Huffman Codes

Sept 23, 2025

(Another greedy algorithm)

Goal: Encode items ("a", "b", "c", etc) in binary.
We know the frequency of each item
Want the smallest encodings given the frequencies.

text sample: b a a c b d a b b a a a b d . . . .

f

120	14	3	6	1	4	2
a	b	c	d	e	g	h

$\Sigma = 150$

We could give them all 3 bits. → 450 bits for message.

Or we could give a the encoding "1" and all others get 3 bits

Cost of encoding = $\sum_{\text{elt } x} f(x) \cdot \text{bits}(x)$

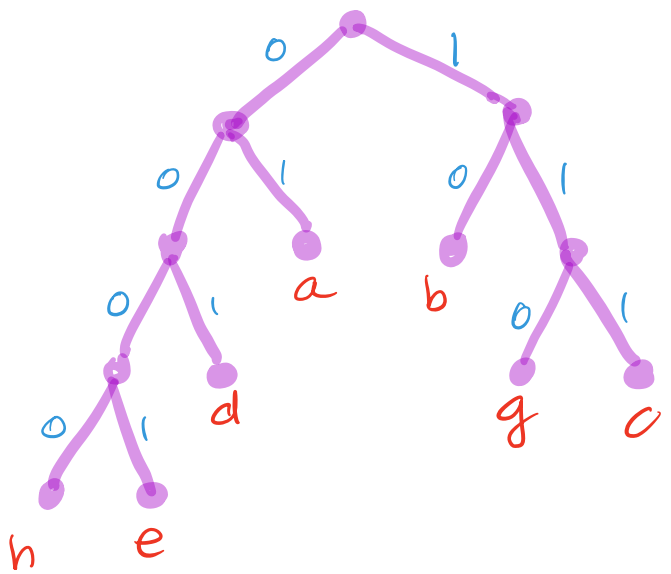
Problem: One of the other chars has an encoding that starts with 1

1 0 1 0 1 1 0 0 0 1 1 0 . . . . is encoded message.

Goal: Want prefix-free encoding of elements that optimizes encoded message length given the known element frequencies

Def^n A code is a prefix code, a.k.a. prefix-free code, if no element's code is the prefix of some other element's code.

prefix-free code tree (binary alphabet)



← tree structure representing any and all prefix-free codes.

a	b	c	d	e	g	h
120	14	3	6	1	4	2
01	10	111	001	0001	110	0000
2	2	3	3	4	3	4

= 315 ?

150 char msg

⇒ 2.13 bits/char

Greed would indicate that the more frequent elements should have shortest encodings.

How do we construct a coding tree?

Huffman devised the following alg and proved it is optimal.

Alg Huffman ($F[1..n], n$)

make each element into a one element tree
with weight = its frequency.
and element at leaf

Put all the trees into a Priority Queue

While Q has ≥ 2 items (trees)
Extract 2 trees with min freq f_1 and f_2
Combine them
Insert combined tree with frequency $f_1 + f_2$

Claim: the resulting tree is Optimal for messages with those relative frequencies.

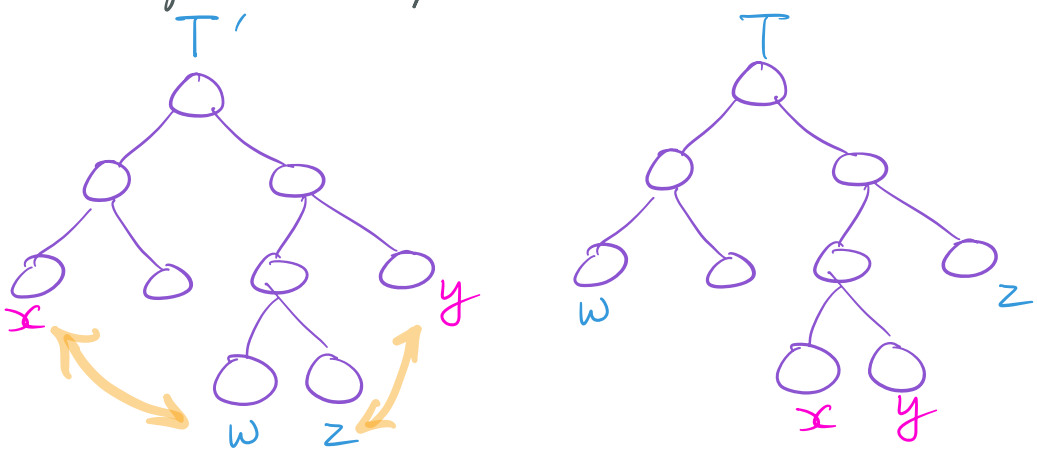
Proof: By induction on #elts ie tree size.

Base - clearly true for trees of size 1.

Let n be number of leaves in tree, $n > 1$
Ind Hyp: Claim holds $\forall$ trees with fewer leaves.

Have n elements; $\$$ x & y are least frequent, with frequencies f_x and f_y.

Let T' be any optimal tree for F. $\$$ x & y are not together as leaves at greatest depth.



Claim: T has total message length $\leq$ that of T'

Proof:

$f(w) = f(x) + \delta_1, \delta_1 \geq 0$ $\quad f(z) = f(y) + \delta_2, \delta_2 \geq 0$

$d'(w) = d(w) - \gamma_1, \gamma_1 \geq 0$ $\quad d'(z) = d(z) - \gamma_2$

before: $f_x \cdot d_x + f_w \cdot d_w$

now: $f_x \cdot (d_x + \gamma_1) + f_w \cdot (d_w - \gamma_1)$

$f_x \cdot \gamma_1 - f_w \cdot \gamma_1$

... and same with f_y and f_z.

$f_w \geq f_x$ so
result of trade
is decrease (or zero)

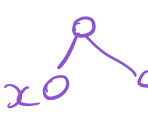
So $\exists$ an optimal tree that has x and y (least frequent elts) as siblings at greatest depth.

Let $T'_{xy} = T'$ with x and y replaced by an element XY, with frequency $f(x)+f(y)$.

Claim: T'_{xy} is optimal for its frequencies.

Proof: By contradiction.
$\nexists$ T''_{xy} that has lower cost

Let T'' be T''_{xy} but with XY replaced with



for the student to prove.

But then T'' has lower cost than T', which has optimally lowest cost
$\Rightarrow\Leftarrow$.

$\therefore$ By Ind Hyp, T''_{xy} can be constructed by Huffman's alg.
Hence Huffman's Alg produces an optimal code tree.

Complexity

- Keep element-trees in a priority queue.

$O(n)$ priority queue ops:
$2n-1$ inserts
$2n-2$ Extract-mins.

Implemented as a standard Binary Heap,
- each op is $O(\lg n)$

∴ running time is $O(n \lg n)$

Review: Priority Queue.

Operations:

Insert(el, key)

ExtractMin() // returns an elt with lowest value of key

Init()

IsEmpty() // returns true if contains no elements

Heapify()

DecreaseKey(el, newKey)