Binary code: mapping from some alphabet to strings of 0s & 1s.

one is prefix free if no code word is a prefix of any other

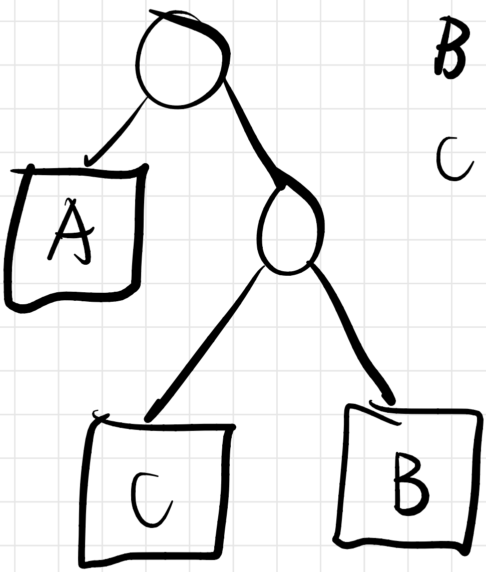
Ex: 7-bit ASCII is prefix free

Morse code: characters to sequences of dots & dashes

E: • (0)   S: ••• (000)

Can visualize prefix free codes as a binary tree with characters stored in leaves

A : 0
B : 11
C : 10



not (necessarily) a binary search tree

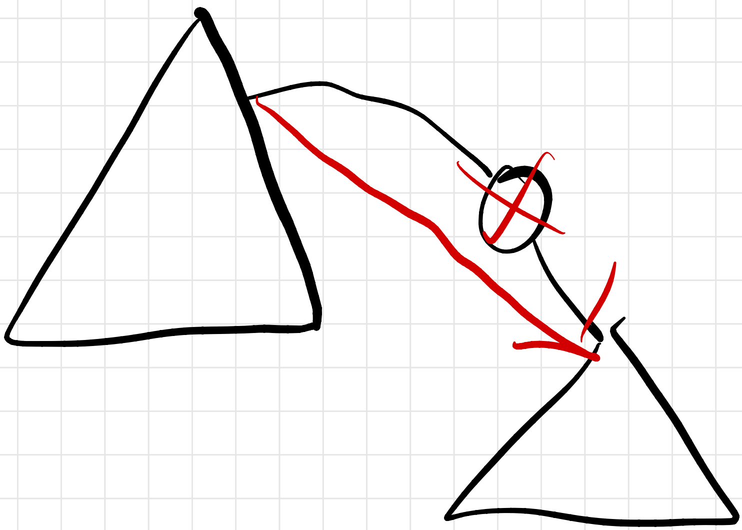
Given an array $f[1..n]$ of frequencies where $f[i]$ is ~~A~~ time character i appears.

Goal: Find a prefix free code / binary tree to minimize

$$\sum_{i=1}^{n} f[i] \cdot depth(i)$$

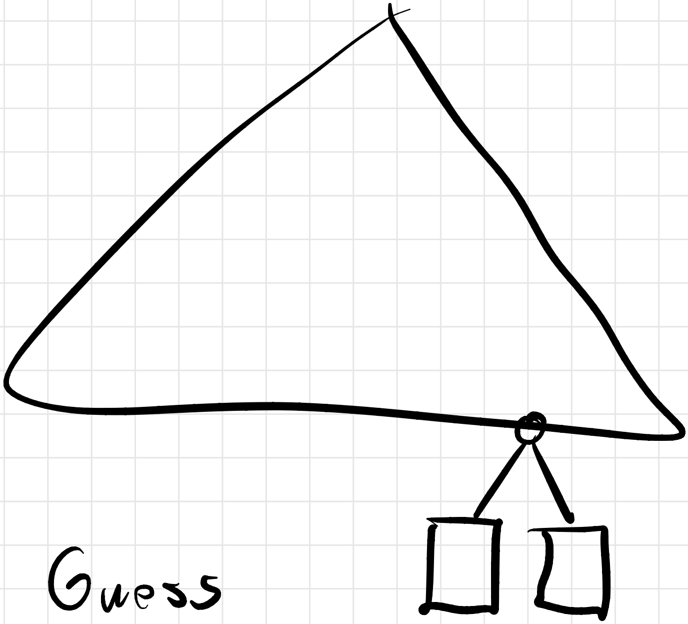
need not be a BST!
characters at leaves!

Observation: Optimal tree is <u>full</u>. Every node has 0 or 2 children.



=> Max depth nodes are leaves with leaf siblings.

If we pluck off sibling leaves, we get a smaller, full tree!



Idea: Guess sibling leaves. Treat parent as a character with freq. = sum of leaves' frequencies & recurse.

merging

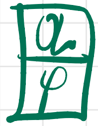
Huffman '51:

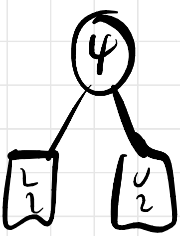
- Set least frequent two characters as siblings.
- Merge the two siblings into a single character & recurse.

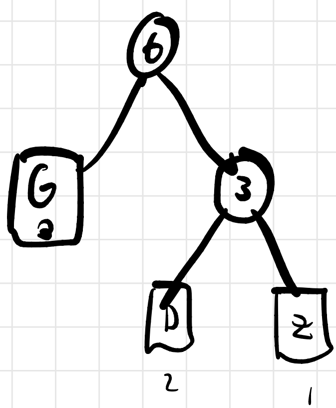
(if $n=1$ just place the root $\equiv$ use empty codeword)

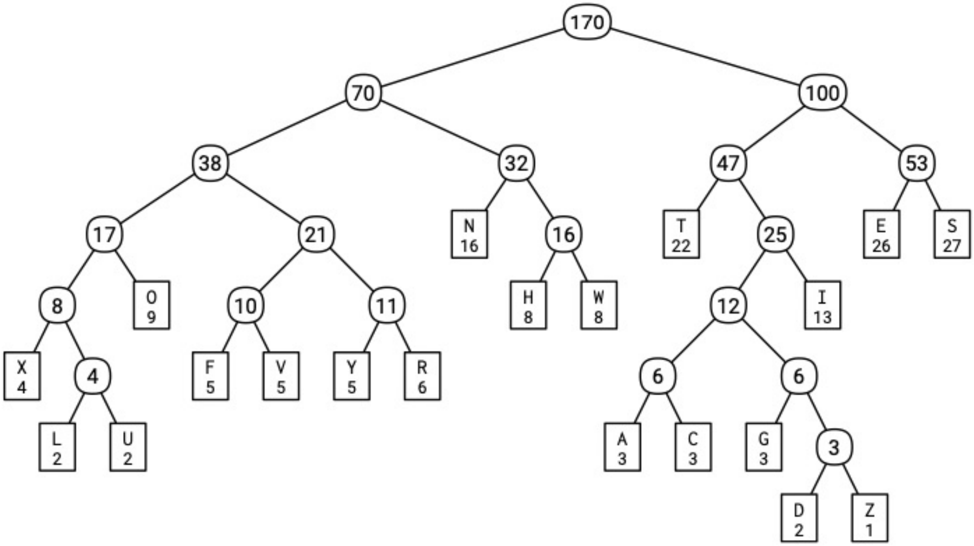
THISSENTENCECONTAINSTHREEASTHREECSTWODSTWENTYSIXESFIVEFST
HREEGSEIGHTHSTHIRTEENISTWOLSSIXTEENNSNINEOSSIXRSTWENTYSEV
ENSSTWENTYTWOTSTWOUSFIVEVSEIGHTWSFOURXSFIVEYSANDONLYONEZ[6]

A	C	~~D~~	E	F	G	H	I	~~L~~	N	O	R	S	T	~~U~~	V	W	X	Y	~~Z~~	DZ
3	3	~~2~~	26	5	3	8	13	~~2~~	16	9	6	27	22	~~2~~	5	8	4	5	~~1~~	3







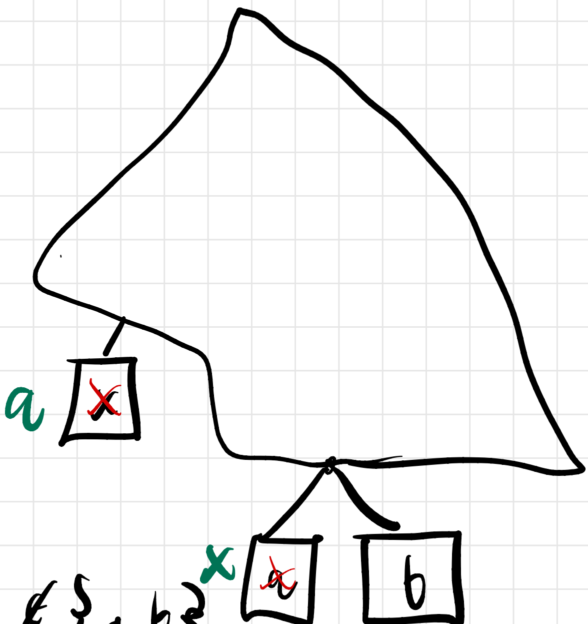


Optimal encoding has 649 bits.

Lemma: Let x & y be the two least frequent characters (break ties arbitrarily). There is an optimal code tree in which x & y are siblings (of max depth).

Proof: Let x be the least frequent char & y second. Let T be an optimal code tree & d be its depth.

T is full so its has sibling leaves a+b at depth d.



Suppose $x \notin \{a,b\}$.

Swap x + a to create tree T'.

$$\begin{aligned}
\text{cost}(T') = \text{cost}(T) &+ f[x]\cdot(\text{depth}_T(a) - \text{depth}_T(x)) \\
&- f[a]\cdot(\text{depth}_T(a) - \text{depth}_T(x)) \\
&= (f[x] - f[a])\cdot (\text{depth}_T(a) - \text{depth}_T(x)) \\
&\leq \text{cost}(T)
\end{aligned}$$

If $y \neq b$, swap y & b. Cost goes down farther.

T' is the tree we wanted.

Thm: Huffman codes are optimal prefix-free codes.

If $n=1$, then yes.

otherwise, we'll say characters 1 & 2 are least frequent.

Let T be a code tree with 1 & 2 as siblings.

Let $T' = T \setminus \{1, 2\}$.

Treat parent of 1 & 2 as a new character $n+1$ where

$$f[n+1] = f[1] + f[2]$$

T' is a code tree for $3 \ldots n+1$.

$$\mathrm{cost}(T) = \sum_{i=1}^{n} f[i] \cdot \mathrm{depth}_T(i)$$

$$= \sum_{i=3}^{n+1} f[i] \cdot \mathrm{depth}_T(i) + f[1] \cdot \mathrm{depth}_T(1) + f[2] \cdot \mathrm{depth}_T(2) - f[n+1] \cdot \mathrm{depth}_T(n+1)$$

$$= \mathrm{cost}(T') + f[1] \cdot \mathrm{depth}_T(1) + f[2] \cdot \mathrm{depth}_T(1) - f[n+1] \cdot (\mathrm{depth}_T(1) - 1)$$

$$= \text{cost}(T') + f[1] + f[2] + (f[1] + f[2] - f[n+1]) \cdot (\text{depth}_T(1) - 1)$$

$$= \text{cost}(T') + f[1] + f[2]$$

Best T with siblings 1 & 2 found by minimizing cost(T').

Gives optimal answer by prev. lemma.

Use a priority queue with two operations

Insert(e, k): insert object e with key (priority) k

ExtractMin: Removes & returns element of smallest key.

Builds arrays

$L[1..2n-1]$: left children

$R[1..2n-1]$: right children

$P[1..2n-1]$: parents

root gets label $2n-1$

```
BuildHuffman(f[1 .. n]):
    for i ← 1 to n
        L[i] ← 0; R[i] ← 0
        Insert(i, f[i])
    for i ← n + 1 to 2n − 1
        x ← ExtractMin()        ⟨⟨Find two rarest characters⟩⟩
        y ← ExtractMin()
        f[i] ← f[x] + f[y]      ⟨⟨Merge into a new character⟩⟩
        Insert(i, f[i])
        L[i] ← x; P[x] ← i      ⟨⟨Update tree pointers⟩⟩
        R[i] ← y; P[y] ← i
    P[2n − 1] ← 0
```

✓ leaves have no children

↑ root has no parent

$O(n) \cdot O(\log n) = O(n \log n)$ time

```
HuffmanEncode(A[1..k]):
    m ← 1
    for i ← 1 to k
        HuffmanEncodeOne(A[i])

HuffmanEncodeOne(x):
    if x < 2n − 1
        HuffmanEncodeOne(P[x])
        if x = L[P[x]]
            B[m] ← 0
        else
            B[m] ← 1
        m ← m + 1
```

```
HuffmanDecode(B[1..m]):
    k ← 1
    v ← 2n − 1
    for i ← 1 to m
        if B[i] = 0
            v ← L[v]
        else
            v ← R[v]

        if L[v] = 0
            A[k] ← v
            k ← k + 1
            v ← 2n − 1
```

$O(m)$ time each