CSE 421
Algorithms

Huffman Codes:
An Optimal Data Compression Method

Compression Example

100k file, 6 letter alphabet:

File Size:
ASCII, 8 bits/char: 800kbits
2^7 > 6; 3 bits/char: 300kbits

Why?
Storage, transmission vs 5 Ghz cpu

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 "strings" of source

Fixed/variable length codes
all code words equal length?
Prefix codes
no code word is prefix of another (unique decoding)
Prefix Codes = Trees

Greedy Idea #1

Put most frequent under root, then recurse …

Greedy Idea #2

Top down: Divide letters into 2 groups, with ~50% weight in each; recurse (Shannon-Fano code)

Again, not terrible

But this tree can easily be improved! (How?)
Greedy idea #3

Bottom up: 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 $B(T) = \sum_{c \in C} \text{freq}(c) \times \text{depth}(c)$
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.

How: an exchange argument

Claim: If we flip an inversion, cost never increases.
Why? All other things being equal, better to give more frequent letter the shorter code.

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

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, \( f(u) \leq f(v) \)
(why must they exist?)
Then (a,u) and (b,v) are inversions. Swap them.

Lemma 2

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

\[ f'(c) = \begin{cases} f(c), & \text{if } c \neq x, y, z \\ f(x)+f(y), & \text{if } c = z \end{cases} \]

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

\[ T = T' \]

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

\[ B(T) = \sum_{c \in C} d_T(c) \cdot f(c) \]
\[ B(T) - B(T') = d_T(x) \cdot (f(x) + f(y)) - d_T(z) \cdot f'(z) \]
\[ = (d_T(z) + 1) \cdot f'(z) - d_T(z) \cdot f'(z) \]
\[ = f'(z) \]

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 \( T' \)

---

**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, …