Huffman Encoding

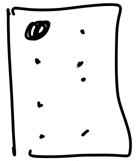
Learning Goals

- Describe "Binary code," "Prefix free", "Average letter length"
- Explain connection between binary codes + trees
- Describe Huffman's alg.
- Analyze runtime of Huffman's alg
- Describe impact of data structures alg runtime
- Prove correctness of Huffman's alg

Binary Codes

ex: $\Sigma = \{a, b, c, d, \ldots, z\}$

def: Given an alphabet Σ, a binary code is a function $f: \Sigma \to \{0,1\}^*$

ex: Braille  10101011, ASCII, Morse Code
· — — · —

Suppose you have a message where the letter "a" occurs 50% of the time, "b" 30%, and "c" 20%. Which is the best binary encoding of $\Sigma = \{a, b, c\}$?

A): $f(a) = 00$, $f(b) = 01$, $f(c) = 10$

B) $f(a) = 0$, $f(b) = 1$, $f(c) = 01$

C) $f(a) = 0$, $f(b) = 10$, $f(c) = 11$

A): $f(a) = 00$
$f(b) = 01$
$f(c) = 10$

Doesn't take advantage of differing occurance rates

B) $f(a) = 0$
$f(b) = 1$
$f(c) = 01$

Ambiguous for decoding

$01 \to c$?
$01 \to ab$?

$\boxed{C)}$ $f(a) = 0$
$f(b) = 10$
$f(c) = 11$

Not ambiguous, average bits per letter is small

Average letter length: $L(f) = \sum_{i \in \Sigma} |f(i)| \cdot p(i)$

$|f(i)|$: # of bits in $f(i)$

ex: $\boxed{C}$ $|f(a)|\,p(a) + |f(b)| \cdot p(b) + |f(c)| \cdot p(c)$

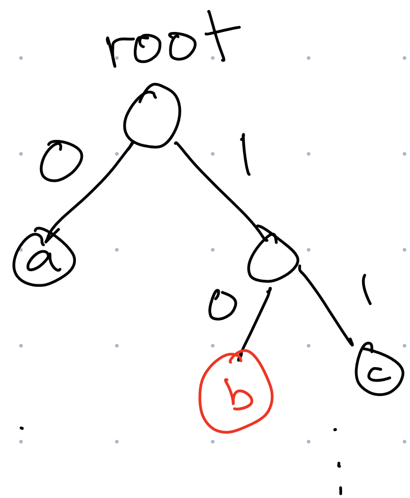
$= 1 \cdot 0.5 + 2 \cdot 0.3 + 2 \cdot 0.2 = 1.5$

Binary Trees & Binary Codes

$f(a)=0$ $f(a)=0$

$f(b)=01$ $f(b)=10$

$f(c)=11$ $f(c)=11$

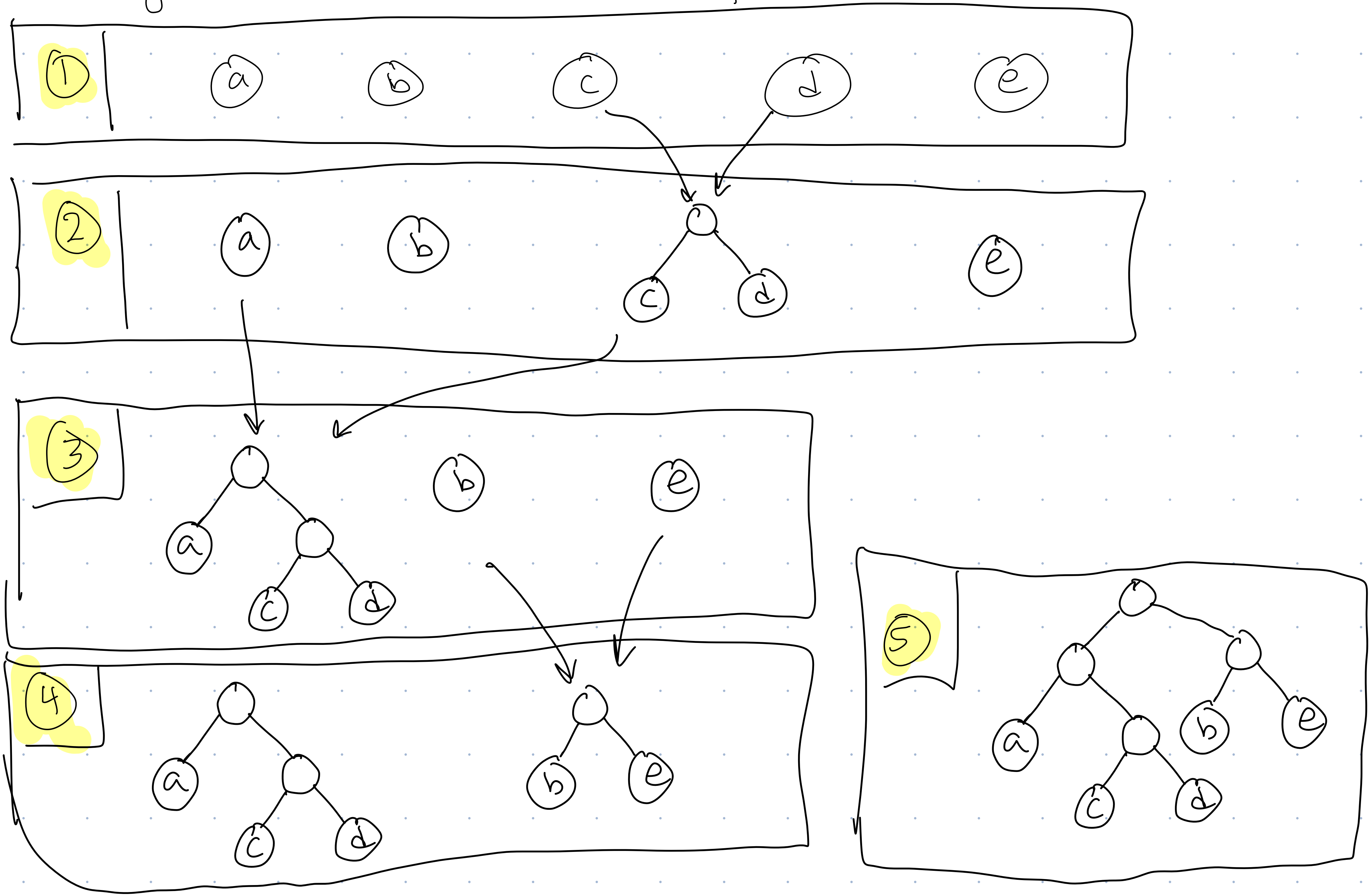


There is ambiguity for decoding if multiple letters lie on same path

$\rightarrow$ $\underbrace{0}_{a}\underbrace{10}_{b}\underbrace{11}_{c}\underbrace{0}_{a}1\ldots$

<u>def</u>: A code is "prefix free" if all letters are at leaves in corresponding binary tree.

Merge Trees to Create Prefix Free Codes



Optimal Binary Encoding Problem

Input: Σ (alphabets of symbols)

$p : \Sigma \to \mathbb{R}$ (probabilities / frequency for each symbol)

Output: $f : \Sigma \to \{0,1\}^*$ s.t.

- f is prefix free
- minimize average letter length

Huffman's Algorithm

For each $i \in \Sigma$:

- Create a tree with one node, label i
- Give tree weight $p(i)$

While there is more than one tree:

- Merge two trees with smallest weight
- Set weight of merged tree to be sum of weights.

---

i	$p(i)$
a	.3
b	.25
c	.2
d	.15
e	.1

- Use Huffman's algorithm to create a binary code
- What is the average letter length of your code
- What is the runtime of Huffman's in terms of $|\Sigma| = n$? Ideas to improve?
- Why greedy?