4.8 Huffman Codes

These lecture slides are supplied by Mathijs de Weerd

Data Compression

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

Data Compression

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. We can encode 2^5 different symbols using a fixed length of 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?

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?

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) = 11
c(e) = 01
c(k) = 001
c(l) = 10
c(u) = 000

Q. What is the meaning of 1001000001 ?

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

Q. What is the meaning of 1001000001 ?
A. "leuk"

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

Q. What is the meaning of 1001000001 ?
A. “leuk”

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

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

Q. What is the meaning of 1001000001 ?
A. "leuk"

Q. How to encode symbols such that on average encoding is smallest?
Q. Find a prefix code tha has the lowest possible average bits per letter.

Optimal Prefix Codes

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

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

We would like to find a prefix code that is has the lowest possible average bits per letter.

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

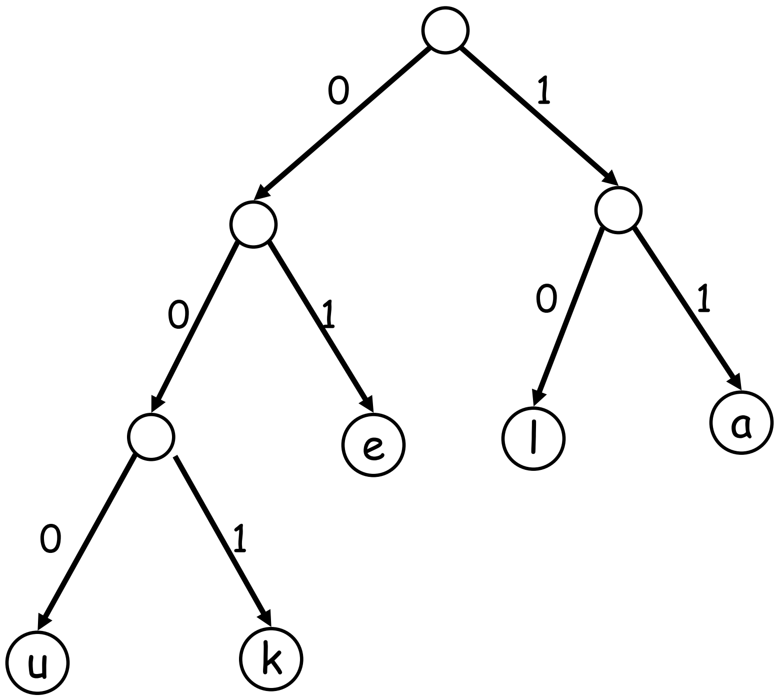
Ex. c(a) = 11
c(e) = 01
c(k) = 001
c(l) = 10
c(u) = 000

Q. What is the size of the encoded text?
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 : $2*f_a + 2*f_e + 2*f_k + 3*f_l + 3*f_u = 2.2$

Representing Prefix Codes using Binary Trees

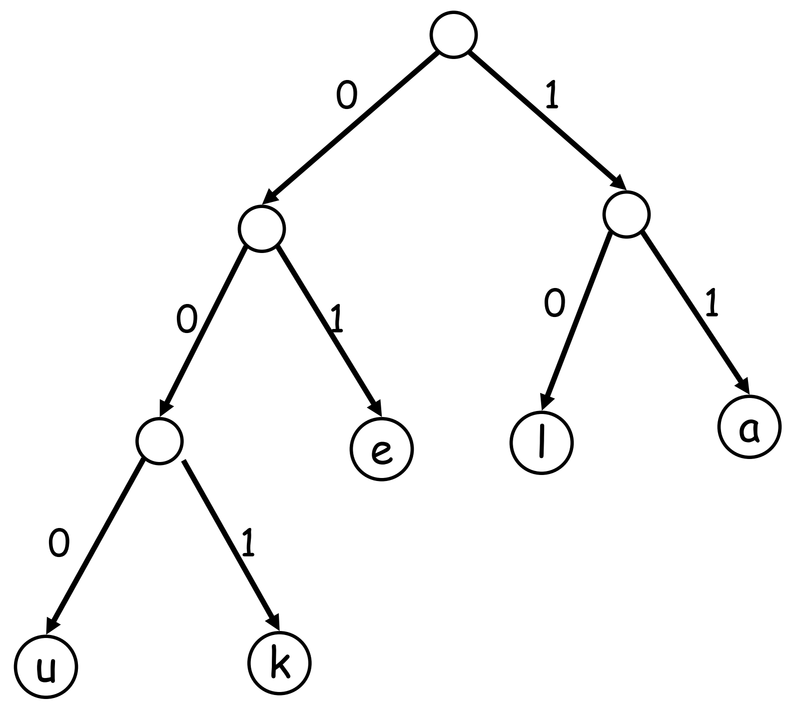
Ex. c(a) = 11
c(e) = 01
c(k) = 001
c(l) = 10
c(u) = 000



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

Representing Prefix Codes using Binary Trees

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

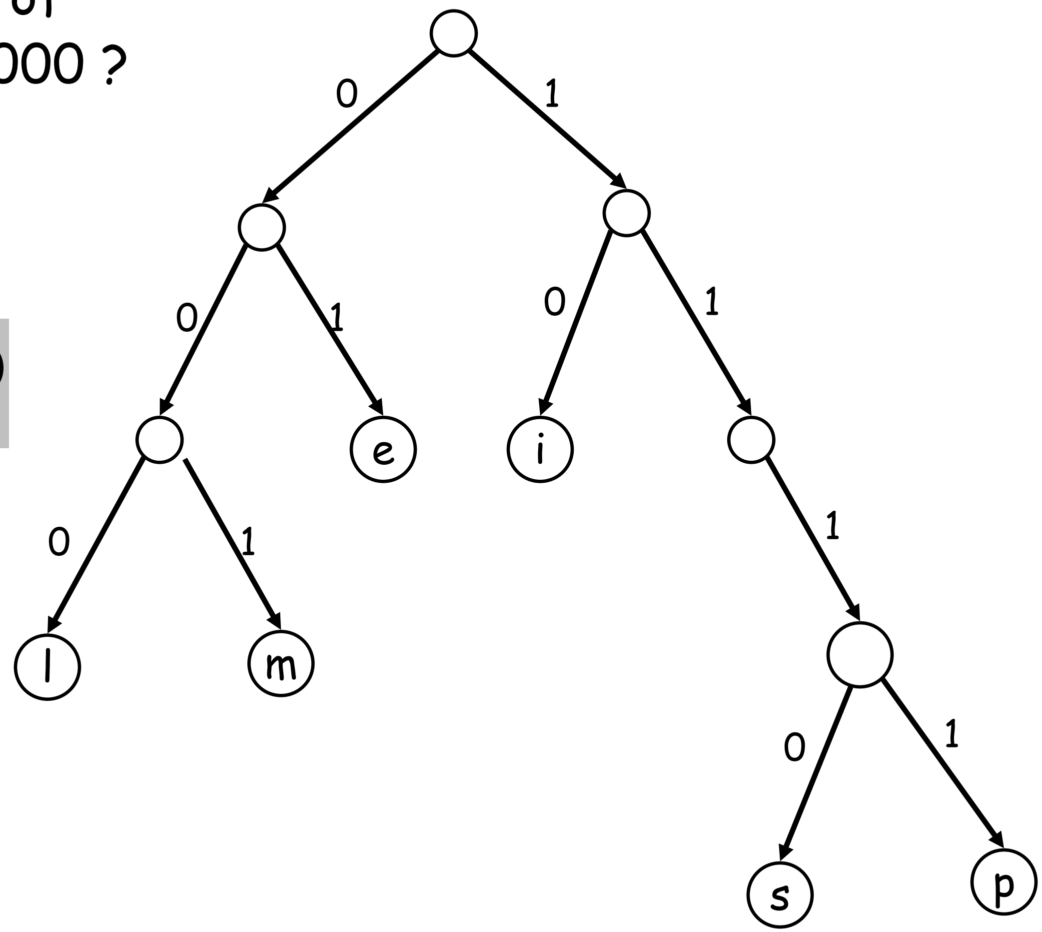


Q. How does the tree of a prefix code look?
A. Only the leaves have a label. (Unique decoding property)
Pf. An encoding of x is a prefix of an encoding of y if and only if the path of x is a prefix of the path of y.

Representing Prefix Codes using Binary Trees

Q. What is the meaning of
111010001111101000 ?

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

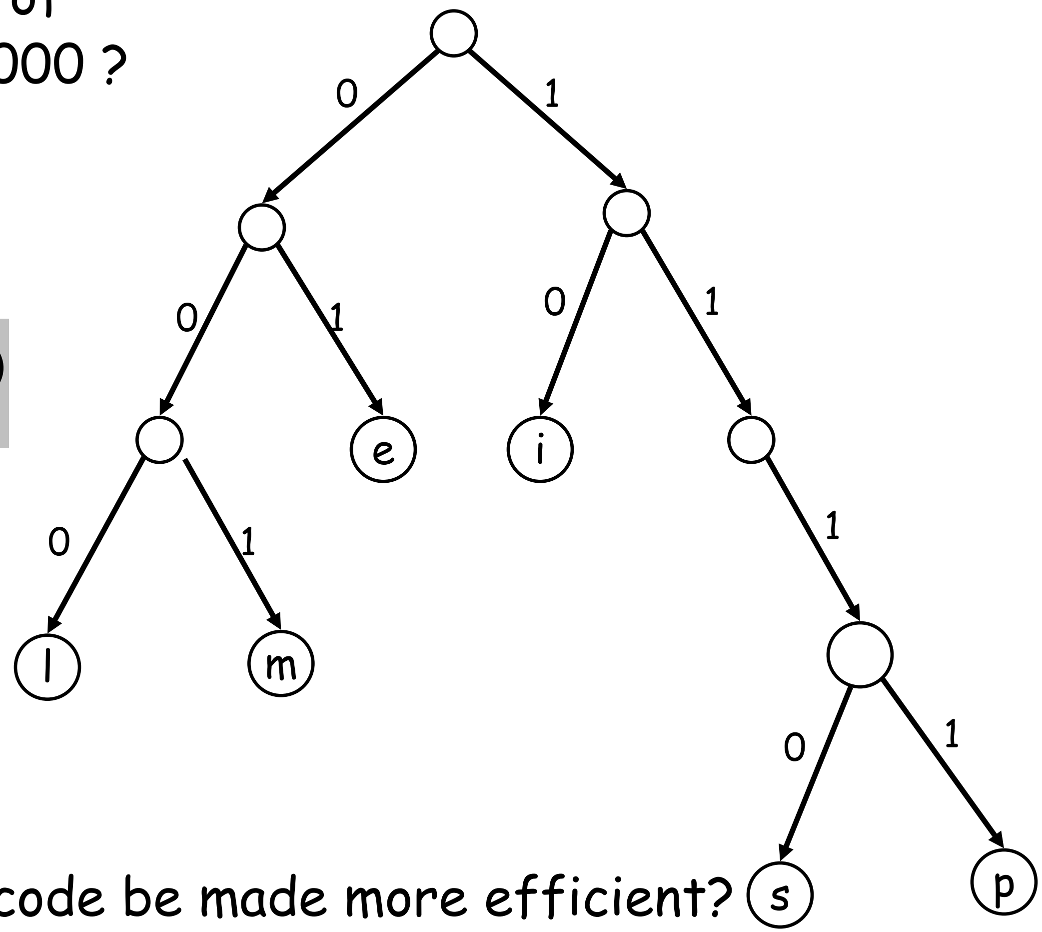


Representing Prefix Codes using Binary Trees

Q. What is the meaning of
111010001111101000 ?

A. "simpel"

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



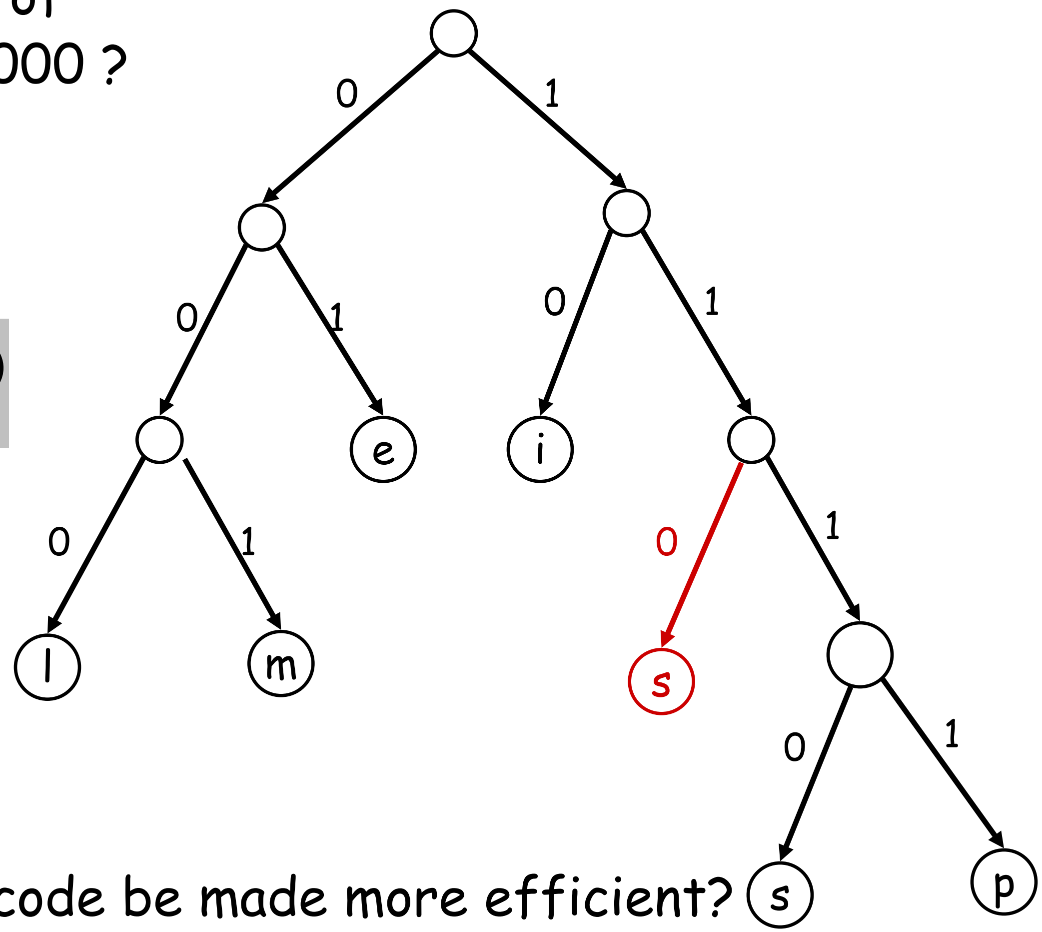
Q. How can this prefix code be made more efficient?

Representing Prefix Codes using Binary Trees

Q. What is the meaning of
111010001111101000 ?

A. "simpel"

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



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

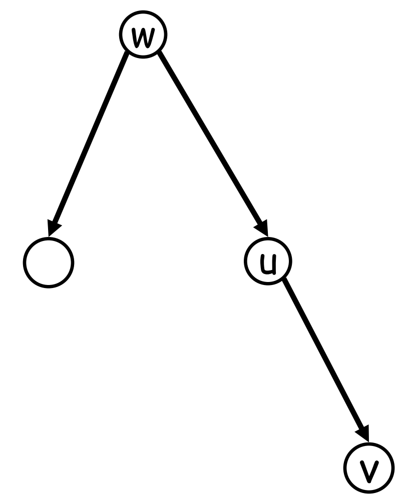
A. Change encoding of p and s to a shorter one.

This tree is now full.

Representing Prefix Codes using Binary Trees

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

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



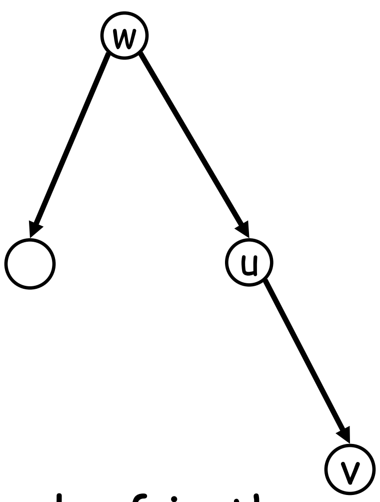
Representing Prefix Codes using Binary Trees

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

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.
- This means there is a node u with only one child v.
- Case 1: u is the root; delete u and use v as the root
- Case 2: u is not the root
  - 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.

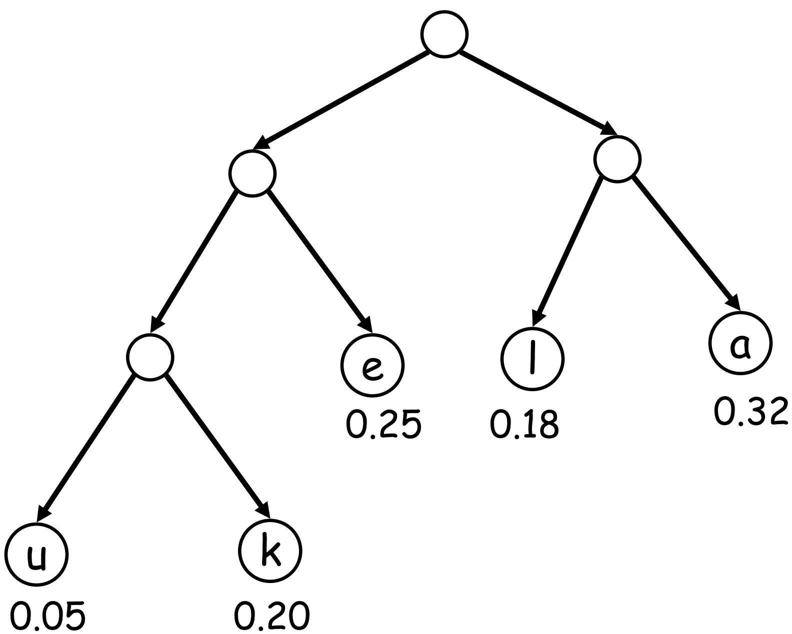
Optimal Prefix Codes: False Start

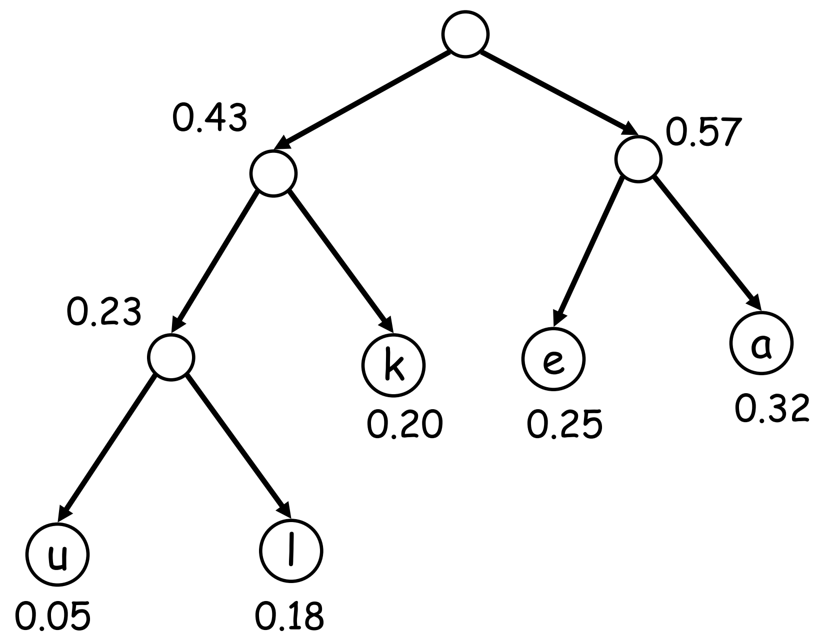
Q. Where in the tree of an optimal prefix code should letters be placed with a high frequency?

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.

Greedy template. 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

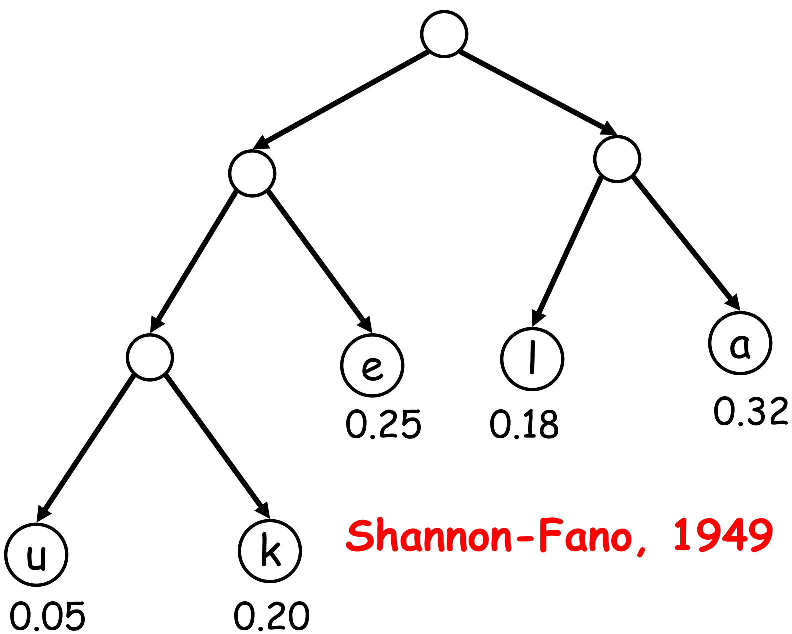




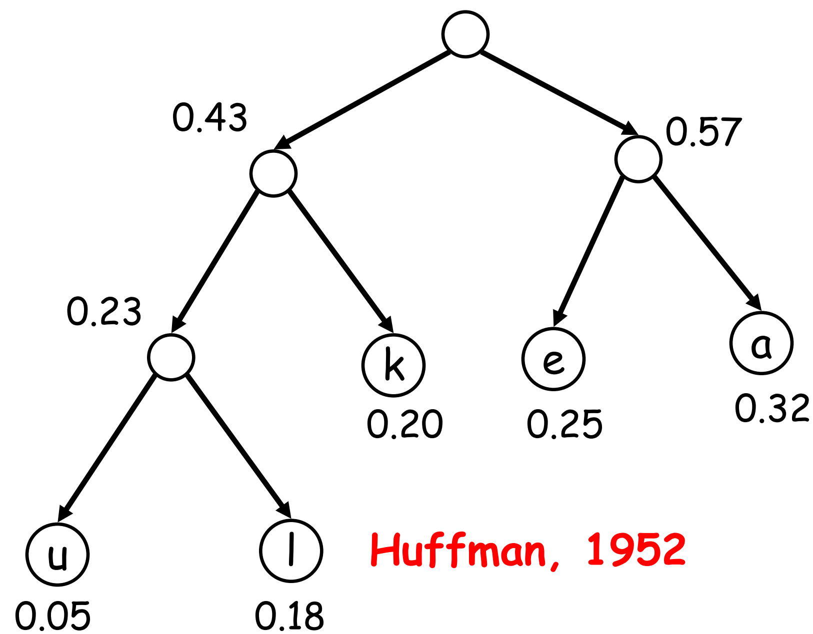
Optimal Prefix Codes: Huffman Encoding

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

Greedy template. 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



Shannon-Fano, 1949



Huffman, 1952

Optimal Prefix Codes: Huffman Encoding

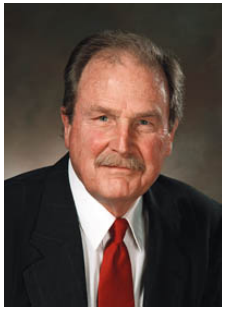
Observation. Lowest frequency items should be at the lowest level in tree of optimal prefix code.

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. 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. [Huffman, 1952] 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.



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 ω in S′ with fω=fy+fz
      T′ = Huffman(S′)
      T = add two children y and z to leaf ω from T′
      return T
   }
}
```

Q. What is the time complexity?

Optimal Prefix Codes: Huffman Encoding

```
Huffman(S) {
   if |S|=2 {
      return tree with root and 2 leaves                O(1)
   } else {
      let y and z be lowest-frequency letters in S      O(logn)
      S' = S                                            O(1)
      remove y and z from S'
      insert new letter ω in S' with fω=fy+fz           O(logn)
      T' = Huffman(S')                                  T(n-1)
      T = add two children y and z to leaf ω from T'    O(1)
      return T
   }
}
```

Q. What is the time complexity?

A. $T(n) = T(n-1) + O(n)$
so $O(n^2)$

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

A. Use priority queue for S: $T(n) = T(n-1) + O(\log n)$ so $O(n \log n)$

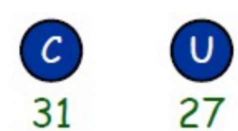
Huffman Code Construction

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

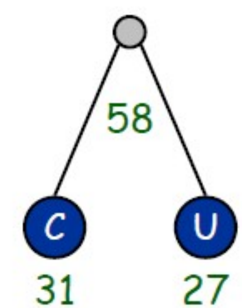
Huffman Code Construction

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



Huffman Code Construction

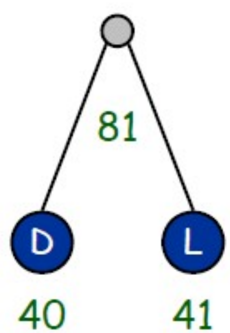
Char	Freq
E	125
T	93
A	80
O	76
I	73
N	71
S	65
R	61
	58
H	55
L	41
D	40
C	31
U	27

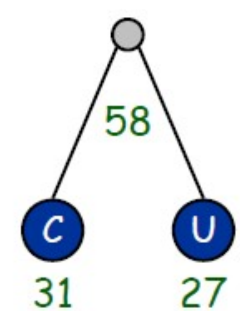


Huffman Code Construction

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

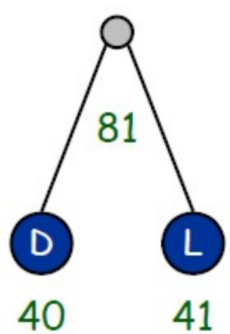


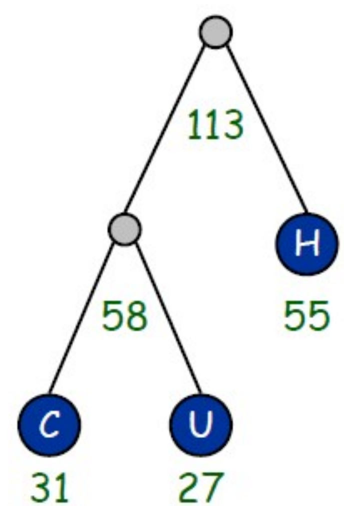


Huffman Code Construction

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

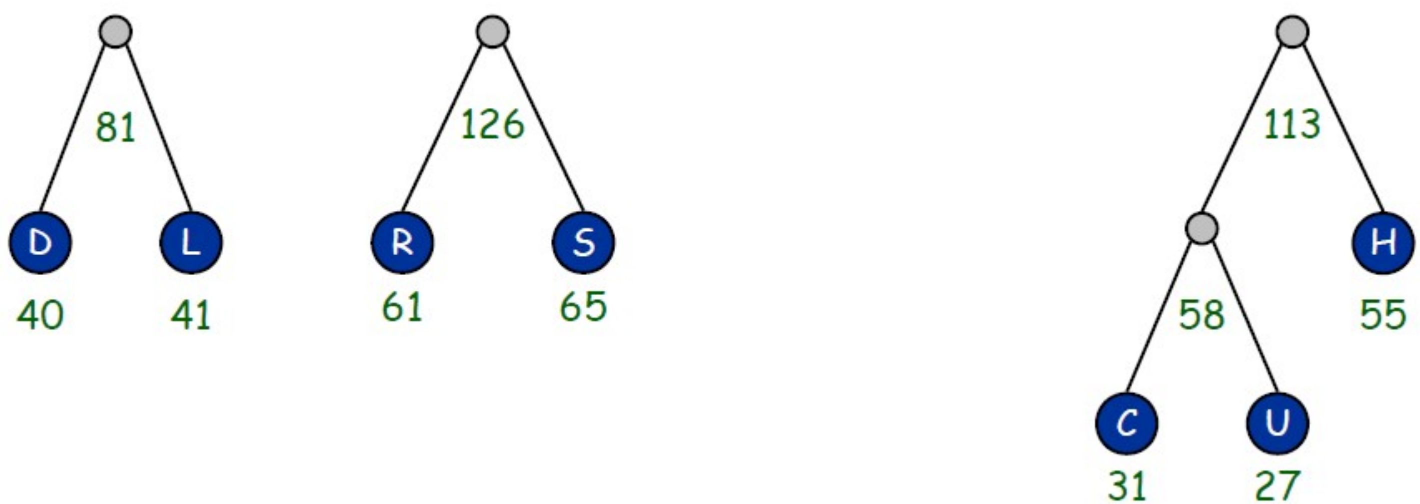
	58
H	55





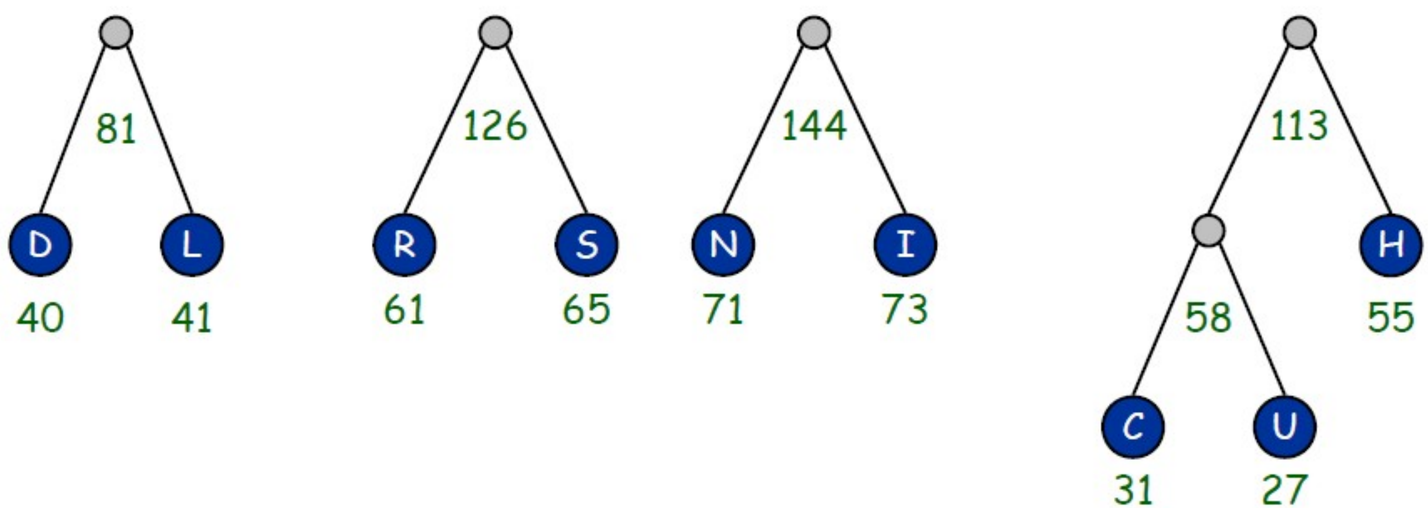
Huffman Code Construction

Char	Freq
	126
E	125
	113
T	93
	81
A	80
O	76
I	73
N	71
S	65
R	61



Huffman Code Construction

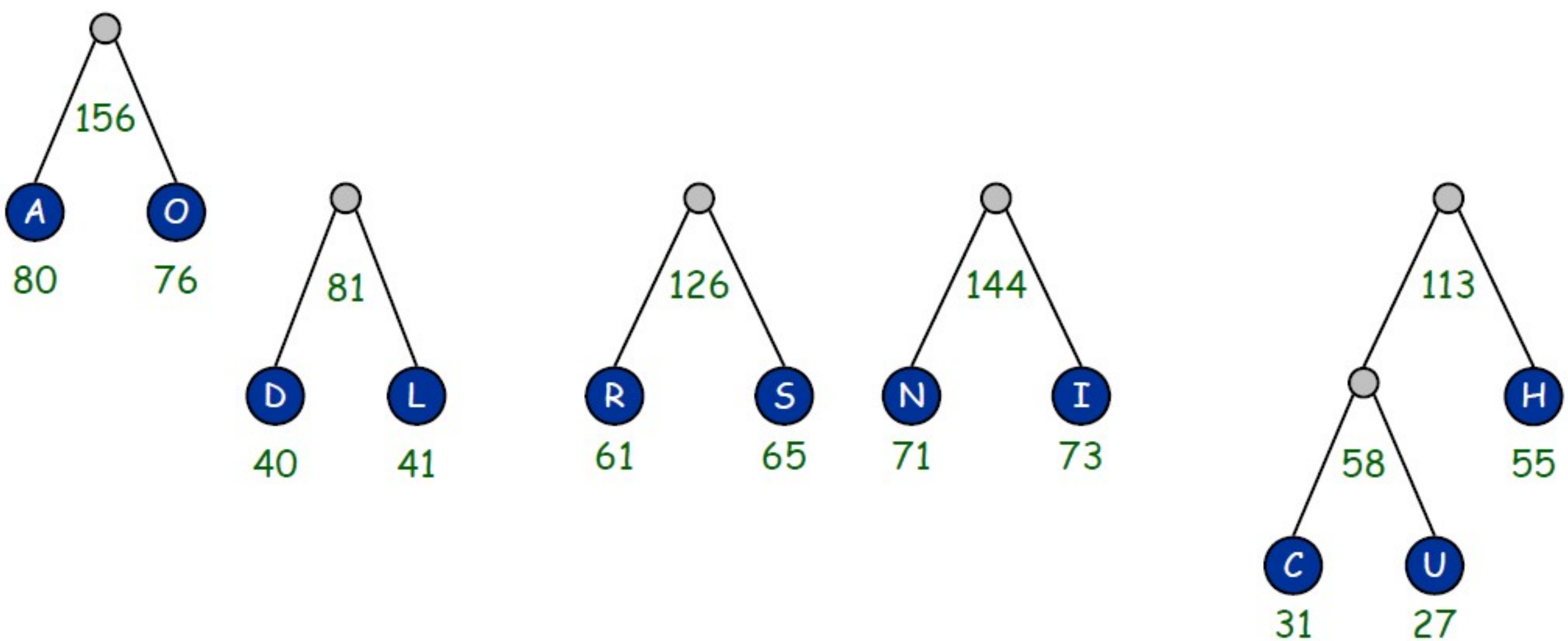
Char	Freq
	144
	126
E	125
	113
T	93
	81
A	80
O	76
I	73
N	71



Huffman Code Construction

Char	Freq
	156
	144
	126
E	125
	113
T	93
	81

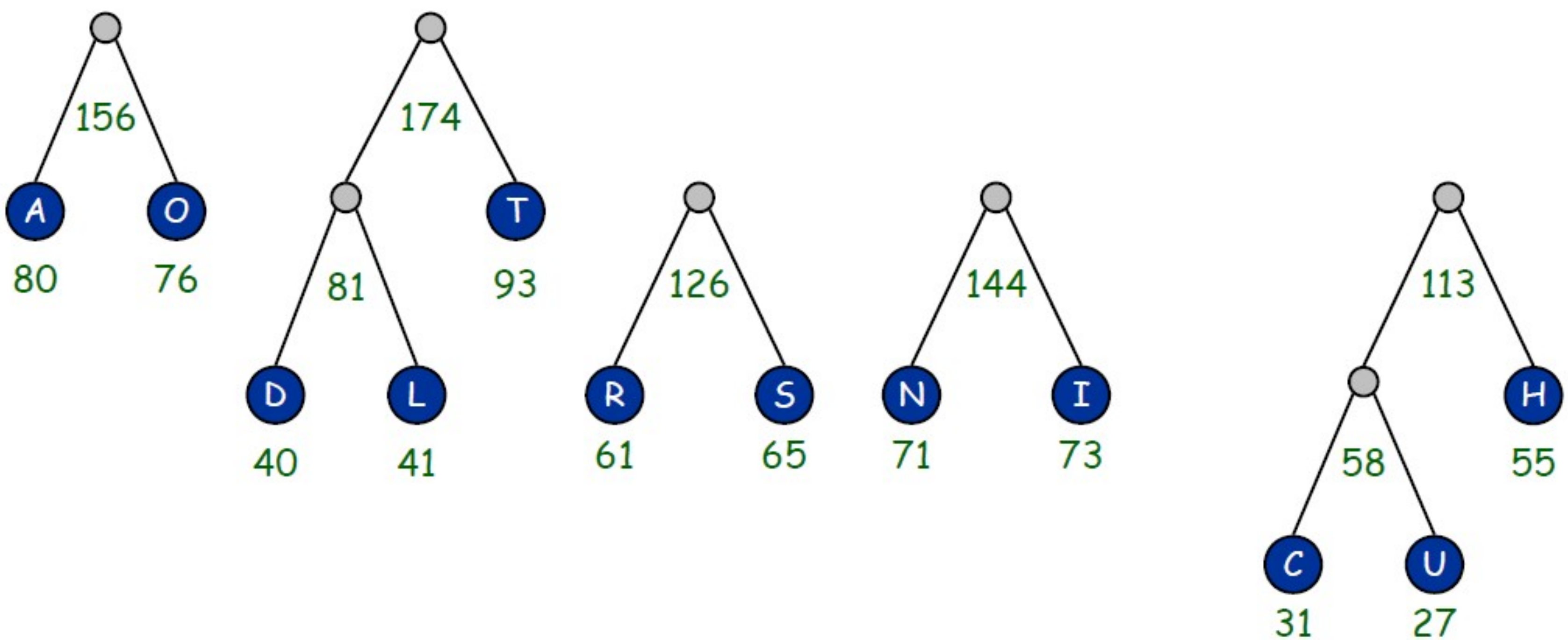
A	80
O	76



Huffman Code Construction

Char	Freq
	174
	156
	144
	126
E	125
	113

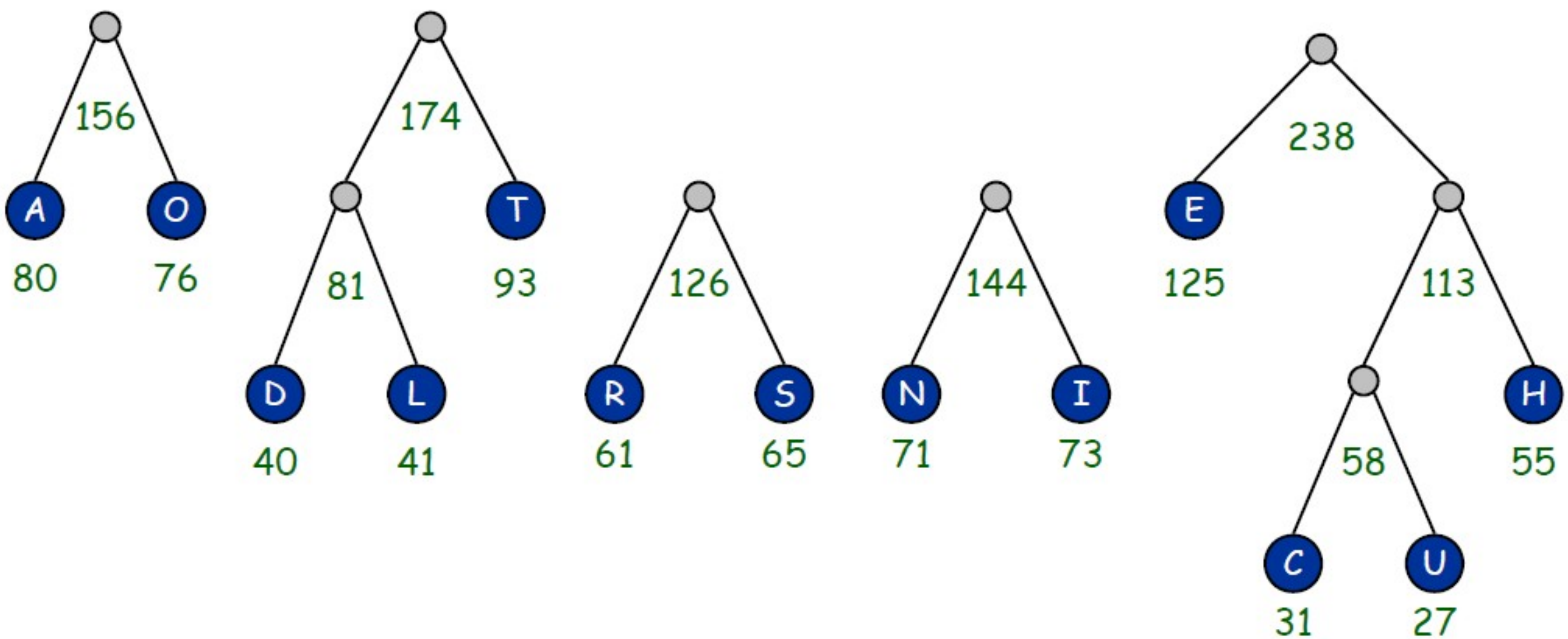
T	93
	81



Huffman Code Construction

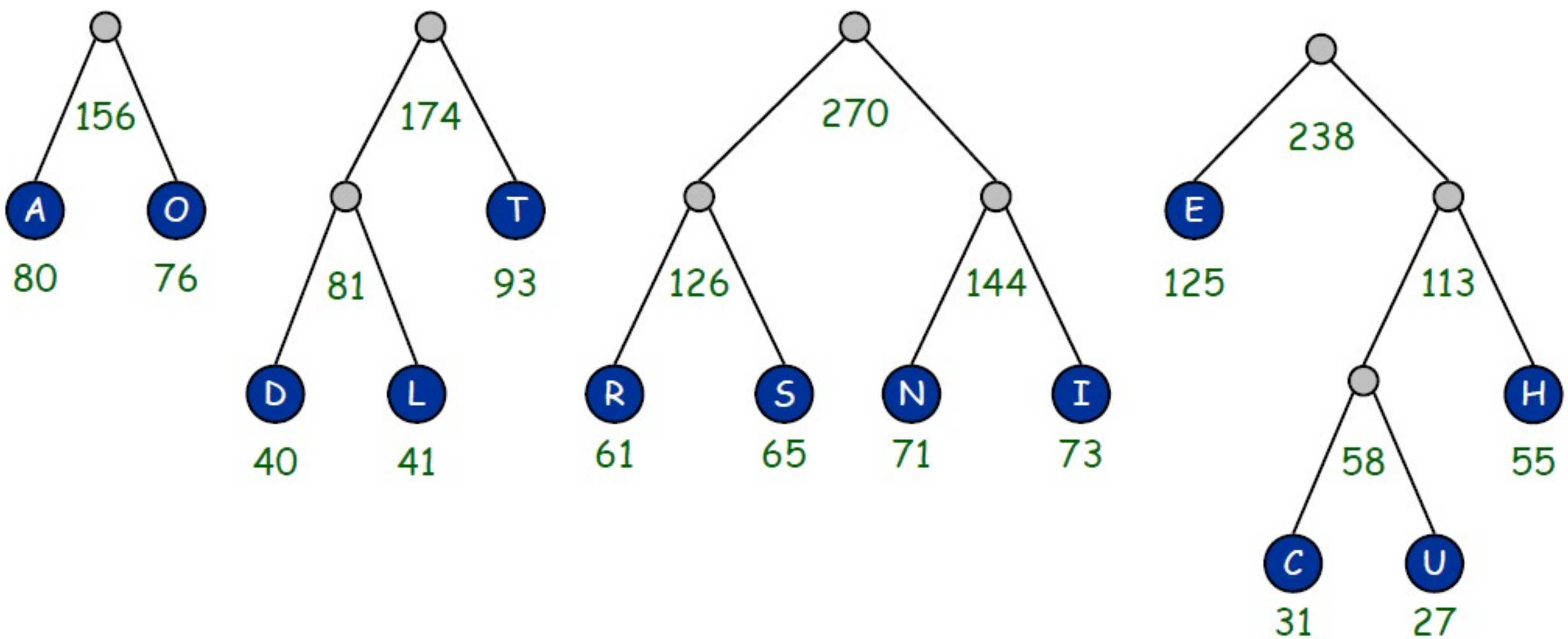
Char	Freq
	238
	174
	156
	144
	126

E	125
	113



Huffman Code Construction

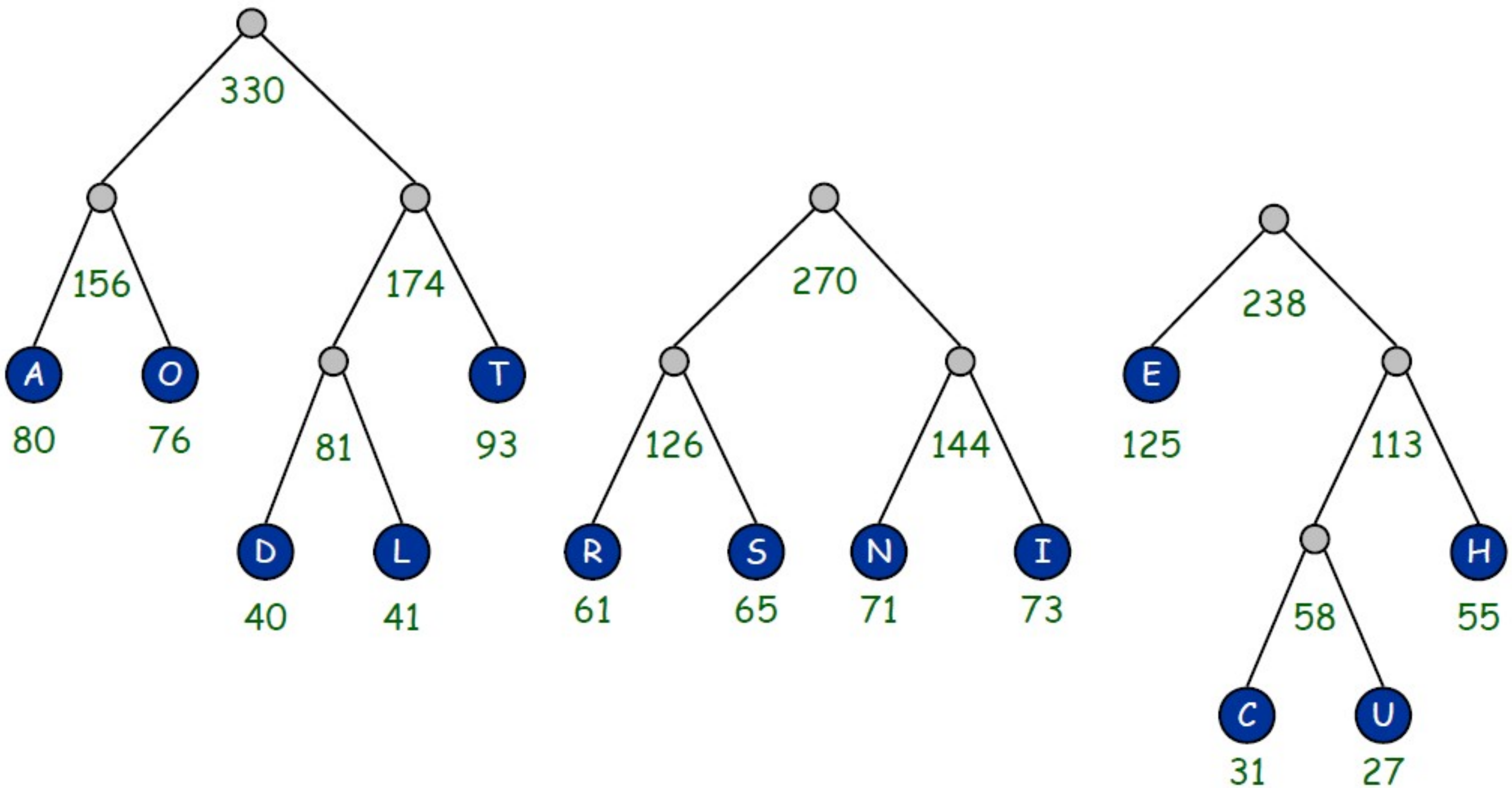
Char	Freq
	270
	238
	174
	156
	144
	126



Huffman Code Construction

Char	Freq
	330
	270
	238

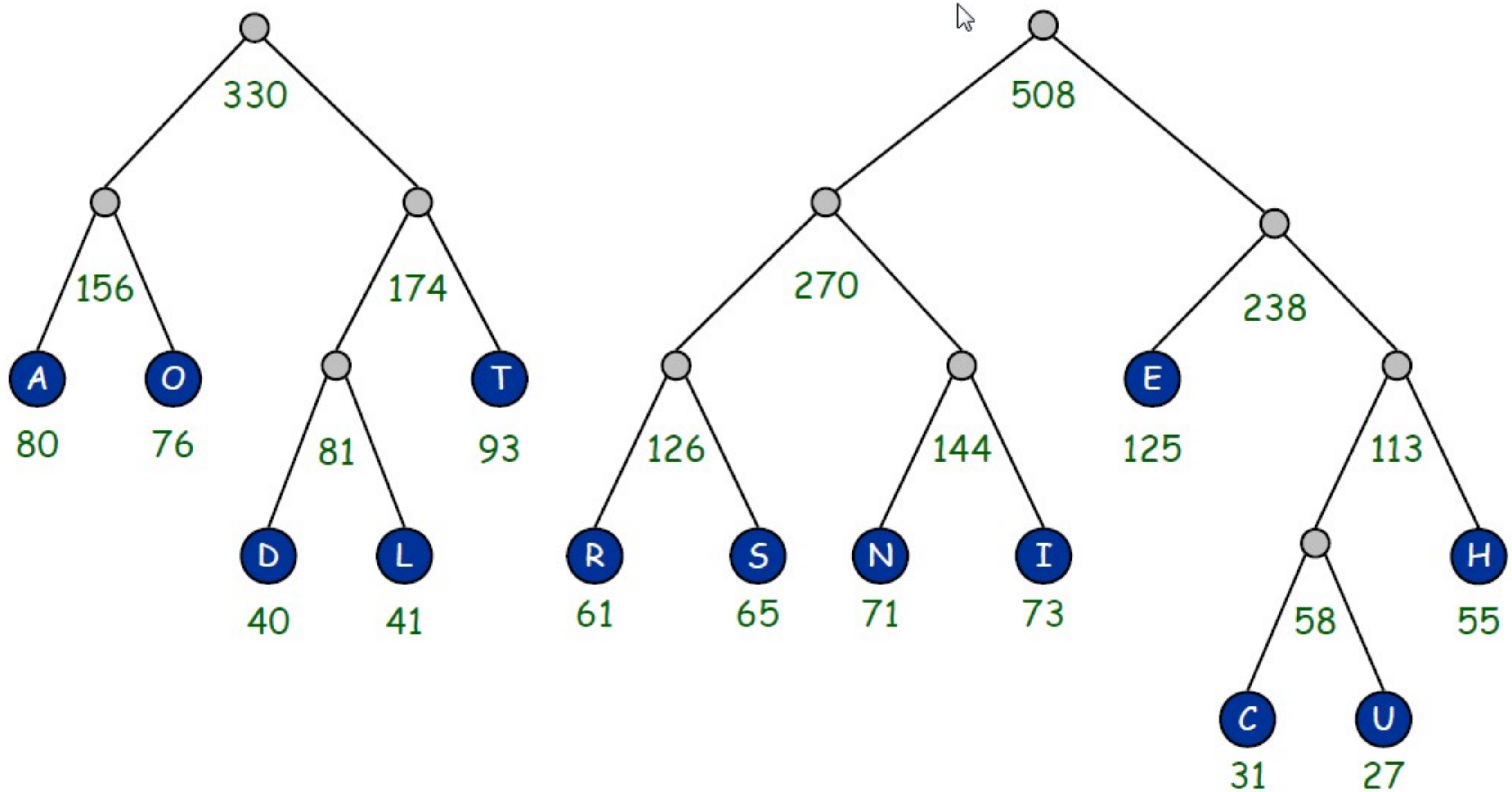
	174
	156



Huffman Code Construction

Char	Freq
	508
	330

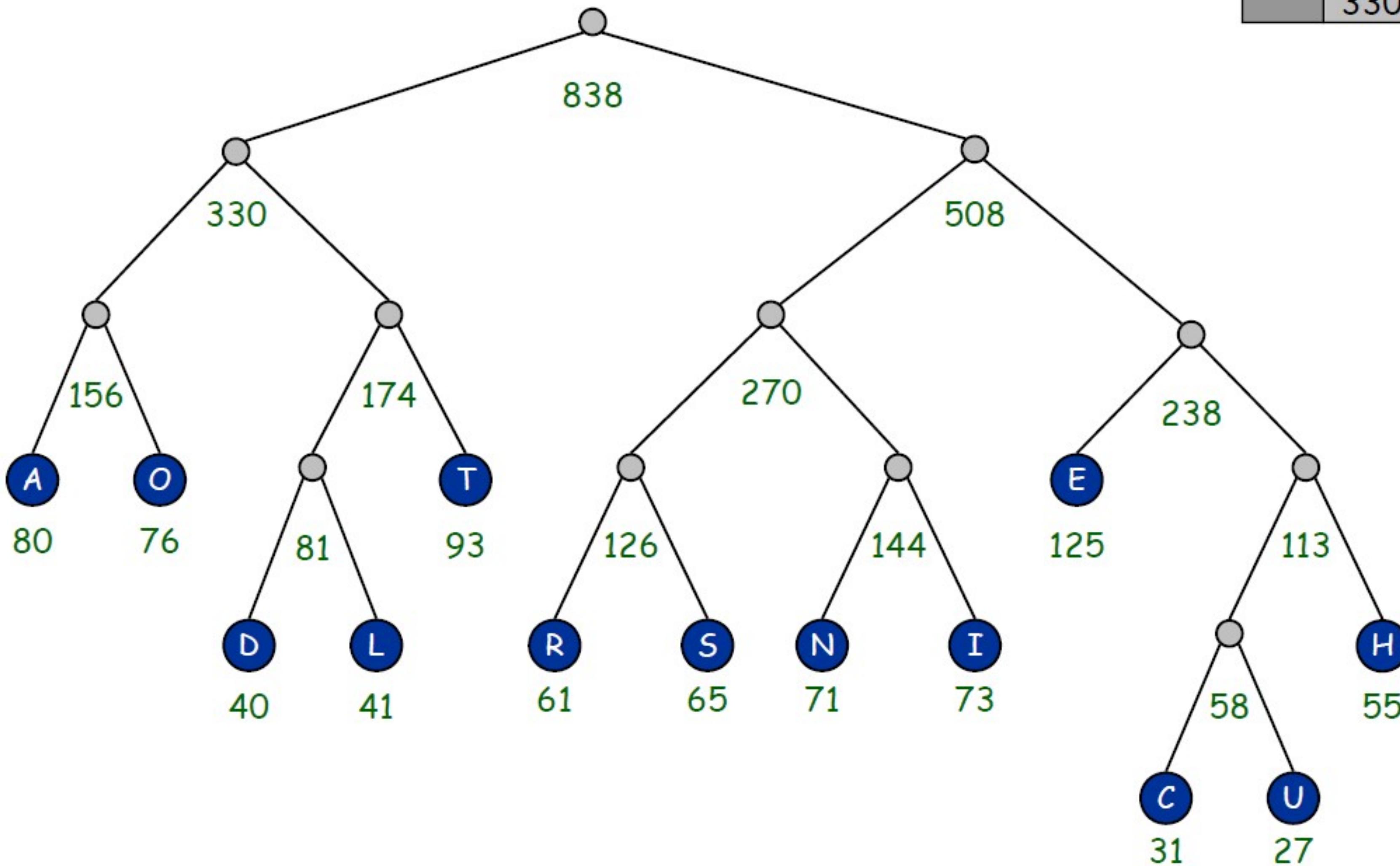
	270
	238



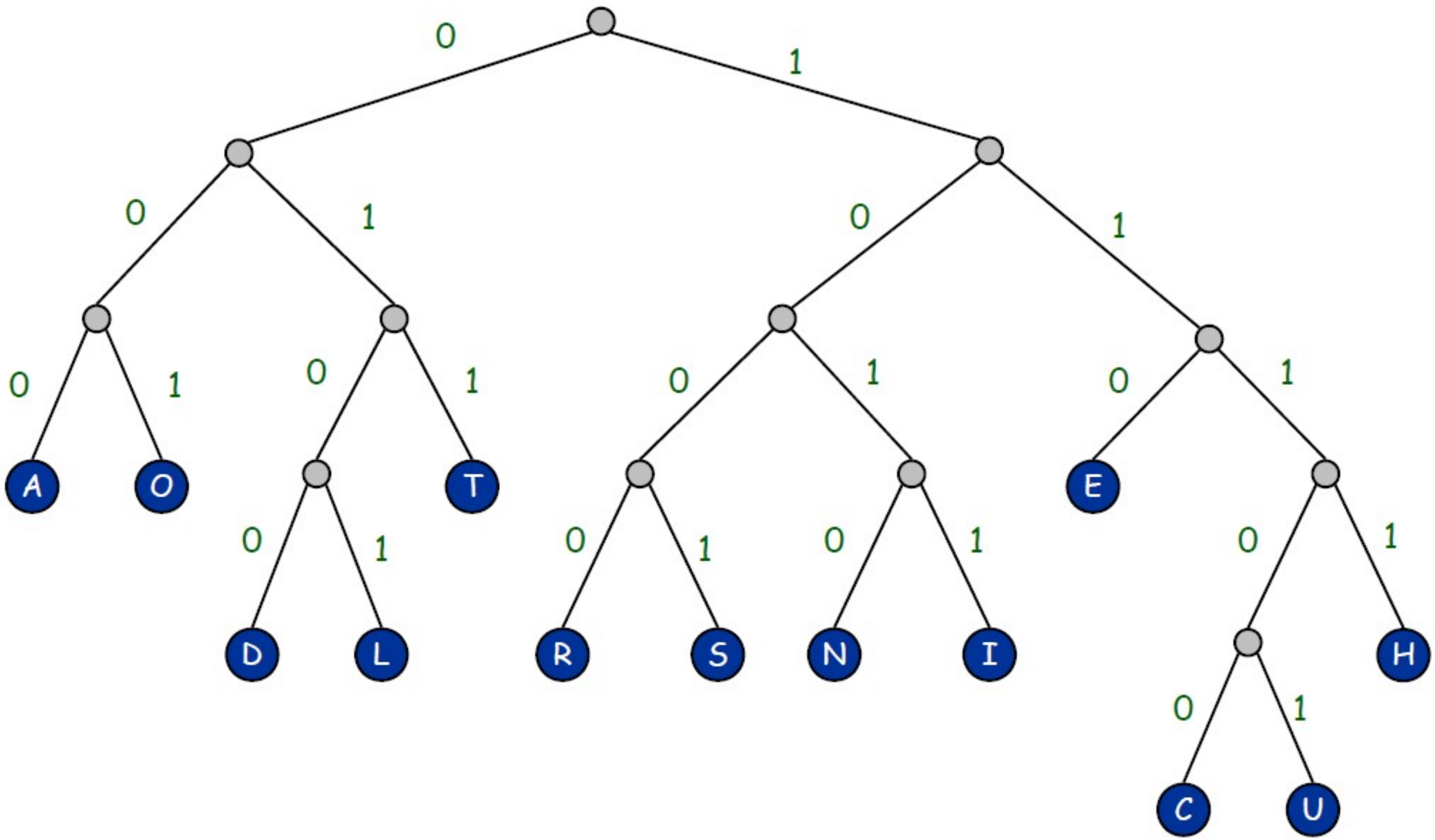
Huffman Code Construction

Char	Freq
	838

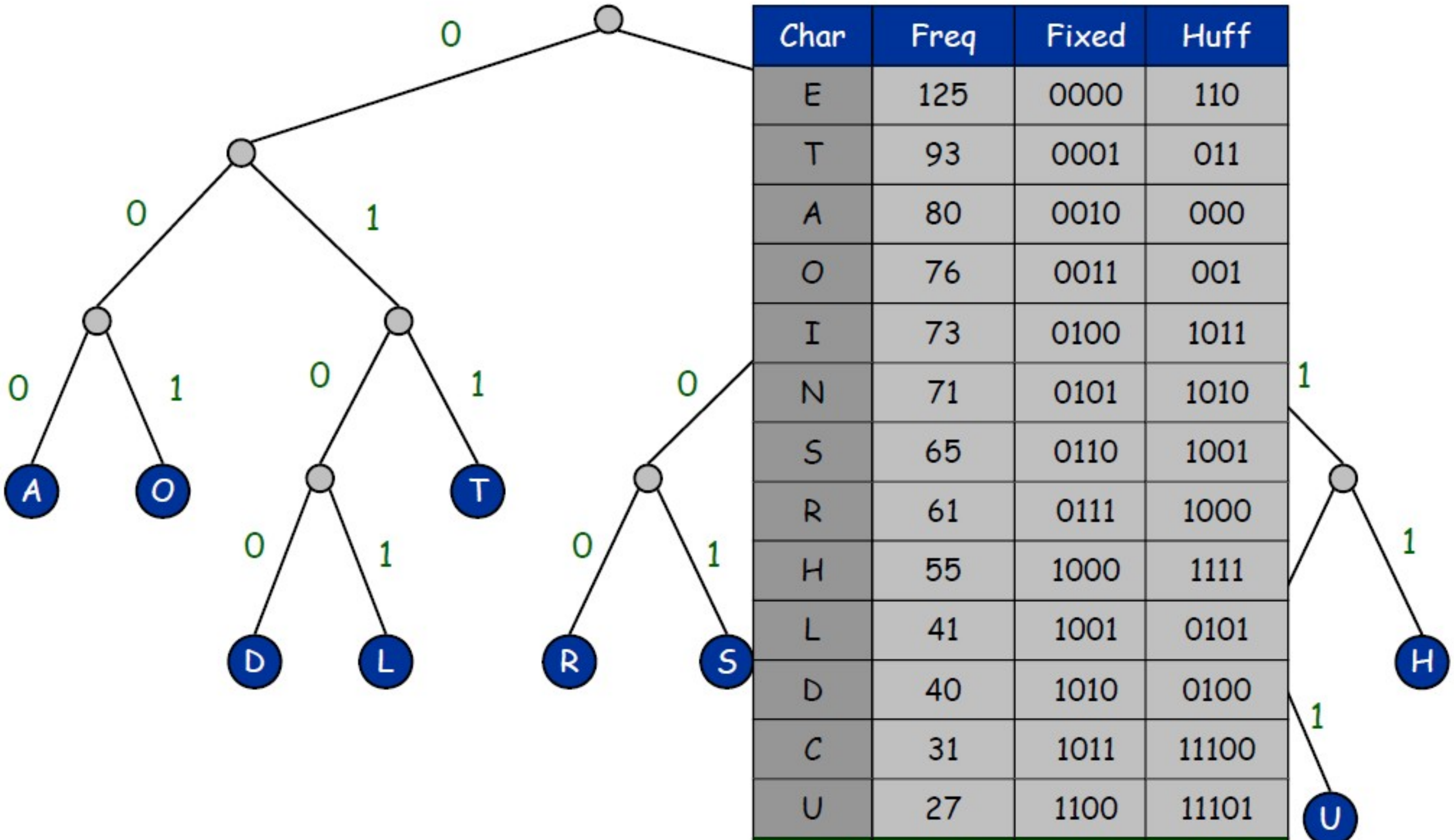
	508
	330



Huffman Code Construction



Huffman Code Construction



Char	Freq	Fixed	Huff
E	125	0000	110
T	93	0001	011
A	80	0010	000
O	76	0011	001
I	73	0100	1011
N	71	0101	1010
S	65	0110	1001
R	61	0111	1000
H	55	1000	1111
L	41	1001	0101
D	40	1010	0100
C	31	1011	11100
U	27	1100	11101
Total	838	4.00	3.62

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. Huffman code for S achieves the minimum ABL of any prefix code.
Pf. by induction, based on optimality of T' (y and z removed, ω added)
(see next page)

Claim. $ABL(T')=ABL(T)-f_{\omega}$
Pf.

Huffman Encoding: Greedy Analysis

Claim. Huffman code for S achieves the minimum ABL of any prefix code.
Pf. by induction, based on optimality of T' (y and z removed, ω added)
(see next page)

Claim. $ABL(T') = ABL(T) - f_\omega$
Pf.

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

Huffman Encoding: Greedy Analysis

Claim. Huffman code for S achieves the minimum ABL of any prefix code.
Pf. (by induction over n=|S|)

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 there is no shorter code than root and two leaves.

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

Step: (by contradiction)

Huffman Encoding: Greedy Analysis

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

Pf. (by induction)

Base: For n=2 there is no shorter code than root and 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)

- *Idea of proof:*
  - Suppose other tree Z of size n is better.
  - Delete lowest frequency items y and z from Z creating Z'
  - Z' cannot be better than T' by IH.

Huffman Encoding: Greedy Analysis

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

Pf. (by induction)

Base: For n=2 there is no shorter code than root and two leaves.

Hypothesis: Suppose Huffman tree T' for S' 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 tree Z such that ABL(Z) < ABL(T).
- Then there is also a tree Z for which leaves y and z exist that are siblings and have the lowest frequency (see observation).
- Let Z' be Z with y and z deleted, and their former parent labeled ω.
- Similar T' is derived from S' in our algorithm.
- We know that ABL(Z')=ABL(Z)-f_ω, as well as ABL(T')=ABL(T)-f_ω.
- But also ABL(Z) < ABL(T), so ABL(Z') < ABL(T').
- Contradiction with IH.