

4.8 Huffman Codes

- Encoding in bits
 - Prefix codes
 - Tree representation
- Huffman codes:
 - Algorithm
 - Proof

Updated from lecture slides provided by Mathijs de Weerd, TU Delft.

Encoding in bits

Q. Given a text that uses 32 symbols (26 different letters, space, and some punctuation characters), how can we encode this text in bits?

A. Encode 2^5 different symbols using 5 bits per symbol.
This is called **fixed length encoding**.

Ex. $c(a) = 00000$

$c(b) = 00001$

$c(e) = 00100$

What is 0000000100?

Data Compression

Q. Some symbols (e, t, a, o, i, n) are used far more often than others. How can we use this to reduce our encoding?

A. Encode these characters with fewer bits, and the others with more bits.

Ex. $c(a) = 01$
 $c(b) = 010$
 $c(e) = 1$

What is 0101?

Q. How do we know when the next symbol begins?

- i) Use a separation symbol (like the pause in Morse), or
- ii) make sure that there is no ambiguity by ensuring that **no** code is a **prefix** of another one.

Prefix Codes

Definition. A **prefix code** for a set S is a function c that maps each $x \in S$ to 1s and 0s in such a way that for $x, y \in S$, $x \neq y$, $c(x)$ is not a prefix of $c(y)$.

Ex. $c(a) = 01$

$c(b) = 010$

$c(e) = 1$

Q. Is this a prefix code?

A. No, $c(a)$ is a prefix of $c(b)$.

Ex. $c(a) = 11$

$c(e) = 01$

$c(k) = 001$

$c(l) = 10$

$c(u) = 000$

Q. Is this a prefix code?

A. Yes.

Q. What is the meaning of 1001000001 ?

A. “leuk”

Towards Optimal Prefix Codes

Ex. $c(a) = 11$

$c(e) = 01$

$c(k) = 001$

$c(l) = 10$

$c(u) = 000$

Optimal code: small representation (on average)

Suppose frequencies are known in a text:

$f_a=0.4$, $f_e=0.2$, $f_k=0.2$, $f_l=0.1$, $f_u=0.1$

Q. How to encode symbols such that on average encoding is smallest?

A. Find a prefix code that has the lowest possible average bits per letter.

Definition. The **average bits per letter** of a prefix code c is the sum over all symbols x in S of its frequency f_x times the number of bits of its encoding:

$$ABL(c) = \sum_{x \in S} f_x \cdot |c(x)|$$

Towards Optimal Prefix Codes

Ex. $c(a) = 11$

$c(e) = 01$

$c(k) = 001$

$c(l) = 10$

$c(u) = 000$

Optimal code: smallest ABL

$$ABL(c) = \sum_{x \in S} f_x \cdot |c(x)|$$

Suppose frequencies are known in a text:

$f_a=0.4$, $f_e=0.2$, $f_k=0.2$, $f_l=0.1$, $f_u=0.1$

Q. What is the ABL in this text given $c()$ above?

A. $2*f_a + 2*f_e + 3*f_k + 2*f_l + 3*f_u = 2.3$

Q. How can we improve the ABL?

A. Swap k and l . So $c(k) = 10$ and $c(l) = 001$

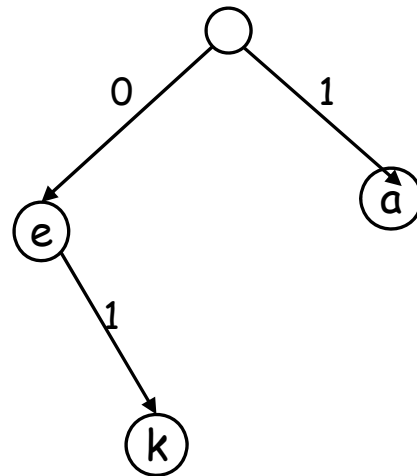
ABL then is: $2*f_a + 2*f_e + 2*f_k + 3*f_l + 3*f_u = 2.2$

Representing Codes using Binary Trees

Def. The tree representation of a binary code is a binary tree where

- children are uniquely identified by an edge label 0 or 1
- nodes are labeled with symbol x iff path is labeled with encoding $c(x)$

Ex. $c(a) = 1$
 $c(e) = 0$
 $c(k) = 01$

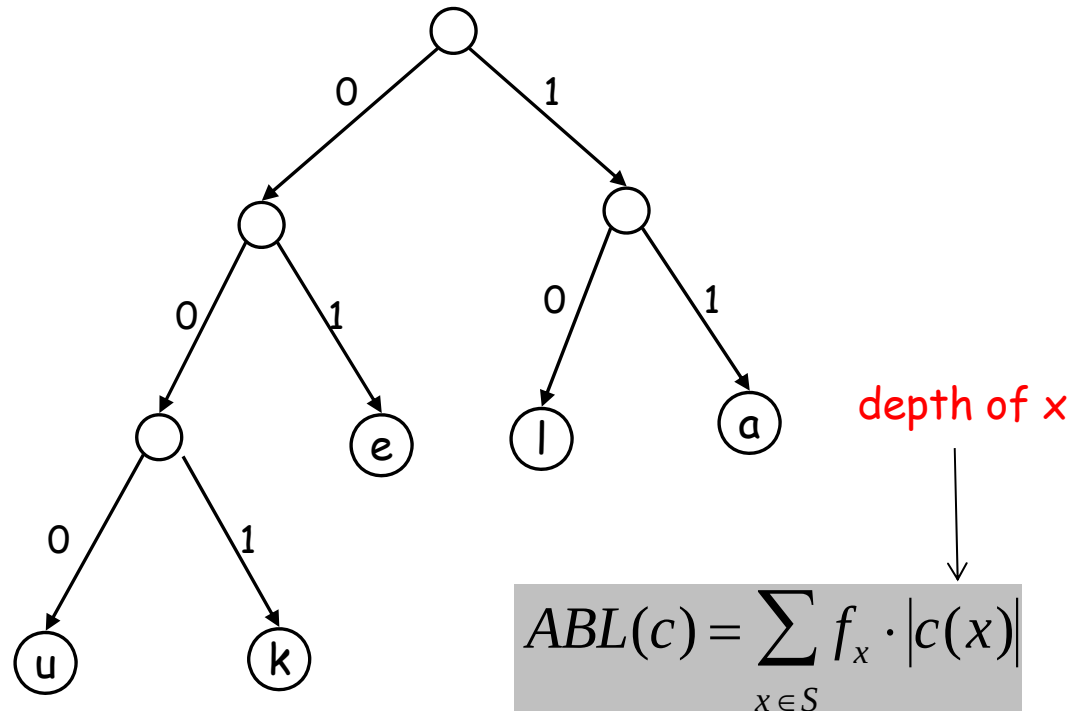


depth of x

$$ABL(c) = \sum_{x \in S} f_x \cdot |c(x)|$$

Representing Prefix Codes using Binary Trees

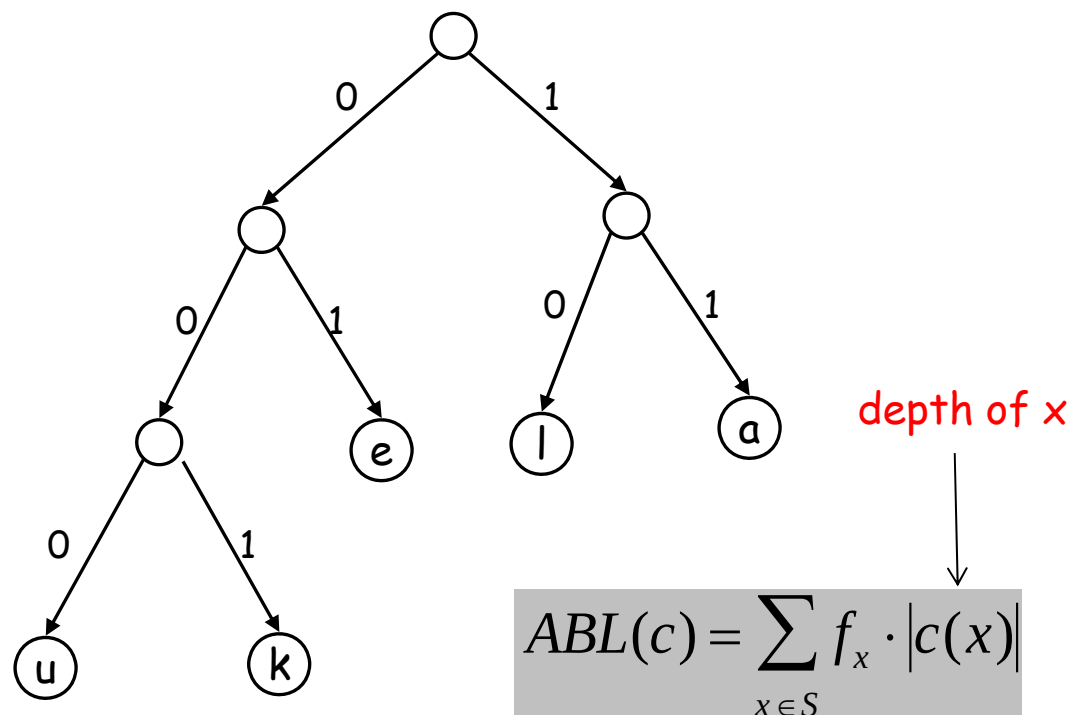
Ex. $c(a) = 11$
 $c(e) = 01$
 $c(k) = 001$
 $c(l) = 10$
 $c(u) = 000$



Q. How does the tree of a prefix code look like?

Representing Prefix Codes using Binary Trees

Ex. $c(a) = 11$
 $c(e) = 01$
 $c(k) = 001$
 $c(l) = 10$
 $c(u) = 000$

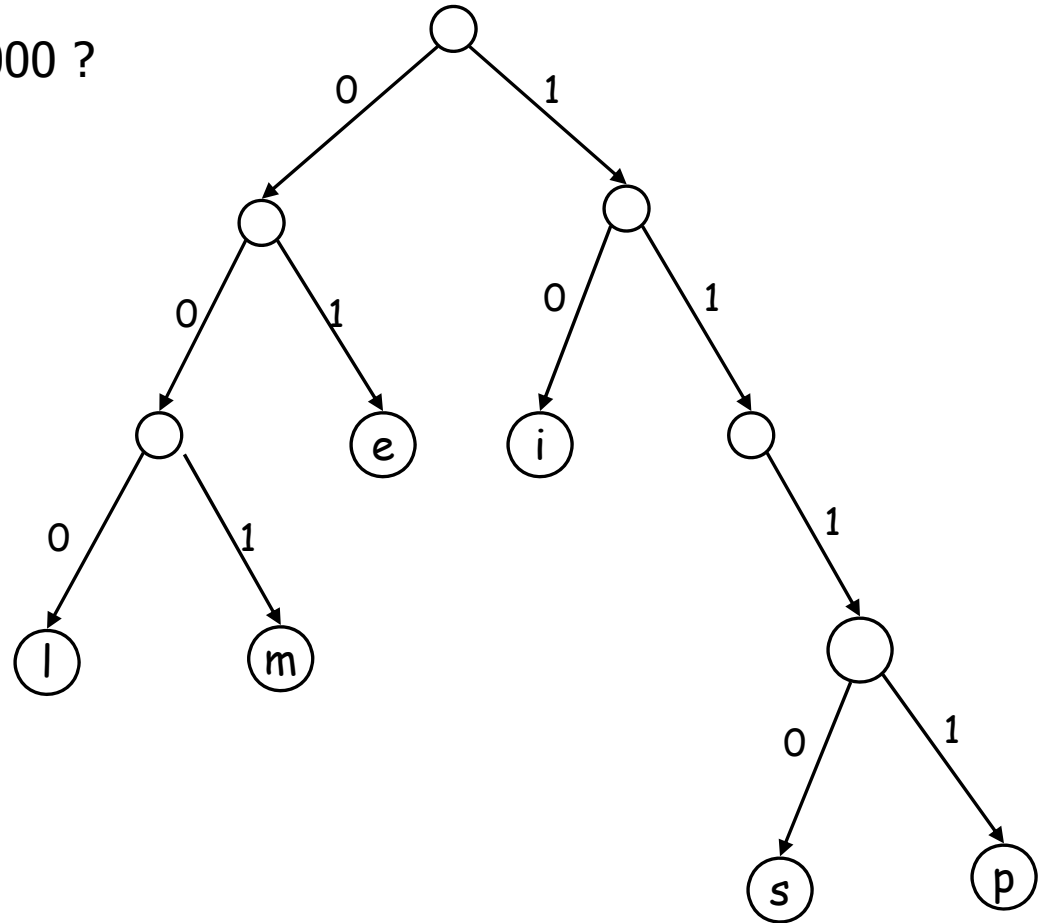


Observation. In the tree of a prefix code only the leaves have a label.

Pf. An encoding of x is a prefix of an encoding of y if and only if the path of x is a prefix of the path of y .

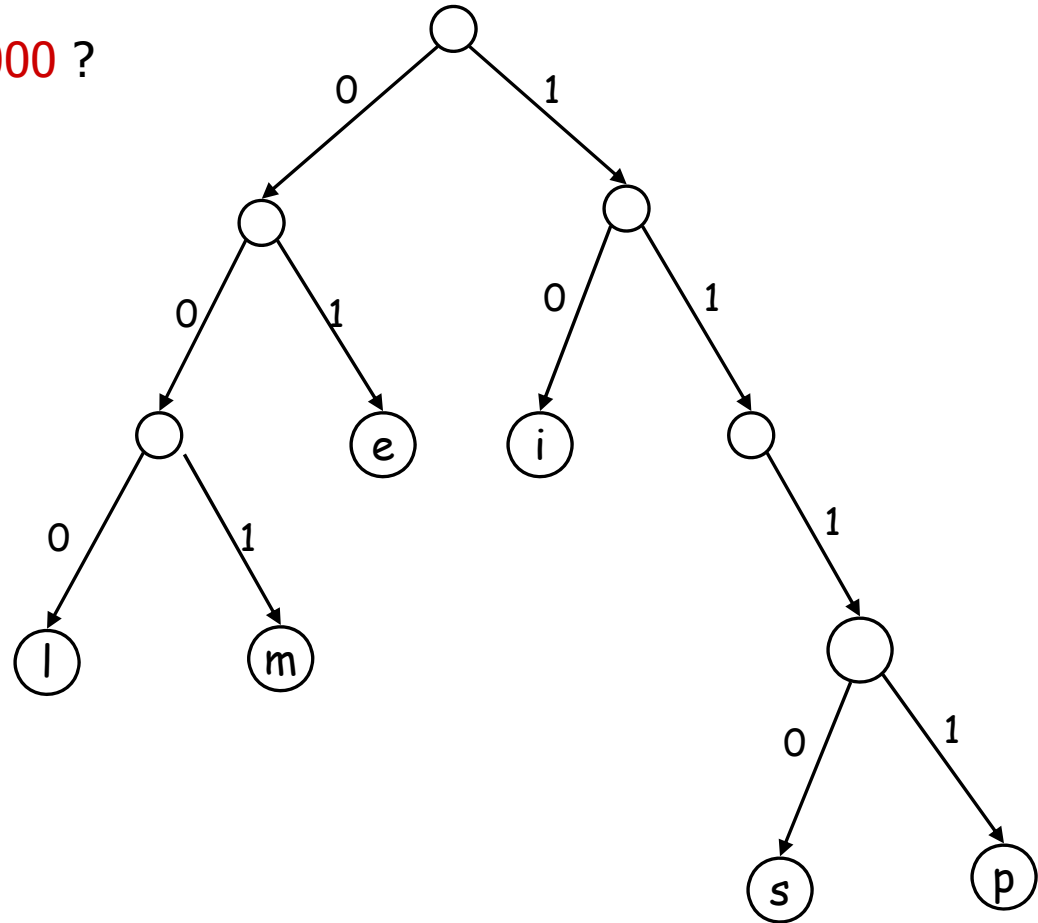
Representing Prefix Codes using Binary Trees

Q. What is the meaning of
11101000111101000 ?



Representing Prefix Codes using Binary Trees

Q. What is the meaning of
11101000111101000 ?



Representing Prefix Codes using Binary Trees

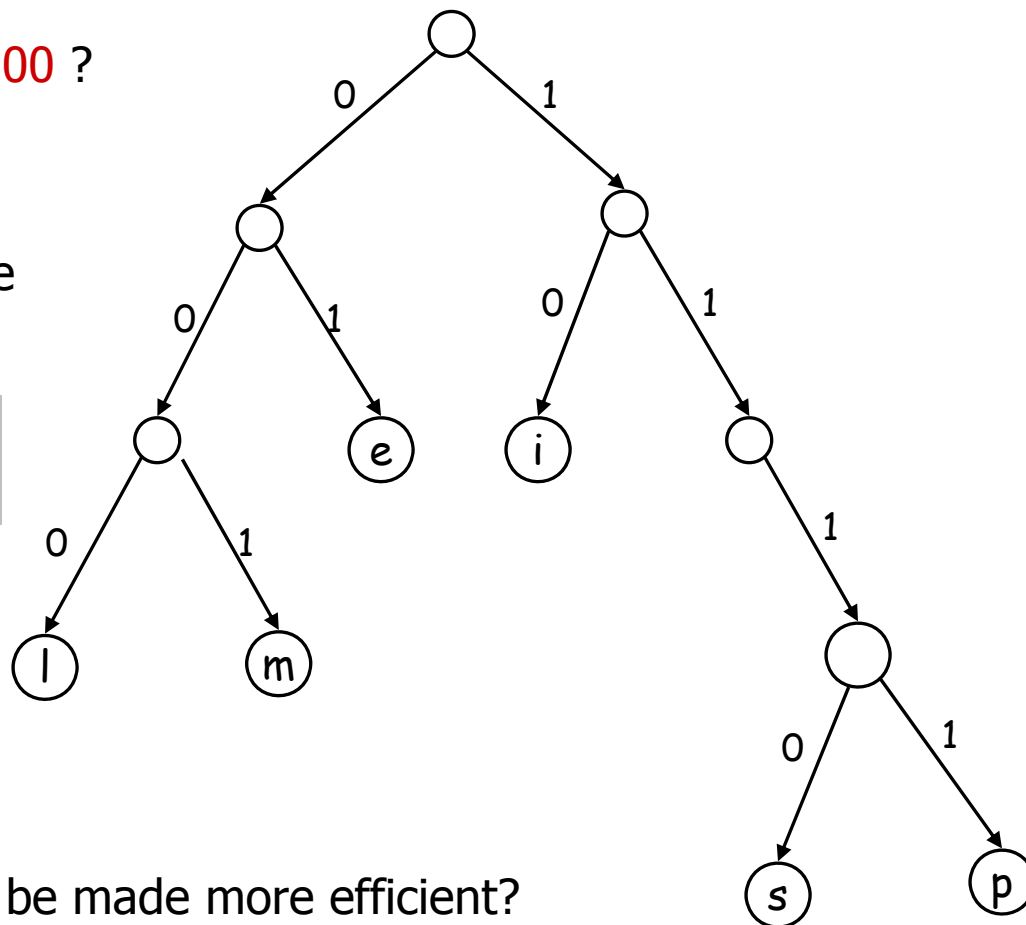
Q. What is the meaning of

11101000111101000 ?

A. “simpel”

Efficiency in terms of the tree representation is defined by

$$ABL(T) = \sum_{x \in S} f_x \cdot \text{depth}_T(x)$$



Q. How can this prefix code be made more efficient?

Representing Prefix Codes using Binary Trees

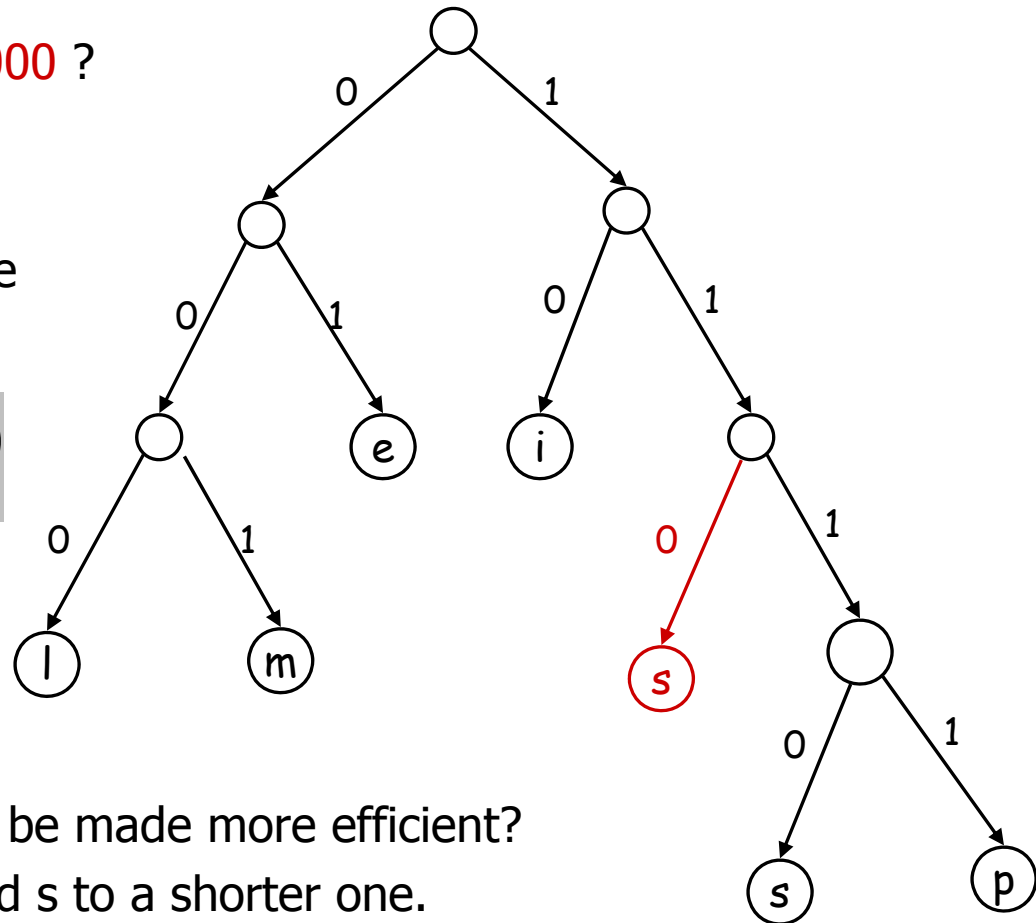
Q. What is the meaning of

11101000111101000 ?

A. “simpel”

Efficiency in terms of the tree representation is defined by

$$ABL(T) = \sum_{x \in S} f_x \cdot \text{depth}_T(x)$$



Q. How can this prefix code be made more efficient?

A. Change encoding of p and s to a shorter one.

This tree is now full.

Representing Prefix Codes using Binary Trees

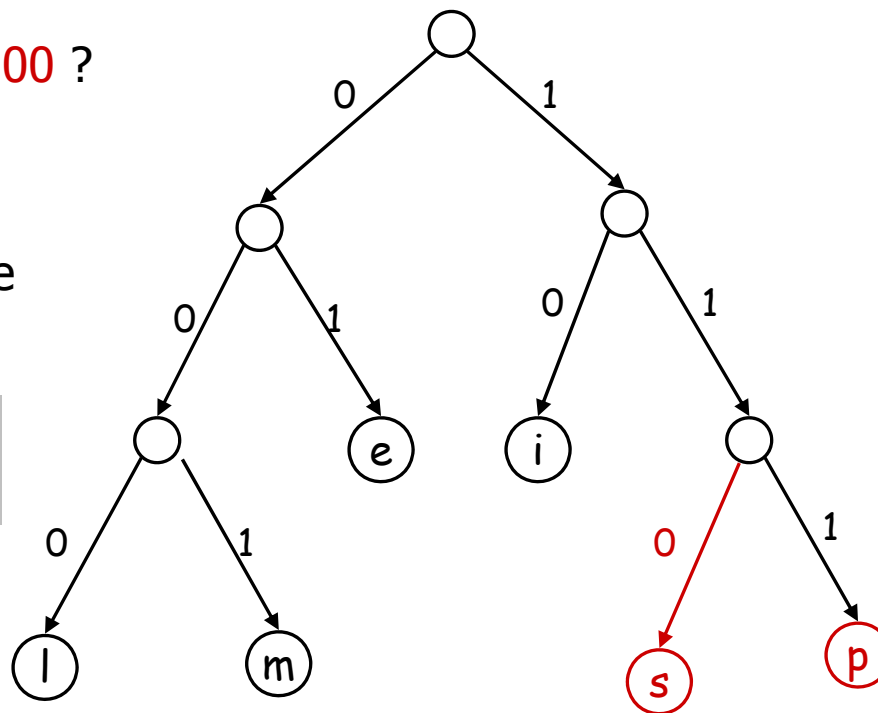
Q. What is the meaning of

11101000111101000 ?

A. “simple”

Efficiency in terms of the tree representation is defined by

$$ABL(T) = \sum_{x \in S} f_x \cdot \text{depth}_T(x)$$



Q. How can this prefix code be made more efficient?

A. Change encoding of p and s to a shorter one.

This tree is now **full**.

Q. When can we improve a prefix code (and when is it optimal)?

Representing Prefix Codes using Binary Trees

Definition. A tree is **full** if every node that is not a leaf has two children.

remember: only leaves have a label!

Claim. The binary tree corresponding to the **optimal** prefix code is full.

Pf.

Representing Prefix Codes using Binary Trees

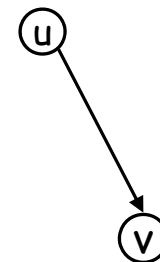
Definition. A tree is **full** if every node that is not a leaf has two children.

remember: only leaves have a label!

Claim. The binary tree corresponding to the **optimal** prefix code is full.

Pf. (by contradiction)

- Suppose T is binary tree of optimal prefix code and is not full.
- Thus there is a node u (without label) with only one child v .



- Clearly this new tree T' has a smaller ABL than T . Contradiction.

Representing Prefix Codes using Binary Trees

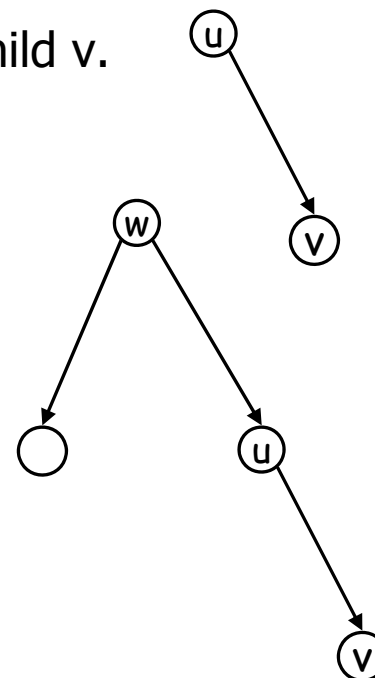
Definition. A tree is **full** if every node that is not a leaf has two children.

remember: only leaves have a label!

Claim. The binary tree corresponding to the **optimal** prefix code is full.

Pf. (by contradiction)

- Suppose T is binary tree of optimal prefix code and is not full.
- Thus there is a node u (without label) with only one child v .
- Case 1: u is the root
- Case 2: u is not the root



Representing Prefix Codes using Binary Trees

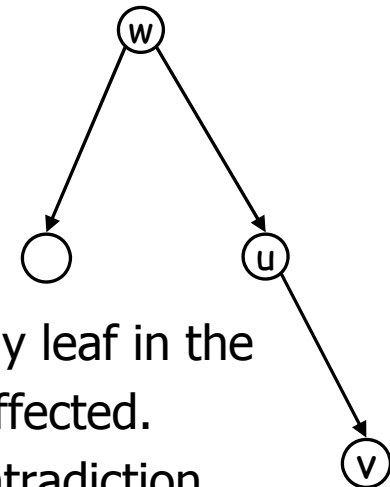
Definition. A tree is **full** if every node that is not a leaf has two children.

remember: only leaves have a label!

Claim. The binary tree corresponding to the **optimal** prefix code is full.

Pf. (by contradiction)

- Suppose T is binary tree of optimal prefix code and is not full.
- Thus there is a node u (without label) with only one child v . (v)
- Case 1: u is the root; create T' : delete u and use v as the root
- Case 2: u is not the root; create T' :
 - let w be the parent of u
 - delete u and make v be a child of w in place of u
- In both cases the number of bits needed to encode any leaf in the subtree of v is decreased. The rest of the tree is not affected.
- Clearly this new tree T' has a smaller ABL than T . Contradiction.



Optimal Prefix Codes: False Start

Q. Where in the tree of an *optimal* prefix code should letters be placed with a high frequency?

Optimal Prefix Codes: False Start

Q. Where in the tree of an *optimal* prefix code should letters be placed with a high frequency?

A. Near the top.

Q. How to create an optimal prefix code (tree) in a greedy way? (1 min)

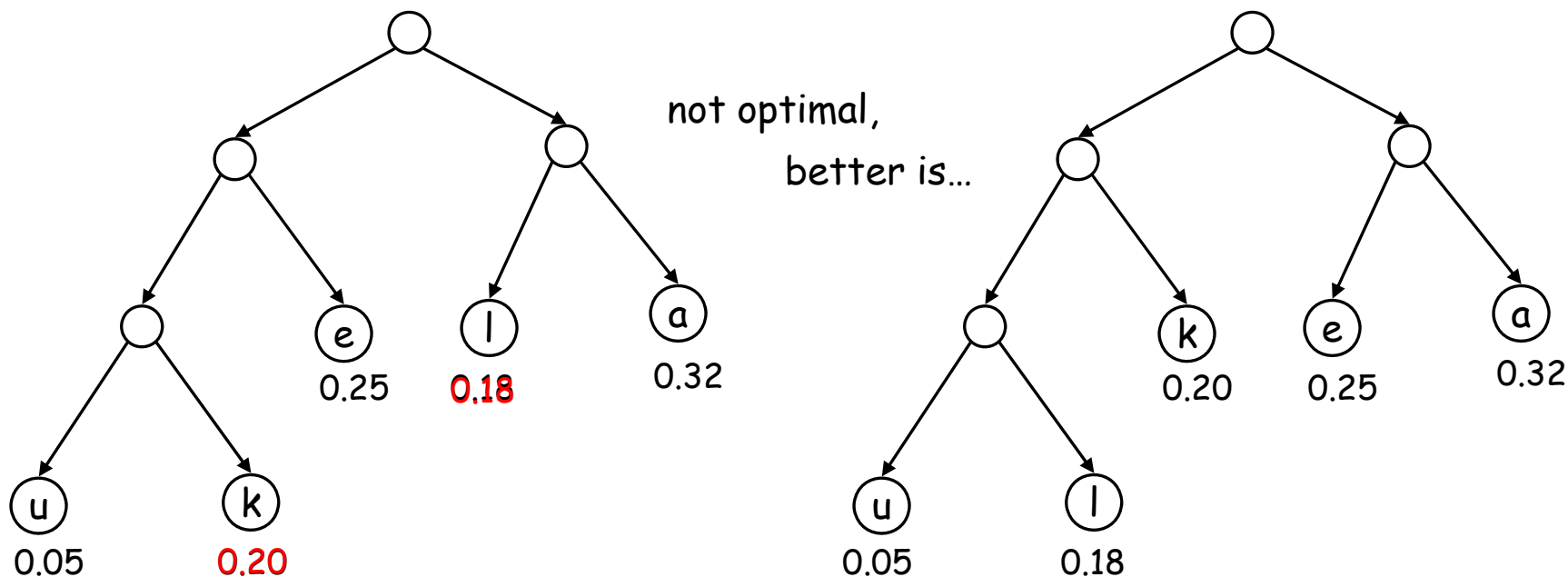
Optimal Prefix Codes: False Start

Q. Where in the tree of an *optimal* prefix code should letters be placed with a high frequency?

A. Near the top.

Greedy template (1). Create tree **top-down**, split S into two sets S_1 and S_2 with (almost) equal frequencies. Recursively build tree for S_1 and S_2 .

[Shannon-Fano, 1949] $f_a=0.32$, $f_e=0.25$, $f_k=0.20$, $f_l=0.18$, $f_u=0.05$



Optimal Prefix Codes: Huffman Encoding

Observation. Lowest frequency items should be at the lowest level in tree of optimal prefix code.

Observation. For $n > 1$, the lowest level always contains at least two leaves.

Observation. The order in which items appear in a level does not matter.

Q. How to find an *optimal* prefix code?

Optimal Prefix Codes: Huffman Encoding

Observation. Lowest frequency items should be at the lowest level in tree of optimal prefix code.

Observation. For $n > 1$, the lowest level always contains at least two leaves.

Observation. The order in which items appear in a level does not matter.

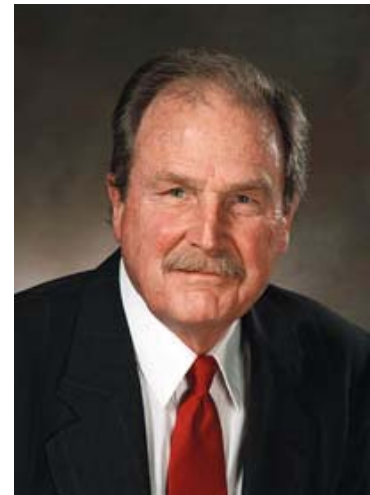
Claim (Siblings). There is an optimal prefix code with tree T^* where the **two lowest-frequency letters** are assigned to leaves that are siblings in T^* .

Greedy template (2). Create tree **bottom-up**.

Make two leaves for two lowest-frequency letters y and z .

Recursively build tree for the rest using a meta-letter for yz .

[Huffman, 1952]



Optimal Prefix Codes: Huffman Encoding

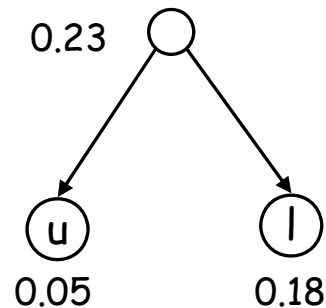
Greedy template (2). Create tree **bottom-up**.

Make two leaves for two lowest-frequency letters y and z.

Recursively build tree for the rest using a meta-letter for yz.

[Huffman, 1952]

$f_a=0.32$, $f_e=0.25$, $f_k=0.20$, $f_l=0.18$, $f_u=0.05$



Optimal Prefix Codes: Huffman Encoding

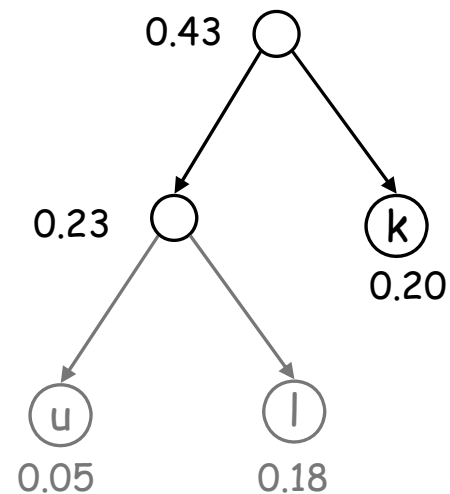
Greedy template (2). Create tree **bottom-up**.

Make two leaves for two lowest-frequency letters y and z.

Recursively build tree for the rest using a meta-letter for yz.

[Huffman, 1952]

$f_a=0.32$, $f_e=0.25$, $f_k=0.20$, $f_\omega=0.23$



Optimal Prefix Codes: Huffman Encoding

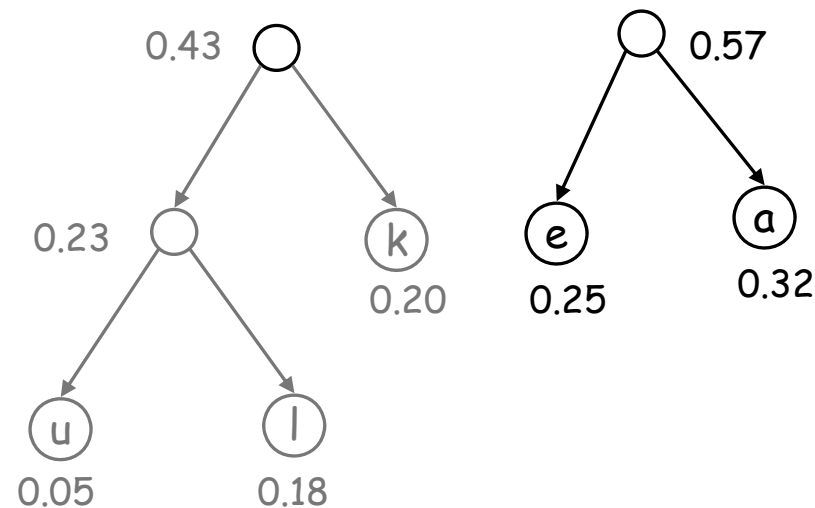
Greedy template (2). Create tree **bottom-up**.

Make two leaves for two lowest-frequency letters y and z.

Recursively build tree for the rest using a meta-letter for yz.

[Huffman, 1952]

$f_a=0.32$, $f_e=0.25$, $f_w=0.43$



Optimal Prefix Codes: Huffman Encoding

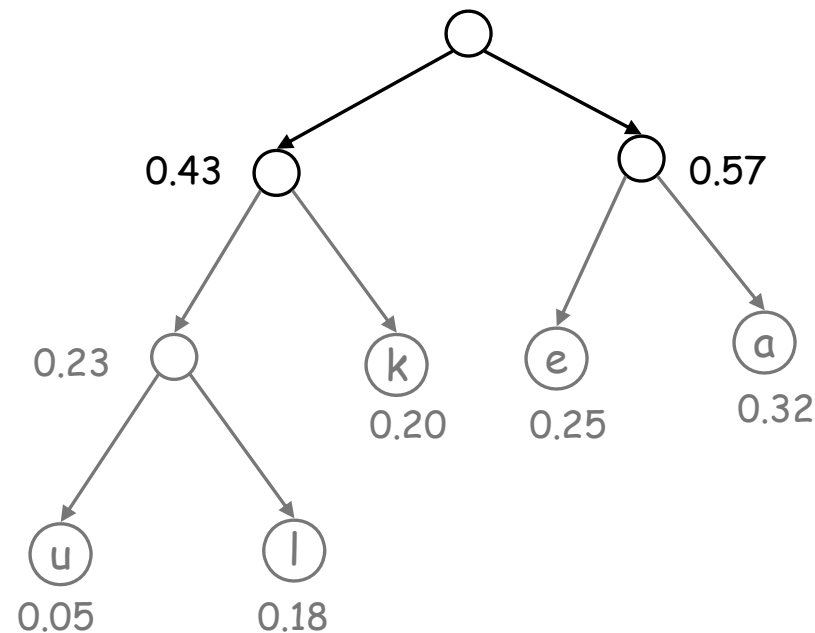
Greedy template (2). Create tree **bottom-up**.

Make two leaves for two lowest-frequency letters y and z.

Recursively build tree for the rest using a meta-letter for yz.

[Huffman, 1952]

$$f_{\omega_1} = 0.43 \quad f_{\omega_2} = 0.57$$



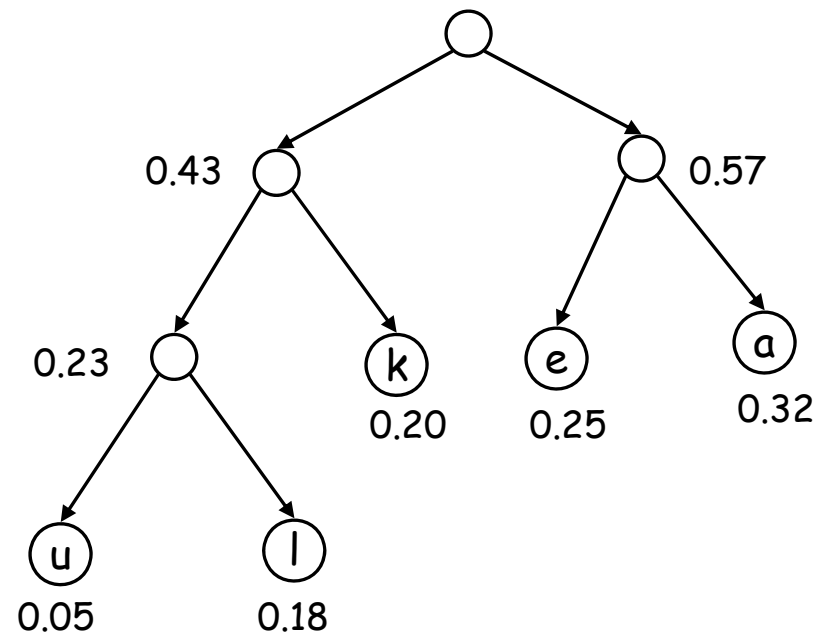
Optimal Prefix Codes: Huffman Encoding

Greedy template (2). Create tree **bottom-up**.

Make two leaves for two lowest-frequency letters y and z.

Recursively build tree for the rest using a meta-letter for yz.

[Huffman, 1952]



Optimal Prefix Codes: Huffman Encoding

```
Huffman(S) {  
  if |S|=2 {  
    return tree with root and 2 leaves  
  } else {  
    let y and z be lowest-frequency letters in S  
    S' = S  
    remove y and z from S'  
    insert new letter  $\omega$  in S' with  $f_{\omega}=f_y+f_z$   
    T' = Huffman(S')  
    T = add two children y and z to leaf  $\omega$  from T'  
    return T  
  }  
}
```

$O(1)$

$O(\log n)$

$O(1)$

$O(\log n)$

$T(n-1)$

$O(1)$

Q. How to implement finding lowest-frequency letters efficiently?

A. Use priority queue for S

Q. What is the time complexity?

A. $T(n) = T(n-1) + O(\log n)$

$T(n) = O(n \log n)$

Huffman Code Construction

Character count in text.

Char	Freq
E	125
T	93
A	80
O	76
I	73
N	71
S	65
R	61
H	55
L	41
D	40
C	31
U	27

Huffman Code Construction

Char	Freq
E	125
T	93
A	80
O	76
I	73
N	71
S	65
R	61
H	55
L	41
D	40
C	31
U	27

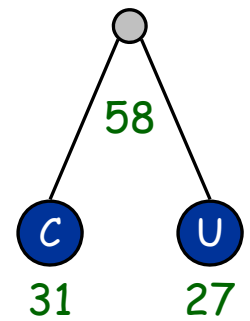
C
31

U
27

Huffman Code Construction

Char	Freq
E	125
T	93
A	80
O	76
I	73
N	71
S	65
R	61
	58
H	55
L	41
D	40

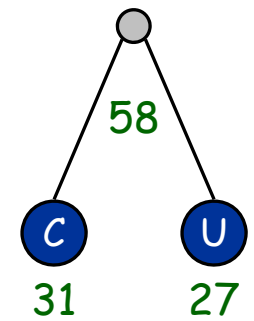
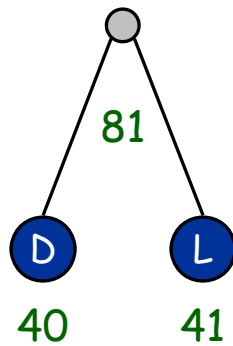
C	31
U	27



Huffman Code Construction

Char	Freq
E	125
T	93
	81
A	80
O	76
I	73
N	71
S	65
R	61
	58
H	55

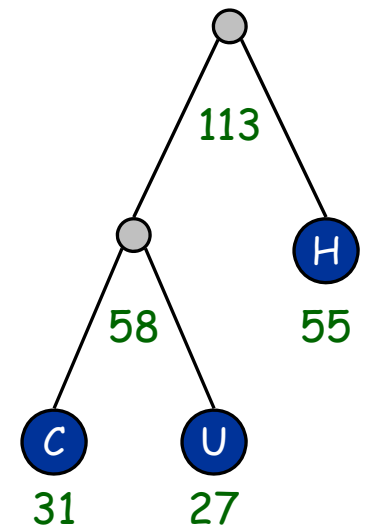
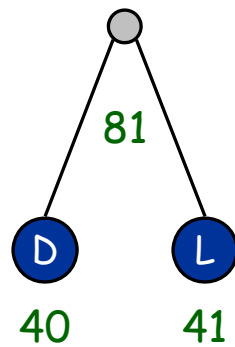
L	41
D	40



Huffman Code Construction

Char	Freq
E	125
	113
T	93
	81
A	80
O	76
I	73
N	71
S	65
R	61

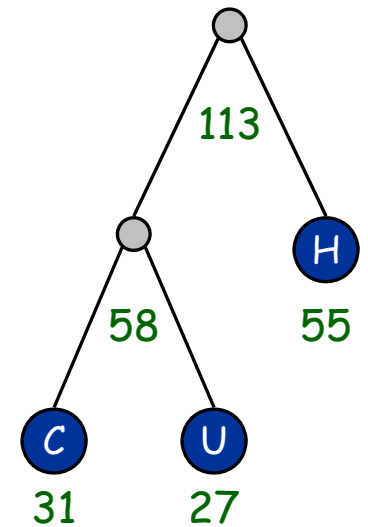
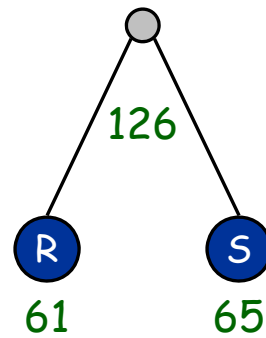
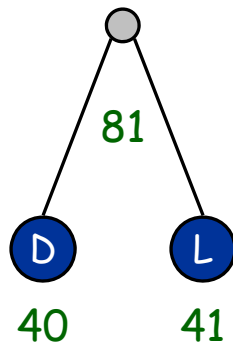
	58
H	55



Huffman Code Construction

Char	Freq
	126
E	125
	113
T	93
	81
A	80
O	76
I	73
N	71

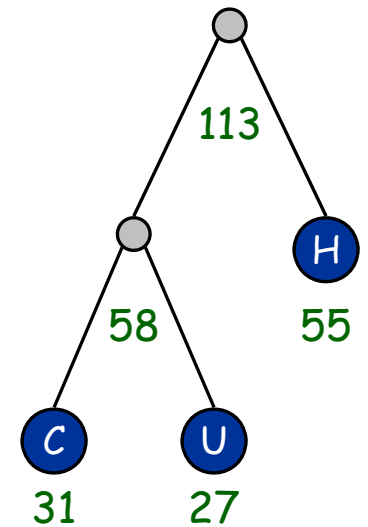
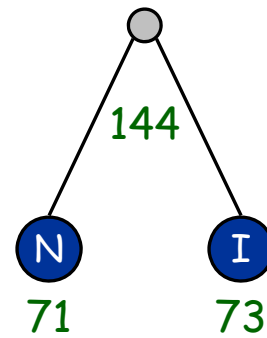
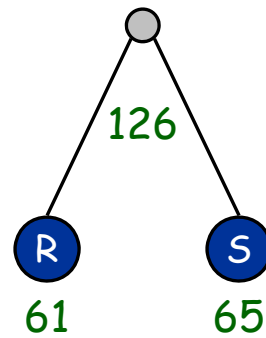
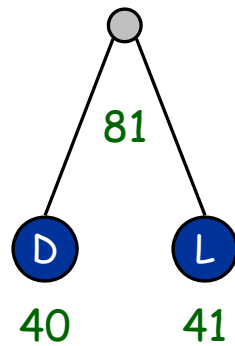
S	65
R	61



Huffman Code Construction

Char	Freq
	144
	126
E	125
	113
T	93
	81
A	80
O	76

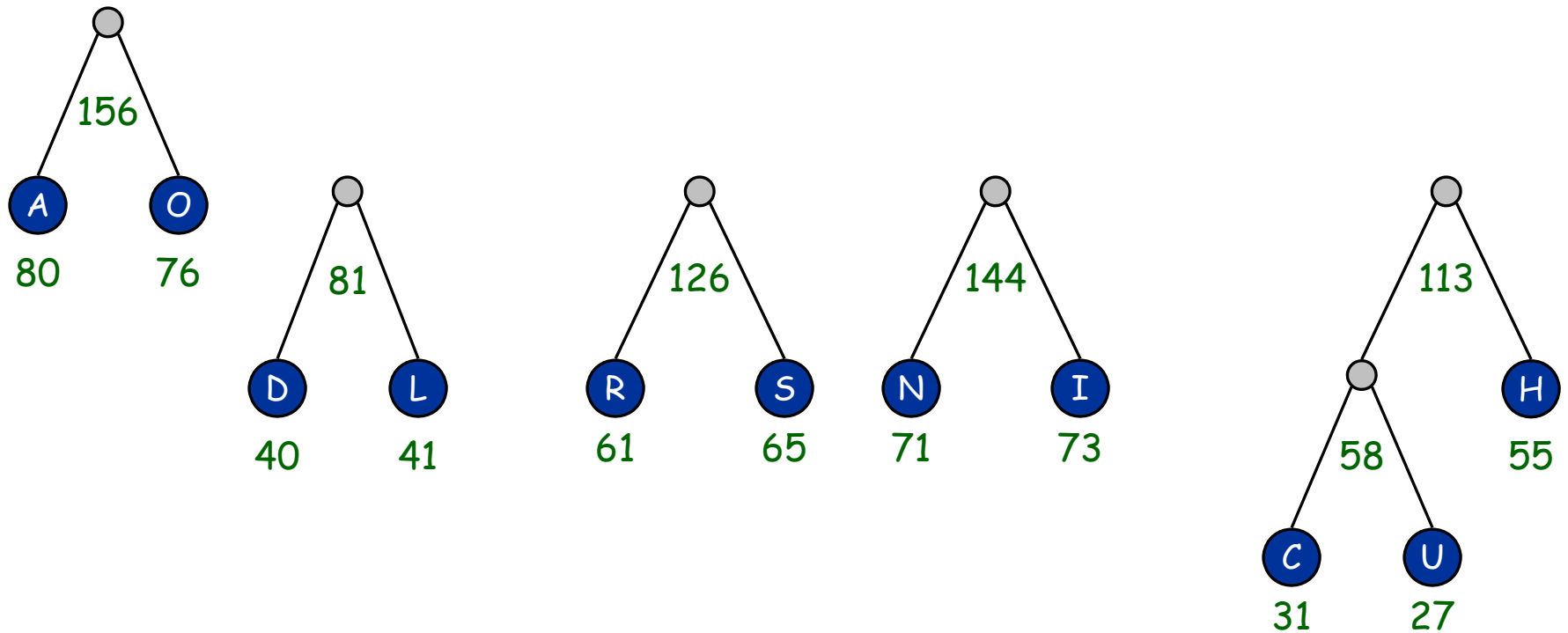
I	73
N	71



Huffman Code Construction

Char	Freq
	156
	144
	126
E	125
	113
T	93
	81

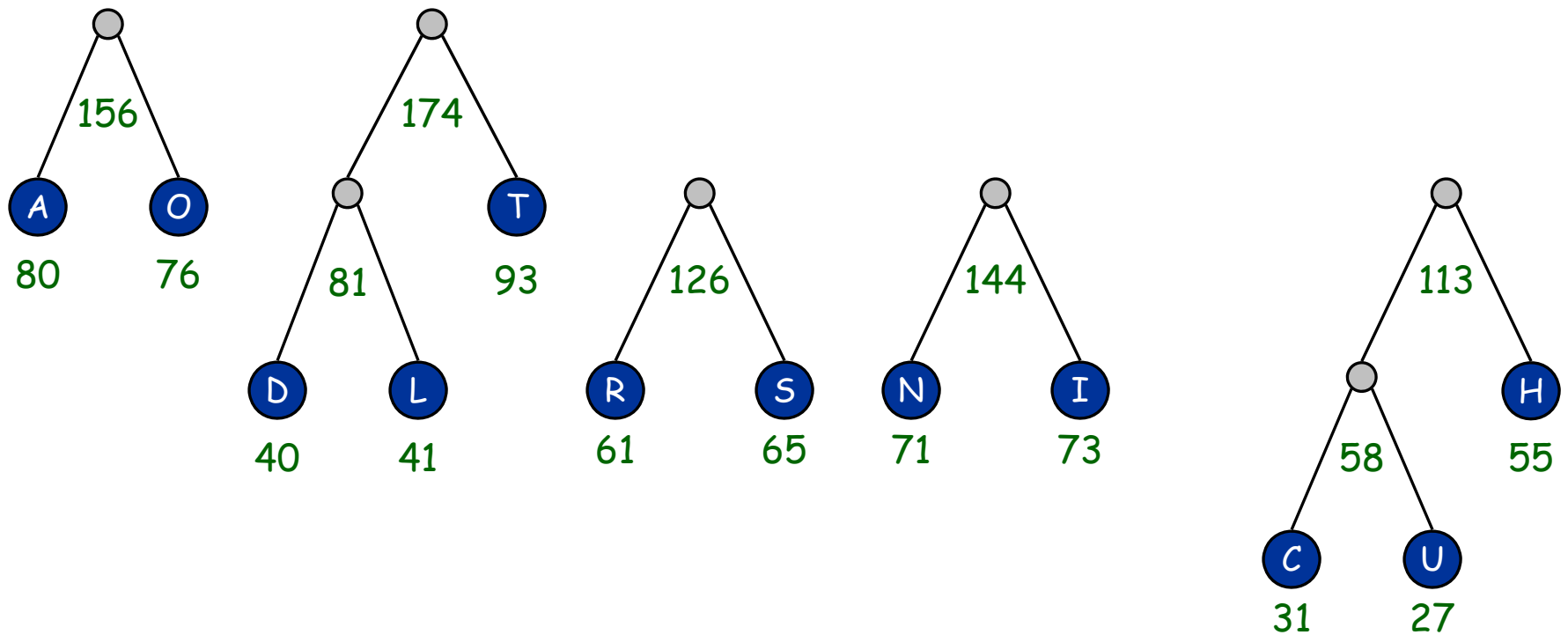
A	80
O	76



Huffman Code Construction

Char	Freq
	174
	156
	144
	126
E	125
	113

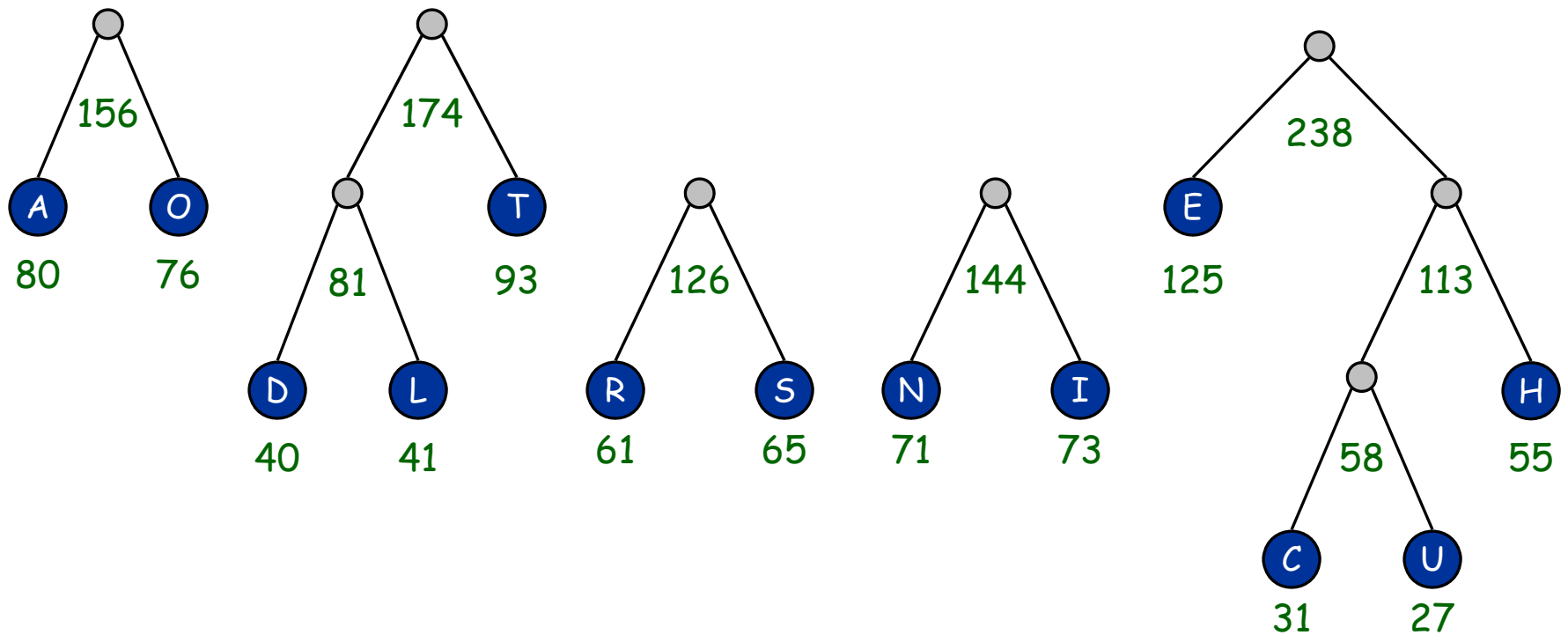
T	93
	81



Huffman Code Construction

Char	Freq
	238
	174
	156
	144
	126

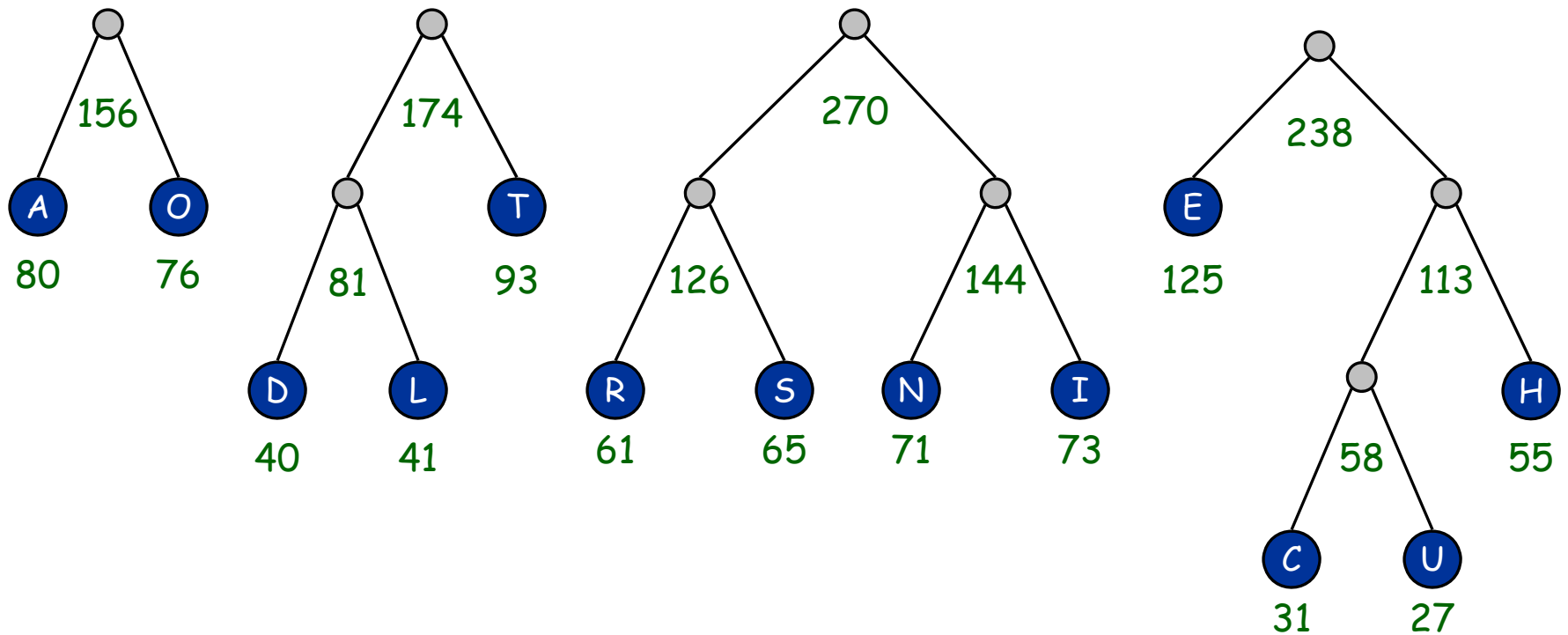
E	125
	113



Huffman Code Construction

Char	Freq
	270
	238
	174
	156

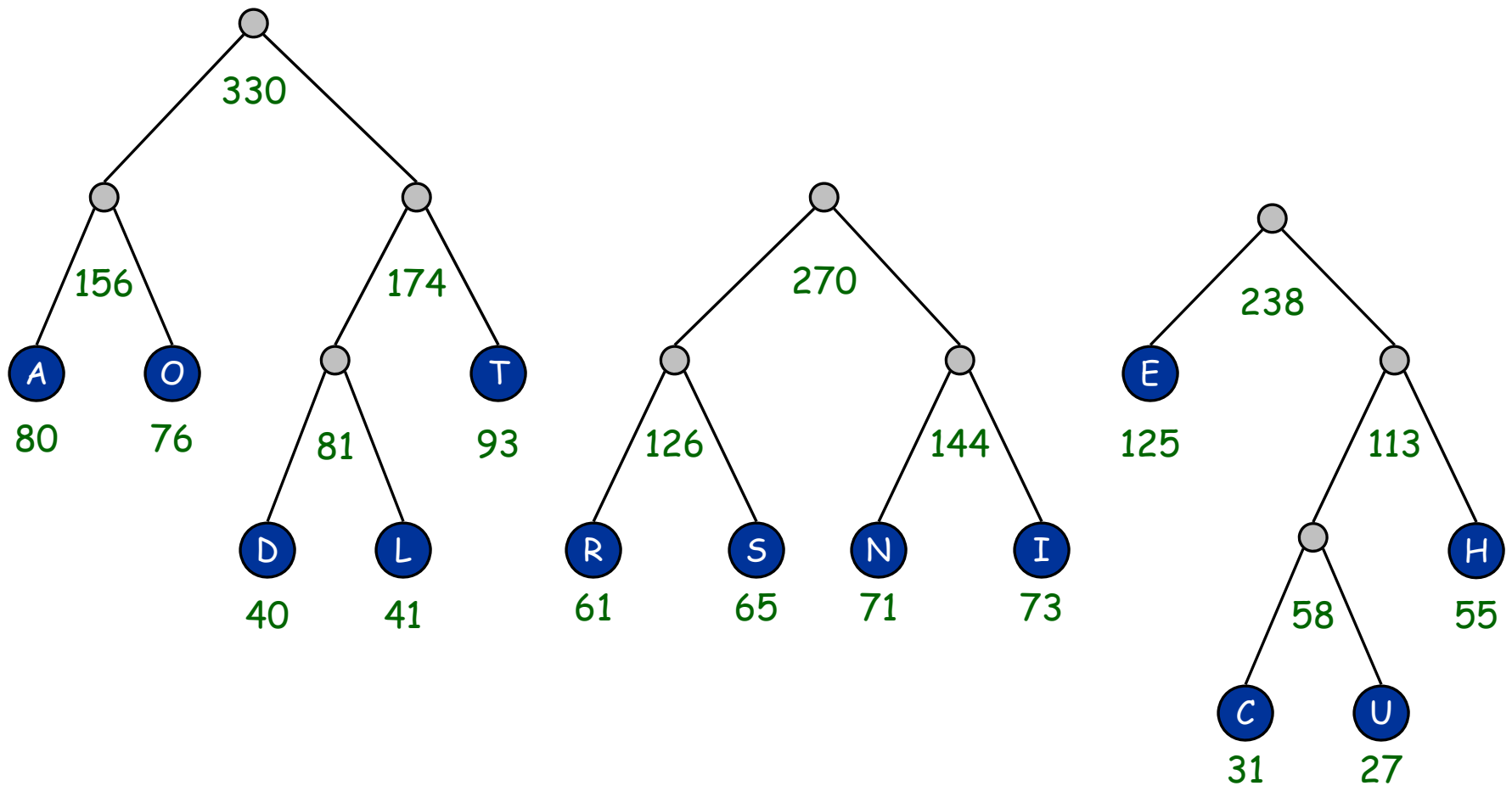
	144
	126



Huffman Code Construction

Char	Freq
	330
	270
	238

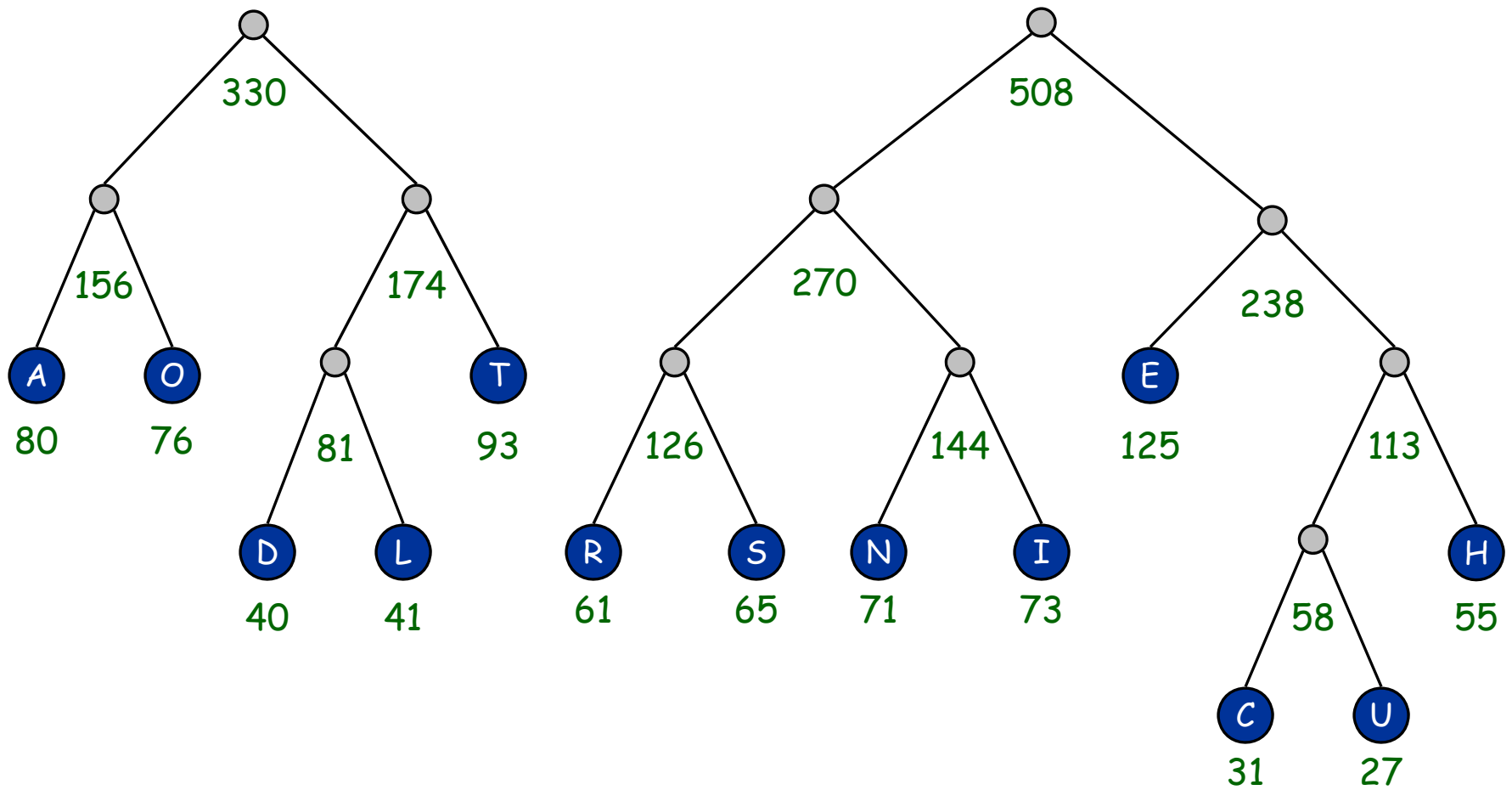
	174
	156



Huffman Code Construction

Char	Freq
	508
	330

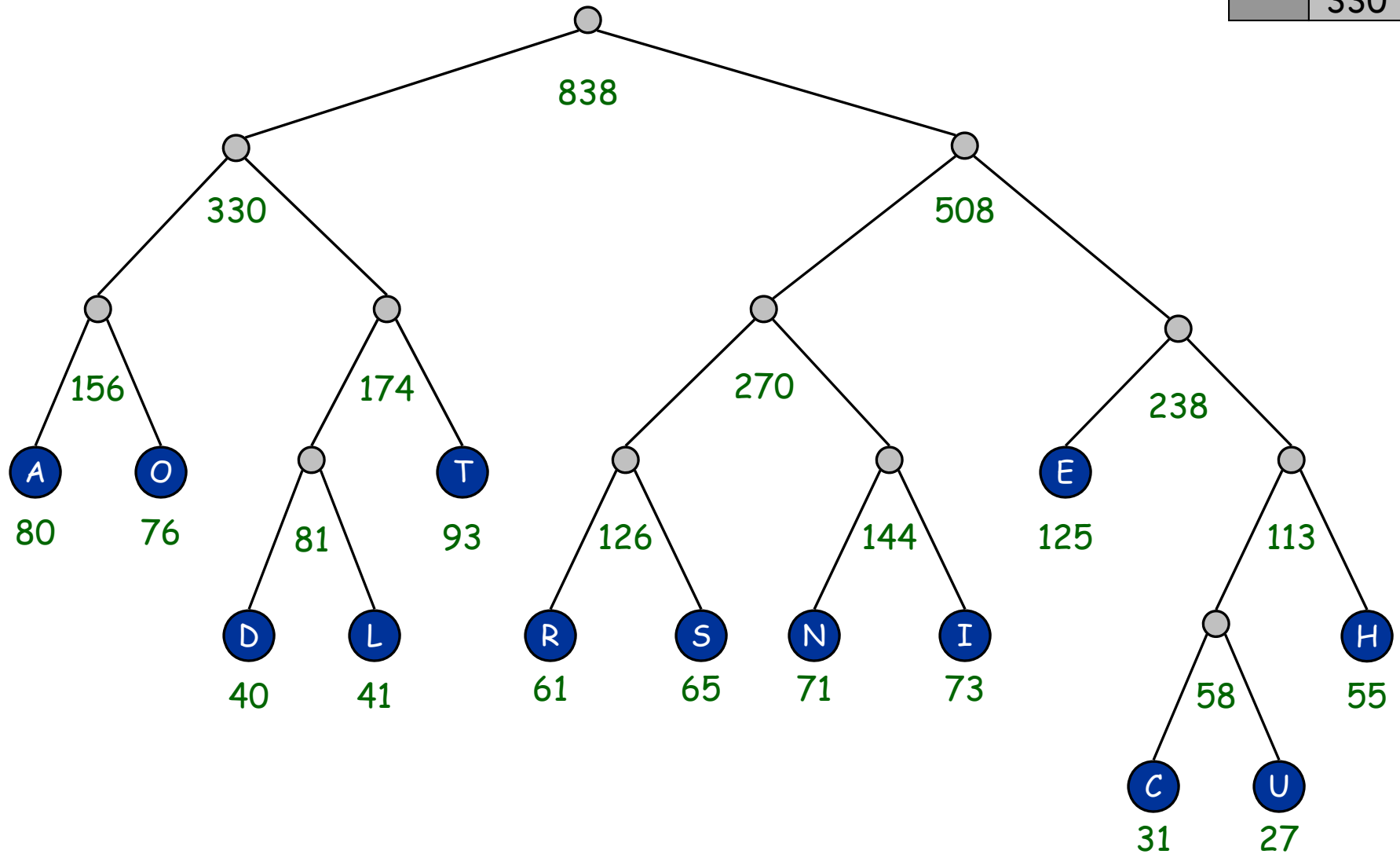
	270
	238



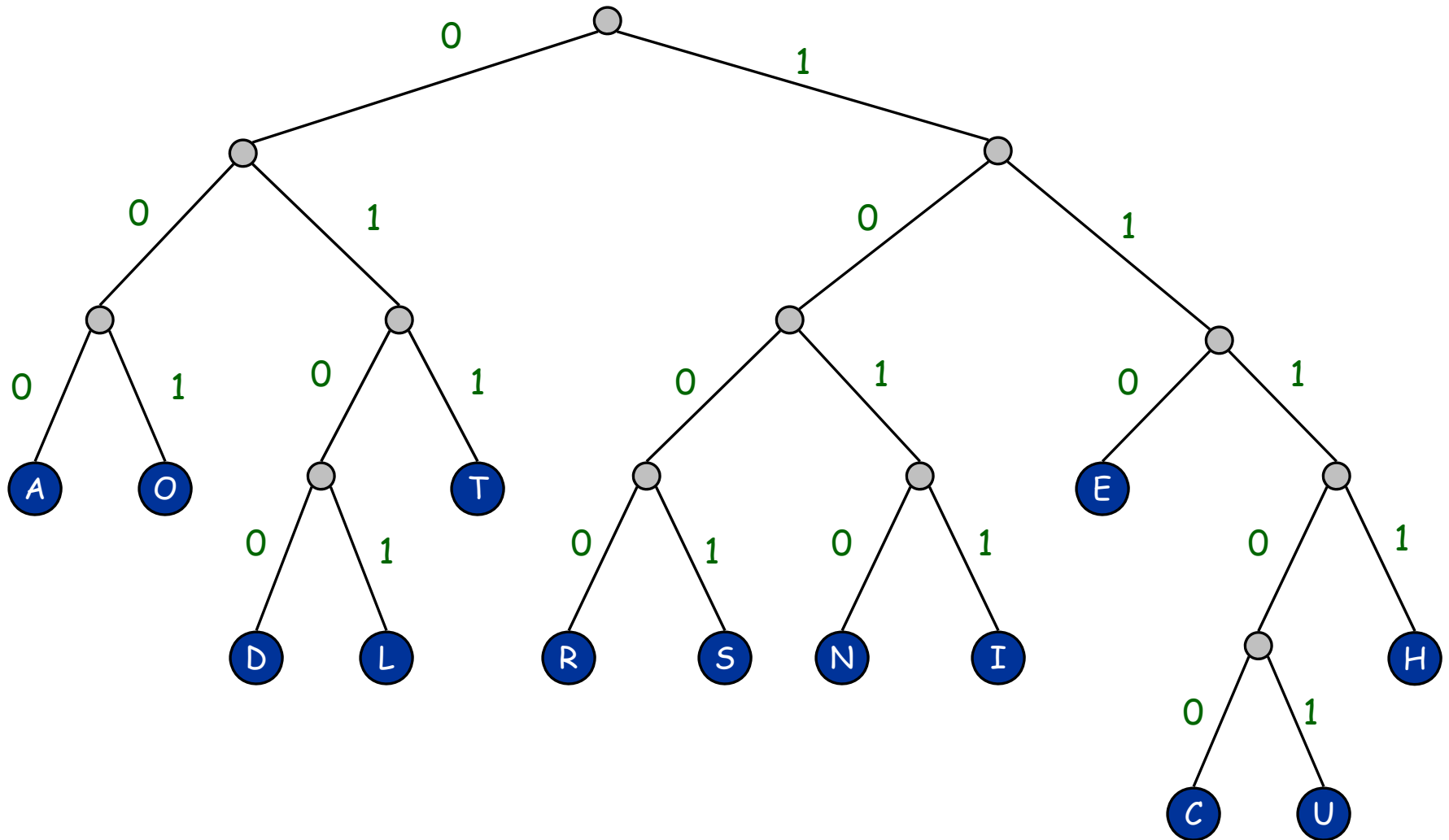
Huffman Code Construction

Char	Freq
	838

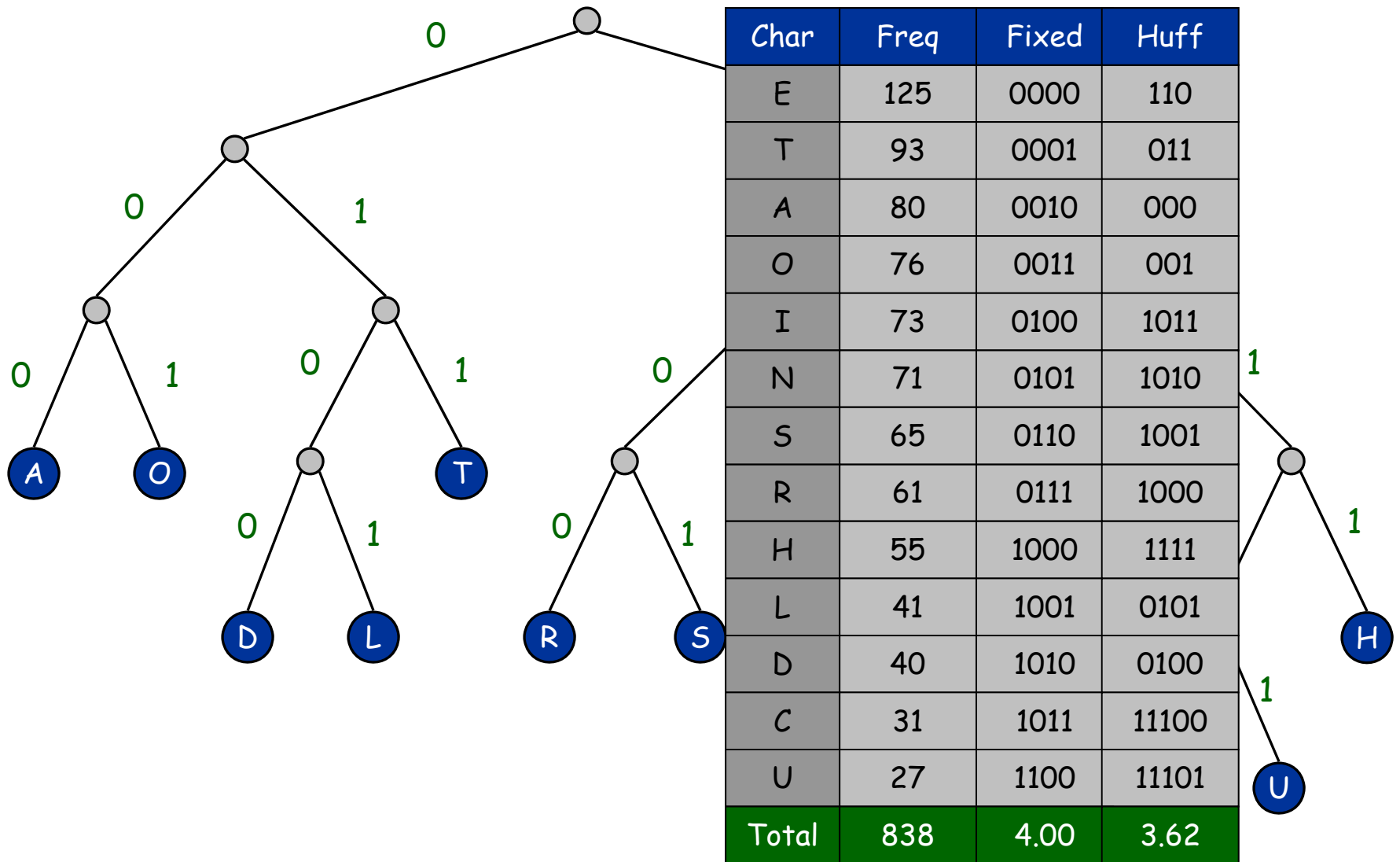
	508
	330



Huffman Code Construction



Huffman Code Construction



Claim: There is an optimal prefix code, with corresponding tree T^* , in which the two lowest frequency letters are assigned to leaves that are siblings in T^* .

Proof Sketch:

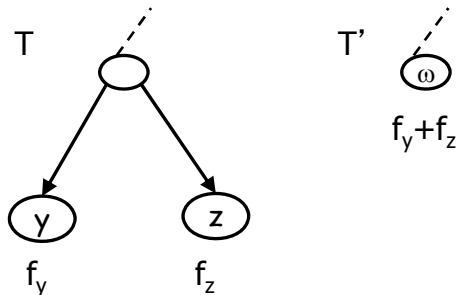
- (1) Optimal tree T^* is full.
- (2) Lowest frequency letter v is assigned to highest depth node, to minimize $ABL(T^*)$.
- (3) Sibling of v must be a leaf (or else, you can create an internal node with at least two leaves at higher depth with potentially larger frequencies).
- (4) Lowest frequencies letters are assigned to leaves that are siblings.

Huffman Encoding: Greedy Analysis

Claim. $ABL(T') = ABL(T) - f_\omega$ when T' is T with y and z replaced by ω

Pf.

$$\begin{aligned}
 ABL(T) &= \sum_{x \in S} f_x \cdot \text{depth}_T(x) \\
 &= f_y \cdot \text{depth}_T(y) + f_z \cdot \text{depth}_T(z) + \sum_{x \in S, x \neq y, z} f_x \cdot \text{depth}_T(x) \\
 &= (f_y + f_z) \cdot (1 + \text{depth}_T(\omega)) + \sum_{x \in S, x \neq y, z} f_x \cdot \text{depth}_T(x) \\
 &= f_\omega \cdot (1 + \text{depth}_T(\omega)) + \sum_{x \in S, x \neq y, z} f_x \cdot \text{depth}_T(x) \\
 &= f_\omega + \sum_{x \in S'} f_x \cdot \text{depth}_{T'}(x) \\
 &= f_\omega + ABL(T')
 \end{aligned}$$



This proof can be found on page 174.

Huffman Encoding: Greedy Analysis

Claim. Huffman code for S achieves the minimum ABL of any prefix code.

Pf. (by induction over $n=|S|$)

Base: For $n=2$, the shortest code uses 1-bit from root to two leaves.

Hypothesis: Suppose Huffman tree T' for S' of size $n-1$ with ω instead of y and z is optimal. (IH)

Step: (by contradiction)

- Suppose Huffman tree T for S is not optimal.
- So there is some other tree Z such that $ABL(Z) < ABL(T)$.
- The tree Z has leaves y and z that are siblings and have the lowest frequency (using Siblings Claim).
- Let Z' be Z with y and z deleted, and their former parent labeled ω .
- Similarly to T' derived for S' (with ω instead of y and z) by algorithm.
- We know that $ABL(Z') = ABL(Z) - f_\omega$, as well as $ABL(T') = ABL(T) - f_\omega$.
- But also $ABL(Z) < ABL(T)$, so $ABL(Z') < ABL(T')$.
- Contradiction with IH.