CSE 417 Autumn 2025

Lecture 19: Huffman codes

Glenn Sun

Homework this week

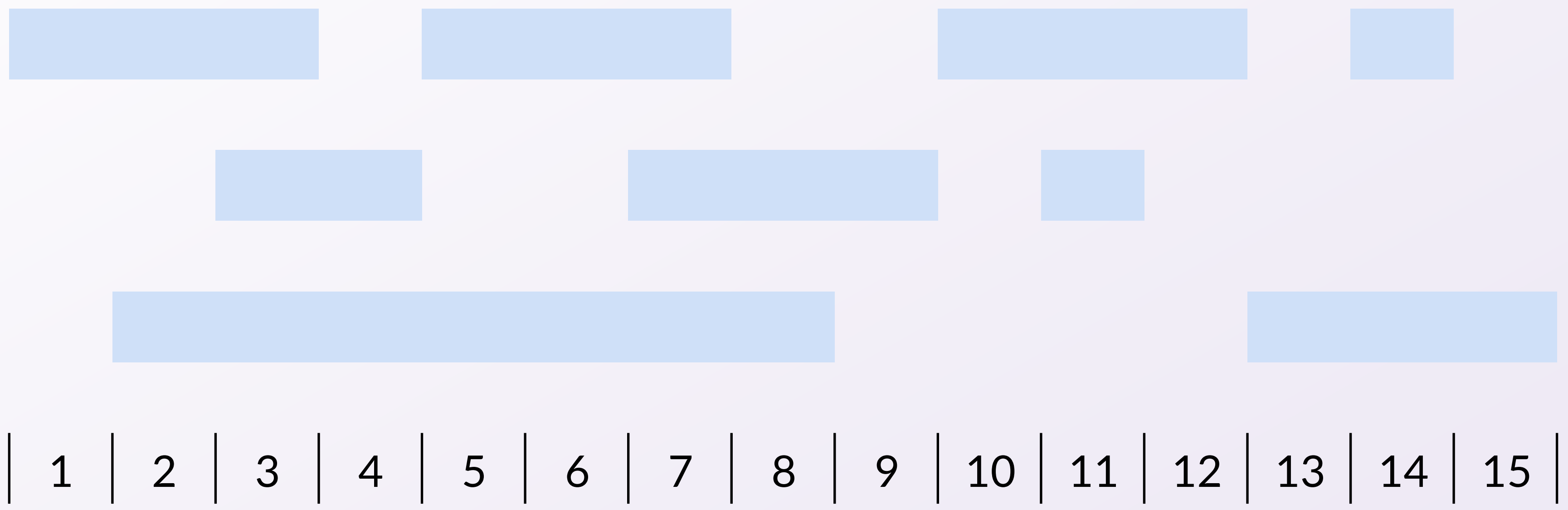
- P11: Extending the algorithm from today’s lecture
- P11X.1/2: Finding counterexamples
- P12: Your own greedy algorithm

Wrapping up from Wednesday

Meeting scheduler

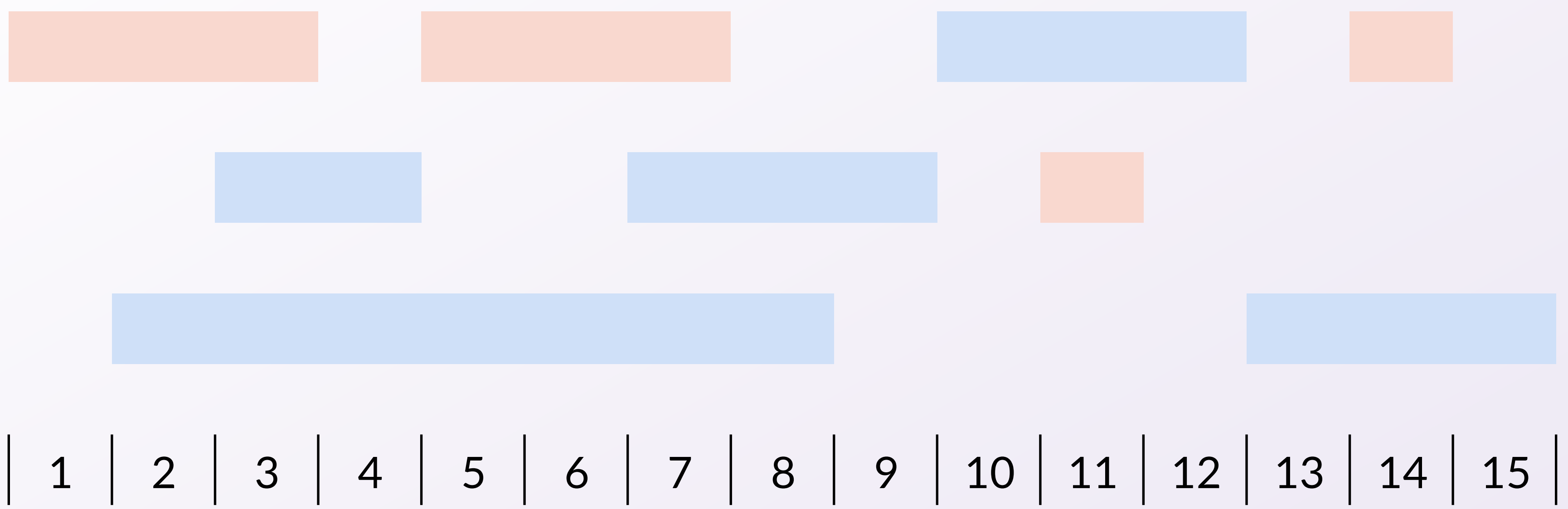
Input: List of time intervals (booking requests for a meeting room)

Output: Maximum number of meetings that can be booked



Our greedy solution

Algorithm: Pick the meeting that would end first.



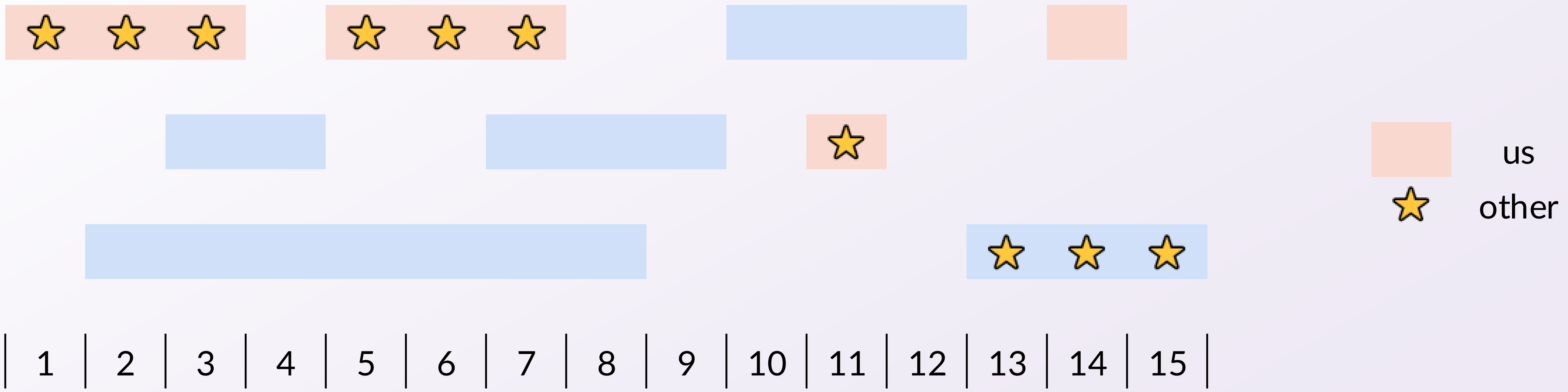
Exchange argument for meeting scheduler

Suppose another schedule was different and look at the first time our schedules differed.

Exchange argument for meeting scheduler

In the  solution, we could've picked our solution instead:

- Compatible with previous picks because previous are same
- Compatible with future picks because finishes earliest



Exchange argument for meeting scheduler

In the ⭐ solution, we could've picked our solution instead:

- Compatible with previous picks because previous are same
- Compatible with future picks because finishes earliest

Because this process can transform any solution into ours while maintaining (or increasing) the number of meetings, our solution must be optimal!

Prefix codes

Encoding messages

Goal: Transmit a message over a digital signal using few bits.

Standard encoding used by computers is ASCII:

A = 65 = 0100 0001

8 bits/letter

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
64	40	100	@	@	96	60	140	`	`
65	41	101	A	A	97	61	141	a	a
66	42	102	B	B	98	62	142	b	b
67	43	103	C	C	99	63	143	c	c
68	44	104	D	D	100	64	144	d	d
69	45	105	E	E	101	65	145	e	e
70	46	106	F	F	102	66	146	f	f

Better encodings

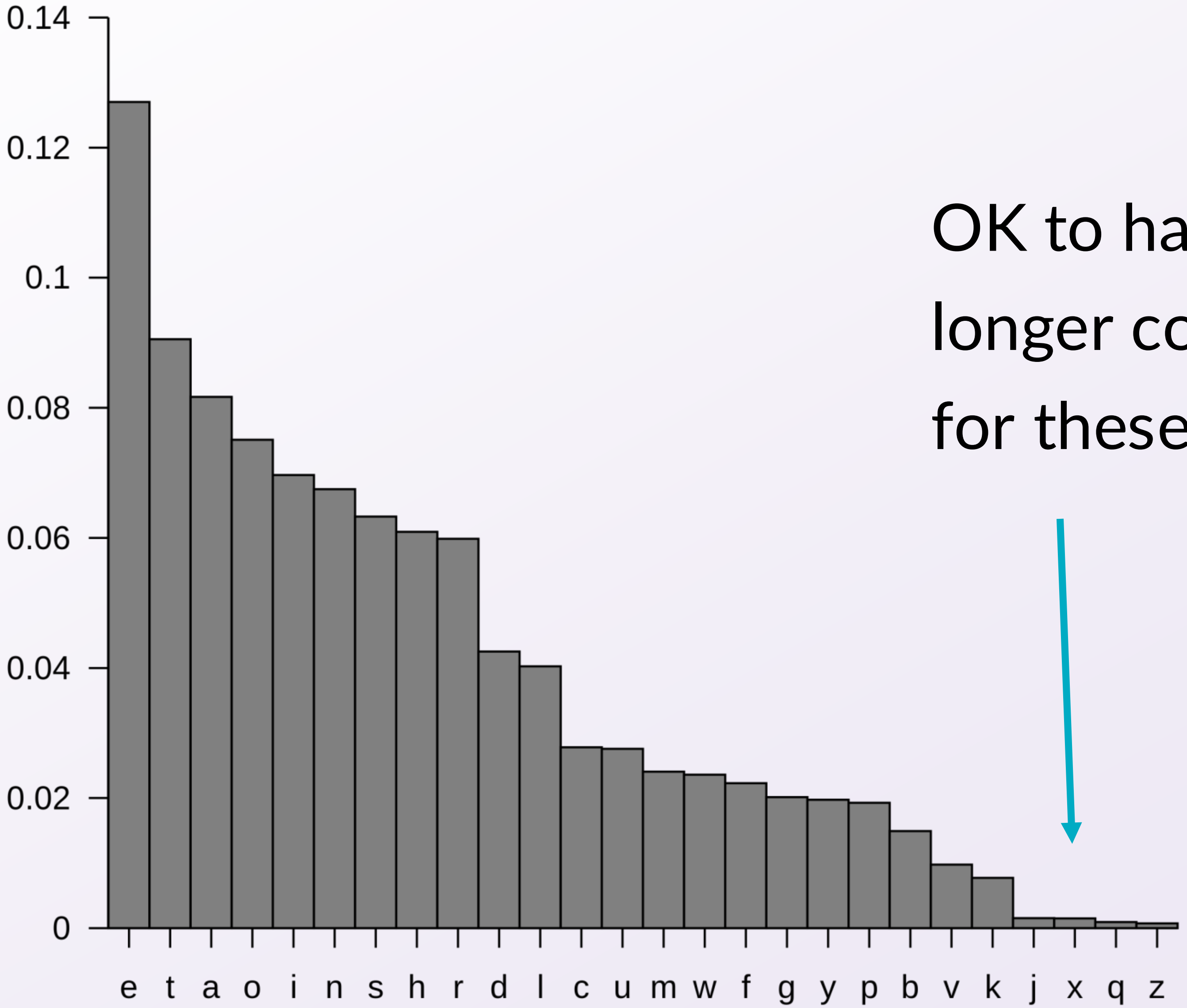
To communicate English letters + basic punctuation, we would need about $\mathbf{32 = 2^5}$ symbols.

So we can get away with just 5 bits.

But can we do even better?

Distribution of English letters

use shorter codes for these



OK to have longer codes for these

Prefix codes

If we allow codes of different lengths, it may become impossible to uniquely decode a message!

Symbol	Codeword
A	0
B	1
C	00
D	01
E	10
F	11

How to decode “000”?

Could be AAA, AC, or CA!

Prefix codes

We will require prefix codes (also called prefix-free codes): no codeword should be a prefix of another codeword.

Symbol	Codeword
A	00
B	01
C	100
D	101
E	110
F	111

Now, “AC” is 00100.

Can only decode this as “AC”.

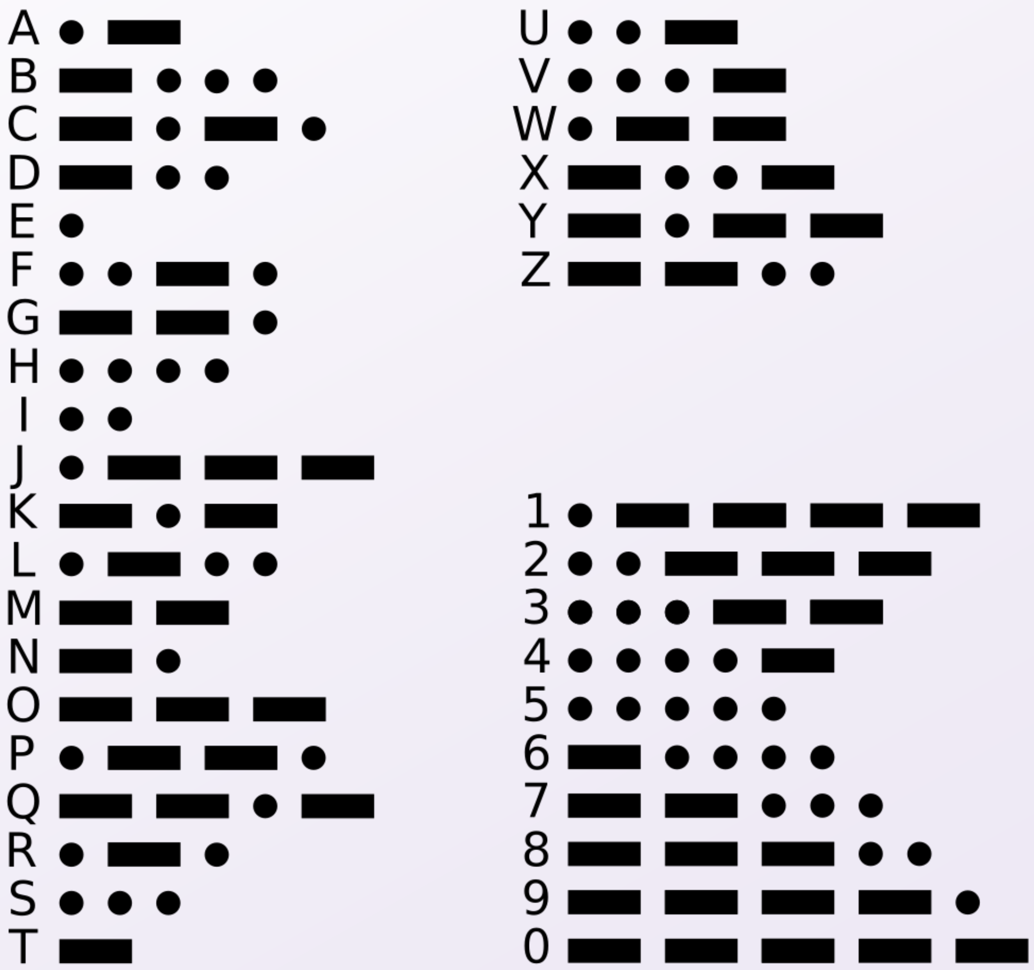
Side note: Morse code

Consider Morse code as being made up of “dits” and “dahs”.

Q: Is Morse code prefix free?

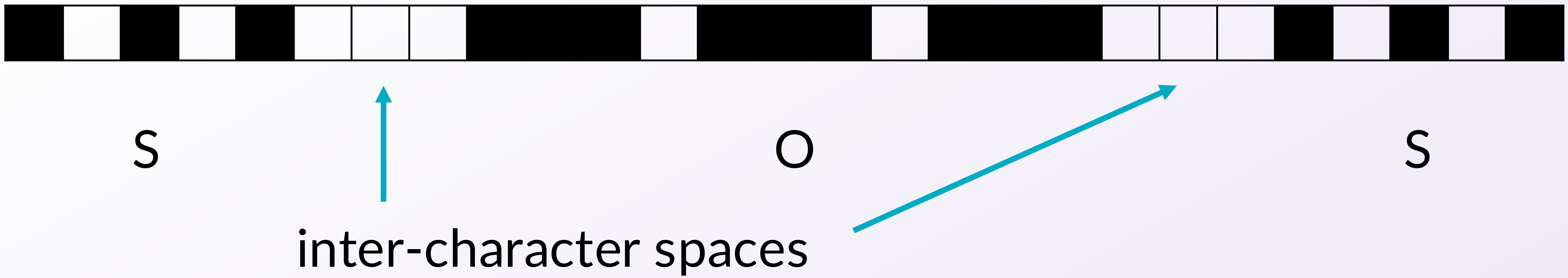
A: No! e.g. E is a prefix of A.

Morse code uses extra spacing between letters to allow decoding. Thus, in terms of “dits” and “dahs”, it is actually a ternary code!



Side note: Morse code

But if you consider it made up of "on" and "off", it is a prefix code.



Decoding a prefix code

Given a table of codewords and an encoded message, how to decode it?

Algorithm: Take the first prefix that is a codeword!

Symbol	Codeword
A	00
B	01
C	100
D	101
E	110
F	111

Example: Decode “10100”.

- “1” is not a codeword.
- “10” is not a codeword”.
- “101” is D, take this and remove “101” from string”

Decoding a prefix code

Algorithm: Take the first prefix that is a codeword!

Q: Is this a greedy algorithm?

A: Kind of? It is not an optimization problem, but makes “greedy-like” choices.

Why is it correct? The first codeword must be correct, because of the prefix-free property. (No other codeword can start with it.) Thus, we can repeat the argument to show that it’s all correct.

Calculating how good a code is

Take the average codeword length weighted by frequency:

$$f_1 c_1 + f_2 c_2 + \cdots + f_n c_n$$

where $f_1 + f_2 + \cdots + f_n = 1.$

Symbol	Codeword	Frequency
A	00	0.25
B	01	0.25
C	100	0.125
D	101	0.125
E	110	0.125
F	111	0.125

Example:

0.25(2+2) + 0.125(3+3+3+3)

= 2.5

Calculating how good a code is

Take the average codeword length weighted by frequency:

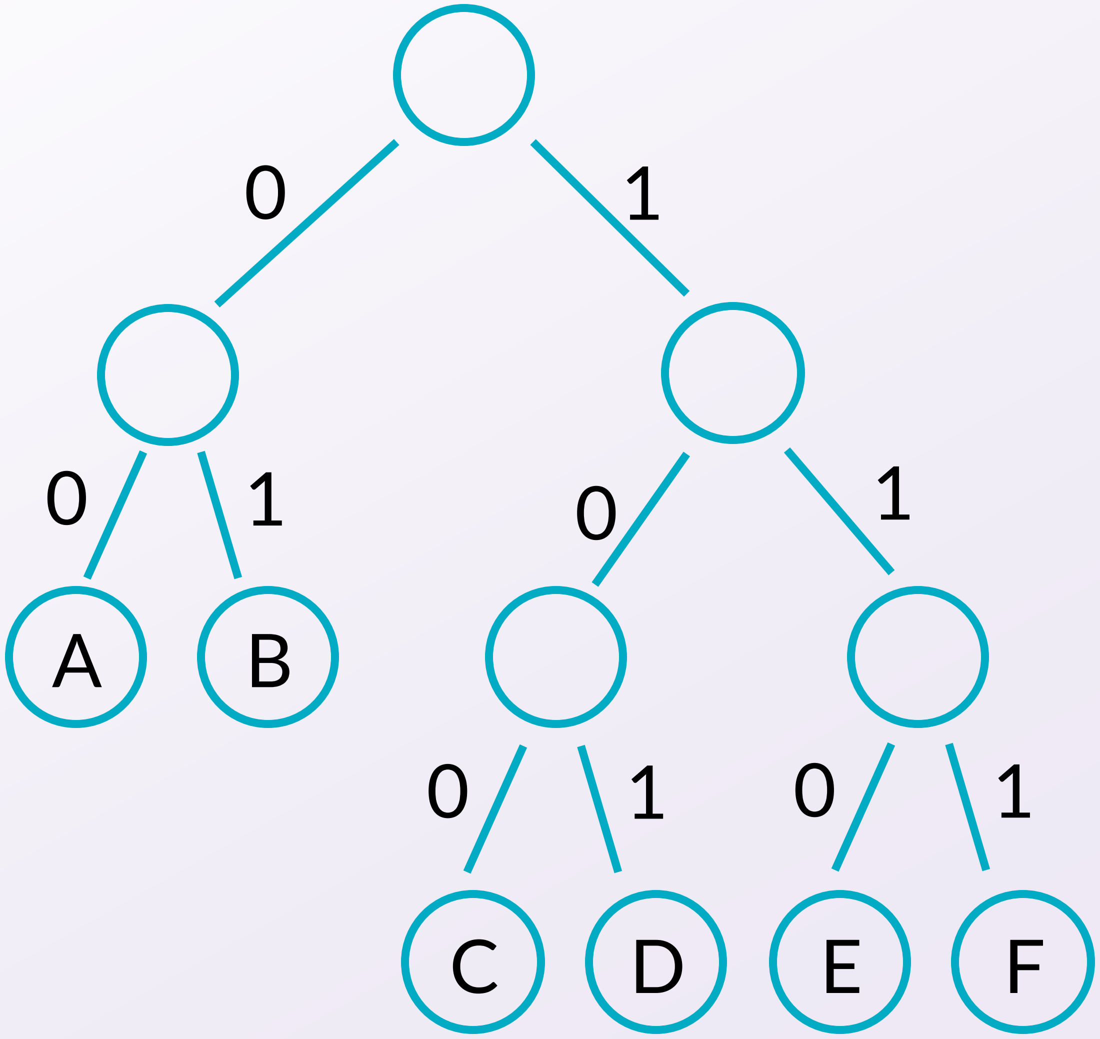
$$f_1 c_1 + f_2 c_2 + \cdots + f_n c_n$$

where $f_1 + f_2 + \cdots + f_n = 1$.

- ASCII: 8 bits/letter
- Morse: slightly > 9 bits/letter
- Huffman: slightly > 4 bits/letter!

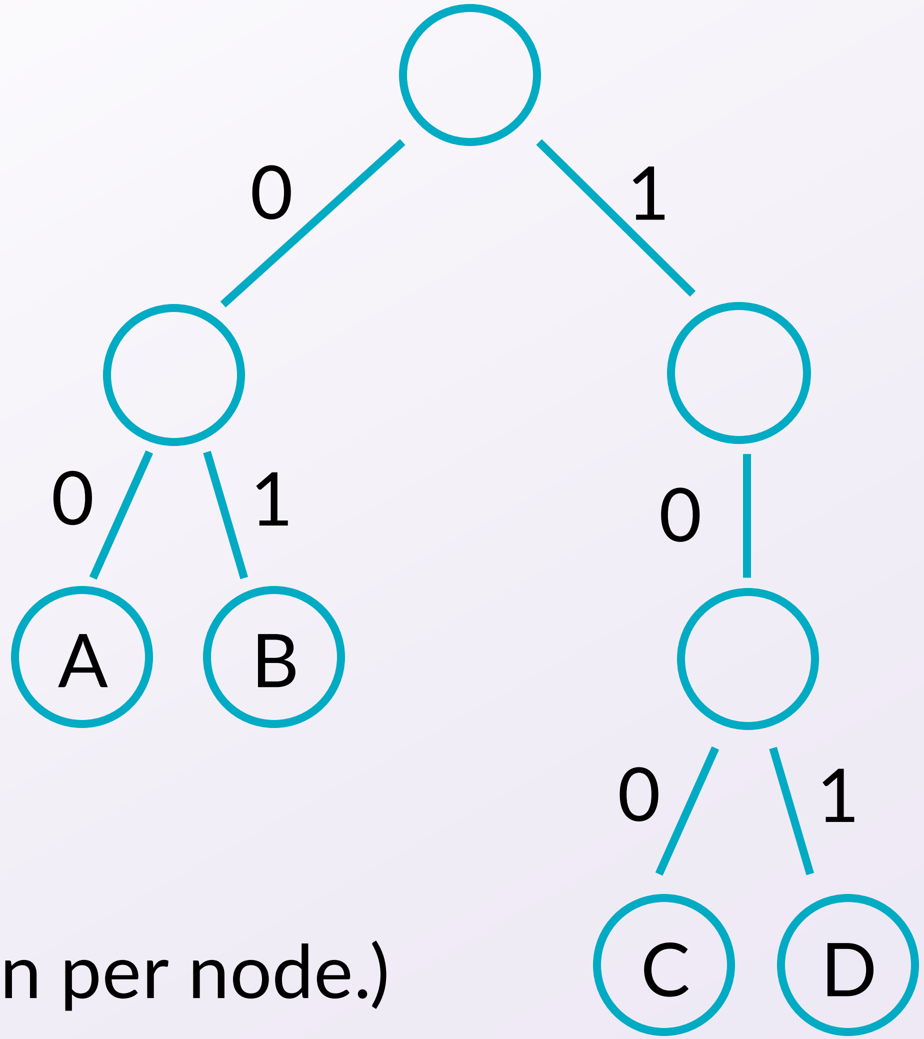
Viewing prefix codes as binary trees

Symbol	Codeword
A	00
B	01
C	100
D	101
E	110
F	111



But not necessarily full binary trees

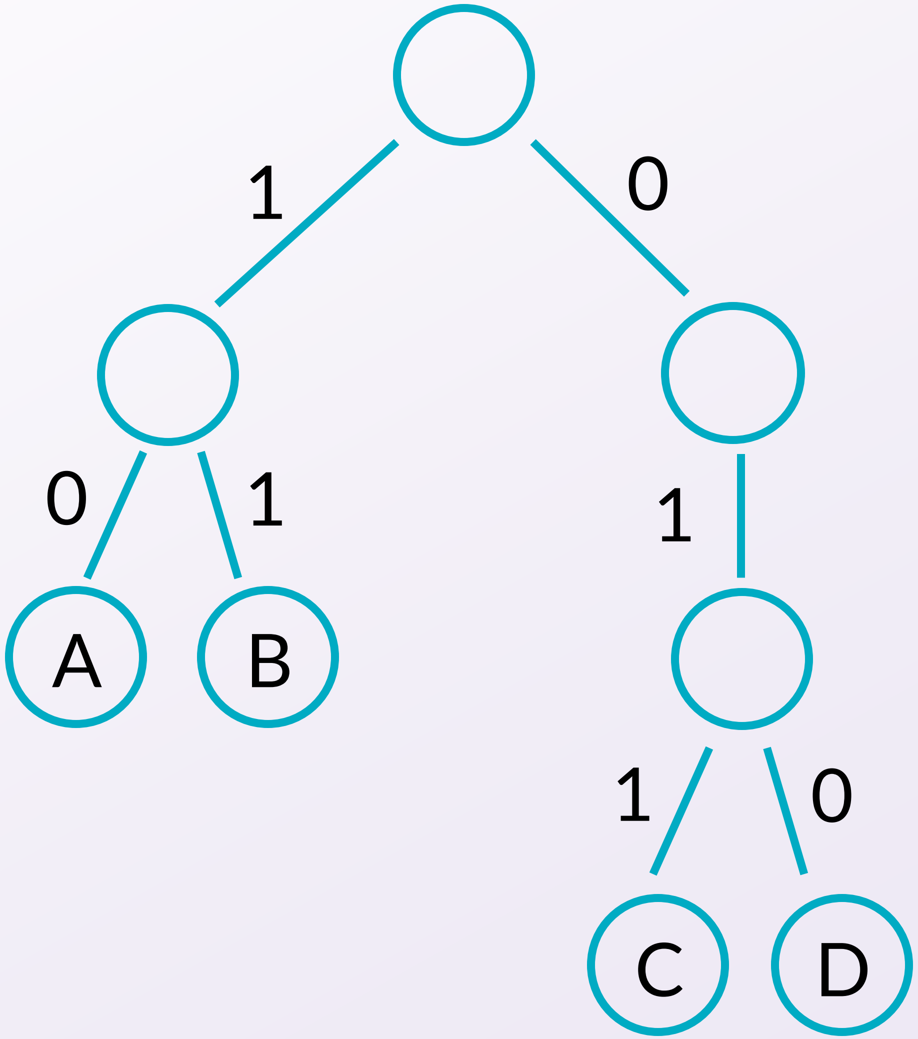
Symbol	Codeword
A	00
B	01
C	100
D	101



(Full binary tree has 0 or 2 children per node.)

Swapping 0/1 doesn't affect quality

Symbol	Codeword
A	10
B	11
C	011
D	010



Summary

- Can view any prefix code as a binary tree
- Can arbitrarily pick 0/1 labels for the edges of any binary tree to get a prefix code
- “Prefix-free” is equivalent to all symbols being leaves

Huffman codes

Huffman’s idea

Input: List of frequencies (sum = 1, or alternatively sometimes counts of each symbol)

Goal: Find a prefix code with minimum average length.

Greedy algorithm:

- Start with all symbols getting a node.
- **Choose the two least frequent symbols** and combine them into a subtree, setting the frequency of the subtree to be the sum.

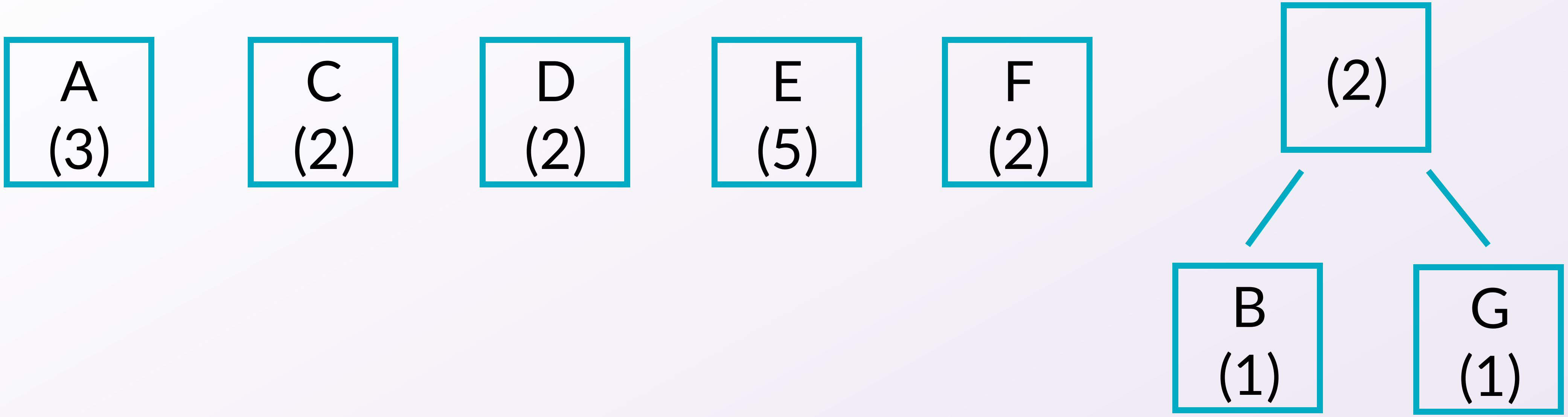
Example of Huffman's idea

A	B	C	D	E	F	G
(3)	(1)	(2)	(2)	(5)	(2)	(1)

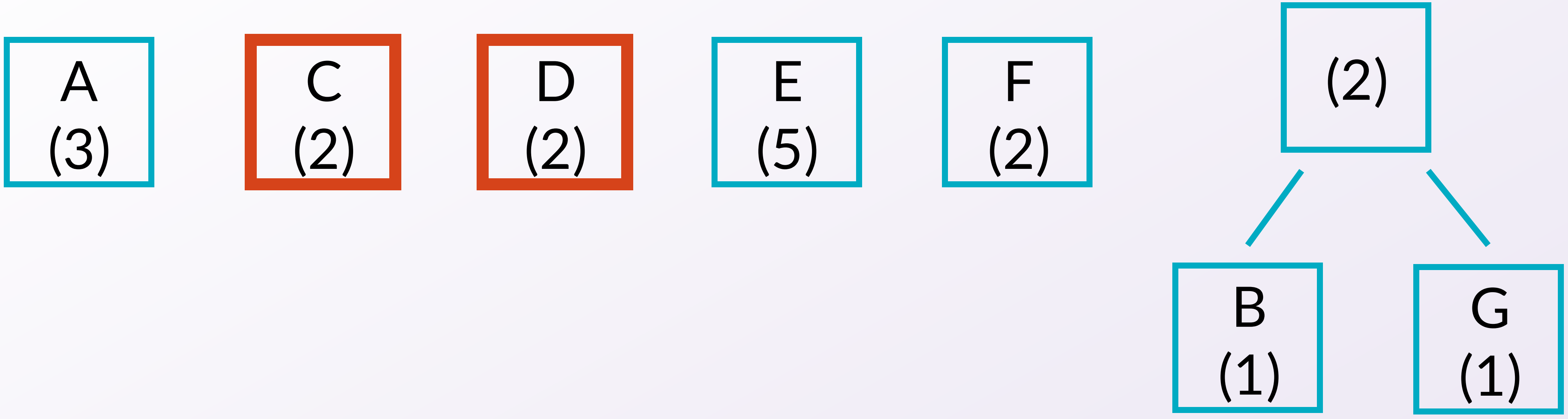
Example of Huffman's idea

A	B	C	D	E	F	G
(3)	(1)	(2)	(2)	(5)	(2)	(1)

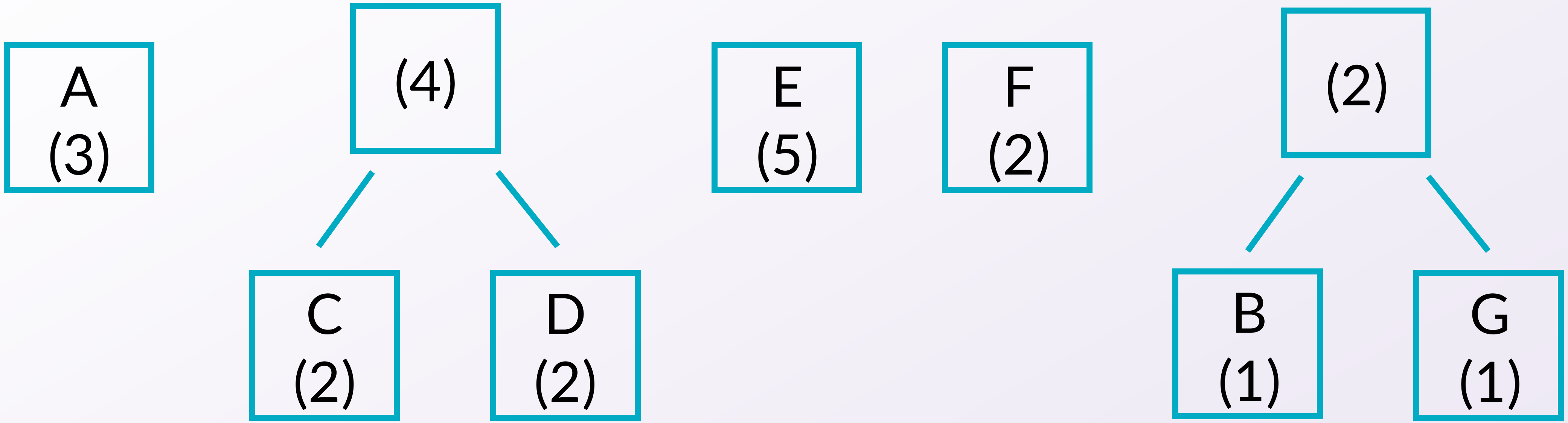
Example of Huffman's idea



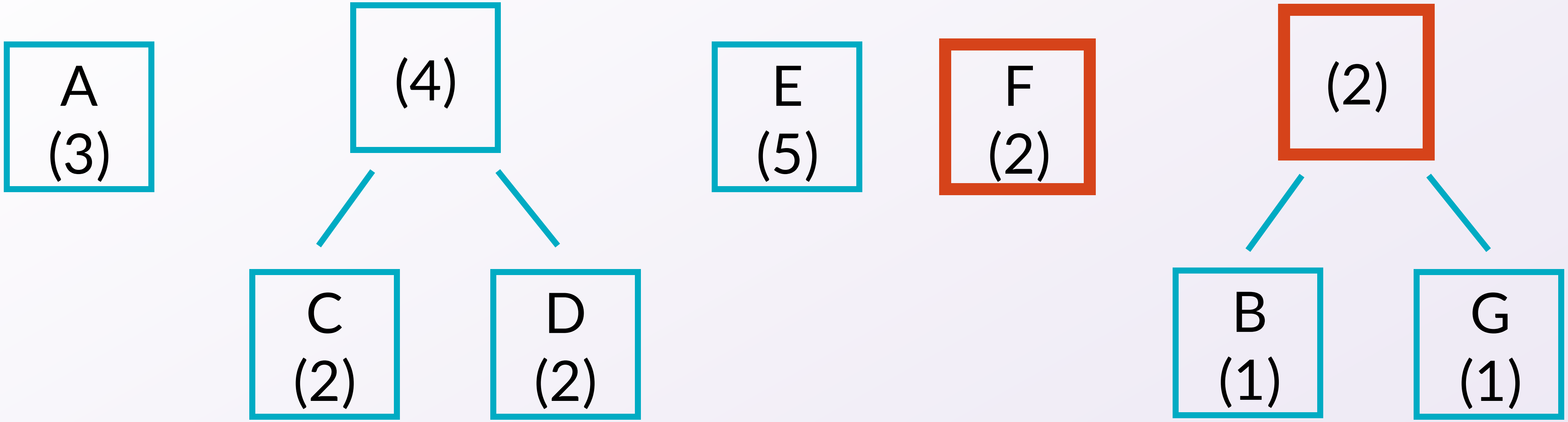
Example of Huffman's idea



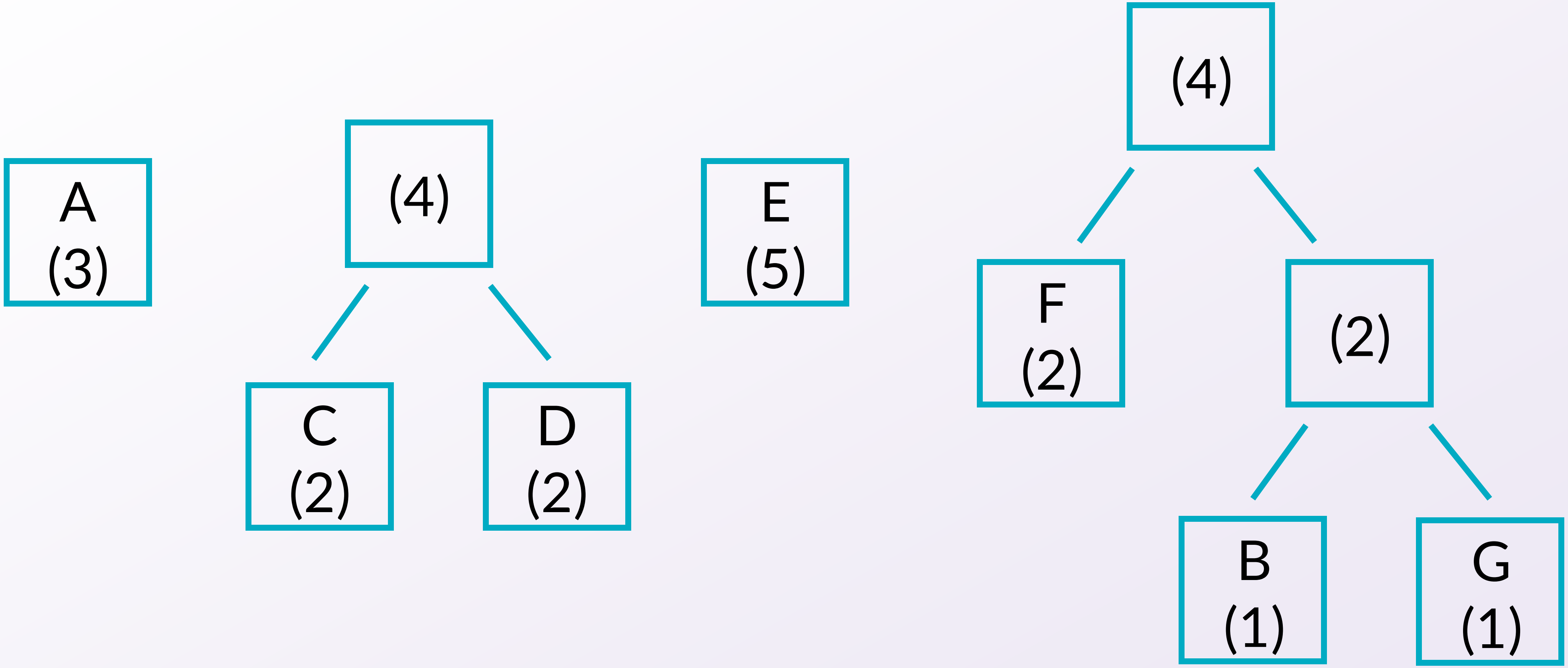
Example of Huffman's idea



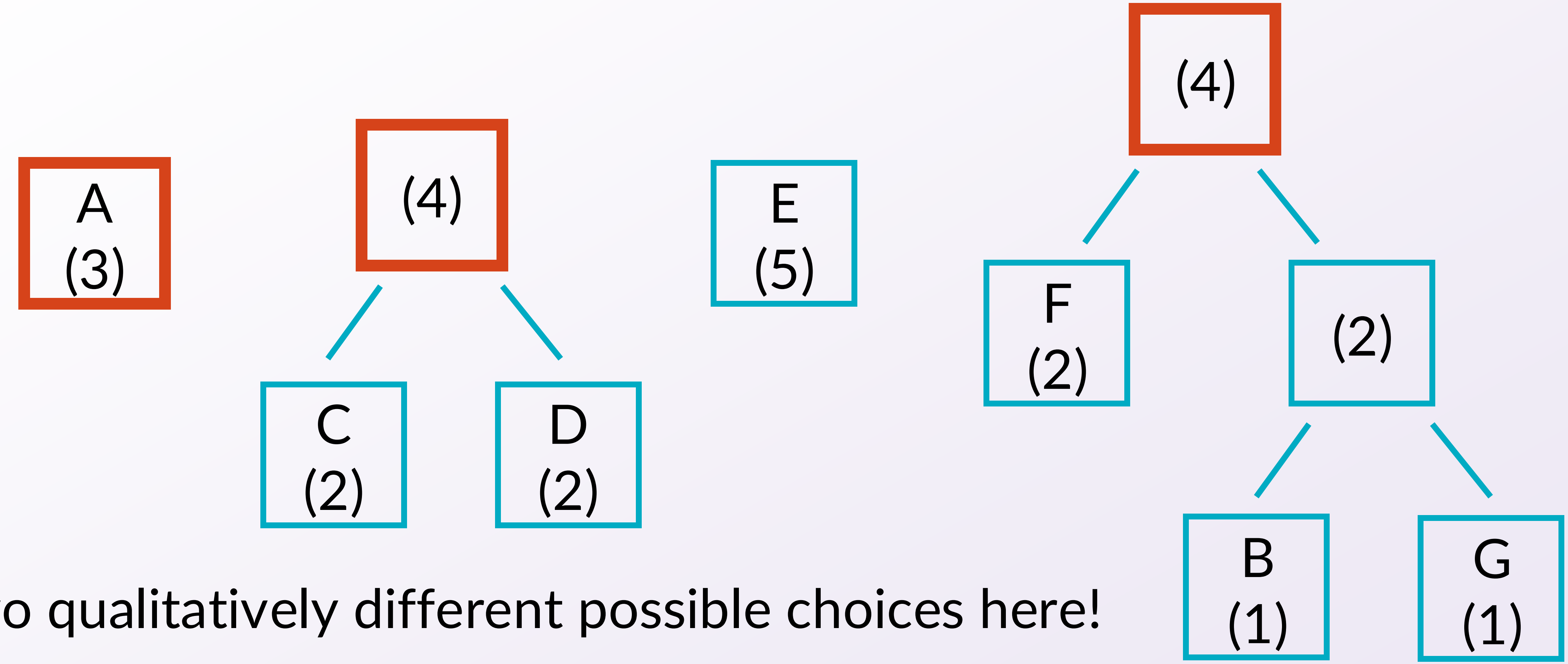
Example of Huffman’s idea



Example of Huffman's idea



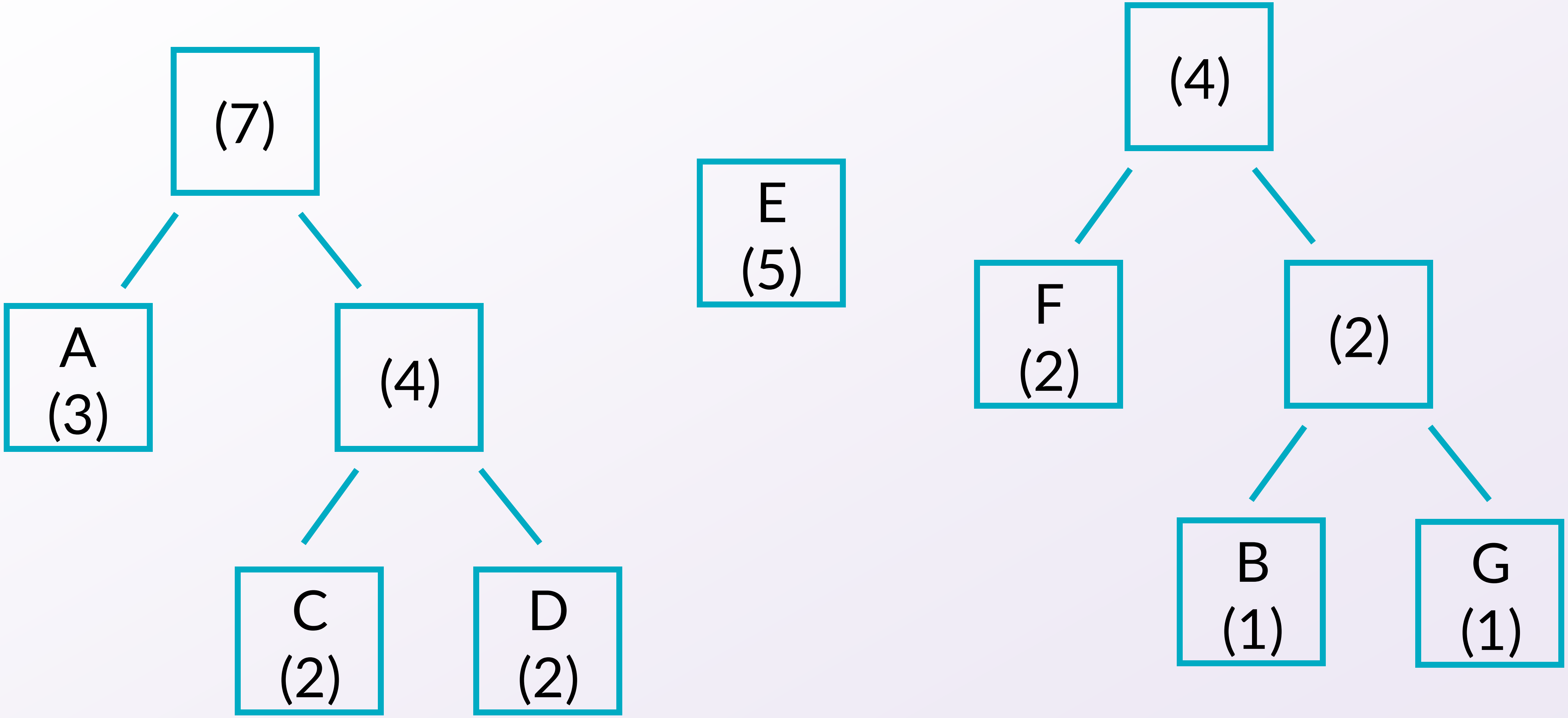
Example of Huffman's idea



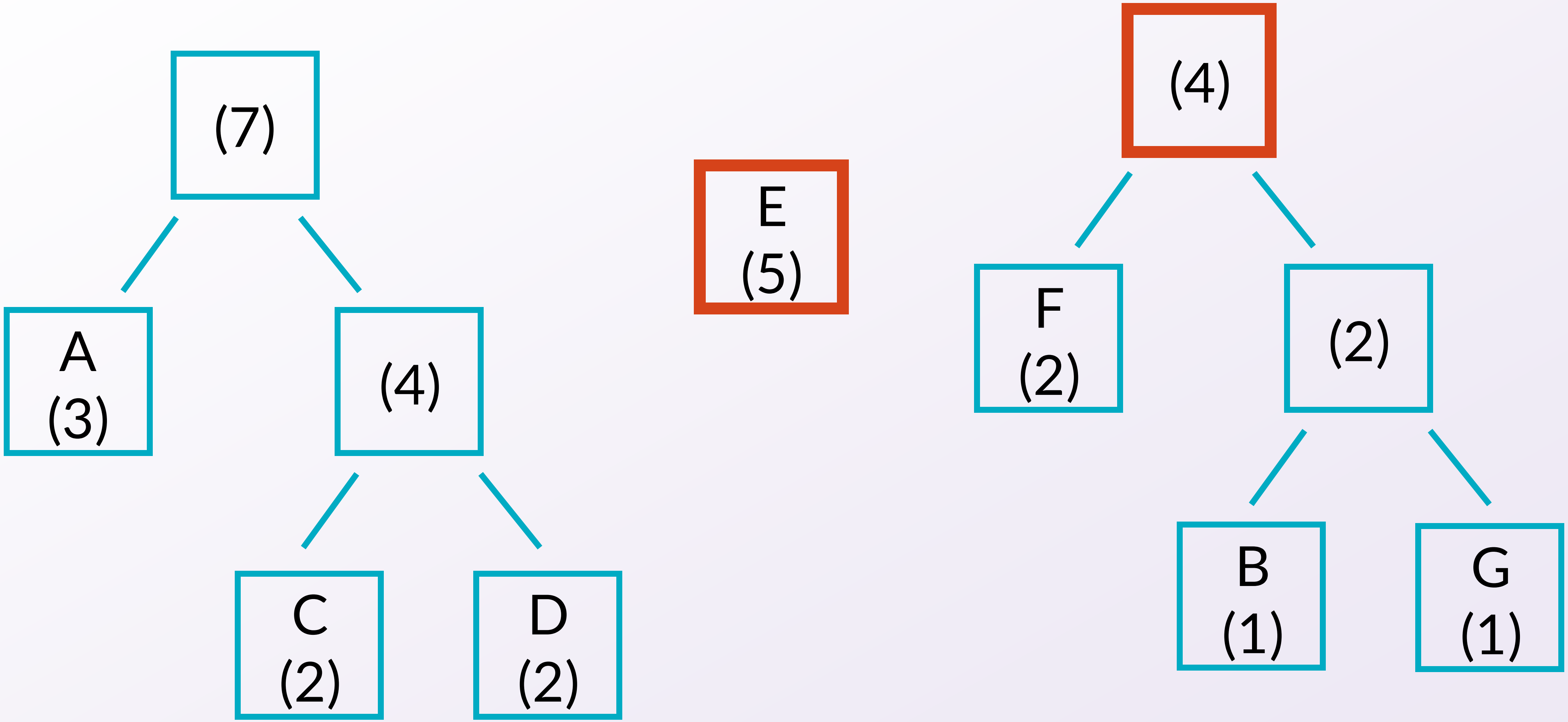
Two qualitatively different possible choices here!

Choose either one, either will work!

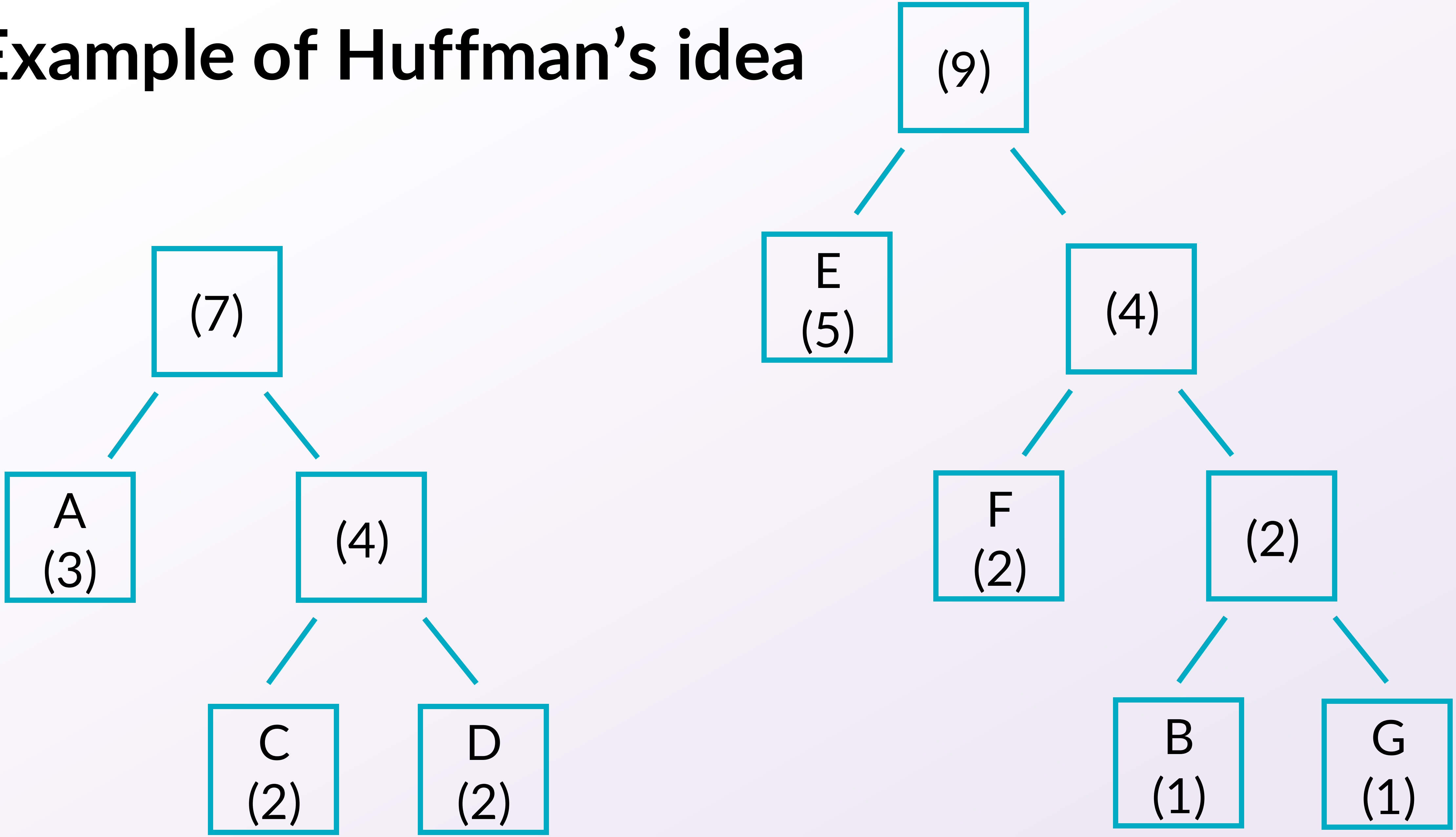
Example of Huffman’s idea



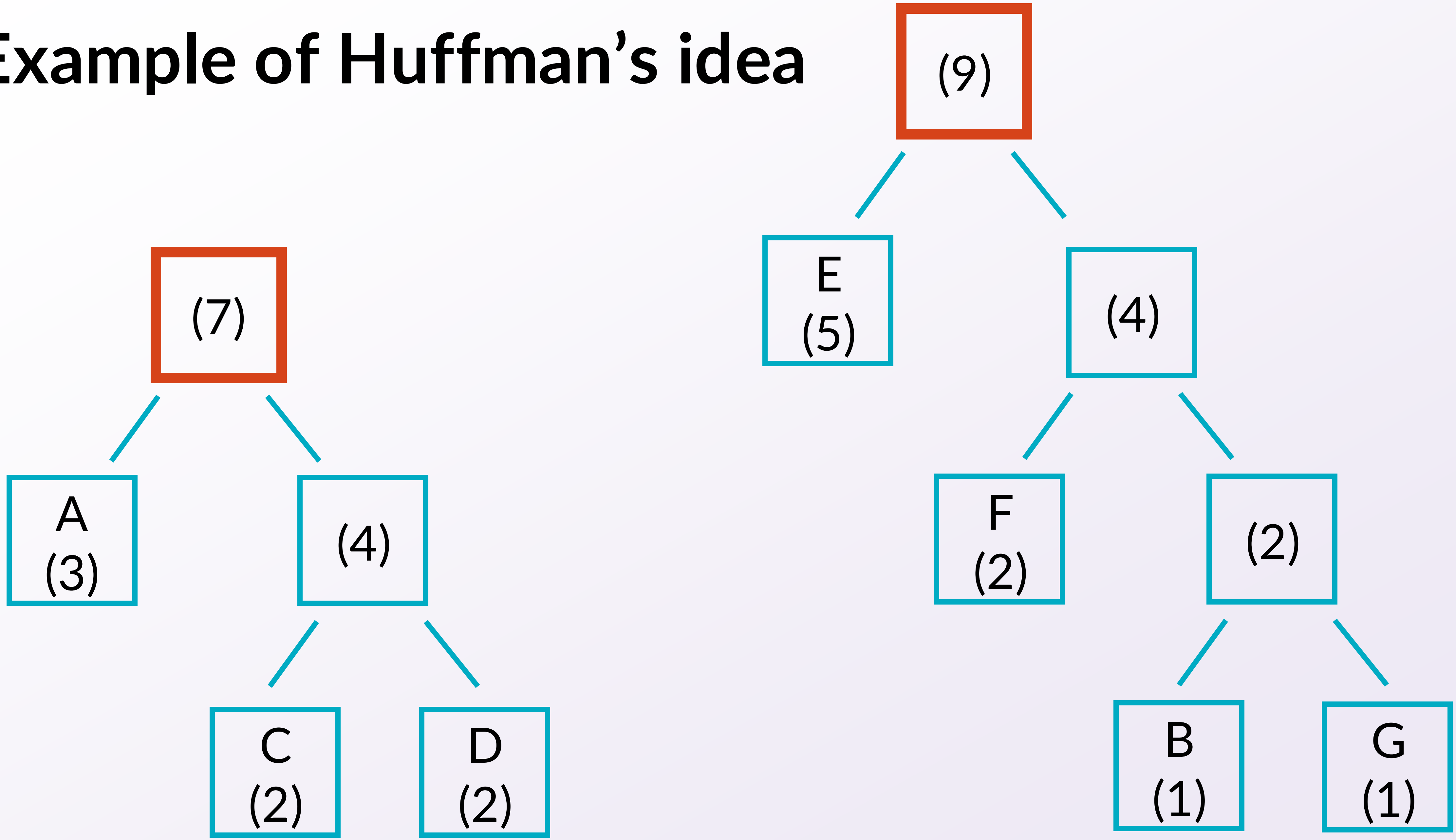
Example of Huffman's idea

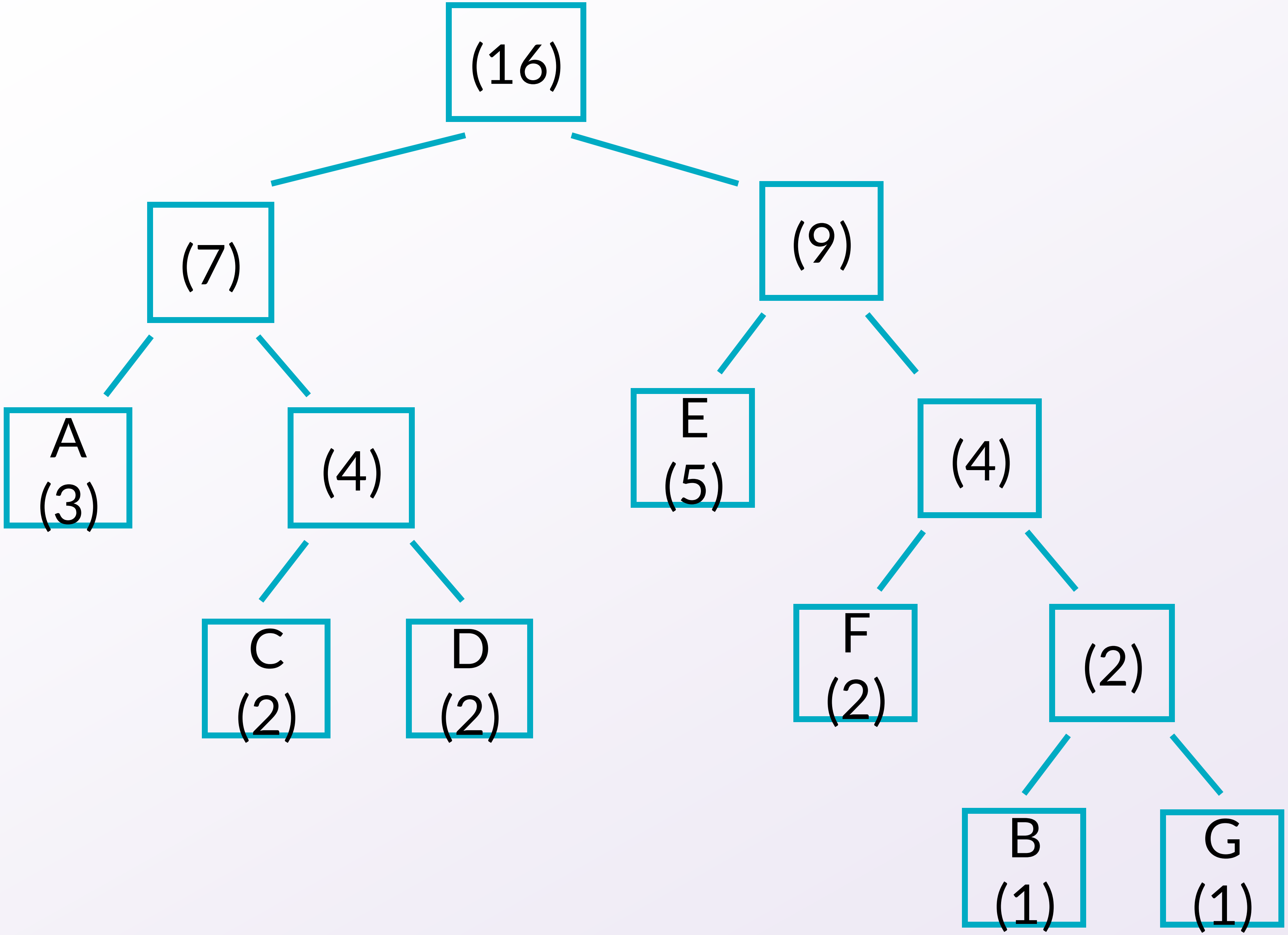


Example of Huffman's idea

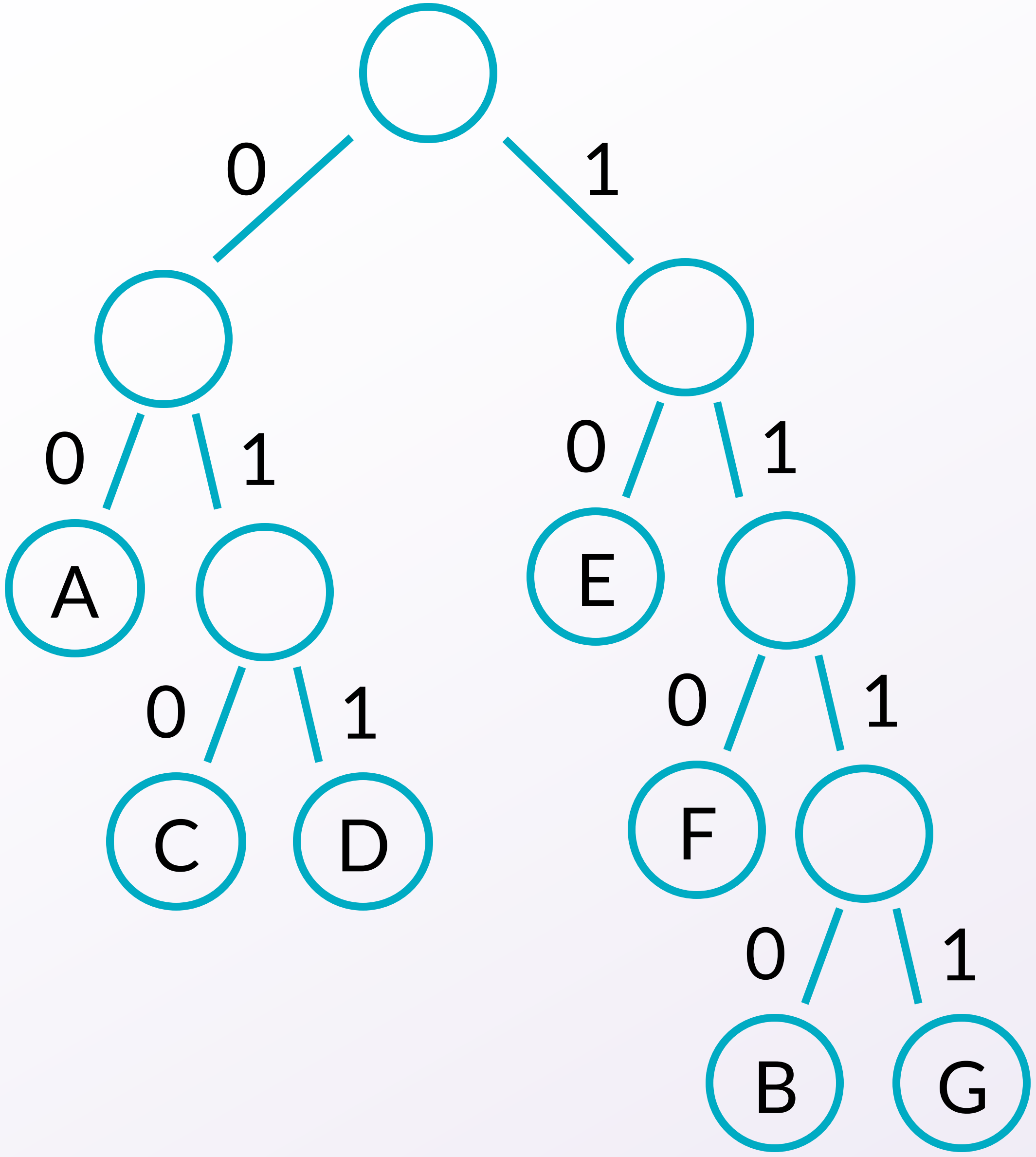


Example of Huffman's idea





(16)
(7)
(9)
A
(3)
(4)
E
(5)
(4)
C
(2)
D
(2)
F
(2)
(2)
B
(1)
G
(1)



Symbol	Codeword
A	00
B	1110
C	010
D	011
E	10
F	110
G	1111

How did Huffman do?

Symbol	Codeword	Count	Frequency
A	00	3	3/16
B	1110	1	1/16
C	010	2	2/16
D	011	2	2/16
E	10	5	5/16
F	110	2	2/16
G	1111	1	1/16

Average length:

2.625 bits/symbol

Recall basic exchange argument

To show that your solution is optimal,

- Consider a different solution. Usually, it helps to consider the **first time that it differs**.
- Show that you can **change the other solution** to be more like yours, while improving or maintaining the quality.
- Conclude that yours is optimal!

Huffman exchange argument

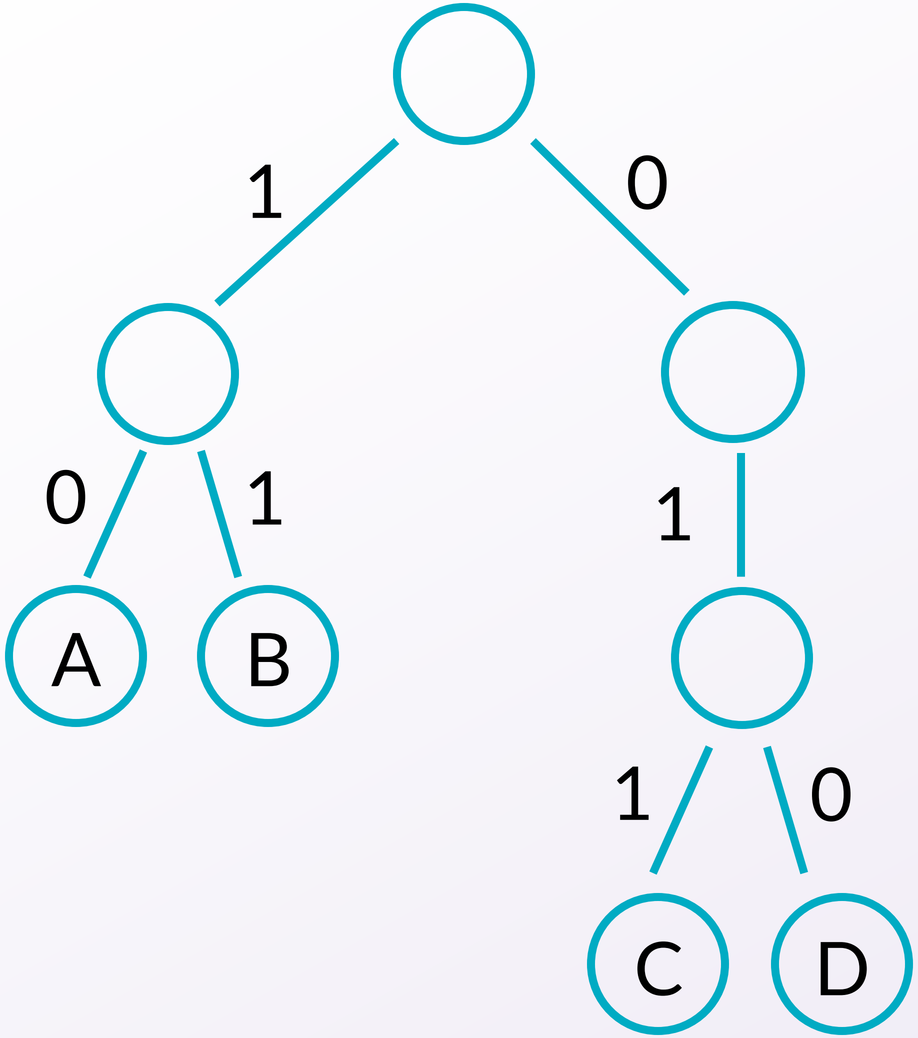
Consider any prefix code that is different from yours.

Every prefix code corresponds to a binary tree.

Two ways the tree could be different:

- It was not a ***full*** binary tree
- It merged nodes in a different order than you.

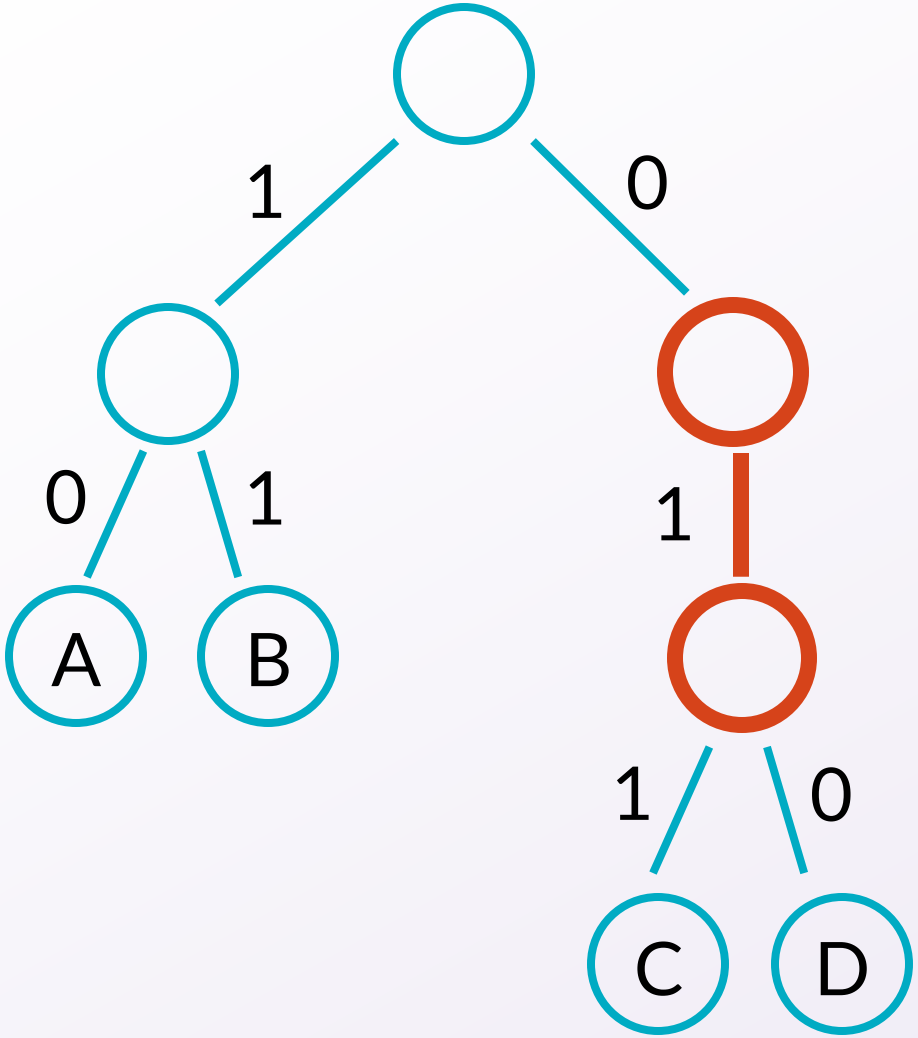
Optimal solutions are full binary trees



Suppose a solution is not a full binary tree.

Q: How can you change it to a better solution?

Optimal solutions are full binary trees

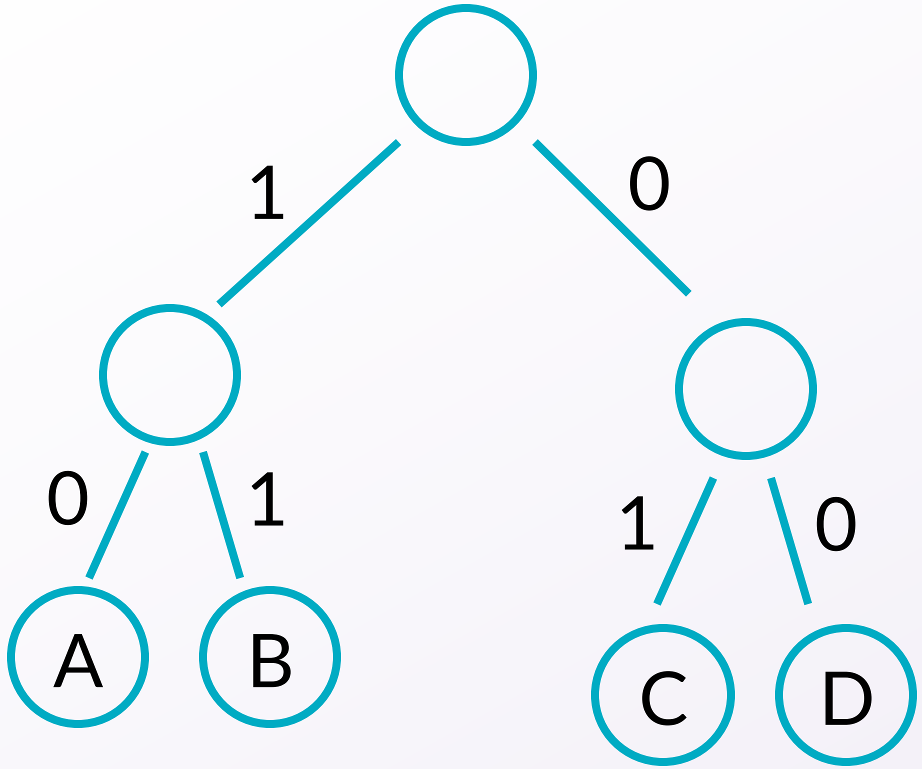


Suppose a solution is not a full binary tree.

Q: How can you change it to a better solution?

A: Contract the nodes with one child!

Optimal solutions are full binary trees



Suppose a solution is not a full binary tree.

Q: How can you change it to a better solution?

A: Contract the nodes with one child!

Codewords can only become shorter when doing this.

Merging nodes in a different order

Claim: There is an optimal prefix code that puts the two least frequent symbols together.

Our exchange strategy:

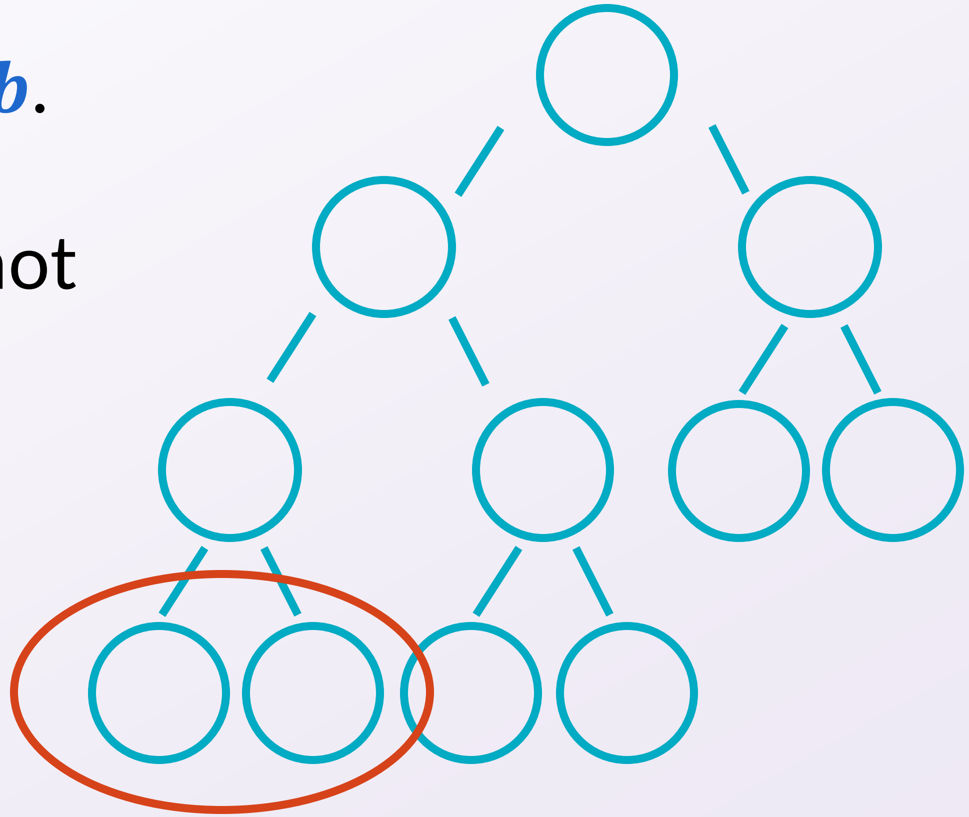
- Look at a prefix code that does not do this.
- Swap in the least frequent symbols, improving the quality.

Merging nodes in a different order

Call the two least frequent symbols $\boldsymbol{a}$ and $\boldsymbol{b}$.

Look at another full binary tree that does not put together $\boldsymbol{a}$ and $\boldsymbol{b}$.

Look at two lowest-depth siblings in this graph.



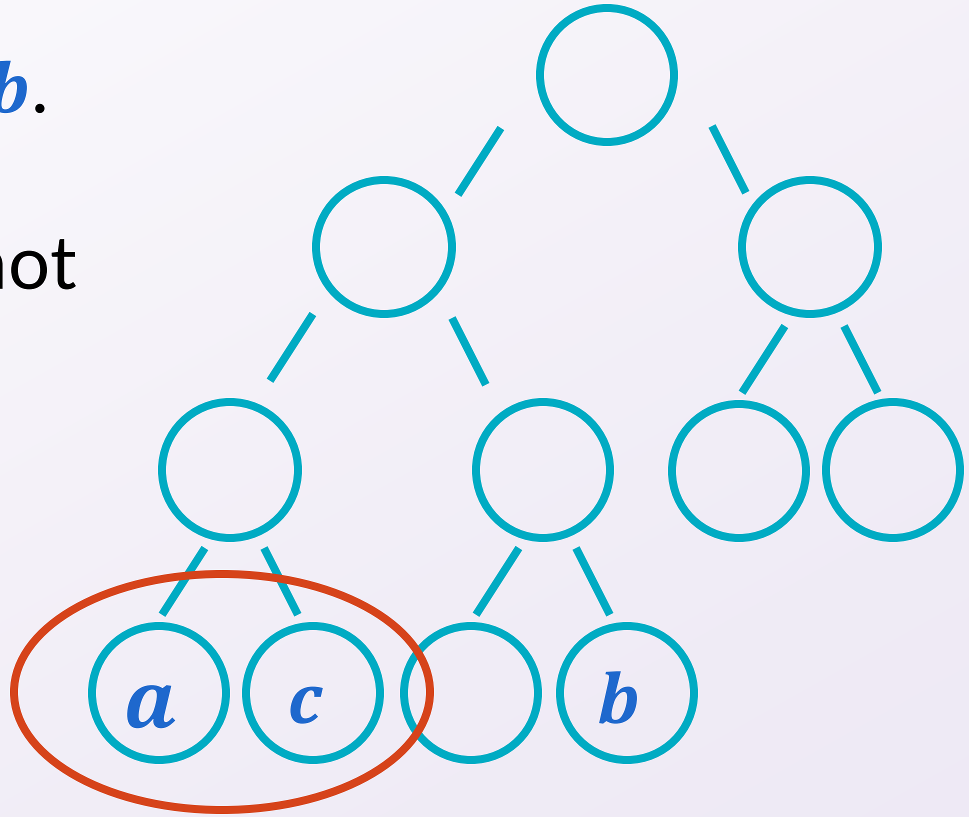
Merging nodes in a different order

Call the two least frequent symbols a and b.

Look at another full binary tree that does not put together a and b.

Look at two lowest-depth siblings in this graph.

Case 1: One of them is a or b.



Merging nodes in a different order

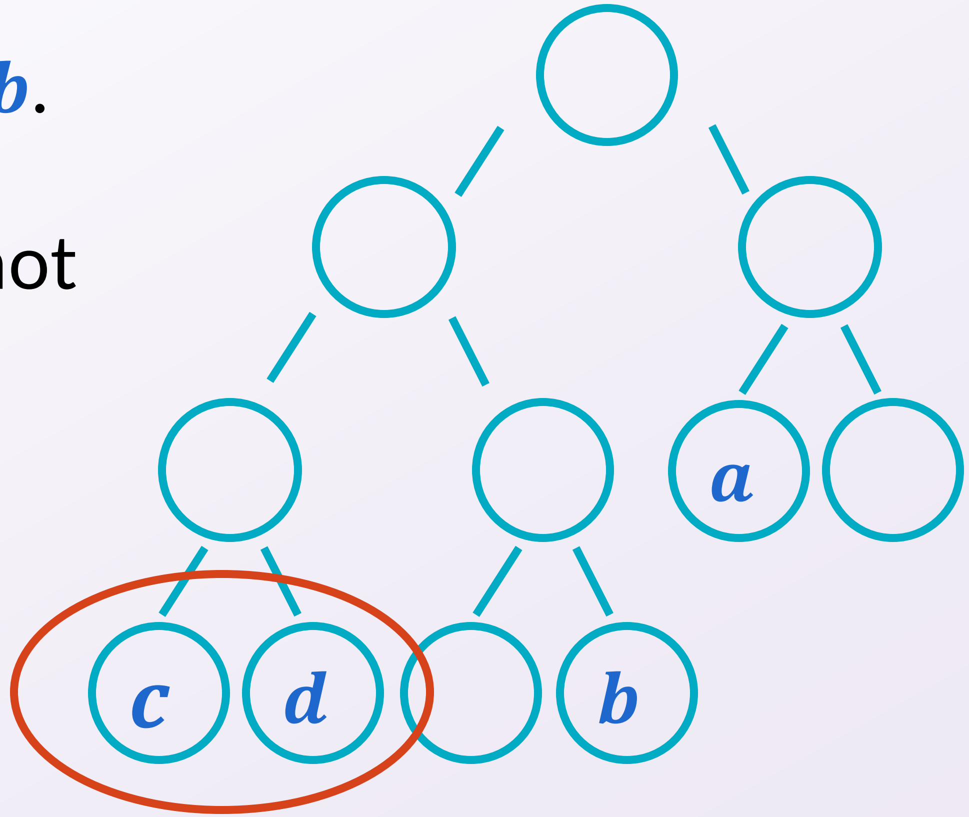
Call the two least frequent symbols a and b.

Look at another full binary tree that does not put together a and b.

Look at two lowest-depth siblings in this graph.

Case 1: One of them is a or b.

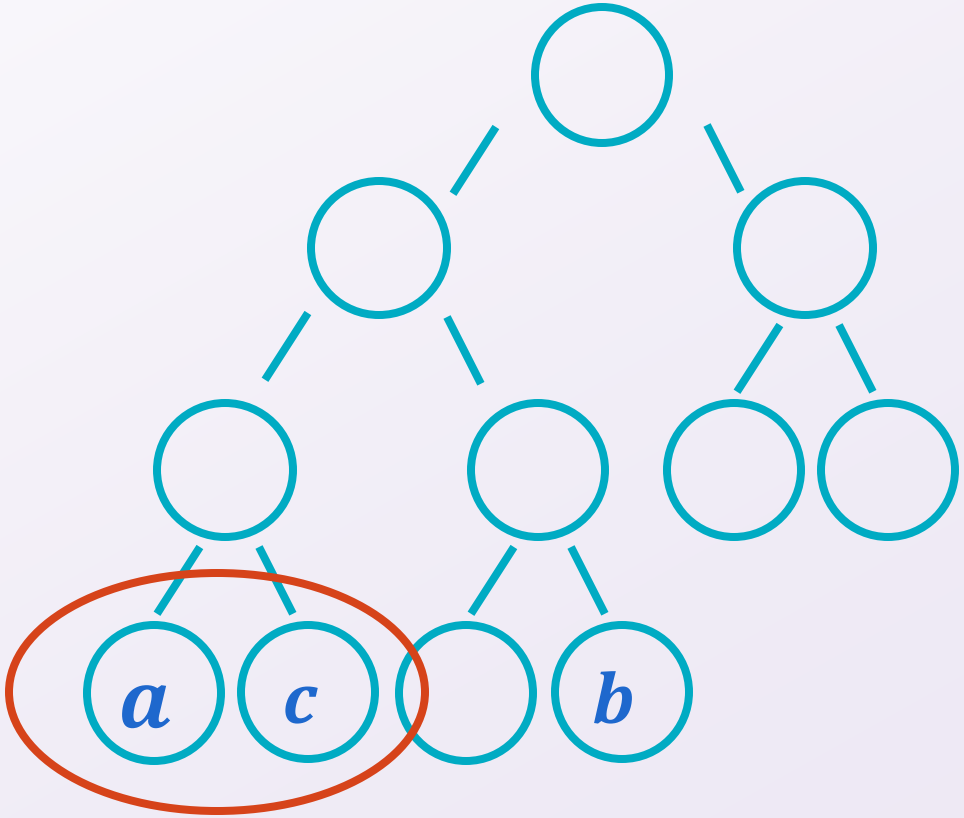
Case 2: Neither is a or b.



Merging nodes in a different order

Case 1: One of them is a or b.

Since c is on the lowest level and is more frequent than b, swapping them can only decrease the average length (or stay same)!



Merging nodes in a different order

Case 1: One of them is $\boldsymbol{a}$ or $\boldsymbol{b}$.

Formally, $\boldsymbol{f_c \geq f_b}$ and currently $\boldsymbol{c_c \geq c_b}$, and they contribute $\boldsymbol{f_c c_c + f_b c_b}$ to the length.

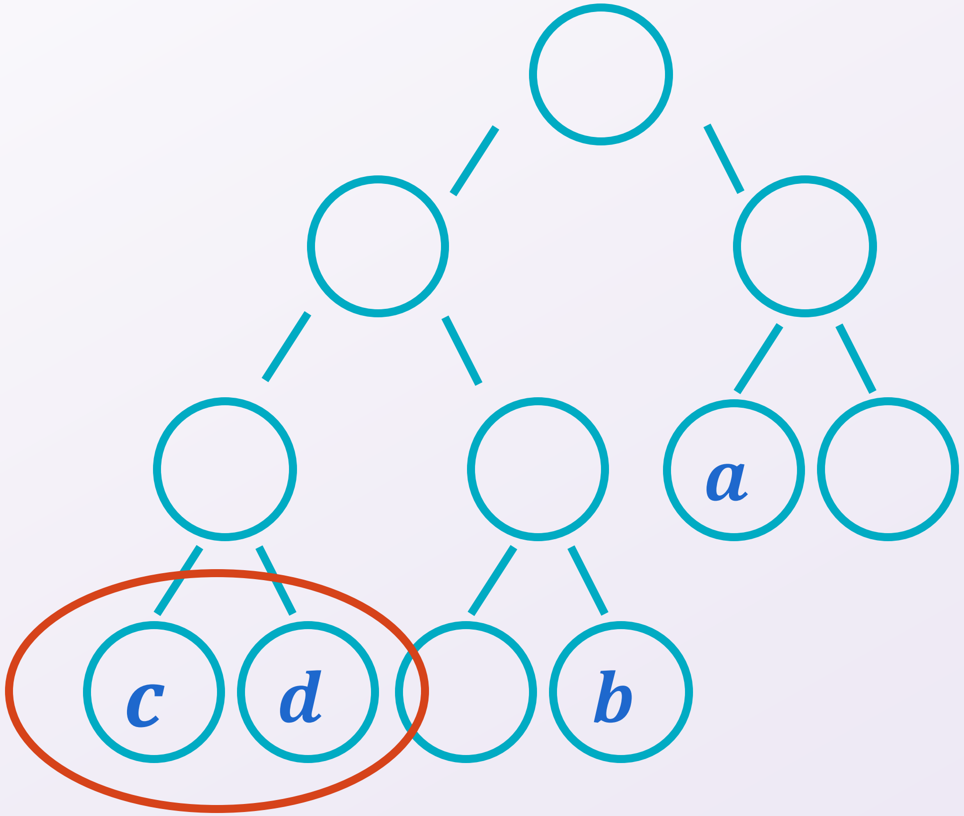
After swapping, they will contribute $\boldsymbol{f_c c_b + f_b c_c}$.

Since $\boldsymbol{0 \leq (f_c - f_b)(c_c - c_b) = (f_c c_c + f_b c_b) - (f_c c_b + f_b c_c)}$, the new quantity is smaller!

Merging nodes in a different order

Case 2: Neither is a or b.

We swap both with the same argument as Case 1 both times!



Final reminders

I have OH now-12:30pm:

- Meet at front of classroom, we'll walk over together
- CSE (Allen) 214 if you're coming later

Nathan has online OH 12–1pm:

- https://washington.zoom.us/my/nathanbrunelle