Advanced Data Structures and Algorithms

Associate Professor Dr. Raed Ibraheem Hamed

University of Human Development, College of Science and Technology Computer Science Department

2015 – 2016

Introduction

What is Compression? Compression is the process of encoding data more efficiently to achieve a reduction in file size

Advantages of Compression

1) When compressed, the quantity of bits used to store the information is reduced.
2) Files that are smaller in size will result in shorter transmission times when they are transferred on the Internet.
3) Compressed files also take up less storage space.
4) File compression can zip up several small files into a single file for more convenient email transmission.

OBJECTIVES

After reading this topic, the reader should be able to:

- Realize the need for data compression.
- Differentiate between lossless and lossy compression.
- Understand three **lossless compression** encoding techniques: run-length, Huffman, and Lempel Ziv.
- Understand two **lossy compression** methods: JPEG and MPEG.

Data compression methods

Data compression means sending or storing a smaller number of bits.

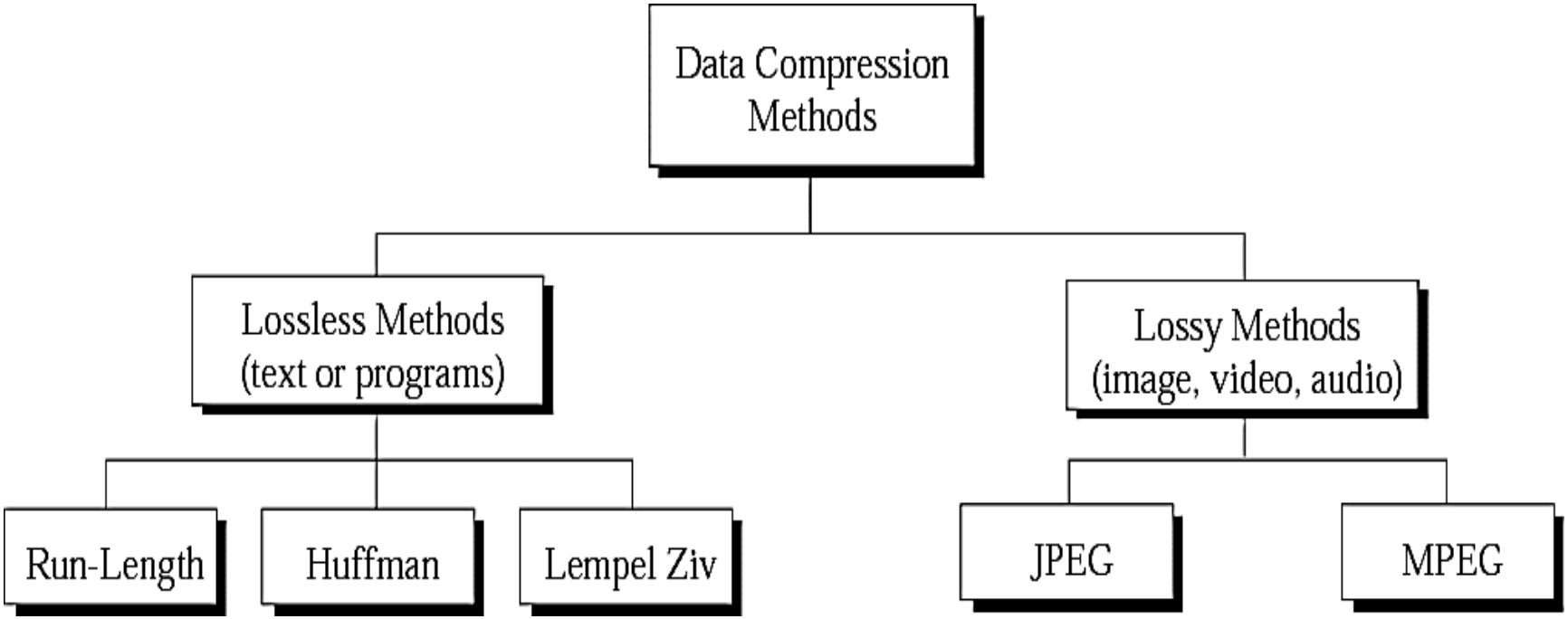


Figure 15-1

Lossless Compression Methods

Lossless compression

- In lossless data compression, the integrity of the data is preserved.
- The original data and the data after compression and decompression are exactly the same because the compression and decompression algorithms are exactly the inverse of each other.
- Example:
  - Run-length encoding
  - Huffman encoding
  - Lempel Ziv (L Z) encoding (dictionary-based encoding)

Run-length encoding

- It does **not** need knowledge of the frequency of occurrence of symbols and can be very **efficient** if data are represented as 0s and 1s.
- For example:

BBBBBBBBBAAAAAAAAAAAAAAAAANMMMMMMMMMM

a. Original Data

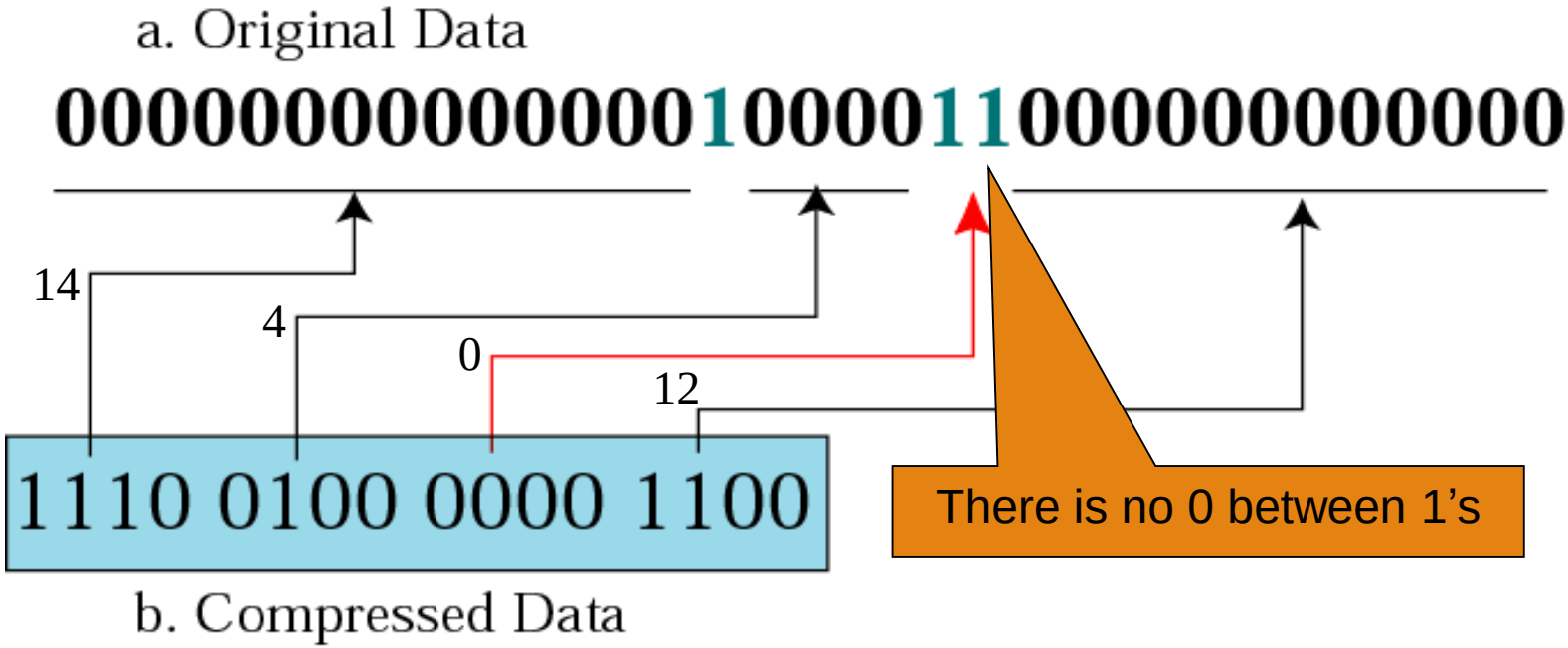
B09A16N01M10

b. Compressed Data

Run-length encoding for two symbols

We can encode one symbol which is more frequent than the other.

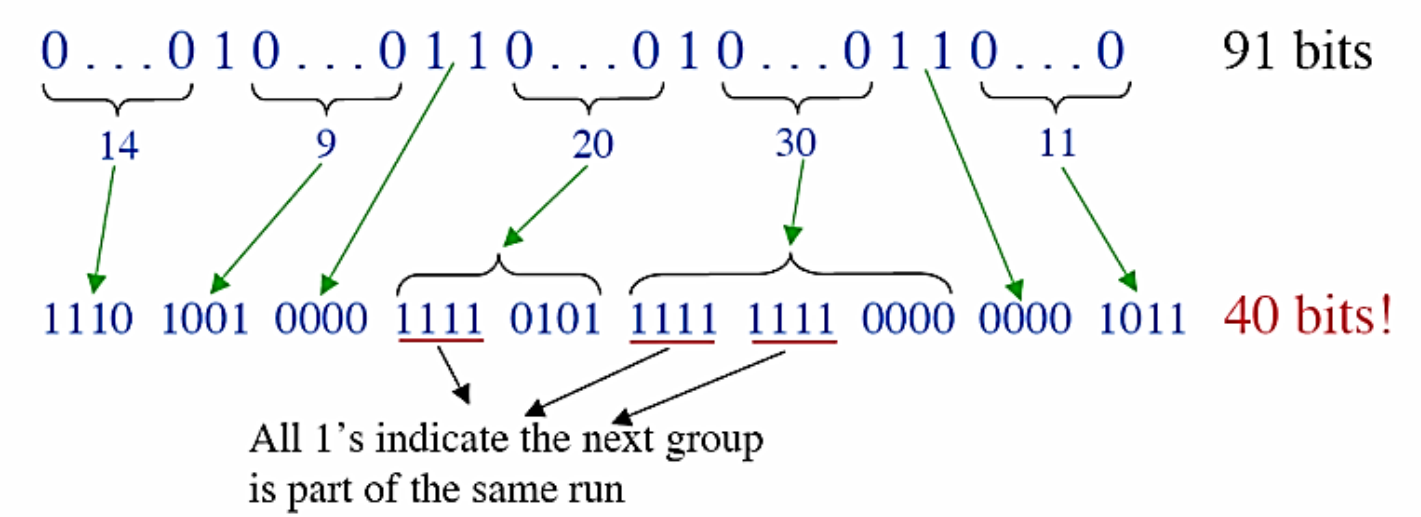
This example only encode 0's between 1's.



Binary Run-length encoding

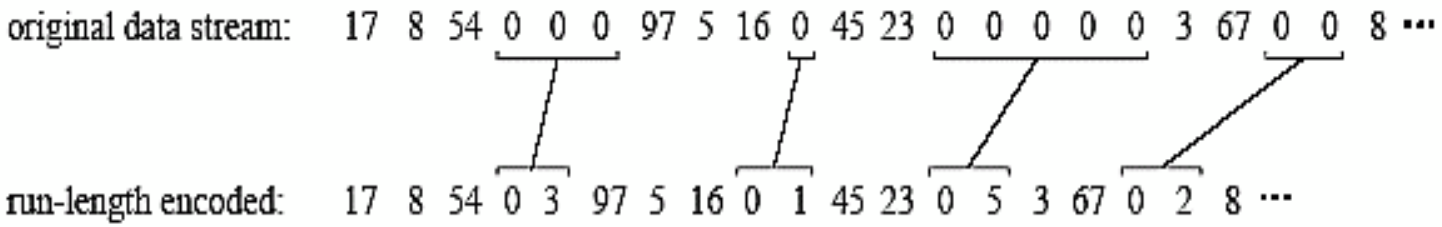
- ✓ Code the run length of 0's using k bits. Transmit the code.
- ✓ Do not transmit runs of 1's.
- ✓ Two consecutive 1's are implicitly separately by a zero-length run of zero.

Example: suppose we use **k = 4** bits to encode the run length (maximum run length of **15**) for following bit patterns.



Example: run-length encoding for a data sequence having frequent runs of *zeros*

Data files frequently contain the same character repeated many times in a row. For example, text files use multiple spaces to separate sentences, indent paragraphs, format tables & charts, etc.



Example of run-length encoding. Each run of zeros is replaced by two characters in the compressed file: a zero to indicate that compression is occurring, followed by the number of zeros in the run.

Note: **many single zeros in the data can make the encoded file larger than the original.**

Huffman coding

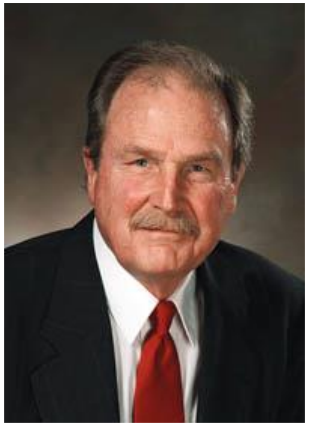
The following algorithm generates Huffman code:

- Find (or assume) the probability of each values occurrence.
- Initialization: Put all **nodes** in an list, keep it sorted at all times (e.g., ABCDE).
- Take the two symbols with the lowest probability, and place them as leaves on a binary tree.
- Form a new row in the table replacing the these two symbols with a new symbol. This new symbol forms a branch node in the tree. Draw it in the tree with branches to its leaf (component) symbols
- Assign the new symbol a probability equal to the sum of the component symbol's probability.

Huffman coding

- Repeat the above until there is only one symbol left. This is the root of the tree.
- Nominally assign 1's to the right hand branches and 0's to the left hand branches at each node.
- Read the code for each symbol from the root of the tree.

David Huffman



Huffman coding

- In Huffman coding, you assign shorter codes to symbols that occur more frequently and longer codes to those that occur less frequently.
- The process of building the tree begins by counting the occurrences of each symbol in the text to be encoded.
- **For example:**

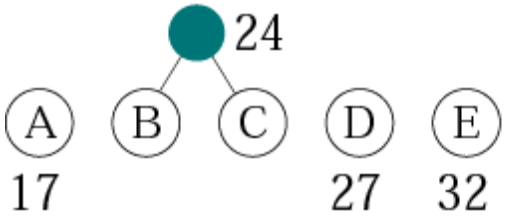
Character	***A***	***B***	***C***	***D***	***E***
Frequency	**17**	**12**	**12**	**27**	**32**

Table 15.1 Frequency of characters

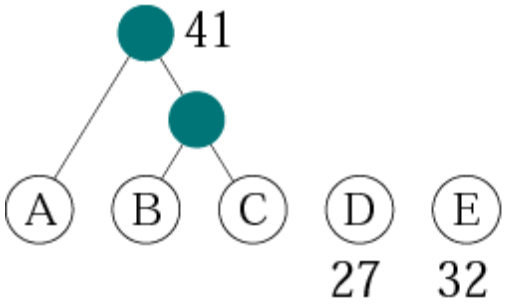
Figure 15-4

Huffman coding

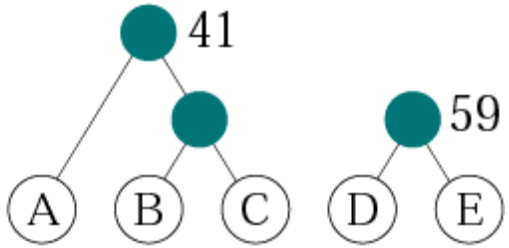
a.



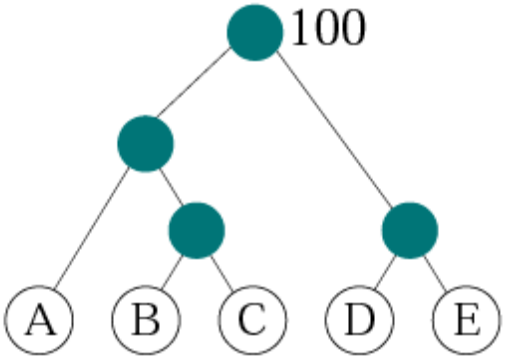
b.



c.



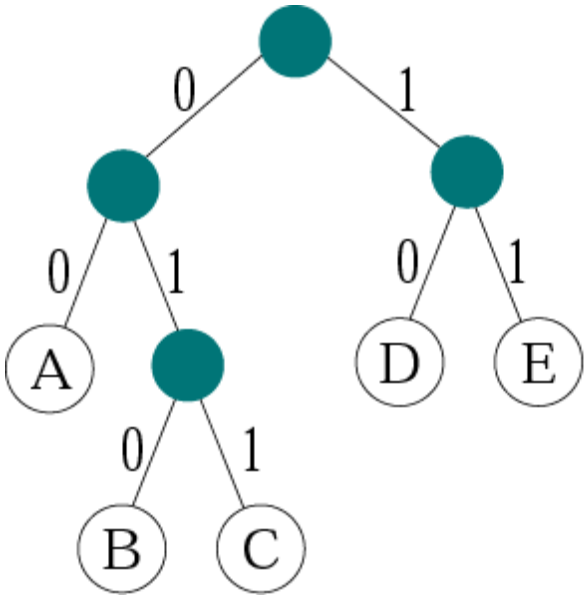
d.



e.

Figure 15-5

Final tree and code



A: 00 D: 10
B: 010 E: 11
C: 011

Code

Figure 15-6

Huffman encoding

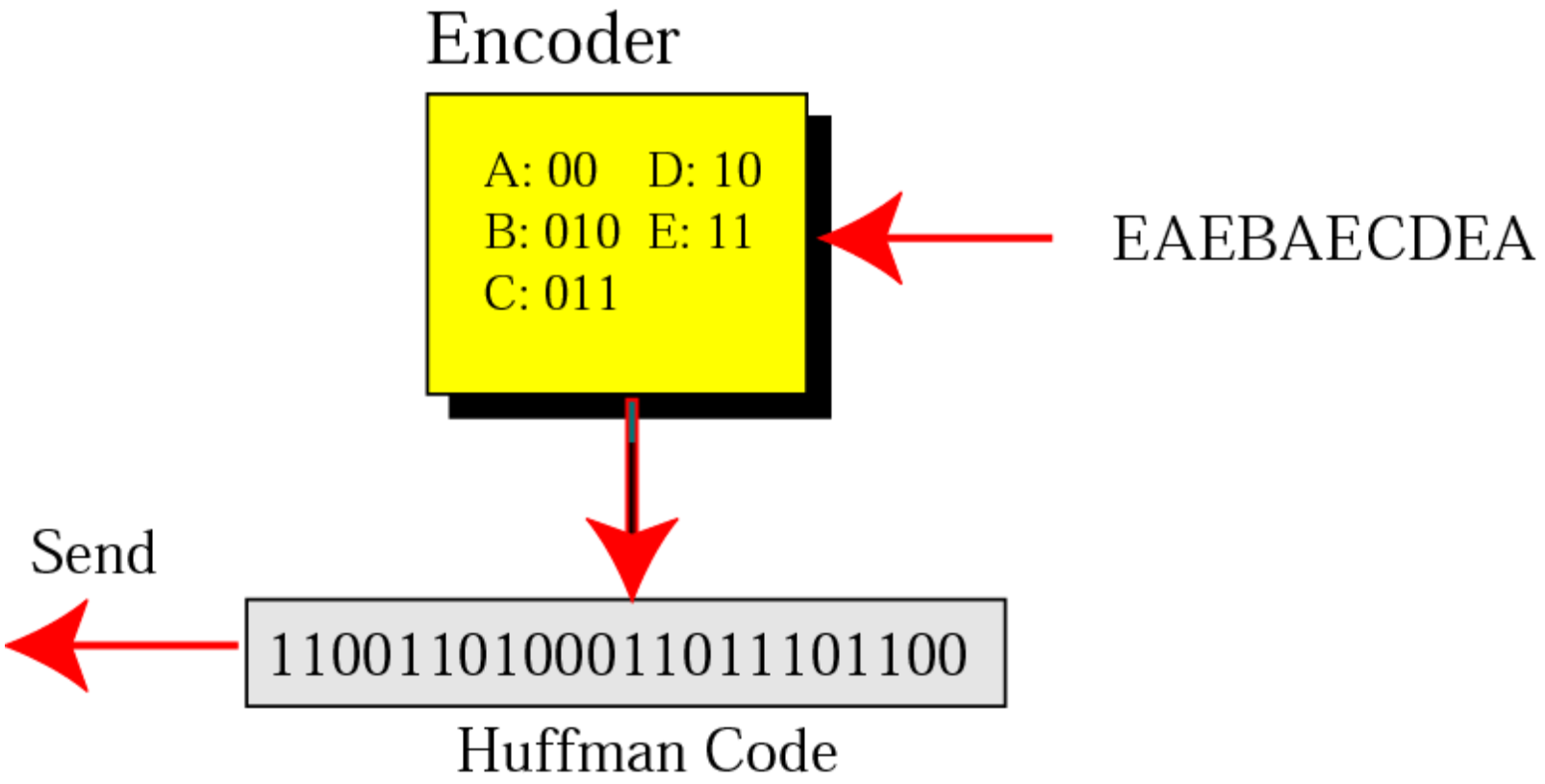
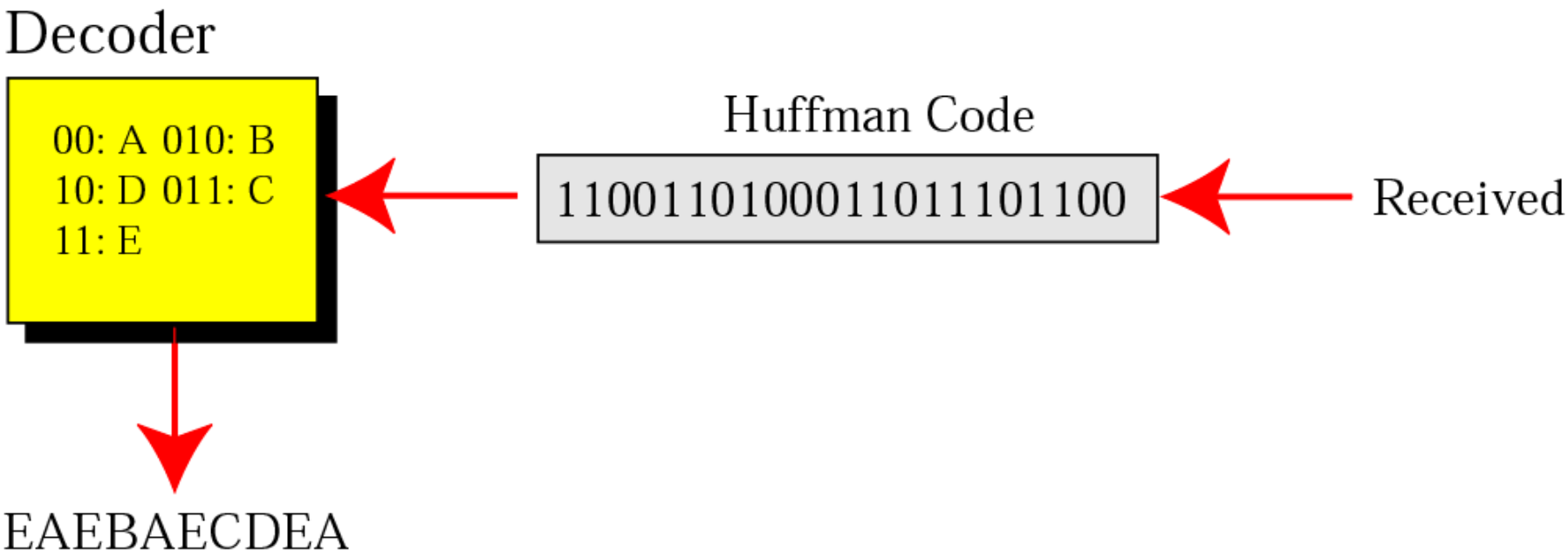


Figure 15-7

Huffman decoding



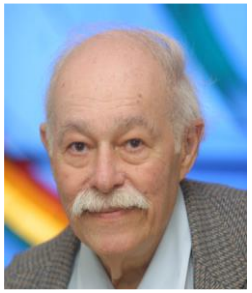
Huffman coding

- The beauty of Huffman coding is that **no code in the prefix of another code.**
- There is **no ambiguity** in encoding.
- The receiver can decode the received data **without ambiguity.**
- Huffman code is called **instantaneous (immediate) code** because the decoder can unambiguously decode the bits instantaneously with the minimum number of bits.

Lempel Ziv encoding

- LZ encoding is an example of a category of algorithms called **dictionary-based** encoding.
- The **idea** is to create a dictionary (**table**) of strings used during the communication session.
- The compression algorithm extracts the **smallest substring** that **cannot be found in the dictionary** from the remaining non-compressed string.

Abraham Lempel Jacob Ziv



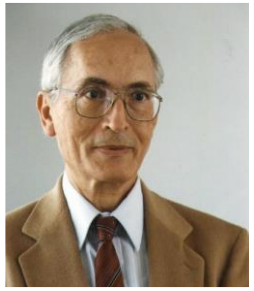


Figure 15-8:Part I

Example of Lempel Ziv encoding

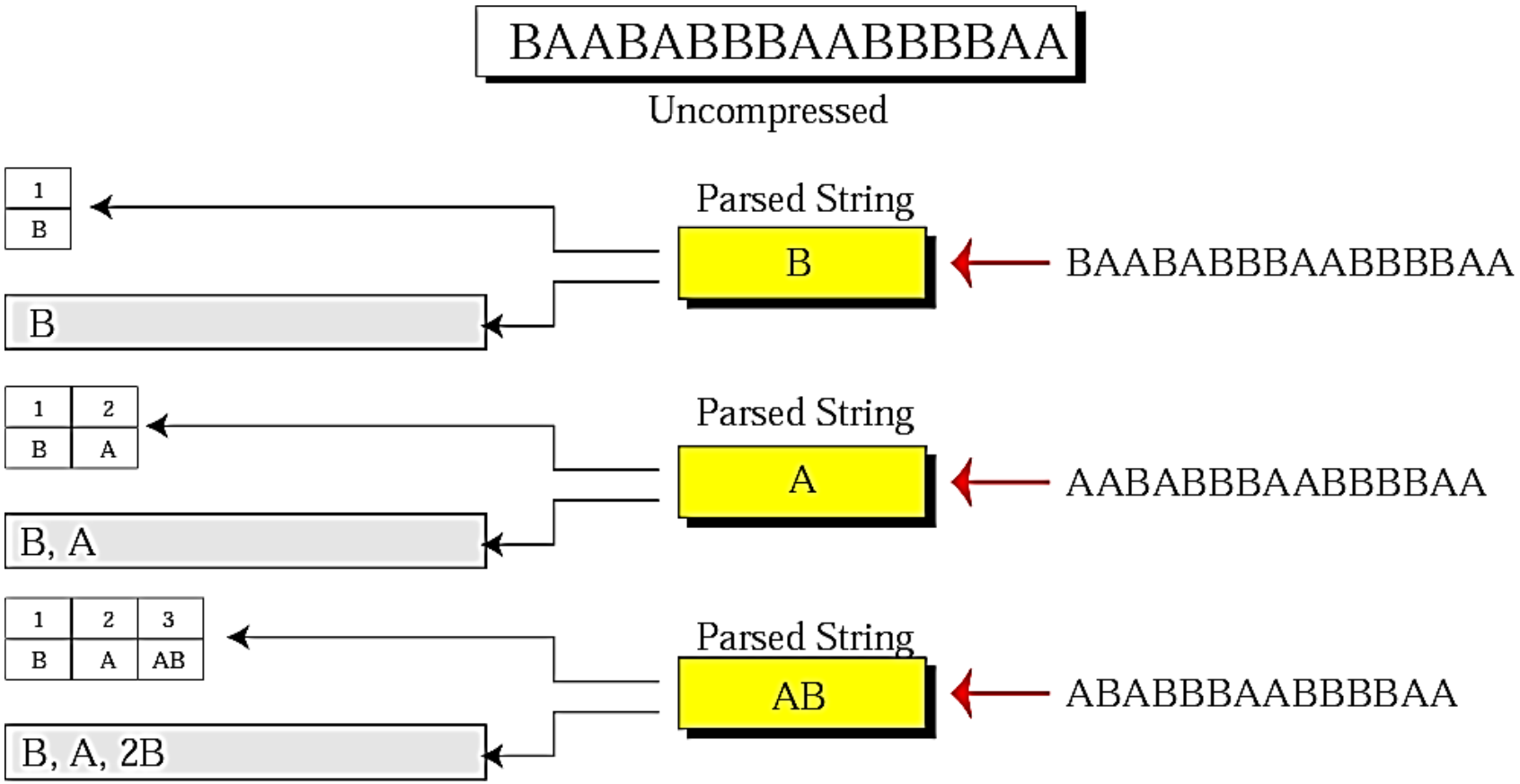


Figure 15-8:Part 2

Example of Lempel Ziv encoding

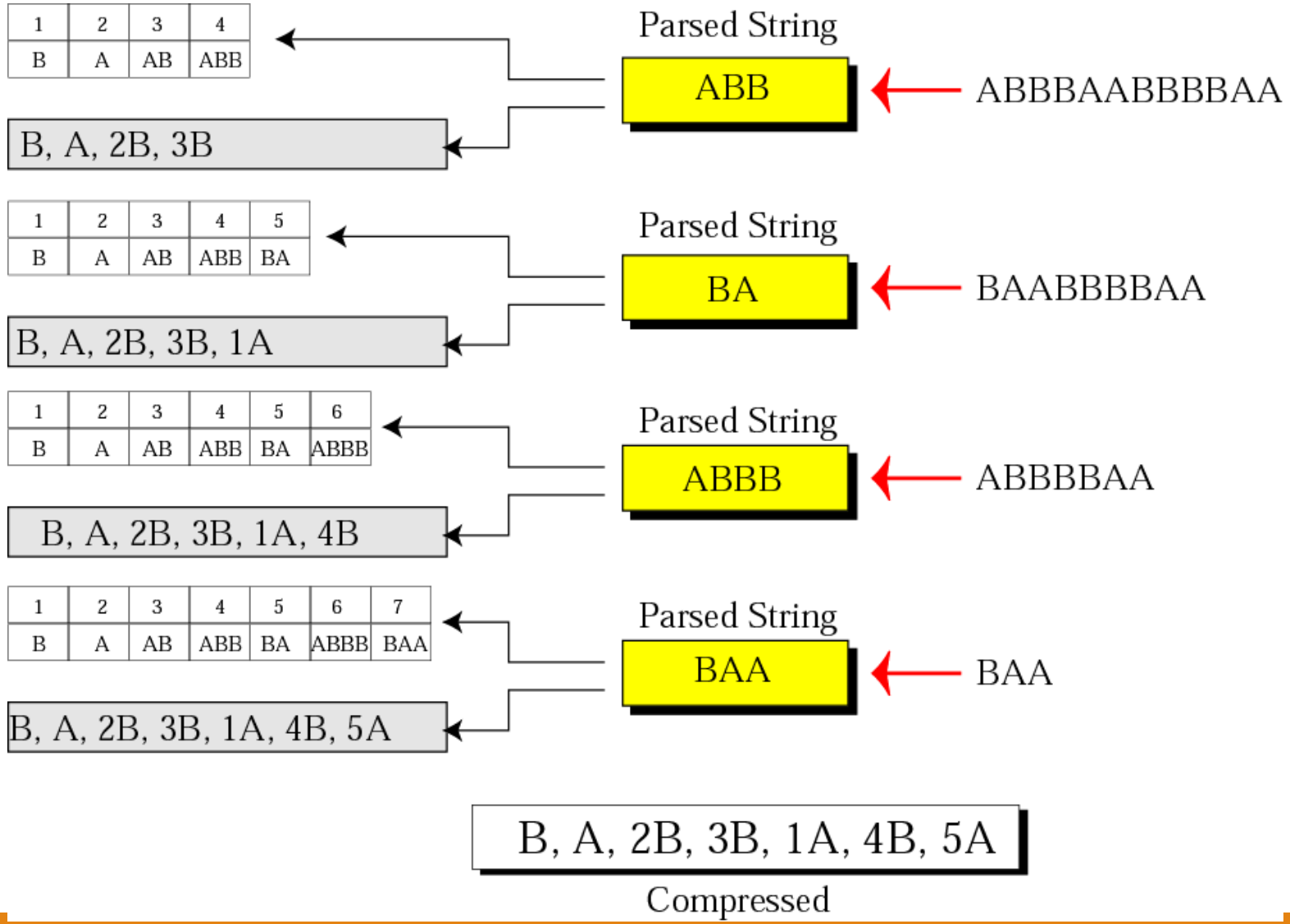


Figure 15-9: Part I

Example of Lempel Ziv decoding

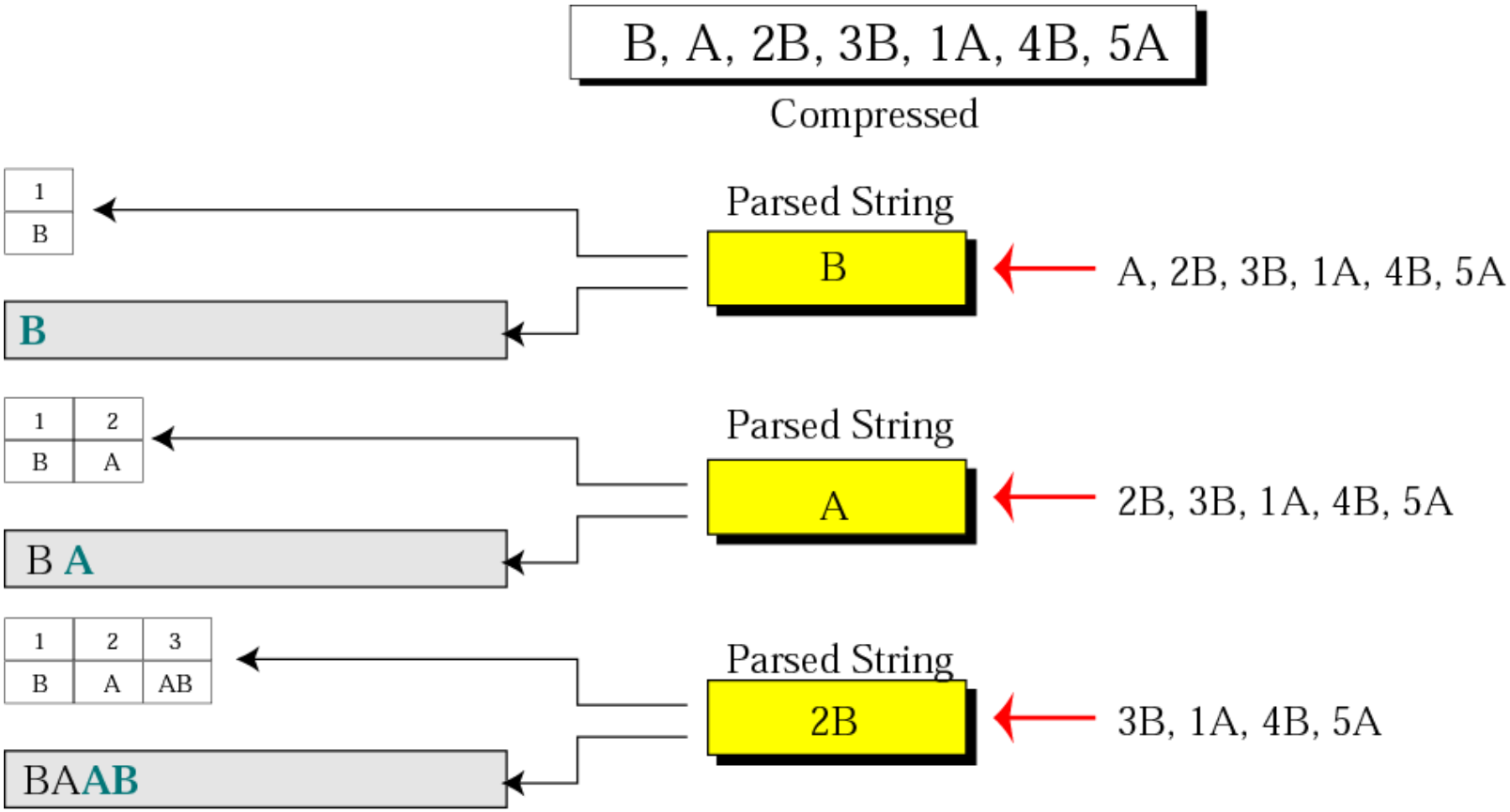
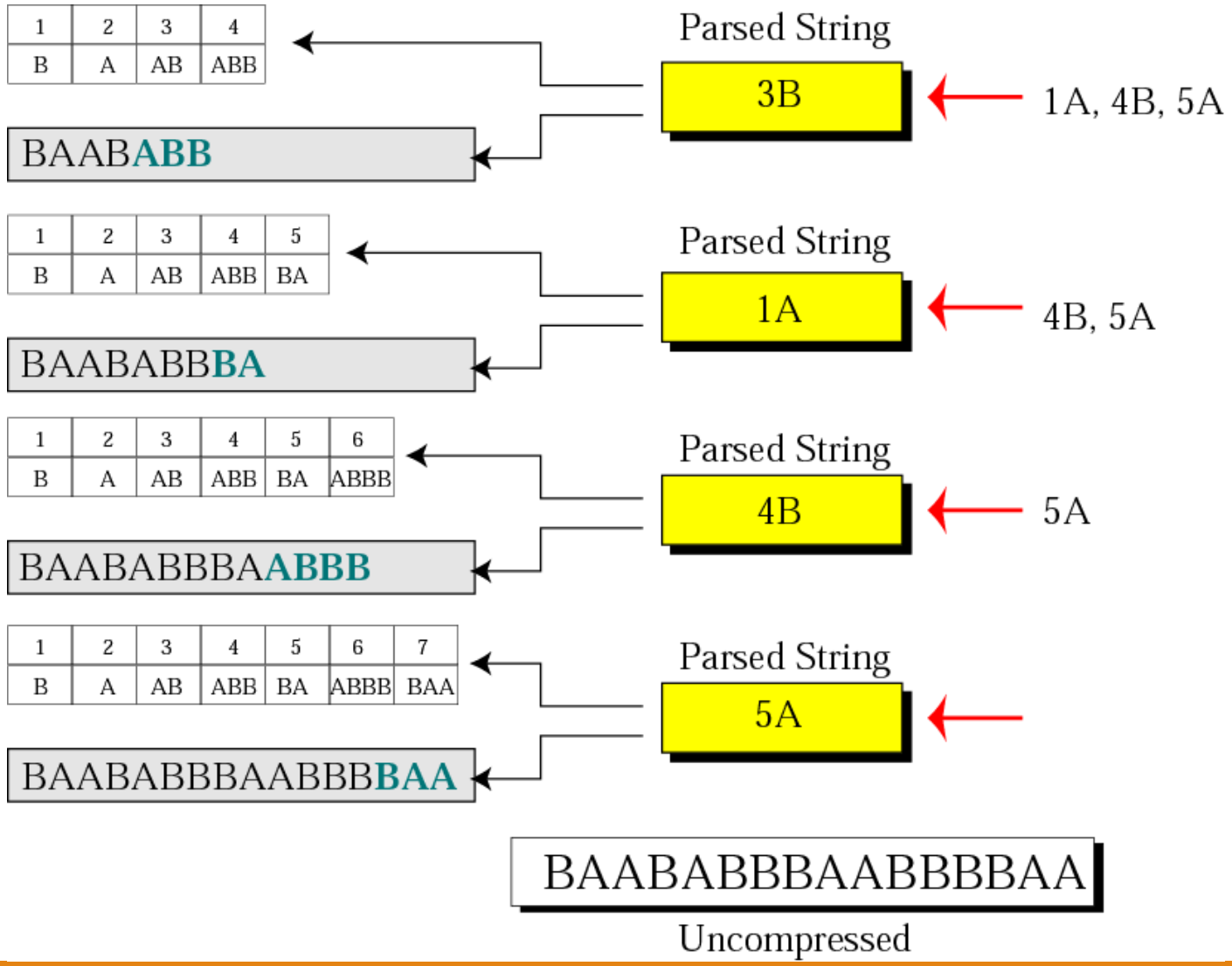


Figure 15-9: Part II

Example of Lempel Ziv decoding




Thank
you