CS 3100
Data Structures and Algorithms 2
Lecture 15: Huffman Encoding

Co-instructors: Robbie Hott and Tom Horton
Fall 2023

Readings in CLRS 4th edition:
- Chapter 16

Announcements

- Upcoming dates
    - PS3 (Greedy Algorithms) due October 20, 2023 at 11:59pm
    - PA3 (Clustering) due October 29, 2023 at 11:59pm

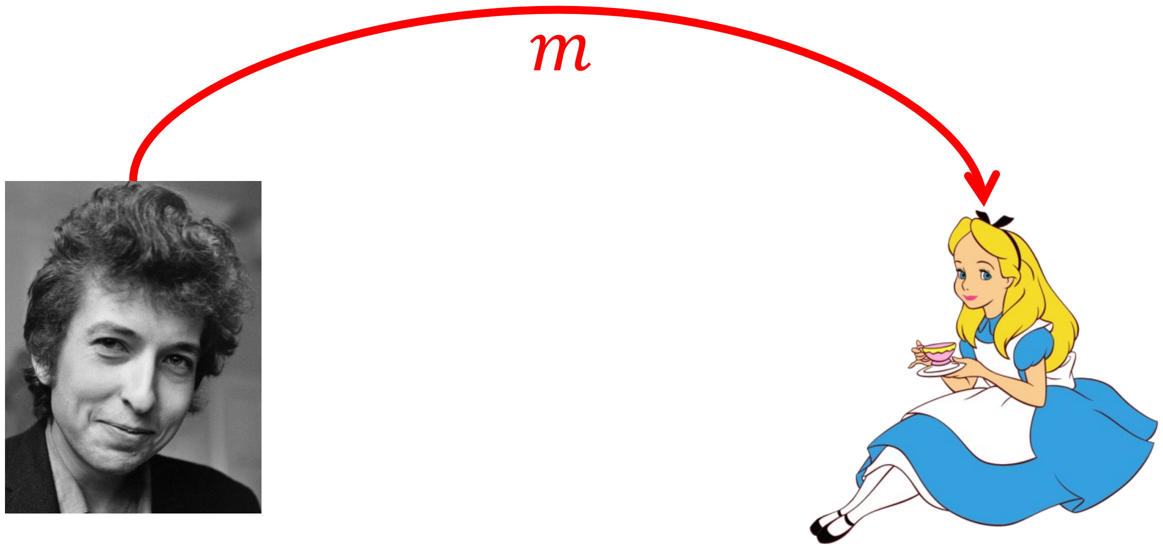
- Course email (comes to both professors and head TAs):

cs3100@cshelpdesk.atlassian.net

Message Encoding

Problem: need to electronically send a message to two people at a distance.

Channel for message is binary (either on or off)



How efficient is this?

wiggle wiggle wiggle like a gypsy queen
wiggle wiggle wiggle all dressed in green

Each character requires 4 bits

$$\ell_c = 4$$

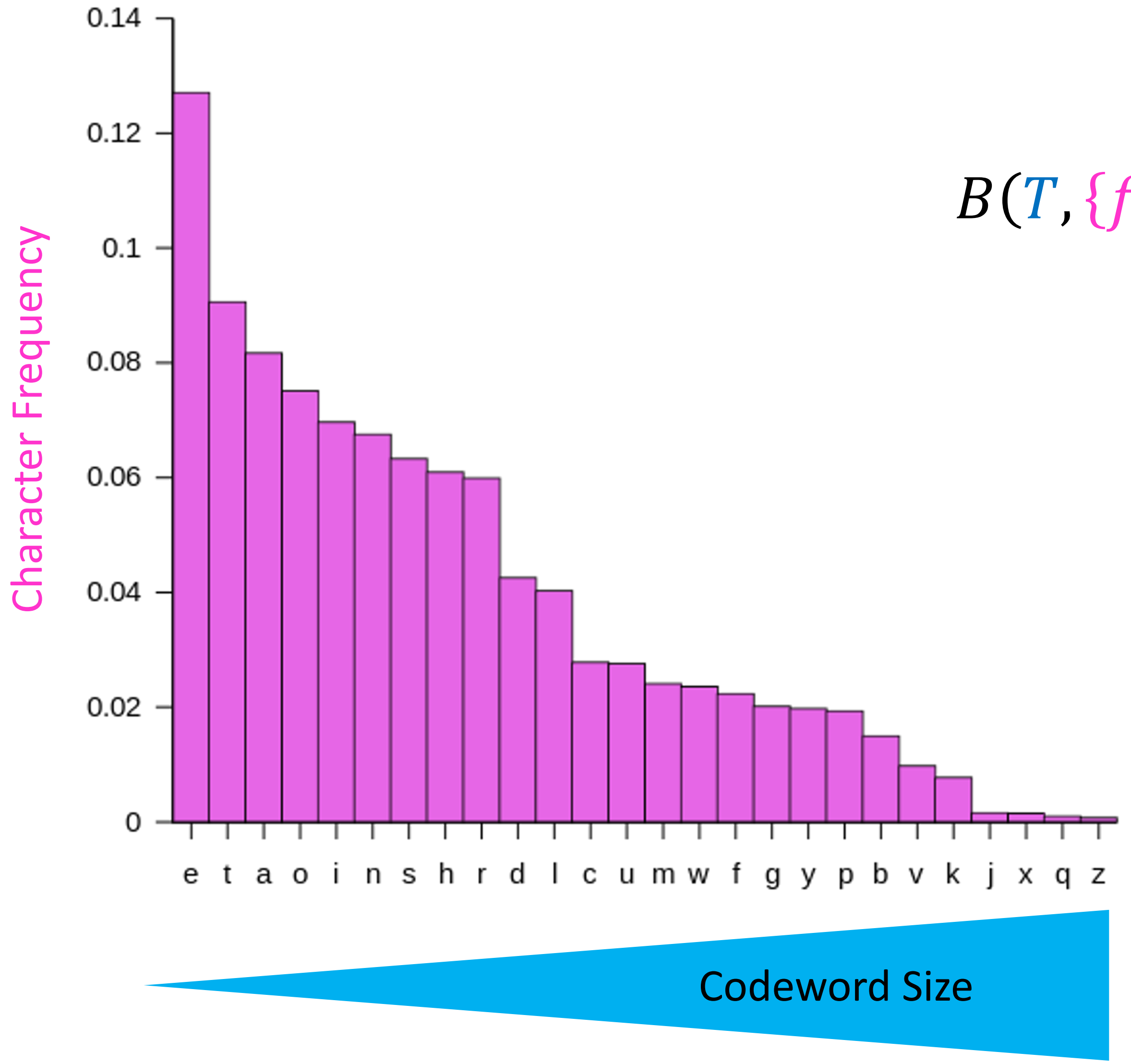
Cost of encoding:

$$B(T, \{f_c\}) = \sum_{character\ c} \ell_c f_c = 68 \cdot 4 = 272$$

Better Solution: Allow for different characters to have different-size encodings (high frequency → short code)

Character Frequency	Encoding
a: 2	0000
d: 2	0001
e: 13	0010
g: 14	0011
i: 8	0100
k: 1	0101
l: 9	0110
n: 3	0111
p: 1	1000
q: 1	1001
r: 2	1010
s: 3	1011
u: 1	1100
w: 6	1101
y: 2	1110

More efficient coding



When this is big

$$B(T,\{f_c\}) = \sum_{\text{character } c} \ell_c f_c$$

Make this small

Prefix-Free Code

A prefix-free code is codeword table T such that for any two characters c_1, c_2, if $c_1 \neq c_2$ then $code(c_1)$ is not a prefix of $code(c_2)$

g	0
e	10
l	110
i	1110
w	11110
…	…

1111011100011010

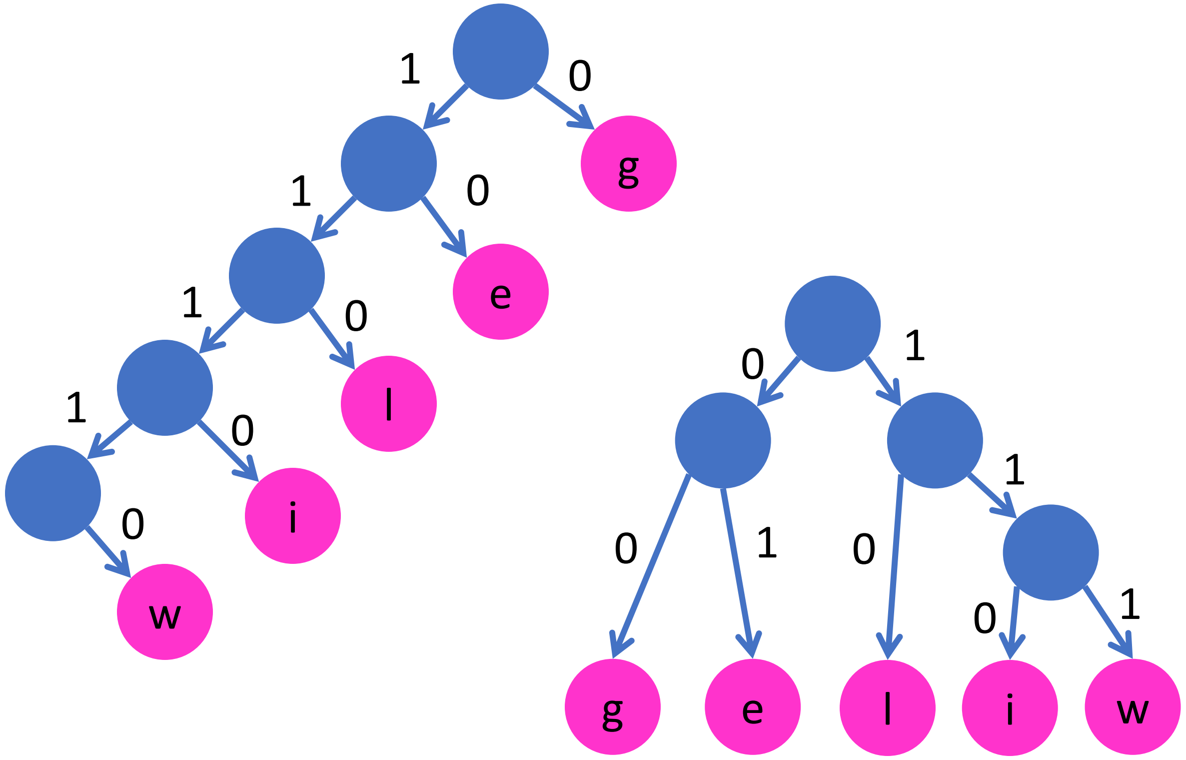
w i gg l e

Binary Trees = Prefix-free Codes

I can represent any prefix-free code as a binary tree

I can create a prefix-free code from any binary tree

g	0
e	10
l	110
i	1110
w	11110
…	…



g	00
e	01
l	10
i	110
w	111
…	…

Goal: Shortest Prefix-Free Encoding

Input: A set of character frequencies $\{f_c\}$

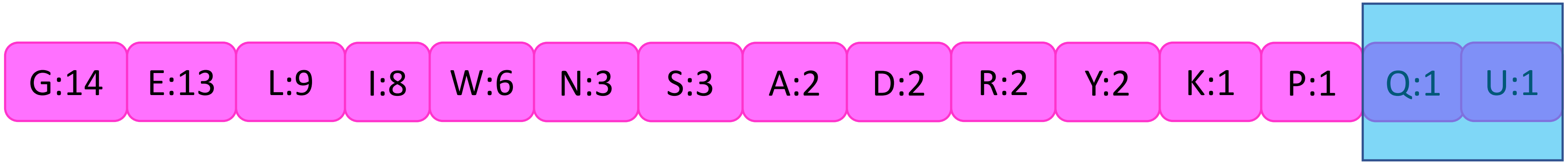
Output: A prefix-free code T which minimizes

$$B(T, \{f_c\}) = \sum_{character\ c} \ell_c f_c$$

Huffman Coding!!

Huffman Algorithm

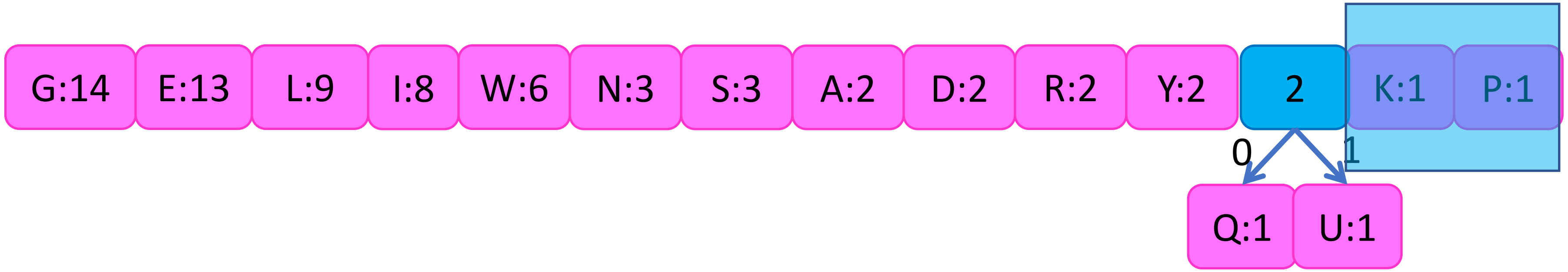
Choose the least frequent pair, combine into a subtree



G:14 E:13 L:9 I:8 W:6 N:3 S:3 A:2 D:2 R:2 Y:2 K:1 P:1 Q:1 U:1

Huffman Algorithm

Choose the least frequent pair, combine into a subtree



G:14 E:13 L:9 I:8 W:6 N:3 S:3 A:2 D:2 R:2 Y:2 2 K:1 P:1

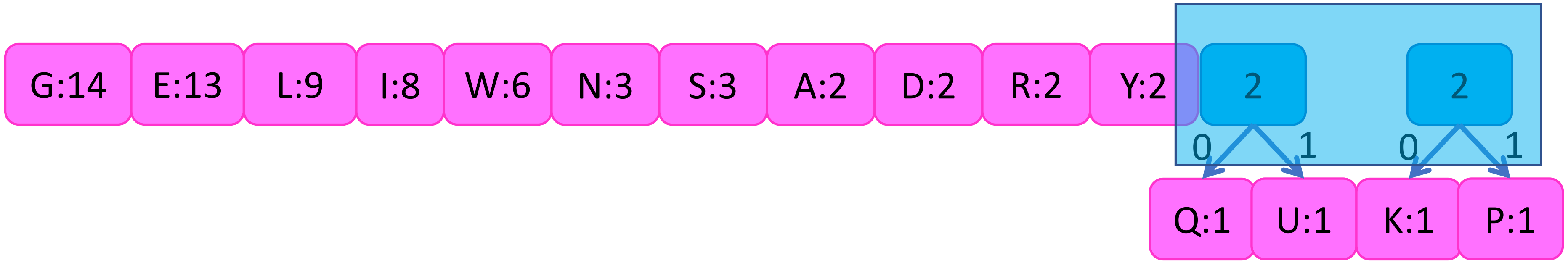
0 1

Q:1 U:1

Subproblem of size $n - 1$!

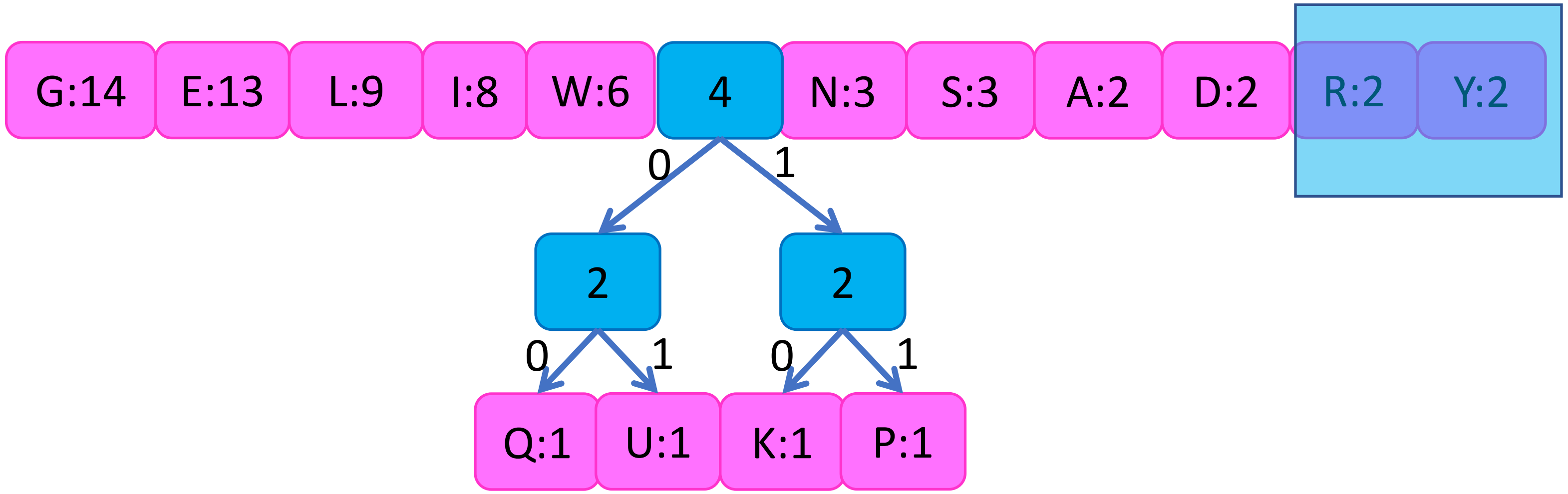
Huffman Algorithm

Choose the least frequent pair, combine into a subtree



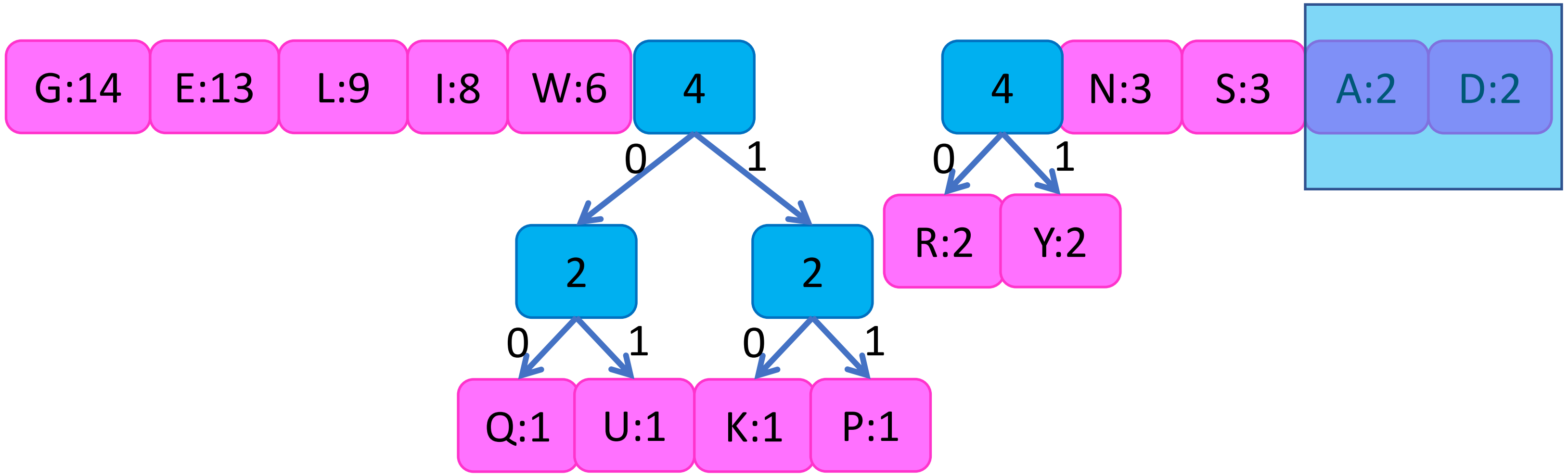
Huffman Algorithm

Choose the least frequent pair, combine into a subtree



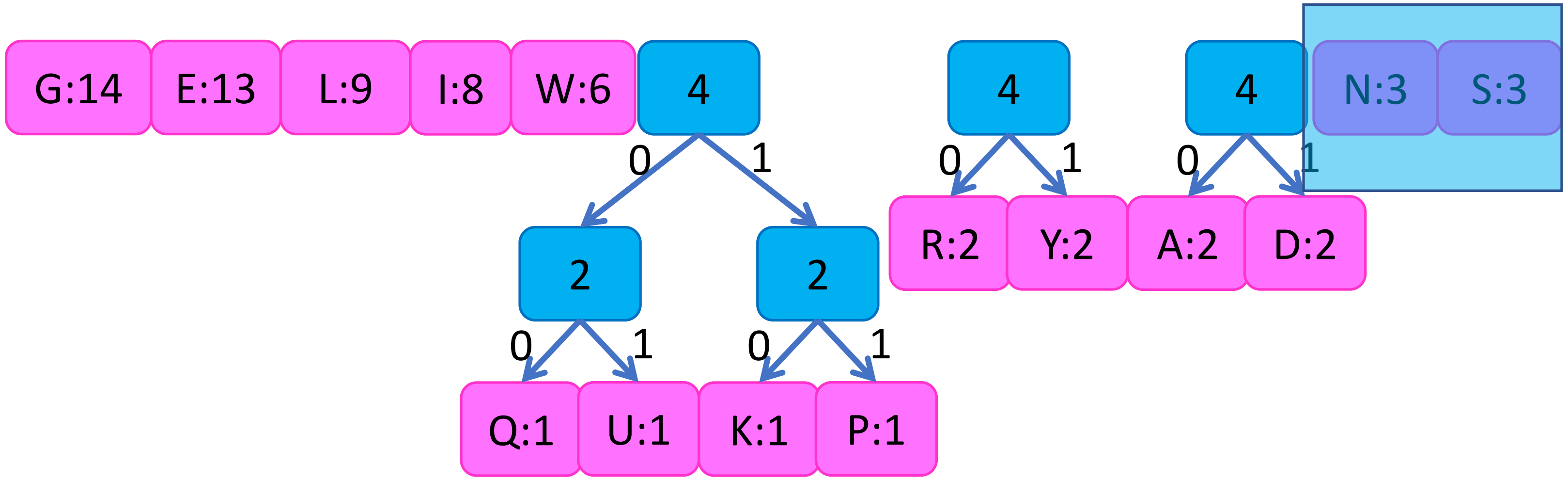
Huffman Algorithm

Choose the least frequent pair, combine into a subtree



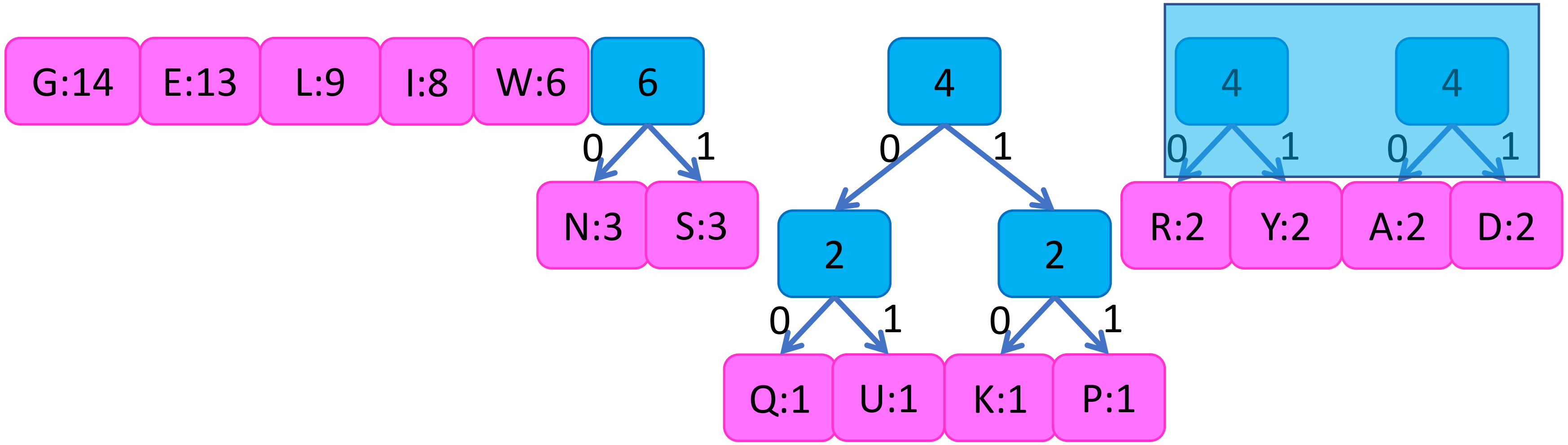
Huffman Algorithm

Choose the least frequent pair, combine into a subtree



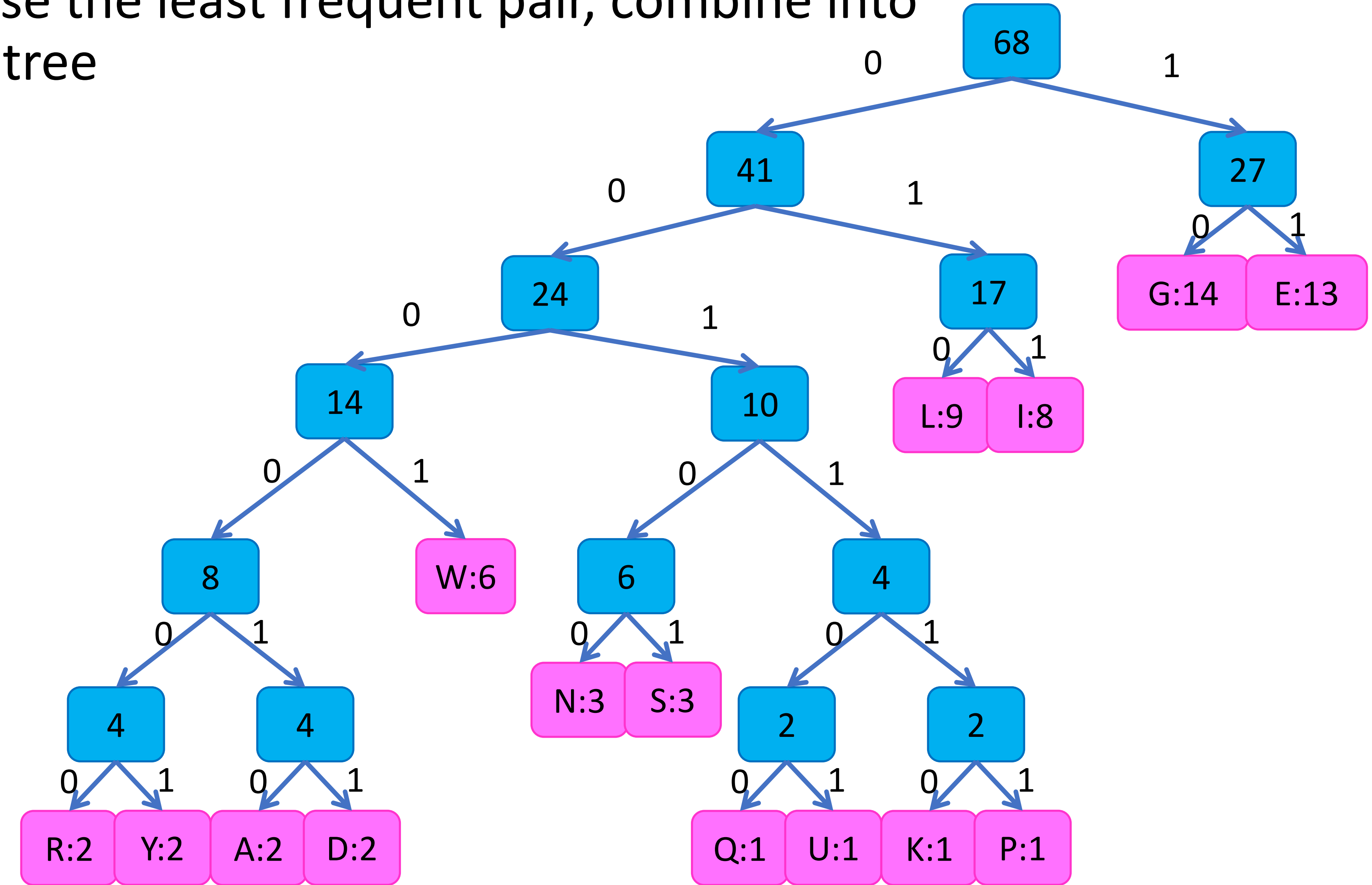
Huffman Algorithm

Choose the least frequent pair, combine into a subtree



Huffman Algorithm

Choose the least frequent pair, combine into a subtree



Exchange argument

Shows correctness of a greedy algorithm

Idea:

- Show exchanging an item from an arbitrary optimal solution with your greedy choice makes the new solution no worse
- How to show my sandwich is at least as good as yours:
    - Show: “I can remove any item from your sandwich, and it would be no worse by replacing it with the same item from my sandwich”

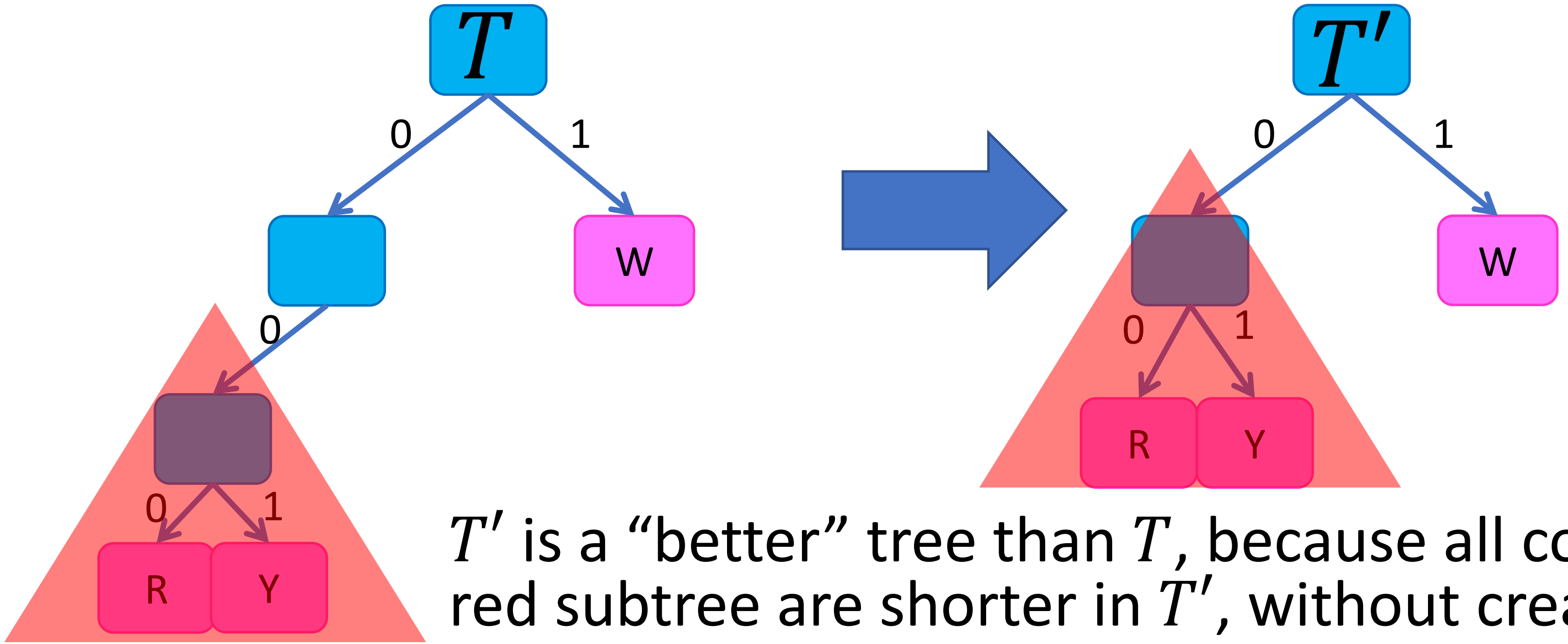
Showing Huffman is Optimal

Overview:

- Show that there is **an** optimal tree in which the least frequent characters are siblings
  - Exchange argument
- Show that making them siblings and solving the new smaller sub-problem results in **an** optimal solution
  - Optimal Substructure argument

Showing Huffman is Optimal

First Step: Show any optimal tree is “full” (each node has either 0 or 2 children)



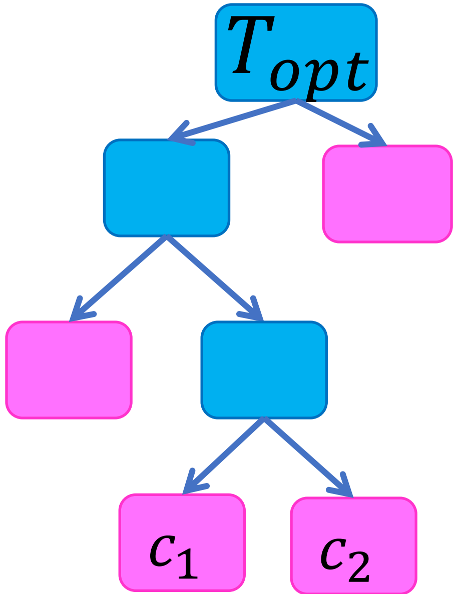
T' is a “better” tree than T, because all codes in red subtree are shorter in T', without creating any longer codes

Huffman Exchange Argument

Claim: if c_1, c_2 are the least-frequent characters, then there is an optimal prefix-free code s.t. c_1, c_2 are siblings

- i.e. codes for c_1, c_2 are the same length and differ only by their last bit

Case 1: Consider some optimal tree T_{opt}. If c_1, c_2 are siblings in this tree, then claim holds

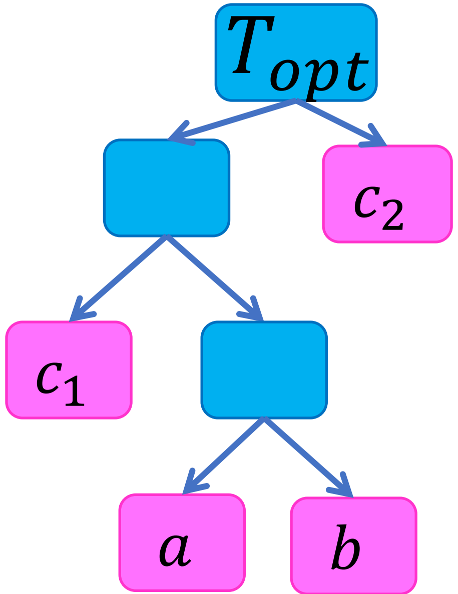


Huffman Exchange Argument

Claim: if c_1, c_2 are the least-frequent characters, then there is an optimal prefix-free code s.t. c_1, c_2 are siblings

- i.e. codes for c_1, c_2 are the same length and differ only by their last bit

Case 2: Consider some optimal tree T_{opt}, in which c_1, c_2 are not siblings



Let a, b be the two characters of lowest depth that are siblings
(Why must they exist?)

Idea: show that swapping c_1 with a does not increase cost of the tree.
Similar for c_2 and b
Assume: $f_{c1} \leq f_a$ and $f_{c2} \leq f_b$

Case 2: c_1, c_2 are not siblings in T_{opt}

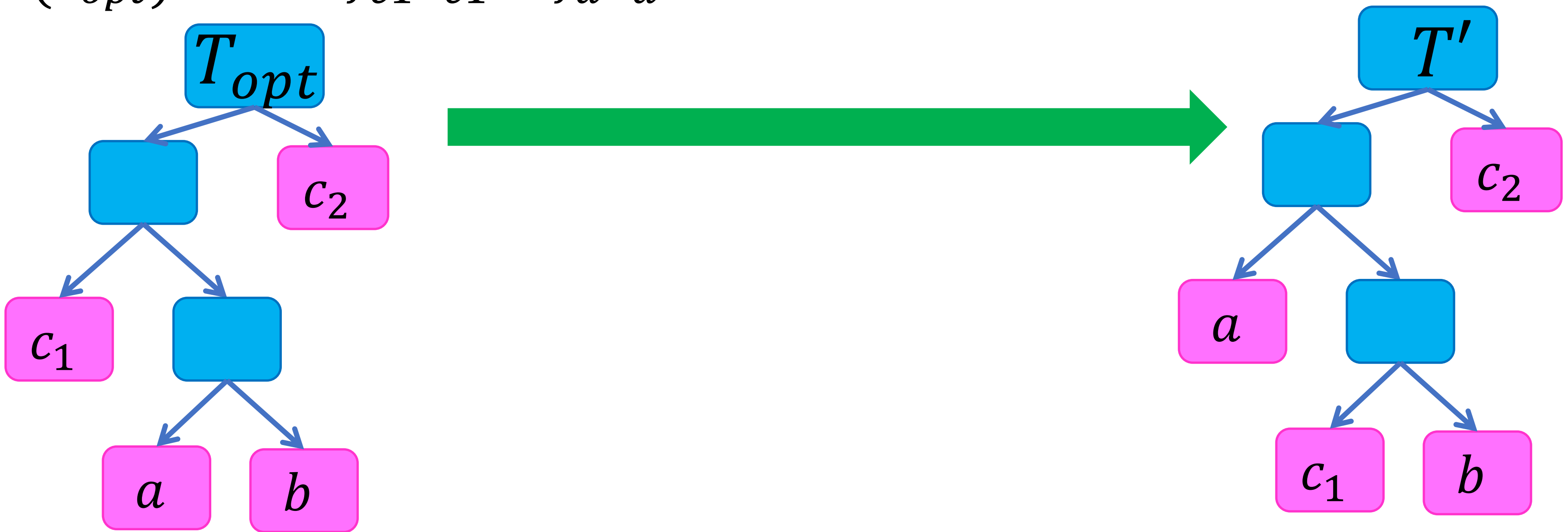
- **Claim**: the least-frequent characters (c_1, c_2), are siblings in some optimal tree

$a, b =$ lowest-depth siblings

Idea: show that swapping c_1 with a does not increase cost of the tree.
Assume: $f_{c1} \le f_a$

$B(T_{opt}) = C + f_{c1}\ell_{c1} + f_a\ell_a$

$B(T') = C + f_{c1}\ell_a + f_a\ell_{c1}$



Case 2: c_1, c_2 are not siblings in T_{opt}

- **Claim**: the least-frequent characters (c_1, c_2), are siblings in some optimal tree

$a, b =$ lowest-depth siblings

Idea: show that swapping c_1 with a does not increase cost of the tree.
Assume: $f_{c1} \leq f_a$

$$B(T_{opt}) = C + f_{c1}\ell_{c1} + f_a\ell_a$$

$$B(T') = C + f_{c1}\ell_a + f_a\ell_{c1}$$

$\geq 0 \Rightarrow T'$ optimal

$$B(T_{opt}) - B(T') = C + f_{c1}\ell_{c1} + f_a\ell_a - (C + f_{c1}\ell_a + f_a\ell_{c1})$$
$$= f_{c1}\ell_{c1} + f_a\ell_a - f_{c1}\ell_a - f_a\ell_{c1}$$
$$= f_{c1}(\ell_{c1} - \ell_a) + f_a(\ell_a - \ell_{c1})$$
$$= (f_a - f_{c1})(\ell_a - \ell_{c1})$$

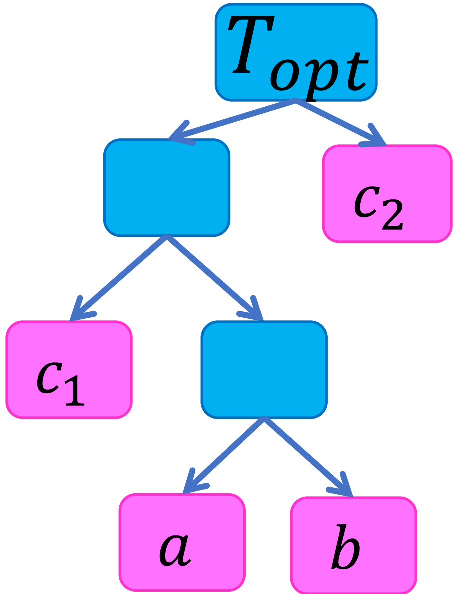
Case 2: c_1, c_2 are not siblings in T_{opt}

- **Claim**: the least-frequent characters (c_1, c_2), are siblings in some optimal tree

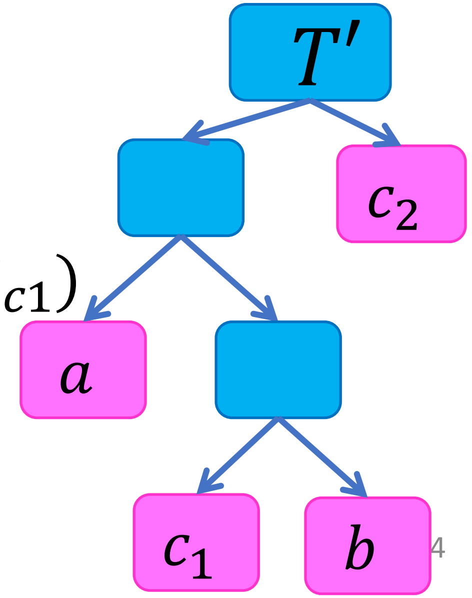
a, b = lowest-depth siblings

Idea: show that swapping c_1 with a does not increase cost of the tree.
Assume: $f_{c1} \leq f_a$

$$B(T_{opt}) = C + f_{c1}\ell_{c1} + f_a\ell_a$$



$$B(T') = C + f_{c1}\ell_a + f_a\ell_{c1}$$



$$B(T_{opt}) - B(T') = \underset{\geq 0}{(f_a - f_{c1})}\underset{\geq 0}{(\ell_a - \ell_{c1})}$$

$$B(T_{opt}) - B(T') \geq 0$$

T' is also optimal!

Case 2:Repeat to swap c_2, b!

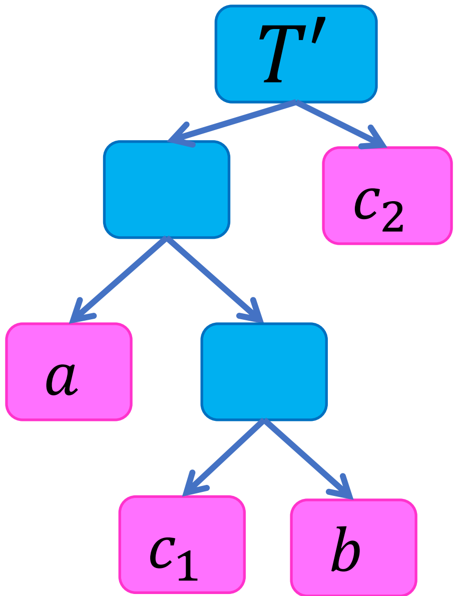
- Claim: the least-frequent characters (c_1, c_2), are siblings in some optimal tree

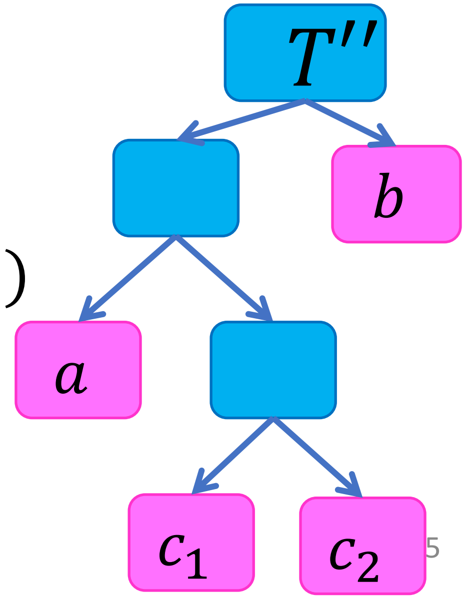
$a, b =$ lowest-depth siblings

Idea: show that swapping c_2 with b does not increase cost of the tree.
Assume: $f_{c2} \leq f_b$

$$B(T') = C + f_{c2}\ell_{c2} + f_b\ell_b$$

$$B(T'') = C + f_{c2}\ell_b + f_b\ell_{c2}$$





$$B(T') - B(T'') = \underbrace{(f_b - f_{c2})}_{\geq 0}\underbrace{(\ell_b - \ell_{c2})}_{\geq 0}$$

$$B(T') - B(T'') \geq 0$$

T'' is also optimal! Claim holds!

Showing Huffman is Optimal

Overview:

- Show that there is **an** optimal tree in which the least frequent characters are siblings
  - Exchange argument
- Show that making them siblings and solving the new smaller sub-problem results in **an** optimal solution
  - Optimal Substructure argument

Proving Optimal Substructure

Goal: show that if x is in an optimal solution, then the rest of the solution is an optimal solution to the subproblem.

Usually by Contradiction:

- Assume that x must be an element of my optimal solution
- Assume that solving the subproblem induced from choice x, then adding in x is not optimal
- Show that removing x from a better overall solution must produce a better solution to the subproblem

Huffman Optimal Substructure

Goal: show that if c_1, c_2 are siblings in an optimal solution, then an optimal prefix free code can be found by using a new character with frequency $f_{c_1} + f_{c_2}$ and then making c_1, c_2 its children.

By Contradiction:

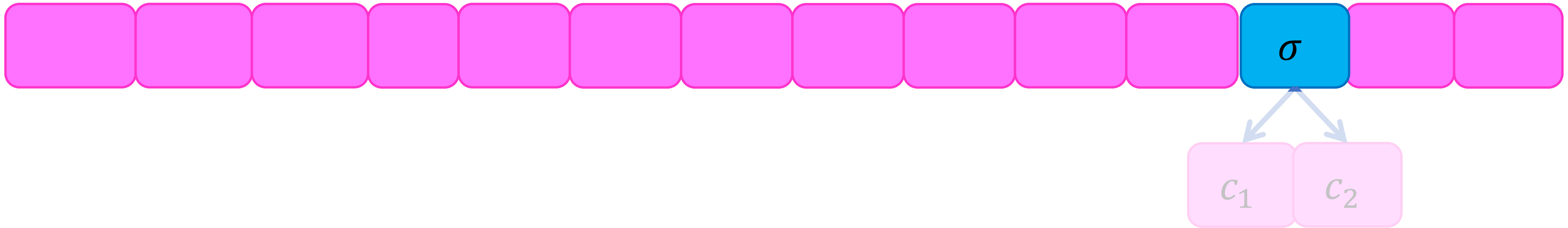
- Assume that c_1, c_2 are siblings in at least one optimal solution
- Assume that solving the subproblem with this new character, then adding in c_1, c_2 is not optimal
- Show that removing c_1, c_2 from a better overall solution must produce a better solution to the subproblem

Finishing the Proof

Show Recursive Substructure

- Show treating c_1, c_2 as a new “combined” character gives optimal solution

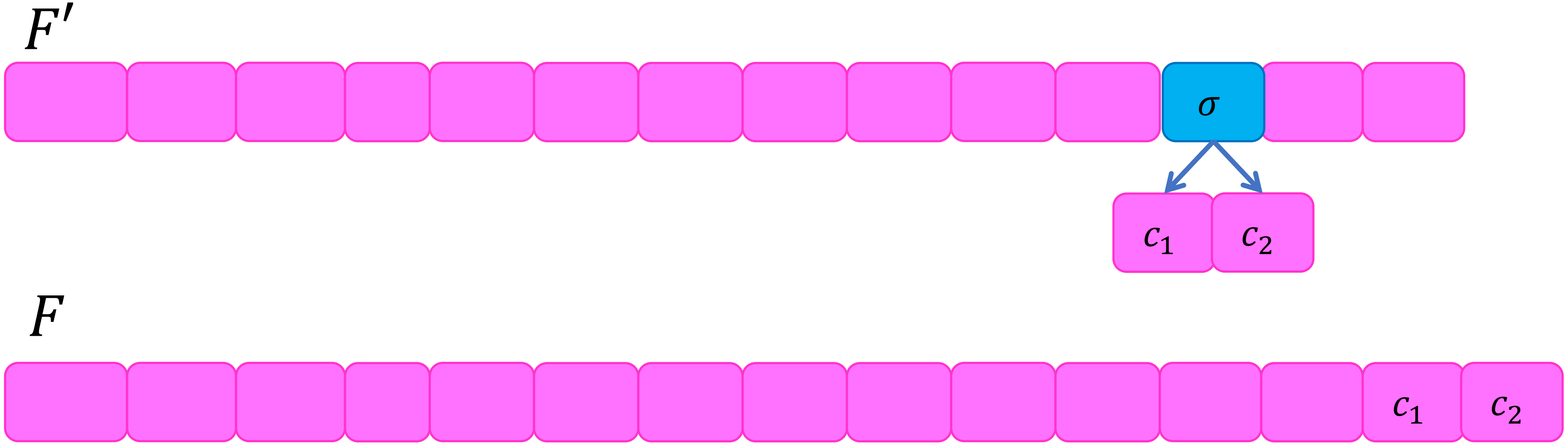
Why does solving this smaller problem:



Give an optimal solution to this?:

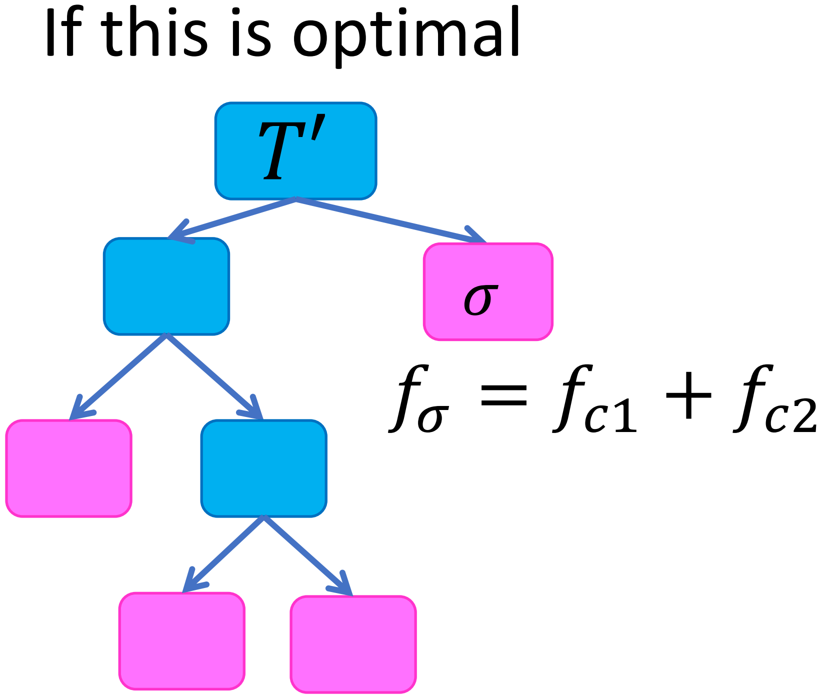
Substructure

Claim: An optimal solution for F involves finding an optimal solution for F', then adding c_1, c_2 as children to σ

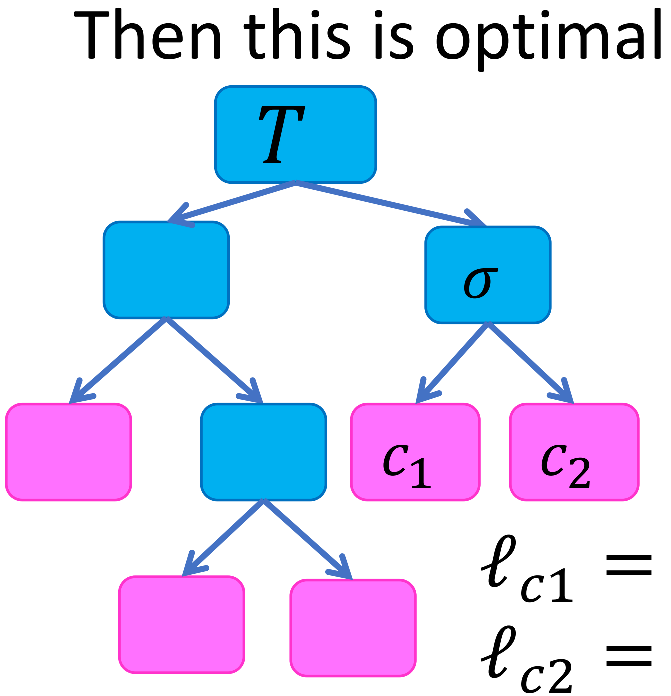


Substructure

Claim: An optimal solution for F involves finding an optimal solution for F', then adding c_1, c_2 as children to σ



$$f_\sigma = f_{c1} + f_{c2}$$



$$\ell_{c1} = \ell_\sigma + 1$$
$$\ell_{c2} = \ell_\sigma + 1$$

$$B(T') = B(T) - f_{c1} - f_{c2}$$

Substructure

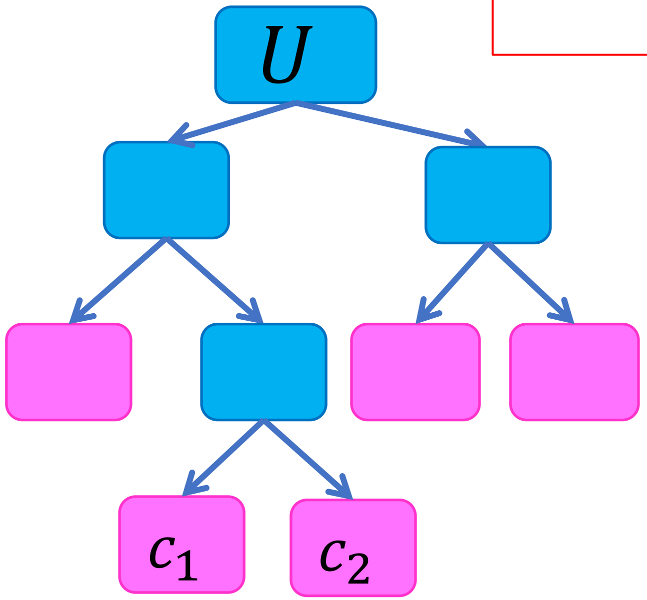
Claim: An optimal solution for F involves finding an optimal solution for F', then adding c_1, c_2 as children to σ

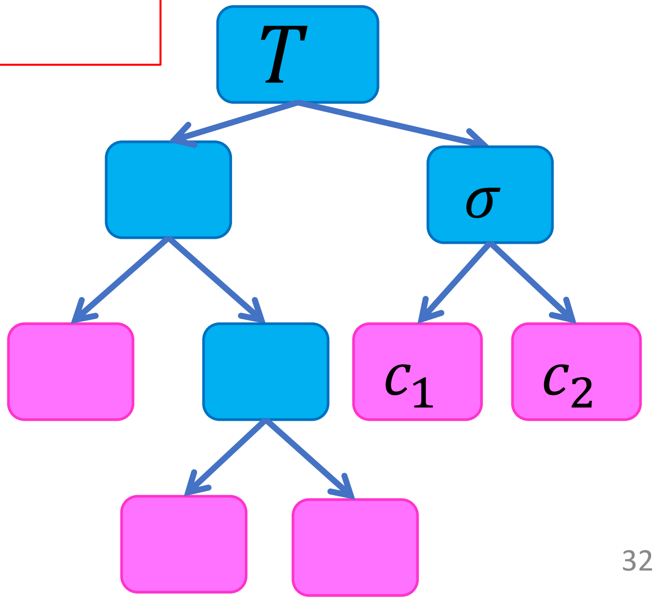
Toward contradiction

Suppose T is not optimal

Let U be a lower-cost tree

$$B(U) < B(T)$$



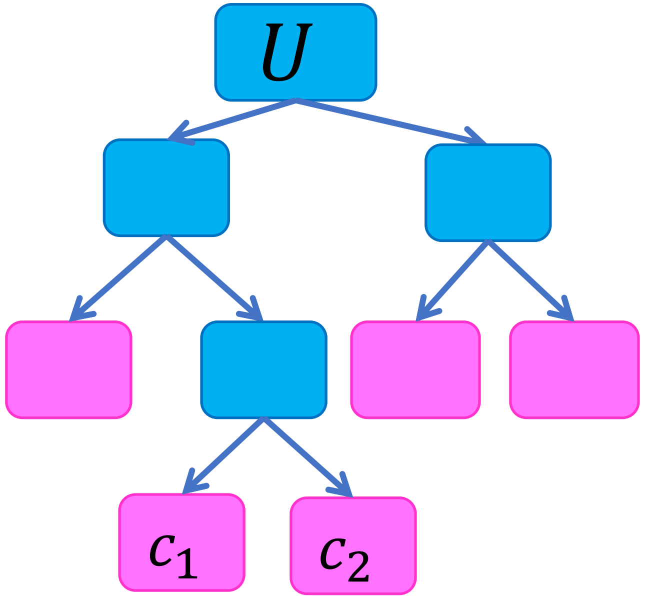


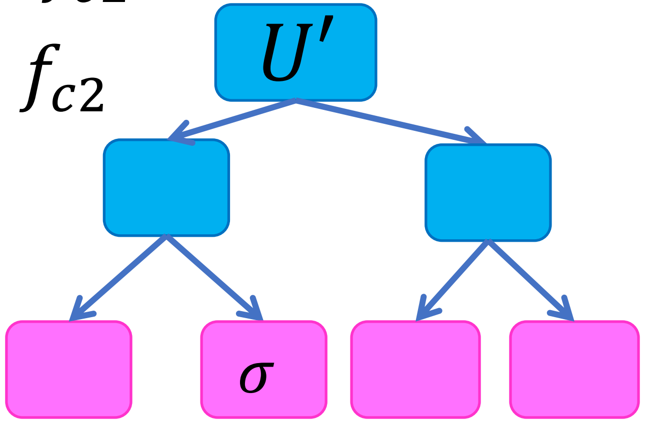
Substructure

Claim: An optimal solution for F involves finding an optimal solution for F', then adding c_1, c_2 as children to σ

$$B(U) < B(T)$$

$$\begin{aligned} B(U') &= B(U) - f_{c1} - f_{c2} \\ &< B(T) - f_{c1} - f_{c2} \\ &= B(T') \end{aligned}$$

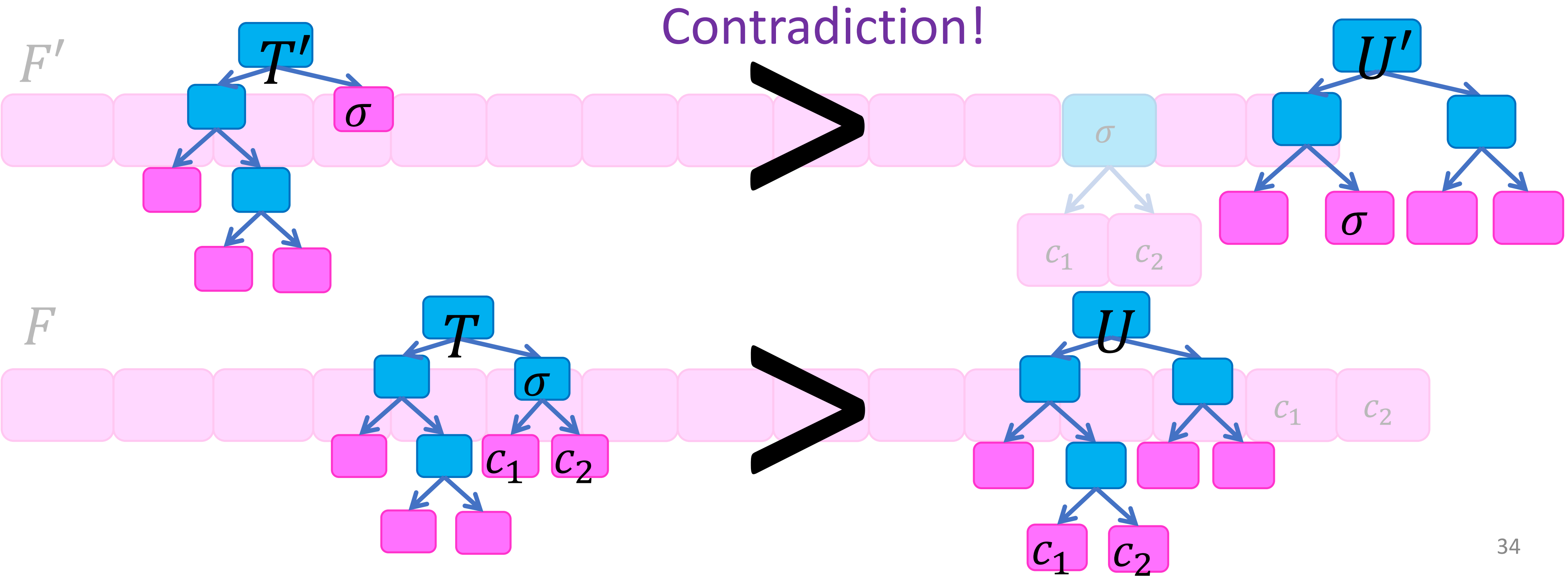




Contradicts optimality of T', so T is optimal!

Optimal Substructure

Claim: An optimal solution for F involves finding an optimal solution for F', then adding c_1, c_2 as children to σ



Bridge Crossing

Bridge Crossing

n friends need to cross a bridge in the dark, but only have one flashlight. In addition, the bridge can only hold the weight of two people at a time. Given the walking speeds of each person $S = \{s_1, s_2, \dots, s_n\}$, give an algorithm that gets all *n* people across the bridge as quickly as possible.

***Assume* $s_1 \leq s_2 \leq \cdots \leq s_n$

***If two people cross together, they walk at the slower person's speed*