COMP6341 - Multimedia and Human Computer Interaction

Compression - Lossless

Week 6 - Session 1

Raymond Bahana
rbahana@binus.edu

Session Learning Outcomes

Upon completion of this session, students are expected to be able to

- LO 6 - Distinguish the different compression principles, techniques and multimedia compression standards

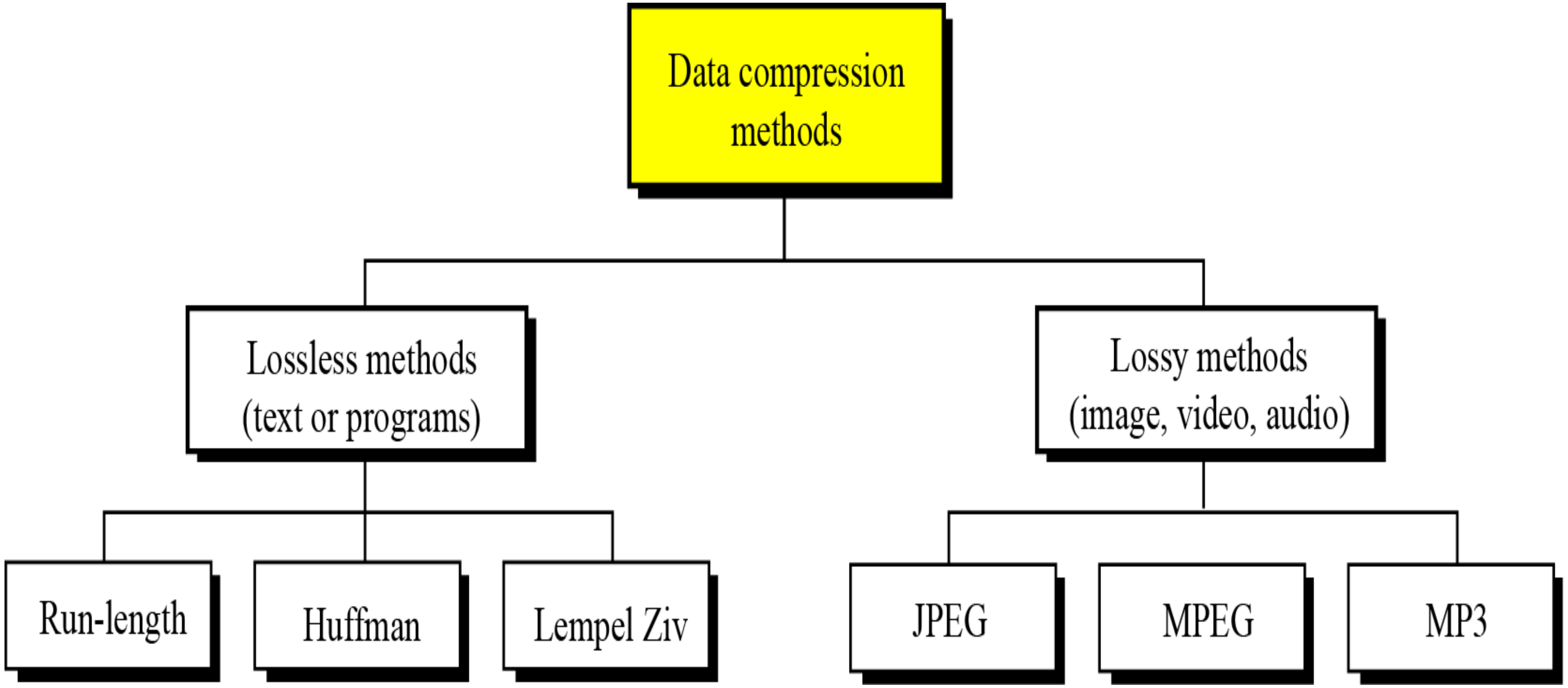
Objectives

- Describe lossless compression.
- Describe run-length encoding and how it achieves compression.
- Describe Huffman coding and how it achieves compression.
- Describe Lempel Ziv encoding and the role of the dictionary in encoding and decoding.

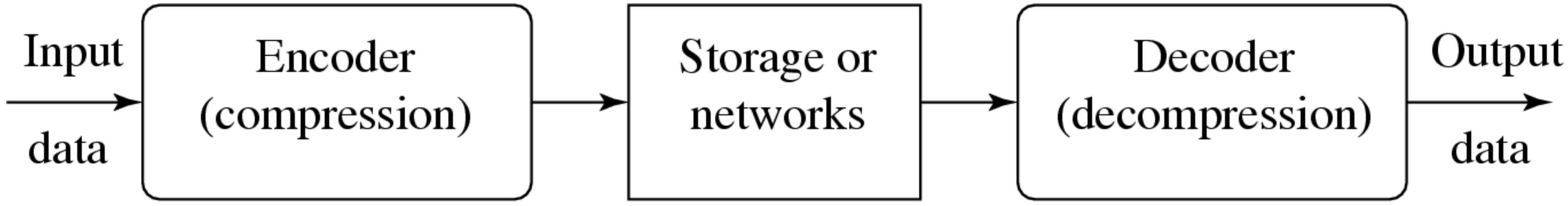
Data Compression

- Data compression implies sending or storing a smaller number of bits.
- Although many methods are used for this purpose, in general these methods can be divided into two broad categories:
  - Lossless method
  - Lossy method

Data Compression



Compression Scheme



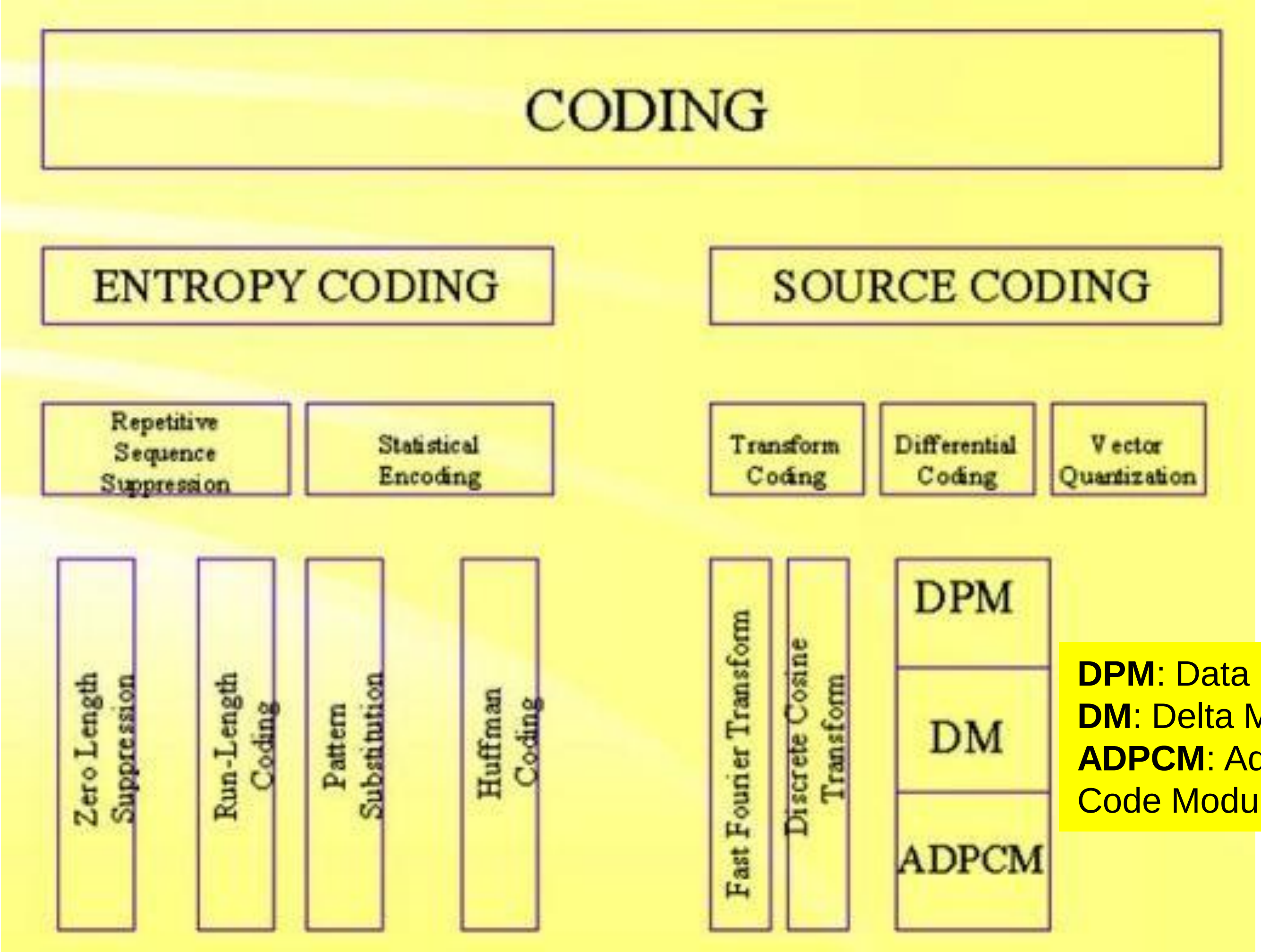
Taxonomy of Compression Techniques

- Based on the lossless and lossy compresion, the encoding is broadly classified as:
  - Entropy Encoding (leading to lossless compression)

    To compress digital data by representing frequently occurring patterns with few bits and rarely occurring patterns with many bits.
  - Source Encoding (leading to lossy compression)

    Encoding done at the source of the data before it is stored or transmitted

Bird's Eye View of Coding Techniques



DPM: Data Protection Manager
DM: Delta Modulation
ADPCM: Adaptive Differential Pulse Code Modulation

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, in these methods, the compression and decompression algorithms are exact inverses of each other: no part of the data is lost in the process.

Lossless Compression

- Redundant data is removed in compression and added during decompression.
- Lossless compression methods are normally used when we cannot afford to lose any data.

Lossless Compression Algorithms

- Repetitive Sequence Suppression
- Run-Length Encoding (RLE)
- Pattern Substitution
- Entropy Encoding
  - Shannon-Fano Algorithm
  - Huffman Coding
  - Arithmetic Coding
- Lempel-Ziv-Welch (LZW) Algorithm

Repetitive Sequence Suppression

- Fairly straight forward to understand and implement.
- Simplicity is their downfall: NOT best compression ratios.
- Some methods have their applications, e.g. Component of JPEG, Silence Suppression.

Simple Repetition Suppression

- If a sequence a series on, **n** successive tokens appears
- Replace series with a token and a count number of occurrences.
- Usually need to have a special ***flag*** to denote when the repeated token appears

Simple Repetition Suppression

- Example:

  89400000000000000000000000000000000

  we can replace with:

  894f32

  where f is the ***flag*** for zero.

Simple Repetition Suppression

- Compression savings depend on the content of the data.
- Applications of this simple compression technique include:
  - Suppression of zero's in a file (Zero Length Suppression)
    - Silence in audio data, pauses in conversation etc.
    - Blanks in text or program source files
    - Backgrounds in simple images
  - Other regular image or data tokens

Run-Length Encoding

- Run-length encoding is probably the simplest method of compression.
- It can be used to compress data made of any combination of symbols.
- It does not need to know the frequency of occurrence of symbols and can be very efficient if data is represented as 0s and 1s.

Run-Length Encoding

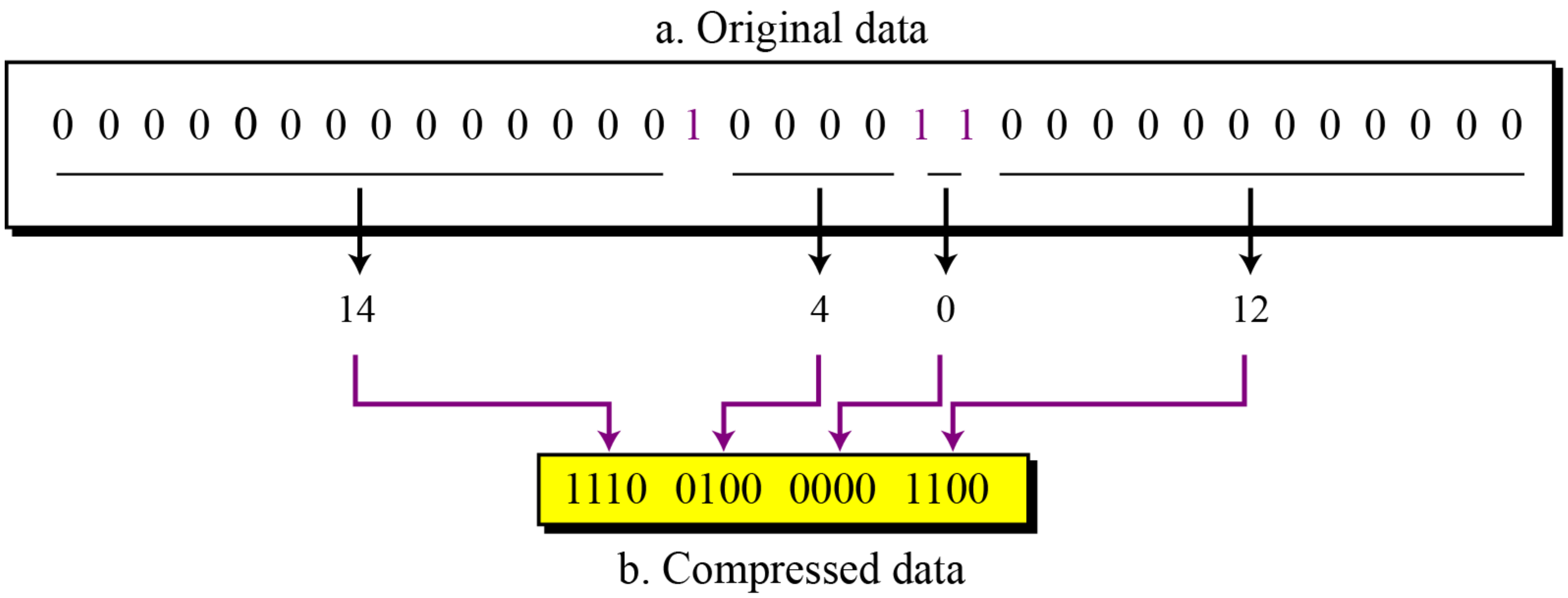
- The general idea behind this method is to replace consecutive repeating occurrences of a symbol by one occurrence of the symbol followed by the number of occurrences.
- The method can be even more efficient if the data uses only two symbols (for example 0 and 1) in its bit pattern and one symbol is more frequent than the other.

Run-Length Encoding

a. Original data BBBBBBBBBAAAAAAAAAAAAAAAANMMMMMMMMMM

b. Compressed data B09A16N01M10

Run-Length Encoding



Pattern Substitution

- This is a simple form of statistical encoding.
- Here we substitute a frequently repeating pattern(s) with a code.
- The code is shorter than the pattern giving us compression.
- A simple Pattern Substitution scheme could employ predefined codes.

Pattern Substitution

- For example replace all occurrences of pattern of characters ‘and’ with the predefined code ’&’.
- So:

  and you and I

- Becomes:

  & you & I

- Similar for other codes— commonly used words

Pattern Substitution

- Example:

  This book is an exemplary example of a book on multimedia and networking. Nowhere else will you find this kind of coverage and completeness. This is truly a one-stop-shop for all that you want to know about multimedia and networking.

- If we simply count, there are a total of 193 characters without counting blanks and 232 with blanks (white space).

Pattern Substitution

- If we group words such as **a**, **about**, **all**, **an**, **and**, **for**, **is**, **of**, **on**, **that**, **this**, **to**, and **will**, they occur 2, 1, 1, 1, 3, 1, 2, 2, 1, 1, 3,1, and 1 times, respectively.
- All of them have a blank character on either side, unless when they happen to be the first word or last word of a sentence.
- The sentence delimiter **period** is always followed by a blank character.

Pattern Substitution

- The words multimedia and networking appear twice each.
- Let us represent the group of words that we identified for the text under consideration by 1, 2, 3, 4, 5, 6, 7, 8, 9, +, &, =, and #.
- Let us also substitute **multimedia** by m* and **networking** by n*.
- The resulting coded string will be:

Pattern Substitution

- The resulting coded string will be:

```
&  b  o o k 7  4 e x e m p l  a r y
sp e  x a m p  l e 8 1 b o o  k 9 m
*  5  n * . N  o w h e r e sp e l s
e  #  y o u sp f i n d & k i  n d 8
c  o  v e r a  g e 5 c o m p  l e t
e  n  e s s .  & 7 t r u l y  1 o n
e  -  s t o p  - s h o p 6 3  + y o
u  sp w a n t  = k n o w 2 m  * 5 n
*  .
```

- That is a total of 129 characters and 33.16% compression.

Shannon-Fano Algorithm

- A top-down approach
- Sort the symbols according to the frequency count of their occurrences.
- Recursively divide the symbols into two parts, each with approximately the same number of counts, until all parts contain only one symbol.

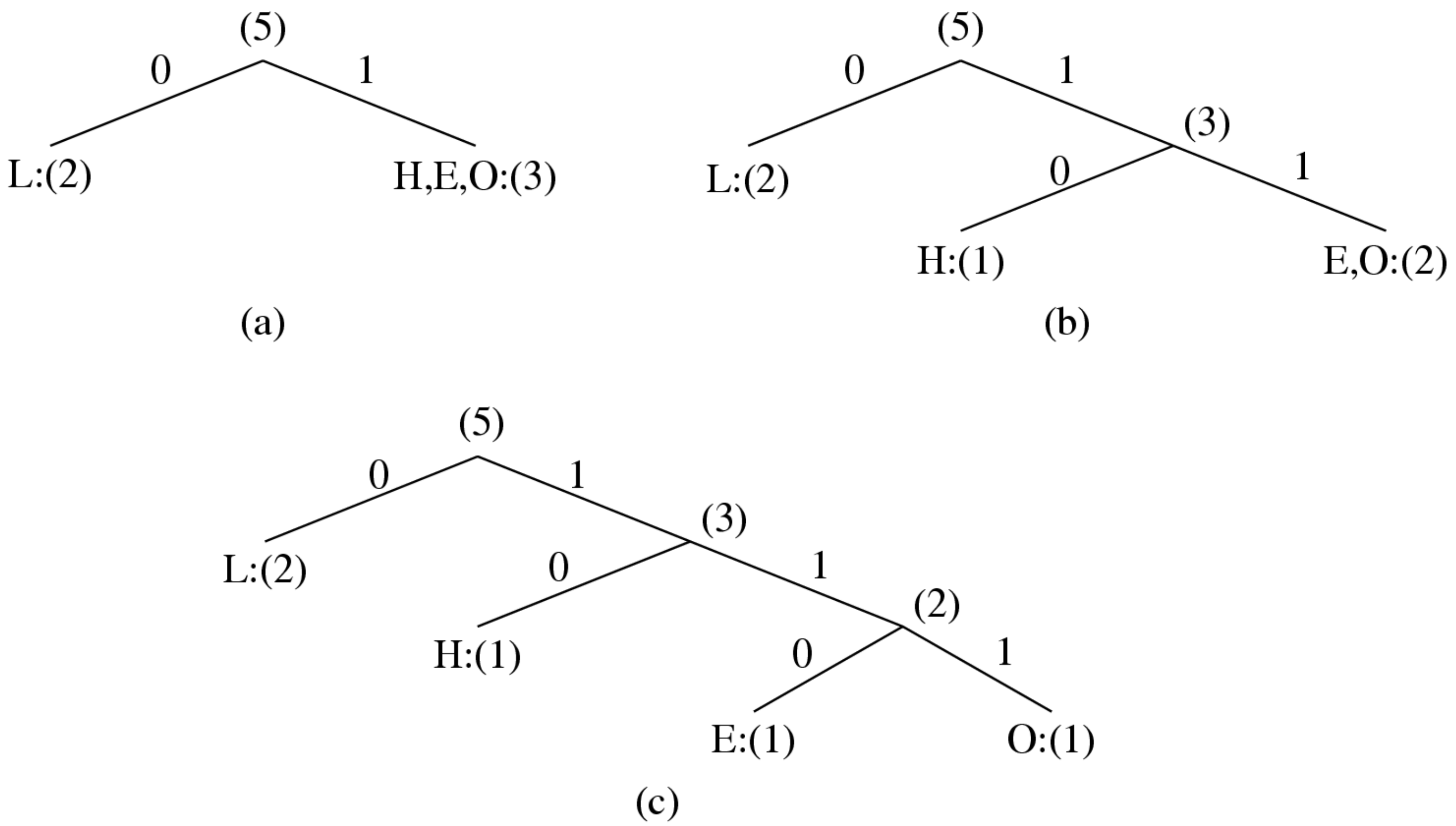
Shannon-Fano Algorithm

- 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



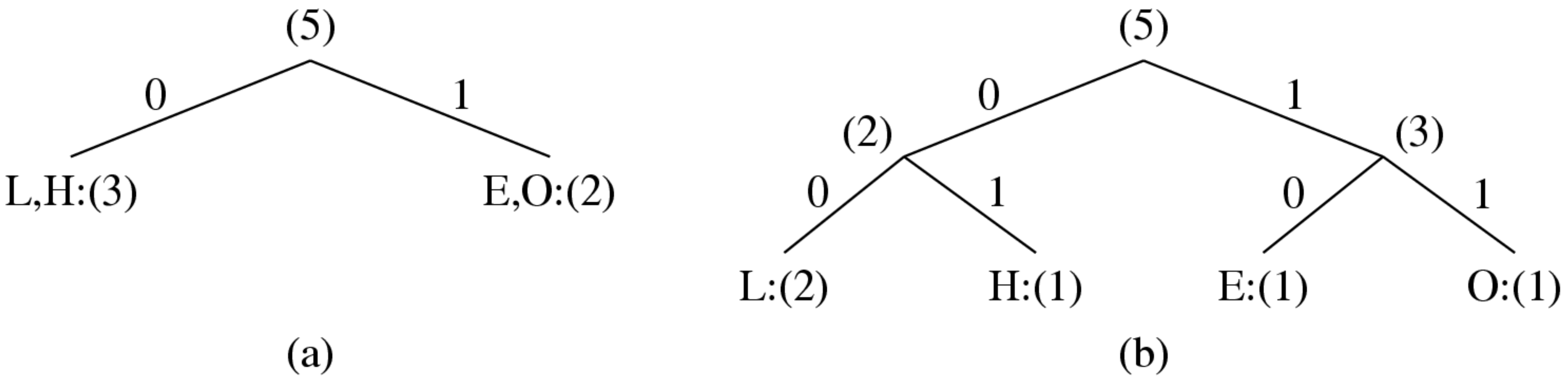
Coding Tree for HELLO by Shannon-Fano.

Shannon-Fano Algorithm

❖ Result of Performing Shannon-Fano on HELLO

Symbol	Count	Code	# of bits used
L	2	0	2 (1 bit x2)
H	1	10	2 (2 bits x1)
E	1	110	3 (3 bits x1)
O	1	111	3 (3 bits x1)
		TOTAL # of bits:	10

Shannon-Fano Algorithm



Another coding tree for HELLO by Shannon-Fano.

Shannon-Fano Algorithm

- Another Result of Performing Shannon-Fano on HELLO

Symbol	Count	Code	# of bits used
L	2	00	4 (2 bits x2)
H	1	01	2 (2 bits x1)
E	1	10	2 (2 bits x1)
O	1	11	2 (2 bits x1)
		TOTAL # of bits:	10

Huffman Coding

- Huffman coding <u>assigns shorter codes to symbols </u>that <u>occur more frequently</u> and <u>longer codes to </u>those that <u>occur less frequently</u>.
- A bottom-up approach
- Example, imagine we have a text file that uses only five characters (A, B, C, D, E).
- Before we can assign bit patterns to each character, we assign each character a weight based on its frequency of use.

Huffman Coding

- In this example, assume that the frequency of the characters

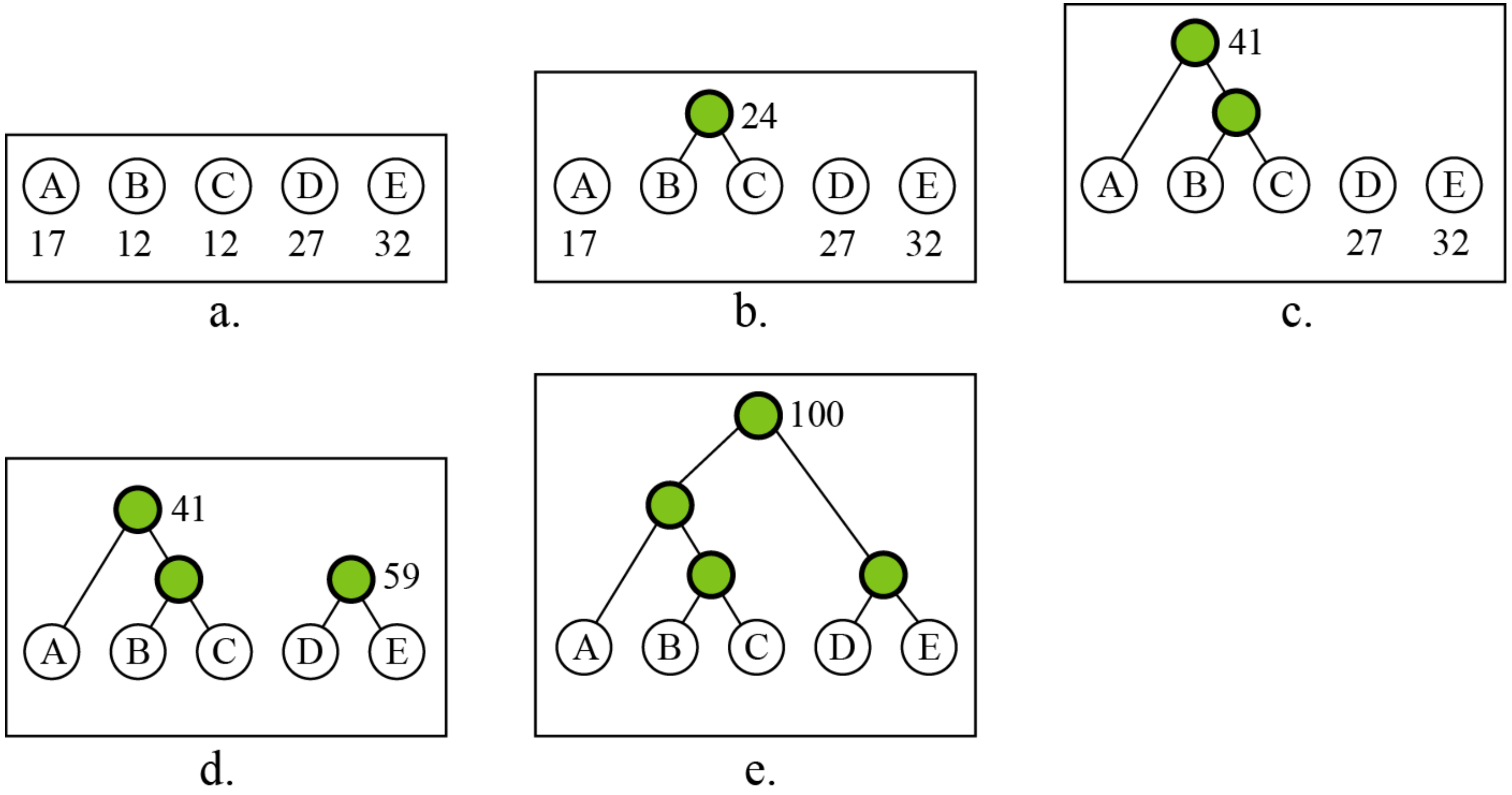
Frequency of characters

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

Huffman Coding

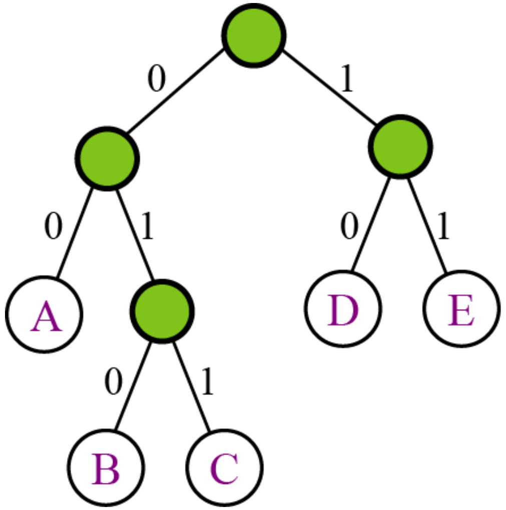
- Put all symbols on a list sorted according to their frequency counts (optional)
- Repeat until the list has only one symbol left:
  - From the list pick two symbols with the lowest frequency counts.
  - Form a Huffman sub-tree that has these two symbols as child nodes and create a parent node
  - Assign the sum of the children's frequency counts to the parent and insert it into the list such that the order is maintained

Huffman Coding



Huffman Coding

- A character’s code is found by starting at the root and following the branches that lead to that character.
- The code itself is the bit value of each branch on the path, taken in sequence.

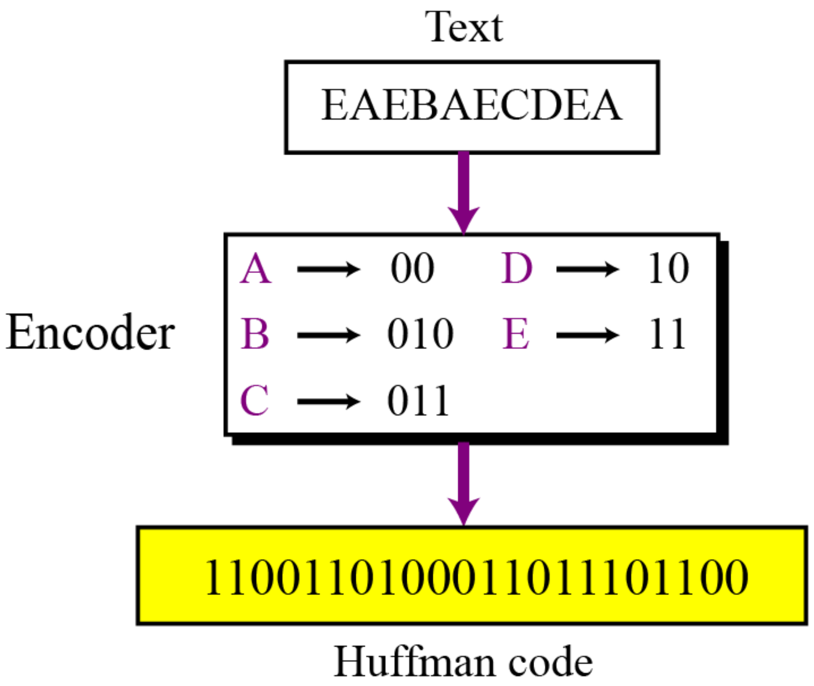


A: 00	D: 10
B: 010	E: 11
C: 011	

Code

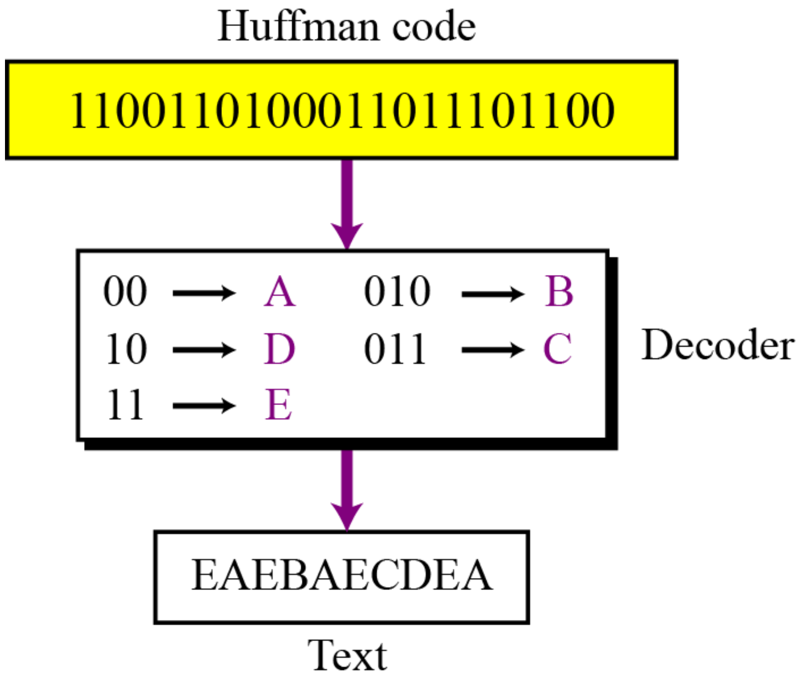
Huffman Coding - Encoding

❖ Let see how to encode text using the code for those five characters.



Huffman Coding - Decoding

❖ The recipient has a very easy job in decoding the data it receives.



Arithmetic Coding

- A widely used entropy coder
- Also used in JPEG
- Only problem is it’s speed due possibly complex computations due to large symbol tables,
- Good compression ratio (better than Huffman coding), entropy around the Shannon Ideal value.

Arithmetic Coding

- Why better than Huffman?
  - Huffman coding etc. use an integer number (k) of bits for each symbol
    - hence k is never less than 1.

Lempel-Ziv Algorithm

- Lempel Ziv (LZ) encoding is an example of a category of algorithms called dictionary-based encoding.
- The idea is to create a dictionary (a table) of strings used during the communication session.
- If both the sender and the receiver have a copy of the dictionary, then previously-encountered strings can be substituted by their index in the dictionary to reduce the amount of information transmitted.

Lempel-Ziv Algorithm

- There are many variations of Lempel Ziv around, but they all follow the same basic idea.
- The idea is to parse the sequence into distinct phrases. The version we
- For example, we have the string

AABABBBABAABABBBABBABB

Lempel-Ziv Algorithm

- We start with the shortest phrase on the left that we haven't seen before.
- This will always be a single letter, in this case A:

  A|ABABBBABAABABBBABBABB

- We now take the next phrase we haven't seen. We've already seen A, so we take AB:

  A|AB|ABBBABAABABBBABBABB

Lempel-Ziv Algorithm

- The next phrase we haven't seen is ABB, as we've already seen AB. Continuing, we get B after that:

  A|AB|ABB|B|ABAABABBBABBABB

- and you can check that the rest of the string parses into

  A|AB|ABB|B|ABA|ABAB|BB|ABBA|BB

- Because we've run out of letters, the last phrase on the end is a repeated one. That's O.K.

Lempel-Ziv Algorithm

- For each phrase we see, we stick it in a dictionary.
- The next time we want to send it, we don't send the entire phrase, but just the number of this phrase.
- Consider the following table

1	2	3	4	5	6	7	8	9
A	AB	ABB	B	ABA	ABAB	BB	ABBA	BB
0A	1B	2B	0B	2A	5B	4B	3A	7

- The 2nd row gives the phrases, and the 3rd row their encodings.

Lempel-Ziv Algorithm

- That is, when we're encoding the ABAB from the sixth phrase, we encode it as 5B.
- This maps to ABAB since the fifth phrase was ABA, and we add B to it.
- Here, the empty set 0 should be considered as the 0'th phrase and encoded by 0.
- Last piece into binary might give

  0;0|1;1|10;1|0;1|10;0|101;1|100;1|011;0|0111

Lempel-Ziv-Welch (LZW) Algorithm

- A very common compression technique.
- Used in GIF files (LZW), Adobe PDF file (LZW), UNIX compress (LZ Only)
- Patented — LZW not LZ.

Lempel-Ziv-Welch (LZW) Algorithm

- LZW Constructs Its Own Dictionary
- Problems:
  - Too many bits per word,
  - Everyone needs a dictionary,
  - Only works for English text.

Lempel-Ziv-Welch (LZW) Algorithm

- Solution:
  - Find a way to build the dictionary adaptively.
  - Terry Welch improvement (1984), Patented LZW Algorithm
    - LZW introduced the idea that only the initial dictionary needs to be transmitted to enable decoding:

      The decoder is able to build the rest of the table from the encoded sequence.

Activities

1. Create Huffman trees and codes for the following sets of letters with the given probabilities:

   a. A (0.20), B (0.09), C (0.15), D (0.11), E (0.40), F (0.05)

   b. A (0.05), C (0.04), E (0.16), G (0.02), I (0.04), L (0.07), M (0.09), N (0.08), O (0.12), R (0.08), S (0.09), T (0.10), U (0.04), Y (0.02)

Activities

2. The following character counts were obtained from a document:

D (894),	E (3320),
F (698),	M (661),
O (1749),	R (1600)

Create a Huffman tree and code for these characters and encode the word 'freedom'.