



Faculty of Engineering and Technology  
Electrical and Computer Engineering Department  
Information and Coding Theory  
ENEE5304

Report

Project No. 1

**“Huffman Code”**

- **Prepared by:** Ameena Jadallah 1171018
- **Instructor:** Dr. Wael Hashlamoun
- **Section No.:** 1
- **Date:** 20/12/2021

## Introduction and Theory

this project is on contains story, or any text. These texts are read then analyzed based on frequency of each character, After that we will use them to find: the codeword for the characters and the average number of bits/characters for whole story then we will calculate the entropy of the alphabet. Finally, we will find number of bits needed to encode story then calculate the percentage of the compression accomplished by using the Huffman encoding as compared to Normal ASCII code.

### Huffman code

Huffman code is a specific kind of ideal prefix code that is ordinarily utilized to decrease information size. The most common way of finding or utilizing such a code continues through Huffman coding, an algorithm created by David A. Huffman. The result from Huffman's calculation can be seen as a variable-length code table for encoding a source image. As in other entropy encoding strategies, It lots a more limited length codeword for a character with high recurrence and a more extended length codeword for a character with less recurrence.

#### Huffman Encoding Algorithm

1. Arrange the source symbols in a decreasing order of probability. (sorting stage)
2. The last two symbols of lowest probability are assigned a 0 and 1.
3. The probability of the last two symbols are add together to form a new symbol with prob.  $= \sum \text{prob. of last two symbols}$
4. The procedure is repeat until we end up with only two symbols. write them digits 0&1.
5. The code for each source symbol is found by backward and tracing the sequence of 0's and 1's to the symbol and its successors.
6. Result: Huffman code is an optimal code

#### Advantages of Huffman Encoding

This encoding scheme results in saving a lot of storage space, since binary codes generated are of different lengths, generates shorter binary codes for encoding symbols that appear more frequently in the input, and The binary codes generated are prefix-free

#### Disadvantages of Huffman Encoding

Lossless data encoding schemes achieve a lower compression ratio compared to lossy encoding techniques. Thus, Huffman's techniques encoding is suitable only for encoding text and program files and are but not for digital images, Huffman encoding is a relatively slower process since it uses two passes- for building the statistical model and for encoding. finally, Since the length of all the binary codes is different, it probably becomes results in incorrect decoding and subsequently, a wrong output.

## Results and analysis

In this part of the report, we will answer theoretically and mathematically the questions, then simulate the project with the result using JAVA code.

### 1. Reading File

The first step in the project is to find the characters with their frequency in the story “Shooting an Elephant by George Orwell” by reading the file “story.text” and convert to upper case in string, then store in string builder object.

### 2. Character frequency (f)

Now to get the frequency of the character we make a method take the string which contain the story as a parameter Then, we made a counter array which we put away in it the recurrence of each character. From that point forward, We made another array called charArray which we put away the characters. additionally, I made charfreq array to make the arrangement all together .

### 3. probabilities of the characters (P)

I will divide the frequency for each character on the length of the all Story to get the probability of each character :

$$P(x) = \frac{\text{The frequency of character}}{\text{The Length of all story}}$$

### 4. entropy of the alphabet (H)

Calculating using summation of multiply the probability of each character with the negative log of its probability:

$$H = -1(\sum P(x) \log_2(P(x))) = 3.76$$

### 5. average number of bits/character for the whole story (L)

we calculated by multiply the probability of each symbol with its codeword length and summation all result together

$$L = \sum P(x)l(x) = 3.78$$

### 6. number of bits needed to encode the story

- Calculating by using Huffman by summation of the length multiplying by frequency of each codes Word.

$$\sum l(x)f(x) = 140$$

- But Calculating by using ASCII is each character in ASCII consist of 8 bits and as we find the length of the text :

$$8 * \text{number of all bits} \text{ or } 8 * \text{Length of whole story} = 296$$

## 7. percentage of compression

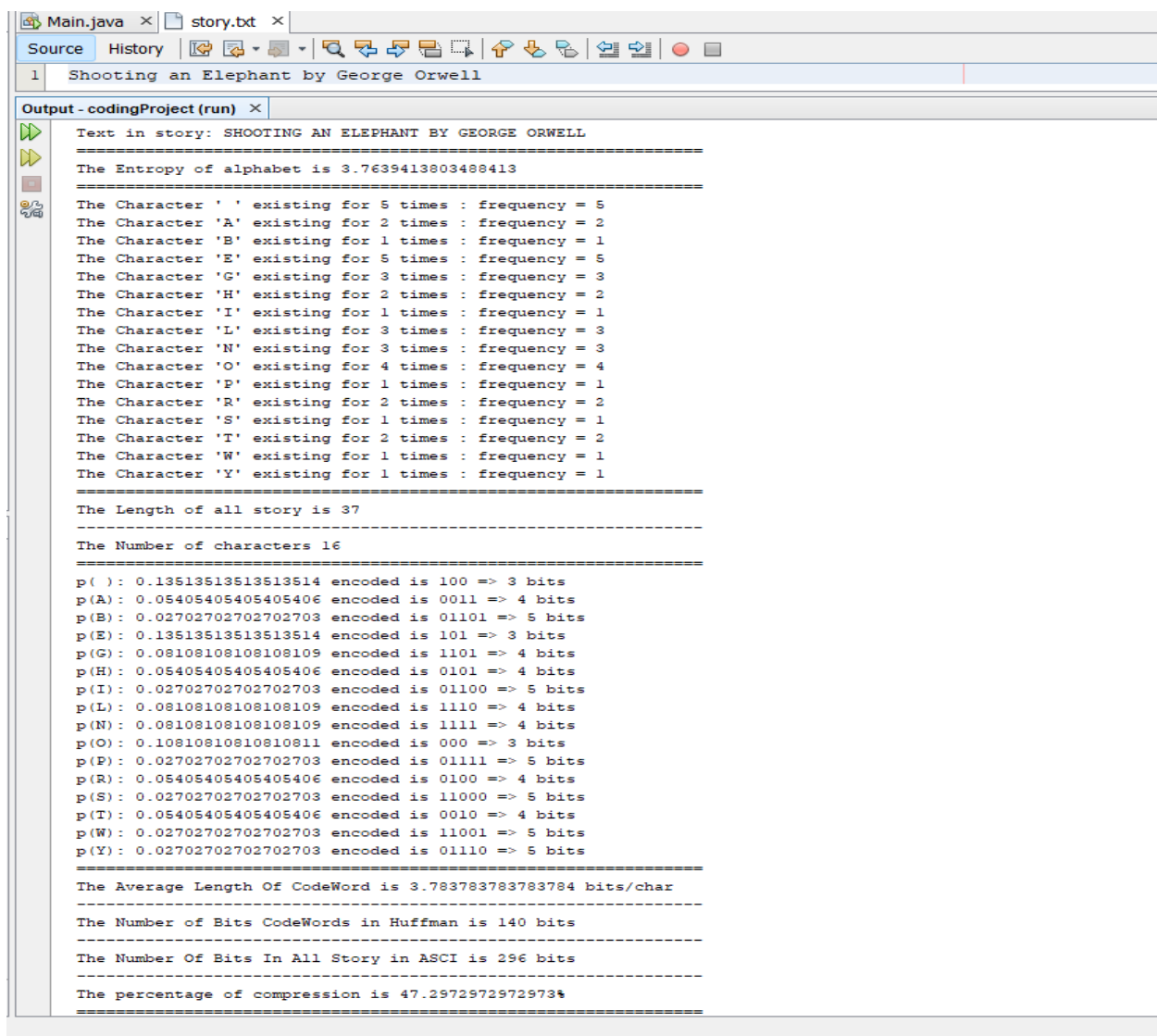
we divided number of bits using Huffman code by the number of bits using ASCII.

$$\text{percentage} = \frac{\text{\#bits huffman code}}{\text{\#bits ASCII code}} * 100\% = 47.29\%$$

## 8. The code words for the character

To find the code word for those character so first we after calculate the probability will divide each frequency by the length of the story (number of characters). The input probability specifies the probability of occurrence for each of the input symbols. The length of probability must equal the length of symbols. The function also returns average code word length.

## 9. This figure show the result of simulation of the code :



```
Source History | [Icons]
1 Shooting an Elephant by George Orwell

Output - codingProject (run) x
Text in story: SHOOTING AN ELEPHANT BY GEORGE ORWELL
=====
The Entropy of alphabet is 3.7639413803488413
=====
The Character ' ' existing for 5 times : frequency = 5
The Character 'A' existing for 2 times : frequency = 2
The Character 'B' existing for 1 times : frequency = 1
The Character 'E' existing for 5 times : frequency = 5
The Character 'G' existing for 3 times : frequency = 3
The Character 'H' existing for 2 times : frequency = 2
The Character 'I' existing for 1 times : frequency = 1
The Character 'L' existing for 3 times : frequency = 3
The Character 'N' existing for 3 times : frequency = 3
The Character 'O' existing for 4 times : frequency = 4
The Character 'P' existing for 1 times : frequency = 1
The Character 'R' existing for 2 times : frequency = 2
The Character 'S' existing for 1 times : frequency = 1
The Character 'T' existing for 2 times : frequency = 2
The Character 'W' existing for 1 times : frequency = 1
The Character 'Y' existing for 1 times : frequency = 1
=====
The Length of all story is 37
=====
The Number of characters 16
=====
p( ) : 0.13513513513513514 encoded is 100 => 3 bits
p(A) : 0.05405405405405406 encoded is 0011 => 4 bits
p(B) : 0.02702702702702703 encoded is 01101 => 5 bits
p(E) : 0.13513513513513514 encoded is 101 => 3 bits
p(G) : 0.08108108108108109 encoded is 1101 => 4 bits
p(H) : 0.05405405405405406 encoded is 0101 => 4 bits
p(I) : 0.02702702702702703 encoded is 01100 => 5 bits
p(L) : 0.08108108108108109 encoded is 1110 => 4 bits
p(N) : 0.08108108108108109 encoded is 1111 => 4 bits
p(O) : 0.10810810810810811 encoded is 000 => 3 bits
p(P) : 0.02702702702702703 encoded is 01111 => 5 bits
p(R) : 0.05405405405405406 encoded is 0100 => 4 bits
p(S) : 0.02702702702702703 encoded is 11000 => 5 bits
p(T) : 0.05405405405405406 encoded is 0010 => 4 bits
p(W) : 0.02702702702702703 encoded is 11001 => 5 bits
p(Y) : 0.02702702702702703 encoded is 01110 => 5 bits
=====
The Average Length Of CodeWord is 3.783783783783784 bits/char
=====
The Number of Bits CodeWords in Huffman is 140 bits
=====
The Number Of Bits In All Story in ASCII is 296 bits
=====
The percentage of compression is 47.2972972972973%
```

## Conclusion

We noticed that according our results that we obtained from running our program improvement when using Huffman coding over ASCII coding with a compression rate of equal 47.29. The average bit/symbol is 3.783 which is almost close to the minimum of entropy 3.763 meaning Huffman coding is ideal to be used in this case.

Huffman algorithm is mainly efficient in compressing text or program files, Images, without losing any piece of information. like they are often used in prepress, are better handled by other compression algorithms. and can be used to improve coding for texts and compress data for efficient transmission. Also, the prefix-free code provides a reliable coding of characters for this much more efficient encoding compared to ASCII.

## References

1. [https://en.wikipedia.org/wiki/Huffman\\_coding](https://en.wikipedia.org/wiki/Huffman_coding) [in 18/12/21]
2. <https://www.studytonight.com/data-structures/huffman-coding> [19/12/21]
3. Course Slides [19/12/21]

## Appendix

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.PriorityQueue;
import java.util.Scanner;
import java.util.TreeMap;

class Node {

    Node left, right;
    double value;
    String character;

    public Node(double value, String character) {
        this.value = value;
        this.character = character;
        left = null;
        right = null;
    }

    public Node(Node left, Node right) {
        this.value = left.value + right.value;
        character = left.character + right.character;
        if (left.value < right.value) {
            this.right = right;
            this.left = left;
        } else {
            this.right = left;
            this.left = right;
        }
    }
}

/* Huffman coding , decoding */
public class Main {

    static final boolean readFromFile = false;
    static final boolean newTextBasedOnOldOne = false;

    static PriorityQueue<Node> nodes = new PriorityQueue<>((o1, o2) -> (o1.value < o2.value) ? -1 : 1);
    static TreeMap<Character, String> codes = new TreeMap<>();
    static int i;
    static int length_arr = 0;
    static String str;
    static int counter[] = new int[256];
    static char[] charArray = new char[256];
    static int[] charfreq = new int[256];
    static double LengthCodeWord = 0;
    static int NumBitCodeWord = 0;
    static double entropy = 0;
    static String text = "";
    static String encoded = "";
    static String decoded = "";
    static int ASCII[] = new int[128];

    // main function
    public static void main(String[] args) throws IOException {
        handleNewText();
        prec_comp();
    }

    //reading text file
    private static boolean handleNewText() throws IOException {
        int oldTextLength = text.length();
        text = readFile("story.txt");
        if (newTextBasedOnOldOne && (oldTextLength != 0 && !Similarity())) {
            System.out.println("Not Valid ");
        }
    }
}
```

```

        text = "";
        return true;
    }
    ASCII = new int[128];
    encoded = "";
    decoded = "";
    nodes.clear();
    codes.clear();
    System.out.println("Text in story: " + text);
    System.out.println("=====");
    find_entropy(nodes, true);
    buildTree(nodes);
    codeWord(nodes.peek(), "");
    encodeText();
    return false;
}

static String readFile(String fileName) throws IOException {
    BufferedReader br = new BufferedReader(new FileReader(fileName));
    try {
        StringBuilder sb = new StringBuilder();
        String line = br.readLine();
        while (line != null) {
            sb.append(line);
            line = br.readLine();
        }
        String str = sb.toString();
        String UpperStr = str.toUpperCase();
        return UpperStr;
    } finally {
        br.close();
    }
}

private static boolean Similarity() {
    boolean flag = true;
    for (int i = 0; i < text.length(); i++) {
        if (ASCII[text.charAt(i)] == 0) {
            flag = false;
            break;
        }
    }
    return flag;
}

/*1. Find the average number of bits/character for the whole story
2. find the number of bits needed to encode the story*/
public static void encodeText() {

    find_frequency(text);
    encoded = "";
    for (int i = 0; i < length_arr; i++) {
        encoded = codes.get(charArray[i]);
        System.out.println("p(" + charArray[i] + "): " + 1.0 * charfreq[i] / text.length() + " encoded is "
+ encoded + " => " + encoded.length() + " bits");
        LengthCodeWord = LengthCodeWord + (encoded.length() * 1.0 * charfreq[i] / text.length());
        NumBitCodeWord = NumBitCodeWord + encoded.length() * charfreq[i];
    }
    System.out.println("=====");
    System.out.println("The Average Length Of CodeWord is " + LengthCodeWord + " bits/char");
    System.out.println("-----");
    System.out.println("The Number of Bits CodeWords in Huffman is " + NumBitCodeWord + " bits");
    System.out.println("-----");

}

private static void buildTree(PriorityQueue<Node> vector) {
    while (vector.size() > 1) {
        vector.add(new Node(vector.poll(), vector.poll()));
    }
}

```



```

// generate the codewords

private static void codeWord(Node node, String s) {
    int num_bit = 0;
    if (node != null) {
        if (node.right != null) {
            codeWord(node.right, s + "1");
            num_bit = num_bit + 1;
        }

        if (node.left != null) {
            codeWord(node.left, s + "0");
            num_bit = num_bit + 1;
        }

        if (node.left == null && node.right == null) {
            codes.put(node.character.charAt(0), s);
        }
    }
}

//1. Find the average number of bits/character for the whole story
public static void find_frequency(String str) {
    for (i = 0; i < str.length(); i++) {
        counter[(int) str.charAt(i)]++;
    }
    int sum_freq_all_letter = 0;
    int index = 0;
    // Print Frequency of characters
    for (i = 0; i < 256; i++) {
        if (counter[i] != 0 && (char) i != '\n') {
            // Arrays.fill(charArray, (char) i);
            charfreq[index] = (int) counter[i];
            charArray[index] = (char) i;
            System.out.println("The Character " + "'" + charArray[index] + "'" + " existing for " +
charfreq[index] + " times : frequency = " + charfreq[index]);
            sum_freq_all_letter = sum_freq_all_letter + counter[i];
            length_arr = length_arr + 1;
            index = index + 1;
        }
    }
    System.out.println("=====");
    System.out.println("The Length of all story is " + sum_freq_all_letter);
    System.out.println("-----");
    System.out.println("The Number of characters " + length_arr);
    System.out.println("=====");
}

//2. Find the entropy of the alphabet.
private static void find_entropy(PriorityQueue<Node> vector, boolean printIntervals) {
    if (printIntervals) {
        for (int i = 0; i < text.length(); i++) {
            ASCII[text.charAt(i)]++;
        }
    }
    double probability;
    for (int i = 0; i < ASCII.length; i++) {
        if (ASCII[i] > 0) {
            vector.add(new Node(ASCII[i] / (text.length() * 1.0), ((char) i) + ""));
            if (printIntervals) {
                probability = 1.0 * ASCII[i] / (text.length());
                entropy += -probability * Math.log(probability) / Math.log(2);
            }
        }
    }
    System.out.println("The Entropy of alphabet is " + entropy);
    System.out.println("=====");
}

/*3. If ASCII code is used, find the number of bits needed to encode the story

```

```
4. Find the percentage of compression accomplished by using the Huffman encoding as compared to ASCII
code.*/
static void prec_comp() {
    int NumberBitInAllStory = 8 * text.length();
    System.out.println("The Number Of Bits In All Story in ASCII is " + NumberBitInAllStory + " bits");
    System.out.println("-----");
    double percentage = (1.0 * NumBitCodeWord / NumberBitInAllStory) * 100;
    System.out.println("The percentage of compression is " + percentage + "%");
    System.out.println("=====");
}
}
```