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:
    - ASCII, 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 (45% + 26% * 2, 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 \( R(T) = \sum_{c \in C} \text{freq}(c) \times \text{depth}(c) \)
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.

\[ \text{Claim: If we flip an inversion, cost never increases.} \]

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

\[ \text{before} \quad \text{after} \]

\[ \begin{align*}
& (d(x) \cdot f(x) + d(y) \cdot f(y)) - (d(x) \cdot f(y) + d(y) \cdot f(x)) = \\
& (d(x) - d(y)) \cdot (f(x) - f(y)) \geq 0
\end{align*} \]

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:

\[ B(T) = \sum_{c \in C} d_c \cdot f(c) \]
\[ B(T) - B(T') = d_y \cdot (f(x) + f(y)) - d_x \cdot f'(z) = \\
= (d_y \cdot f(x) + d_x \cdot f_y) - (d_x \cdot f(y) + d_y \cdot f(x)) = \\
= (d_x - d_y) \cdot (f(x) - f(y)) \geq 0 \]

Suppose \( \hat{T} \) (having x & y as siblings) is better than \( T \), i.e.
\[ B(\hat{T}) < B(T) \].
Collapse x & y to z, forming \( \hat{T}' \); as above:
\[ B(\hat{T}) - B(\hat{T}') = f'(z) \]
Then:
\[ B(\hat{T}') = B(\hat{T}) - f'(z) < B(T) - f'(z) = B(T') \]
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 \( C - \{x,y\} \cup \{z\} \) and for all \( c \) in \( C' \) define
\[ f'(c) = \begin{cases} f(c), & \text{if } c = x,y,z \\ f(x) + f(y), & \text{if } c = z \end{cases} \]

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, …