Please provide the introduction / overview on this lesson

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

Overview

In this chapter, you are going to learn about:

- Definition of Huffman coding
- Concept of Huffman coding tree
- Properties of Huffman coding

1. Introduction > 1.2 Learning Content

Please make sure the hierarch of the content is well formed.
Please organize the lesson in 3-5 main topics and use 3-level headings.

Level 1	Level 2	Level 3
1. Huffman Coding Tree	1.1 Basic Algorithm	
	1.2 Building a Tree	
	1.3 Examples	
2. Properties of Huffman Coding	2.1 Extended Huffman Coding	

ID Will do it by looking at 1.1 Lesson overview

Image Processing	
I. General knowledge in image p rocessing and multimedia	1. Introduction to Image Processing 2. Data Structure and Color of Images 3. Ms. Visual Studio 2008 and OpenCV 4. Introduction to Multimedia Systems 5. Introduction to Video and Lossless Compression 6. Huffman Coding 7. LZ77 8. LZ78 9. LZW
II. Advance knowledge in image segmentation and luminance	10. Sampling 11. Image Segmentation-I 12. Image Segmentation-II 13. Luminance and Histogram Equalization

1. Introduction > 1.4 Learning Objectives

Please provide objective of the lesson by high light keyword and follow (Audience, Behavior, Condition, Degree) to write the objective

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

Objective
Upon completion of this chapter, you will be able to: • Understand concept of **Huffman coding tree** • Understand **properties** of Huffman coding

Please provide keywords of the lesson with explanation

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

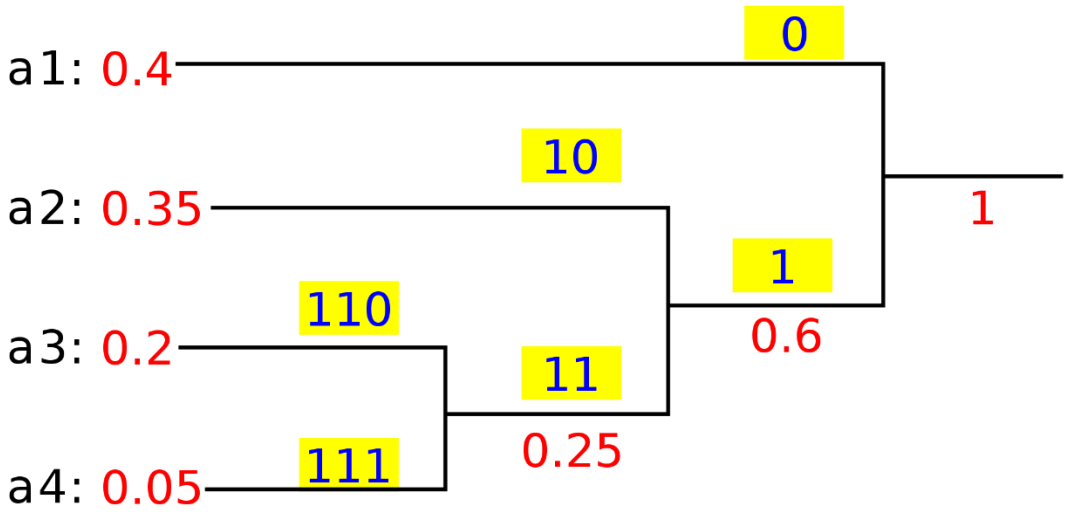
Keywords	Description
Tree traversal	also known as **tree search** is a form of graph **traversal** and refers to the process of visiting (checkin g and/or updating) each node in a tree data structure, exactly once. Such traversals are classified by the order in which the nodes are visited.
Code	is a mapping of **source messages** (words from the source alphabet **alpha**) into **codewords** (words of the code alphabet **beta**). For example, string **alpha = { a, b, c, d, e, f, g, space}**. For purposes of explanation, **beta** will be taken to be **{0, 1}**.
Prefix code	is a type of code system distinguished by its possession of the "**prefix property**", which requires tha t there is no whole code word in the system that is a prefix of any other code word in the system.

2. Learn> Topic: 1. Huffman Coding Tree

(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

- Q: What is **Huffman coding**?
- A: **Huffman coding** is a compression technique used to reduce the number of bi ts needed to send or store a message.
- It's based on the idea that frequently-appearing letters should have shorter bit re presentations and less common letters should have longer representations.
- It works well for text and fax transmissions.



Example of Huffman Coding Tree

2. Learn> Topic: 1. Huffman Coding Tree

(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

▪ Purpose of Huffman coding:
- ➢ Proposed by Dr. David A. Huffman and published in the 1952 paper.
  - ○ “A Method for the Construction of Minimum Redundancy Codes”
- ➢ Applicable to many forms of data transmission.
  - ○ Example: text files

▪ Huffman coding uses a specific method for choosing the representation for each symbol, resulting in a prefix code.
- ➢ It means that the bit string representing some particular symbol is never a p refix of the bit string representing any other symbol

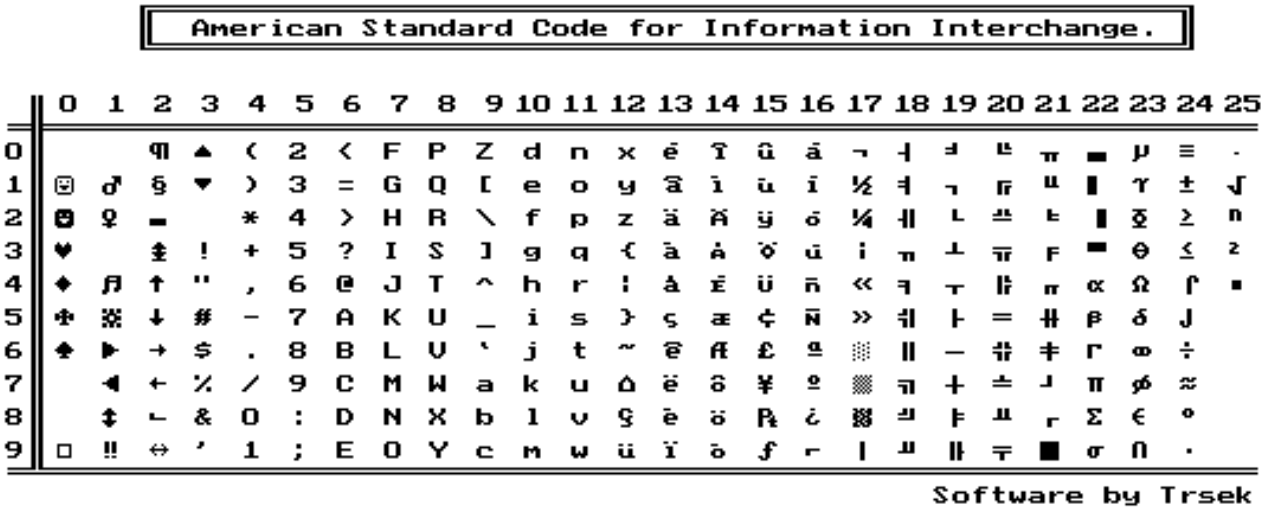
Symbol	Code
a1	0
a2	10
a3	110
a4	111

Example of a prefix code

(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

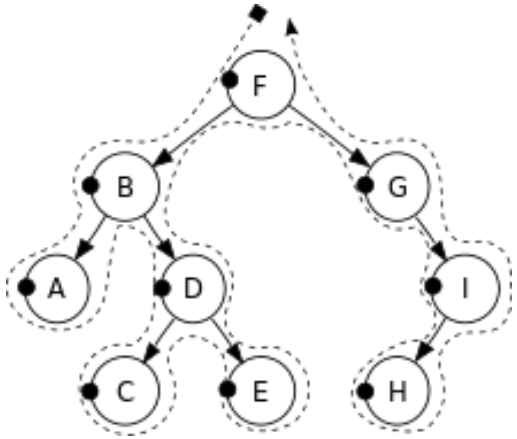
- Huffman coding is a **form of statistical coding**.
- Not all characters occur with the same frequency.
- Yet all characters are allocated the same amount of space.
  - 1 char = 1 byte
- Code word lengths are no longer fixed like ASCII.
- Code word lengths vary and will be shorter for the more frequently used characters.



(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

- The (real) basic algorithm of Huffman coding:
  1) **Scan** text to be compressed and tally occurrence of all characters.
  2) **Sort** or **prioritize** characters based on number of occurrences in text.
  3) **Build** Huffman code tree based on prioritized list.
  4) **Perform** a traversal of tree to determine all codewords.
  5) **Scan** text again and create new file using the Huffman codes.



Sorted binary tree preorder

(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

- Consider the following short word:
  - **Mississippi**
- Count up the occurrences of all characters in the text.
- What characters are present?

M	i	s	s	i	s	s	i	p	p	i

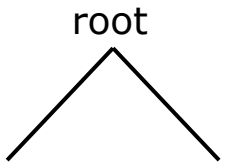
- What is the frequency of each character in the text?

Character	Frequency
M	1
i	4
s	4
p	2

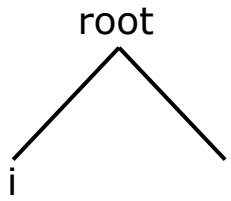
(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

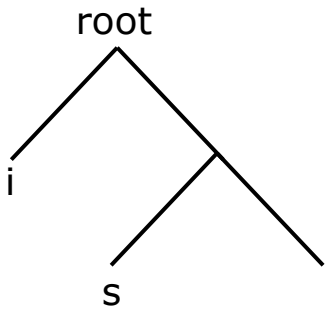
- Next, create binary tree nodes with character and frequency of each character.



- Place nodes in a priority queue.
  - The higher the occurrence, the higher the priority in the tree.

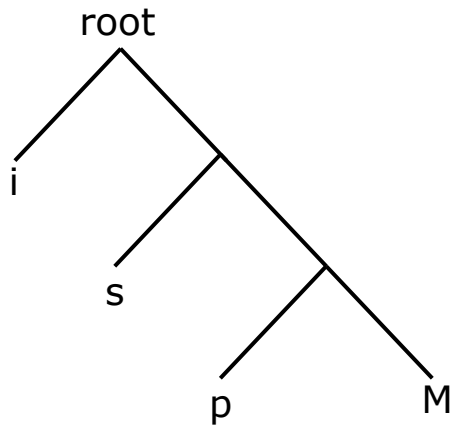


- Add another node on the right side if it still rests a character.



(1) Learning Contents	☒ A : Text-based + Audio ☐ B : Text-based + Video ☐ C : Only Video

▪ So, we do like the previous step until the last character.

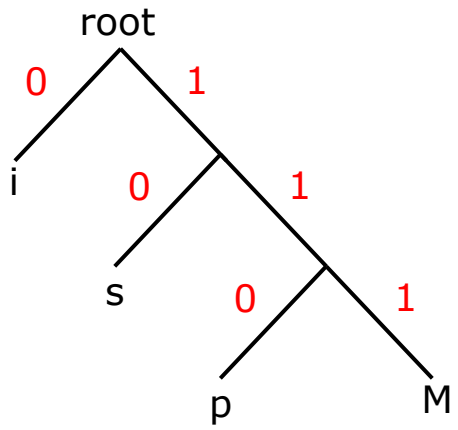


▪ After we finish putting all character on the tree, we have to add code (**0** or **1**) on both side of our tree.
▪ If we add code **0** on the left side, the right side must be **1**.
▪ We repeat adding code until the last node of the tree.

(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

- Assume that we add code **0** to left side and **1** to right side.

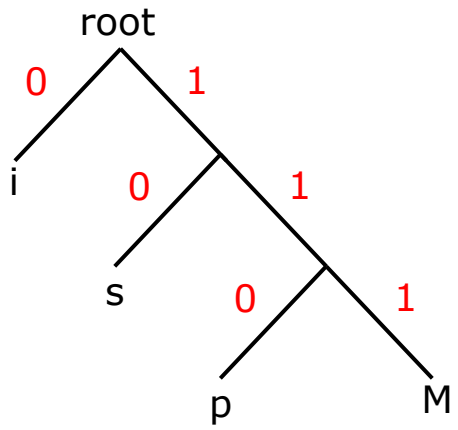


- P(i) = 4/11, P(s) = 4/11, P(p) = 2/11, and P(M) = 1/11.
- So, we put symbol “i” on the top left of the tree.
- Then following by symbol “s”, “p”, and “M”.

(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

- Now we find codeword of each symbol which starts from the root.



- i → 0
- s → 10
- p → 110
- M → 111

2. Learn> Topic: 1.3 Examples

(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

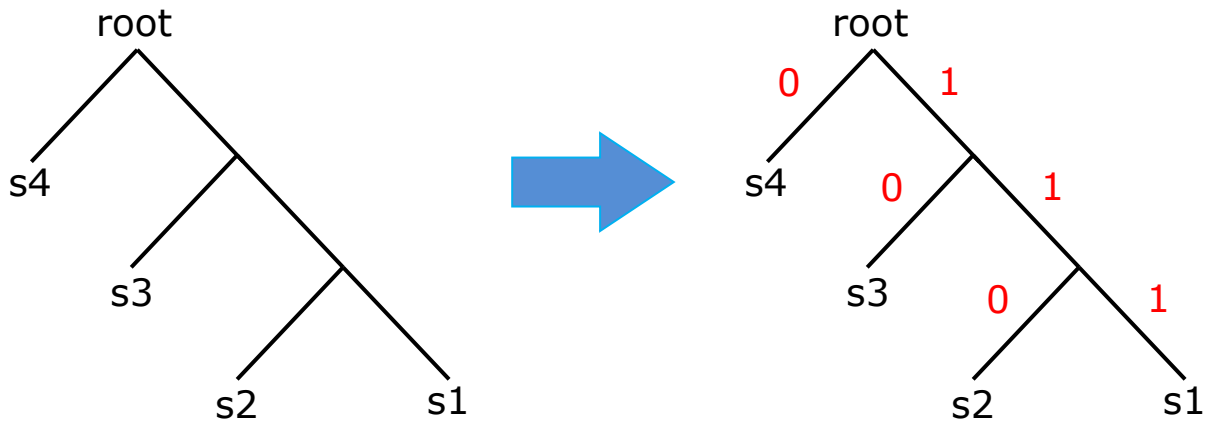
- Example1: Assume that we have 4 symbols: s1, s2, s3, s4. We know that:
  - P(s1) = 0.125
  - P(s2) = 0.125
  - P(s3) = 0.25
  - P(s4) = 0.5
  - Find codeword of each symbol?

- A: First, we have to draw a Huffman tree by putting symbol with the highest prob ability on the top left and symbol with the lowest probability on the bottom right.
  - We know that P(s4) = 0.5 , P(s3) = 0.25, P(s2) = P(s1) = 0.125.
  - So, we put “s4” on the top and then follow by “s3”, “s2”, and “s1”.
  - It doesn’t matter if we put “s1” before “s2” (the same probability).

(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

▪ A (cont.): Now we start to build the Huffman tree.



- s4 → 0
- s3 → 10
- s2 → 110
- s1 → 111

(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

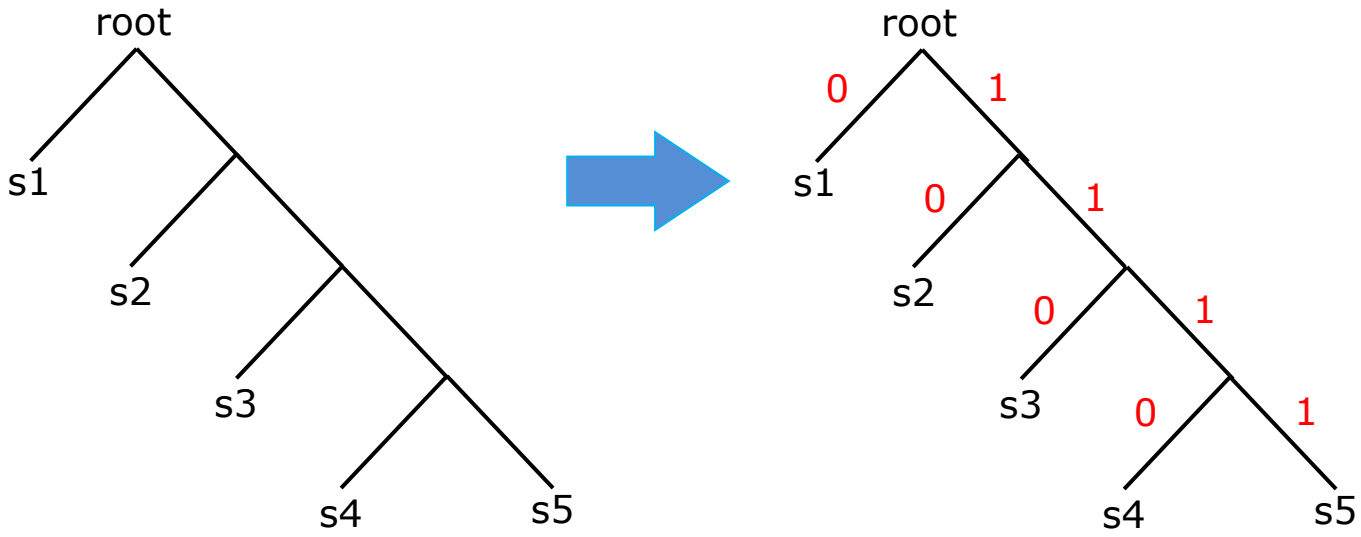
- Example2: Assume that we have 5 symbols: s1, s2, s3, s4, s5. We know that:
  - P(s1) = 0.4
  - P(s2) = 0.2
  - P(s3) = 0.2
  - P(s4) = 0.1
  - P(s5) = 0.1
  - Find codeword of each symbol?

- A: First, we have to draw a Huffman tree by putting symbol with the highest prob ability on the top left and symbol with the lowest probability on the bottom right.
  - We know that P(s1) = 0.4 , P(s2) = P(s3) = 0.2, P(s4) = P(s5) = 0.1.
  - So, we put “s1” on the top and then follow by “s2”, “s3”, “s4”, and “s5”.
  - It doesn’t matter if we put “s3” before “s2” and “s5” before “s4”.

(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

▪ A (cont.): Now we start to build the Huffman tree.



- s1 → 0
- s2 → 10
- s3 → 110
- s4 → 1110
- s5 → 1111

(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

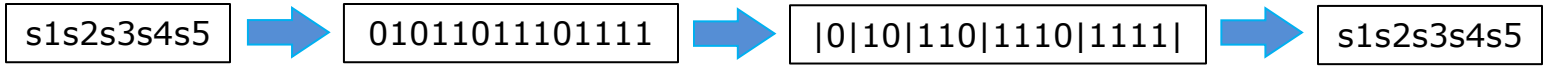
▪ Huffman coding uses longer codewords for symbols with **smaller** probabilities and **shorter** codewords for symbols that often occur.

- Example: s1 → **0** and s5 → **1111**.

▪ The two longest codewords differ only in the **last bit**.

- Example: s4 → 1110 and s5 → 1111.

▪ The codewords are **prefix codes** and uniquely decodable.



▪ According to **Kraft's inequality**: Average codeword length (***E***) >= Entropy (***H***).

(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

- Difference between “average length” and “entropy” gives the percent of optimal.
- The optimal case is when the average length of a code is equal to the entropy.
  - For example, if average length is 1 and entropy is 0.72.
  - So, 1 - 0.72 = 0.28 → 28% (not far from optimal).
- If both “average length” and “entropy” are 1, the compression is optimal.

- From Kraft's inequality, every code can be **improved** by only decreasing its code word lengths, so that equality holds.
- Suppose that the probability of the i^{th} symbol is p_i and codeword length l_i:
  - The average codeword length is:

$$E = \sum_{i=1}^{n} p_i l_i$$

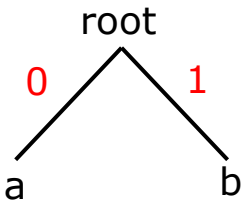
(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

▪ Huffman coding is not effective for cases when there are small number of symbol s and the probabilities are highly skewed.

▪ Example: A source has 2 symbols a and b. P(a) = 0.9 and P(b) = 0.1.

- ***H*** = $0.9\log_2(1/0.9) + 0.1\log_2(1/0.1)$
- ***H*** = 0.4690 bit
- ***E*** = 0.9x1 + 0.1x1
- ***E*** = 1 bit
- 1 - 0.4690 = 0.531 = **53.1%**
- Thus, it is **far from optimal!**

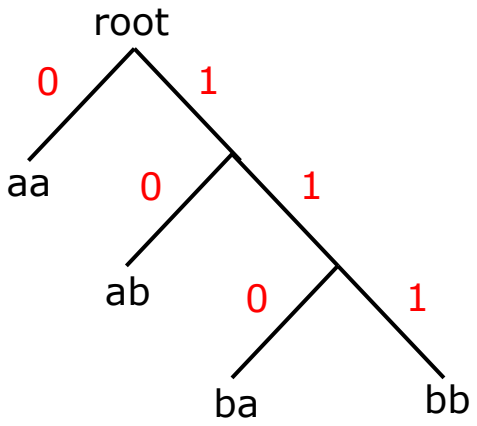


a → 0
b → 1
l_a = 1 and l_b = 1

(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

- But, we can encode a group symbols together and get better performance.
- For the previous example, an extended source has symbols {aa, ab, ba, bb} and
  - P(aa) = P(a) x P(a) = 0.81
  - P(ab) = P(a) x P(b) = 0.09
  - P(ba) = P(b) x P(a) = 0.09
  - P(bb) = P(b) x P(b) = 0.01
  - $\boldsymbol{H} = 0.81\log_2(1/0.81) + [0.09\log_2(1/0.09)]x2 + 0.01\log_2(1/0.01)$
  - $\boldsymbol{H} = 0.9259$ bit
  - $\boldsymbol{E} = 0.81x1 + 0.09x2 + 0.09x3 + 0.01x3$
  - $\boldsymbol{E} = 1.29$ bit
  - 1.29 - 0.9259 = 0.3641 = **36.41%**
  - Thus, it is **much better!**



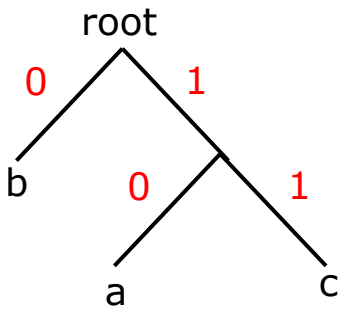
aa → 0
ab → 10
ba → 110
bb → 111
$l_{aa} = 1$, $l_{ab} = 2$, $l_{ba} = 3$, and $l_{bb} = 3$

(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

- Q: Find average codeword length? Input string: abbcbcabab.
- A: Probability of each symbol:
  - P(a) = 3/10 = 0.3
  - P(b) = 5/10 = 0.5
  - P(c) = 2/10 = 0.2

  - ***H*** = $0.3\log_2(1/0.3) + 0.5\log_2(1/0.5) + 0.2\log_2(1/0.2)$
  - ***H*** = 1.479 bit
  - ***E*** = 0.3x2 + 0.5x1 + 0.2x2
  - ***E*** = 1.5 bit
  - 1.5 - 1.479 = 0.021 = **2.1%**
  - Thus, it is **near optimal!**

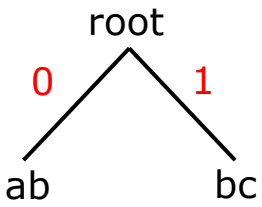


b → 0
a → 10
c → 11
l_a = 2, l_b = 1, and l_c = 2

(1) Learning Contents

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

- Assume that we make group of 2 symbols: abbcbcabab. Find ***E***?
- Probability of group of symbol:
  - P(ab) = 3/5 = 0.6
  - P(bc) = 2/5 = 0.4
  - ***H*** = $0.6\log_2(1/0.6) + 0.4\log_2(1/0.4)$
  - ***H*** = 0.966 bit
  - ***E*** = (0.6x1 + 0.4x1)/2 (make group of 2 symbols)
  - ***E*** = 0.5 bit
  - ***E*** < ***H***
  - Thus, **we cannot make group of 2 symbols!**



ab → 0
bc → 1
l_{ab} = 1, and l_{bc} = 1

Please give a lesson summary.
Each topic can be summarized into a sentence, diagram, or even a word.

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

Summarize

▪ **Huffman coding** is a compression technique used to reduce the number of b its needed to send or store a message. It works well for text and fax transmissi ons.

▪ The (real) basic algorithm of Huffman coding:

1) **Scan** text to be compressed and tally occurrence of all characters.
2) **Sort** or **prioritize** characters based on number of occurrences in text.
3) **Build** Huffman code tree based on prioritized list.
4) **Perform** a traversal of tree to determine all codewords.
5) **Scan** text again and create new file using the Huffman codes.

▪ Huffman coding uses longer codewords for symbols with **smaller** probabilitie s and **shorter** codewords for symbols that often occur. The two longest codew ords differ only in the **last bit**. The codewords are **prefix codes** and uniquely decodable.

Provide references if you think the students need.

Reference

- https://en.wikipedia.org/wiki/Huffman_coding
- www.utdallas.edu/~daescu/huffman.ppt
- https://en.wikipedia.org/wiki/Prefix_code
- https://en.wikipedia.org/wiki/Variable-length_code#Uniquely_decodable_codes
- http://cs.stackexchange.com/questions/21351/gap-between-the-average-length-of-a-huffman-code-and-its-entropy

4. Outro > 4.3 Assignment

Please provide the assignment such as exercise , discussion, research topic, Short essay, case studies, ….

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

Assignment
1) Draw a Huffman tree and calculate average codeword length? If we have: ➢ Input string: “gotoyooggy” (double quotation marks doesn’t count!) 2) Draw a Huffman tree and calculate average codeword length? If we have: ➢ Input string: “alibaba bali la” (double quotation marks doesn’t count!)

4. Outro > 4.4 Next Lesson

This is the end of the lesson.
Ending message and introduction to next lesson including lesson title and topics should be given.

☒ A : Text-based + Audio
☐ B : Text-based + Video
☐ C : Only Video

Overview

- Introduce to LZ77
- Concept of LZ77

Next Lesson Title	**LZ77** 1. Concept of LZ77 2. Examples