Lecture 19: Data Compression and Huffman Codes

Tim LaRock

larock.t@northeastern.edu

bit.ly/cs3000syllabus

Business

Homework 5 grades posted

- Request regrades on GradeScope ASAP!

Midterm 2 approximately halfway graded

- Grades should be out by Wednesday night

Extra Credit Assignment 1 open until Sunday night

Extra Credit Assignment 2 to be released this evening and due Thursday 5PM

- Optional Greedy Algorithms and Information Theory assignment
- Points will be added to your 2nd lowest homework grade

Final Exam to be released Thursday 6PM and due Monday at Midnight

- Exam is cumulative, all topics fair game
- Review during lecture on Thursday – form for questions will go out tonight

This Week

- Today: Greedy algorithms + proof strategies
  - Data Compression, Huffman Codes, Information theory
- Tomorrow: More greedy algorithms/info theory
  - Clustering; community detection in graphs/networks
- Wednesday: Advanced topics and course wrap-up
  - If we haven’t talked about something you hoped we would, feel free to send me an email and I may be able to improvise a brief discussion!
- Thursday: Final Exam Review

Last time: Files on Tape

1	1	1	2	2	3	3	3	4	4

$\mathbb{E}[cost] = \frac{26}{4}$

We can modify the order of the files on the tape, resulting in a permutation π where $\pi(i)$ returns the index of the file in the ith block. We can then rewrite the *expected* (average) cost of accessing file *k* as

$$\mathbb{E}[cost(\pi)] = \frac{1}{n}\sum_{k=1}^{n}\sum_{i=1}^{k} L[\pi(i)]$$

Intuitively: To minimize average cost, we should store the smallest files first, otherwise we will need to unnecessarily spend time skipping the large files to read smaller ones!

2	2	4	4	1	1	1	3	3	3

$$\mathbb{E}[cost(\pi)] = \frac{2 + 4 + 7 + 10}{4} = \frac{23}{4}$$

Last time: Files on Tape

Input: A set of files labeled $1 \ldots n$ with lengths $L[i]$
Output: An ordering of the files on the tape

Repeat until all files are on the tape:

1. Find the unwritten file with minimum length (break ties arbitrarily)
2. Write that file to the tape

How can we show this is optimal?

Last time: Files on Tape

Claim: $\mathbb{E}[cost(\pi)]$ is minimized when $L[\pi(i)] \leq L[\pi(i+1)]$ for all i.

Proof:

Let $\mathrm{a} = \pi(i)$ and $b = \pi(i+1)$ and suppose $L[a] > L[b]$ for some index i.

If we swap the files a and b on the tape, then the cost of accessing a increases by $L[b]$ and the cost of accessing b decreases by $L[a]$.

Overall, the swap changes the expected cost by $\frac{L[b]-L[a]}{n}$.

This change represents an improvement because $L[b] < L[a]$.

Thus, if the files are out of length-order, we can decrease expected cost by swapping pairs to put them in order.

Key Point: If we had some other potentially optimal solution π^*, we can transform it into the optimal solution by iteratively swapping files that are out of length-order.

Data Compression and Huffman Codes

Data Compression

- How do we store strings of text compactly?

- A binary code is a mapping from $\Sigma \to \{0,1\}^*$
  - Simplest code: assign numbers $1,2,\ldots,|\Sigma|$ to each symbol, map to binary numbers of $\lceil \log_2 |\Sigma| \rceil$ bits
  - Morse Code:

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

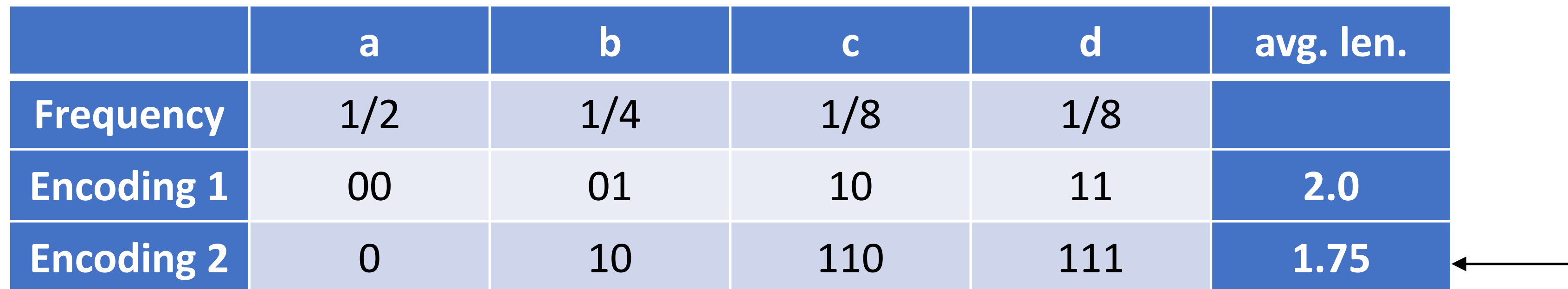
Data Compression

- Letters have uneven frequencies!
  - Want to use short encodings for frequent letters, long encodings for infrequent leters

	a	b	c	d	avg. len.
Frequency	1/2	1/4	1/8	1/8	
Encoding 1	00	01	10	11	**2.0**
Encoding 2	0	10	110	111	**1.75**

Data Compression

- Letters have uneven frequencies!
  - Want to use short encodings for frequent letters, long encodings for infrequent leters



	a	b	c	d	avg. len.
Frequency	1/2	1/4	1/8	1/8	
Encoding 1	00	01	10	11	**2.0**
Encoding 2	0	10	110	111	**1.75**

$$1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3\frac{1}{8} + 3\frac{1}{8}$$

$$= \frac{1}{2} + \frac{1}{2} + \frac{3}{4} = 1.75$$

Data Compression

- What properties would a good code have?
  - Easy to encode a string

    Encode(KTS) = – ● – – ● ● ●
  - The encoding is short on average (bits per letter given frequencies)

    $\leq$ 4 bits per letter (30 symbols max!)
  - Easy to decode a string?

    Decode(– ● – – ● ● ●) =

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 ●–●	

Prefix Free Codes

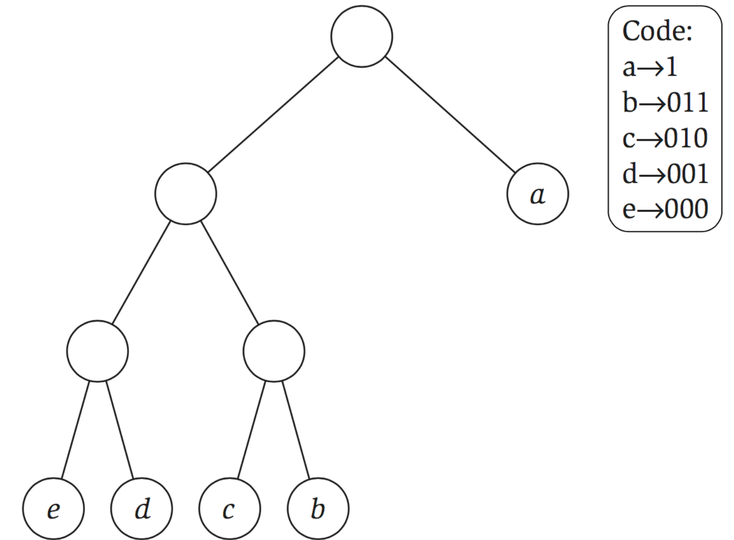
- Cannot decode if there are ambiguities
  - e.g. $\mathrm{enc}(\text{“}E\text{”})$ is a prefix of $\mathrm{enc}(\text{“}S\text{”})$

- Prefix-Free Code:
  - A binary $\mathrm{enc}\colon\ \Sigma \to \{0,1\}^*$ such that
    for every $x \neq y \in \Sigma$, $\mathrm{enc}(x)$ is not a prefix of $\mathrm{enc}(y)$

  - Any fixed-length code is prefix-free

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

Prefix Free Codes



- Can represent a prefix-free code as a binary tree
  - Left child = 0
  - Right child = 1

- Encode by going up the tree (or using a table)
  - d a b → 0 0 1 1 0 1 1
- Decode by going down the tree
  - 0 1 1 0 0 0 1 0 0 1 0 1 0 1 0 1 1 ← beadcab

Huffman Codes

- (An algorithm to find) an **optimal** prefix-free code

- **optimal** = $\min_{\text{prefix-free } T} \text{len}(T) = \sum_{i \in \Sigma} f_i \cdot \text{len}_T(i)$
  - Note, optimality depends on what you're compressing
  - H is the 8th most frequent letter in English (6.094%) but the 20th most frequent in Italian (0.636%)

	a	b	c	d
Frequency	1/2	1/4	1/8	1/8
Encoding	0	10	110	111

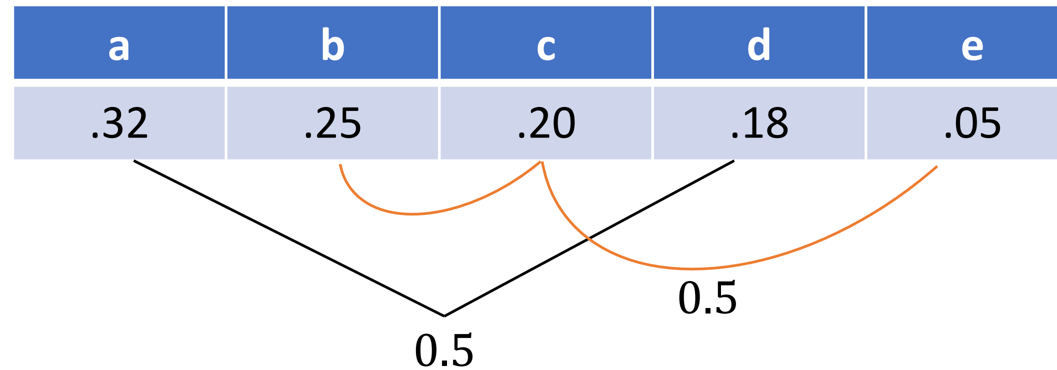
Huffman Codes

- First Try: split letters into two sets of roughly equal frequency and recurse
  - Balanced binary trees should have low depth

a	b	c	d	e
.32	.25	.20	.18	.05

Huffman Codes

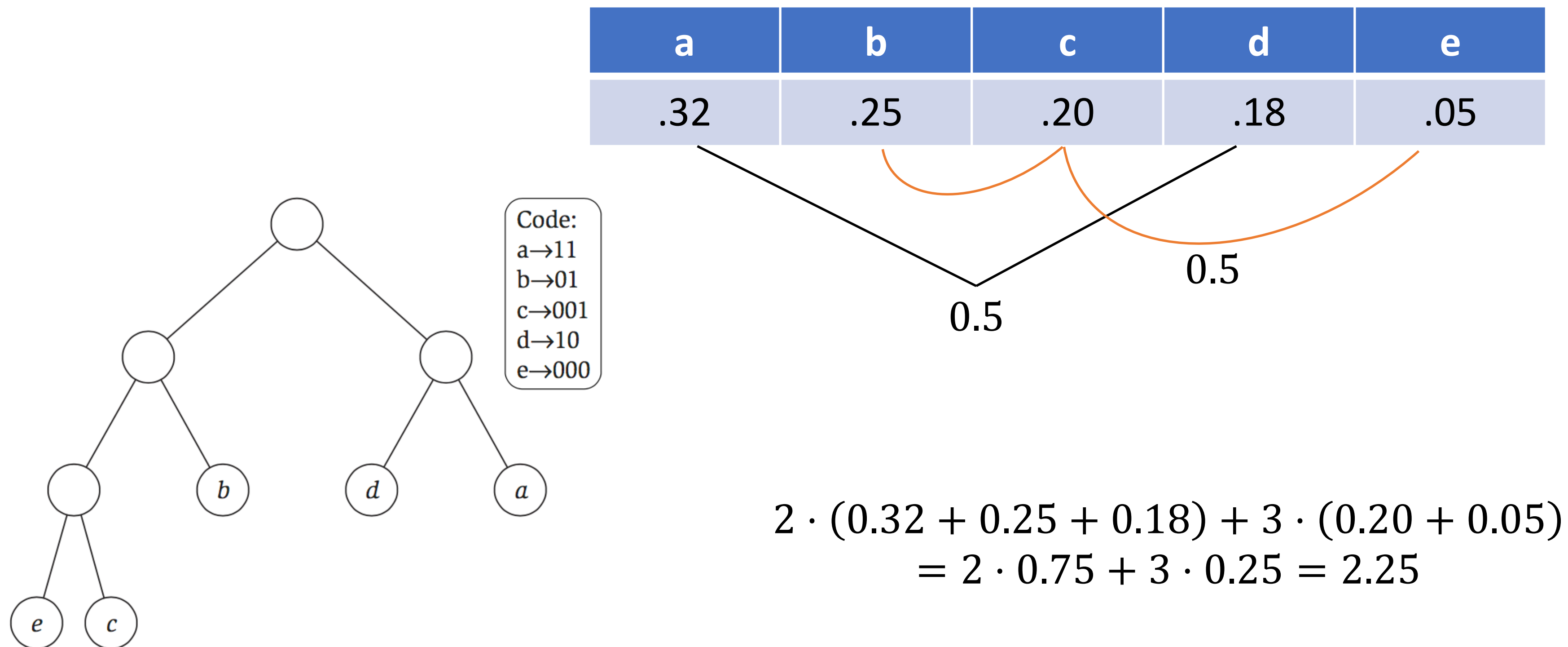
- First Try: split letters into two sets of roughly equal frequency and recurse
  - Balanced binary trees should have low depth



a	b	c	d	e
.32	.25	.20	.18	.05

Huffman Codes

- First Try: split letters into two sets of roughly equal frequency and recurse
  - Balanced binary trees should have low depth



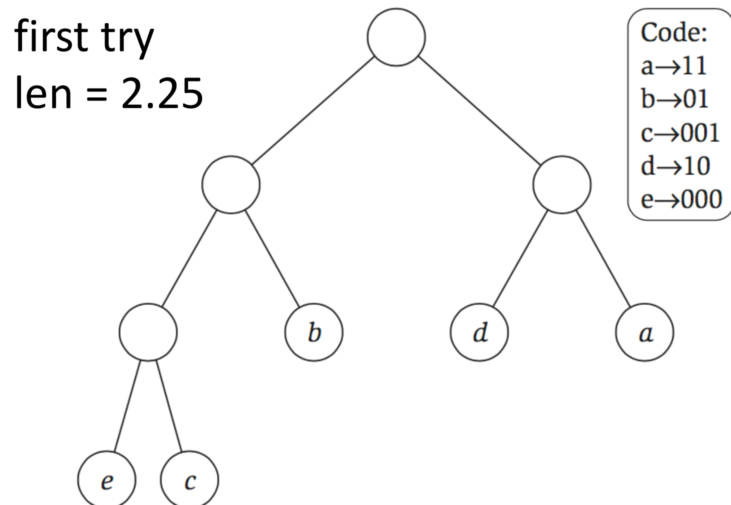
a	b	c	d	e
.32	.25	.20	.18	.05

$$2 \cdot (0.32 + 0.25 + 0.18) + 3 \cdot (0.20 + 0.05)$$
$$= 2 \cdot 0.75 + 3 \cdot 0.25 = 2.25$$

Huffman Codes

- First Try: split letters into two sets of roughly equal frequency and recurse

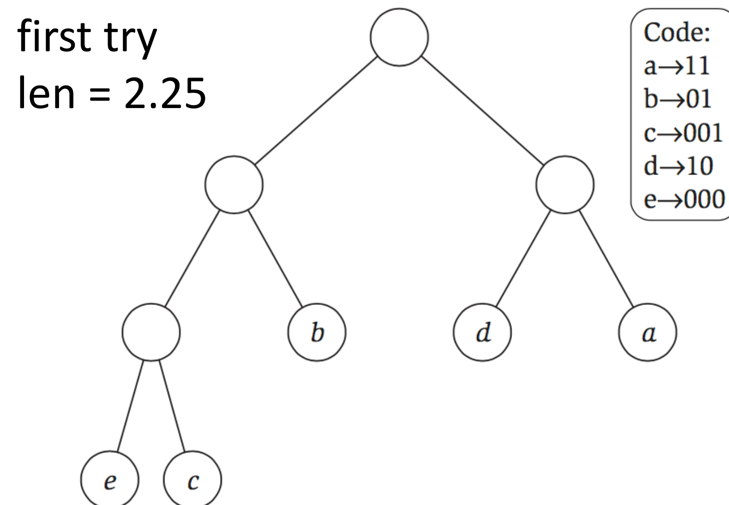
a	b	c	d	e
.32	.25	.20	.18	.05

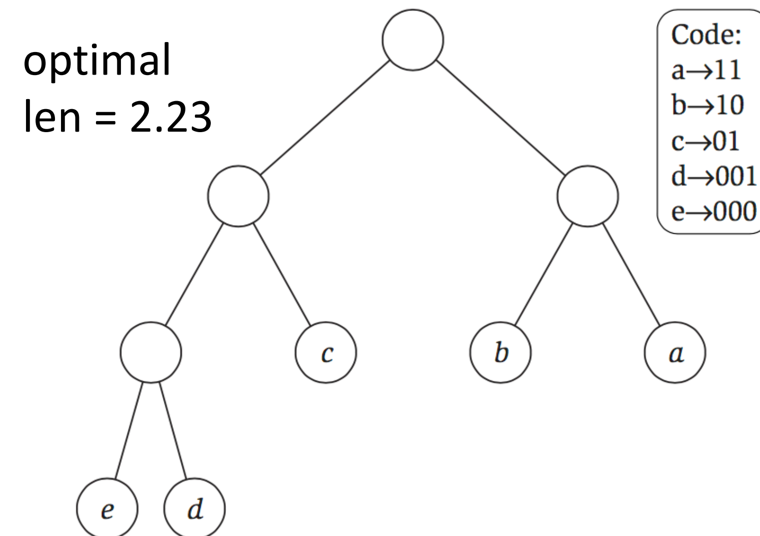


Huffman Codes

- First Try: split letters into two sets of roughly equal frequency and recurse

a	b	c	d	e
.32	.25	.20	.18	.05

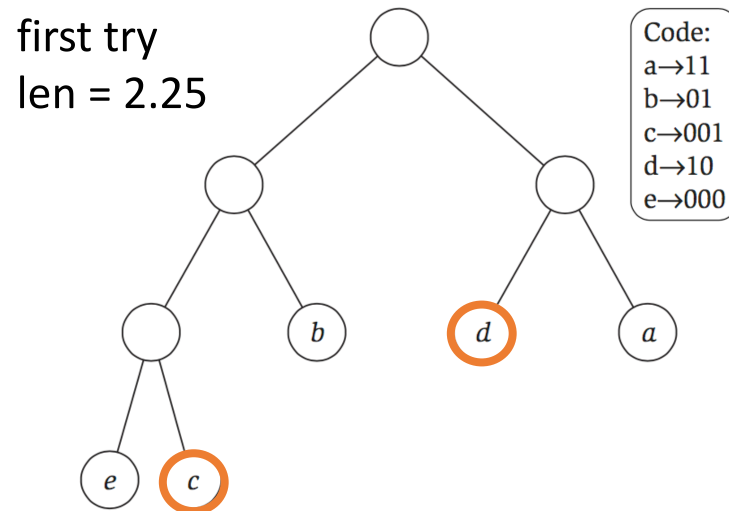


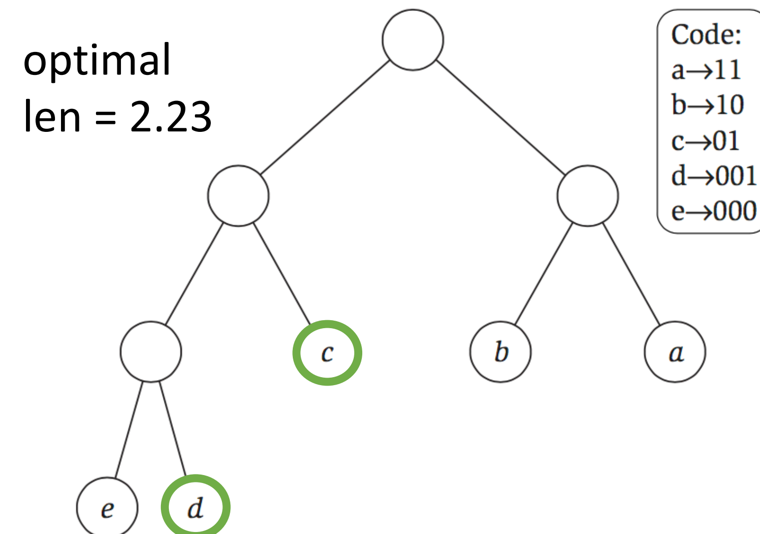


Huffman Codes

- First Try: split letters into two sets of roughly equal frequency and recurse

a	b	c	d	e
.32	.25	.20	.18	.05





Huffman Codes

- Huffman's Algorithm: pair up the two letters with the lowest frequency and recurse

a	b	c	d	e
.32	.25	.20	.18	.05

Huffman Codes

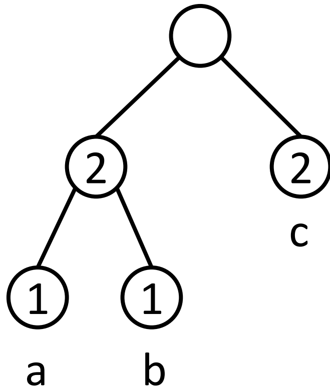
- Huffman's Algorithm: pair up the two letters with the lowest frequency and recurse
- Theorem: Huffman's Algorithm produces a prefix-free code of optimal length
  - We'll prove the theorem using an exchange argument

Huffman Codes

- Theorem: Huffman's Alg produces an optimal prefix-free code
- (1) In an optimal prefix-free code (a tree), every internal node has exactly two children

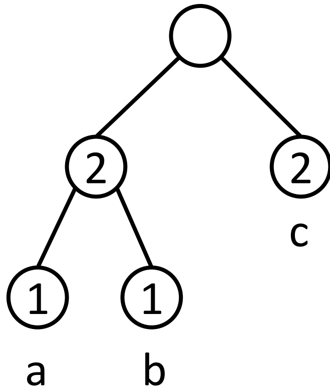
Huffman Codes

- Theorem: Huffman's Alg produces an optimal prefix-free code
- (1) In an optimal prefix-free code (a tree), every internal node has exactly two children



Huffman Codes

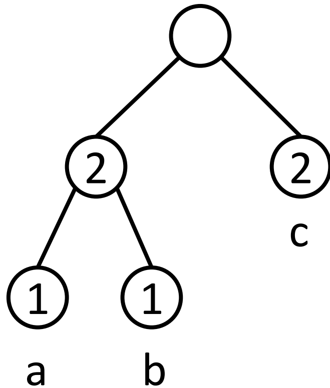
- Theorem: Huffman's Alg produces an optimal prefix-free code
- (1) In an optimal prefix-free code (a tree), every internal node has exactly two children



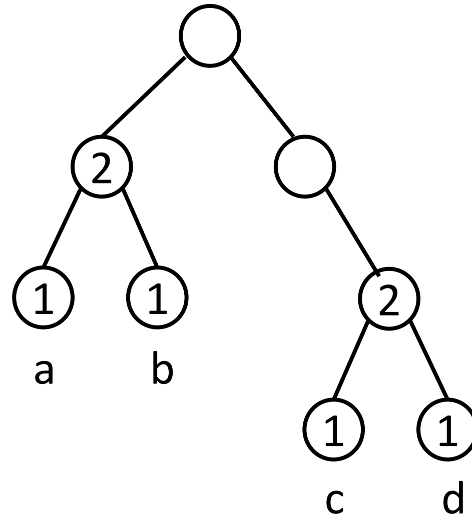
Adding another internal node anywhere would only raise the average length!

Huffman Codes

- Theorem: Huffman's Alg produces an optimal prefix-free code
- (1) In an optimal prefix-free code (a tree), every internal node has exactly two children

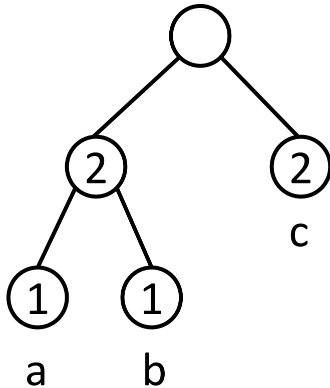


Adding another internal node anywhere would only raise the average length!

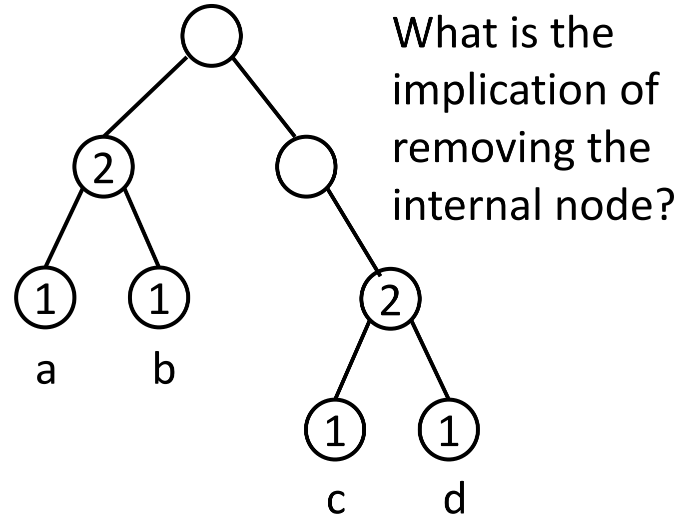


Huffman Codes

- Theorem: Huffman’s Alg produces an optimal prefix-free code
- (1) In an optimal prefix-free code (a tree), every internal node has exactly two children

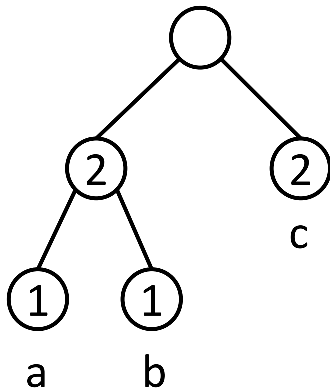


Adding another internal node anywhere would only raise the average length!

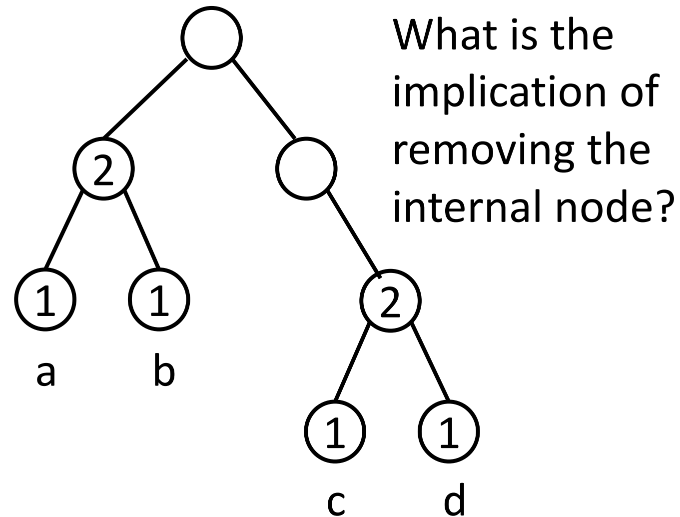


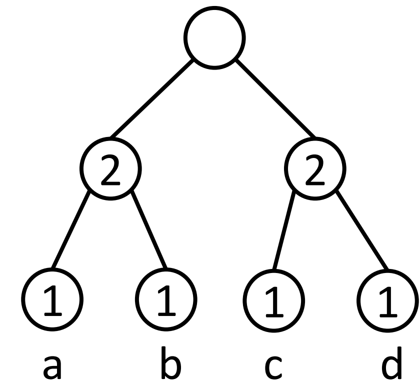
Huffman Codes

- **Theorem:** Huffman’s Alg produces an optimal prefix-free code
- (1) In an optimal prefix-free code (a tree), every internal node has exactly two children



Adding another internal node anywhere would only raise the average length!



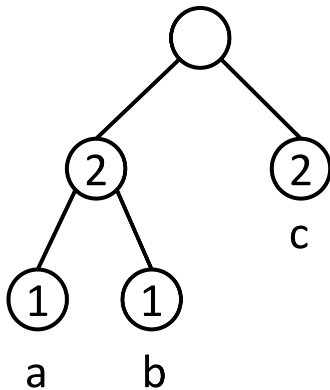


A strictly shorter code!

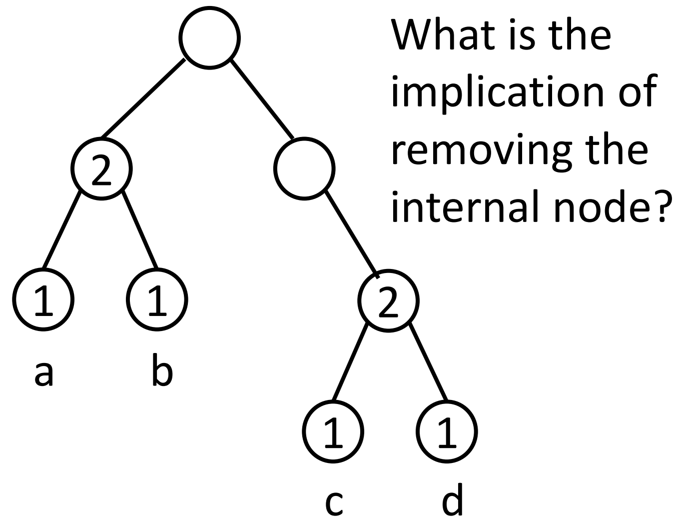
Huffman Codes

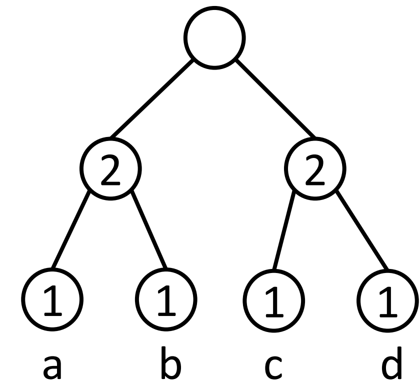
- Theorem: Huffman's Alg produces an optimal prefix-free code
- (1) In an optimal prefix-free code (a tree), every internal node has exactly two children

Implication: If a code tree has depth d, there are at least 2 leaves at depth d that are siblings!



Adding another internal node anywhere would only raise the average length!





A strictly shorter code!

Huffman Codes

- Theorem: Huffman's Alg produces an optimal prefix-free code
- (2) If x, y have the lowest frequency, then there is an optimal code where x, y are siblings and are at the bottom of the tree

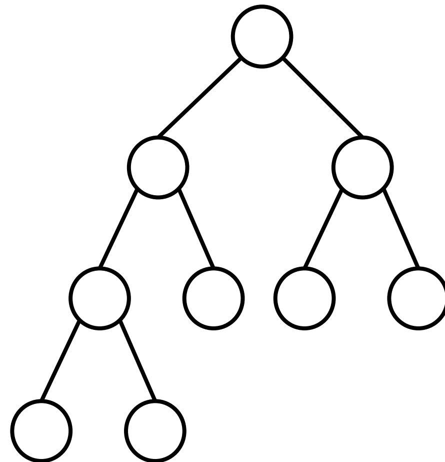
Huffman Codes

- Theorem: Huffman's Alg produces an optimal prefix-free code
- (2) If x, y have the lowest frequency, then there is an optimal code where x, y are siblings and are at the bottom of the tree

Suppose someone gave you the optimal tree, but with no labels.

Ex: $\Sigma = \{a, b, c, d, e\}$, with $f_a > f_b > f_c > f_d > f_e$

How should you label the leaves?



Huffman Codes

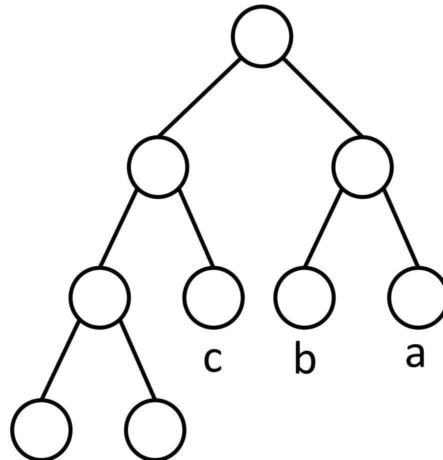
- Theorem: Huffman's Alg produces an optimal prefix-free code
- (2) If x, y have the lowest frequency, then there is an optimal code where x, y are siblings and are at the bottom of the tree

Suppose someone gave you the optimal tree, but with no labels.

Ex: $\Sigma = \{a, b, c, d, e\}$, with $f_a > f_b > f_c > f_d > f_e$

How should you label the leaves?

By definition, the highest frequency symbols should be on the highest leaves!



Huffman Codes

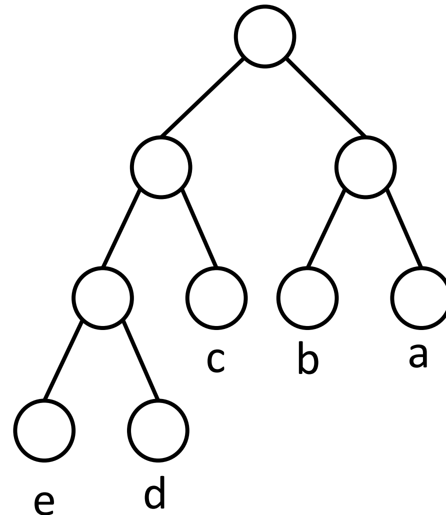
- Theorem: Huffman's Alg produces an optimal prefix-free code
- (2) If x, y have the lowest frequency, then there is an optimal code where x, y are siblings and are at the bottom of the tree

Suppose someone gave you the optimal tree, but with no labels.

Ex: $\Sigma = \{a, b, c, d, e\}$, with $f_a > f_b > f_c > f_d > f_e$

How should you label the leaves?

By definition, the highest frequency symbols should be on the highest leaves!



Given what we proved in (1), the two least frequent symbols will be siblings at the lowest depth!

Huffman Codes

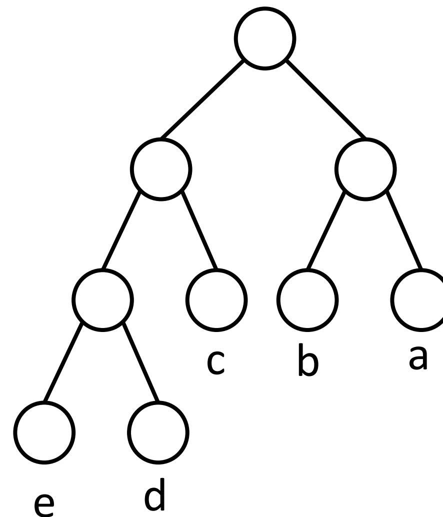
- Theorem: Huffman's Alg produces an optimal prefix-free code
- (2) If x, y have the lowest frequency, then there is an optimal code where x, y are siblings and are at the bottom of the tree

Suppose someone gave you the optimal tree, but with no labels.

Ex: $\Sigma = \{a, b, c, d, e\}$, with $f_a > f_b > f_c > f_d > f_e$

How should you label the leaves?

By definition, the highest frequency symbols should be on the highest leaves!



Given what we proved in (1), the two least frequent symbols will be siblings at the lowest depth!

Implication: The first step of Huffman's Algorithm is towards an optimal code!

Huffman Codes

- Theorem: Huffman's Alg produces an optimal prefix-free code
- Proof by Induction on the Number of Letters in Σ:

Huffman Codes

- Theorem: Huffman's Alg produces an optimal prefix-free code
- Proof by Induction on the Number of Letters in Σ:
  - Base case ($|\Sigma| = 2$): rather obvious

Huffman Codes

- Theorem: Huffman's Alg produces an optimal prefix-free code
- Proof by Induction on the Number of Letters in Σ:
    - Base case ($|\Sigma| = 2$): rather obvious

Inductive Step: If Huffman's algorithm is optimal for $|\Sigma| = k - 1$, then it is optimal for $|\Sigma| = k$.

Huffman Codes

- Theorem: Huffman's Alg produces an optimal prefix-free code
- Proof by Induction on the Number of Letters in Σ:
  - Base case ($|\Sigma| = 2$): rather obvious

Inductive Step: If Huffman's algorithm is optimal for $|\Sigma| = k - 1$, then it is optimal for $|\Sigma| = k$.

Suppose we have frequencies $f_1 \geq f_2 \geq \cdots \geq f_{k-1} \geq f_k$.

Huffman Codes

- Theorem: Huffman's Alg produces an optimal prefix-free code
- Proof by Induction on the Number of Letters in Σ:
  - Base case ($|\Sigma| = 2$): rather obvious

Inductive Step: If Huffman's algorithm is optimal for $|\Sigma| = k - 1$, then it is optimal for $|\Sigma| = k$.

Suppose we have frequencies $f_1 \geq f_2 \geq \cdots \geq f_{k-1} \geq f_k$.

Based on Huffman's alg and what we proved in (1) and (2), we merge f_{k-1} and f_k to get a new symbol w. Now we have

$\Sigma' = \{1, 2, \cdots, k - 2, w\}$, where $f_w = f_{k-1} + f_k$

Huffman Codes

- Theorem: Huffman's Alg produces an optimal prefix-free code
- Proof by Induction on the Number of Letters in Σ:
  - Base case ($|\Sigma| = 2$): rather obvious

Inductive Step: If Huffman's algorithm is optimal for $|\Sigma| = k - 1$, then it is optimal for $|\Sigma| = k$.

Suppose we have frequencies $f_1 \geq f_2 \geq \cdots \geq f_{k-1} \geq f_k$.

Based on Huffman's alg and what we proved in (1) and (2), we merge f_{k-1} and f_k to get a new symbol w. Now we have

$\Sigma' = \{1, 2, \cdots, k-2, w\}$, where $f_w = f_{k-1} + f_k$

Now $|\Sigma'| = k - 1$, which is optimal by the inductive hypothesis.

Huffman Codes

- Theorem: Huffman's Alg produces an optimal prefix-free code

We showed that…

1) In an optimal prefix-free code tree, every internal node has exactly two children
2) If symbols x, y have the lowest frequency, then there is an optimal code where x, y are siblings and are at the bottom of the tree
3) Every Huffman code satisfies these two properties by definition. Therefore, a code produced by Huffman's algorithm is an optimal prefix-free code. We proved this by induction on the number of symbols.

An Experiment

- Take the Dickens novel *A Tale of Two Cities*
  - File size is 799,940 bytes
- Build a Huffman code and compress

char	frequency	code
‘A’	48165	1110
‘B’	8414	101000
‘C’	13896	00100
‘D’	28041	0011
‘E’	74809	011
‘F’	13559	111111
‘G’	12530	111110
‘H’	38961	1001

char	frequency	code
‘I’	41005	1011
‘J’	710	1111011010
‘K’	4782	11110111
‘L’	22030	10101
‘M’	15298	01000
‘N’	42380	1100
‘O’	46499	1101
‘P’	9957	101001
‘Q’	667	1111011001

char	frequency	code
‘R’	37187	0101
‘S’	37575	1000
‘T’	54024	000
‘U’	16726	01001
‘V’	5199	1111010
‘W’	14113	00101
‘X’	724	1111011011
‘Y’	12177	111100
‘Z’	215	1111011000

  - File size is now 439,688 bytes

	Raw	Huffman
Size	799,940	439,688

Huffman Codes

- Huffman's Algorithm: pair up the two letters with the lowest frequency and recurse
- Theorem: Huffman's Algorithm produces a prefix-free code of optimal length
- In what sense is this code really optimal? (Bonus material…will not test you on this)

Length of Huffman Codes

- What can we say about Huffman code length?
  - Suppose $f_i = 2^{-\ell_i}$ for every $i \in \Sigma$
  - Then, $\mathrm{len}_T(i) = \ell_i$ for the optimal Huffman code

Letter	a	b	c	d
Frequency	2^{-1}	2^{-2}	2^{-3}	2^{-3}
Code	0	01	110	111
Length	1	2	3	3

Length of Huffman Codes

- What can we say about Huffman code length?
  - Suppose $f_i = 2^{-\ell_i}$ for every $i \in \Sigma$
  - Then, $\mathrm{len}_T(i) = \ell_i$ for the optimal Huffman code
  - Length of the code is the sum of $2^{-\ell_i} \cdot \ell_i$ for all i

Letter	a	b	c	d
Frequency	2^{-1}	2^{-2}	2^{-3}	2^{-3}
Code	0	01	110	111
Length	1	2	3	3

Length of Huffman Codes

- What can we say about Huffman code length?
  - Suppose $f_i = 2^{-\ell_i}$ for every $i \in \Sigma$
  - Then, $\text{len}_T(i) = \ell_i$ for the optimal Huffman code
  - Length of the code is the sum of $2^{-\ell_i} \cdot \ell_i$ for all i
  - $\text{len}(T) = \sum_{i \in \Sigma} f_i \cdot \log_2\left({}^{1}/_{f_i}\right)$
    - $\log_2(f_i) = -\ell_i$
    - $\log_2\left(\frac{1}{f_i}\right) = -\ell_i$

Letter	a	b	c	d
Frequency	2^{-1}	2^{-2}	2^{-3}	2^{-3}
Code	0	01	110	111
Length	1	2	3	3

Entropy

- Given a set of frequencies (aka a probability distribution) the entropy is

$$H(f) = \sum_i f_i \cdot \log_2 \left({}^1/_{f_i} \right)$$

Entropy

- Given a set of frequencies (aka a probability distribution) the entropy is

$$H(f) = \sum_i f_i \cdot \log_2 \left({}^{1}/_{f_i} \right)$$

- Entropy is a “measure of randomness”
- Entropy was introduced by Shannon in 1948 and is the foundational concept in:
    - Data compression
    - Error correction (communicating over noisy channels)
    - Security (passwords and cryptography)

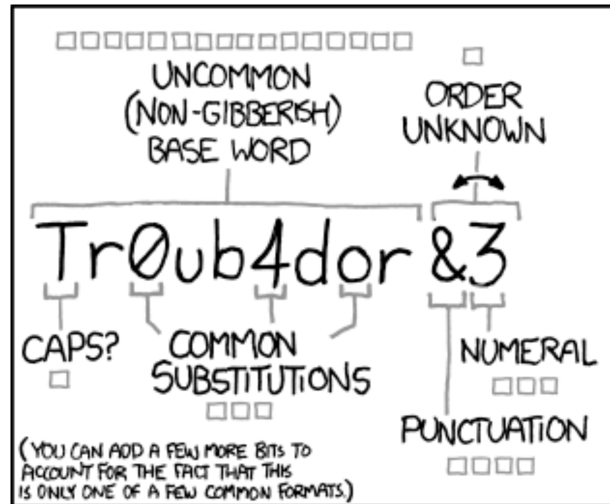
Entropy of Passwords

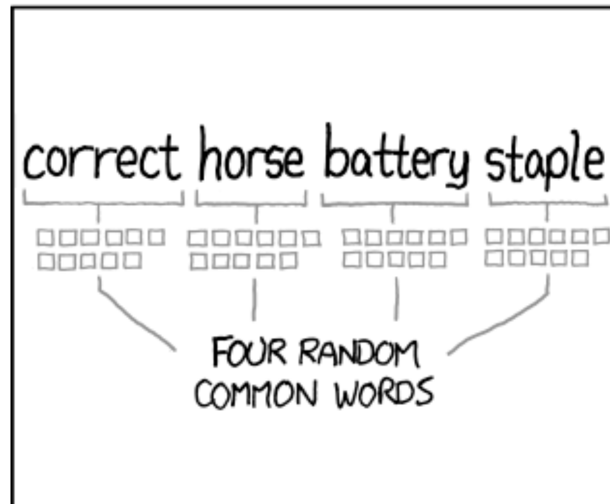
- Your password is a specific string, so $f_{pwd} = 1.0$

- To talk about security of passwords, we have to model them as **random**
  - Random 16 letter string: $H = 16 \cdot \log_2 26 \approx 75.2$
  - Random IMDb movie: $H = \log_2 1764727 \approx 20.7$
  - Your favorite IMDb movie: $H \ll 20.7$

- Entropy measures how difficult passwords are to guess “on average”

Entropy of Passwords





Entropy and Compression

- Given a set of frequencies (probability distribution) the entropy is
$$H(f) = \sum_i f_i \cdot \log_2 \left({}^{1}\!/_{f_i} \right)$$
- Suppose that we generate string S by choosing n random letters independently with frequencies f
- Any compression scheme requires at least $H(f)$ bits-per-letter to store S (as $n \to \infty$)
  - Huffman codes are truly optimal!

But Wait!

- Take the Dickens novel *A Tale of Two Cities*
  - File size is 799,940 bytes
- Build a Huffman code and compress

char	frequency	code
‘A’	48165	1110
‘B’	8414	101000
‘C’	13896	00100
‘D’	28041	0011
‘E’	74809	011
‘F’	13559	111111
‘G’	12530	111110
‘H’	38961	1001

char	frequency	code
‘I’	41005	1011
‘J’	710	1111011010
‘K’	4782	11110111
‘L’	22030	10101
‘M’	15298	01000
‘N’	42380	1100
‘O’	46499	1101
‘P’	9957	101001
‘Q’	667	1111011001

char	frequency	code
‘R’	37187	0101
‘S’	37575	1000
‘T’	54024	000
‘U’	16726	01001
‘V’	5199	1111010
‘W’	14113	00101
‘X’	724	1111011011
‘Y’	12177	111100
‘Z’	215	1111011000

  - File size is now 439,688 bytes
- But we can do better!

	Raw	Huffman	gzip	bzip2
Size	799,940	439,688	301,295	220,156

What do the frequencies represent?

- Real data (e.g. natural language, music, images) have patterns between letters
  - U becomes a lot more common after a Q
- Possible approach: model pairs of letters
  - Build a Huffman code for pairs-of-letters
  - Improves compression ratio, but the tree gets bigger
  - Can only model certain types of patterns
- Zip is based on an algorithm called LZW that tries to identify patterns based on the data

Entropy and Compression

- Given a set of frequencies (probability distribution) the entropy is
$$H(f) = \sum_i f_i \cdot \log_2 \left({}^1\!/_{f_i} \right)$$
- Suppose that we generate string S by choosing n random letters independently with frequencies f
- Any compression scheme requires at least $H(f)$ bits-per-letter to store S
    - Huffman codes are truly optimal if and only if there is no relationship between different letters!

Wrap-up

Reading and Extra Credit Assignment: Will send out an announcement this evening

Tomorrow: Greedy algorithm for clustering and an application

Wednesday: Advanced topics and course wrap (let me know if you want to hear about something in particular!)

Thursday: Final exam review