Algorithms

Lesson #3:
Greedy Algorithms (cont.)

(Based on slides by Prof. Dana Shapira)

Binary codes

Example

	a	b	c	d	e	f	
Frequency	45	13	12	16	9	5	
Fixed-length Codeword (C_1)	000	001	010	011	100	101	$L(C_1) = 300$
Variable-length codeword (C_2)	0	101	100	111	1101	1100	$L(C_2) = 224$

Which one is cheaper?

Prefix-free Codes

- Codes in which no codeword is also a prefix of some other codeword.
  - **Example:**

	a	b	c	d	e	f
Variable-length codeword	0	101	100	111	1101	1100

110001001101

- Easy to encode and decode using prefix-free codes.
- No Ambiguity !! **Uniquely Decipherable (UD)**

Prefix-free Codes

	a	b	c	d	e	f
Variable-length codeword	0	101	100	111	~~1101~~	1100
	0	101	100	111	110	1100

- 1100 = 110 0 = “f"
- or
- 1100 = 110 + 0 = “ea"

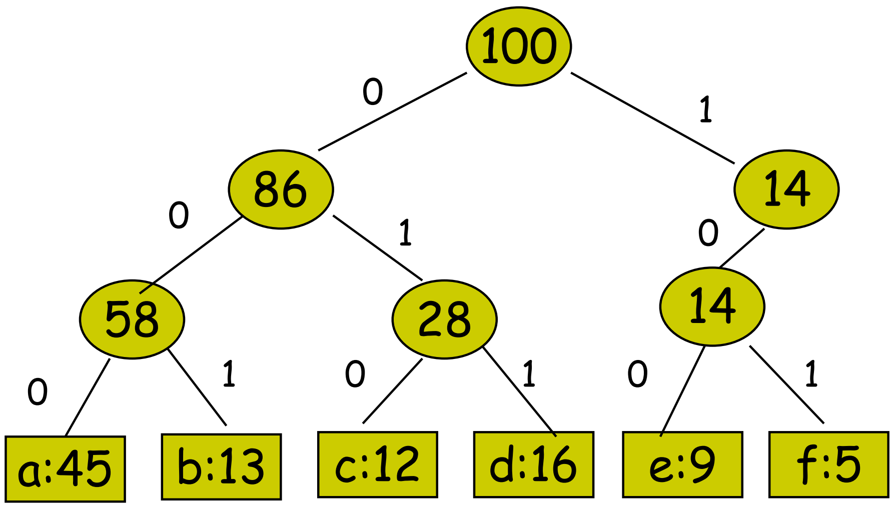
Prefix-free Codes

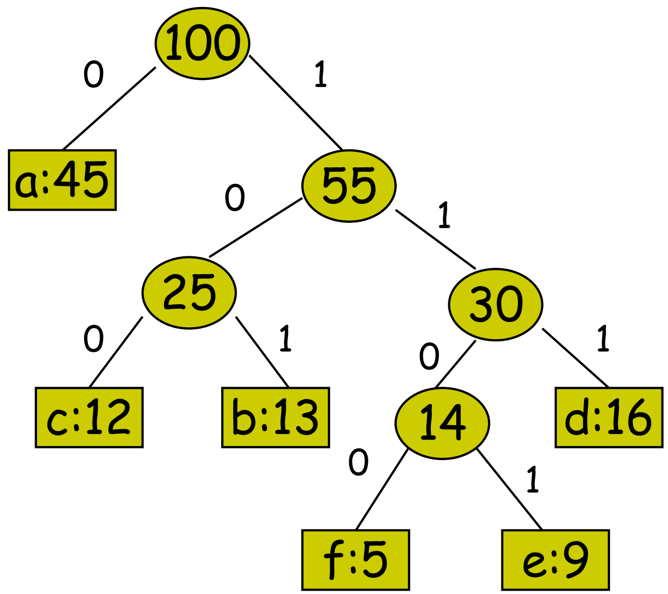
- Represented as a binary tree
  - each edge represents either 0 (left) or 1 (right)
  - each leaf corresponds to a codeword which is the sequence of 0s and 1s traversed from the root to reach it.
- Since no prefix is shared, all codewords are at the leaves, and decoding a string means following edges, according to the sequence of 0s and 1s in the string, until a leaf is reached.

Tree Representation

	a	b	c	d	e	f
Frequency	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101

	a	b	c	d	e	f
Frequency	45	13	12	16	9	5
Variable-length codeword	0	101	100	111	1101	1100





1

Building an optimal binary code

Problem: We are given a set of symbols with frequencies.

How can we build an optimal code for them?

v	w	x	y	z
7	15	6	5	6

Shannon-Fano algorithm

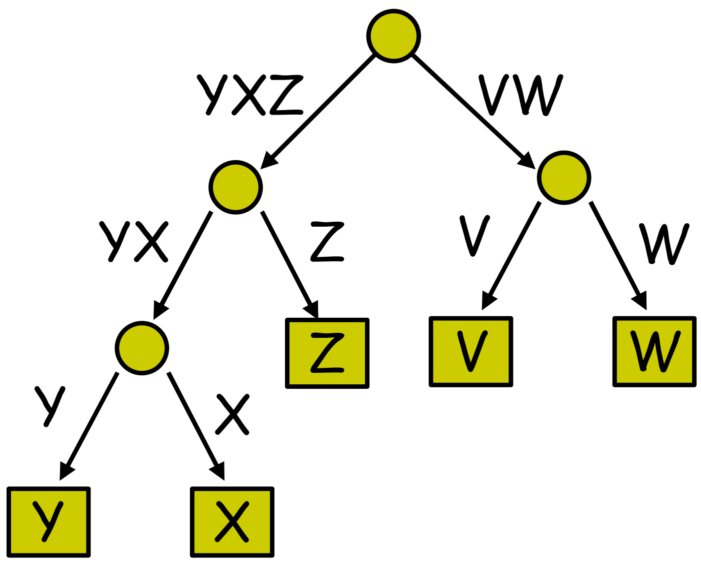
- Sort the symbols by frequency
- Always split as close to the middle as possible (w.r.t. frequency)

V	W	X	Y	Z
7	15	6	5	6

Y	X	Z	V	W
5	6	6	7	15
	17			22

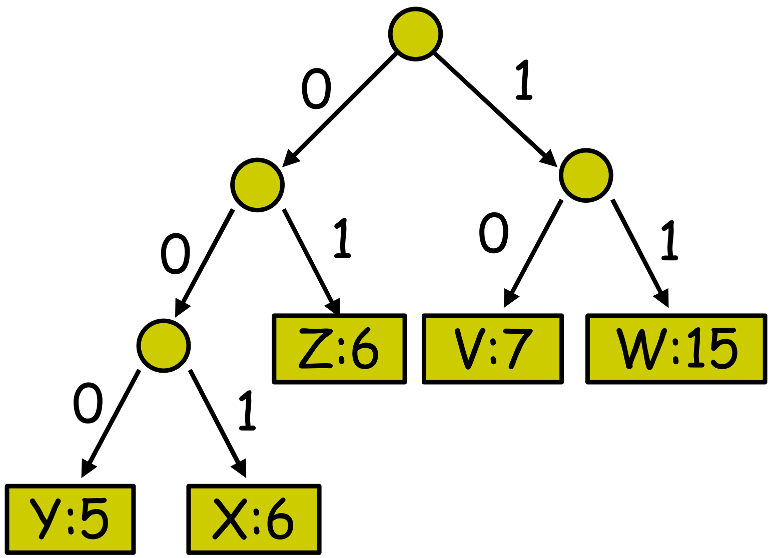
Y	X	Z
5	6	6
11		6

Total: 39



V	W	X	Y	Z
7	15	6	5	6

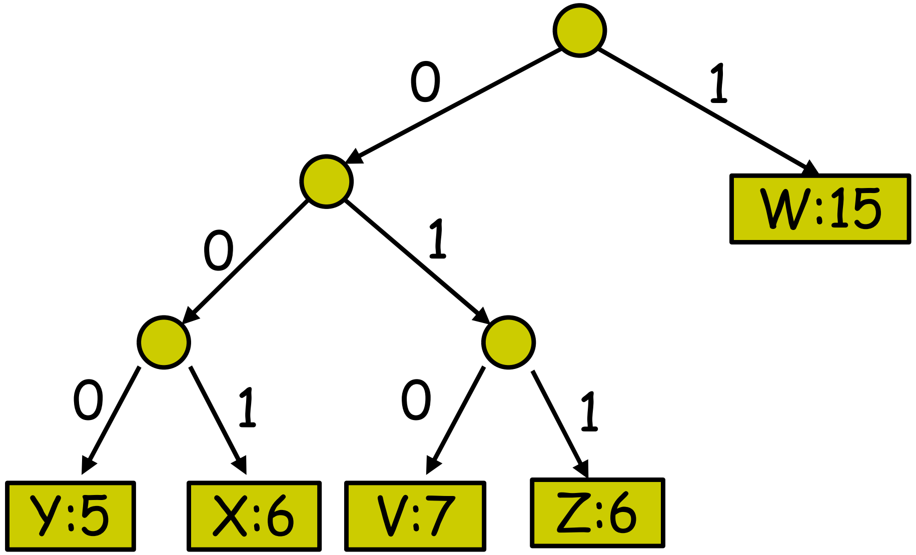
The Shannon-Fano code:



L = 89

Is this optimal?

A better code:

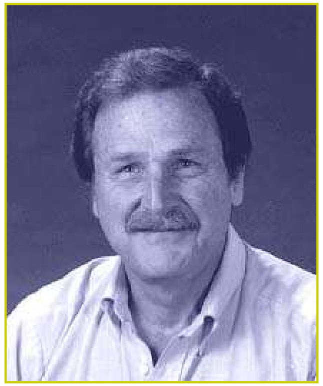


L = 87

Huffman Algorithm

- Greedy
  - The <u>two smallest nodes</u> are chosen at each step, and this local decision results in a globally optimal encoding tree.
- bottom-up manner
  - Starts with a set of $|\Sigma|$ leaves and performs a sequence of $|\Sigma| - 1$ "merging" operations to create the final tree.

Professor David A. Huffman
(August 9, 1925 - October 7, 1999)

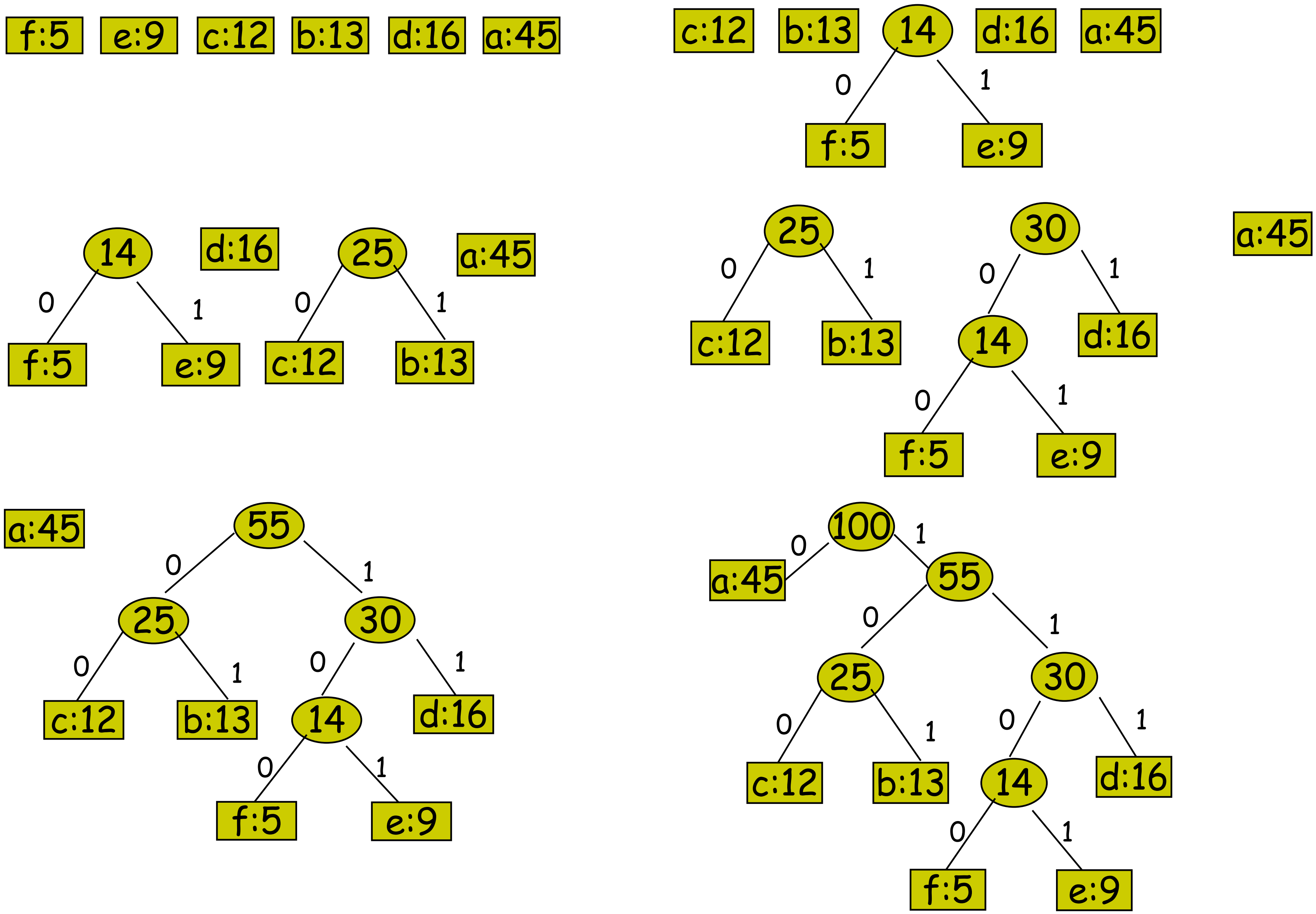


Huffman Algorithm

```
HUFFMAN(Σ)
n ← |Σ|
Q ← Σ
for i ← 1 to n - 1
      do  ALLOCATE-NODE(z)
       left[z] ← x ← EXTRACT-MIN(Q)
       right[z] ← y ← EXTRACT-MIN(Q)
       w[z] ← w[x] + w[y]
       INSERT(Q, z)
return EXTRACT-MIN(Q)
```

What is the running time?

Huffman Algorithm



Optimality of Huffman Codes

Theorem:

Given weights $w_1,\dots,w_n$. Huffman Algorithm assigns code lengths $l_1,\dots,l_n$ such that $L = \sum w_i l_i$ is minimal.

Lemma 1:

An optimal code for a file is always represented by a ***full* binary tree**, in which every non-leaf node has two children.

Lemma 2:

In an optimal tree the two lowest weights w_{n-1} and w_n are in the lowest level.

Optimality of Huffman Codes

Lemma 3:

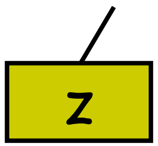
In an optimal tree the two lowest weights w_{n-1} and w_n can be assumed to be brothers.

Optimality of Huffman Codes

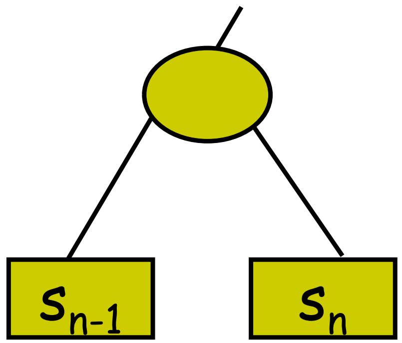
Proof that Huffman is optimal:

By induction on n.

Given $\Sigma = (s_1,\dots,s_n)$ with multiplicities $(w_1,\dots,w_n)$, Huffman replaces symbols s_{n-1}, s_n by a new symbol z with multiplicity $w_{n-1}+w_n$, then constructs a tree T' for $\Sigma'=(s_1,\dots,s_{n-2},z)$, then obtains T by replacing



by



By induction assumption, T' is optimal

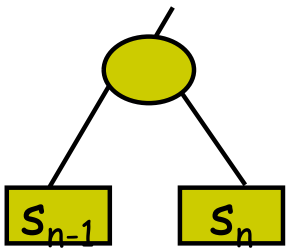
What is the relation between L(T) and L(T')?

$$L(T) = L(T') + w_{n-1} + w_n \qquad (1)$$

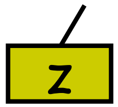
Suppose for a contradiction that there exists a tree T^* for Σ with $L(T^*) < L(T)$

W.l.o.g. s_{n-1}, s_n are neighbors in T^*

Can replace



in T^* by



By (1), we got a tree for Σ' better than T'. **Contradiction**