Huffman Codes: An Optimal Data Compression Method

## Data Compression
- Binary character code ("code")
  - each k-bit source string maps to unique code word (e.g., k=8)
  - "compression" alg: concatenate code words for successive k-bit "characters" of source
- Fixed/variable length codes
  - all code words equal length?
  - Prefix codes
    - no code word is prefix of another (simplifies decoding)

## Compression Example
- 100k file, 6 letter alphabet:
- File Size:
  - ASCIl, 8 bits/char: 800kbits
  - 2^3>6: 3 bits/char: 300kbits
  - 00,01,10 for a,b,d; 11xx for c,e,f:
    - 2.52 bits/char 74%*2 + 26%*4: 252kbits
  - Optimal?
  - Why?
    - Storage, transmission vs 1Ghz cpu

## Prefix Codes = Trees

## Greedy Idea #1
- Put most frequent under root, then recurse ...

## Greedy Idea #1
- Put most frequent under root, then recurse ...
  - Too greedy: unbalanced tree
Greedy idea #2

Group least frequent letters near bottom

Huffman’s Algorithm (1952)

Algorithm:
- Insert node for each letter into priority queue by freq
- While queue length > 1 do
- Remove smallest 2; call them x, y
- Make new node z from them, with f(z) = f(x) + f(y)
- Insert z into queue

Analysis:
- \( O(n) \) heap ops: \( O(n \log n) \)

Goal:
- Minimize \( B(T) = \sum_{c \in C} freq(c) \times depth(c) \)

Correctness: ???

Correctness Strategy

Optimal solution may not be unique, so cannot prove that greedy gives the only possible answer.

Instead, show that greedy’s solution is as good as any.

Defn: A pair of leaves is an inversion if
- \( \text{depth}(x) \geq \text{depth}(y) \)
- \( \text{freq}(x) \geq \text{freq}(y) \)

Claim: If we flip an inversion, cost never increases.

Why? All other things being equal, better to give more frequent letter the shorter code.

\[
\text{before} \quad \text{after}
\]
\[
(d(x) \times f(x) + d(y) \times f(y)) - (d(x) \times f(y) + d(y) \times f(x)) =
(d(x) - d(y)) \times (f(x) - f(y)) \geq 0
\]

I.e. non-negative cost savings.
Lemma 1: “Greedy Choice Property”

The 2 least frequent letters might as well be siblings at deepest level

- Let a be least freq, b 2nd
- Let u, v be siblings at max depth, f(u) ≤ f(v)
- Then (a,u) and (b,v) are inversions. Swap them.

Proof:

\[ B(T) = \sum_{c \in C} \frac{d_r(c) \cdot f(c)}{2^n} \]

\[ B(T) - B(T') = d_r(x) \cdot (f(x) + f(y)) - d_r(z) \cdot f'(z) \]

\[ = (d_r(z) + 1) \cdot f'(z) - d_r(z) \cdot f'(z) \]

\[ = f'(z) \]

Suppose \( f'(z) \) (having x & y as siblings) is better than \( T \), i.e., \( B(T') < B(T) \). Collapse x & y to z, forming \( \tilde{T} \).

\[ B(\tilde{T}) - B(\tilde{T}') = f'(z) \]

Then:

\[ B(\tilde{T}) = B(\tilde{T}') - f'(z) < B(T) - f'(z) = B(T') \]

Contradicting optimality of \( T' \)

Lemma 2: “Optimal Substructure”

Let (C, f) be a problem instance: C an n-letter alphabet with letter frequencies f(c) for c in C. For any x, y in C, let C’ be the (n-1) letter alphabet C\{x,y} ∪ \{z\} and for all c in C’ define

\[ f'(c) = f(c), \quad \text{if } c \neq x, y, z \]

Let \( T' \) be an optimal tree for (C’,f’). Then

\[ T \]

is optimal for (C,f) among all trees having x, y as siblings

Theorem: Huffman gives optimal codes

Proof: induction on |C|

- Basis: n=1,2 – immediate
- Induction: n>2
  - Let x,y be least frequent
  - Form C’, f’, & z, as above
  - By induction, \( T' \) is opt for (C’,f’)
  - By lemma 2, \( T' \rightarrow T \) is opt for (C,f) among trees with x, y as siblings
  - By lemma 1, some opt tree has x, y as siblings
  - Therefore, \( T \) is optimal.

Data Compression

- Huffman is optimal.
- BUT still might do better!
  - Huffman encodes fixed length blocks. What if we vary them?
  - Huffman uses one encoding throughout a file. What if characteristics change?
  - What if data has structure? E.g. raster images, video…
  - Huffman is lossless. Necessary?
- LZW, MPEG, …