Course Code 005636 (Fall 2017)

Multimedia

Multimedia Data Compression (Lossless Compression Algorithms)

Prof. S. M. Riazul Islam, Dept. of Computer Engineering, Sejong University, Korea

E-mail: riaz@sejong.ac.kr

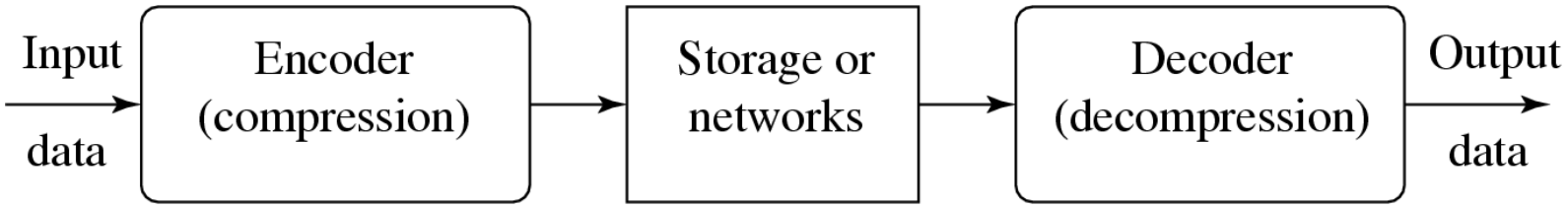
Outline

- Introduction to Data Compression
- Basics of Information Theory
- Run-Length Coding (RLC)
- Variable-Length Coding (VLC)
- Dictionary-Based Compression

Introduction

- **Compression**: the process of coding that will effectively reduce the total number of bits needed to represent certain information.

- The figure below depicts a general data compression scheme, in which compression is performed by an encoder and decompression is performed by a decoder.



A General Data Compression Scheme.

Introduction

- If the compression and decompression processes induce no information loss, then the compression scheme is **lossless**; otherwise, it is **lossy**.

- **Compression ratio**:

$$compression\,ratio = \frac{B_0}{B_1}$$

- B_0 – number of bits before compression
- B_1 – number of bits after compression

- In general, we would desire any codec (encoder/decoder scheme) to have a compression ratio much larger than 1.0.

- The higher the compression ratio, the better the lossless compression scheme, as long as it is computationally feasible.

Basics of Information Theory

- The *entropy* η of an information *source* with alphabet $S = \{s_1, s_2, \ldots, s_n\}$ is:

$$\eta = H(S) = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i}$$

$$= -\sum_{i=1}^{n} p_i \log_2 p_i$$

- p_i – probability that symbol s_i will occur in S.

- $\log_2 \frac{1}{p_i}$

indicates the amount of information (self-information as defined by Shannon) contained in s_i, which corresponds to the number of bits needed to encode s_i.

Basics of Information Theory

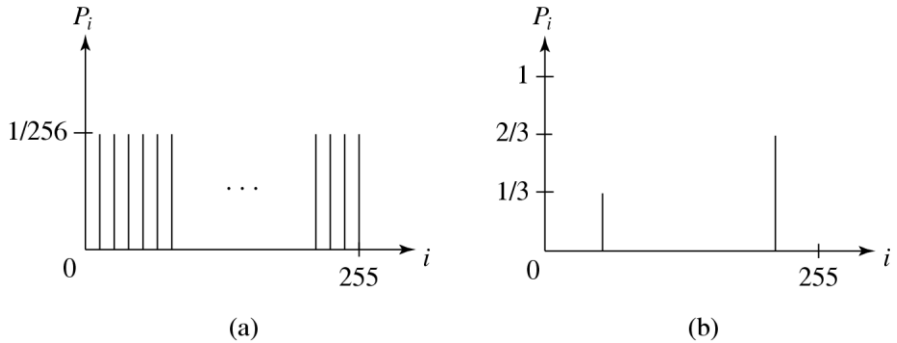
- What is entropy?

is a measure of the number of specific ways in which a system may be arranged, commonly understood as a measure of the disorder of a system.

- As an example, if the information source S is a gray-level digital image, each s_i is a gray-level intensity ranging from 0 to $(2^k - 1)$, where k is the number of bits used to represent each pixel in an uncompressed image.

- We need to find the entropy of this image; which the number of bits to represent the image after compression.

Distribution of Gray-Level Intensities



Histograms for Two Gray-level Images.

- The figure above (left) shows the histogram of an image with uniform distribution of gray-level intensities, i.e., $\forall i\ p_i = 1/256$. Hence, the entropy of this image is:

$$\log_2 256 = 8$$

- The figure above (right) shows the histogram of an image with two possible values (binary image). Its entropy is 0.92.

$$\begin{aligned} \eta &= \frac{1}{3} \cdot \log_2 3 + \frac{2}{3} \cdot \log_2 \frac{3}{2} \\ &= 0.33 \times 1.59 + 0.67 \times 0.59 = 0.52 + 0.40 = 0.92 \end{aligned}$$

Distribution of Gray-Level Intensities

- It is interesting to observe that in the above uniform-distribution example (The figure (left) in the previous slide) we found that $\alpha = 8$, the minimum average number of bits to represent each gray-level intensity is at least 8. No compression is possible for this image.

- In the context of imaging, this will correspond to the "worst case," where neighboring pixel values have no similarity.

Run-Length Coding (RLC)

- One of the simplest forms of data compression.
- The basic idea is that if the information source has the property that symbols tend to form continuous groups, then such symbol and the length of the group can be coded.
- Consider a screen containing plain black text on a solid white background. There will be many long runs of white pixels in the blank space, and many short runs of black pixels within the text.
- Let us take a hypothetical single scan line, with B representing a black pixel and W representing white: WWWWWBWWWWBBBWWWWWWBWWW
- If we apply the run-length encoding (RLE) data compression algorithm to the above hypothetical scan line, we get the following: 5W1B4W3B6W1B3W
- The run-length code represents the original 21 characters in only 14.

Variable-Length Coding

- variable-length coding (VLC) is one of the best-known entropy coding methods
  - Shannon–Fano algorithm.
  - Huffman coding.

The Shannon-Fano Algorithm

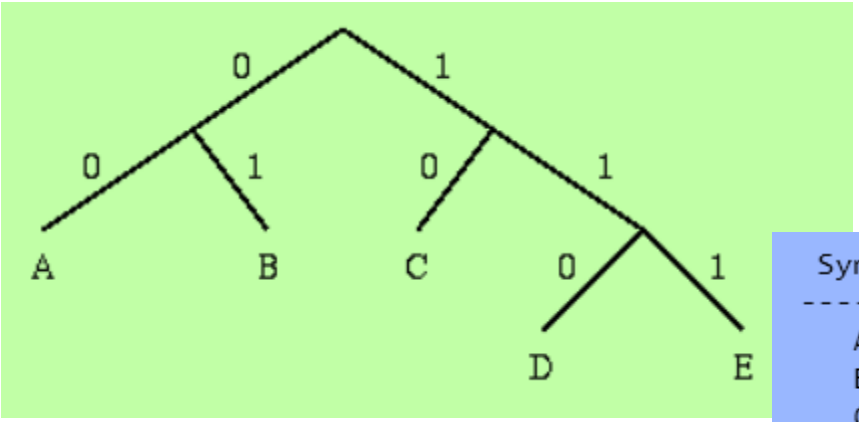
Symbol	A	B	C	D	E
Count	15	7	6	6	5

Encoding for the Shannon-Fano Algorithm:

- A top-down approach

1. Sort symbols according to their frequencies/probabilities, e.g., ABCDE.

2. Recursively divide into two parts, each with approx. same number of counts.



Symbol	Count	log(1/p)	Code	Subtotal (# of bits)
A	15	1.38	00	30
B	7	2.48	01	14
C	6	2.70	10	12
D	6	2.70	110	18
E	5	2.96	111	15
			TOTAL (# of bits):	89

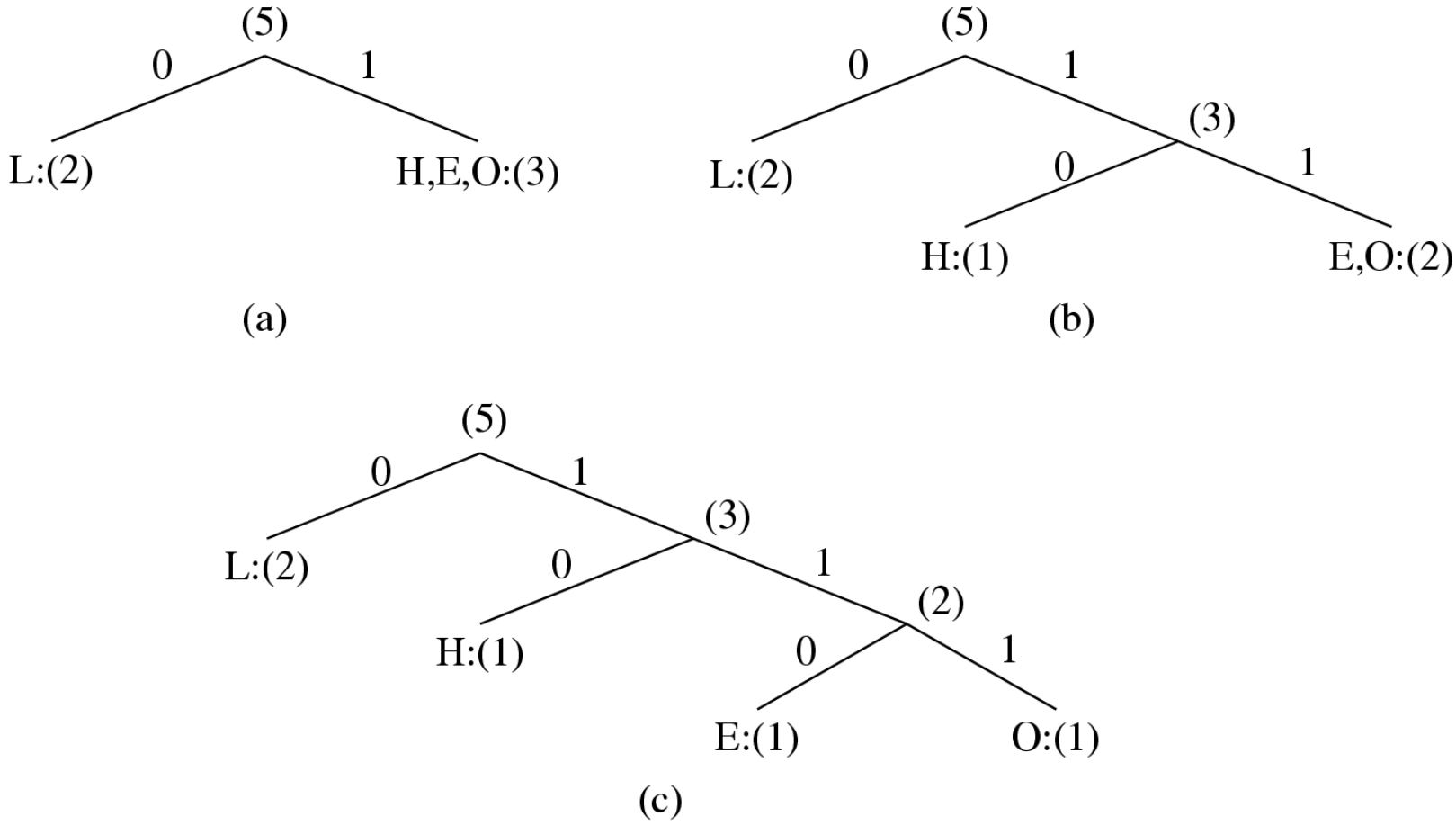
Shannon–Fano Algorithm

- To illustrate the algorithm, let us suppose the symbols to be coded are the characters in the word HELLO.

- The frequency count of the symbols is
  - Symbol H E L O
  - Count 1 1 2 1

- The encoding steps of the Shannon–Fano algorithm can be presented in the following top-down manner:
  - 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



Coding Tree for HELLO by Shannon-Fano.

Result of Performing Shannon-Fano on HELLO

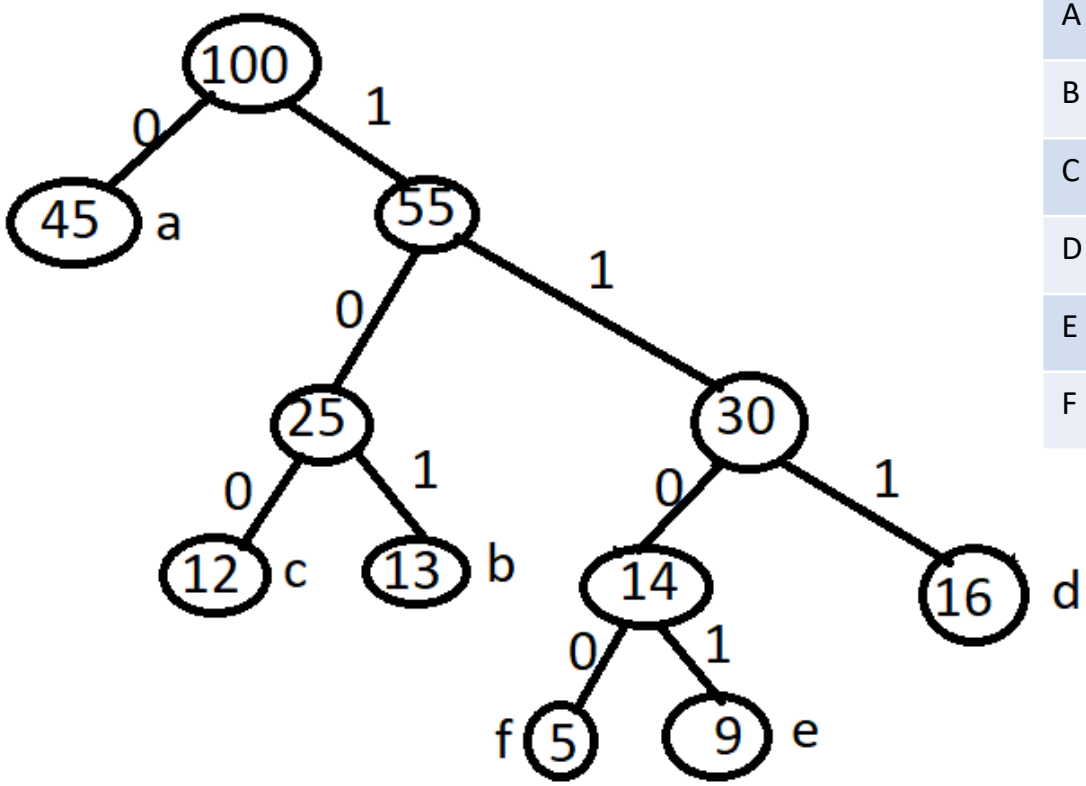
Symbol	Count	$\text{Log}_2 \frac{1}{p_i}$	Code	# of bits used
L	2	1.32	0	2
H	1	2.32	10	2
E	1	2.32	110	3
O	1	2.32	111	3
			TOTAL # of bits:	10

Huffman Coding

- The Shannon–Fano algorithm delivers satisfactory coding results for data compression, but it was soon outperformed and overtaken by the Huffman coding method.
- VLC: Algorithm

1. Initialization: put all symbols on the list sorted according to their frequency counts.
2. Repeat until the list has only one symbol left.
   (a) From the list, pick two symbols with the lowest frequency counts. Form a Huffman subtree that has these two symbols as child nodes and create a parent node for them.
   (b) Assign the sum of the children's frequency counts to the parent and insert it into the list, such that the order is maintained.
   (c) Delete the children from the list.
3. Assign a codeword for each leaf based on the path from the root.

Huffman Coding Example



Symbols	Frequency/ count	Code
A	45	0
B	13	101
C	12	100
D	16	111
E	9	1101
F	5	1100

Huffman Coding

- The Huffman algorithm requires prior statistical knowledge about the information source, and such information is often not available.
- This is particularly true in multimedia applications, where future data is unknown before its arrival, as for example in live (or streaming) audio and video.
- Even when the statistics are available, the transmission of the symbol table could represent heavy overhead
- The solution is to use adaptive Huffman coding compression algorithms, in which statistics are gathered and updated dynamically as the data stream arrives.

Huffman Coding: Practice

- Find the code of each symbol in order to encode the following message using Huffman Coding (space is also a symbol):

THIS IS A GOOD BOOK

- Do It Yourself

- Also,

i. Calculate the entropy

ii. Apply Shanon-Fano Algorithm

iii. See, which one (Huffman or Shannon-Fano) provides better compression?

Dictionary-Based Coding

- The Lempel-Ziv-Welch (LZW) algorithm employs an adaptive, dictionary-based compression technique.

- Unlike variable-length coding, in which the lengths of the codewords are different, LZW uses fixed-length codewords to represent variable length strings of symbols/characters that commonly occur together, such as words in English text.

- As in the other adaptive compression techniques, the LZW encoder and decoder builds up the same dictionary dynamically while receiving the data—the encoder and the decoder both develop the same dictionary.

Dictionary-Based Coding

- LZW proceeds by placing longer and longer repeated entries into a dictionary, then emitting (sending) the code for an element rather than the string itself, if the element has already been placed in the dictionary.

- Remember, the LZW is an adaptive algorithm, in which the encoder and decoder independently build their own string tables. Hence, there is no overhead involving transmitting the string table.

- LZW is used in many applications, such as UNIX compress, GIF for images, WinZip, and others.

LZW Compression Algorithm

```
BEGIN
    s = next input character;
    while not EOF
        {
          c = next input character;

          if s + c exists in the dictionary
             s = s + c;
        else
          {
          output the code for s;
          add string s + c to the dictionary with a new code;
          s = c;
          }
        }
    output the code for s;
END
```

Example: LZW Compression for String ABABBABCABABBA

code	string
1	A
2	B
3	C

A B A B B A B C A B A B B A

Compression ratio: 1.56

s	c	output	code	string
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A	2	5	BA
A	B			
AB	B	4	6	ABB
B	A			
BA	B	5	7	BAB
B	C	2	8	BC
C	A	3	9	CA
A	B			
AB	A	4	10	ABA
A	B			
AB	B			
ABB	A	6	11	ABBA
A	EOF	1		

LZW: Revisited

A B A B B A B C A B A B B A

S (Present Char)	C (Next Char)	O/P (Encoded Message: Code of S)		Code	String (New Word)
Initial Dictionary				1 2 3	A B C
A	B	1		4	AB
B	A	2		5	BA
A	B				
AB	B	4		6	ABB
B	A				
BA	B	5		7	BAB
B	C	2		8	BC
C	A	3		9	CA
A	B				
AB	A	4		10	ABA
A	B				
AB	B				
ABB	A	6		11	ABBA
A	EOF	1			

1 2 4 5 2 3 4 6 1

LZW: Decompression

```
BEGIN
    s = NIL;
    while not EOF
        {
          k = next input code;
          entry = dictionary entry for k;
          output entry;
          if (s != NIL)
             add string s + entry[0] to dictionary
             with a new code;
          s = entry;
        }
END
```

LZW: Decompression

1 2 4 5 2 3 4 6 1

S	K (Next Encoded Code)	Entry/Output (Decoded Message: Char implied by K)		Code	String (Word)
Initial Dictionary				1 2 C	A B C
NIL	1	A			
A	2	B		4	AB
B	4	AB		5	BA
AB	5	BA		6	ABB
BA	2	B		7	BAB
B	3	C		8	BC
C	4	AB		9	CA
AB	6	ABB		10	ABA
ABB	1	A		11	ABBA
A	EOF				

A B A B B A B C A B A B B A

Extension Assignment (EA) # 1

- *Implementation of LZW Compression and Decompression*
- See the uploaded material to get the details of EA 1
- Deadline: November 13, 2017 (Monday)
- Submit Hardcopy on November 14, 2017 (Tuesday) in the class.

Q&A