

# Introduction to Multimedia Technology (MM301)

*By Dr. Heba Hamdy and Dr. Ahmed Anter  
Assistant Professor, Multimedia Dep.*

## Variable-Length Coding (VLC)

Entropy Based  
Variable Length  
Exact(Lossless)

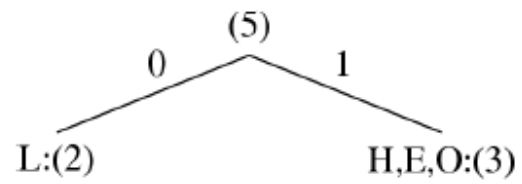
- **Shannon-Fano Algorithm** — a top-down approach
- 1. Sort the symbols according to the frequency count of their occurrences.
- 2. Recursively divide the symbols into two parts, each with approximately the same number of counts, until all parts contain only one symbol.
- **An Example: coding of “HELLO”**

Symbol	H	E	L	O
Count	1	1	2	1

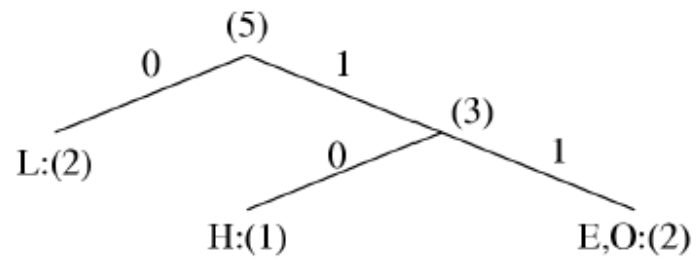
- Frequency count of the symbols in “HELLO”.



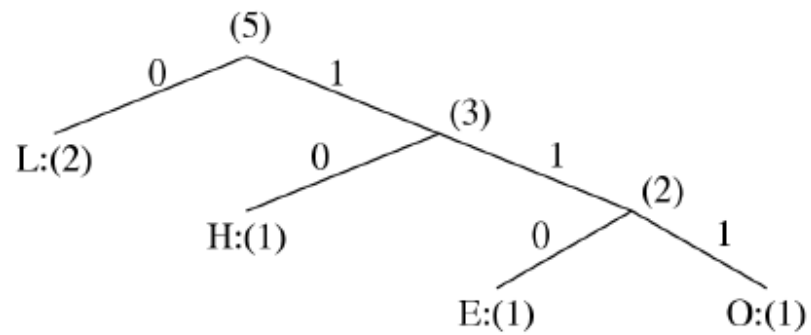
## Shannon-Fano Algorithm



(a)



(b)



(c)

Coding Tree for HELLO by Shannon-Fano.

## Result of Performing Shannon-Fano on HELLO

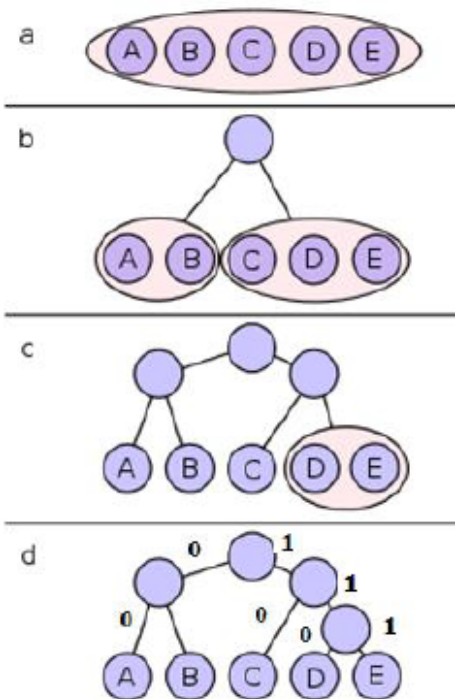
Symbol	Count	$\text{Log}_2 \frac{1}{p_i}$	Code	# of bits used
L	2	1.32	0	1
H	1	2.32	10	2
E	1	2.32	110	3
O	1	2.32	111	3
TOTAL # of bits:				9



## Shannon-Fano Algorithm Example

Symbol	A	B	C	D	E
Count	15	7	6	6	5
Probabilities	0.38461538	0.17948718	0.15384615	0.15384615	0.12820513

Symbol	A	B	C	D	E
Code	00	01	10	110	111



## Exercise

If you have the following data:

A	A	C	B	B	D	A	B	B	B	A	A	A	A
A	A	C	B	B	D	A	B	B	B	A	A	A	A
A	A	C	B	B	D	A	D	D	D	A	A	A	A
A	A	C	B	B	D	A	E	C	C	C	C	C	C
A	A	C	C	B	B	A	D	C	C	C	C	C	C
A	C	C	C	B	B	A	B	B	B	C	C	C	A
A	C	C	C	B	B	A	B	B	B	C	C	C	A
A	C	C	C	B	B	A	B	B	B	C	C	C	A

How you can find the coding using Shannon-Fano coding



Symbols = 112

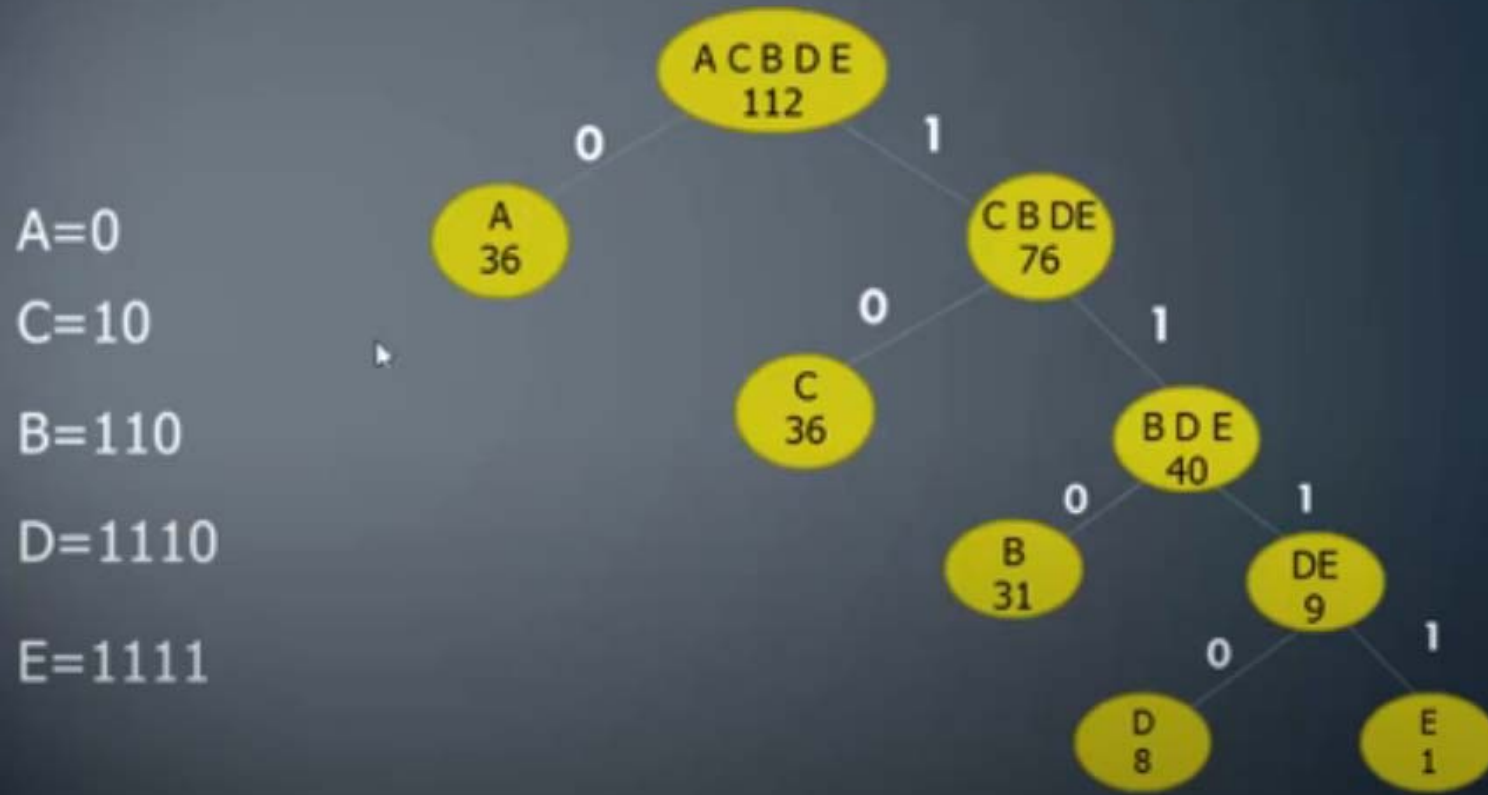
A = 36

B=31

C=36

D=8

E=1



A=0

C=10

B=110

D=1110

E=1111

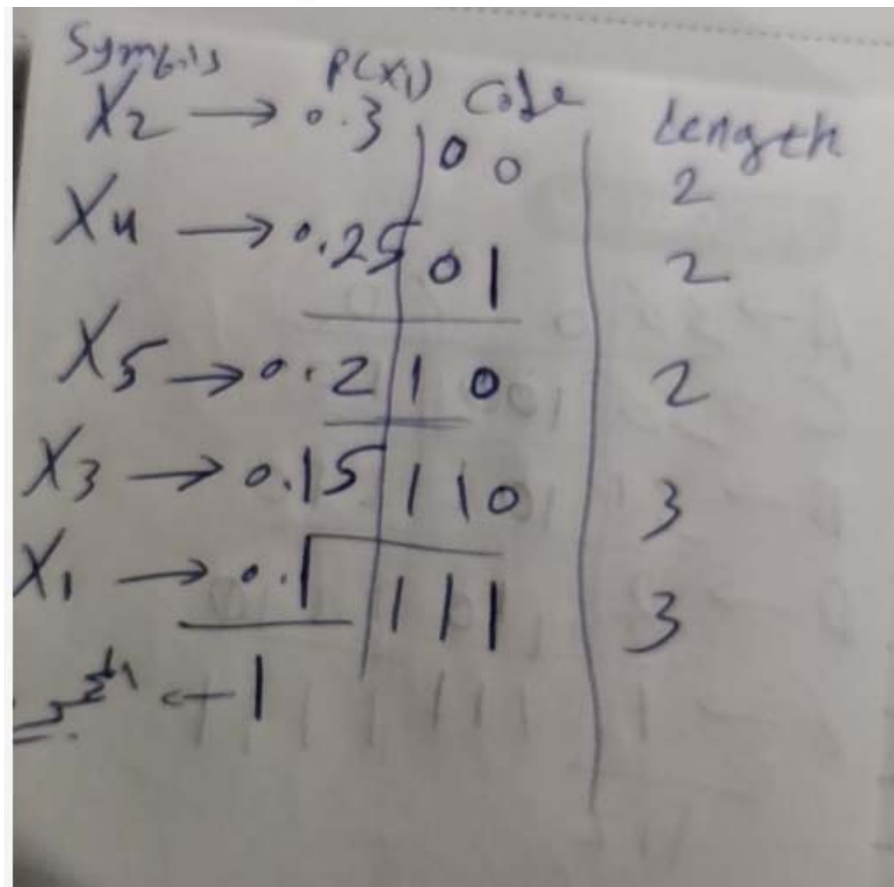
## Same Solution

Symbols	Code	Code length
A → 30	0	1
C → 30	10	2
B → 31	110	3
D → 8	1110	4
E → 1	1111	4
Total.	112	



## Exercise

Q1 : A source produce 5 independent symbols (  $x_1, x_2, x_3, x_4, x_5$  ) with its corresponding probabilities 0.1, 0.3, 0.15, 0.25, 0.2 . design a binary code for the above source symbol using Shannon – fanon method.



The image shows a handwritten solution for the Shannon-Fano coding exercise. It lists the symbols, their probabilities, the resulting binary codes, and their lengths. The symbols are ordered by decreasing probability:  $x_2$  (0.3),  $x_4$  (0.25),  $x_5$  (0.2),  $x_3$  (0.15), and  $x_1$  (0.1). The codes are assigned by splitting the list at each step:  $x_2$  and  $x_4$  get code '00',  $x_5$  gets '01',  $x_3$  gets '110', and  $x_1$  gets '111'. The total number of symbols is noted as 5 at the bottom.

Symbols	$P(x_i)$	code	length
$x_2 \rightarrow$	0.3	00	2
$x_4 \rightarrow$	0.25	01	2
$x_5 \rightarrow$	0.2	10	2
$x_3 \rightarrow$	0.15	110	3
$x_1 \rightarrow$	0.1	111	3
المجموع =	5		

## Exercise

Symbols	a	b	c	d	e	f	g	h
Count	100	400	1000	300	600	700	400	500

Handwritten solution for Huffman coding:

Symbols	Count	Code	length
c	1000	00	2
f	700	01	2
e	600	100	3
h	500	101	3
b	400	110	3
g	400	1110	4
d	300	11110	5
a	100	11111	5
<b>Total</b>	<b>4000</b>		

$$CR = \frac{4000 \times 8}{(1000 \times 2) + (700 \times 2) + (600 \times 3) + (500 \times 3) + (400 \times 3) + (400 \times 4) + (300 \times 5) + (100 \times 5)}$$

$$= 1.7$$



## Homework

The following are gray levels counts taken from an Image Matrix.  
Write code and length for each symbol using Shannon and  
calculate CR

Level	10	70	120	170	140	200	220	230
Count	150	150	400	1000	400	300	700	300

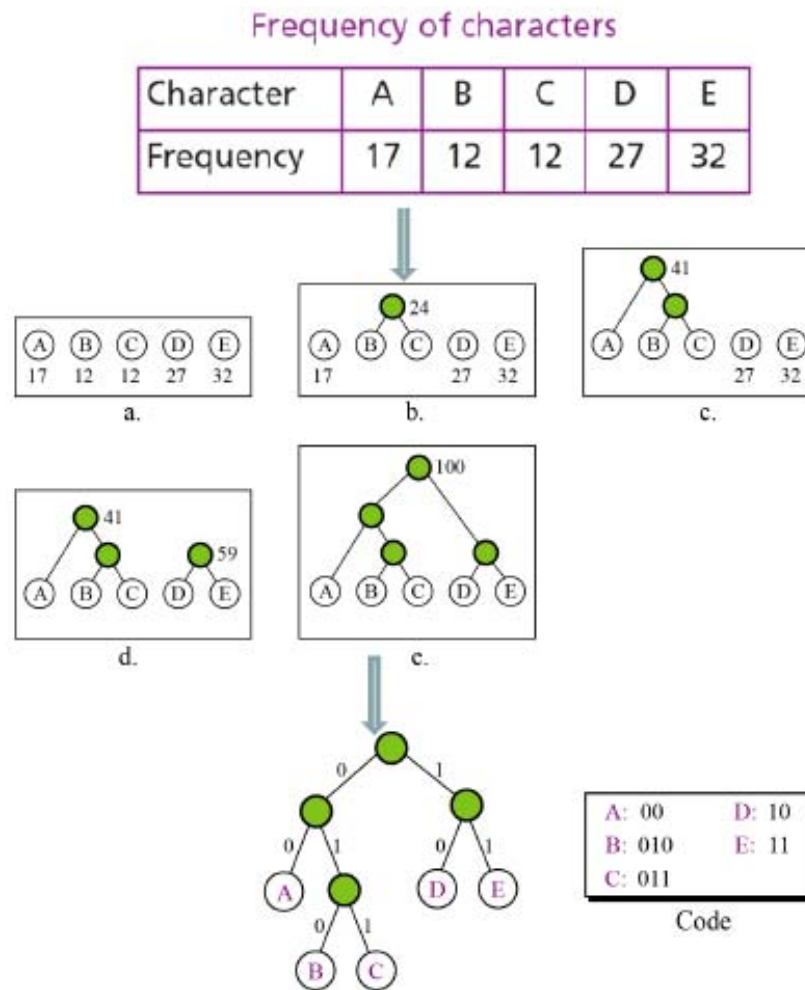
## Huffman Coding

- Assign fewer bits to symbols that occur more frequently and more bits to symbols appear less often.
- There's no unique Huffman code and every Huffman code has the same average code length.
- Algorithm:
  - ① Make a leaf node for each code symbol
    - ◆ Add the generation probability of each symbol to the leaf node
  - ② Take the two leaf nodes with the smallest probability and connect them into a new node
    - ◆ Add 1 or 0 to each of the two branches
    - ◆ The probability of the new node is the sum of the probabilities of the two connecting nodes
  - ③ If there is only one node left, the code construction is completed. If not, go back to (2)



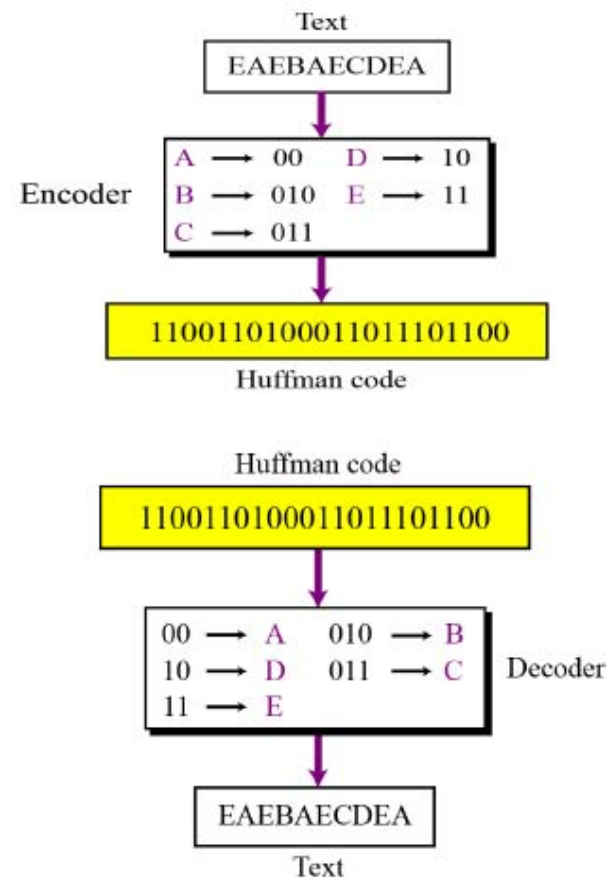
# Huffman Coding

- Example

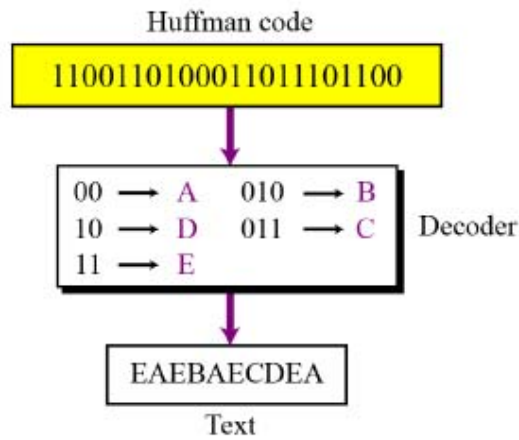


# Huffman Coding

- Encoding



- Decoding





## Adaptive Huffman Coding

- **Adaptive Huffman Coding:** statistics are gathered and updated dynamically as the data stream arrives.

ENCODER

-----

```
Initial_code();
```

```
while not EOF
```

```
{
```

```
    get(c);
```

```
    encode(c);
```

```
    update_tree(c);
```

```
}
```

DECODER

-----

```
Initial_code();
```

```
while not EOF
```

```
{
```

```
    decode(c);
```

```
    output(c);
```

```
    update_tree(c);
```

```
}
```

## Adaptive Huffman Coding

- *Initial\_code* assigns symbols with some initially agreed upon codes, without any prior knowledge of the frequency counts.
- *update\_tree* constructs an Adaptive Huffman tree.

It basically does two things:

- (a) increments the frequency counts for the symbols (including any new ones).
- (b) updates the configuration of the tree.





## Notes on Adaptive Huffman Tree Updating

- Nodes are numbered in order from left to right, bottom to top. The numbers in parentheses indicates the count.
- The tree must always maintain its *sibling* property, i.e., all nodes (internal and leaf) are arranged in the order of increasing counts.

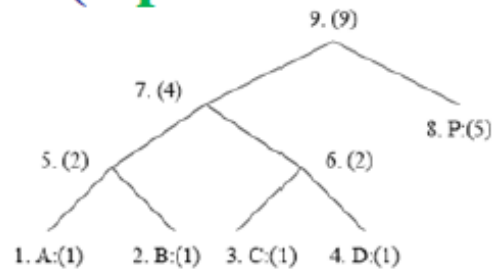
If the sibling property is about to be violated, a *swap* procedure is invoked to update the tree by rearranging the nodes.

- When a swap is necessary, the farthest node with count  $N$  is swapped with the node whose count has just been increased to  $N+1$ .

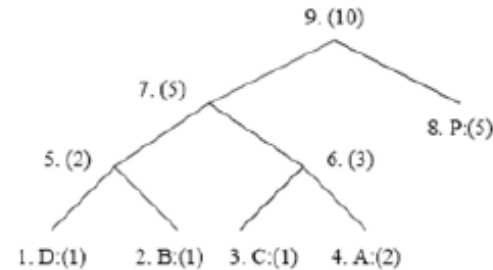


# Example1: Adaptive Huffman Coding

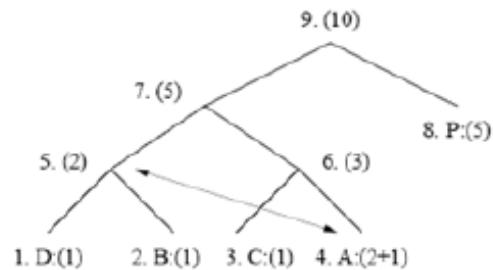
## (Update the Huffman tree when receive AAA)



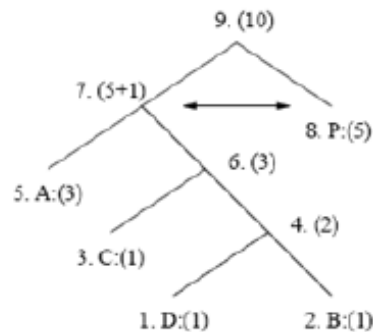
(a) A Huffman tree



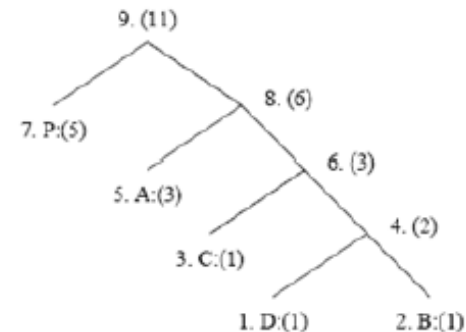
(b) Receiving 2nd 'A' triggered a swap



(c-1) A swap is needed after receiving 3rd 'A'



(c-2) Another swap is needed



(c-3) The Huffman tree after receiving 3rd 'A'

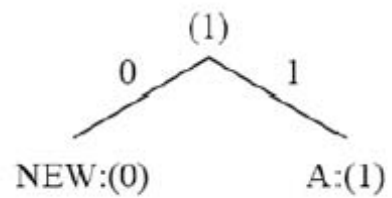


## Another Example: Adaptive Huffman Coding (Create the Huffman tree)

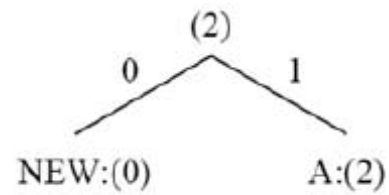
- An additional rule: if any character/symbol is to be sent the first time, it must be preceded by a special symbol, NEW. The initial code for NEW is 0. The *count* for NEW is always kept as 0 (the count is never increased); hence it is always denoted as NEW:(0).



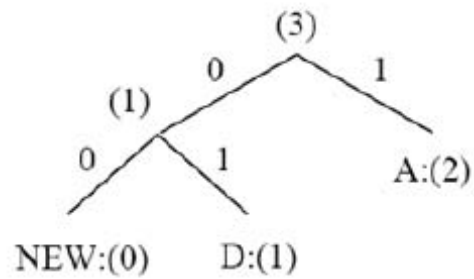
## Adaptive Huffman tree for AADCCDD.



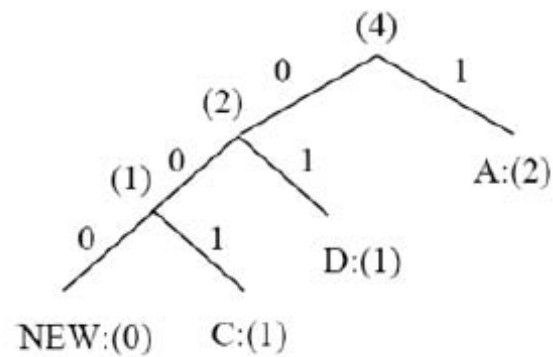
"A"



"AA"



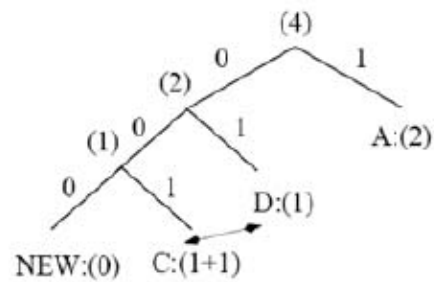
"AAD"



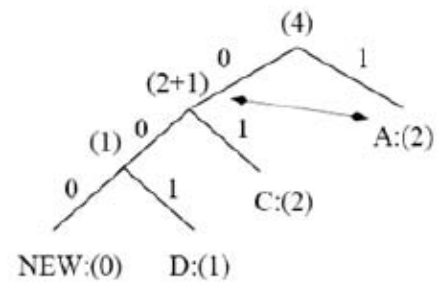
"AADC"



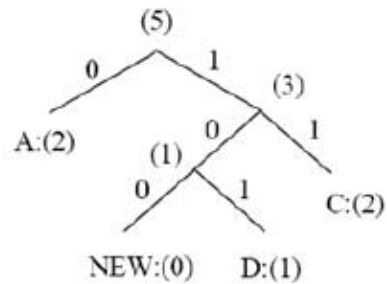
## Adaptive Huffman tree for AADCCDD



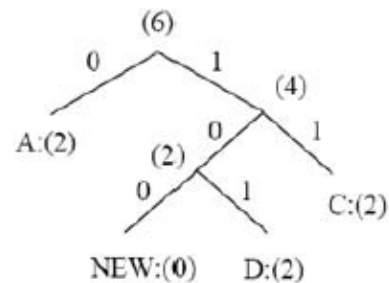
"AADCC" Step 1



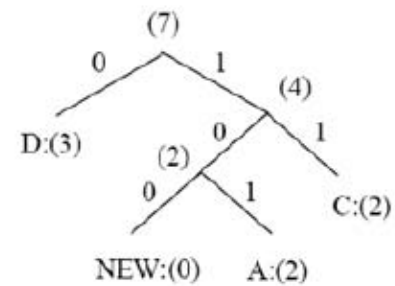
"AADCC" Step 2



"AADCC" Step 3



"AADCCD"



"AADCCDD"

**Then Calculate Efficiency, Compression Ratio, Compression Factor, and Entropy.**

**A    A   D C   C D D**

**101 101 0 11 11 0 0**

**Frequency ( $F$ )**

**Bit Length( $L$ );**

$L(A)=6/2$  ;  $L(D)=3/3$ ;  $L(C)=4/2$ ;

**Efficiency ( $EF$ )** =  $\sum L_i * F_i * \text{Log}_2( L_i * F_i)$

$EF = 3*2*\log_2(3*2) + 1*3 * \log_2(1*3) + 2*2* \log_2(2*2)=28.26\%$

**Compression Facto( $CF$ )** =  $1-(13/(7*8))= 77\%$

**Compression Ratio** =  $56/13=4.3$

**Entropy** =  $\sum P_i * \log(1/P_i)$  ← **Calculate Entropy**





# Thanks