

# Compression Algorithms

CE2103 - Algorithms and Data Structures II



# Disclaimer / Descargo de Responsabilidad

---

Esta presentación corresponde a una guía usada por el profesor durante las clases. La misma ha sido modificada para ser utilizado en el modelo de cursos asistidos por tecnología. No es una versión final, por lo que la misma podría requerir todavía hacer algunos ajustes. Para aspectos de evaluación esta presentación es solo una guía, por lo que el estudiante debe profundizar con el material de lectura asignado y lo discutido en clases para aspectos de evaluación.

This presentation corresponds to a guide material used by the professor during classes. It has been modified to be used in the model of technology-assisted courses. It is not a final version, so it may still require some adjustments. For evaluation aspects, this presentation is only a guide, so the student should delve with the assigned reading material and what has been discussed in class.

# Introduction

---

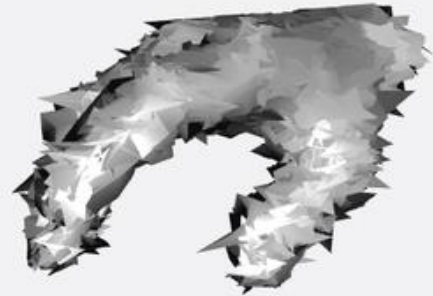
- Bit-rate reduction involves encoding information using fewer bits than the original representation.
  
- Types of compression:
  - ◆ **Lossless compression.** Reduces bits by identifying and eliminating statistical redundancy, **no information is lost.**
  - ◆ **Lossy compression.** Reduces bits by identifying **unnecessary information and removing it.**
  - ◆ **Arithmetic Coding.** Used in lossless and lossy algorithms. The frequently seen symbols are encoded with fewer bits than rarely seen symbols. **Encodes the entire message into a single number.**

# Lossy Algorithms

Hand: Original PNG Data (63 KByte)



Hand: 90% JPEG Data (56 KByte)



# Lossy Compression

---

- Lossy compression is compression in which **some of the information** from the original message sequence **is lost**.
- This means the original sequences **cannot be regenerated** from the compressed sequence.

# Lossy Compression

---

- Lossy compression is the class of data encoding methods that **uses inexact approximations** to represent the content.
- Well-designed lossy compression technology often reduces file sizes significantly before degradation is noticed by the end-user.

# Lossy Compression

---

→ Lossy compression is most commonly used to compress multimedia data (audio, video, and images), especially in applications such as streaming media.



# Lossy Compression (Examples)

---





# Lossy Compression (Example)

---



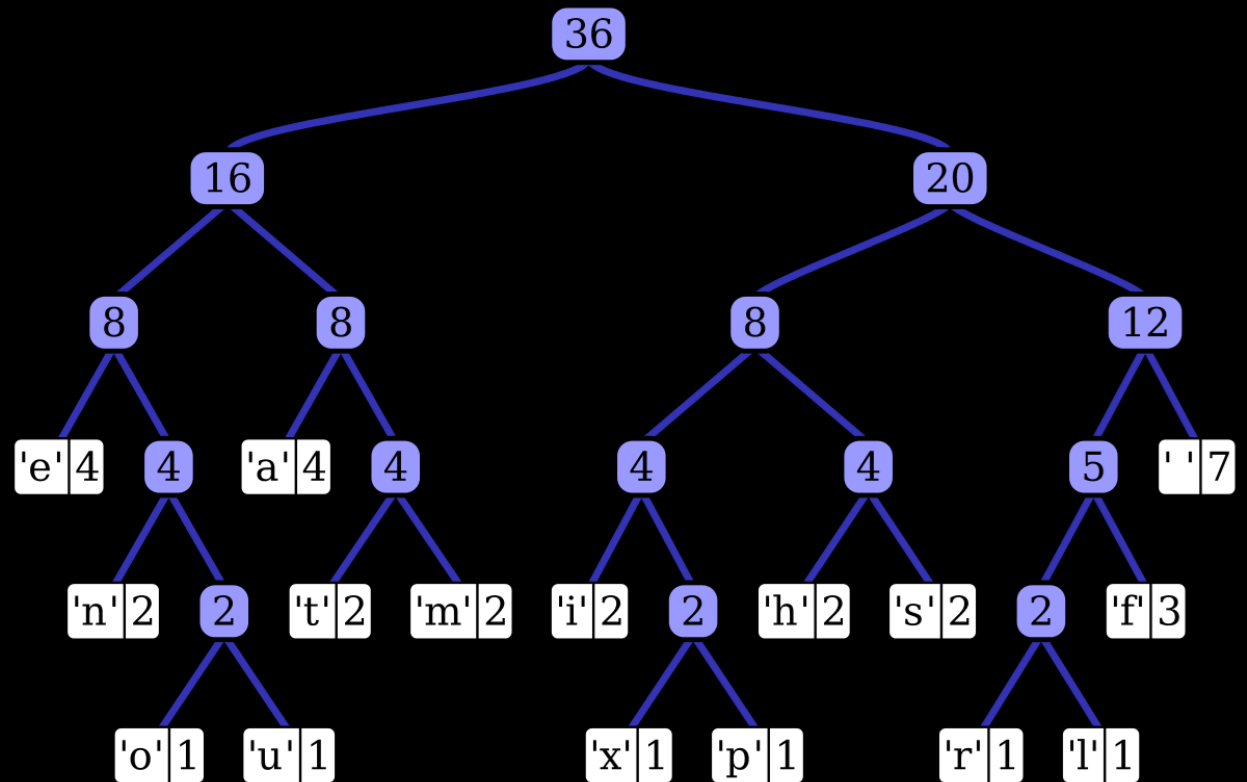
# Lossless Compression

# Introduction

---

- Reduces bits by identifying and eliminating statistical redundancy.
- No information is lost. This algorithm allows the original data to be perfectly reconstructed from the compressed data.
- Examples of Lossless Compression Algorithms:
  - ◆ Huffman Code.
  - ◆ LZ77.
  - ◆ LZ78.
  - ◆ LZW.

# Huffman Code



# Huffman Code

---

- David Huffman developed the algorithm as a student in a class on information theory at MIT in 1950.
- The idea behind Huffman coding is assign binary codes the most short as possible to those symbols that occur most frequently in the data.
- Symbols infrequently binary codes will be assigned greater length.
- The Huffman algorithm build a binary tree, this tree is used to assign codes to the input symbols.



- David Huffman developed the algorithm as a student in a class on information theory at MIT in 1950.
- The idea behind Huffman coding is assign binary codes the most short as possible to those symbols that occur most frequently in the data.
- Symbols infrequently binary codes will be assigned greater length.
- The Huffman algorithm build a binary tree, this tree is used to assign codes to the input symbols.



# Huffman Code (Strategy)

---

1. A first pass on the data to compress is performed to obtain the statistics of symbols.
1. Symbols are arranged in a list according to its probability. This ordered list symbols are the initial node to the tree construction. If the list contains only one node, finish.
1. Select the two nodes (L and R) from the list with the smallest probabilities.
1. A new node (F) is created and built a subtree being F the father of L and R the left and right respectively children. The arc between the F and L nodes are labeled 0 and the arc between nodes F and R is labeled as one.
1. F is added to the sorted list, with the probability value equal to the sum of to L and R.

# Huffman Code (Example)

---

1

9	12	23	29	32	64	66
E	D	G	A	C	B	F



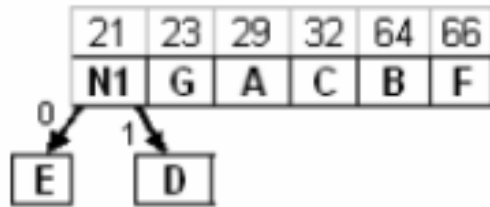
# Huffman Code (Example)

---

1

9	12	23	29	32	64	66
E	D	G	A	C	B	F

2



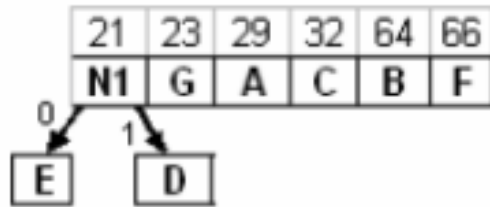
# Huffman Code (Example)

---

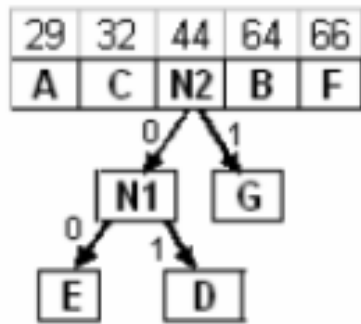
1

9	12	23	29	32	64	66
E	D	G	A	C	B	F

2



3

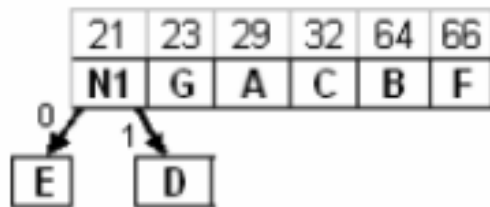


# Huffman Code (Example)

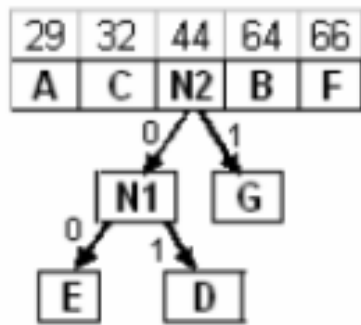
1

9	12	23	29	32	64	66
E	D	G	A	C	B	F

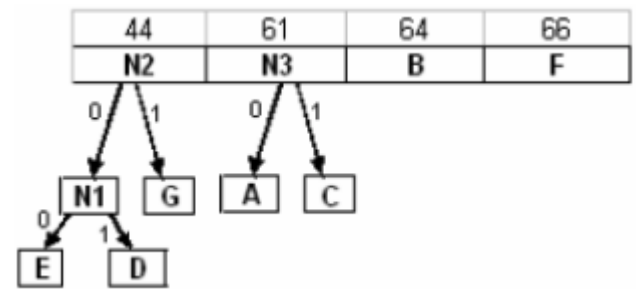
2



3



4

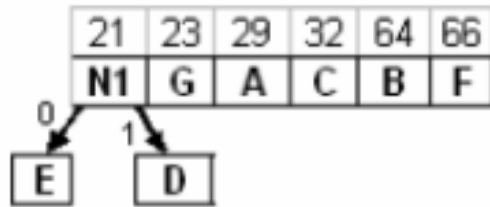


# Huffman Code (Example)

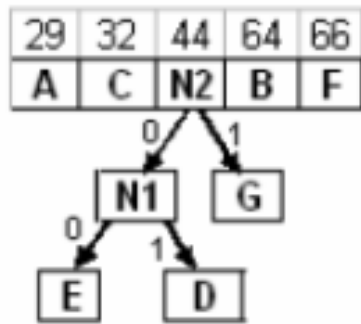
1

9	12	23	29	32	64	66
E	D	G	A	C	B	F

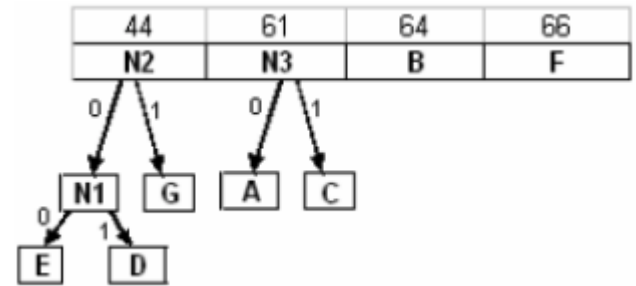
2



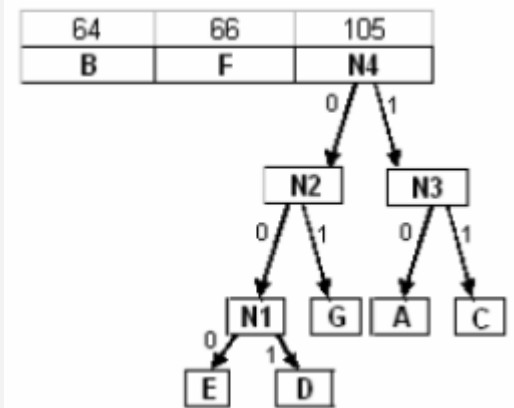
3



4

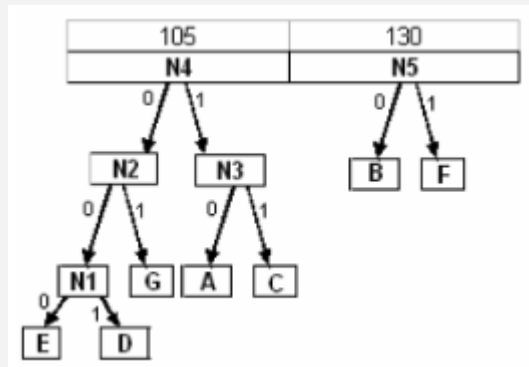


5

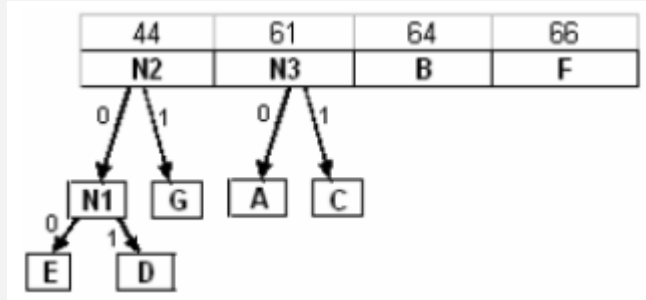


# Huffman Code (Example)

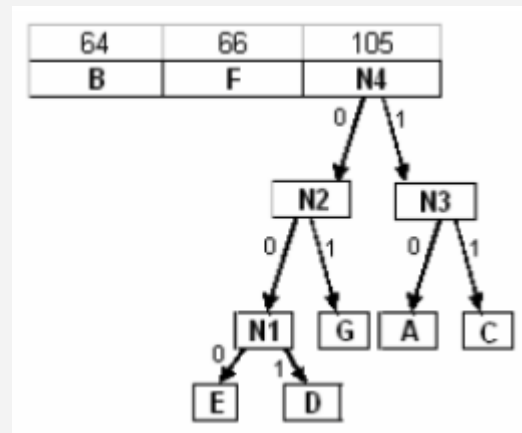
6



4

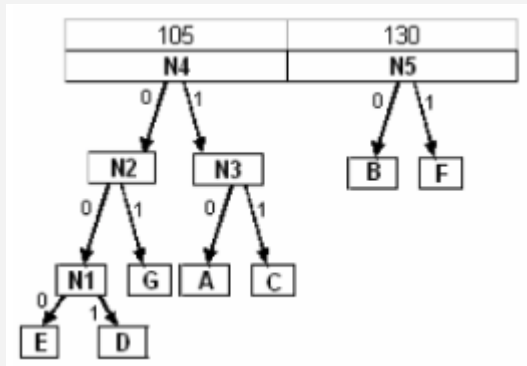


5

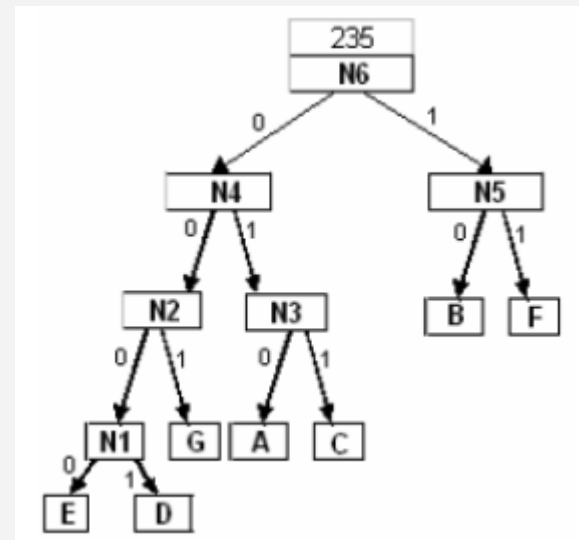


# Huffman Code (Example)

6



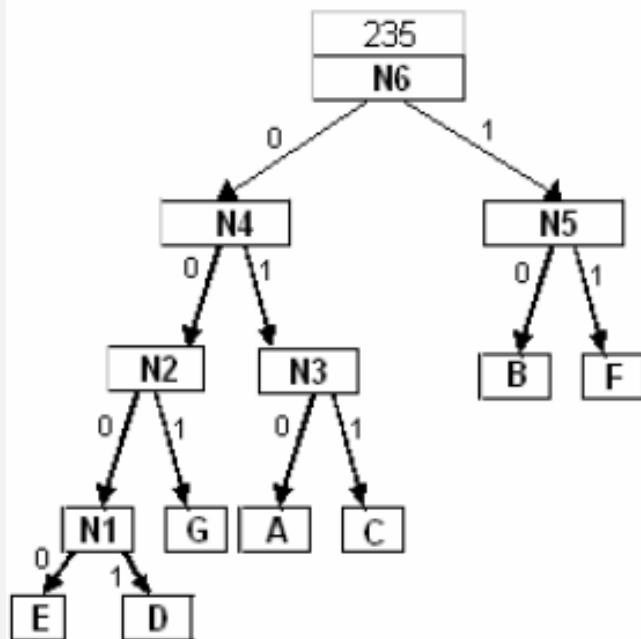
7



# Huffman Code (Example)

---

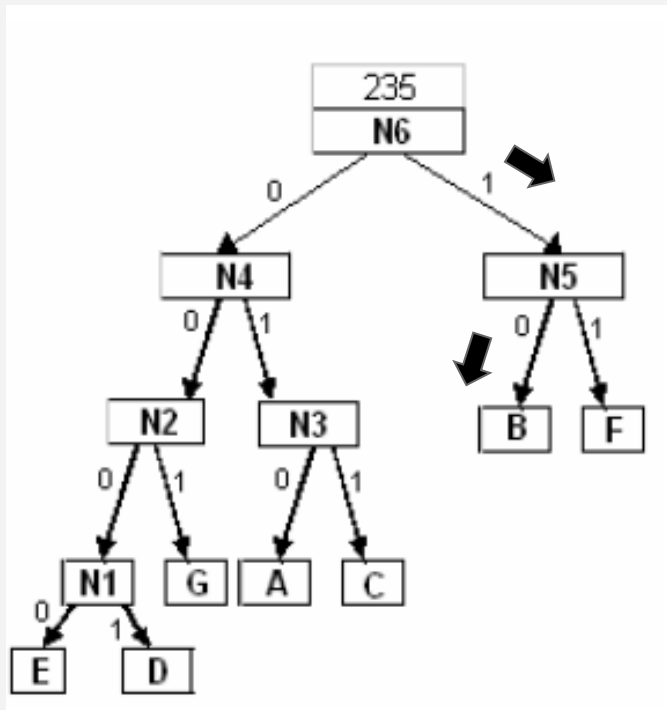
RESULT



OUTPUT

# Huffman Code (Example)

## RESULT



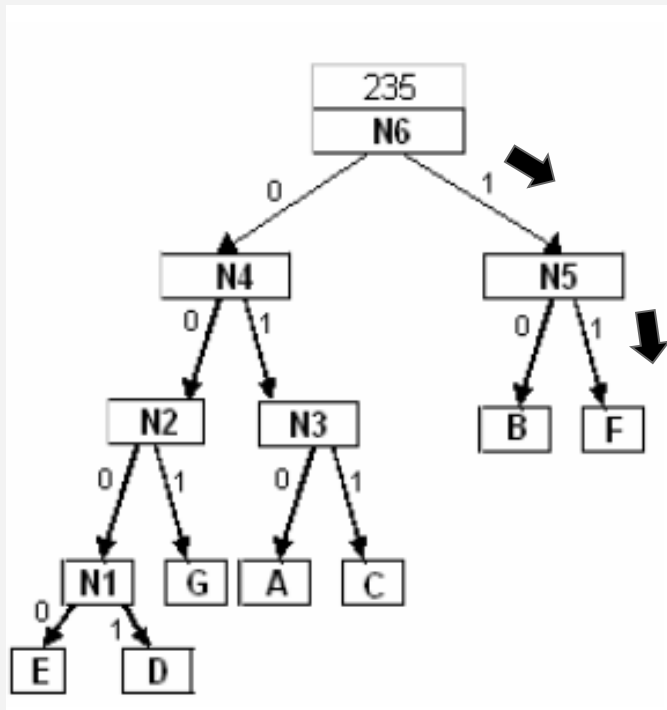
## OUTPUT

Símbolo	Código
B	10



# Huffman Code (Example)

## RESULT

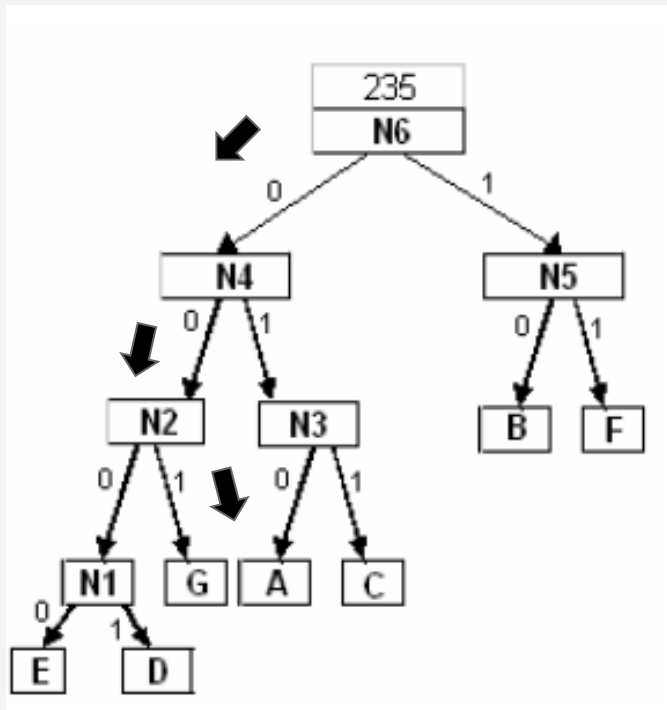


## OUTPUT

Símbolo	Código
B	10
F	11

# Huffman Code (Example)

## RESULT

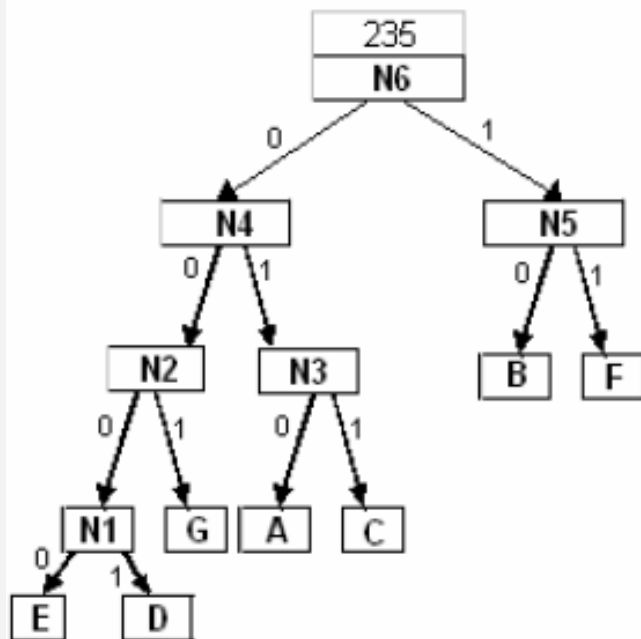


## OUTPUT

Símbolo	Código
B	10
F	11
G	001

# Huffman Code (Example)

## RESULT

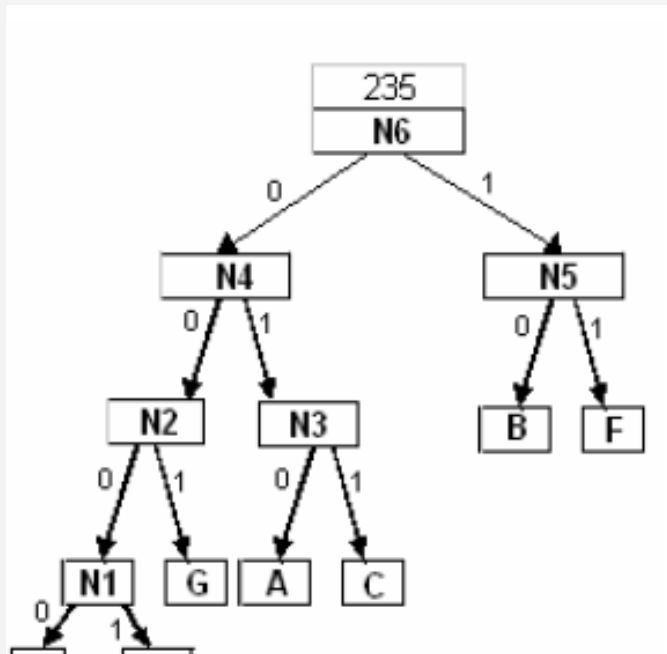


## OUTPUT

Símbolo	Código
B	10
F	11
G	001
A	010
C	011
E	0000
D	0001

# Huffman Code (Example)

## RESULT



## OUTPUT

Símbolo	Código
B	10
F	11
G	001
A	010
C	011
E	0000
D	0001

- These codes are stored with the compressed data, so the decompressor can reconstruct the original data.
- The compressor constructs the Huffman tree and starts decompression positioned at the root of the tree.

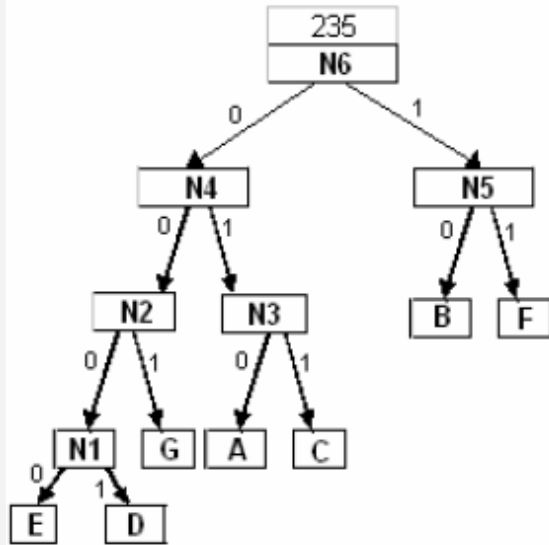
# Huffman Code (Decompression)

---

1. The decompressor reads the first bit of the compressed data, if zero, moves to the right child, if it is one, it moves to the left child.
2. The process is repeated, the next bit of the compressed data is read and is moving towards the right child if the read bit is a zero or left if the bit read is a one.
3. When the process reaches a leaf on the tree, the decompressor find the original code and returns the value of the symbol as uncompressed data.
4. The decompressor is again placed in the root of the tree and the same process is repeated.

# Huffman Code (Decompression)

## RESULT



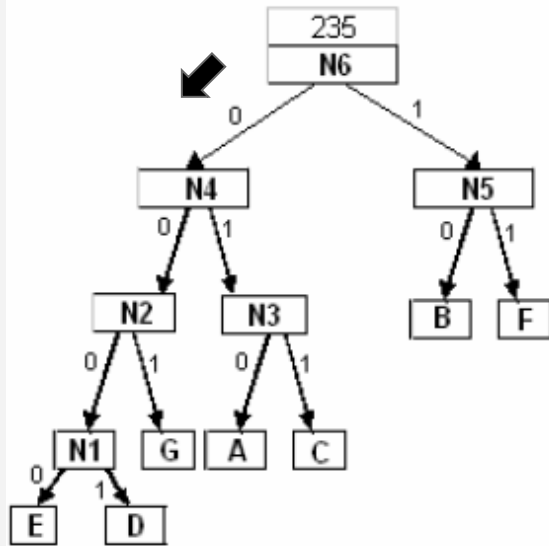
## OUTPUT

Símbolo	Código
B	10
F	11
G	001
A	010
C	011
E	0000
D	0001

→ Decompress the following sequence: 011010110000

# Huffman Code (Decompression)

## RESULT



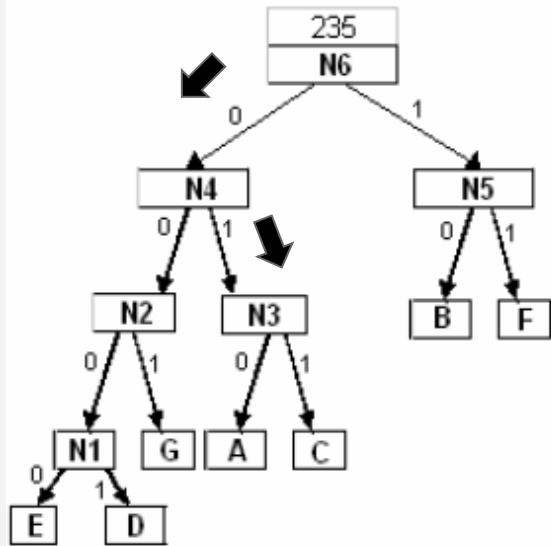
## OUTPUT

Símbolo	Código
B	10
F	11
G	001
A	010
C	011
E	0000
D	0001

→ Decompress the following sequence: **0**11010110000

# Huffman Code (Decompression)

## RESULT



## OUTPUT

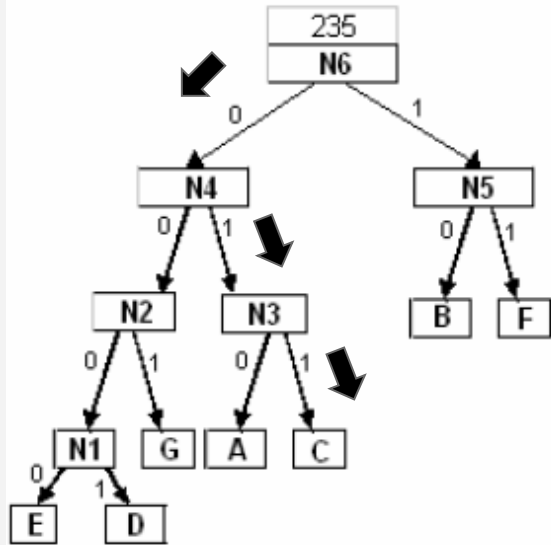
Símbolo	Código
B	10
F	11
G	001
A	010
C	011
E	0000
D	0001

→ Decompress the following sequence: **01**1010110000



# Huffman Code (Decompression)

## RESULT



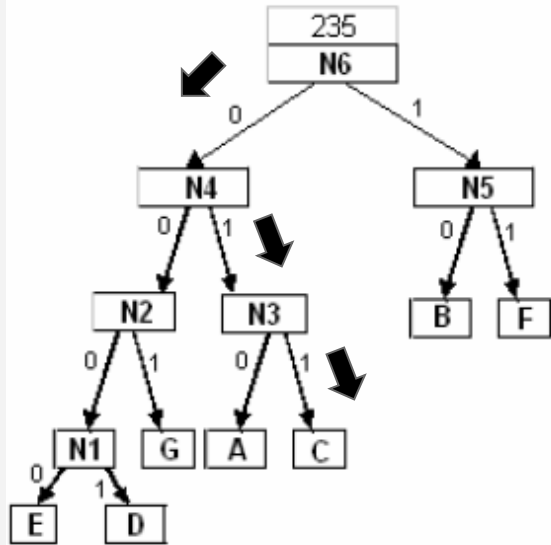
## OUTPUT

Símbolo	Código
B	10
F	11
G	001
A	010
C	011
E	0000
D	0001

→ Decompress the following sequence: **011**010110000

# Huffman Code (Decompression)

## RESULT



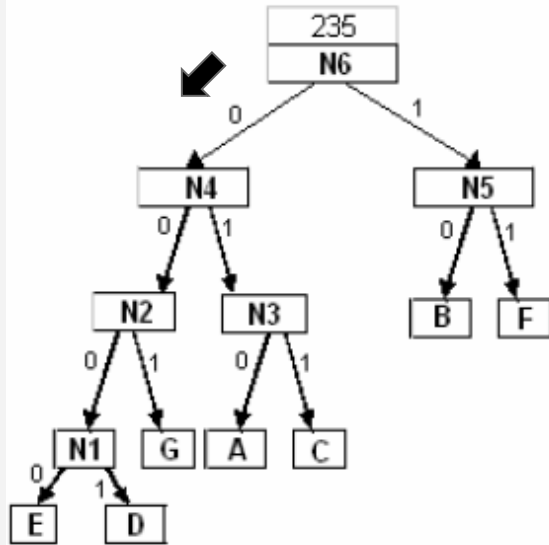
## OUTPUT

Símbolo	Código
B	10
F	11
G	001
A	010
C	011
E	0000
D	0001

→ Decompress the following sequence: 011010110000

# Huffman Code (Decompression)

## RESULT



## OUTPUT

Símbolo	Código
B	10
F	11
G	001
A	010
C	011
E	0000
D	0001

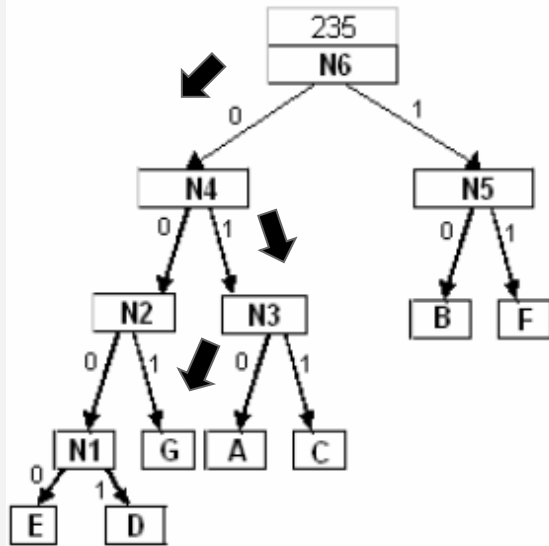
→ Decompress the following sequence: **011**010110000

## RESULT

C

# Huffman Code (Decompression)

## RESULT



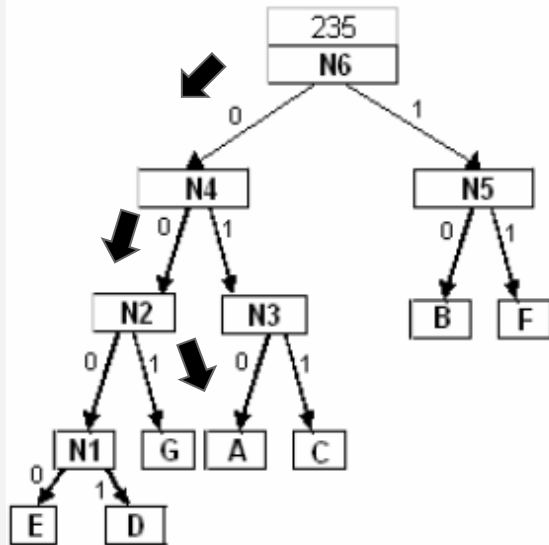
## OUTPUT

Símbolo	Código
B	10
F	11
G	001
A	010
C	011
E	0000
D	0001

→ Decompress the following sequence: 011010110000

# Huffman Code (Decompression)

## RESULT



## OUTPUT

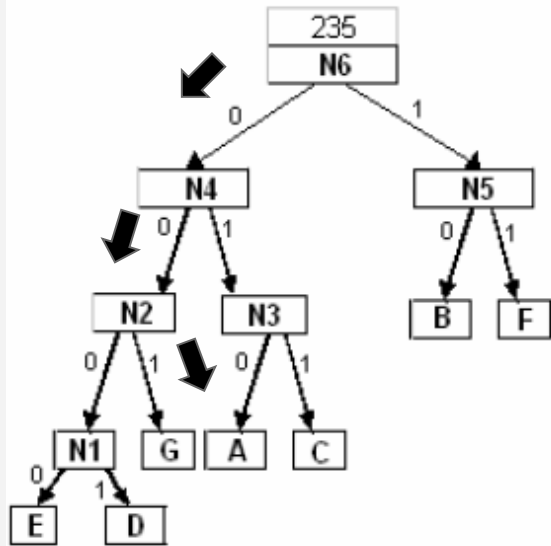
Símbolo	Código
B	10
F	11
G	001
A	010
C	011
E	0000
D	0001

→ Decompress the following sequence: 011010110000

C A

# Huffman Code (Decompression)

## RESULT



## OUTPUT

Símbolo	Código
B	10
F	11
G	001
A	010
C	011
E	0000
D	0001

→ Decompress the following sequence: 011010110000

C A F E

# Huffman Code (Example)

---

→ Compress the following text:

“abfabcaecedba”



# Huffman Code (Example)

---

→ Compress the following text:

“abfabcaecedba”

**RESULT**

*frequency table*

(f,1)	(d,1)	(c,2)	(e,2)	(b,3)	(a,4)
-------	-------	-------	-------	-------	-------

**OUTPUT**

# Huffman Code (Example)

---

→ Compress the following text:

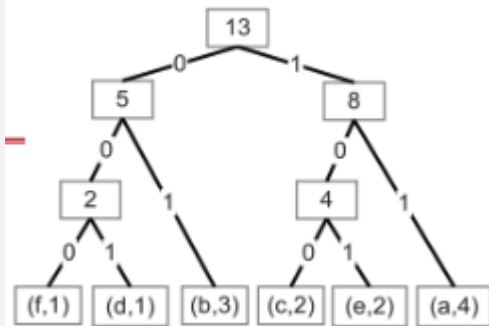
“abfabcaecedba”

**RESULT**

*frequency table*

(f,1)	(d,1)	(c,2)	(e,2)	(b,3)	(a,4)
-------	-------	-------	-------	-------	-------

*Huffman tree*



**OUTPUT**

# Huffman Code (Example)

→ Compress the following text:

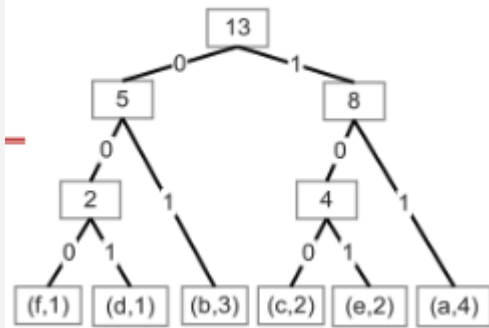
“abfabcaecedba”

RESULT

*frequency table*

(f,1)	(d,1)	(c,2)	(e,2)	(b,3)	(a,4)
-------	-------	-------	-------	-------	-------

*Huffman tree*



OUTPUT

*Huffman code*

a	11
e	101
c	100
b	01
d	001
f	000

11010001101100111011001010010111

# LZ77



# LZ77

---

- Developed by Abraham **L**empel and Jacob **Z**iv in 19**77**. Also called LZ1.
- The principle of this method is to use part of the entrance sequence as a dictionary.
- The encoder keeps an intermediate string (or buffer) memory called "window" and moves the data entered in it, from right to left, as to be encoding symbols.
- The method is based on a sliding window.

# LZ77

---

- Developed by Abraham Lempel and Jacob Ziv in 1977. Also called LZ1.
- The principle of this method is to use part of the entrance sequence as a dictionary.
- The encoder keeps an intermediate string (or buffer) memory called "window" and moves the data entered in it, from right to left, as to be encoding symbols.
- The method is based on a sliding window.



Abraham Lempel in 2007



- Developed by Abraham **L**empel and Jacob **Z**iv in 19**77**. Also called LZ1.
- The principle of this method is to use part of the entrance sequence as a dictionary.
- The encoder keeps an intermediate string (or buffer) memory called "window" and moves the data entered in it, from right to left, as to be encoding symbols.
- The method is based on a sliding window.



Abraham Lempel in 2007

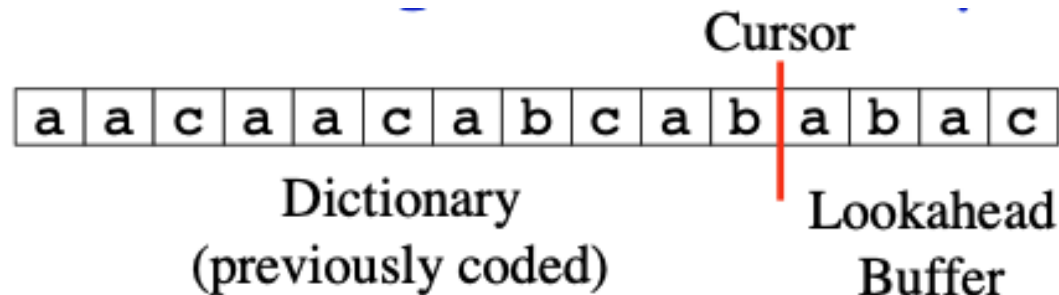


# LZ77 (Strategy)

---

→ The basic algorithm is very simple and loops executing the following steps:

1. Find the longest match of a string starting at the cursor and completely contained in the lookahead buffer to a string starting in the dictionary.



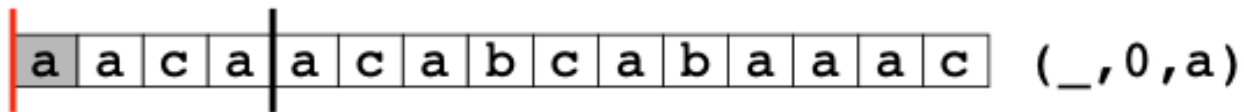
1. Output a triple  $(p, n, c)$  where:
  - $p$  = position of the longest match that starts in the dictionary (relative to the cursor).
  - $n$  = length of the longest match.
  - $c$  = next char in buffer beyond longest match.





# LZ77 (Example)


---


→ Compress the following text: "aacaacabcabaaac"



 Dictionary (size = 6)

 Longest match

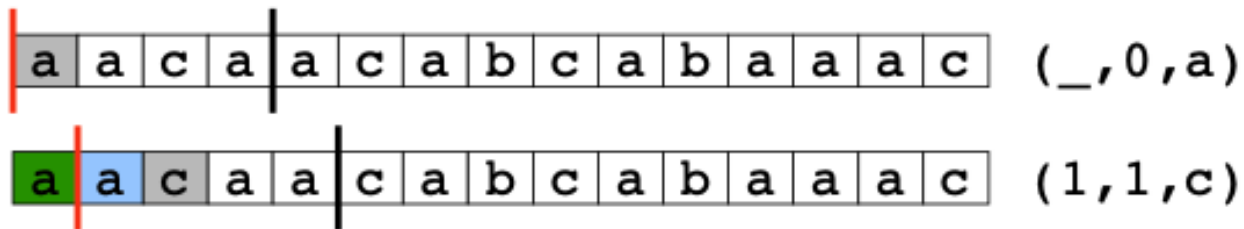
 Buffer (size = 4)


 Next character


# LZ77 (Example)

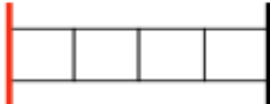
---


→ Compress the following text: "aacaacabcabaaac"



 Dictionary (size = 6)

 Longest match

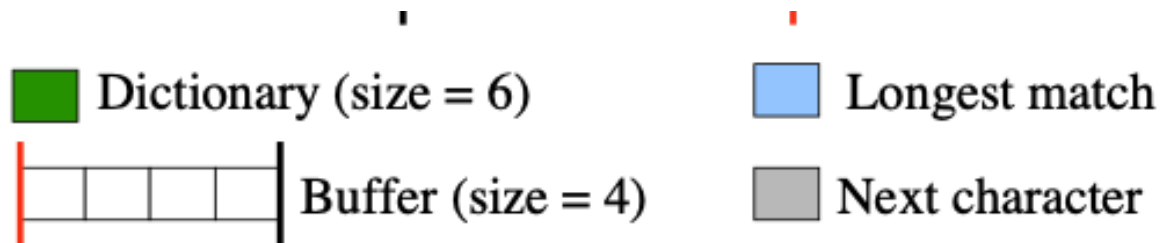
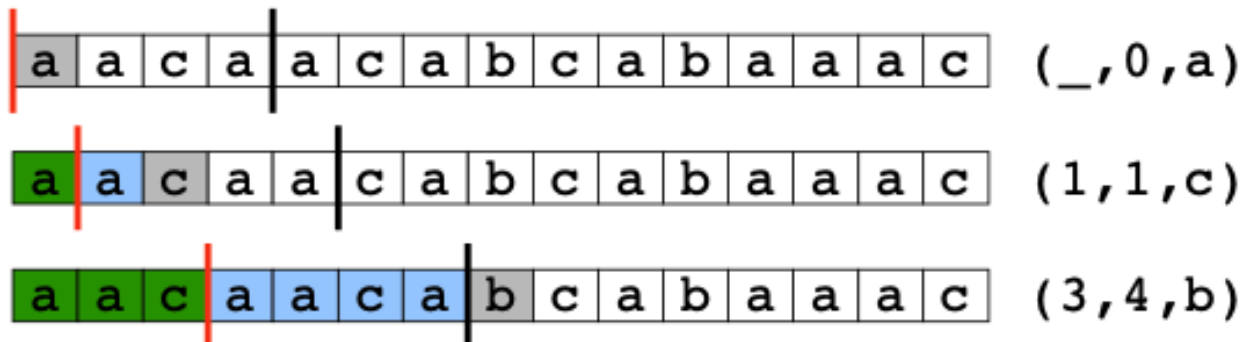
 Buffer (size = 4)

 Next character

# LZ77 (Example)

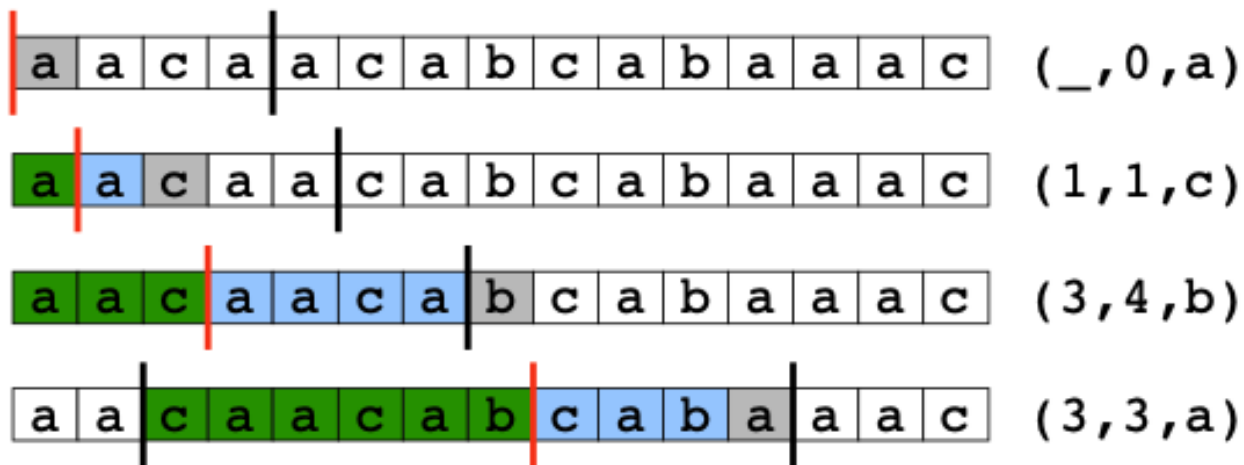
---

→ Compress the following text: "aacaacabcabaaac"



# LZ77 (Example)

→ Compress the following text: "aacaacabcabaaac"



Dictionary (size = 6)

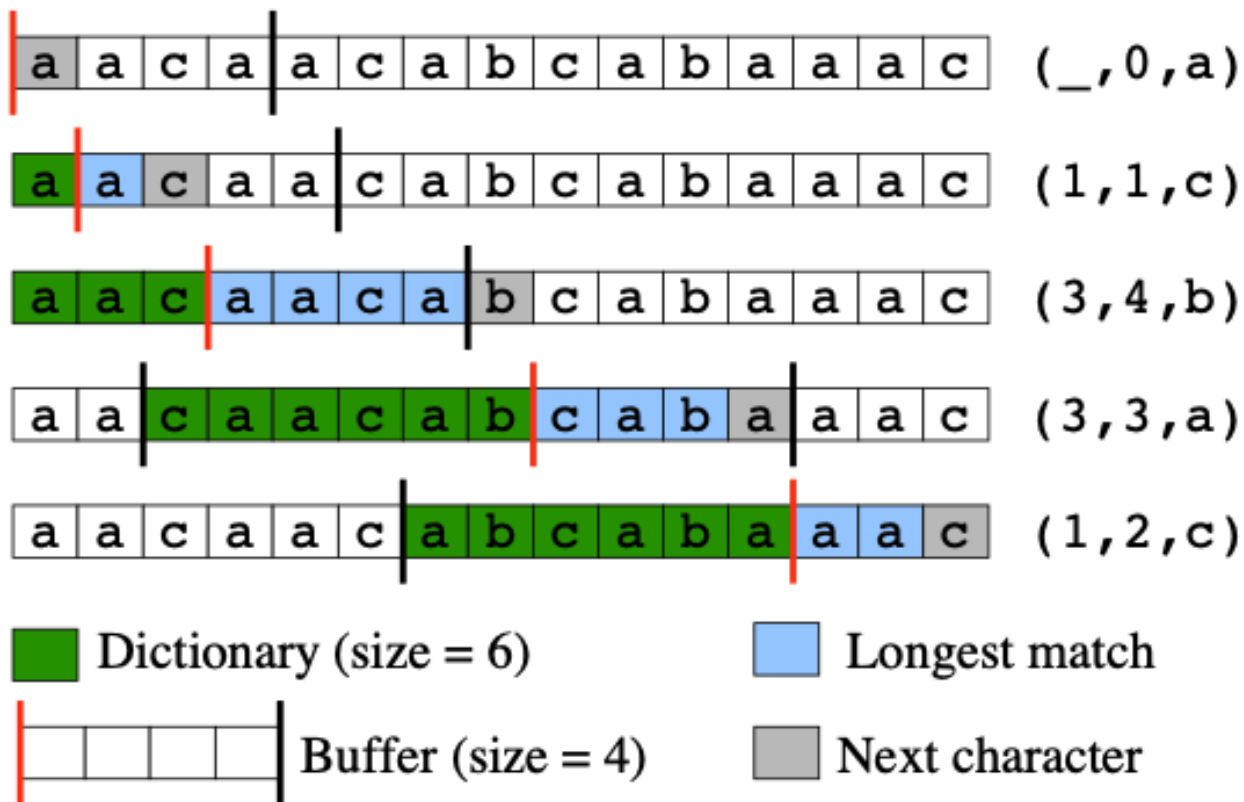
Longest match

Buffer (size = 4)

Next character

# LZ77 (Example)

→ Compress the following text: "aacaacabcabaaac"



# LZ77 (Decompression)

---

→ The decoder read each triplet  $(p, n, c)$ . Two cases can occur:

1.  $p$  is equal to zero

- $c$  is written to the output file.

1.  $p$  is different to zero

- The substring in the buffer starting position  $p$  (count from right to left) and length  $N$  is copied to the output file.
- $c$  is written to the output file.

# LZ77 (Decompression)

**Remember, this  
was the initial  
instruction**

→ Compress the following text: "aacaacabcabaaac"

(\_, 0, a)

(1, 1, c)

(3, 4, b)

(3, 3, a)

(1, 2, c)

# LZ77 (Decompression)

---

→ Compress the following text: "aacaacabcabaaac"

(\_, 0, a)

(1, 1, c)

(3, 4, b)

(3, 3, a)

(1, 2, c)



**And this was the  
output**



# LZ77 (Decompression)

---

→ Compress the following text: "aacaacabcabaaac"

➔ ( \_, 0 , a )

( 1 , 1 , c )

( 3 , 4 , b )

**a**

( 3 , 3 , a )

( 1 , 2 , c )

# LZ77 (Decompression)

---

→ Compress the following text: "aacaacabcabaaac"

(\_, 0, a)

➔ (1, 1, c)

(3, 4, b)

**aac**

(3, 3, a)

(1, 2, c)

# LZ77 (Decompression)

---

→ Compress the following text: "aacaacabcabaaac"

(\_, 0, a)

(1, 1, c)

➔ (3, 4, b)

**aac****aacab**

(3, 3, a)

(1, 2, c)

# LZ77 (Decompression)

---

→ Compress the following text: "aacaacabcabaaac"

(\_, 0, a)

(1, 1, c)

(3, 4, b)

**aacaacab****caba**

**→** (3, 3, a)

(1, 2, c)

# LZ77 (Decompression)

---

→ Compress the following text: "aacaacabcabaaac"

(\_, 0, a)

(1, 1, c)

(3, 4, b)

(3, 3, a)

**aacaacabcaba****aac**

**→** (1, 2, c)

# LZ77 (Example)

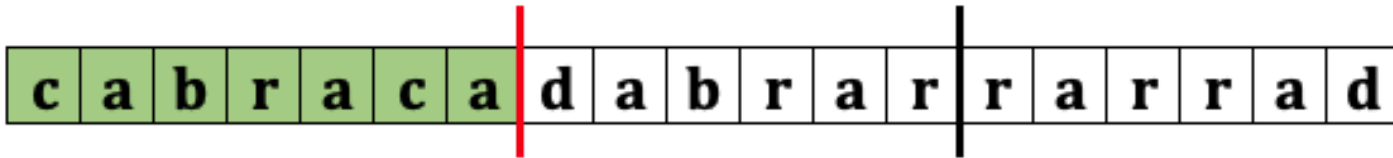
---

- Compress the following text: “cabracadabrarrarrad”
  - ◆ Dictionary (size = 7)
  - ◆ Buffer (size = 6)

# LZ77 (Example)

---

- Compress the following text: “cabracadabrarrarrad”
- ◆ Diccionario (size = 7)
  - ◆ Buffer (size = 6)
  - ◆ Using full Diccionario.



# LZ77 (Example)

---

→ Compress the following text: “cabracadabrarrarrad”

- ◆ Diccionario (size = 7)
- ◆ Buffer (size = 6)
- ◆ Using full Diccionario.

c	a	b	r	a	c	a	d	a	b	r	a	r	r	a	r	r	a	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

<\_, 0, d>

c	a	b	r	a	c	a	d	a	b	r	a	r	r	a	r	r	a	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

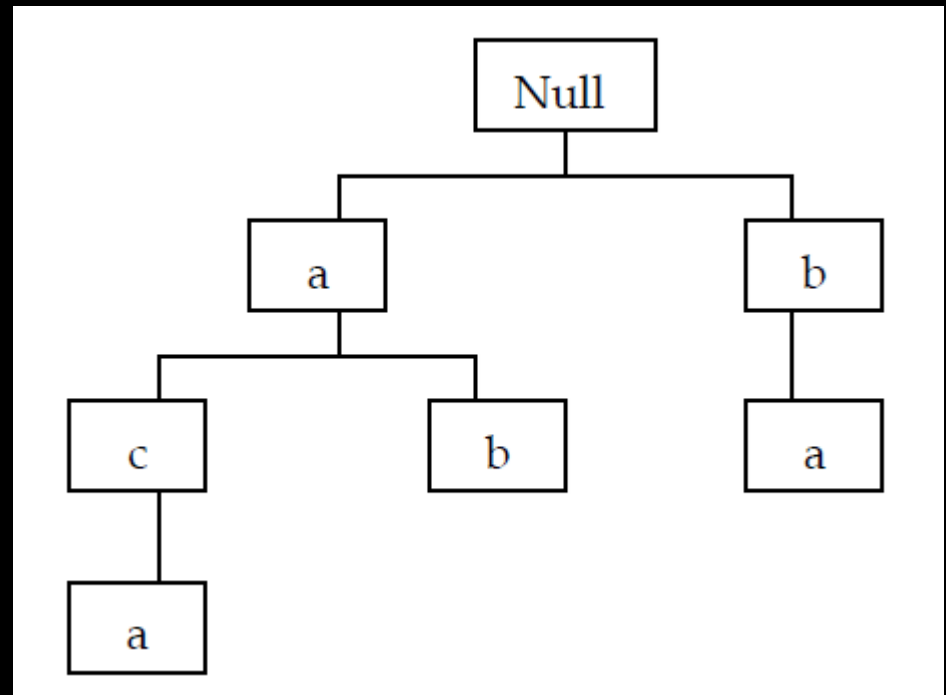
<7, 4, r>

c	a	b	r	a	c	a	d	a	b	r	a	r	r	a	r	r	a	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

<3, 5, d>



# LZ78



# LZ78

---

- Developed by Abraham **L**empel and Jacob **Z**iv in 19**78**. Also called LZ2.
- LZ78 does not use a window on the data to be compressed.
- There are a dictionary containing the chains that have occurred previously.
- The dictionary is initially empty and its size is limited by available memory.

# LZ78

---

- The codeword consists only of two fields, a pointer or index field that identifies the position of a string in the dictionary and a field containing the last symbol has been read from the entrance.
- The representation of the dictionary can be a tree structure called trie.

- The codeword consists only of two fields, a pointer or index field that identifies the position of a string in the dictionary and a field containing the last symbol has been read from the entrance.
- The representation of the dictionary can be a tree structure called trie.

# LZ78 (Strategy)

---

→ The basic algorithm is very simple and loops executing the following steps:

While there are symbols at the entrance:

$S = \text{Null}$ ,  $\text{Pos} = 0$

$X = \text{next input symbol}$ ,  $S = SX$

    While  $S$  exists in the dictionary

$\text{Pos} = \text{Pos}(S)$

$X = \text{next input symbol}$ ,  $S = SX$

    Output:  $(\text{Pos}, X)$

    Save  $S$  in the dictionary

# LZ78 (Example)

---

→ Compress the following text: “abacabacaba”

# LZ78 (Example)

---

→ Compress the following text: “**a**bacabacaba”

Diccionario		Codeword
0	Null	
1	a	(0, a)

# LZ78 (Example)

---

→ Compress the following text: “**ab**acabacaba”

Diccionario		Codeword
0	Null	
1	a	(0, a)
2	b	(0, b)



# LZ78 (Example)

---

→ Compress the following text: “**ab**acabacaba”

Diccionario		Codeword
0	Null	
1	a	(0, a)
2	b	(0, b)
3	ac	(1, c)

# LZ78 (Example)

---

→ Compress the following text: “**a****b****a****c****a****b****a****c****a****b****a**”

Diccionario		Codeword
0	Null	
1	a	(0, a)
2	b	(0, b)
3	ac	(1, c)
4	ab	(1, b)

# LZ78 (Example)

---

→ Compress the following text: “**a****b****a****c****a****b****a****c****a****b****a**”

Diccionario		Codeword
0	Null	
1	a	(0, a)
2	b	(0, b)
3	ac	(1, c)
4	ab	(1, b)
5	aca	(3, a)

# LZ78 (Example)

---

→ Compress the following text: “**a****b****a****c****a****b****a****c****a****b****a**”

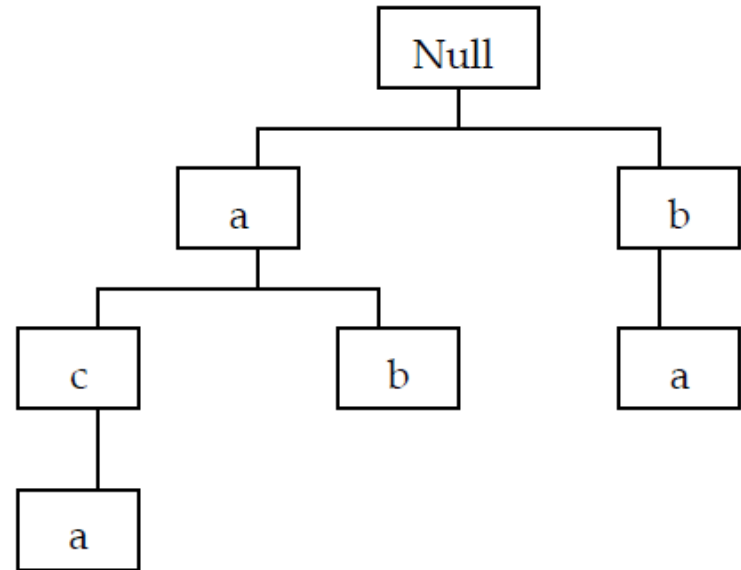
Diccionario		Codeword
0	Null	
1	a	(0, a)
2	b	(0, b)
3	ac	(1, c)
4	ab	(1, b)
5	aca	(3, a)
6	ba	(2, a)

# LZ78 (Example)

---

→ Compress the following text: “**a****b****a****c****a****b****a****c****a****b****a**”

Diccionario		Codeword
0	Null	
1	a	(0, a)
2	b	(0, b)
3	ac	(1, c)
4	ab	(1, b)
5	aca	(3, a)
6	ba	(2, a)



# LZ78 (Decompression)

**Remember, this  
was the initial  
instruction**

→ Compress the following text: “abacabacaba”

Diccionario		Codeword
0	Null	
1	a	(0, a)
2	b	(0, b)
3	ac	(1, c)
4	ab	(1, b)
5	aca	(3, a)
6	ba	(2, a)

# LZ78 (Decompression)

---

→ Compress the following text: “abacabacaba”

Diccionario		Codeword
0	Null	
1	a	(0, a)
2	b	(0, b)
3	ac	(1, c)
4	ab	(1, b)
5	aca	(3, a)
6	ba	(2, a)

**And this was the  
output**

# LZ78 (Decompression)

---

→ Compress the following text: “abacabacaba”

Diccionario		Codeword
0	Null	
1	a	→ (0, a)
2	b	(0, b)
3	ac	(1, c)
4	ab	(1, b)
5	aca	(3, a)
6	ba	(2, a)

**a**



# LZ78 (Decompression)

---

→ Compress the following text: “abacabacaba”

Diccionario		Codeword
0	Null	
1	a	(0, a)
2	b	(0, b)
3	ac	(1, c)
4	ab	(1, b)
5	aca	(3, a)
6	ba	(2, a)



**a****b**

# LZ78 (Decompression)

---

→ Compress the following text: “abacabacaba”

Diccionario		Codeword
0	Null	
1	a	(0, a)
2	b	(0, b)
3	ac	(1, c)
4	ab	(1, b)
5	aca	(3, a)
6	ba	(2, a)



ab**ac**

# LZ78 (Decompression)

---

→ Compress the following text: “abacabacaba”

Diccionario		Codeword
0	Null	
1	a	(0, a)
2	b	(0, b)
3	ac	(1, c)
4	ab	(1, b)
5	aca	(3, a)
6	ba	(2, a)



**abacab**

# LZ78 (Decompression)

---

→ Compress the following text: “abacabacaba”

Diccionario		Codeword
0	Null	
1	a	(0, a)
2	b	(0, b)
3	ac	(1, c)
4	ab	(1, b)
5	aca	(3, a)
6	ba	(2, a)



**abacabaca**

# LZ78 (Decompression)

---

→ Compress the following text: “abacabacaba”

Diccionario		Codeword
0	Null	
1	a	(0, a)
2	b	(0, b)
3	ac	(1, c)
4	ab	(1, b)
5	aca	(3, a)
6	ba	→ (2, a)

**abacabacaba**

# LZ78 (Example)

---

→ Compress the following text: “abracadabrarabarabara”

# LZ78 (Example)

---

→ Compress the following text: “abracadabrarabarabara”

Diccionario		Codeword
0	Null	
1	a	(0, a)
2	b	(0,a)
3	r	(0, a)
4	ac	(1, c)
5	ad	(1, d)
6	ab	(1, b)
7	ra	(3, a)
8	rab	(7, b)
9	ar	(1, r)
10	aba	(6, a)
11		(7,EOF)

**LZW**



# LZW

---

- Developed by Abraham **L**empel and Jacob **Z**iv and Terry **W**elch.
- It is a variant of LZ78 since it is the one that is most commonly used in practice.
- The algorithm maintains a dictionary of strings (sequence of bytes).

# LZW

---

- The codeword table is initialized with one entry of each of the 256 possible byte value. These are strings of length one.
- As the algorithm progresses it will add new strings to the dictionary such that each string is only added if a prefix one byte shorter is already in the dictionary.
  - ◆ For example, John is only added if Joh had previously appeared in the message sequence.

- The dictionary is initialized with one entry of each of the 256 possible byte value. These are strings of length one.
- As the algorithm progresses it will add new strings to the dictionary such that each string is only added if a prefix one byte shorter is already in the dictionary.
  - ◆ For example, John is only added if Joh had previously appeared in the message sequence.

# LZW (Strategy)

---

→ The basic algorithm is very simple and loops executing the following steps:

From  $i = 0$  to 255

Pos( $i$ )  $\leftarrow$   $S_i$

$S = \text{null}$

While there are symbols at the entrance:

$X = \text{next input symbol}$

If  $SX$  exist in the dictionary

$S \leftarrow SX$

$X = \text{next input symbol}$

Write to the output the position of  $S$  in the dictionary.

Add  $SX$  in the dictionary

$S \leftarrow X$

# LZW (Strategy)

---

## LZW ENCODING

```
*    PSEUDOCODE
1    Initialize table with single character strings
2    P = first input character
3    WHILE not end of input stream
4        C = next input character
5        IF P + C is in the string table
6            P = P + C
7        ELSE
8            output the code for P
9            add P + C to the string table
10           P = C
11        END WHILE
12    output code for P
```

# LZW (Example)

---

→ Compress the following text: “abacabaca”

# LZW (Example)

---

→ Compress the following text: “abacabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3		
4		
5		
6		
7		
8		
9		

# LZW (Example)

---

→ Compress the following text: “**a**bacabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3		
4		
5		
6		
7		
8		
9		

**EXECUTION**

C = a  
✓  
P + C = a

**OUTPUT**



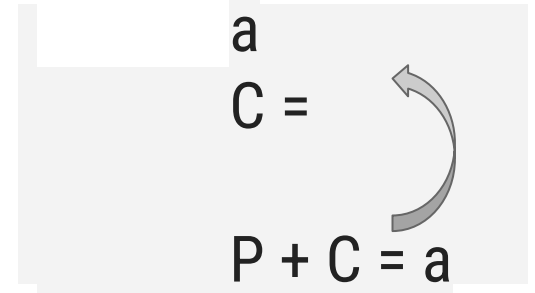
# LZW (Example)

---

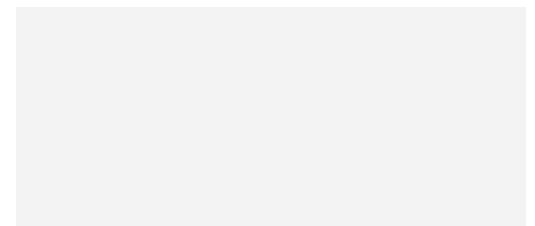
→ Compress the following text: “**a**bacabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3		
4		
5		
6		
7		
8		
9		

**EXECUTION**



**OUTPUT**




# LZW (Example)

---

→ Compress the following text: “**ab**acabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3		
4		
5		
6		
7		
8		
9		

## EXECUTION

a  
C = b  
  
P + C = ab

## OUTPUT

0


# LZW (Example)

---

→ Compress the following text: “**ab**acabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4		
5		
6		
7		
8		
9		

## EXECUTION

a  
C = b  
  
P + C = ab

## OUTPUT

0


# LZW (Example)

---

→ Compress the following text: “a**ba**cabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5		
6		
7		
8		
9		

## EXECUTION

b  
C = a  
  
P + C = ba

## OUTPUT

0 1


# LZW (Example)

---

→ Compress the following text: “ab**ac**abaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6		
7		
8		
9		

## EXECUTION

a  
C = c  
  
P + C = ac

## OUTPUT

0 1 0


# LZW (Example)

---

→ Compress the following text: “aba**ca**baca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7		
8		
9		

## EXECUTION

c  
C = a  
  
P + C = ca

## OUTPUT

0 1 0 2

# LZW (Example)

---

→ Compress the following text: “abac**ab**aca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7		
8		
9		

## EXECUTION

a  
C = b  
✓  
P + C = ab

## OUTPUT

0 1 0 2

# LZW (Example)

---

→ Compress the following text: “abac**ab**aca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8		
9		

## EXECUTION

ab

C =

a



P + C =

## OUTPUT

0 1 0 2 3



# LZW (Example)

---

→ Compress the following text: “abacab**aca**”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8		
9		

## EXECUTION

a  
C = c  
✓  
P + C = ac

## OUTPUT

0 1 0 2 3

# LZW (Example)

---

→ Compress the following text: “abacab**aca**”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8	aca	261
9		

## EXECUTION

ac

C =

a



P + C =

## OUTPUT

0 1 0 2 3 5

# LZW (Example)

---

→ Compress the following text: “abacab**aca**”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8	aca	261
9		

## EXECUTION

ac

C =

a



P + C =

## OUTPUT

0 1 0 2 3 5 0

# LZW (Example)

→ Compress the following text: “abacab**aca**”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8	aca	261
9		

## EXECUTION

ac

C =

a



P + C =

## OUTPUT

97 98 97 99 256 258 97

# LZW (Decompression)

---

→ Compress the following text: “abacabaca”

# LZW (Decompression)

Remember, this  
was the initial  
instruction

→ Compress the following text: “abacabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8	aca	261
9		

OUTPUT

0 1 0 2 3 5 0

# LZW (Decompression)

---

→ Compress the following text: “abacabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8	aca	261
9		

**OUTPUT**

0 1 0 2 3 5 0

**This is the Dictionary  
generated with the  
algorithm**

# LZW (Decompression)

---

→ Compress the following text: “abacabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8	aca	261
9		

**OUTPUT**

0 1 0 2 3 5 0

**Output generated with  
the execution of the  
algorithm.**



# LZW (Decompression)

---

→ Compress the following text: “abacabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8	aca	261
9		

a

**OUTPUT**

0 1 0 2 3 5 0

↑

# LZW (Decompression)

---

→ Compress the following text: “abacabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8	aca	261
9		

a

OUTPUT
0 1 0 2 3 5 0

↑

# LZW (Decompression)

---

→ Compress the following text: “abacabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8	aca	261
9		

ab

**OUTPUT**

0 1 0 2 3 5 0

↑

# LZW (Decompression)

---

→ Compress the following text: “abacabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8	aca	261
9		

aba

OUTPUT
0 1 0 2 3 5 0

↑

# LZW (Decompression)

---

→ Compress the following text: “abacabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8	aca	261
9		

abac

**OUTPUT**

0 1 0 2 3 5 0

↑

# LZW (Decompression)


---

→ Compress the following text: “abacabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8	aca	261
9		

abacab

OUTPUT
0 1 0 2 3 5 0



# LZW (Decompression)


---

→ Compress the following text: “abacabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8	aca	261
9		

abacab**ac**

OUTPUT
0 1 0 2 3 5 0



# LZW (Decompression)

---


→ Compress the following text: “abacabaca”

Index	Dictionary	Codeword
0	a	97
1	b	98
2	c	99
3	ab	256
4	ba	257
5	ac	258
6	ca	259
7	aba	260
8	aca	261
9		

abacabaca

**OUTPUT**

0 1 0 2 3 5 0





# LZW (Example)

---

→ Compress the following text: "waccawacca"

# LZW (Example)

---

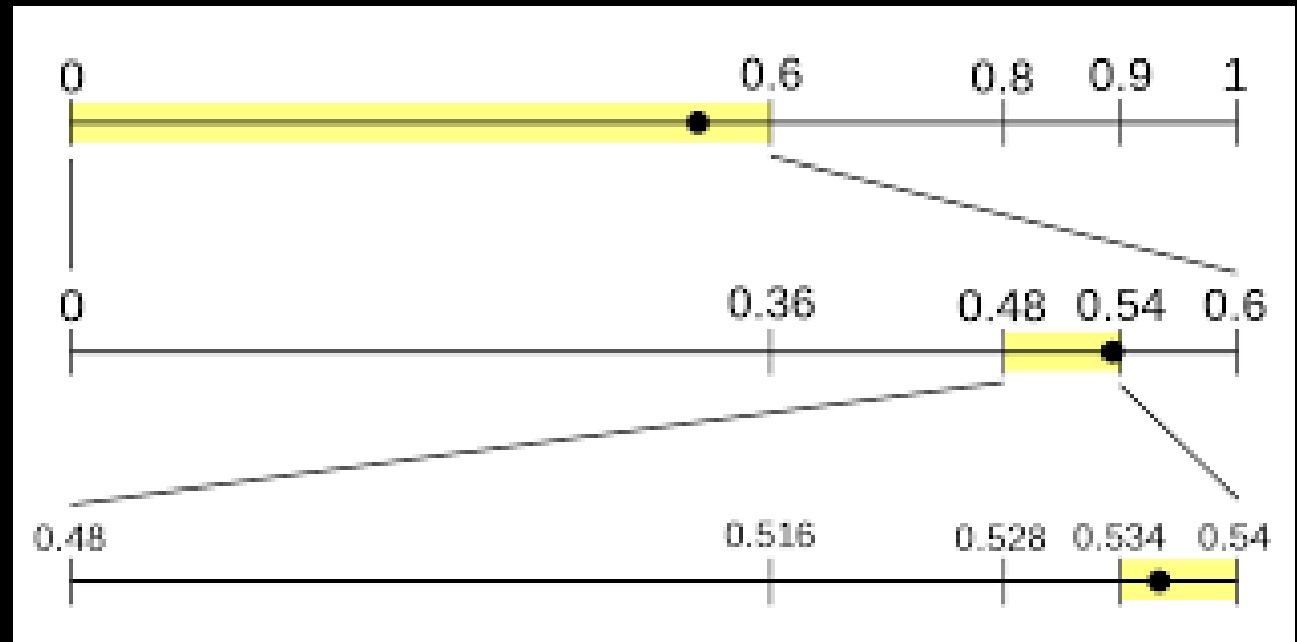
→ Compress the following text: “waccawacca”

Index	Dictionary	Codeword
0	a	97
1	c	99
2	w	119
3	wa	256
4	ac	257
5	cc	258
6	ca	259
7	aw	260
8	wac	261
9	cca	262

**OUTPUT**

2 0 1 1 0 3 5 0

# Arithmetic Encoding



# Arithmetic Encoding

---

- This is a common algorithm used in both lossless and lossy data compression algorithms.
- The frequently seen symbols are encoded with fewer bits than rarely seen symbols. It has some advantages over well-know techniques like Huffman Code.

# Arithmetic Encoding (Strategy)

---

- It converts the entire input data into a single floating point number  $n$  where  $(0.0 \leq n < 1.0)$ .
- The interval is divided into sub-intervals in the ratio of the probability of occurrence frequencies.
- For a startpoint and endpoint of an entire range the lower-limit of a character range is the upper-limit of the previous character.
- Therefore, each interval corresponds to one symbol.
- The reduced interval is partitioned recursively as more symbols are processed.

# Arithmetic Coding (Example)

---

→ Considering the string “ARBER”:

Symbol	A	B	E	R
Frequency	1	1	1	2
Probability	20%	20%	20%	40%

# Arithmetic Coding (Example)

---

→ Considering the string “ARBER”:

Symbol	A	B	E	R
Frequency	1	1	1	2
Probability	20%	20%	20%	40%

→ The initial ranges would respectively be:

Symbol	A	B	E	R
Ranges	[0, 0.2]	[0.2, 0.4]	[0.4, 0.6]	[0.6, 1]

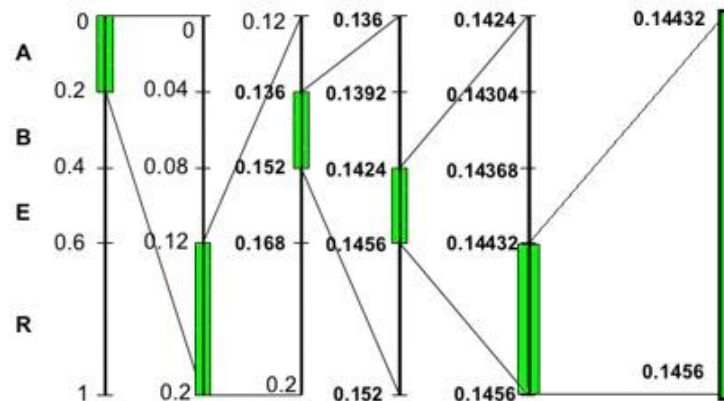
# Arithmetic Coding (Example)

**"ARBER"**

→ The initial ranges would respectively be:

Symbol	A	B	E	R
Ranges	[0, 0.2]	[0.2, 0.4]	[0.4, 0.6]	[0.6, 1]

→ The following image shows the construction of the Arithmetic Code:



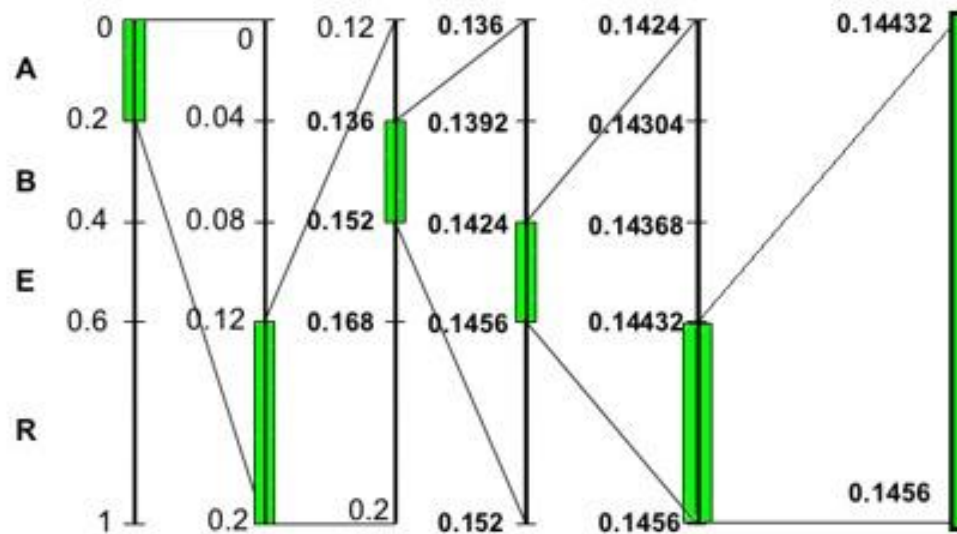
**This is the final  
interval for  
"ARBER"**



# Arithmetic Coding (Example)

→ “ARBER”

→ The following image shows the construction of the Arithmetic Code:

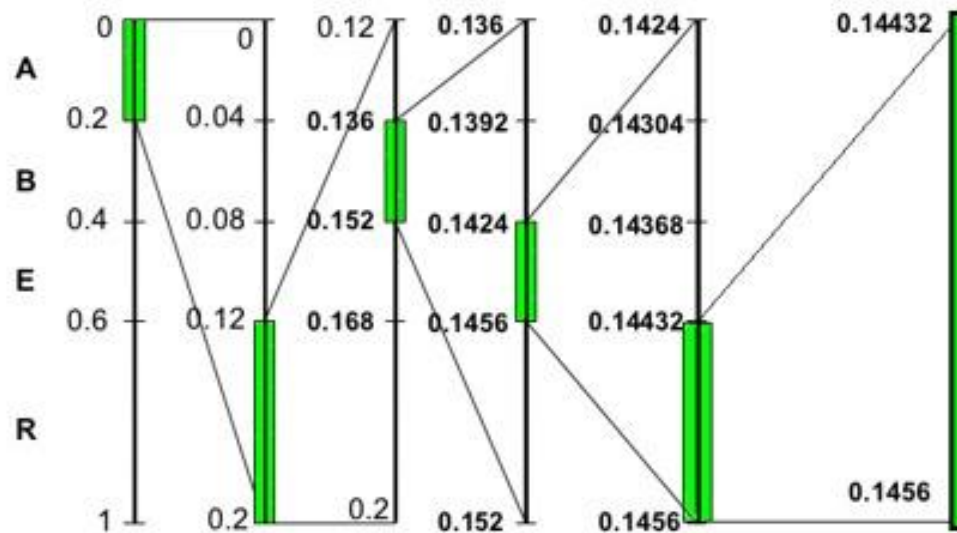


**Is the same  
process to decode**

# Arithmetic Coding (Example)

→ "ARBER"

→ The following image shows the construction of the Arithmetic Code:

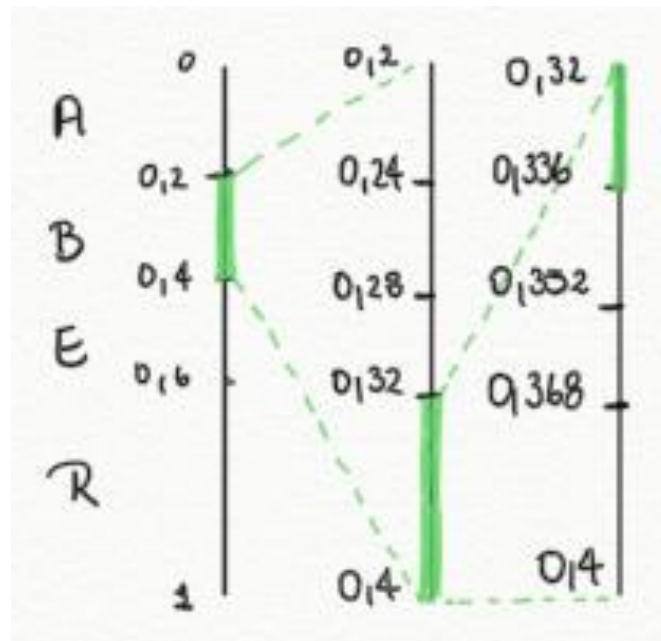


→ Decode the number 0.331

# Arithmetic Coding (Example)

→ "ARBER"

→ The following image shows the construction of the Arithmetic Code:



→ Decode the number 0.331

R/ "BRA"

# Arithmetic Coding (Example)

---

→ Considering the following information:

Symbol	A	B	C
Probability	20%	50%	30%

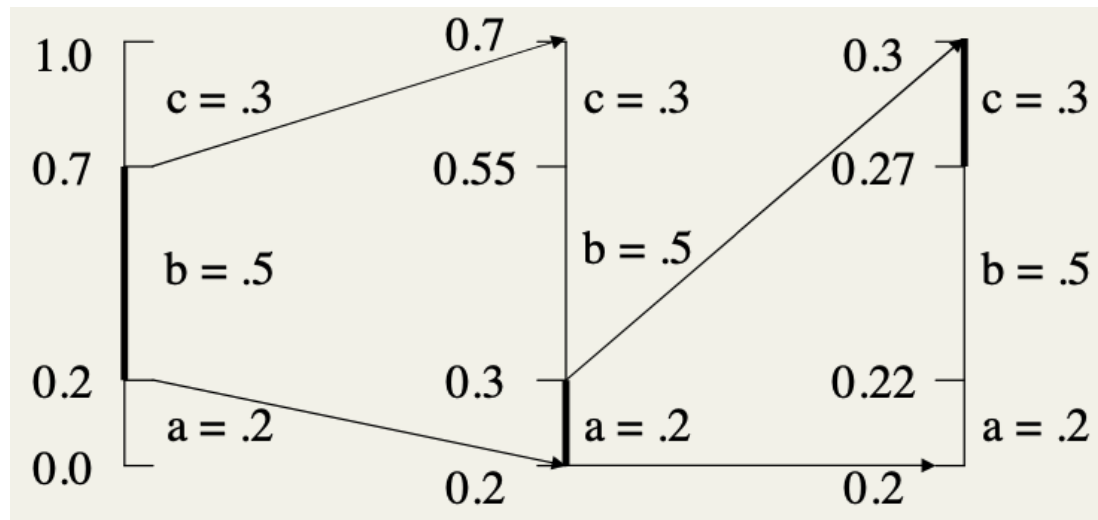
→ Encode the input “BAC”

# Arithmetic Coding (Example)

→ Considering the following information:

Symbol	A	B	C
Probability	20%	50%	30%

→ Encode the input “BAC”

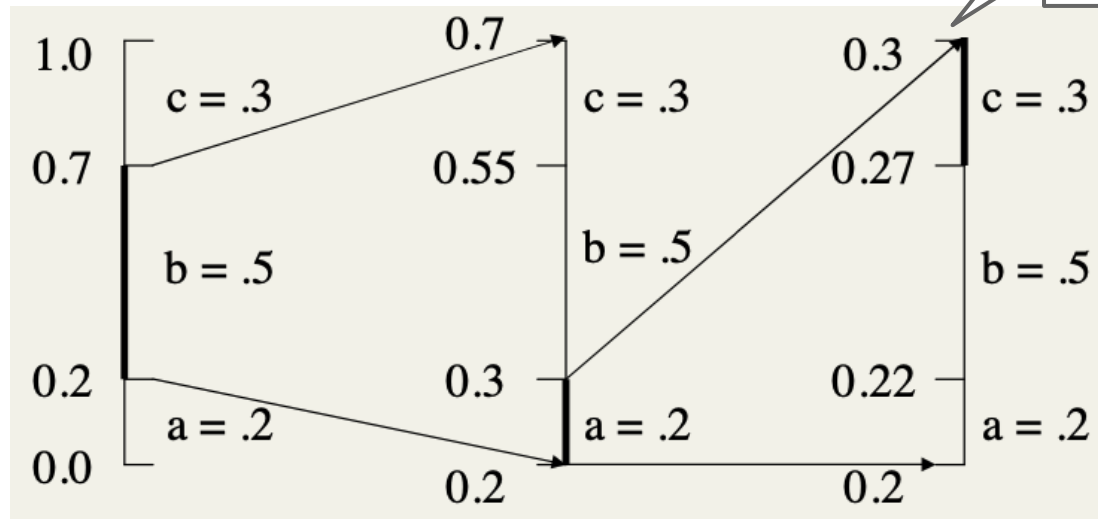


# Arithmetic Coding (Example)

→ Considering the following information:

Symbol	A	B	C
Probability	20%	50%	30%

→ Encode the input “BAC”



**Try to decode the value 0.718**

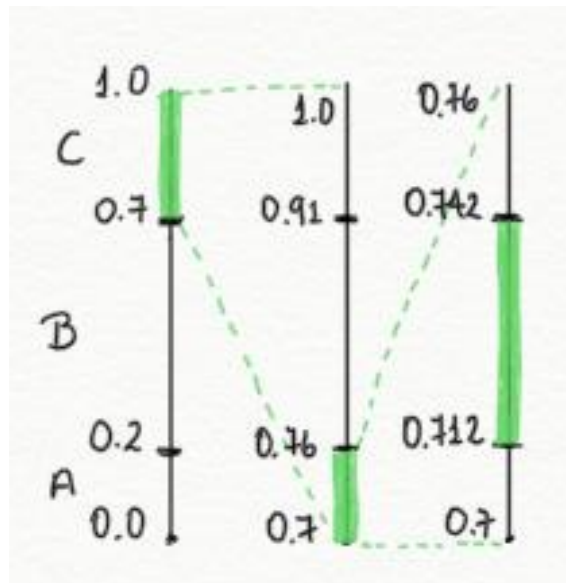
# Arithmetic Coding (Example)

→ Considering the following information:

Symbol	A	B	C
Probability	20%	50%	30%

→ Encode the input “BAC”

**Try to decode the  
value 0.718  
“CAB”**



# Arithmetic Coding (Comparison)

---

## ADVANTAGES

- **Flexibility.** It can be used in conjunction with any model that provide a sequence of event probabilities.
- **Optimality.** Is optimal in theory and very nearly optimal in practice, in the sense of encoding using minimal average code length.

## DISADVANTAGES

- **Slow.** The full precision form of Arithmetic Coding requires at least one multiplication per event and in some implementations up to two multiplications and two divisions per event.



# Compression Algorithms

CE2103 - Algorithms and Data Structures II

