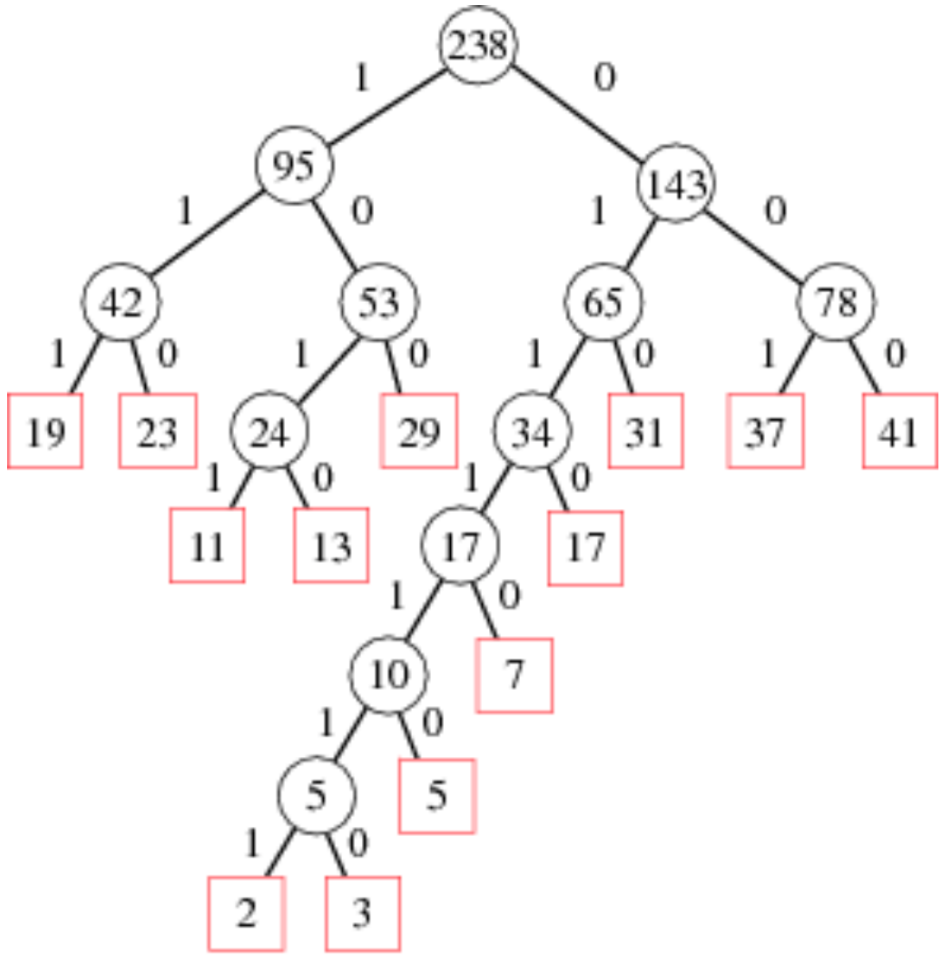
CSCI 3104: Algorithms

Lecture 11: Greedy Algorithms and Huffman Codes

Rachel Cox

Department of Computer Science



Source

What will we learn today?

- ❑ Greedy Algorithms - continued
- ❑ Huffman Codes

Intro to Algorithms, CLRS:
Sections 16.3

Greedy Algorithms

Example: Let S be a set of n objects and let $w: S \to [0, \infty)$ be a weight functions that assigns a unique weight to each object in the set. That is, you may assume for all $a, b \in S$ that $w(a) \neq w(b)$ if a is not the same object as b. Fix a non-negative integer k such that $k \leq n$ and consider the problem of finding a subset $T \subseteq S$ such that $|T| = k$ and the sum

$$w(T) \coloneqq \sum_{t \in T} w(t)$$

e.g. $S = \{3, 4, 5\}$ $n = 3$

$T = \{4, 5\}$ $k = 2$

is maximized. That is, you are trying to find the set of k objects with the maximum total sum of weights values.

One greedy approach to solve this problem is to start with T being empty and then repeatedly select the object with the largest weight value in S, add this object to T, remove this object from S and stop when T contains exactly k objects.

Give an exchange argument to show that this algorithm will always find an optimal solution to the problem. That is, fix an arbitrary optimal solution T_{opt} and argue that it can't be better than the solution T_{greedy} found by the above greedy algorithm.

Hint: Consider what would happen if there was an object $t \in T_{opt}$ but $t \notin T_{greedy}$

Greedy Algorithms

optimal means the maximized weight of k objects.

W(s): weight of s

set of k objects.

Solution: Let T_{greedy} be the solution to the greedy algorithm.

Assume for a contradiction, that the optimal solution, $T_{optimal}$, is different than the greedy solution.

[• Note, since T_{greedy} and $T_{optimal}$ are both solutions, we can assume they both have k objects.]

There is at least one $t \in S$ such that $t \in T_{optimal}$ but $t \notin T_{greedy}$

⇒ since $t \notin T_{greedy}$, there must be some $s \in T_{greedy}$ but not in $T_{optimal}$ because both solutions must be of size k.

Note that $w(s) > w(t)$ because $s \in T_{greedy}$ but $t \notin T_{greedy}$ because the greedy solution consists of the k objects of largest weight and all weights are unique.

Let the total weight of $T_{optimal}$ be W. i.e. $w(T_{optimal}) = W$

Let $w(T_{optimal}) = W$

$$w(T_{optimal} \setminus t) = W - w(t)$$

Consider a new solution T' where

$T' = (T_{optimal} \cup s) \setminus t$ Note: T' has k objects

$\Rightarrow w(T') = W + w(s) - w(t)$

Note that $w(s) > w(t)$

$\Rightarrow w(s) - w(t) > 0$

$\Rightarrow w(T') = W + (w(s) - w(t))$

$> W$

$= w(T_{optimal})$

We've just shown $w(T') > w(T_{optimal})$

- We've contradicted our assumption that $T_{optimal}$ maximized the weight.

Because we've reached a contradiction, our original assumption is false

$\Rightarrow$ The greedy solution and the optimal solution are the same.

$\Rightarrow$ The greedy solution is optimal, as desired ■

Greedy Algorithms

optimal here: fewest number of coins.

Example: Consider the making change problem, where we wish to represent n cents using the fewest number of coins. Suppose we only have the coin denominations of 1, 10, and 15 cents. We have at our disposal an unlimited number of each of the coins.

Consider a greedy algorithm for making change with n cents that at each step takes the largest value coin c which is at most n and then recursively makes change for $n - c$ cents until we get down to 0 cents.

Show that this greedy algorithm is not optimal by finding an n such that the greedy algorithm makes change for n cents with a number of coins that is strictly greater than the minimum possible. Write down your chosen n, explain the choices made by the greedy algorithm, the coins used in the eventual greedy solution, and exhibit a solution with fewer coins.

Greedy Algorithms

Solution: Let $n = 20$ cents

we have the following coins: $\{1, 10, 15\}$

cents remaining $n = 20$	
5	choose a 15 cent coin
4	choose a 1 cent coin
3	choose a 1 cent coin
2	choose a 1 cent coin
1	choose a 1 cent coin
	choose a 1 cent coin

The greedy solution is 1 - 15 cent coin and 5 - 1 cent coins

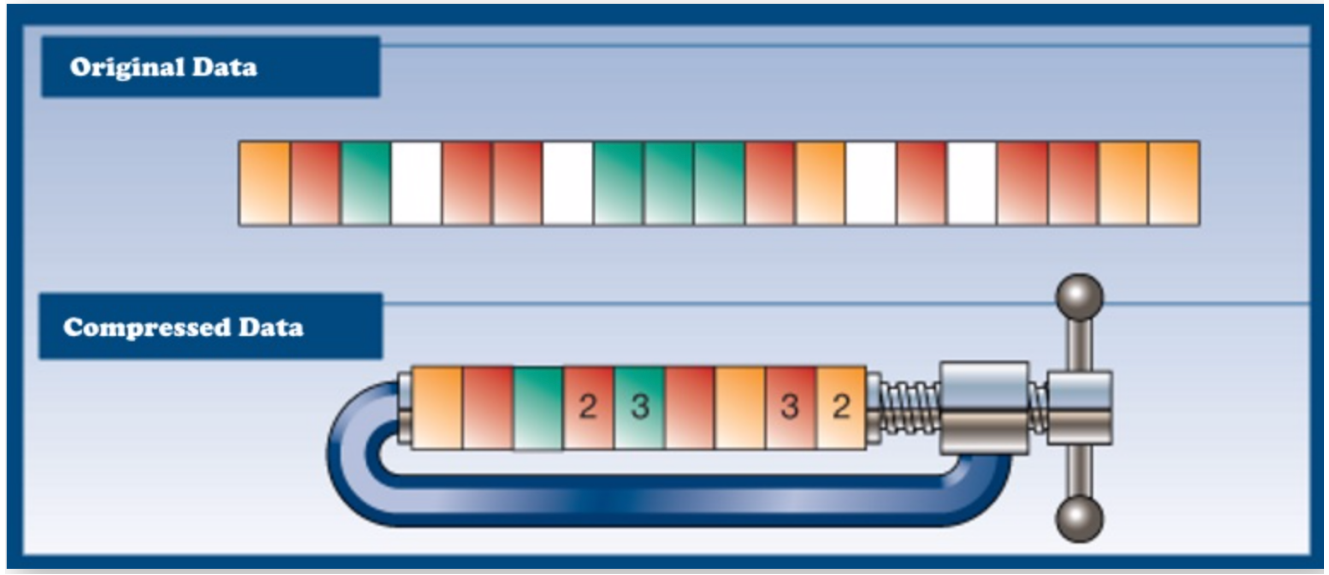
6 coins

This is not optimal because we could have given 20 cents in change with two 10 cent coins.

optimal solution = 2 coins

Huffman Codes

Data compression: Suppose we need to transmit data digitally as a bit string (bit strings are composed of 0's and 1's)



- We need/desire the minimum size bit string that preserves data integrity

- Idea: Use a Greedy Algorithm

Example: What if we want to encode the English alphabet?

We've got 26 letters.

A-1 C-3 E-5

B-2 D-4 ⋮

Z-26

How many bits would be required?

$n = 26$ — 0
13 — 1
6 — 0
3 — 1
1 — 1

$26 = (11010)_2$

5 bits

Huffman Codes

Huffman Codes:

- Used to compress data
- Data is considered to be a sequence of characters

- Greedy Algorithm - uses data on how often a character appears in the string, its frequency, to build an optimal way of representing each character as a binary string.

- Instead of using a fixed number of bits to represent each character, we can save space by using a variable number of bits

  Savings: 20% – 90%

Huffman Codes

Variable Length Encoding: Use fewer bits for frequently used characters and more bits for less frequently used characters.

e.g. e, t, a, o, i are more common and get fewer bits than *q, z, j*

Vs.

Fixed Length Code: Given the number of characters, we use a fixed number of bits to represent all characters.

Z: 11010 , A: 00001

	d	o	n	k	e	y
Frequency (in thousands)	10	20	25	17	23	5
Fixed-Length Codeword	001	101	100	011	010	110
Variable-Length Codeword	1101	00	10	111	01	1100

Huffman Codes

	d	o	n	k	e	y
Frequency (in thousands)	10	20	25	17	23	5
Fixed-Length Codeword	001	101	100	011	010	110
Variable-Length Codeword	1101	00	10	111	01	1100

← Derived later in this lecture.

- In a fixed length code, we represent each character with 3 bits

$\Rightarrow$ 300,000 bits

- In a variable length code, we have 3 letters with 2 bits, 2 letters with 4 bits, and 1 letter with 3 bits.

$$\Rightarrow (10 \cdot 4 + 20 \cdot 2 + 25 \cdot 2 + 17 \cdot 3 + 23 \cdot 2 + 5 \cdot 4) * 1000$$

$$= 247{,}000 \text{ bits}$$

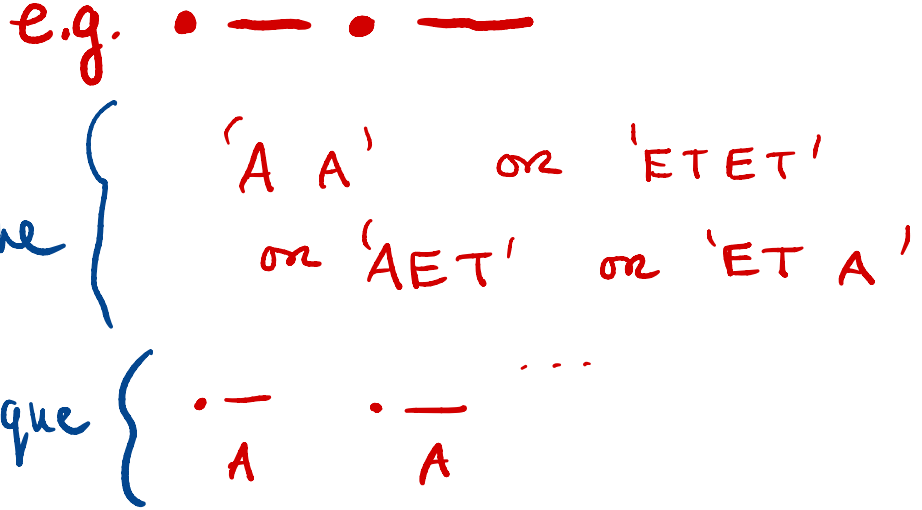
$$\text{Savings} \approx \frac{300{,}000 - 247{,}000}{300{,}000} \approx 18\%$$

Huffman Codes

Morse Code: comprised of dots and dashes

Frequent characters are encoded with short strings

Example:
e: dot,
t: dash,
a: dot-dash



A •–	J •–––	S •••
B –•••	K –•–	T –
C –•–•	L •–••	U ••–
D –••	M ––	V •••–
E •	N –•	W •––
F ••–•	O –––	X –••–
G ––•	P •––•	Y –•––
H ••••	Q ––•–	Z ––••
I ••	R •–•	

Morse code with spaces allows messages to be uniquely decoded.
- without spaces, it doesn't ..

- We want an encoding where no code is a ***prefix*** for another.
- If one code is a prefix for another and that is ambiguous

Huffman Codes

Prefix Codes: For a set S of letters, a prefix code is a function γ that maps each letter $x \in S$ to a sequence of 0's and 1's such that for distinct $x, y \in S$, $\gamma(x)$ is not a prefix of $\gamma(y)$

has the prefix property that no whole code word in the system is the initial segment of any other codeword.

Example: Given γ and a bit string, decode the following from left to right:

0010000011101

c e c a b

$S = \{a, b, c, d, e\}$

$\gamma(a) = 11$

$\gamma(b) = 01$

$\gamma(c) = 001$

$\gamma(d) = 10$

$\gamma(e) = 000$

Huffman Codes

Optimal Prefix Codes: For each $x \in S$, there is a frequency f_x that is the fraction of letter in the text that are x.

For n letters, there are $n \cdot f_x$ that are the letter x.

Also, we have that $\sum_{x \in S} f_x = 1$

optimal prefix code seeks to minimize this.

Total length of encoding: suppose γ is a prefix code

$\gamma(x)$ is prefix code for x

Encoding length $= n \sum_x f_x |\gamma(x)| =$ # of bits in our string

- n: number of characters
- f_x: freq. of each character
- $|\gamma(x)|$: length of each prefix code

Huffman Codes

Relative frequencies

Example: Suppose we have $S = \{a, b, c, d, e\}$ where
$f_a = 0.32,\ f_b = 0.25,\ f_c = 0.20,\ f_d = 0.18,\ f_e = 0.05$
with $\gamma(a) = 11,\ \gamma(b) = 01,\ \gamma(c) = 001,\ \gamma(d) = 10,\ \gamma(e) = 000$

2 bits 2 bits

What is the average number of bits we would need per letter?

same computation as expected value

$$\sum_{x \in S} f_x |\gamma_x|$$

$= .32(2) + .25(2) + .20(3) + .18(2) + .05(3)$

$= .64 + .50 + .60 + .36 + .15$

$= 2.25$ bits ← Average number of bits.

Note: A fixed length code would require 3 bits per character

$\frac{2.25}{3} = 0.75$ ⇒ 25% compression

Huffman Codes

- We want to generate an encoding that is as efficient as possible.
- Let’s build a complete binary tree to generate optimal prefix codes.
- **Goal**: Construct a tree that minimizes weighted average of depth of the leaves.
- We want to minimize the depth of the highest frequency characters.

Huffman Codes

Build a binary tree T:

- Each edge in T represents a bit in the code word. Let's assign values to edges in the following way:

  Edge to left child = 0

  Edge to right child = 1

- External Nodes (leaf nodes)

  Each external node v is associated with a character.

- Internal Nodes

  Sum of frequencies of external nodes of subtree rooted at v.

Huffman Codes

Example: Find variable length codes for the following:

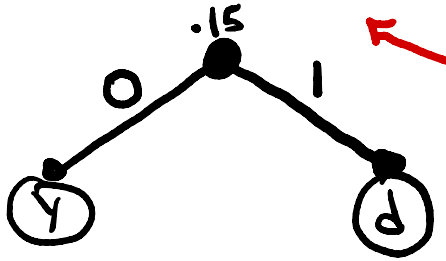
	d	o	n	k	e	y
Frequency (in thousands)	10	20	25	17	23	5
Fixed-Length Codeword	001	101	100	011	010	110
Variable-Length Codeword						

First, order the frequencies of the letters from least to greatest.

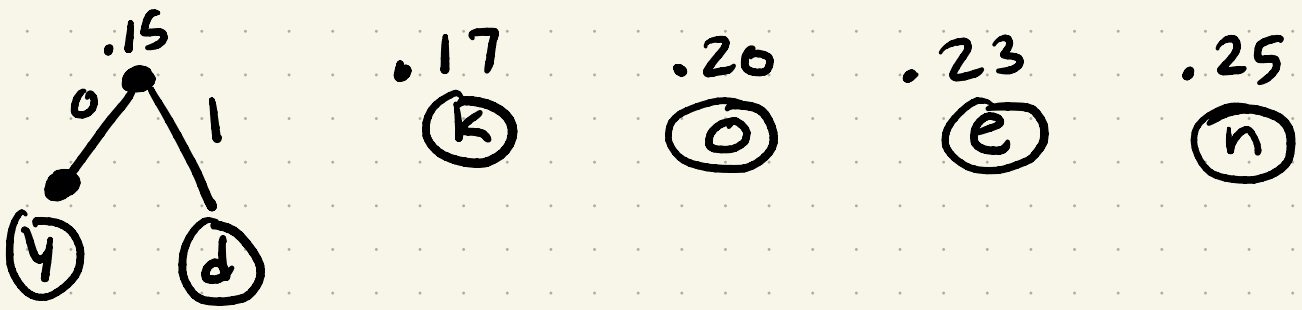
.05 (y) .10 (d) .17 (k) .20 (o) .23 (e) .25 (n)

(relative frequencies)

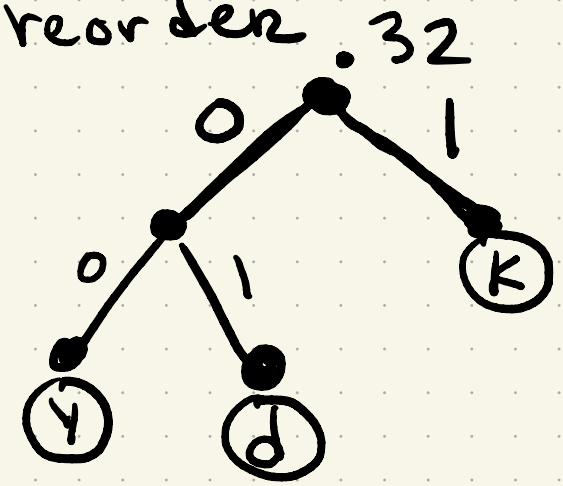
Next: Construct a tree by connecting the smallest two frequencies with a common root.



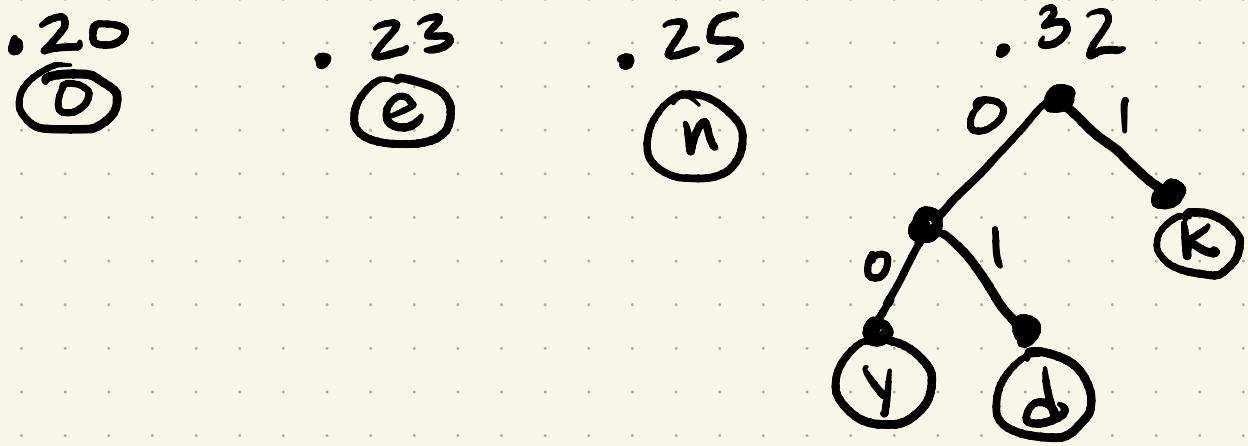
sum of the frequencies of y and d



- Next, append the two smallest subtrees and reorder

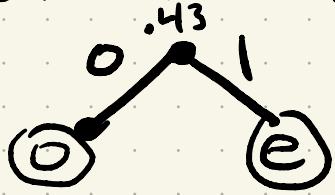


Append

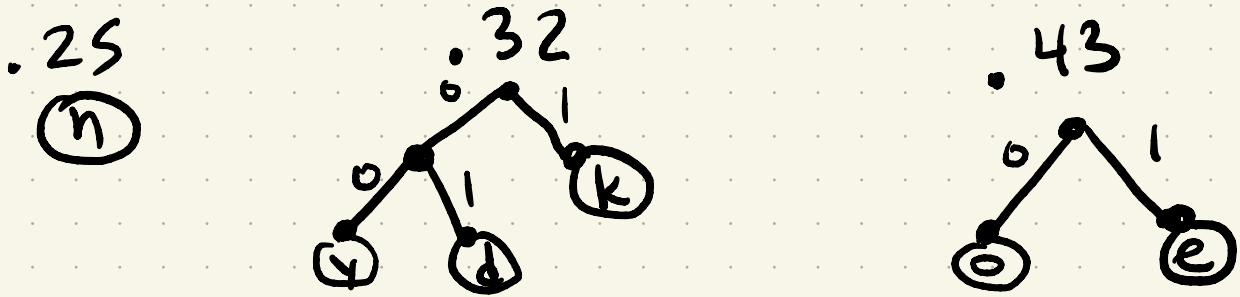


Reorder

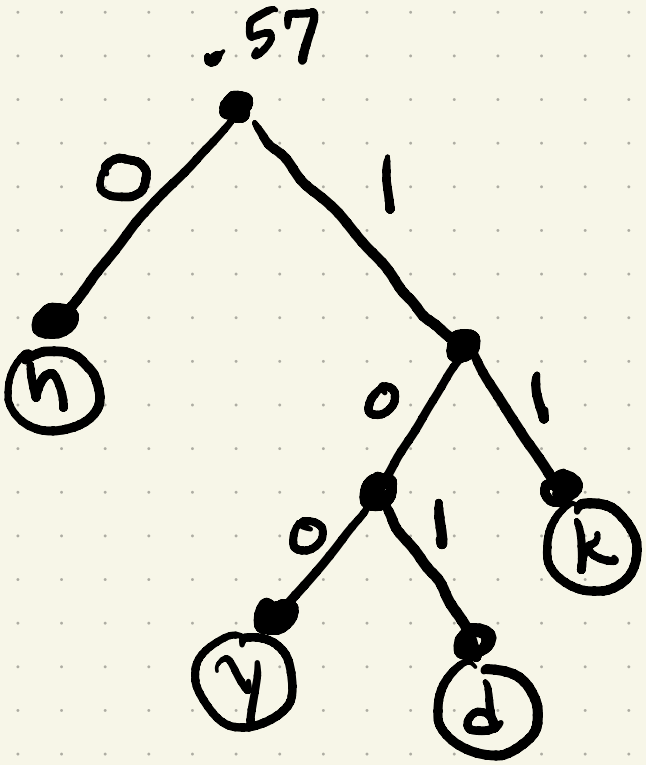
- Append the two smallest subtrees are reorder



Append



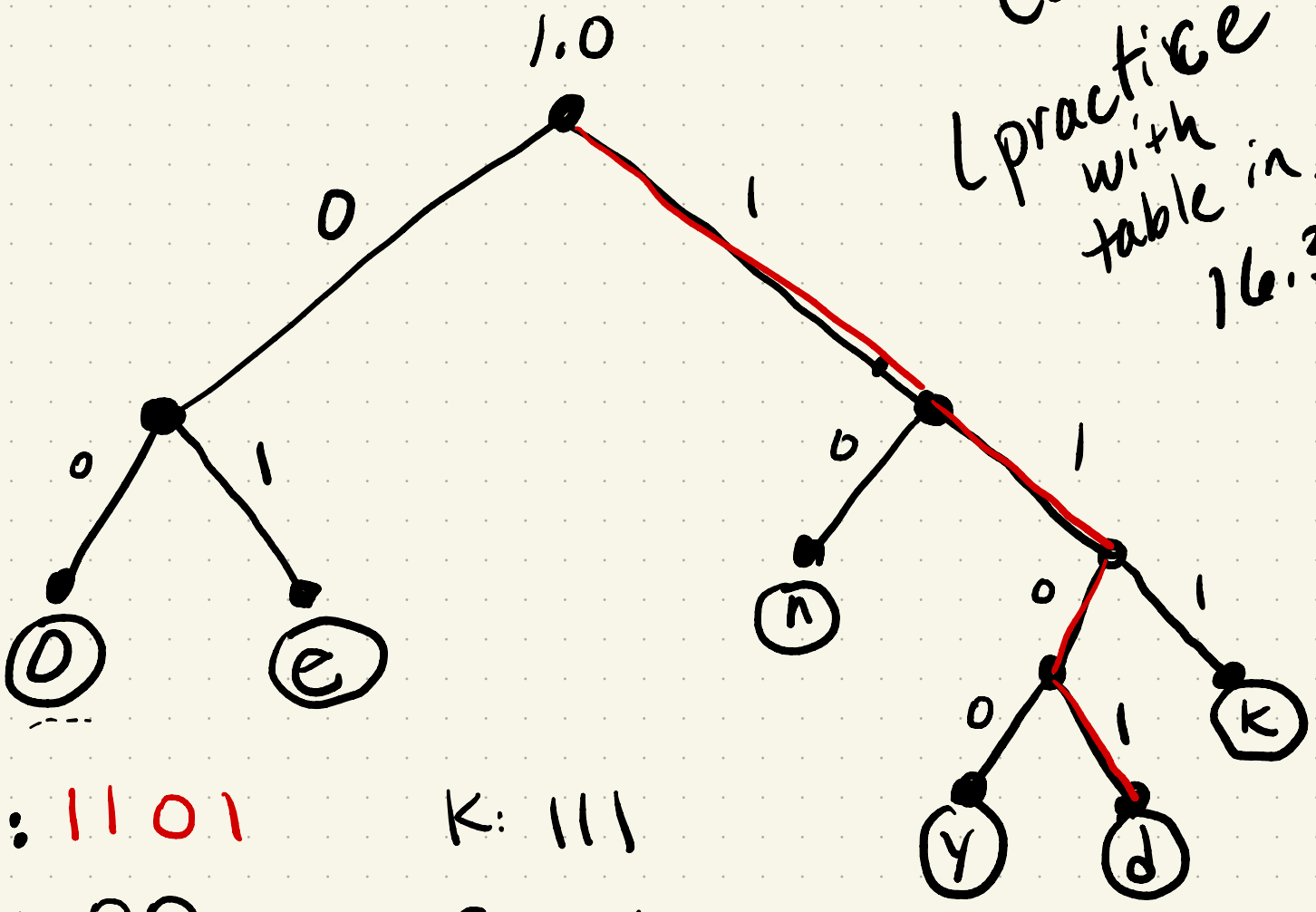
Reorder



Append

This is a Huffman code!

I practice with table in 16.3)



d: 1101

o: 00

n: 10

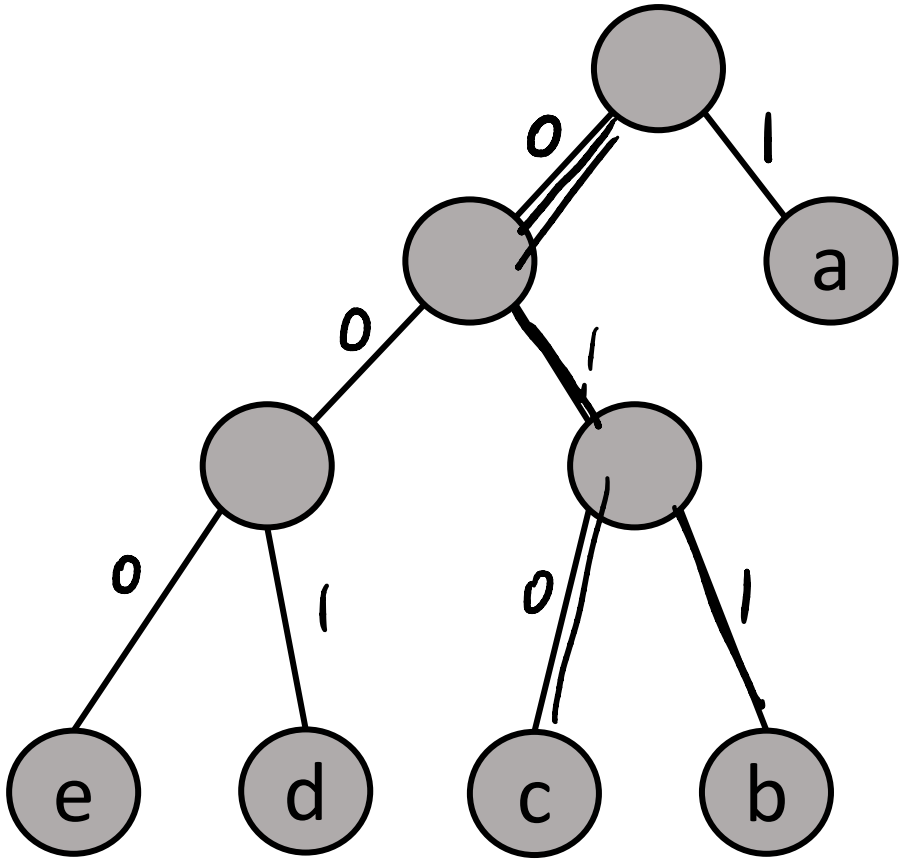
K: 111

e: 01

y: 1100

Huffman Codes

Example: What are the codes for a, b, c, d, and e given the binary tree below.



a: 1

b: 011

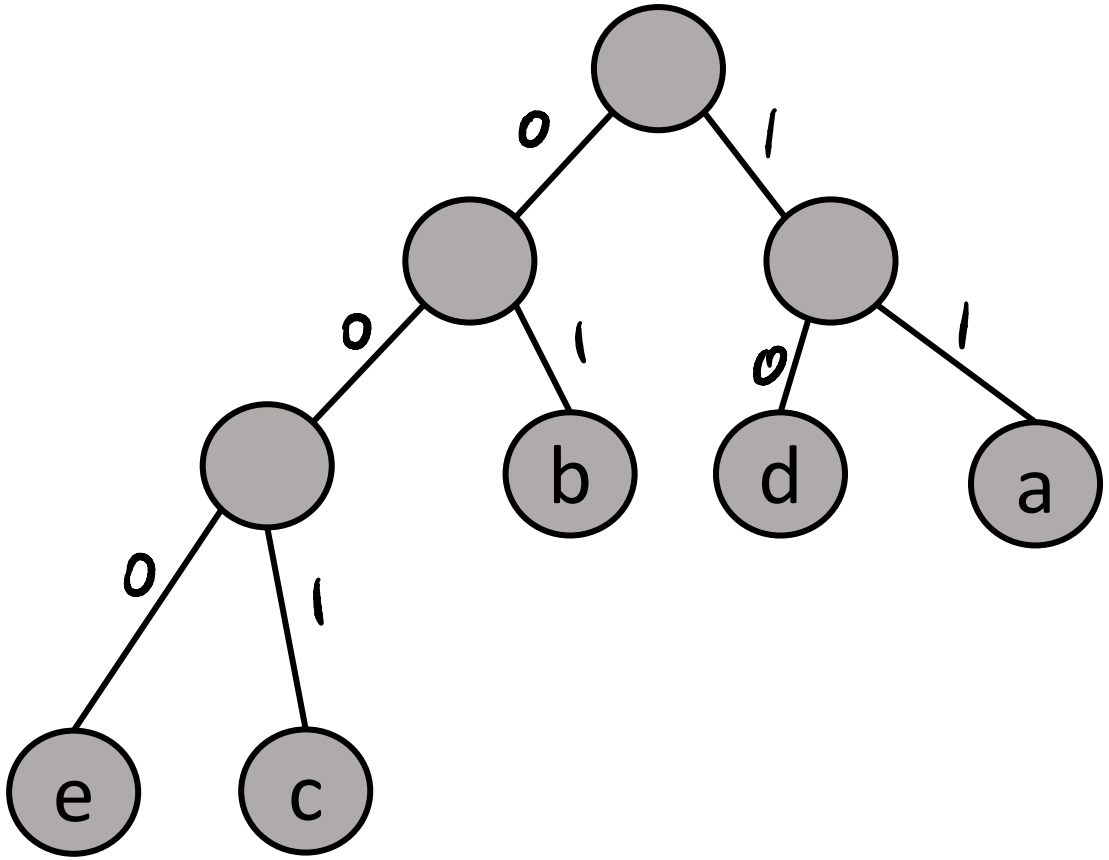
c: 010

d: 001

e: 000

Huffman Codes

Example: What are the codes for a, b, c, d, and e given the binary tree below.



a: 11

b: 01

c: 001

d: 10

e: 000

Huffman Codes

A greedy algorithm that constructs an optimal prefix code called a **Huffman code**.

Assume that C is a set of n characters and that each character $c \in C$ is an object with an attribute $c.freq$ giving its frequency

HUFFMAN(C)

$n = |C|$
$Q = C$
for $i = 1$ **to** $n - 1$
    allocate a new node z
    $z.left = x = \text{EXTRACTMIN}(Q)$
    $z.right = y = \text{EXTRACTMIN}(Q)$
    $z.freq = x.freq + y.freq$
    $\text{INSERT}(Q, z)$
return $\text{EXTRACTMIN}(Q)$

Goal: Construct a tree that minimizes the weighted average of the depth of the leaves.