

Course «Algorithms» · Lecture 12

Greedy Algorithms: Huffman Coding

Textbook: A. Levitin, Section 9.4

Today’s question

How do we write a text using **as few bits as possible?**

- prefix-free codes and code trees
- Huffman’s algorithm (1952), still inside ZIP, PNG, JPEG
- why the Huffman code is optimal
- the four greedy algorithms side by side

Fixed vs. variable length

Fixed-length (like ASCII): every symbol gets the same number of bits.

$$\lceil \log_2 n \rceil \ \text{ bits per symbol}: \qquad n = 5 \ \Rightarrow \ 3 \text{ bits}$$

Variable-length (like Morse): short codewords for frequent symbols.

> Danger: codewords 0, 1, 01. Is “01” one symbol or two? **Ambiguous.**

The fix: a prefix-free code

No codeword is a prefix of another codeword.

Decoding is unique and instant: read bits until they match a codeword, output it, start again. No look-ahead, no backtracking.

Codes as binary trees

- Symbols are the **leaves**
- Left edge = 0, right edge = 1
- Codeword = labels on the path from the root
- Codeword length $\ell(c)$ = depth of the leaf

A leaf never lies on the path to another leaf, so the prefix property is automatic.

What we minimize

Each symbol c has frequency f(c) and codeword length ℓ(c). The **expected code length**, bits per symbol:

$$L = \sum_{c} f(c)\,\ell(c)$$

Goal: a prefix-free code with the smallest possible L.

Huffman’s algorithm

Repeatedly **merge the two least frequent trees** into one whose weight is their sum.

```
Huffman(C with frequencies f)
  put a leaf for each symbol into priority queue Q
  while Q has more than one tree do
    x ← ExtractMin(Q);  y ← ExtractMin(Q)
    z ← new node: left x, right y, f(z) = f(x) + f(y)
    Insert(Q, z)
  return the remaining tree
```

Rare symbols are merged first and sink deepest; frequent ones stay near the root.

Example: five symbols

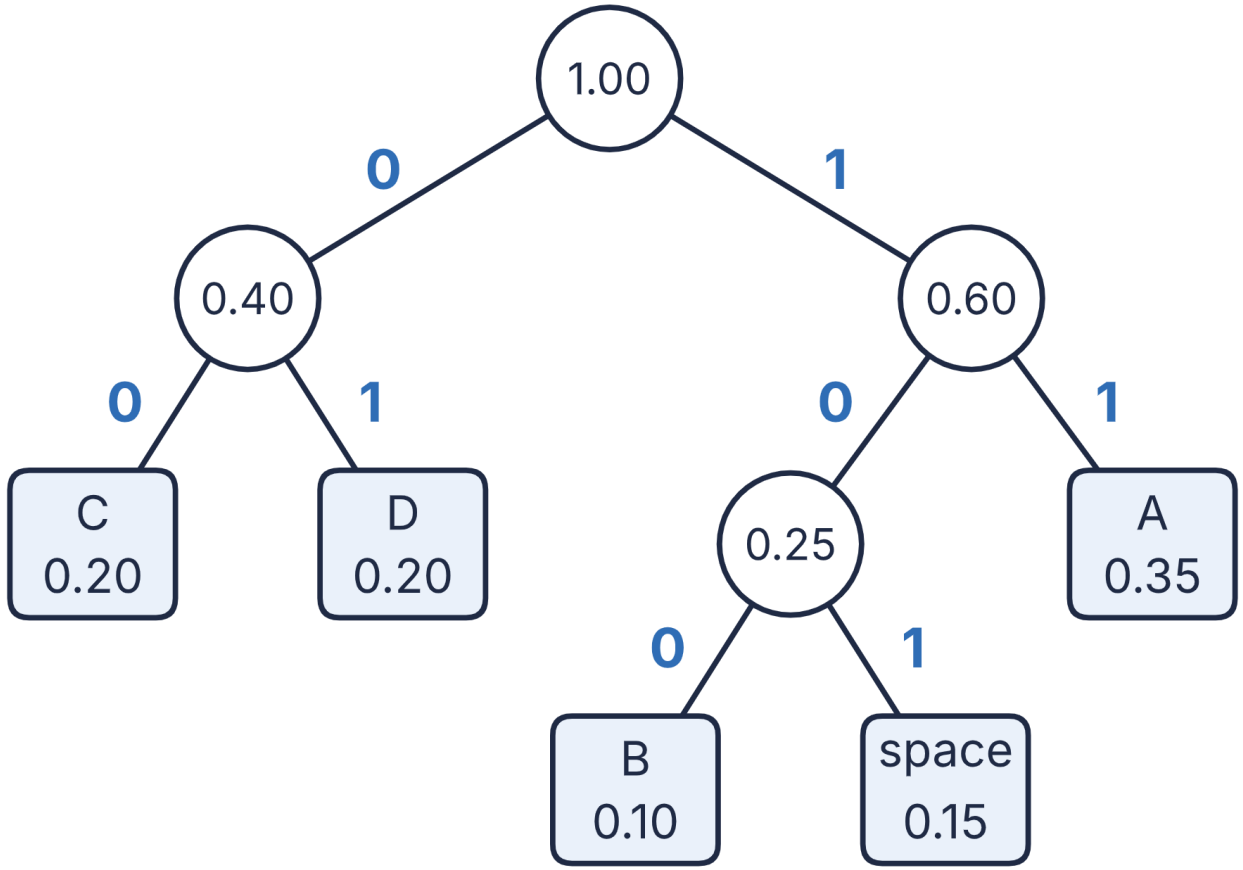
Symbol	A	B	C	D	space
Frequency	0.35	0.10	0.20	0.20	0.15

At each step take the two smallest weights from the queue.

The merges

Step	Queue before the step	Merge	New node
1	B 0.10, space 0.15, C 0.20, D 0.20, A 0.35	B + space	N1 = 0.25
2	C 0.20, D 0.20, N1 0.25, A 0.35	C + D	N2 = 0.40
3	N1 0.25, A 0.35, N2 0.40	N1 + A	N3 = 0.60
4	N2 0.40, N3 0.60	N2 + N3	root, 1.00

The Huffman tree



Lighter subtree on the left (0), heavier on the right (1).

The resulting code

Symbol	f(c)	Codeword	ℓ(c)
A	0.35	11	2
B	0.10	100	3
C	0.20	00	2
D	0.20	01	2
space	0.15	101	3

The two rarest symbols, B and space, got the longest codewords.

How good is it?

$$L = 0.35 \cdot 2 + 0.10 \cdot 3 + 0.20 \cdot 2 + 0.20 \cdot 2 + 0.15 \cdot 3 = 2.25$$

bits per symbol, versus **3** for a fixed-length code. Saving:

$$\frac{3 - 2.25}{3} = 25\%$$

Encoding and decoding

$$ABCD \quad \rightarrow \quad 11 \cdot 100 \cdot 00 \cdot 01 = 111000001$$

Decode 111000001 from the left:

1 → 11 = A

1 → 10 → 100 = B

0 → 00 = C

0 → 01 = D

No choice at any point: that is the prefix property.

Optimality, fact 1: greedy choice

Some optimal tree has the two rarest symbols x, y as **siblings at the maximum depth**.

Take an optimal tree and deepest siblings a, b. Swap x with a and y with b. One swap changes the cost by

$$(f(x) - f(a))(\ell(a) - \ell(x)) \leq 0$$

since f(x) ≤ f(a) and ℓ(x) ≤ ℓ(a). The cost never grows: an exchange argument, as for Prim.

Fact 2: optimal substructure

Replace siblings x, y by one new symbol z:

$$f(z) = f(x) + f(y)$$

$$L(T) = L(T') + f(x) + f(y)$$

The difference is a constant, so T is optimal exactly when T′ is.

Induction: safe greedy merge + optimal smaller problem ⇒ **no prefix code beats Huffman.**

Running time and the entropy limit

- n – 1 merges, each O(log n) with a heap: **O(n log n)**
- Frequencies presorted: two queues give **O(n)**

No prefix code beats the **entropy**, and Huffman is within one bit of it:

$$H = -\sum_{c} f(c) \log_2 f(c) \qquad H \leq L_{\text{Huffman}} < H + 1$$

Our example: H ≈ 2.20 bits, Huffman gives 2.25.

In practice

- Optimal among codes with a whole number of bits **per symbol**; arithmetic coding can do better
- Needs frequencies: two passes, or adaptive Huffman in one pass
- Inside DEFLATE (ZIP, gzip, PNG) and JPEG

The four greedy algorithms

	Prim	**Kruskal**	**Dijkstra**	**Huffman**
Problem	MST	MST	shortest paths from s	optimal prefix code
Greedy choice	cheapest edge leaving the tree	cheapest edge with no cycle	nearest unfinished vertex	merge two lightest trees
Structure	priority queue	sort + union-find	priority queue	priority queue
Time	$\Theta(\|E\| \log \|V\|)$	$\Theta(\|E\| \log \|V\|)$	$\Theta(\|E\| \log \|V\|)$	$\Theta(n \log n)$
Proof	cut property, exchange	cut property, exchange	invariant, contradiction	exchange, substructure

The algorithm is easy; the proof is what makes it an algorithm.

Common mistakes

- **Symbols in internal nodes:** the prefix property breaks
- **Forgetting the code table:** for small inputs it can eat all the savings
- **“The” Huffman code:** ties give different codes, same average length

Summary

- Prefix-free code = binary tree with symbols at the leaves
- Minimize $L = \Sigma\, f(c)\, \ell(c)$
- Huffman merges the two least frequent trees: 2.25 vs. 3 bits
- Optimal by greedy choice + optimal substructure
- $O(n \log n)$; $H \le L < H + 1$

Review questions

1. What is a prefix-free code, and why does it guarantee unambiguous decoding?
2. Build the Huffman code for P 0.4, Q 0.3, R 0.15, S 0.1, T 0.05. Compute the average length and compare with a fixed-length code.
3. Explain the two properties used in the optimality proof.