarrow$ Difference = 0.30
- {A,B} vs {C,D,E}: 0.60 vs 0.40 $\rightarrow$ Difference = 0.20 ✓

Example: Second Level Division

Step 3: Recursively divide each group



Example: Final Code Assignment

Reading codes from root to leaves:

Symbol	Probability	Code	Code Length
A	0.35	00	2
B	0.25	01	2
C	0.20	10	2
D	0.12	110	3
E	0.08	111	3

Verify Prefix-Free Property:

- No code is a prefix of another ✓
- Uniquely decodable ✓

Average Code Length

Average Code Length Formula

$$L_{avg} = \sum_{i=1}^{n} p_i \cdot l_i \tag{3}$$

where p_i is the probability and l_i is the code length of symbol i.

For our example:

$$\begin{aligned} L_{avg} &= 0.35 \times 2 + 0.25 \times 2 + 0.20 \times 2 + 0.12 \times 3 + 0.08 \times 3 \\ &= 0.70 + 0.50 + 0.40 + 0.36 + 0.24 \\ &= \boxed{2.20 \text{ bits/symbol}} \end{aligned}$$

Entropy Comparison

Calculate the entropy:

$$
\begin{aligned}
H(X) &= -\sum_{i=1}^{n} p_i \log_2 p_i \\
&= -(0.35 \log_2 0.35 + 0.25 \log_2 0.25 + 0.20 \log_2 0.20 \\
&\quad + 0.12 \log_2 0.12 + 0.08 \log_2 0.08) \\
&= -(0.35 \times (-1.514) + 0.25 \times (-2) + 0.20 \times (-2.322) \\
&\quad + 0.12 \times (-3.059) + 0.08 \times (-3.644)) \\
&= 0.530 + 0.500 + 0.464 + 0.367 + 0.292 \\
&= \boxed{2.153 \text{ bits/symbol}}
\end{aligned}
$$

Coding Efficiency

Efficiency Formula

$$\eta = \frac{H(X)}{L_{avg}} \times 100\% \qquad (4)$$

For our example:

$$\eta = \frac{2.153}{2.20} \times 100\% = \boxed{97.86\%}$$

Shannon's Source Coding Theorem

The average code length satisfies:

$$H(X) \leq L_{avg} < H(X) + 1 \qquad (5)$$

Verification: $2.153 \leq 2.20 < 3.153$ ✓

Redundancy

Redundancy Formula

$$R = L_{avg} - H(X) \tag{6}$$

For our example:

$$R = 2.20 - 2.153 = \boxed{0.047 \text{ bits/symbol}}$$

Interpretation:

- Lower redundancy = better compression
- Shannon-Fano achieves near-optimal performance
- Huffman coding can achieve even lower redundancy

Practice Problem

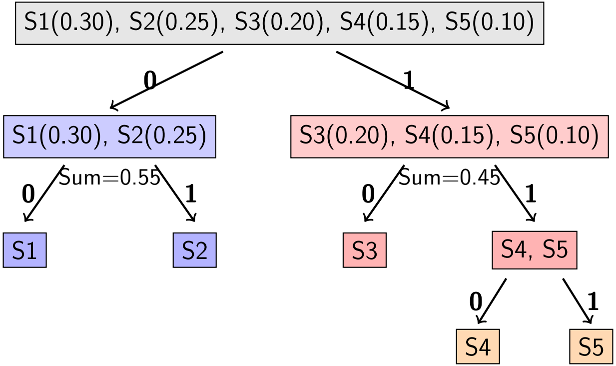
Problem: Construct Shannon-Fano codes for the following source:

Symbol	Probability
S1	0.30
S2	0.25
S3	0.20
S4	0.15
S5	0.10

Tasks:

1. Construct the Shannon-Fano code
2. Calculate average code length
3. Calculate entropy
4. Determine efficiency

Practice Problem: Solution Tree



Practice Problem: Final Codes

Symbol	Probability	Code	Length	$p_i \times l_i$
S1	0.30	00	2	0.60
S2	0.25	01	2	0.50
S3	0.20	10	2	0.40
S4	0.15	110	3	0.45
S5	0.10	111	3	0.30
			Average Code Length:	**2.25**

Entropy:

$$H(X) = -(0.30 \log_2 0.30 + 0.25 \log_2 0.25 + 0.20 \log_2 0.20 + 0.15 \log_2 0.15 + 0.10 \log_2 0.10)$$

$$H(X) \approx 2.185 \text{ bits/symbol}$$

Practice Problem: Efficiency Analysis

Results Summary:

Metric	Value
Entropy $H(X)$	2.185 bits/symbol
Average Code Length L_{avg}	2.25 bits/symbol
Efficiency η	97.11%
Redundancy R	0.065 bits/symbol

Observations

- High efficiency (¿97%) indicates good compression
- $H(X) \leq L_{avg} < H(X) + 1$ is satisfied
- Small redundancy shows near-optimal performance

Encoding Process

To encode a message using Shannon-Fano codes:

1. Construct the Shannon-Fano code table
2. Replace each symbol with its corresponding code
3. Concatenate all codes

Example: Encode "ABCDE" using our code table

Symbol	Code
A $\rightarrow$	00
B $\rightarrow$	01
C $\rightarrow$	10
D $\rightarrow$	110
E $\rightarrow$	111

Encoded message: 00 01 10 110 111 = **000110110111**

Decoding Process

Decoding with Prefix-Free Codes:

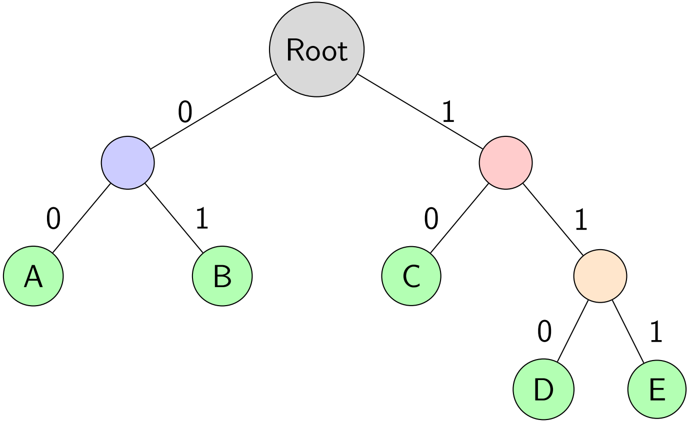
1. Read bits from left to right
2. Match against code table
3. Output symbol when match is found
4. Continue with remaining bits

Example: Decode "1001110110"

- **10** $\rightarrow$ C
- **01** $\rightarrow$ B
- **110** $\rightarrow$ D
- **110** $\rightarrow$ D

Decoded message: CBDD

Code Tree for Decoding



Decoding: Traverse tree from root; 0=left, 1=right; leaf=output symbol

Comparison with Huffman Coding

Shannon-Fano Coding:

- Top-down approach
- Divide and assign bits
- Simpler to understand
- Not always optimal
- Historical importance

Huffman Coding:

- Bottom-up approach
- Merge lowest probability nodes
- More complex construction
- **Always optimal**
- More widely used

Key Difference

Huffman coding is **guaranteed** to produce an optimal prefix-free code, while Shannon-Fano may not achieve the absolute minimum average code length.

Example Where Shannon-Fano is Suboptimal

Consider: Symbols with probabilities: 0.4, 0.3, 0.2, 0.1

Shannon-Fano:

Prob	Code	Len
0.4	0	1
0.3	10	2
0.2	110	3
0.1	111	3

$L_{avg} = 0.4(1) + 0.3(2) + 0.2(3) + 0.1(3)$
$L_{avg} = 1.9$ bits/symbol

Huffman:

Prob	Code	Len
0.4	0	1
0.3	10	2
0.2	110	3
0.1	111	3

(Same result in this case)
$L_{avg} = 1.9$ bits/symbol

Entropy: $H = 1.846$ bits/symbol

When Shannon-Fano Differs

Consider: Probabilities 0.35, 0.17, 0.17, 0.16, 0.15

Shannon-Fano might give:

Prob	SF Code	Length
0.35	00	2
0.17	01	2
0.17	10	2
0.16	110	3
0.15	111	3

$L_{avg}^{SF} = 2 \times 0.69 + 3 \times 0.31 = 2.31$ bits

Huffman achieves: $L_{avg}^{H} = 2.30$ bits (slightly better!)

Real-World Example: Image Compression

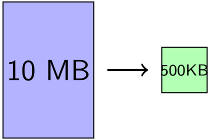
How JPEG Uses Entropy Coding:
Pixel Values in a Photo:

- Most pixels are similar to neighbors
- Small differences are common
- Large differences are rare

Difference	Frequency
0	40%
±1	25%
±2-5	20%
>5	15%

Result:

- Original photo: 10 MB
- After JPEG: 500 KB
- **95% compression!**



Real-World Example: Text Messages

SMS and Messaging Apps:

Scenario

You type "hello" frequently, "xylophone" rarely.

Without Smart Coding:

- Each character = 8 bits
- "hello" = 40 bits
- "xylophone" = 72 bits

With Entropy Coding:

- Common words get short codes
- "hello" → 8 bits (dictionary)
- Rare words = longer codes

Real Impact

WhatsApp handles 100+ billion messages/day. Even 50% compression saves **petabytes** of bandwidth daily!

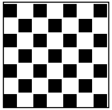
Real-World Example: QR Codes

How QR Codes Store Data Efficiently:
QR Code Modes:

- **Numeric only:** 3.3 bits/char
- **Alphanumeric:** 5.5 bits/char
- **Binary/Byte:** 8 bits/char

Why Variable Length?

- Phone numbers: mostly digits → short codes
- URLs: letters + numbers → medium codes
- Full Unicode: all characters → longer codes



Same principle as Shannon-Fano!

Real-World Applications

Shannon-Fano coding principles are used in:

File Compression:

- ZIP file format (historical)
- Early compression utilities
- Text file compression

Data Communications:

- Fax transmission
- Modem protocols
- Network data compression

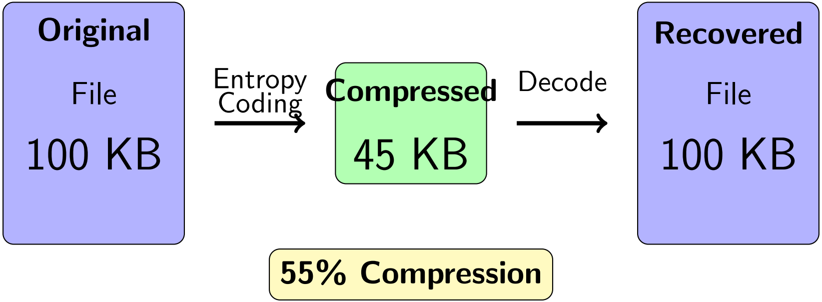
Multimedia:

- Audio codecs
- Video compression
- Image formats

Modern Variants:

- Arithmetic coding
- Range coding
- ANS (Asymmetric Numeral Systems)

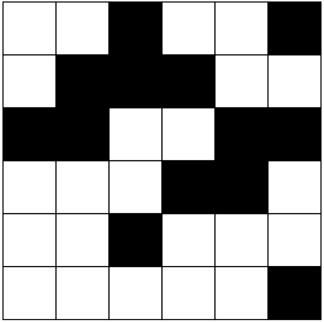
Compression in Practice



Lossless compression guarantees perfect reconstruction!

Exercise 4: Image Pattern Encoding — Question

Problem: Analyze this 6×6 pixel image and create Shannon-Fano codes.



Tasks:

1. Count Black (B) and White (W) pixels
2. Calculate probability of each pixel value

Exercise 4: Image Pattern Encoding — Approach

Step 1: Define Pixel Values

$$W = \text{White } (0) \qquad\qquad B = \text{Black } (1)$$

Step 2: Read image row by row (left to right), group in 3s for readability:

Row	Pixel Sequence
Row 1	`WWB WWB`
Row 2	`WBB BWW`
Row 3	`BBW WBB`
Row 4	`WWW BBW`
Row 5	`WWB WWW`
Row 6	`WWW WWB`

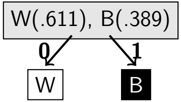
Step 3: Count each symbol

Exercise 4: Solution — Image Pattern Encoding

Pixel Analysis:

Pixel	Count	Prob
W (White)	22	0.611
B (Black)	14	0.389
Total	**36**	**1.00**

Shannon-Fano Tree:



Codes: W = 0, B = 1

Entropy:

$$H(X) = -(0.611 \log_2 0.611 + 0.389 \log_2 0.389)$$
$$= 0.434 + 0.530 = \boxed{0.964 \text{ bits/pixel}}$$

Comparison:

- Fixed 1-bit: $36 \times 1 = 36$ bits
- Shannon-Fano: $36 \times 1 = 36$ bits
- Theoretical min: $36 \times 0.964 \approx 35$ bits

Key Insight

With only 2 symbols, Shannon-Fano gives 1 bit each — no compression gain over fixed encoding!

Exercise 1: Find Symbols and Build Shannon-Fano Code

Problem: Analyze the following data sequence and construct Shannon-Fano codes.

Data Sequence (40 symbols)

`AAB AAC AAB AAA CAB AAB ACA AAB AAC ABB ABA AAB ACA A`

Tasks:

1. Identify all unique symbols in the sequence
2. Count the frequency of each symbol
3. Calculate the probability of each symbol
4. Construct the Shannon-Fano code
5. Calculate average code length and efficiency

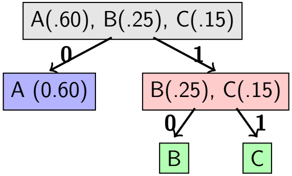
Hint: Count carefully — there are 40 symbols total!

Exercise 1: Solution

Step 1-3: Symbol Analysis

Symbol	Freq	Prob
A	24	0.60
B	10	0.25
C	6	0.15
Total	**40**	**1.00**

Shannon-Fano Tree:



Final Codes: A = 0, B = 10, C = 11

Exercise 1: Solution (Continued)

Final Codes:

Symbol	Probability	Code	Length	$p_i \times l_i$
A	0.60	0	1	0.60
B	0.25	10	2	0.50
C	0.15	11	2	0.30
Average Code Length L_{avg}:				**1.40**

Performance Analysis:

$$
\begin{aligned}
H(X) &= -(0.60 \log_2 0.60 + 0.25 \log_2 0.25 + 0.15 \log_2 0.15) \\
&= -(0.60 \times (-0.737) + 0.25 \times (-2) + 0.15 \times (-2.737)) \\
&= 0.442 + 0.500 + 0.411 = \boxed{1.353 \text{ bits/symbol}}
\end{aligned}
$$

Efficiency: $\eta = \frac{1.353}{1.40} \times 100\% = \boxed{96.64\%}$

Exercise 2: Find Symbols and Build Shannon-Fano Code

Problem: Analyze the following message sequence and construct Shannon-Fano codes.

Message Sequence (50 symbols)

`SUN MON SUN TUE SUN WED SUN MON SUN THU SUN MON FRI SUN MON SAT SUN`

Tasks:

1. Identify all unique symbols (day codes)
2. Count the frequency of each symbol
3. Calculate the probability of each symbol
4. Construct the Shannon-Fano code
5. Calculate average code length, entropy, and efficiency

Hint: This represents weekly schedule data — some days appear more often!

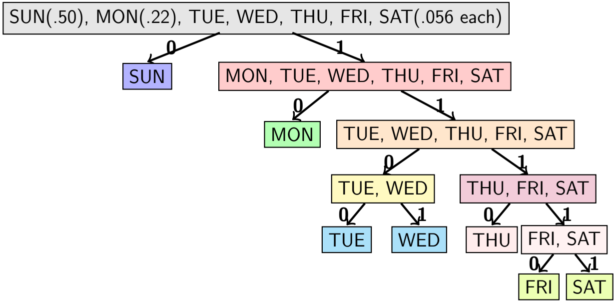
Exercise 2: Solution

Step 1-3: Symbol Analysis (Sorted by probability)

Symbol	Frequency	Probability
SUN	9	0.36
MON	4	0.16
TUE	1	0.04
WED	1	0.04
THU	1	0.04
FRI	1	0.04
SAT	1	0.04
Total	**18**	—

Note: Counting 3-letter day codes as symbols. Each space-separated group = 1 symbol.
Corrected probabilities: SUN=9/18=0.50, MON=4/18=0.222, others=1/18=0.056 each

Exercise 2: Solution — Shannon-Fano Tree



Codes: SUN=0, MON=10, TUE=1100, WED=1101, THU=1110, FRI=11110, SAT=11111

Exercise 2: Solution — Final Codes and Analysis

Final Shannon-Fano Codes:

Symbol	Probability	Code	Length	$p_i \times l_i$
SUN	0.500	0	1	0.500
MON	0.222	10	2	0.444
TUE	0.056	1100	4	0.224
WED	0.056	1101	4	0.224
THU	0.056	1110	4	0.224
FRI	0.056	11110	5	0.280
SAT	0.056	11111	5	0.280
Average Code Length L_{avg}:				**2.176**

Entropy: $H(X) = 2.055$ bits/symbol

Efficiency: $\eta = \frac{2.055}{2.176} \times 100\% = \boxed{94.44\%}$

Redundancy: $R = 2.176 - 2.055 = \boxed{0.121 \text{ bits/symbol}}$

Exercise 3: Find Symbols and Build Shannon-Fano Code

Problem: A sensor transmits the following readings. Construct Shannon-Fano codes.

Sensor Data Sequence (60 readings)

`LOW LOW MED LOW HIG LOW LOW MED LOW LOW CRI LOW MED LOW LOW LOW MED HIG LOW LOW`

Tasks:

1. Identify all unique sensor states
2. Count the frequency of each state
3. Calculate the probability of each state
4. Construct the Shannon-Fano code
5. Calculate average code length, entropy, and efficiency
6. Encode the message: "MED LOW CRI HIG"

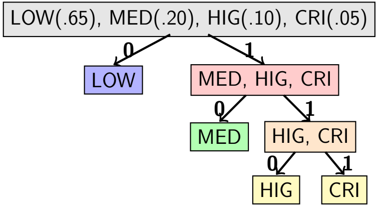
Hint: States are LOW (normal), MED (medium), HIG (high alert), CRI (critical)

Exercise 3: Solution

Symbol Analysis (Sorted by probability)

Symbol	Freq	Prob
LOW	13	0.65
MED	4	0.20
HIG	2	0.10
CRI	1	0.05
Total	**20**	**1.00**

Shannon-Fano Tree:



Final Codes: LOW = 0, MED = 10, HIG = 110, CRI = 111

Exercise 3: Solution — Final Codes and Analysis

Final Shannon-Fano Codes:

Symbol	Probability	Code	Length	$p_i \times l_i$
LOW	0.65	0	1	0.65
MED	0.20	10	2	0.40
HIG	0.10	110	3	0.30
CRI	0.05	111	3	0.15
			Average Code Length L_{avg}:	**1.50**

Entropy: $H(X) = -(0.65 \log_2 0.65 + 0.20 \log_2 0.20 + 0.10 \log_2 0.10 + 0.05 \log_2 0.05)$

$$H(X) = 0.406 + 0.464 + 0.332 + 0.216 = \boxed{1.418 \text{ bits/symbol}}$$

Efficiency: $\eta = \frac{1.418}{1.50} \times 100\% = \boxed{94.53\%}$

Encoding "MED LOW CRI HIG": $10 + 0 + 111 + 110 = \boxed{100111110}$ (9 bits)

Key Takeaways

1. **Entropy** measures the average information content

$$H(X) = -\sum_i p_i \log_2 p_i$$

2. **Shannon-Fano** is a prefix-free, variable-length coding technique
3. **Algorithm:** Sort $\rightarrow$ Divide equally $\rightarrow$ Assign $0/1$ $\rightarrow$ Recurse
4. **Average code length** should satisfy:

$$H(X) \leq L_{avg} < H(X) + 1$$

5. **Efficiency:** $\eta = \frac{H(X)}{L_{avg}} \times 100\%$
6. Shannon-Fano is **near-optimal** but **not always optimal**

Practice Problems

Problem 1: Construct Shannon-Fano codes for:

Symbol	A	B	C	D	E
Probability	0.40	0.20	0.15	0.15	0.10

Problem 2: Given these codes, verify they are prefix-free:

A=0, B=10, C=110, D=1110, E=1111

Problem 3: A source has entropy 2.5 bits/symbol. A code achieves $L_{avg} = 2.7$ bits/symbol. Calculate efficiency and redundancy.

Problem 4: Encode "CABBAGE" using the code from Problem 1.

Important Formulas Summary

Entropy

$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$

Average Code Length

$L_{avg} = \sum_{i=1}^{n} p_i \cdot l_i$

Efficiency

$\eta = \frac{H(X)}{L_{avg}} \times 100\%$

Redundancy

$R = L_{avg} - H(X)$

Thank You!

Questions?

"The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point."

— Claude Shannon, 1948