chang@umbc.edu

More Greedy Algorithms

Huffman Codes - optimal prefix code

ASCII is a fixed length code

$A = 41_{16} = 0100\ 0001$ $\quad$ $a = 61_{16} = 0110\ 0001$

$B = 42_{16} = 0100\ 0010$ $\quad$ $b = 62_{16} = 0110\ 0010$

$\vdots$ $\quad$ $\vdots$

$Z = 5A_{16} = 0101\ 1010$ $\quad$ $z = 7A_{16} = 0111\ 1010$

Each character uses 8 bits, no matter how often the character is used

Variable-length codes

shorter codes for frequently used characters
longer codes for infrequent characters

Example: VCR-plus ← who uses VCRs anymore?

Question: how to decode?

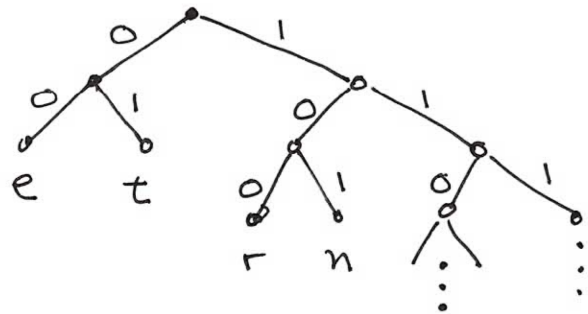
where do letters start?

1000 0101 01

Prefix-free code = prefix code ← boneless chicken = boned chicken

No character has an encoding that is the prefix of another character's encoding

Represent prefix codes as a binary tree:



e = 00
t = 01
r = 100
n = 101

$$\underbrace{1\,0\,0}_{r}\;\underbrace{0\;0}_{e}\;\underbrace{1\,0\,1}_{n}\;\underbrace{0\,1}_{t}$$

Problem: Given an "alphabet" of characters and a frequency for each character, what is the optimal prefix code?

Minimize

frequency of x

$$B(T) = \sum_{x \in C} f(x)\, d_T(x)$$

expected # of bits

tree for prefix code

char

char set

depth of x in tree

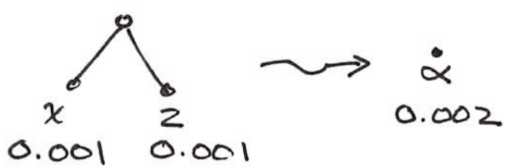
= length of encoding of x

Greedy algorithm: Huffman codes

① Sort characters by frequency

② Take 2 characters with least frequency: x & z.
(These will be placed at the bottom.)

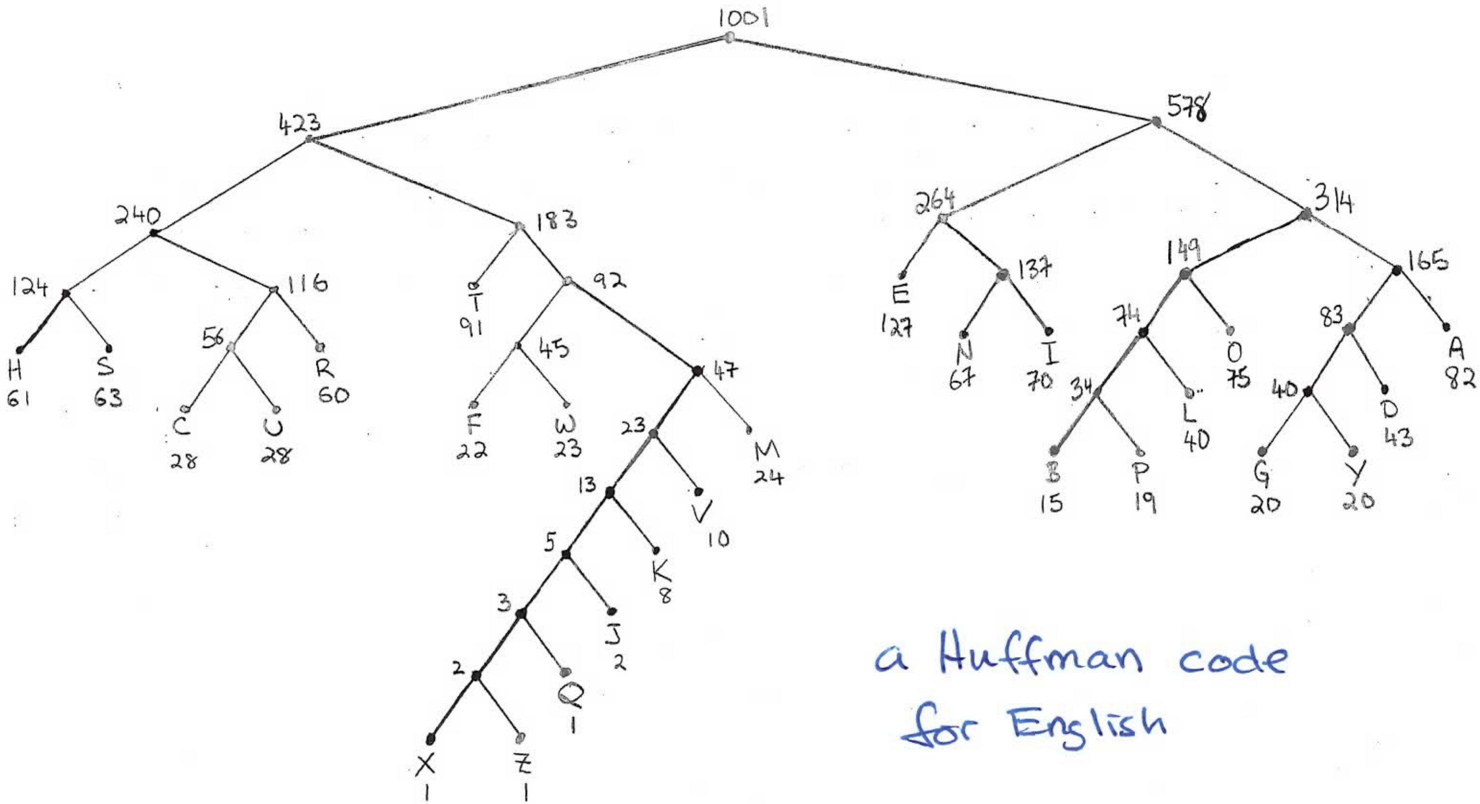
③ Remove x & z from the alphabet and replace them with a "fake" character with



$$C' = (C - \{x, z\}) \cup \{\alpha\}$$

letter	probability	letter	probability
A	.082	N	.067
B	.015	O	.075
C	.028	P	.019
D	.043	Q	.001
E	.127	R	.060
F	.022	S	.063
G	.020	T	.091
H	.061	U	.028
I	.070	V	.010
J	.002	W	.023
K	.008	X	.001
L	.040	Y	.020
M	.024	Z	.001

Letter frequency table for English



a Huffman code for English

Running Time:

Use a heap to store characters by frequency.
Smaller frequencies on top.

Each iteration = 2 ExtractMin + 1 Insert

Build Heap		$\Theta(n)$
Extract Min x2	n times	$2 \times n \times \Theta(\log n)$
Insert		$n \times \Theta(\log n)$
		$\Theta(n \log n)$

Huffman Codes are optimum, but why?

<u>Claim #1:</u> Suppose T is a binary tree for some prefix code.

Let x & y be two characters with lowest frequency.

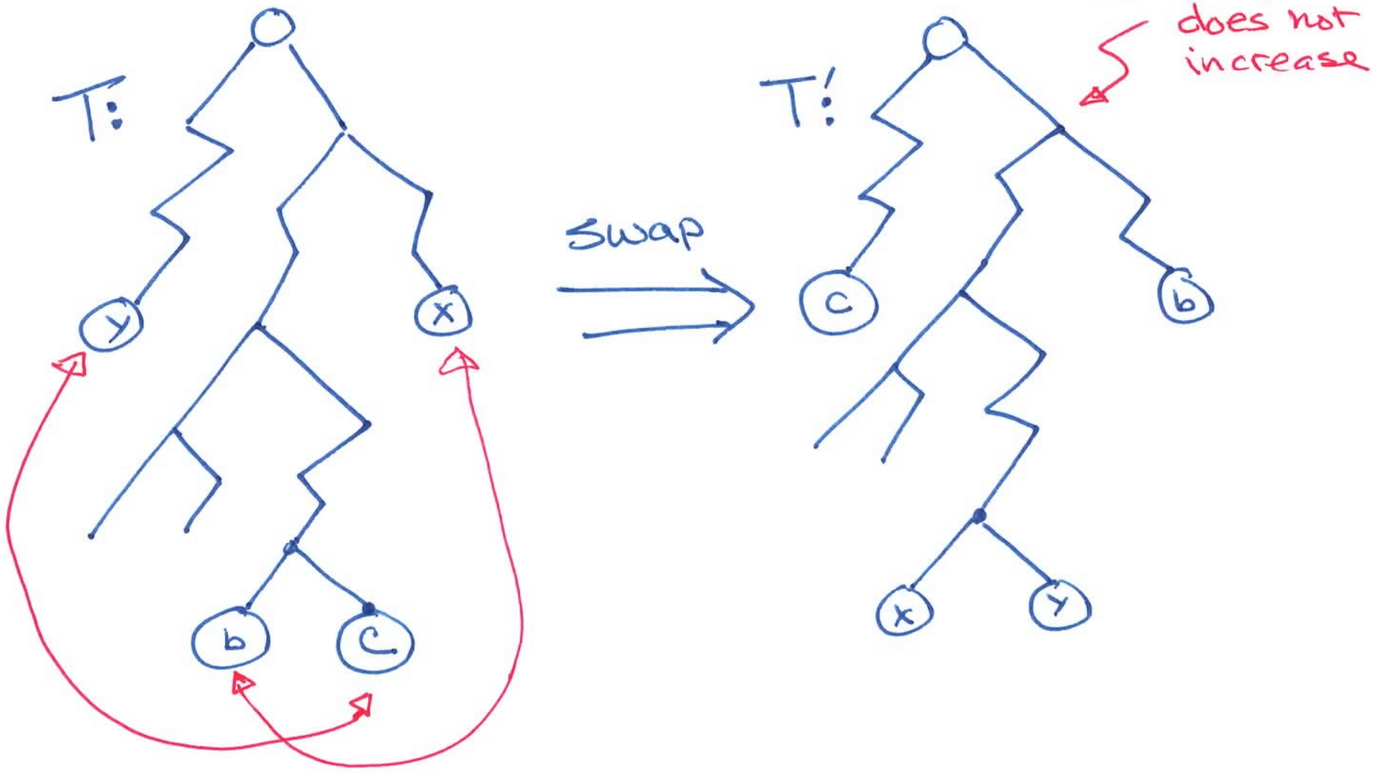
Let b & c be two characters, siblings at maximum depth.

gives $f(x) \le f(b)$ & $f(y) \le f(c)$

Suppose that $f(x) \le f(y)$ & $f(b) \le f(c)$.

Then, swapping x and b and swapping y and c will result in T' with no greater expected code length.

Claim #1:



Proof of Claim #1: (by calculation)

removed x at depth $d_T(x)$

added x at new depth

$$\begin{aligned}
B(T') = B(T) &- f(x)\cdot d_T(x) + f(x)\cdot \underbrace{d_{T'}(x)}_{= d_T(b)} \\
&- f(y)\cdot d_T(y) + f(y)\cdot \underbrace{d_{T'}(y)}_{= d_T(c)} \\
&- f(b)\cdot d_T(b) + f(b)\, d_{T'}(b) \\
&- f(c)\, d_T(c) + f(c)\, d_{T'}(c)
\end{aligned}$$

$$\begin{aligned}
= B(T) &+ f(x)\cdot[d_T(b) - d_T(x)] \\
&+ f(y)\cdot[d_T(c) - d_T(y)] \\
&+ f(b)\cdot[d_T(x) - d_T(b)] \\
&+ f(c)\cdot[d_T(y) - d_T(c)]
\end{aligned}$$

negations

negations

lowest freq.

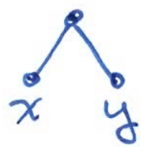
$$= B(T) + \overbrace{\underbrace{[f(b) - f(x)]}_{\geq 0}\underbrace{[d_T(x) - d_T(b)]}_{\leq 0}}^{\leq 0} + \overbrace{\underbrace{[f(c) - f(y)]}_{\geq 0}\underbrace{[d_T(y) - d_T(c)]}_{\leq 0}}^{\leq 0}$$

max depth

$$\leq B(T) \qquad \blacksquare$$

Claim #2: Let T be a binary tree for a prefix code. Suppose x & y are sibling leaves.

Let T' be obtained from T by replacing the subtree



with

where α is assigned the frequency of x & y combined: $f(\alpha) = f(x) + f(y)$.

Then, $B(T') = B(T) - f(x) - f(y)$

Proof of Claim #2: (by calculation)

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

$$= B(T) - f(x)\,d_T(x) - f(y)\,d_T(x) + f(\alpha)\left[d_T(x) - 1\right]$$

$$= B(T) - \left[f(x) + f(y)\right] d_T(x) + f(\alpha)\,d_T(x) - f(\alpha)$$

$$= B(T) - \cancel{f(\alpha)\cdot d_T(x)} + \cancel{f(\alpha)\,d_T(x)} - f(\alpha)$$

$$= B(T) - f(x) - f(y)$$

does not depend on the depth of x & y.

$\boxtimes$

Theorem: Huffman codes are optimum.

Proof by induction: Induction on # of characters.

Induction Hypothesis: Hufman codes are optimum for alphabets with n characters.

Call this $P(n)$.

Base Case: $n = 0$. Trivial.

Induction Case: Prove $P(n) \Rightarrow P(n+1)$

Suppose not. Let X be a prefix code s.t.

$$B(X) < B(T)$$

where T is the tree for a Huffman code.

Let x & y be two characters at maximum depth of T. Then, x & y have the lowest frequency.

By Claim #1, we can swap x & y to maximum depth in X and obtain

$$B(X') \leq B(X)$$

By Claim #2, we can replace x & y (now guaranteed to be siblings) with α in X' and in T, and obtain X'' and T'

$$B(X'') = B(X') - f(x) - f(y)$$
$$B(T') = B(T) - f(x) - f(y)$$

Then, $B(X') \leq B(X) < B(T)$

$$\Rightarrow \quad B(X') - f(x) - f(y) < B(T) - f(x) - f(y)$$

$$\Rightarrow \quad B(X'') < B(T')$$

But, T' is the Huffman code for the n character alphabet $C' = (C - \{x, y\}) \cup \{\alpha\}$.

Furthermore, X'' is also a prefix code for C'.

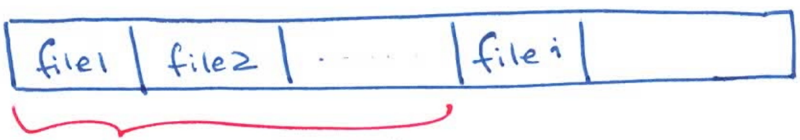
Then, Huffman codes are not optimum for C' which has n characters. This contradicts $P(n)$.

$\boxtimes$

Tape Archive Problem:

n files stored on magnetic tape.

Order matters:



must scan through files 1 thru $i-1$ to reach file i

Some files accessed more frequently.

Two ideas:

1. Put shorter files in front
2. Put more frequently used files in front.

Problems:

1. Lots of short files that are not used often
2. Very long files used most frequently.

Solution: use frequency / length ratio.

i^{th} file has length l_i & frequency p_i

Assume files are already sorted by p_i/l_i ratio

$$p_1/l_1 \geq p_2/l_2 \geq p_3/l_3 \geq \cdots \geq p_n/l_n$$

Show that this minimize expected access time

$$\sum_{i=1}^{n} p_i \,(\text{access time for file } i) = \sum_{i=1}^{n} p_i \underbrace{\left(\sum_{j=1}^{i} l_j\right)}$$

read over all files before
file i and file i itself.

Usual argument: if some ordering X does not place file 1 at the beginning, it could have.

WARNING: Cannot swap to the front.

EXAMPLE: 3 files

$p_1 = 0.80$ $\ell_1 = 80$ $p_2 = 0.10$ $\ell_2 = 100$ $p_3 = 0.10$ $\ell_3 = 9$

$p_1/\ell_1 = 0.01$ $p_2/\ell_2 = 0.001$ $p_3/\ell_3 = 0.0111\ldots$

order:	
123	$0.8(80) + 0.10(80+100) + 0.10(80+100+9) = 100.9$
132	$0.8(80) + 0.10(89) + 0.10(189) = 91.8$
213	$0.10(100) + 0.80(180) + 0.10(189) = 172.9$
231	$0.10(100) + 0.10(109) + 0.80(189) = 172.1$
312	$0.10(9) + 0.80(89) + 0.10(189) = 91$ opt.
321	$0.10(9) + 0.10(109) + 0.80(189) = 163$

swapping 3 to the front increases expected access time

But we can swap adjacent files

S: file file file .... file i file j .. file ...

and $p_j/\ell_j > p_i/\ell_i$. swap file i & file j.

S' = $\underbrace{\text{file . file file ....}}_{\text{access time not changed}}$ file j file i $\underbrace{\text{.... file ....}}_{\text{access time not changed}}$

Expected Access Time

$EAT(S') - EAT(S) = p_i \ell_j - p_j \ell_i$

must read file j to get to file i

no longer need to read file i to get to file j

But, $p_j/\ell_j > p_i/\ell_i \Rightarrow p_i\ell_j - p_j\ell_i < 0$. So, $EAT(S') < EAT(S)$.

Let G be the ordering produced by greedy:

file 1 . file 2 ... file n.

How do we make S look like G

by swapping adjacent files???

Answer: Bubble Sort

Let $S^{(t)}$ be the ordering obtained from Bubble Sorting S according to largest p_i/l_i ratio at the t^{th} step of Bubble Sort.

Then, $EAT(S) \geq EAT(S^{(1)}) \geq EAT(S^{(2)}) \geq \ldots\ldots \geq EAT(S^{(m)})$

sorted!

Proof that Greedy is optimum:

Greedy solution

Suppose some solution S has $EAT(S) < EAT(G)$.

Then, Bubble Sort S by p_i/ℓ_i order.

$$EAT(G) > EAT(S) \geq EAT(S^{(1)}) \geq \ldots \geq EAT(S^{(m)}) = EAT(G)$$

... method

same ordering

$\therefore$ $EAT(G) > EAT(G)$

a contradiction.

$\boxtimes$