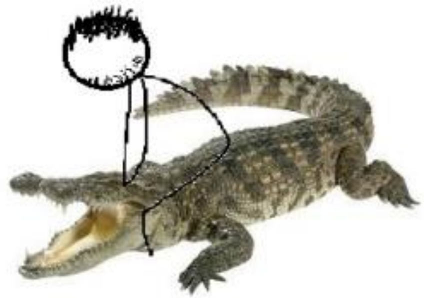
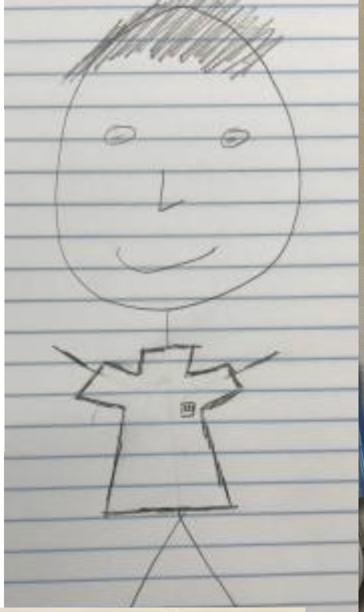
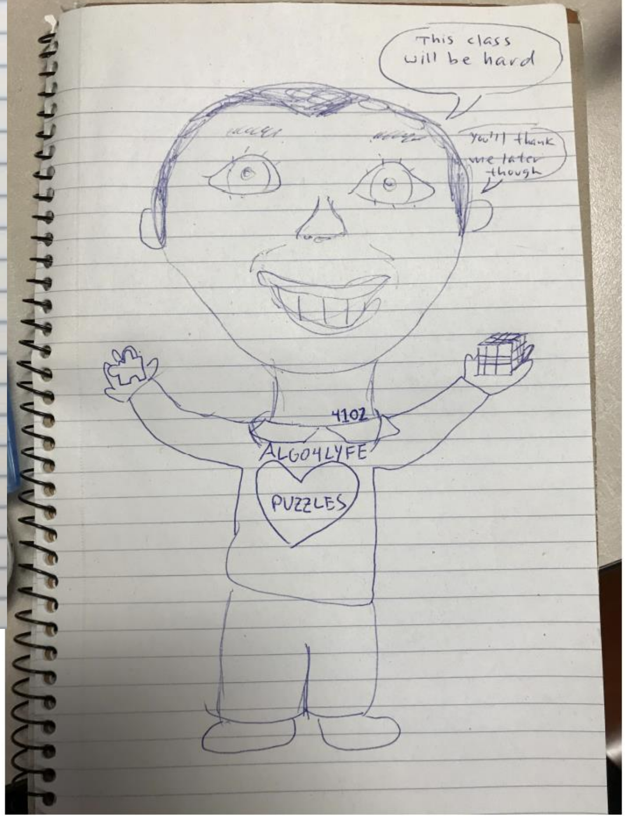
CS4102 Algorithms

Nate Brunelle

Spring 2018







Today's Keywords

- Greedy Algorithms
- Choice Function
- Prefix-free code
- Compression
- Huffman Code

CLRS Readings

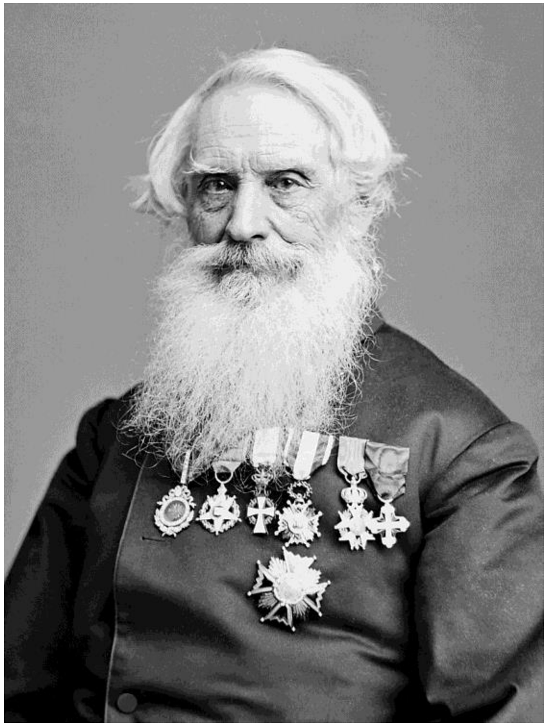
- Chapter 16

Homeworks

- HW6 Due Friday April 13 (Spooky!)
  - Written (use latex)
  - DP and Greedy

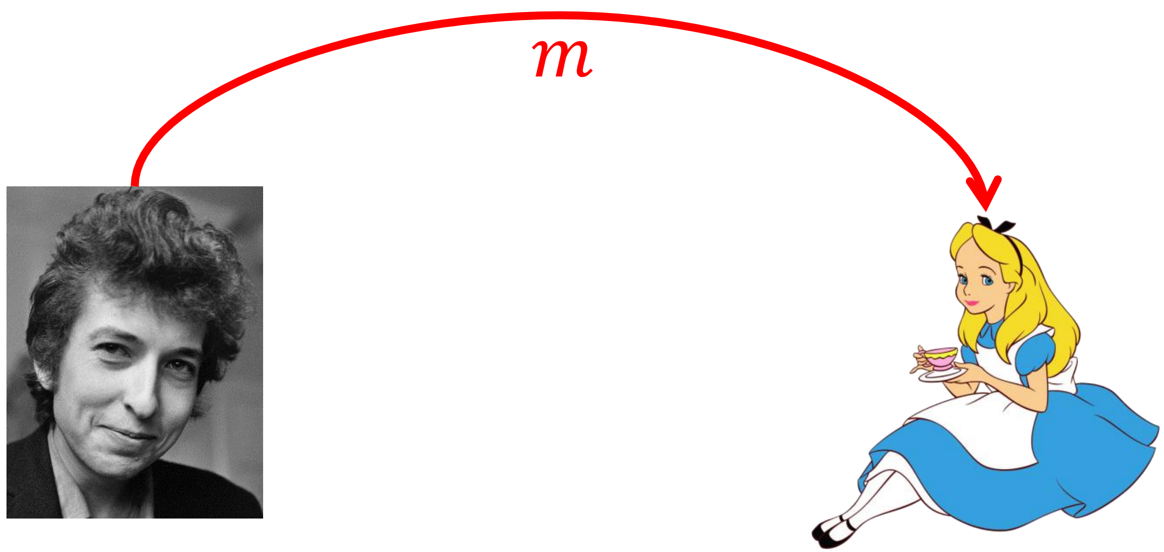
Sam Morse

- Engineer and artist



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 can we do it?

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

- Take the message, send it over character-by character with an encoding

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

How efficient is this?

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

Each character requires 4 bits

$$\ell_c = 4$$

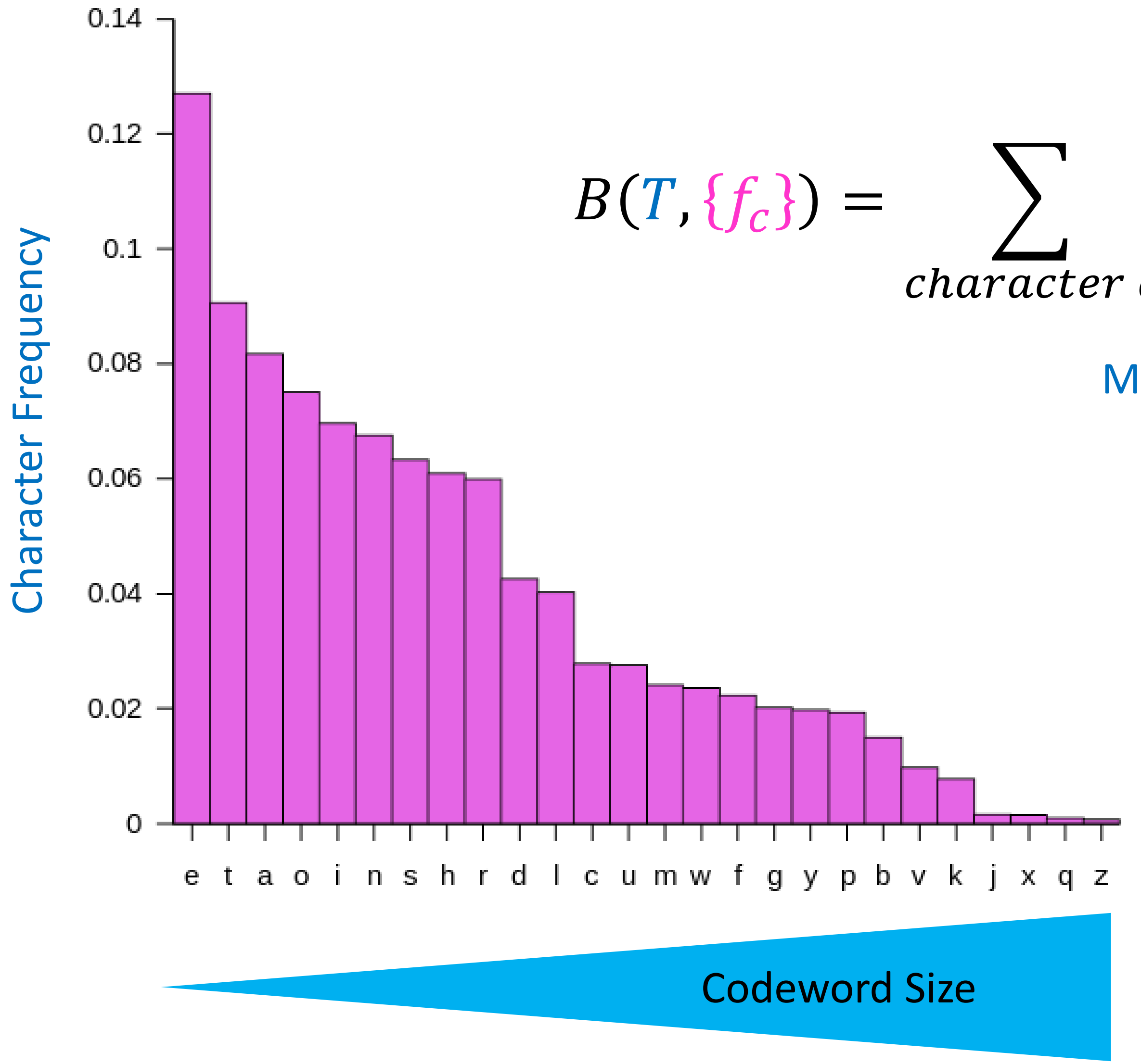
Cost of encoding:

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

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

Character Frequency f_c	Encoding Table T
a: 2	0001
d: 2	0010
e: 13	0011
g: 14	0100
i: 8	0101
k: 1	0110
l: 9	0111
n: 3	1000
p: 1	1001
q: 1	1010
r: 2	1011
s: 3	1100
u: 1	1101
w: 6	1110
y: 2	1111

More efficient coding

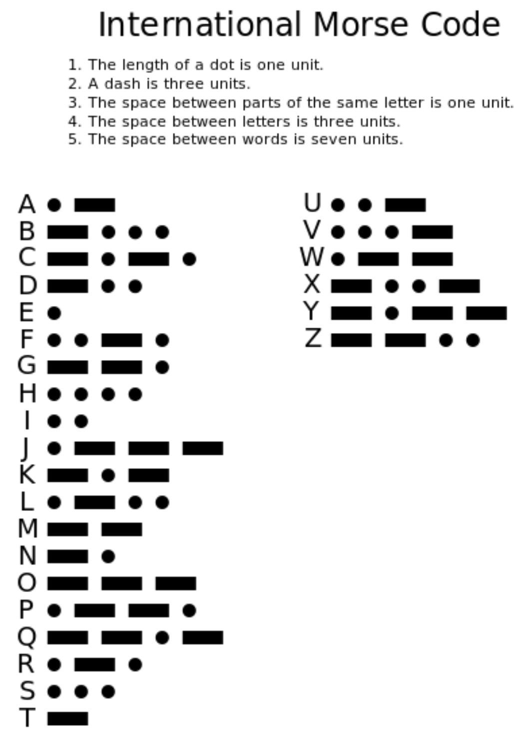


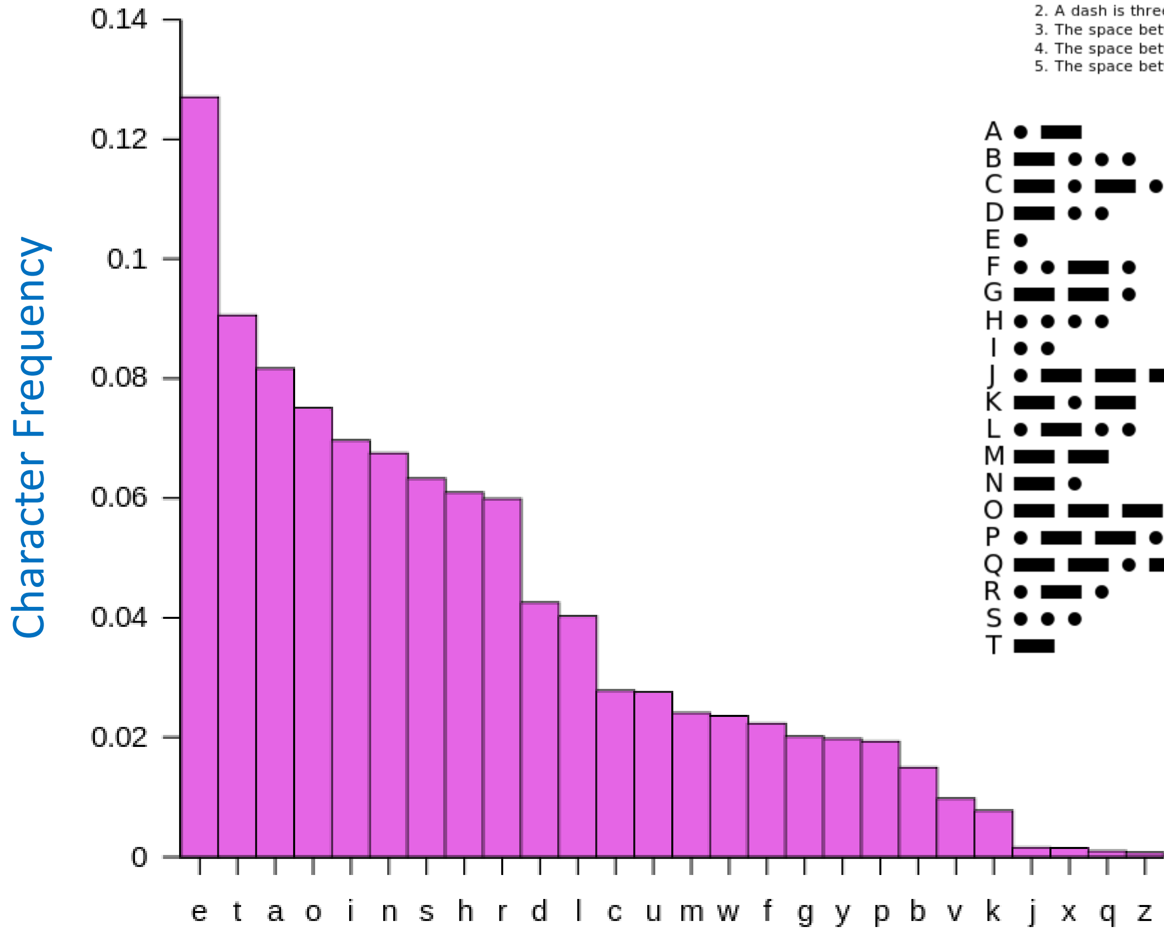
$$B(T, \{f_c\}) = \sum_{character\ c} \ell_c f_c$$

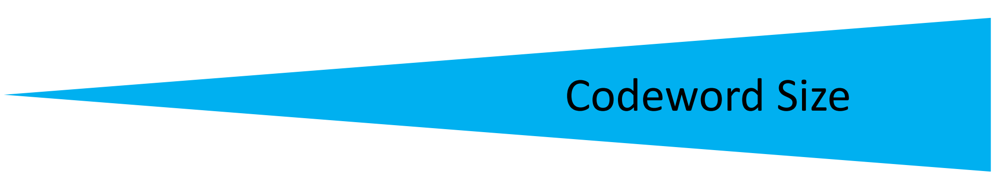
When this is big

Make this small

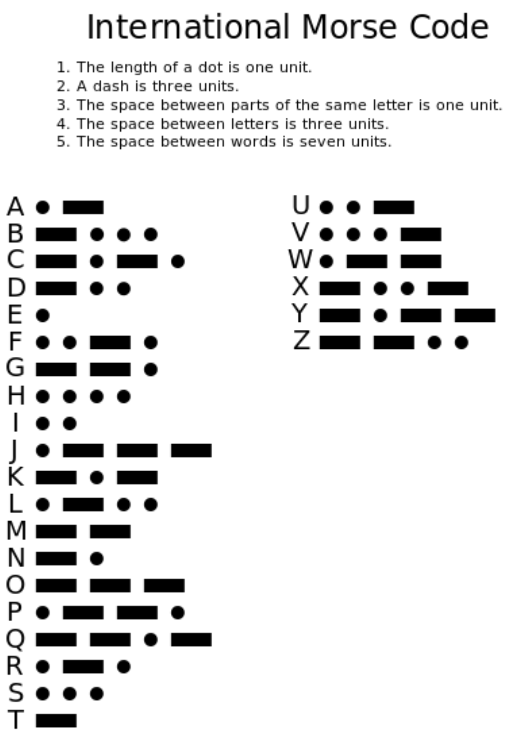
Morse Code


International Morse Code
1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.
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 ▬ ▬ ● ●


Character Frequency
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
e t a o i n s h r d l c u m w f g y p b v k j x q z


Codeword Size

Problem with Morse Code



A A

Decode: • ▬ • ▬

ET ET

R T

EN T

Ambiguous Decoding

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
…	…

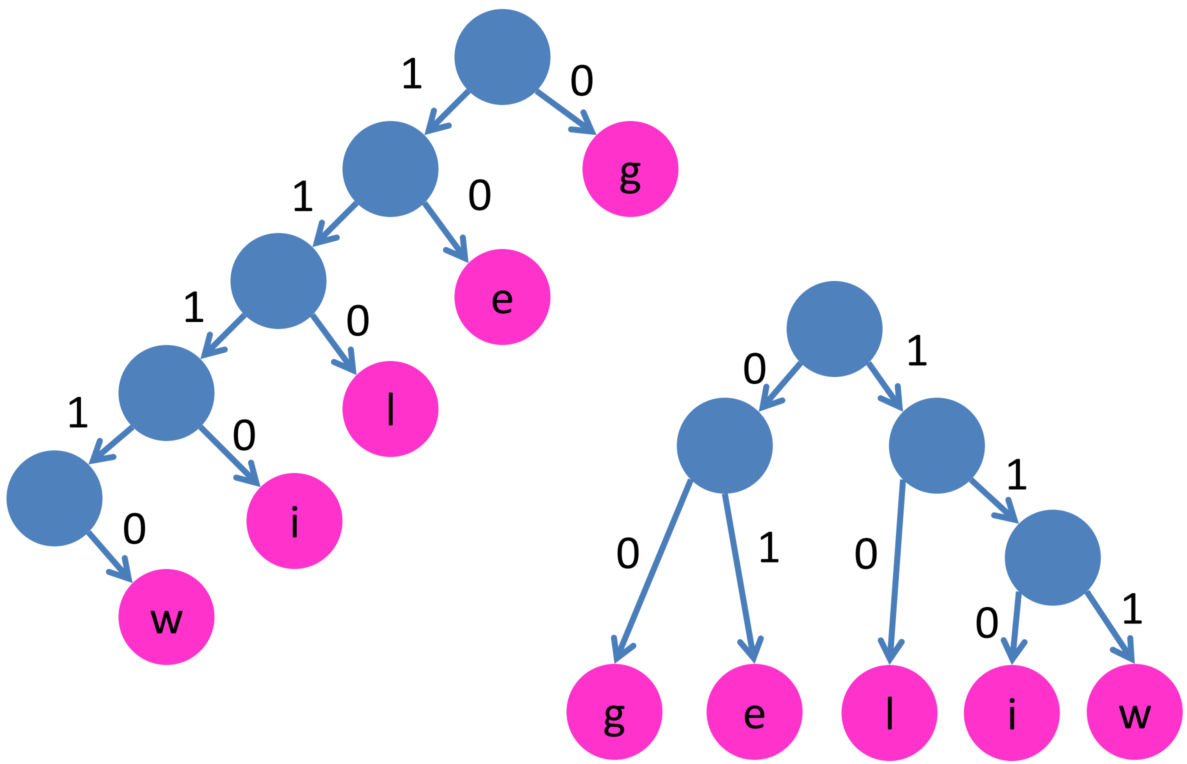
11110111000011010

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 frequences $\{f_c\}$
- Output: A prefix-free code T which minimizes

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

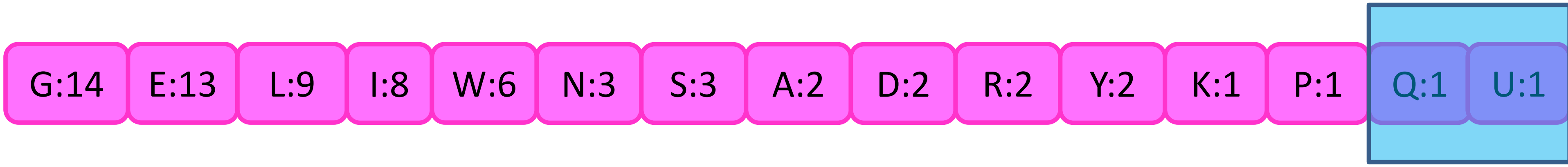
Huffman Coding!!

Greedy Algorithms

- Require Optimal Substructure
  - Solution to larger problem contains the solution to a smaller one
  - Only one subproblem to consider!
- Idea:
  1. Identify a greedy choice property
     - How to make a choice guaranteed to be included in some optimal solution
  2. Repeatedly apply the choice property until no subproblems remain

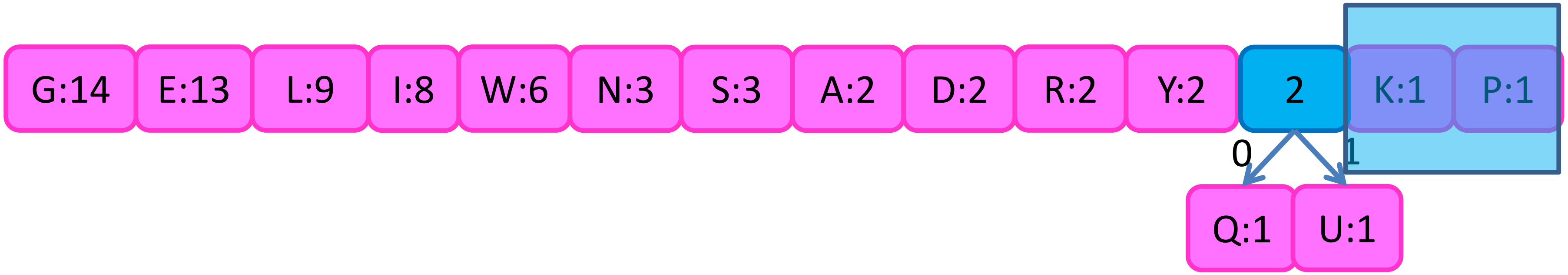
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



Huffman Algorithm

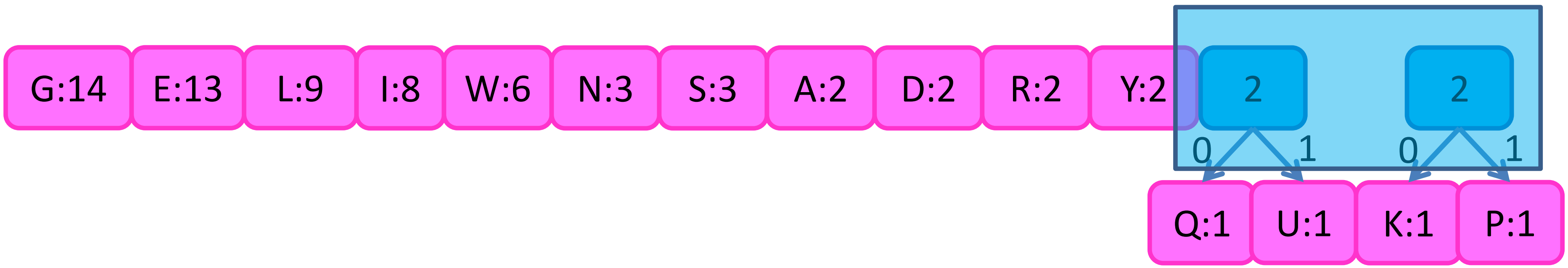
- Choose the least frequent pair, combine into a subtree



Subproblem of size $n-1$!

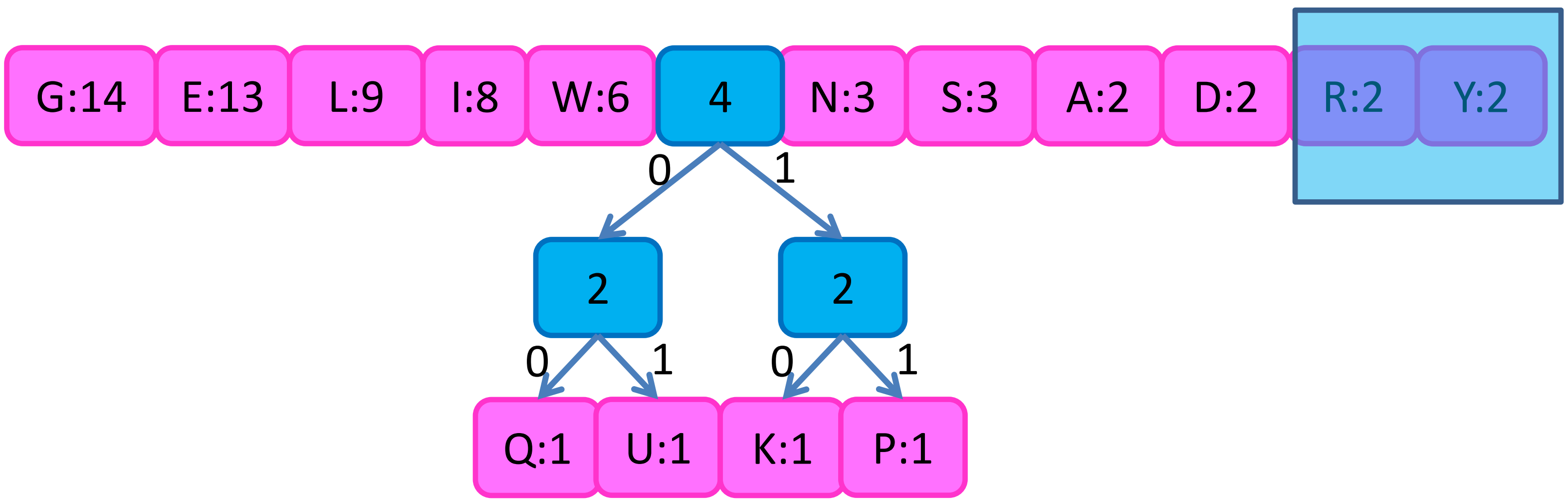
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



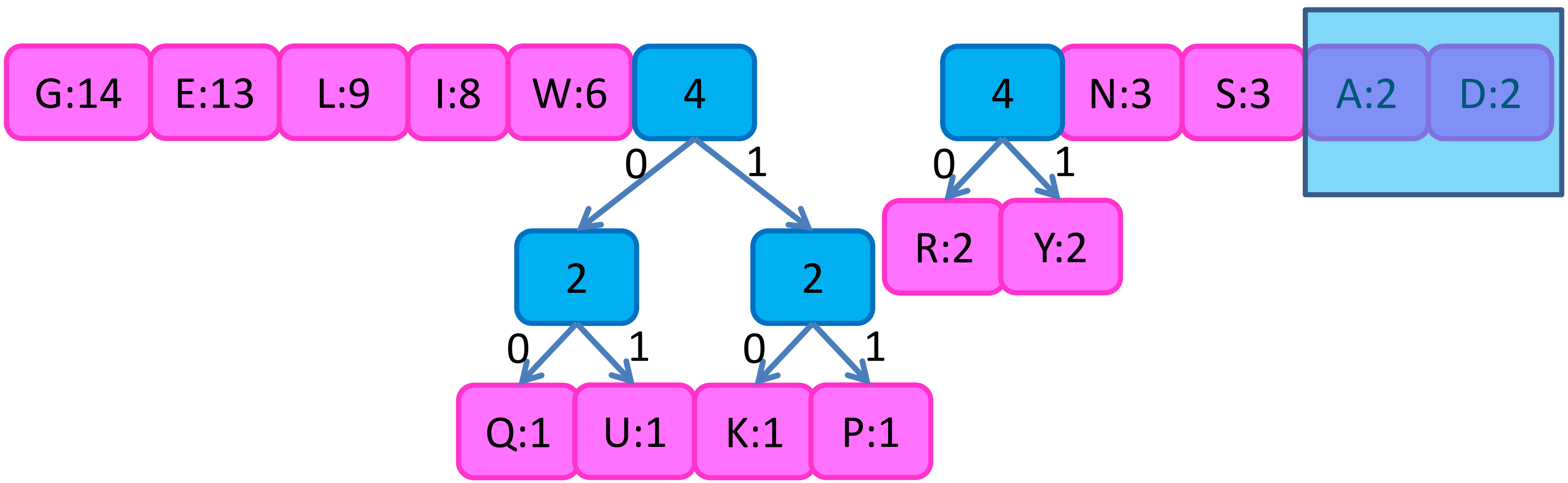
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



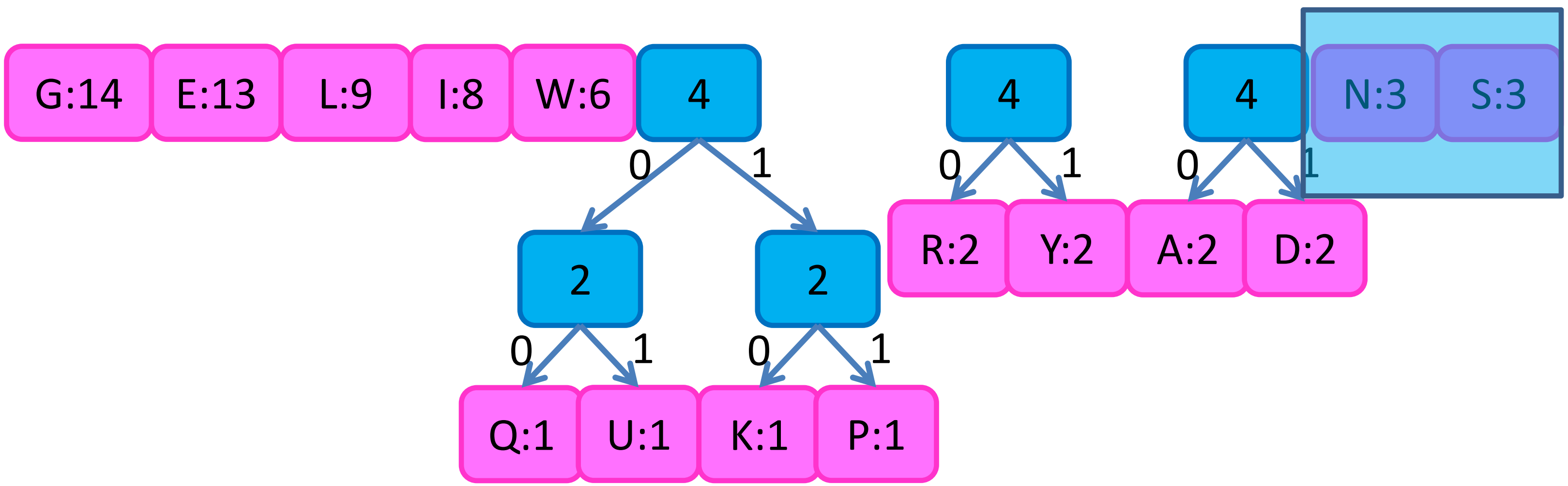
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



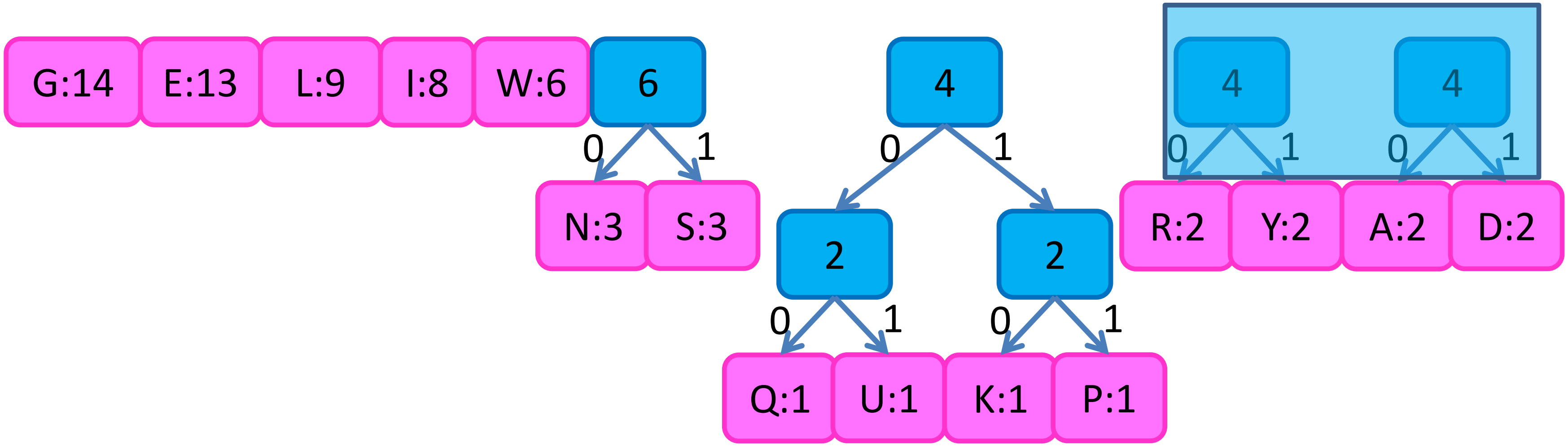
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



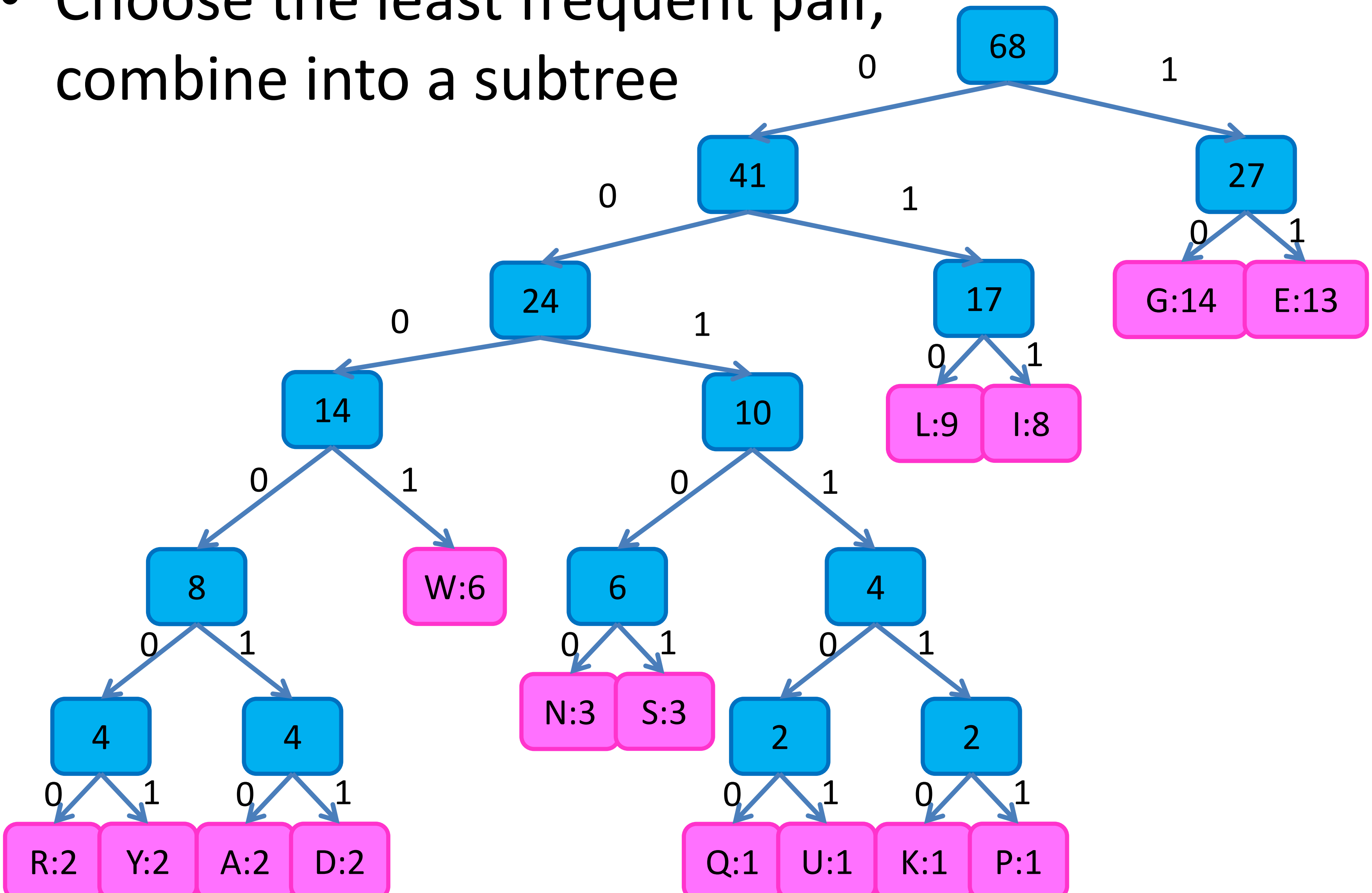
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



Exchange argument

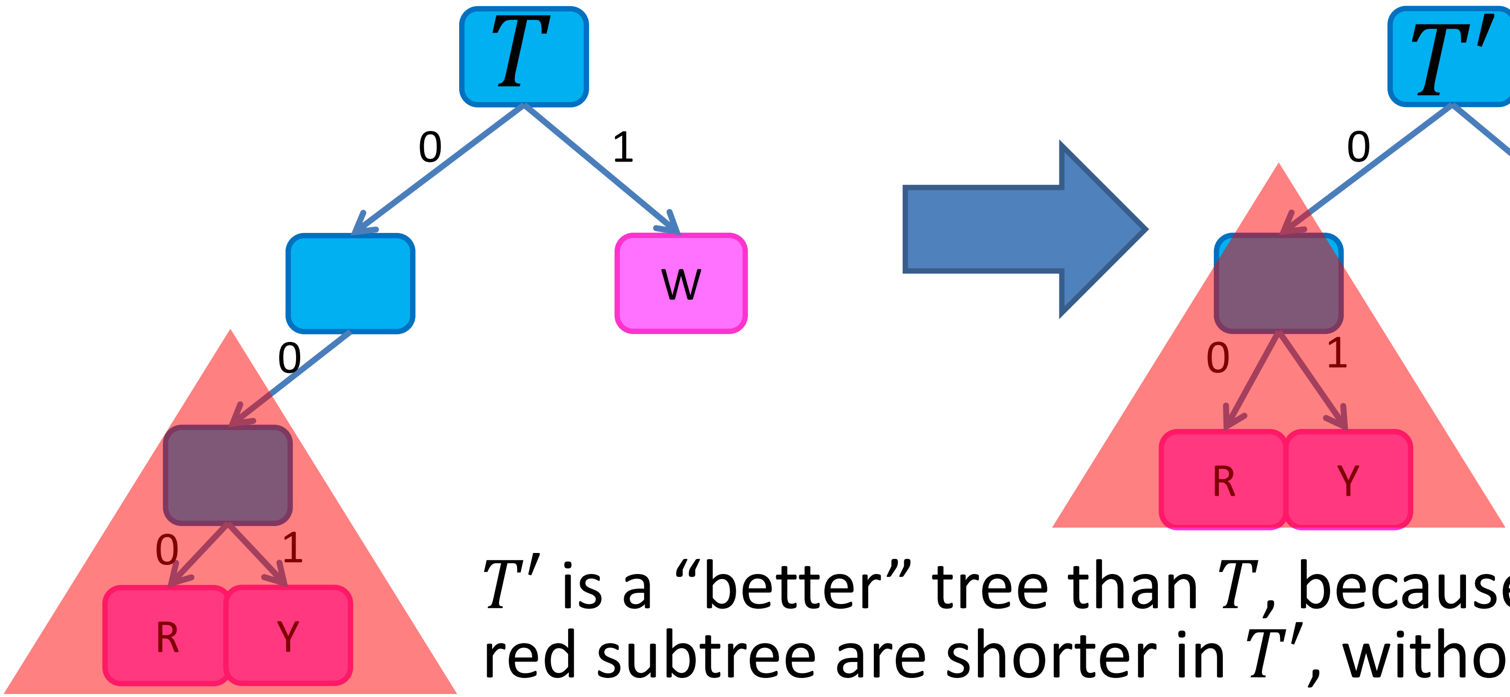
- Shows correctness of a greedy algorithm
- Idea:
  - Show exchanging an item from an arbitrary optimal solution with your greedy choice makes the new solution no worse
  - How to show my sandwich is at least as good as yours:
    - Show: “I can remove any item from your sandwich, and it would be no 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
    - Proof by contradiction

Showing Huffman is Optimal

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

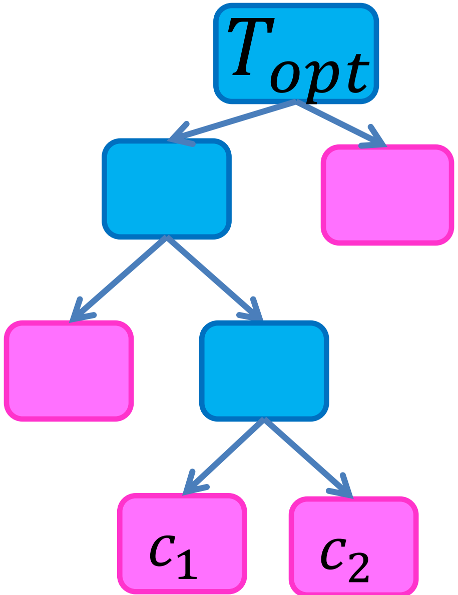


T' is a “better” tree than T, because all codes in red subtree are shorter in T', without creating any longer codes

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

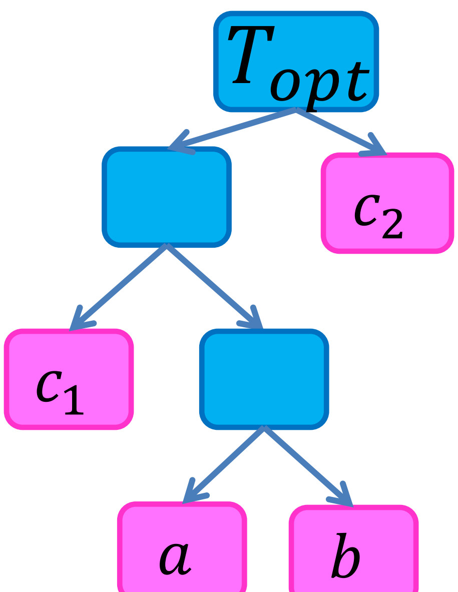
Case 1: Consider some optimal tree T_{opt}. If c_1, c_2 are siblings in this tree, then claim holds



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

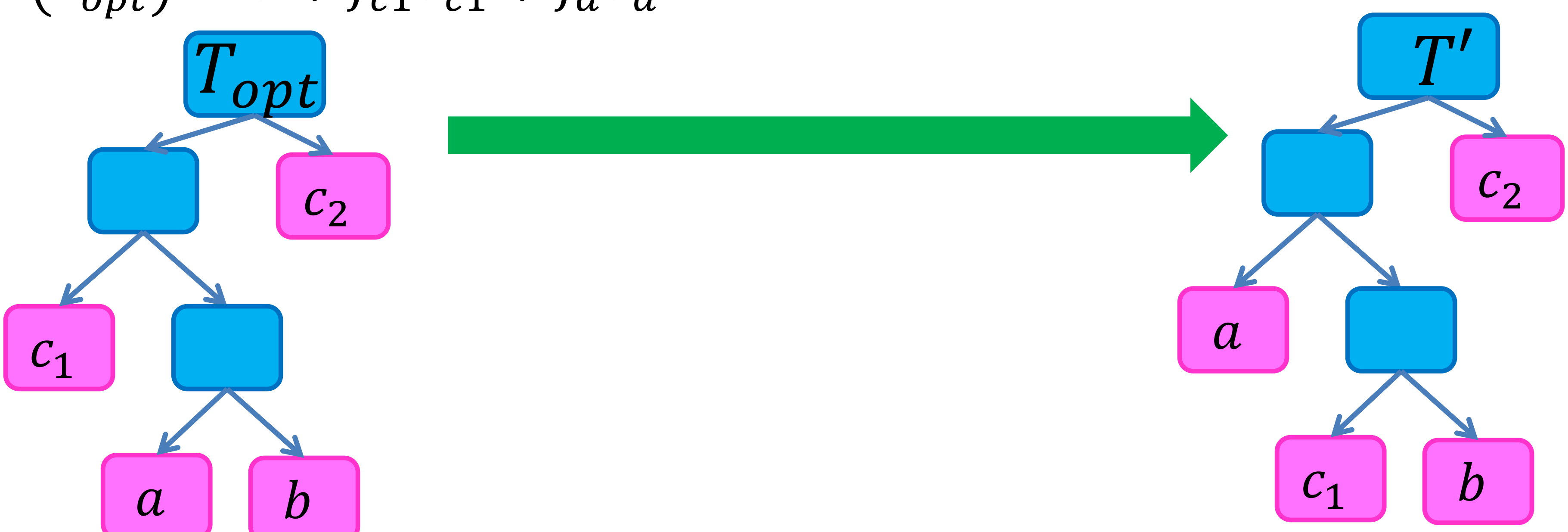
a, b = lowest-depth siblings

Idea: show that swapping c_1 with a does not increase cost of the tree.

Assume: $f_{c1} \leq f_a$

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

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



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: $f_{c1} \leq f_a$

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

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

$$\begin{aligned}
B(T_{opt}) - B(T') &\overset{\geq 0 \Rightarrow T' \text{ optimal}}{=} C + f_{c1}\ell_{c1} + f_a\ell_a - (C + f_{c1}\ell_a + f_a\ell_{c1}) \\
&= f_{c1}\ell_{c1} + f_a\ell_a - f_{c1}\ell_a - f_a\ell_{c1} \\
&= f_{c1}(\ell_{c1} - \ell_a) + f_a(\ell_a - \ell_{c1}) \\
&= (f_a - f_{c1})(\ell_a - \ell_{c1})
\end{aligned}$$

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

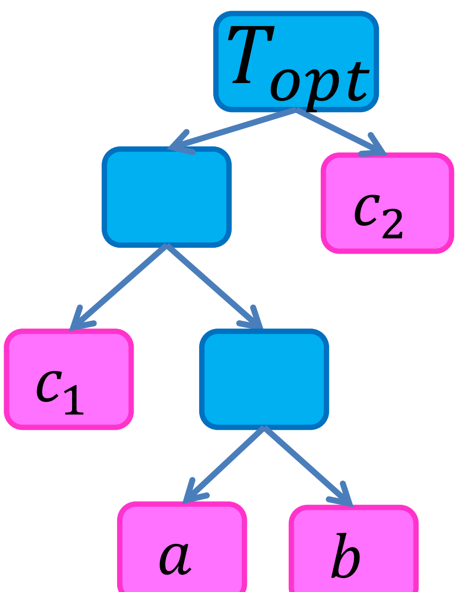
$a, b =$ lowest-depth siblings

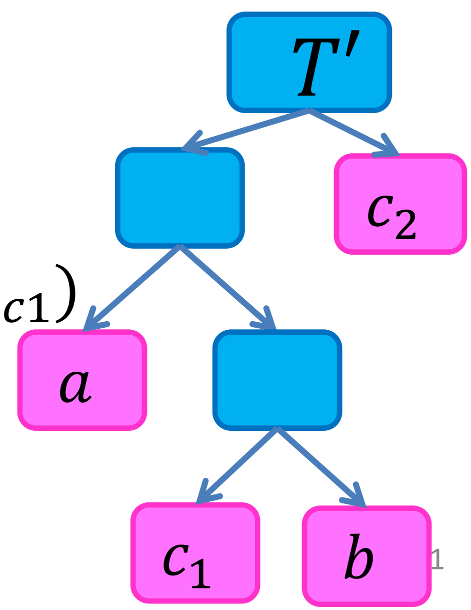
Idea: show that swapping c_1 with a does not increase cost of the tree.

Assume: $f_{c1} \le f_a$

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

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





$$B(T_{opt}) - B(T') = \underbrace{(f_a - f_{c1})}_{\ge 0}\underbrace{(\ell_a - \ell_{c1})}_{\ge 0}$$

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

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

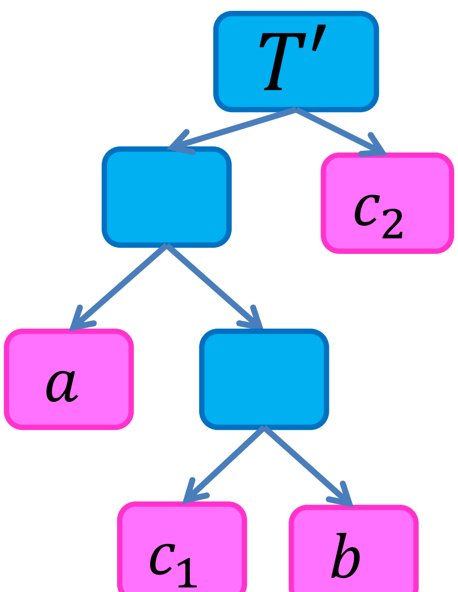
a, b = lowest-depth siblings

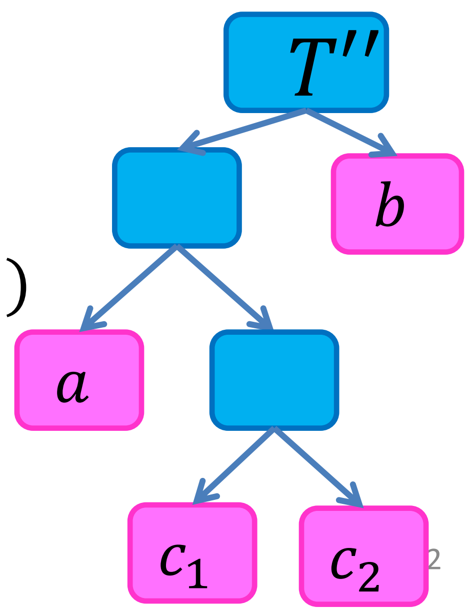
Idea: show that swapping c_2 with b does not increase cost of the tree.

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!

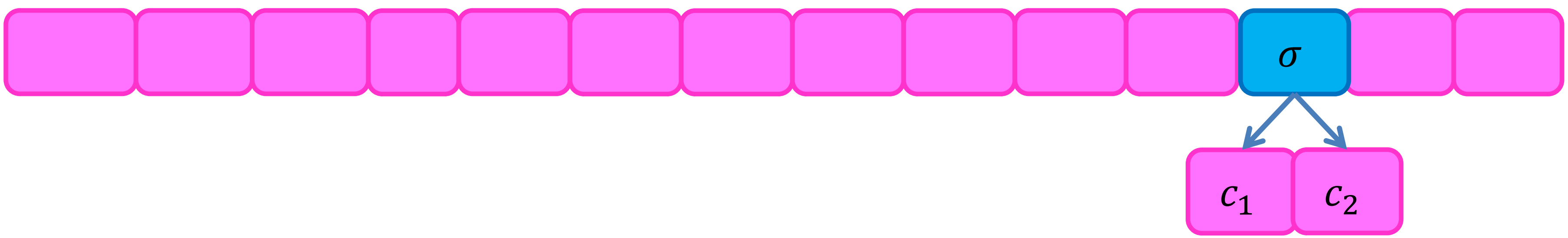
Showing Huffman is Optimal

- 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
    - Proof by contradiction

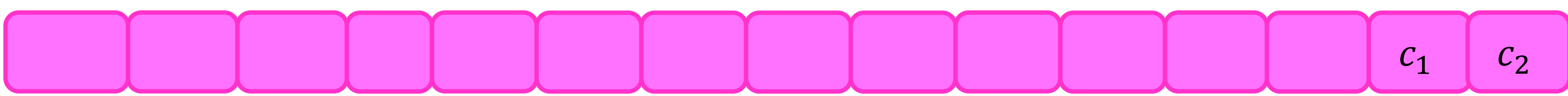
Finishing the Proof

- Show Optimal Substructure
  - Show treating c_1, c_2 as a new "combined" character gives optimal solution

Why does solving this:

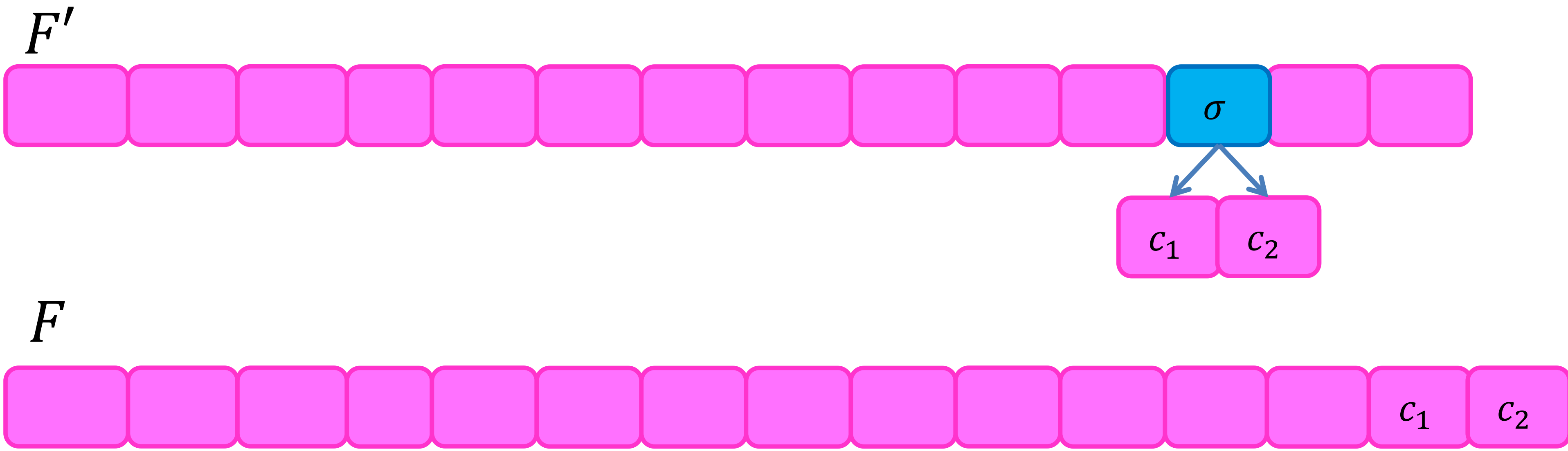


Give an optimal solution to this?:



Optimal Substructure

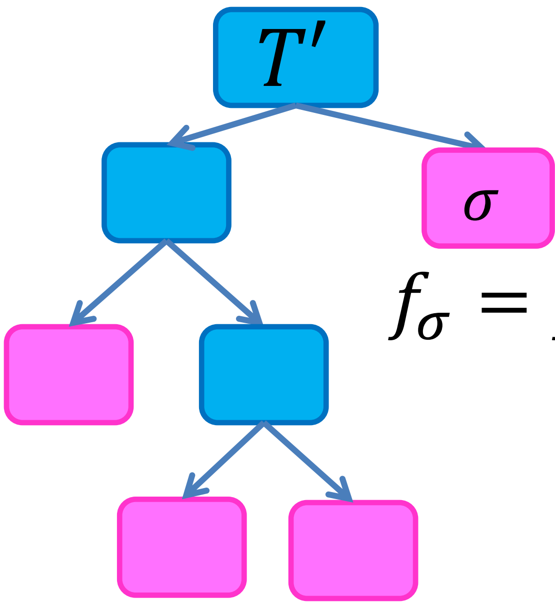
- Claim: An optimal solution for F involves finding an optimal solution for F', then adding c_1, c_2 as children to σ



Optimal Substructure

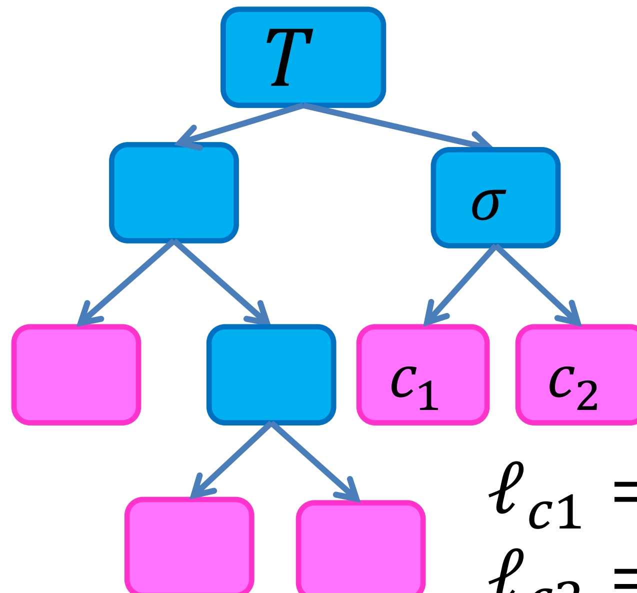
- Claim: An optimal solution for F involves finding an optimal solution for F', then adding c_1, c_2 as children to σ

If this is optimal



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

Then this is optimal



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

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

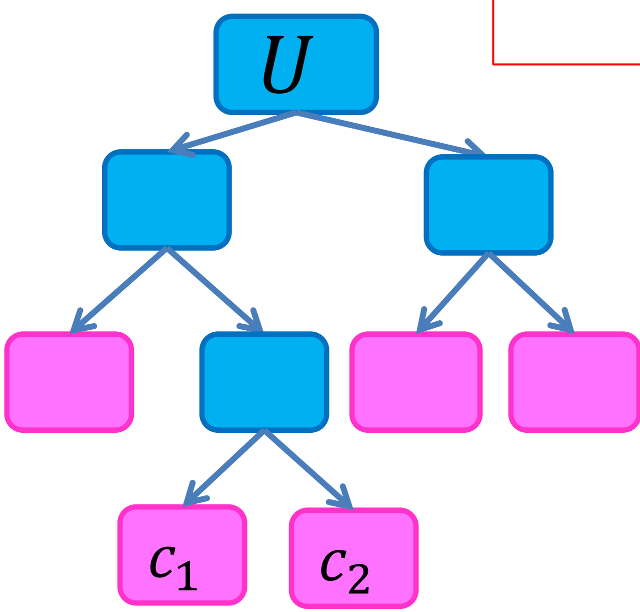
Optimal Substructure

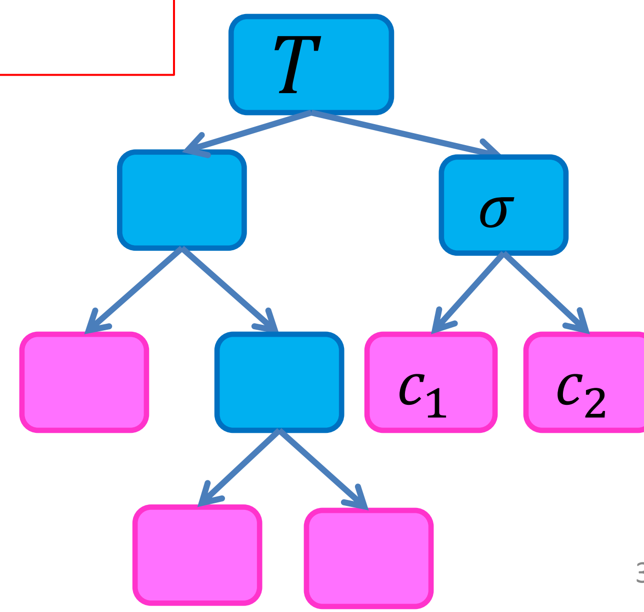
- Claim: An optimal solution for F involves finding an optimal solution for F', then adding c_1, c_2 as children to σ

Toward contradiction

Suppose T is not optimal
Let U be a lower-cost tree

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



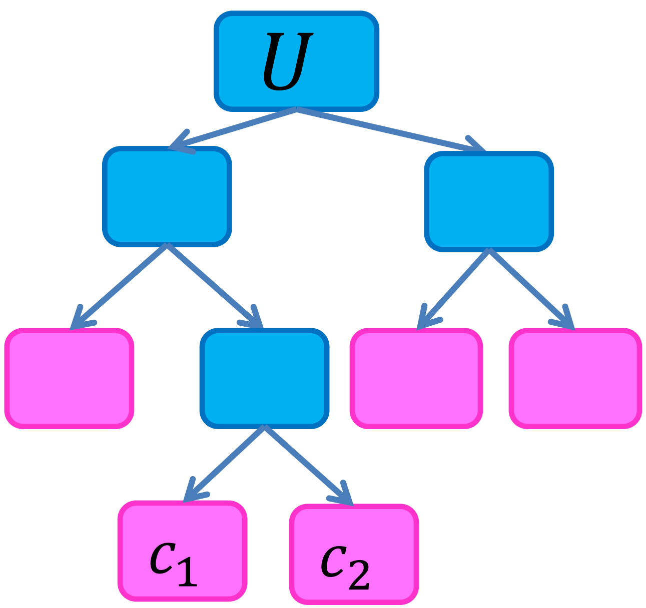


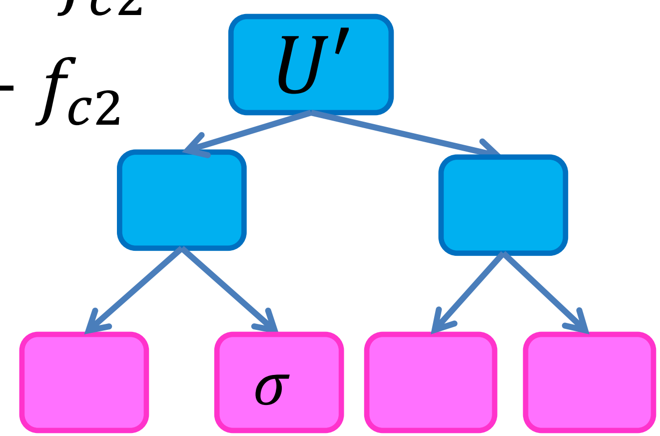
Optimal Substructure

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

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

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





Contradicts optimality of T', so T is optimal!

Entire Huffman Derivation Follows

- Not covered in class, just for your review

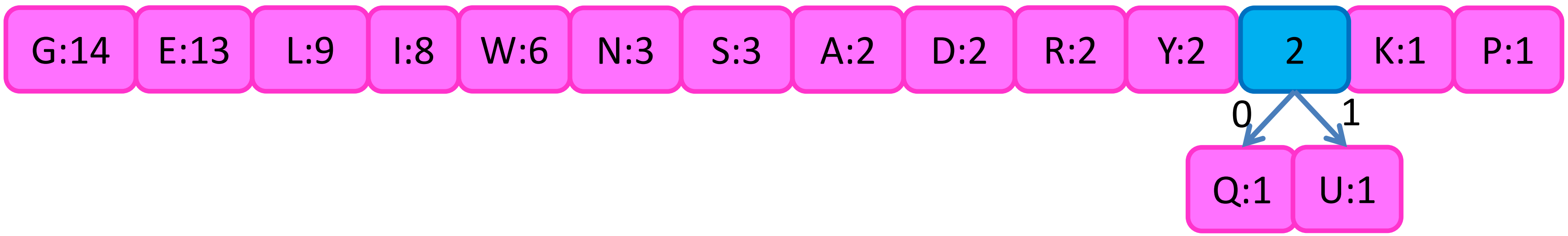
Huffman Algorithm

- 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

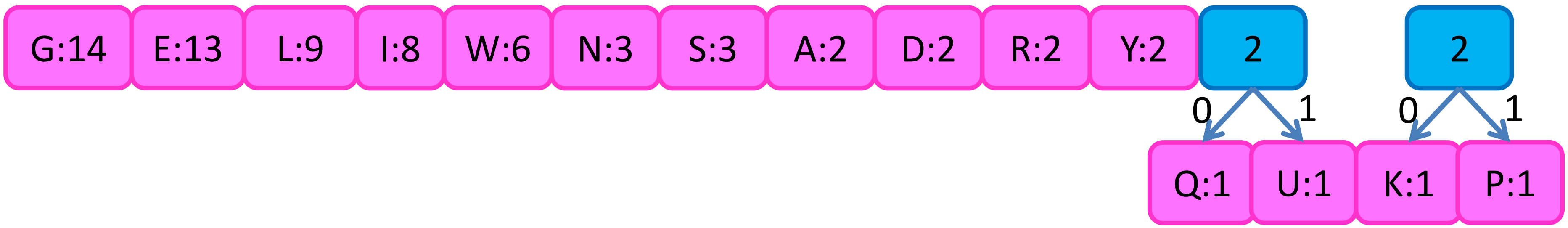
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



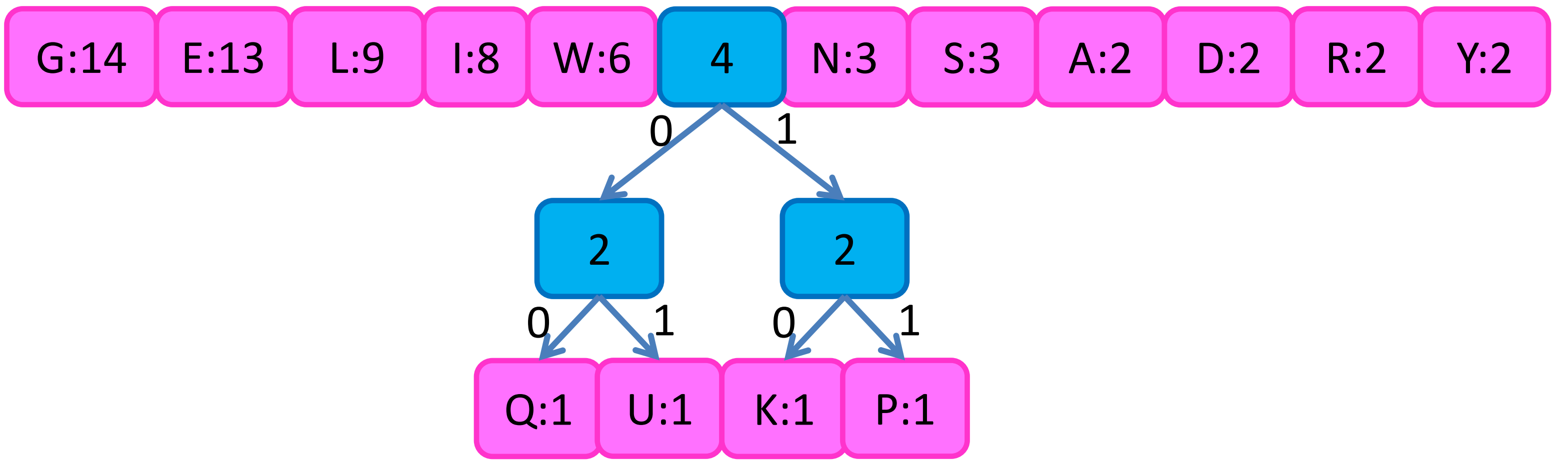
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



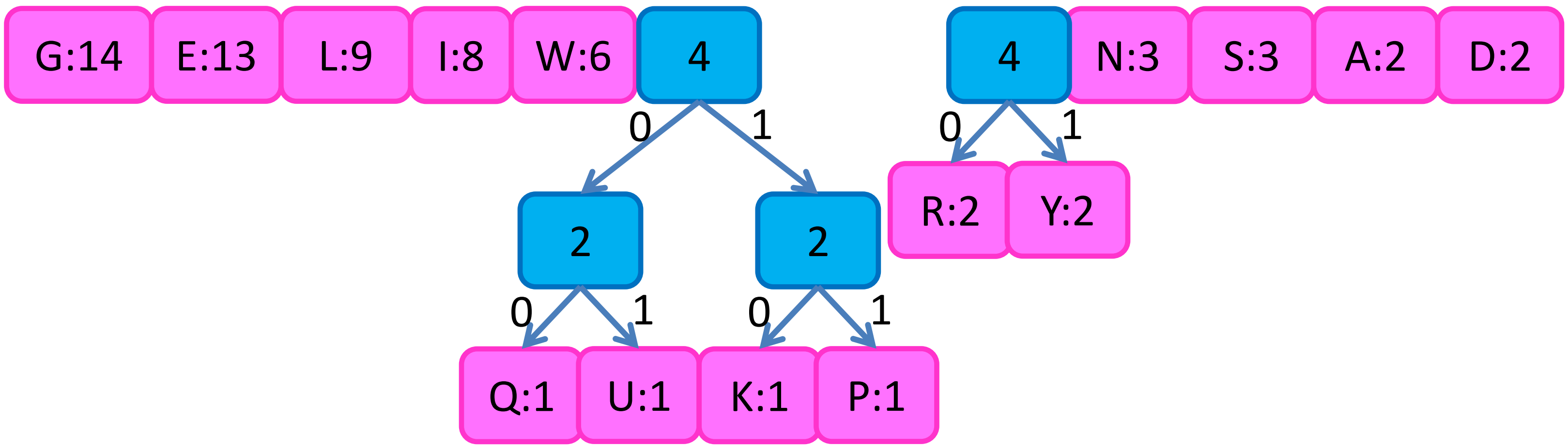
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



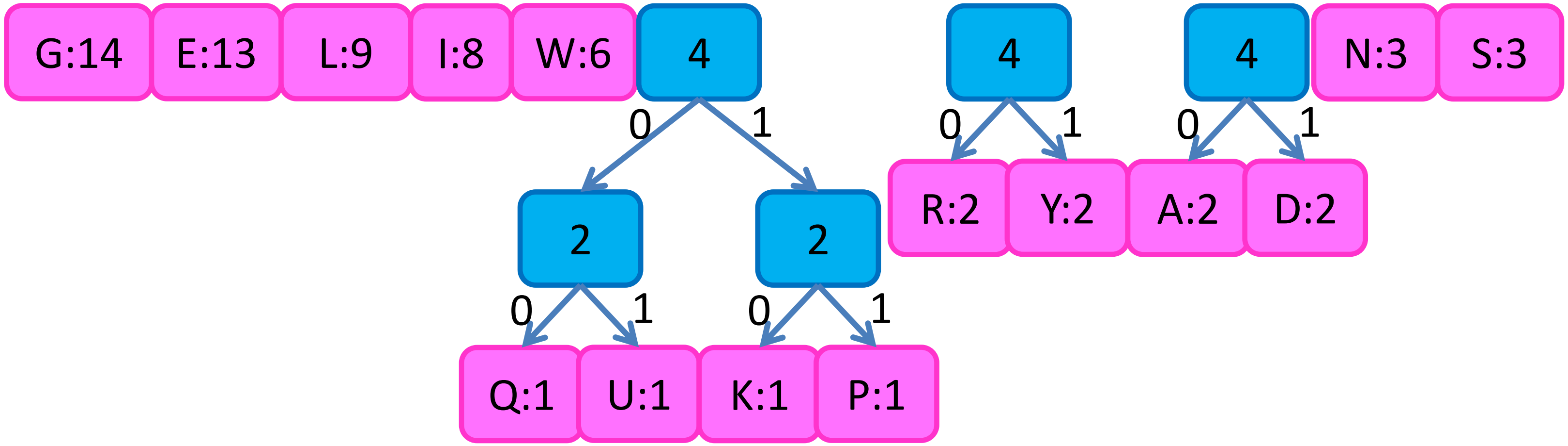
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



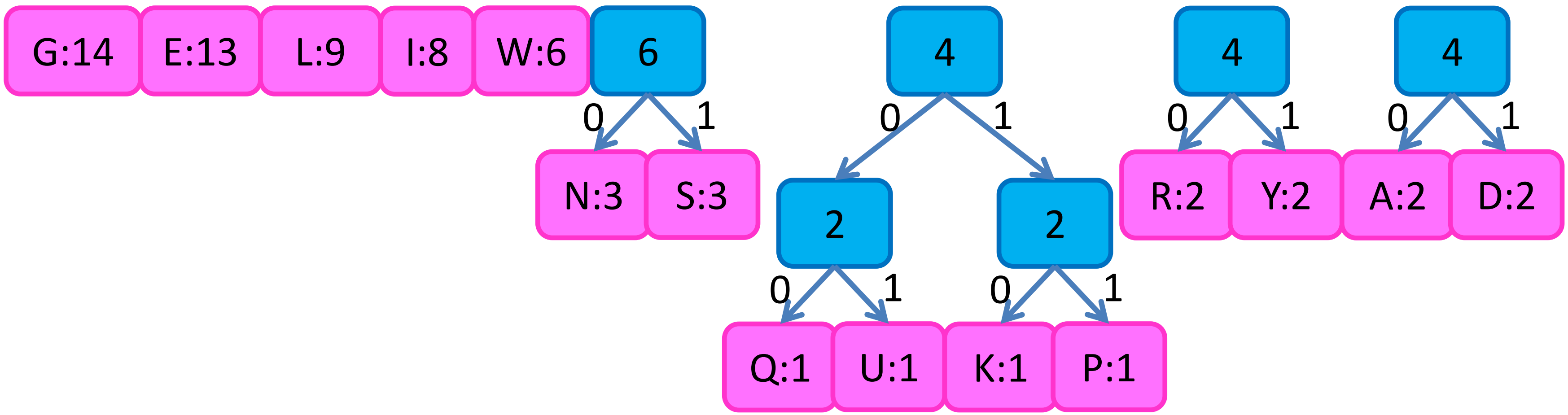
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



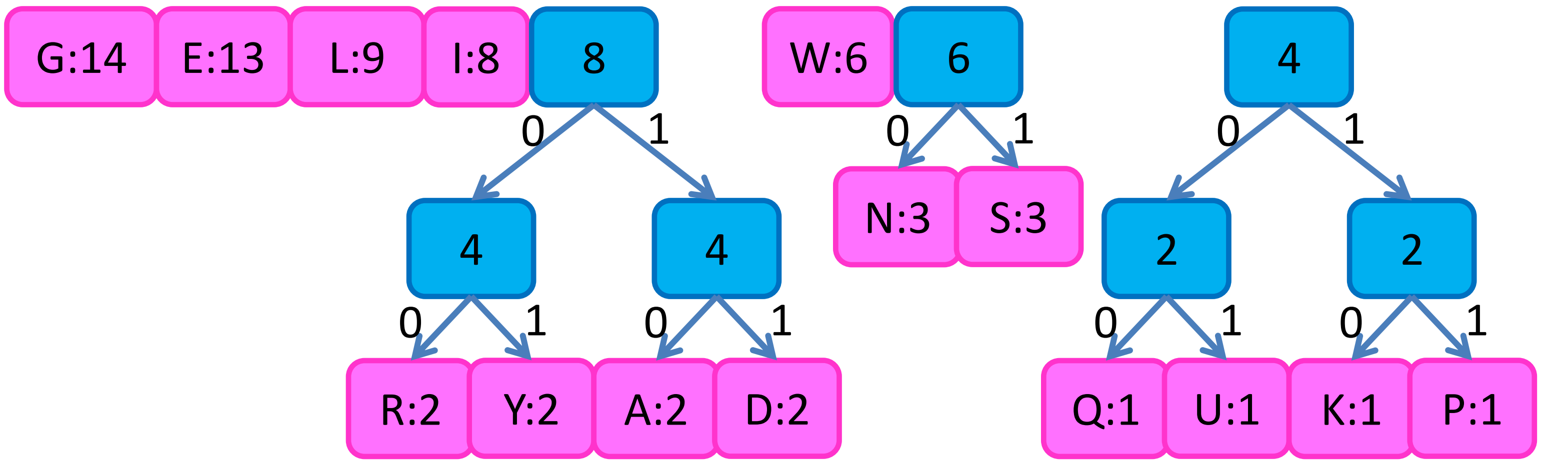
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



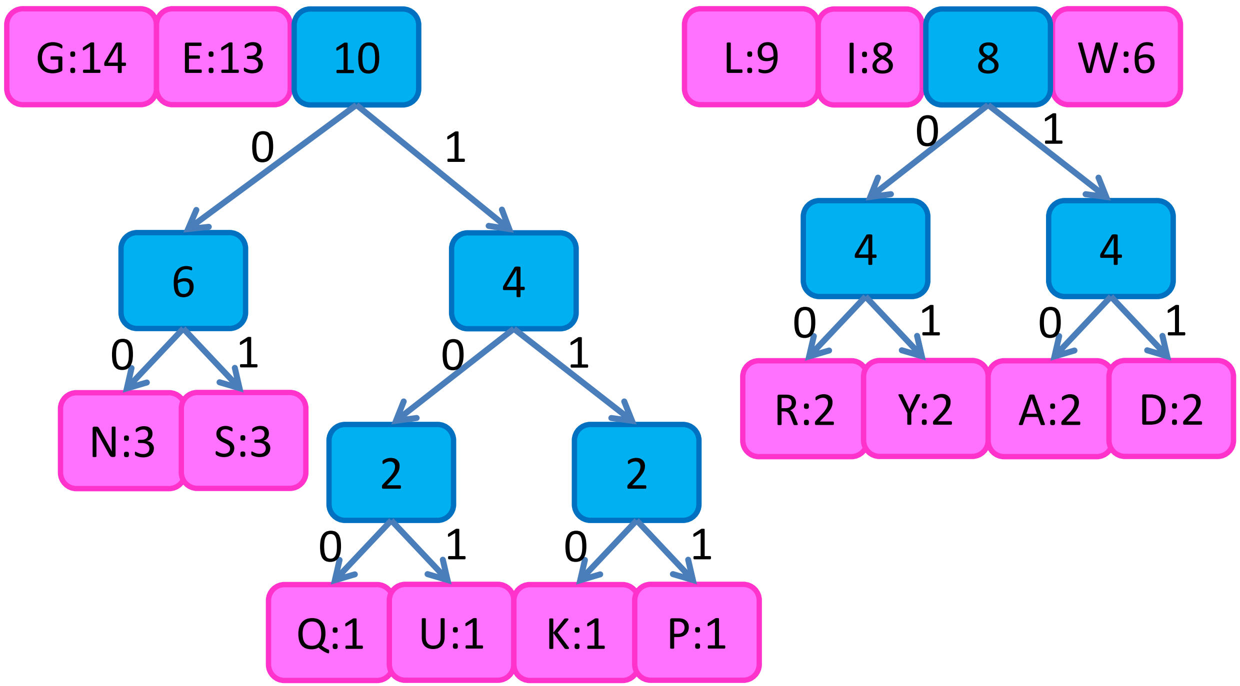
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



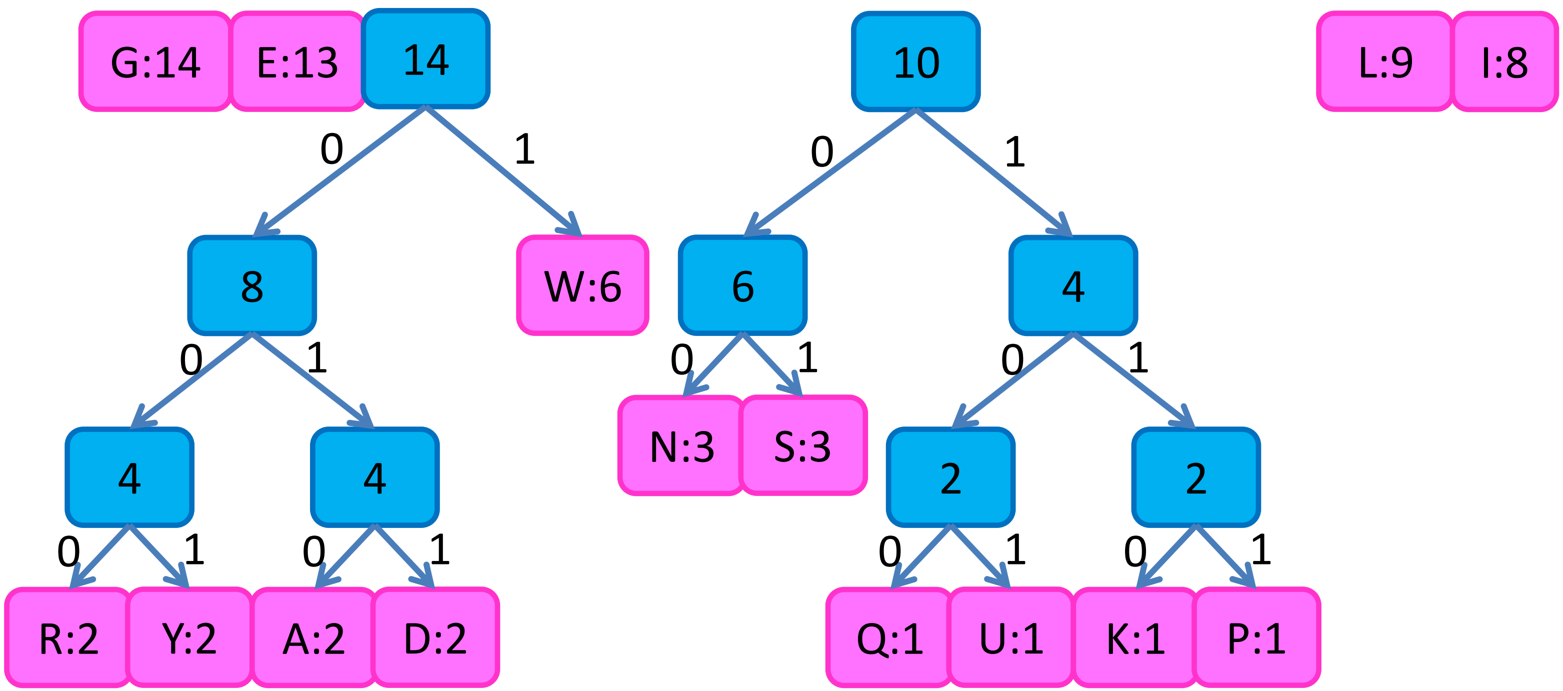
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



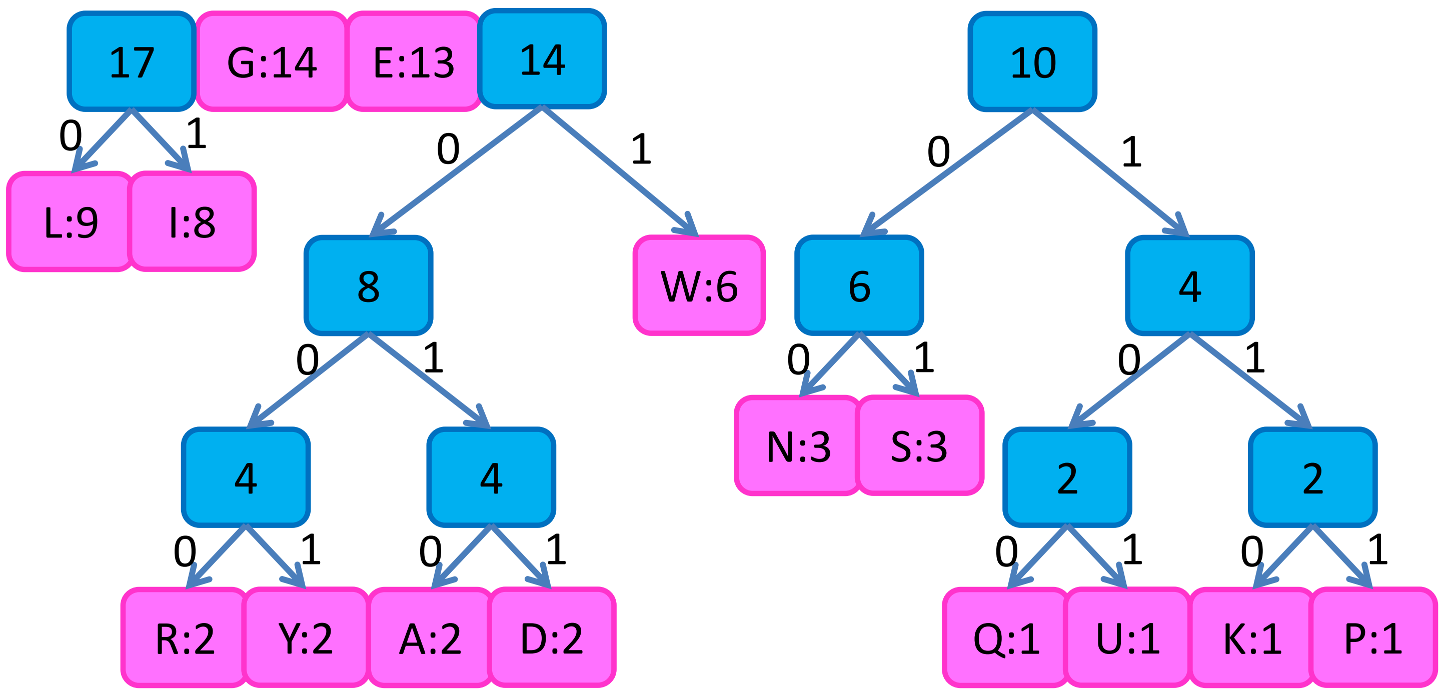
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



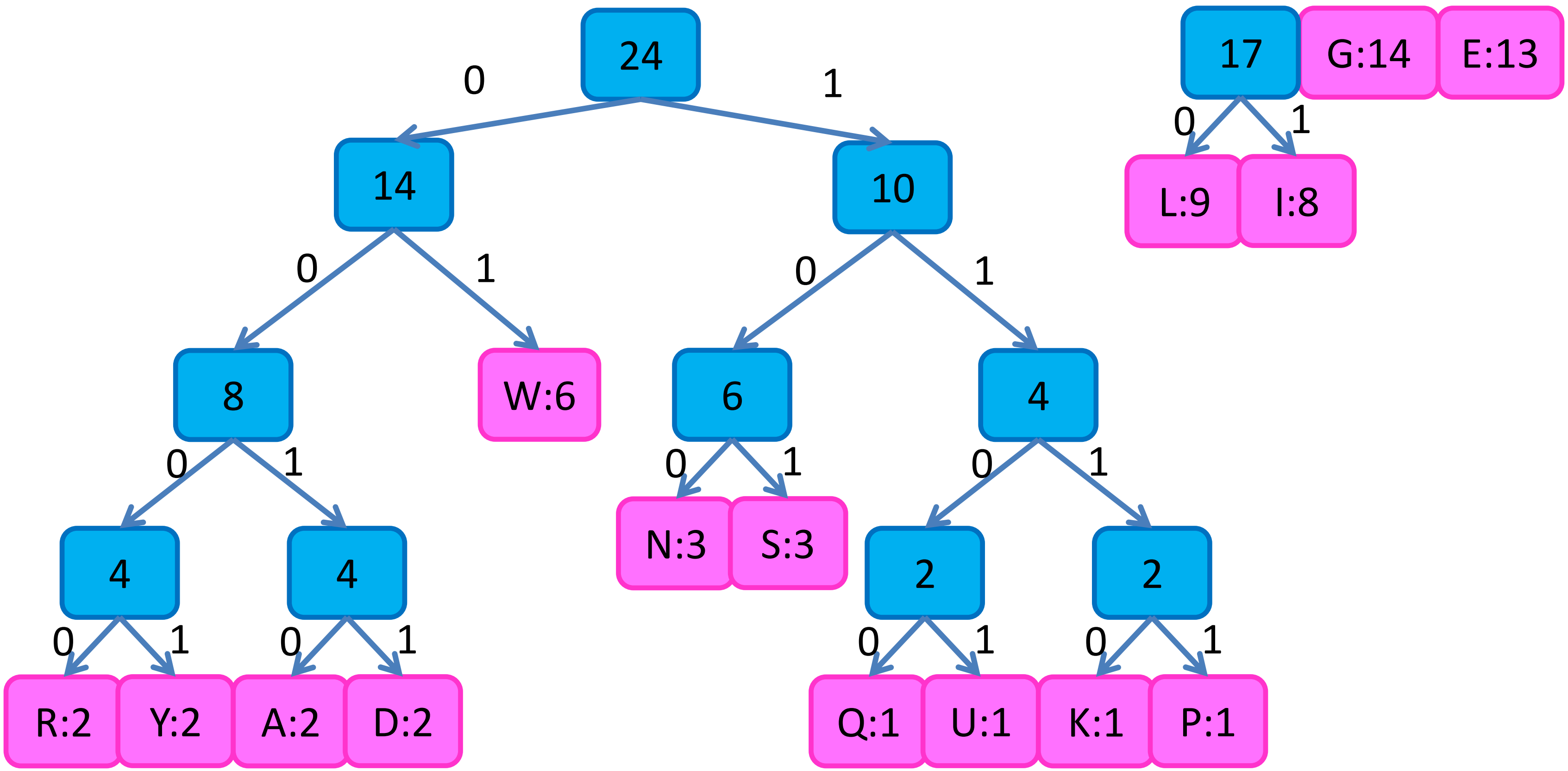
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



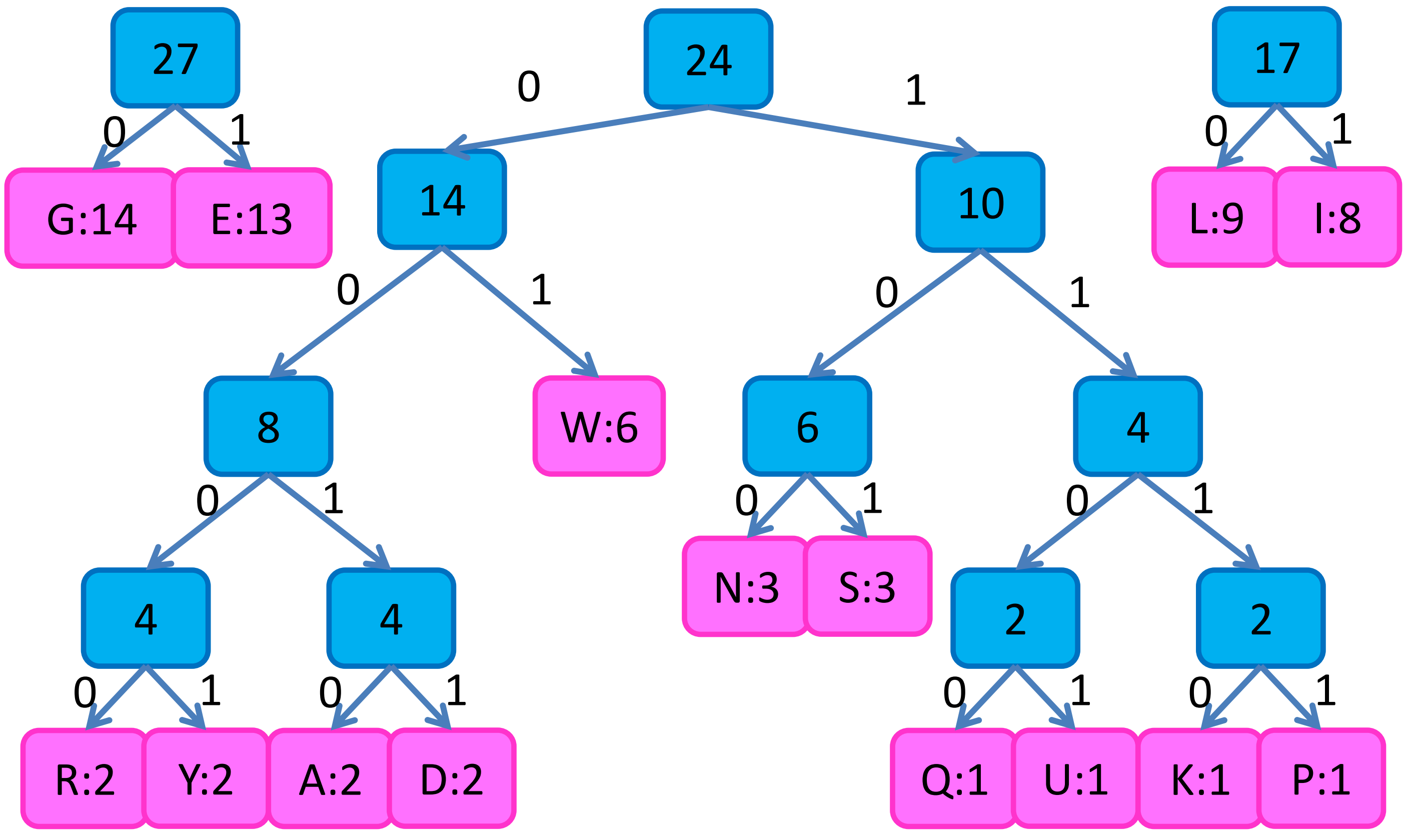
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



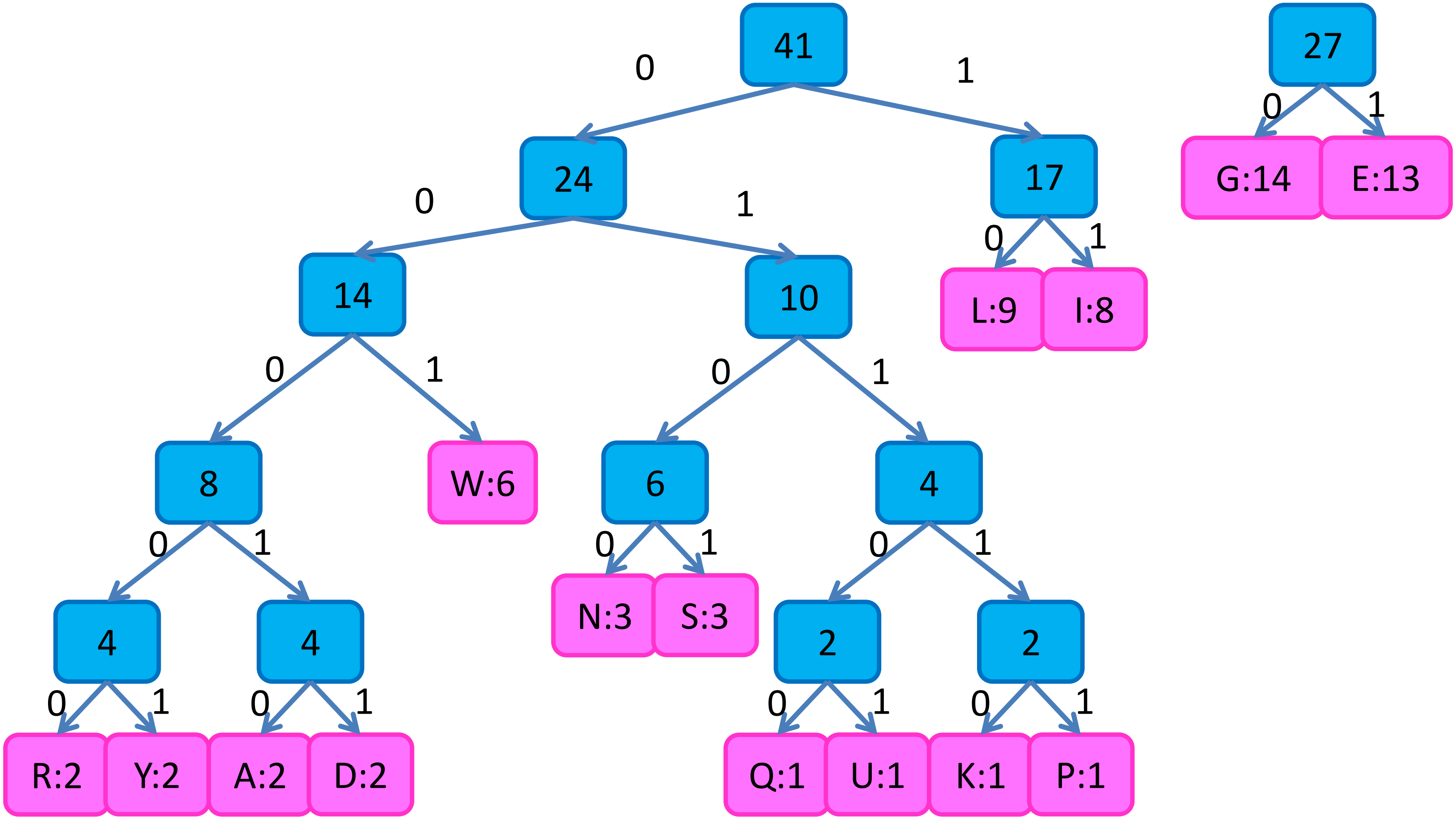
Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



Huffman Algorithm

- Choose the least frequent pair, combine into a subtree



Huffman Algorithm

- Choose the least frequent pair, combine into a subtree