CSE 421
Intro to Algorithms
Winter 2000

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:
  - a 45%
  - b 13%
  - c 12%
  - d 16%
  - e 9%
  - f 5%
- 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 74%*2 +26%*4: 252kbits
  - Optimal?
- Why?
  - Storage, transmission vs 1Ghz cpu

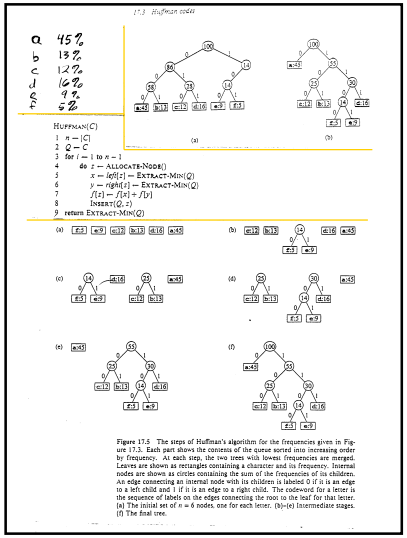


Figure 17.5 The steps of Huffman's algorithm for the frequencies given in Figure 17.3. Each part shows the contents of the queue sorted into increasing order by frequency. At each step, the two trees with lowest frequencies are merged. Leaves are shown as rectangles containing a character and its frequency. Internal nodes are shown as circles containing the sum of the frequencies of its children. An edge connecting an internal node with its children is labeled 0 if it is an edge to a left child and 1 if it is an edge to a right child. The codeword for a letter is the sequence of labels on the edges connecting the root to the leaf for that letter. (a) The initial set of $n = 6$ nodes, one for each letter. (b)–(e) Intermediate stages. (f) The final tree.

Huffman's Algorithm

Algorithm:

```
HUFFMAN(C)
    n ← |C|
    Q ← C
    for i ← 1 to n − 1
        do z ← ALLOCATE-NODE()
           x ← left[z] ← EXTRACT-MIN(Q)
           y ← right[z] ← EXTRACT-MIN(Q)
           f[z] ← f[x] + f[y]
           INSERT(Q, z)
    return EXTRACT-MIN(Q)
```

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

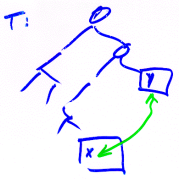
Goal:
Minimize B(T) = $\sum_{c \in C}$ freq(c) * depth(c)

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.

T:



Defn: A pair of leaves is an inversion if
depth(x) ≥ depth(y)
and
freq(x) ≥ 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.

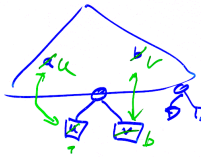
before: $d(x)f(x) + d(y)f(y)$ − after: $(d(x)f(y) + d(y)f(x))$

$= (d(x) - d(y))(f(x) - f(y)) \geq 0$

i.e. cost savings is non-negative

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 be least freq at max depth, v its sibling
- Then (a,u) and (b,v) are inversions. Swap them.

Lemma 2:
"Optimal Substructure"

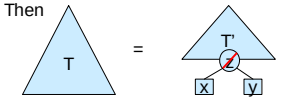
Let (C, f) be a problem instance: C an n-letter alphabet with letter frequencies f(c) for c in C.

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

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

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

Then



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

Proof:

$$B(T) = \sum_{c \in C} d_T(c) \cdot f(c)$$

$$B(T) - B(T') = d_T(x)(f(x)+f(y)) - d_{T'}(z) f(z)$$

$$= (d_{T'}(z)+1) f(z) - d_{T'}(z) f(z)$$

$$= f(z) + f(z)$$

Suppose $\hat{T}$ (having x,y as siblings) better than T: $B(\hat{T}) < B(T)$

Collapse x,y to z, forming $\hat{T}'$.

$B(\hat{T}) - B(\hat{T}') = f(z)$

Then

$B(\hat{T}') = B(\hat{T}) - f(z) < B(T) - f(z) = B(T')$

Contradiction

Theorem:
H. 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'→T is opt for (C,f) among trees with x,y as siblings
  - By lemma 1, some opt tree does have 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, …