

15 Greedy Algorithms: Huffman Codes

15.1 Preliminaries

- A (lossless) data compression technique.

Example: Fig 16.3, Pg 386

Fixed Length Codes: each character will be stored as a binary string of the same length. (Fig 16.4(a), Pg 387).

Can also have *variable length codes*. (Fig 16.4(b), Pg 387).

- Higher frequency characters have smaller codes.
- Turns out that this is optimal.

- Notice that no code is a prefix of the other (the prefix code property).
- Prefix codes make it easy to extract (parse) the characters from the bit string.

Example

Binary tree is a convenient representation for these codes. Eg, Fig 16.4

Notice that this is a *full* binary tree; i.e., each internal (non-leaf) node has two children. (Either you have 2 kids or no kids).

Lemma: An optimal solution requires a full binary tree. (why?)

15.2 Huffman Code Algorithm

- Algorithm will construct an optimal binary tree.
- C = set of leaves representing characters.
- Q = a priority queue (remember, we discussed this several years ago) on frequency.
 - insert
 - extract-min

Huffman(C)

$n \leftarrow |C|$

$Q \leftarrow C$

for $i \leftarrow 1$ **to** $n - 1$ **do**

$z \leftarrow \text{ALLOCATE-NODE}();$

$x \leftarrow \text{left}[z] \leftarrow \text{EXTRACT-MIN}(Q)$

$y \leftarrow \text{right}[z] \leftarrow \text{EXTRACT-MIN}(Q)$

$f[z] \leftarrow f[x] + f[y]$

 INSERT(Q, z)

return EXTRACT-MIN(Q)

Step thru algorithm on example (Fig 16.5).