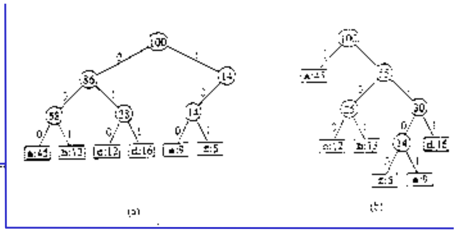
Huffman Codes

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)

a	45%
b	13%
c	12%
d	16%
e	9%
f	5%



HUFFMAN(C)

```
1  n ← |C|
2  Q ← C
3  for i ← 1 to n − 1
4       do z ← ALLOCATE-NODE()
5          x ← left[z] ← EXTRACT-MIN(Q)
6          y ← right[z] ← EXTRACT-MIN(Q)
7          f[z] ← f[x] + f[y]
8          INSERT(Q, z)
9  return EXTRACT-MIN(Q)
```

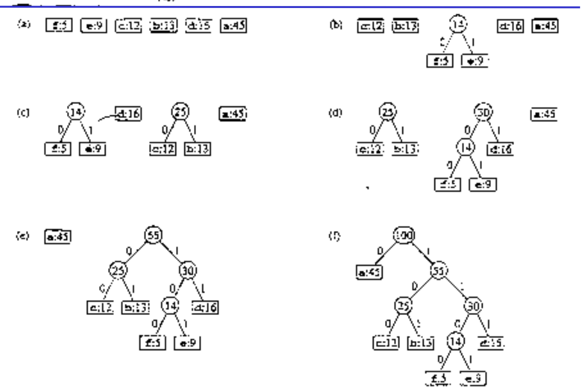


Figure 17.5 The steps of Huffman's algorithm for the frequencies given in Figure 17.3. Each part shows the contents of the queue sorted into increasing order by frequency. At each step, the two trees with lowest frequencies are merged. Leaves are shown as rectangles containing a character and its frequency. 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.

Strategy

1. Opt solution maybe not unique (So cannot prove greedy gives only possible answer.)
2. Show greedy's solution is as good as any.

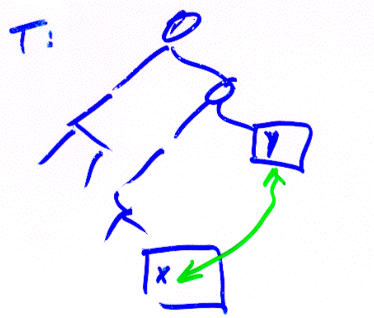
Huffman Code Objective

compressed length =

$$\sum_{c \in C} \text{freq}(c) \cdot \text{length}(c)$$

$$= \sum_{c \in C} \text{freq}(c) \cdot \text{depth}(c)$$

$$= B(T)$$



Defn: a pair of leaves x, y 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 shorter code

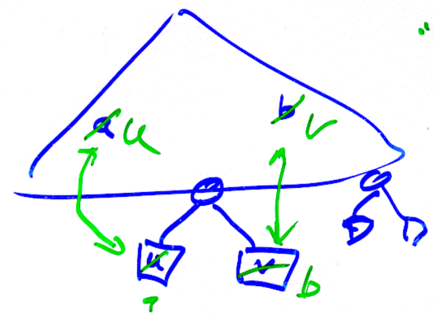
$$\underbrace{d(x)f(x) + d(y)f(y)}_{\text{before}} - \underbrace{\left(d(x)f(y) + d(y)f(x)\right)}_{\text{after}}$$

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

i.e. cost savings is non-negative

Lemma 17.2

2 least frequent chars might as well be siblings, at deepest level.



"greedy choice property"

- Let a be least freq letter, b 2nd.
- Let u be least freq leaf at max depth, v its sibling.
- Then (a,u) & (b,v) are inversions. Swap them.

Modified lemma 17.3

"optimal substructure"

Let C be an n-letter alphabet with frequencies $f(c)$.

$\forall x,y \in C$, let C' be $(n-1)$-letter alphabet $C - \{x,y\} \cup \{z\}$ with freq $f'(c) = \begin{cases} f(c), & c \neq x,y,z \\ f(x)+f(y), & \text{if } c = z \end{cases}$

Let T' be optimal tree for C'.

Then $T = T' - z +$  is opt for C among trees having x,y as siblings.

Proof:

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

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

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

$$= f'(z)$$

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 17.4

Huffman alg. gives optimal code.

Proof: by induction on # of letters.

basis: $n = 1$ trivial

ind: $n \geq 2$

- let x, y be least frequent.
- Form z, C' as above
- By induction T' is opt for C'
- By modified 17.3, $T' \to T$ is opt for C among trees with x, y siblings
- By 17.2 some opt. tree does have x, y siblings

$\therefore$ T is optimal.

Data Compression

Huffman is optimal.

But still might do better!

1. Huffman encodes fixed length blocks.
2. Huffman uses one encoding throughout a file. What if characteristics change?
3. What if data has structure? E.g. raster images, video
4. Lossless

Ziv-Lempel, MPEG, ...