1. Huffman Codes

- How does compression work? Lossy (JPG, MPEG, MP3)
  vs. Lossless (ZIP, RAR)
- Huffman (1952)
  * grad student back then, did this instead of preparing for finals.
    Improved on his professor's work
- Normally, how do we represent a text?
  ASCII. 8 bits per symbol (char). Includes many symbols $, ~, ?, ^, #

  What happens if we only use ≤32 symbols (a-z, space, comma, ...)?

  Now we can do 5 bits per symbol. Fixed length encoding.
- Observation: Some symbols are used more often than others!
  e, a, t, i vs. w, x, z

  "Shortcut": frequent symbols have shorter encodings

  e.g. e ↦ 1

  a ↦ 01 What is "0101"? "aa" or "be"?

  b ↦ 010

  To avoid such ambiguity, we use prefix code (prefix-free code).

Def: A prefix code for alphabet S is a function $c: S \to \{0,1\}^*$, s.t. $\forall x, y \in S$, $x \neq y$, $c(x)$ is not a prefix of $c(y)$.

The example above is not a prefix code!

e.g. $a \mapsto 11$

$e \mapsto 01$

$k \mapsto 001$

$r \mapsto 10$

$u \mapsto 0000$

1001000001

r e u k

Suppose we have a file of 1 billion symbols, with the following frequencies:

$f_a = 0.32 \quad f_e = 0.25 \quad f_k = 0.2 \quad f_r = 0.18 \quad f_u = 0.05$

$\#\text{ bits/symbol} = 2f_a + 2f_e + 3f_k + 2f_r + 4f_u = 2.3 \text{ bits/symbol}$

So the file can be represented with 2.3 billion bits.

Def: Average Bits per Letter for prefix code c is

$$ABL(c) = \sum_{x \in S} f_x \cdot |c(x)|$$

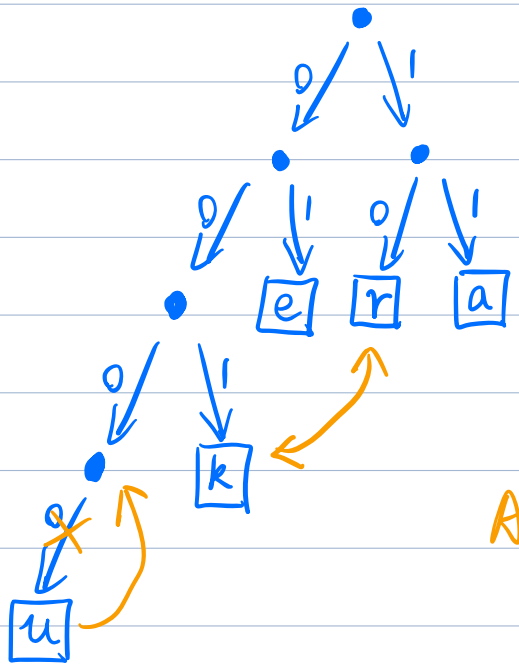
Can we do better? Lower ABL?

Yes! ① $u \mapsto 0000$ ⇒ ABL(c) = 2.25

② swap encoding of r and k

⇒ ABL(c) = 2.23

We can model prefix codes as trees.



Only leaves have labels

1011001 0000

"r a k u"

$$ABL(T) = \sum_{x \in S} f_x \cdot depth_T(x)$$

Claim: The binary tree corresponding to an optimal prefix code is **full**.

↑ each node has either 0 or 2 children.

↑ leaf nodes

Proof: Assume towards contradiction T is the binary tree of an optimal prefix code but is not full.

$\Rightarrow \exists$ some node u with a single child

Then, we can replace u by its child. This decreases some depths, and leaves the others unaffected. Hence ABL decreases, contradicts with T is optimal. ⁂

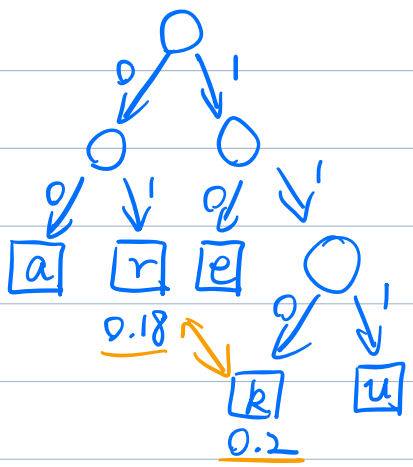
Greedy Attempt #1: [Shannon-Fano/Fano 1949]

↑ Professor of David Huffman

Split S into S_1 and S_2 with almost equal frequencies, then recursively build tree for S_1 and S_2.

$f_a = 0.32$ $f_e = 0.25$ $f_k = 0.2$ $f_r = 0.18$ $f_u = 0.05$

$f_a + f_r = f_e + f_k + f_u = 0.5 \Rightarrow S_1 = \{a, r\},\ S_2 = \{e, k, u\}$



2.25 bits / symbol

Optimal ABL is 2.23!

So this is sub-optimal.

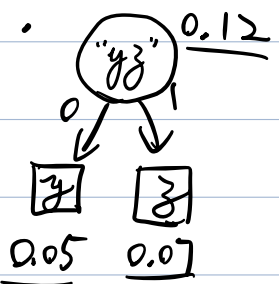
Huffman Encoding

Observation/Intuitions:

1. Lowest frequency symbols should be at the lowest/deepest level
2. The lowest/deepest level always contains at least 2 symbols
3. The order of symbols in a level does not matter
   (we don't care "10110" vs "01101")

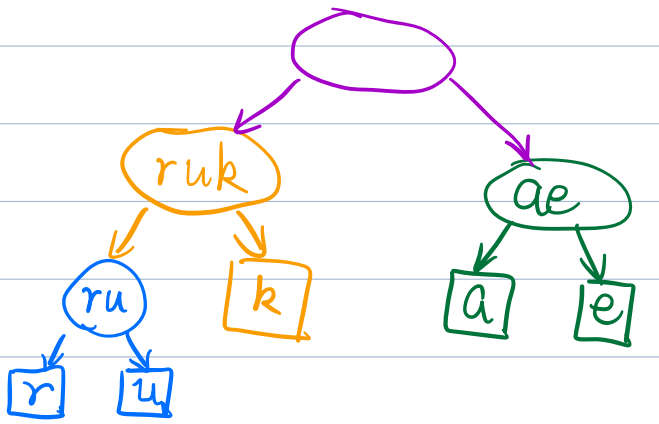
Huffman's Greedy Approach [1952] (build the tree bottom up)

1. Make a node with 2 children leaf nodes for the two lowest freq. symbols y & z.
2. Replace y & z with the "metasymbol" "yz". its freq. is the sum of freq. of y and z.
3. Rinse & Repeat until we have a single metasymbol, it'll be the root of the huffman tree.



e.g. $f_a = 0.32$ $f_e = 0.25$ $f_k = 0.2$ $f_r = 0.18$ $f_u = 0.05$

$f_{ae} = 0.57$ $f_{ru} = 0.23$ $f_{ruk} = 0.43$



1.
2.
3.
4. , done!

Pseudocode:

```
Huffman(S):
1.    If |S|=2:
2.        Return tree with root and two leaves
3.    Let y and z be lowest freq. symbols in S
4.    S'=S
5.    Remove y and z from S'
6.    Insert new symbol w in S' w/ f_w = f_y + f_z
7.    T' = Huffman(S')
8.    T = add children y and z to leaf w in T'
9.    Return T
```