

Faculty of Information Technology

Computer Systems Engineering Department

Information and Coding Theory

ENEE5304

Projects Report

Prepared by:

Arwa Doha 1190324

Yousef Hassan Mahmood 1190317

Instructor:

Dr. Wael Hashlamoun

Section: 1

Date : 7/1/2024

Table of Contents

[Introduction 2](#_bookmark0)

[Theoretical Background 2](#_bookmark1)

[Advantages of Huffman Encoding 2](#_bookmark2)

[Results 3](#_bookmark3)

[The frequency for all characters 3](#_bookmark4)

[The probabilities, the lengths of the codewords, and the codewords for all characters 4](#_bookmark4)

[Fill Table 5](#_bookmark4)

[Other Calculations 5](#_bookmark4)

[Conclusions 6](#_bookmark5)

[References 6](#_bookmark6)

[Appendix: 7](#_bookmark7)

# Introduction:

Huffman coding is a highly effective prefix code that is often used in computer science and information theory to achieve lossless data compression. The main component of this coding method, Huffman's algorithm, creates a variable-length code table for encoding source symbols, such as characters in a file. The estimated probability or frequency connected to each possible value of the source symbol is the basis for creating this table. In accordance with the entropy encoding principles, less common symbols are usually represented by longer bit sequences, whilst more frequent symbols are usually represented by shorter ones.[1]

This project-includes any-literature, including-stories. We will-utilize these texts to identify-the codewords for the characters-and the average-number of bits/characters for the-entire tale-after which we will compute the alphabet's-entropy. These texts are-read and then-evaluated based-on the frequency of each-character. The total number of bits-required to encode-the tale will-then be determined, and the amount-of compression achieved using-Huffman encoding-in comparison-to standard-ASCII code will be-calculated.

## Theoretical Background:

Huffman coding hinges on the probability distribution of symbols exchanged between sender and receiver. Once frequencies are established, the symbols are organized in descending order based on these frequencies. Each branch-will have a distinct-binary digit after-merging the two-symbols with the-lowest frequencies, adding their-frequencies, and doing so.[1]

The binary-digits on the branches-from the previous-node we merged-back to the original-symbol serve as the-code-word for-each symbol. Less bits-are utilized for- more frequently-occurring characters in-Huffman coding, which-is used for lossless-data compression.[1]

## Advantages of Huffman Encoding

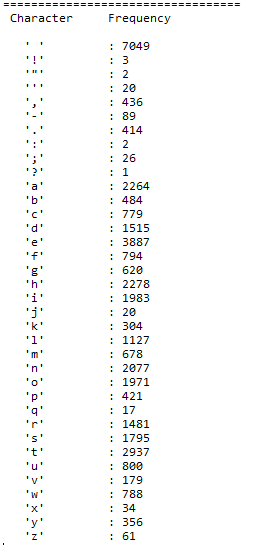
This-encoding scheme-results in saving a lot-of storage space,-since binary-codes generated-are of different-lengths, generates-shorter binary codes-for encoding-symbols that-appear-more frequently in-the input, and-the binary codes-generated are-prefix-free.[2]

# Results

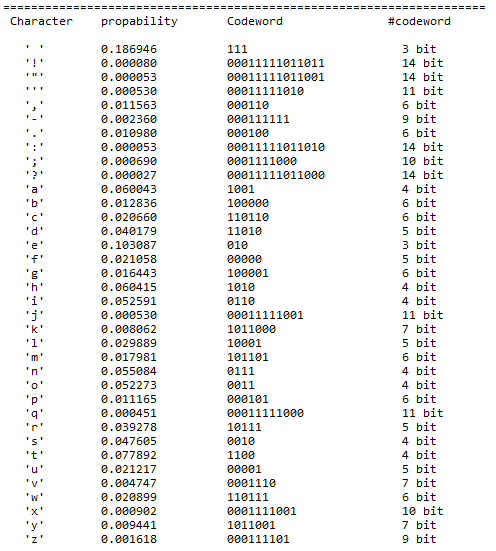
## The presented results are based on the successful implementation of the Huffman coding algorithm using JAVA. The computer program, designed to simulate Huffman coding for a given set of symbols and their probabilities, played a key role in generating the codewords. The efficiency and accuracy of the Huffman coding algorithm were instrumental in achieving a compression rate of 52.68% when compared to the traditional ASCII encoding. The utilization of JAVA in the code ensured robust execution and stream lined the computation, contributing to the overall success of the project.

By using our program we found :

## The frequency for all characters:



## The probabilities, the lengths of the codewords, and the codewords for all characters:



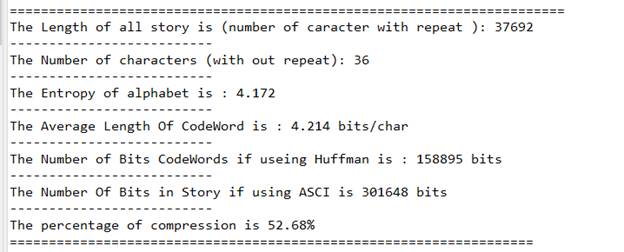
Note that: the-length of the-code words varies-depending on the-frequency of-each symbol; the symbol with-the highest Probability of-appearing in-the-message has the-fewest bits.

## Fill Table :

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **Probability** | **codeword** | **Length of codeword in**  **bits** |
| **a** | 0.060043 | 1001 | 4 bit |
| **b** | 0.012836 | 100000 | 6 bit |
| **c** | 0.020660 | 110110 | 6 bit |
| **d** | 0.040179 | 11010 | 5 bit |
| **E** | 0.103087 | 010 | 3 bit |
| **f** | 0.021058 | 00000 | 5 bit |
| **m** | 0.017981 | 101101 | 6 bit |
| **z** | 0.001618 | 000111101 | 9 bit |
| **space** | 0.186946 | 111 | 3 bit |
| **.(dot)** | 0.010980 | 000100 | 6 bit |

…………………………………………………………………………………………………………………………………

## Other Calculations:



We can see the-stark contrast-between ASCII coding-and Huffman coding,-as well as-how the average-amount of -bits for-each symbol is-dangerously near to the-entropy.

The 52.68% compression rate reflects the significant efficiency of Huffman coding in minimizing data size compared to ASCII. Highlight its efficiency in data representation.

# Conclusions:

In conclusion highlights the notable difference between ASCII and Huffman coding, emphasizing how the average bit count per symbol closely approaches the entropy. Code word lengths vary based on symbol frequency, with the most probable symbol having the shortest code. Another advantage is the absence of prefixes in the codes, enabling instant decoding.

# References:

[1]:<https://en.wikipedia.org/wiki/Huffman_coding>…[Accessed 5 Jan. 2023, 2:18]

[2]:<https://www.studytonight.com/data-structures/huffman-coding> [Accessed 6 Jan. 2023, 15:12]

[3]:<https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/> [Accessed 7 Jan. 2023, 2:37]

# Appendix:

import java.io.FileReader;

import java.io.BufferedReader;

import java.util.PriorityQueue;

import java.io.IOException;

import java.util.TreeMap;

//Define a class 'Node' to represent character data

class Node\_ {

    Node\_ nleft, nRight;

    double nValue;

    String nCharacter;

    public Node\_(double nValue, String nCharacter) {

        this.nValue = nValue;

        this.nCharacter = nCharacter;

        nleft = null;

        nRight = null;

    }

    public Node\_(Node\_ left, Node\_ right) {

        this.nValue = left.nValue + right.nValue;

        nCharacter = left.nCharacter + right.nCharacter;

        if (left.nValue < right.nValue) {

            this.nRight = right;

            this.nleft = left;

        } else {

            this.nRight = left;

            this.nleft = right;

        }

    }

}

// main class

public class Main {

    static PriorityQueue<Node\_> nodes\_ = new PriorityQueue<>((o1, o2) -> (o1.nValue < o2.nValue) ? -1 : 1);

    static TreeMap<Character, String> hcodes = new TreeMap<>();

    static int i;

    static int length\_array = 0;

    static int Leng\_ofall\_story;

    static int counter\_ [] = new int[8213];

    static char[] array\_of\_character = new char[8213];

    static int[] array\_of\_character\_freq = new int[8213];

    static double codeword\_leng = 0;

    static int num\_BitCodeWord = 0;

    static double entropy1 = 0;

    static String text\_ = "";

    static String encoded\_str = "";

    static String decoded\_str = "";

    static int ASCII[] = new int[8213];

    static final boolean newTxt\_BasedOn\_OldTxt = false;

    // main function

    public static void main(String[] args) throws IOException {

        read\_start\_opration\_text();

        percentageOfCompression();

    }

    // This function reads data from a file

    private static boolean read\_start\_opration\_text() throws IOException {

        int old\_txt\_length = text\_.length();

        text\_ = read\_file\_story("data.txt");    //call function of read all data from file and return to store in text

        if(newTxt\_BasedOn\_OldTxt && (old\_txt\_length != 0 && !checkSimilarity()) ){

            System.out.println("Error: Incompatible characters detected or invalid text length. Please check.\n");

            text\_ = "";

            return true;

        }

        ASCII = new int[8228];

        encoded\_str = "";

        decoded\_str = "";

        nodes\_.clear();

        hcodes.clear();

        calculateEntropy(nodes\_, true);

        create\_QueuNode(nodes\_);

        code\_Word(nodes\_.peek(), "");

        codingStory();

        return false;

    }

     //this function to convert all characters to lower case

    static String read\_file\_story(String filesName) throws IOException {

        BufferedReader buff\_reader = new BufferedReader(new FileReader(filesName));

        try {

            StringBuilder strbli = new StringBuilder();

            String line\_str = buff\_reader.readLine();

            while (line\_str != null) {

                strbli.append(line\_str);

                line\_str = buff\_reader.readLine();

            }

            String str = strbli.toString();

            String lowerStr = str.toLowerCase();

            return lowerStr;

        } finally {

            buff\_reader.close();

        }

    }

 // 3. Calculate the necessary number of bits for encoding the story[using:ASCII representation]

 // 4. Calculate the compression percentage achieved by using:[Huffman encoding compared] to [ASCII representation].

    public static void codingStory() {

        find\_chars\_frequency(text\_);

        encoded\_str = "";

        int s;

        double prop;

        System.out.println(" Character    propability   Codeword           #codeword \n");

        for (int i = 0; i < length\_array; i++) {

            encoded\_str = hcodes.get(array\_of\_character[i]);

            prop = 1.0 \* array\_of\_character\_freq[i] / text\_.length();

            System.out.printf("   '%c'        %f        %s",array\_of\_character[i],prop,encoded\_str);

            s = encoded\_str.length();

            while(s < 25) {

                System.out.printf(" ");

                s++;

            }

            System.out.printf("%d bit \n",encoded\_str.length());

            codeword\_leng = codeword\_leng + (encoded\_str.length() \* 1.0 \* array\_of\_character\_freq[i] / text\_.length());

            num\_BitCodeWord = num\_BitCodeWord + encoded\_str.length() \* array\_of\_character\_freq[i];

        }

        System.out.println("\n=======================================================================");

        System.out.println("The Length of all story is (number of caracter with repeat ): " + Leng\_ofall\_story);

        System.out.println("--------------------------");

        System.out.println("The Number of characters (with out repeat): " + length\_array);

        System.out.println("--------------------------");

        System.out.printf("The Entropy of alphabet is : %.3f\n", entropy1);

        System.out.println("--------------------------");

        System.out.printf("The Average Length Of CodeWord is : %.3f bits/char\n", codeword\_leng );

        System.out.println("--------------------------");

        System.out.println("The Number of Bits CodeWords if useing Huffman is : " + num\_BitCodeWord + " bits");

        System.out.println("--------------------------");

    }

     //this function to generate the[codewords]

    private static void code\_Word(Node\_ node\_, String str) {

        int numOf\_bit = 0;

        if (node\_ != null) {

            if (node\_.nRight != null) {

                code\_Word(node\_.nRight, str + "1");

                numOf\_bit = numOf\_bit + 1;

            }

            if (node\_.nleft != null) {

                code\_Word(node\_.nleft, str + "0");

                numOf\_bit = numOf\_bit + 1;

            }

            if (node\_.nleft == null && node\_.nRight == null) {

                hcodes.put(node\_.nCharacter.charAt(0), str);

            }

        }

    }

  //1. this function to find the [average number of bits]/character for the story

    public static void find\_chars\_frequency(String s) {

        for (i = 0; i < s.length(); i++) {

            counter\_[(int) s.charAt(i)]++;

        }

        int sum\_freq\_all\_letters = 0;

        int index\_ = 0;

        System.out.println("==================================");

        System.out.println(" Character     Frequency \n");

        for (i = 0; i < 256; i++) {

            if (counter\_[i] != 0 && (char) i != '\n') {

                array\_of\_character\_freq[index\_] = (int) counter\_[i];

                array\_of\_character[index\_] = (char) i;

                System.out.println("   " + "'" + array\_of\_character[index\_] + "'" + "         : " +  array\_of\_character\_freq[index\_]);

                sum\_freq\_all\_letters = sum\_freq\_all\_letters + counter\_[i];

                length\_array = length\_array + 1;

                index\_ = index\_ + 1;

            }

        }

        Leng\_ofall\_story=sum\_freq\_all\_letters;

        System.out.println("=====================================================================");

    }

     //2. Find the #entropy of alphabet.

    private static void calculateEntropy(PriorityQueue<Node\_> vector, boolean print\_intervals) {

        if (print\_intervals) {

            for (int i = 0; i < text\_.length(); i++) {

                ASCII[text\_.charAt(i)]++;

            }

        }

        double probabilityValue;

        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 (print\_intervals) {

                    probabilityValue = 1.0 \* ASCII[i] / (text\_.length());

                    entropy1 += -probabilityValue \* Math.log(probabilityValue) / Math.log(2);

                }

            }

        }

    }

    // 3. Calculate the number of bits required to encode the story using ASCII code.

 // 4. Determine the compression percentage achieved by Huffman encoding compared to ASCII codeode.

    static void percentageOfCompression() {

        int ASCII\_N = 8 \* text\_.length();

        System.out.println("The Number Of Bits in Story if using ASCI is " + ASCII\_N + " bits");

        System.out.println("--------------------------");

        double percentageVal = (1.0 \* num\_BitCodeWord / ASCII\_N) \* 100;

        System.out.printf("The percentage of compression is %.2f%%\n" , percentageVal);

        System.out.println("===================================================================");

    }

    private static void create\_QueuNode(PriorityQueue<Node\_> vector) { // in this function to add in Queue

        while (vector.size() > 1) {

            vector.add(new Node\_(vector.poll(), vector.poll()));

        }

    }

 // To checks the similarity of characters in the 'text\_' string with their    corresponding ASCII values.

    private static boolean checkSimilarity() {

        boolean bflag = true;

        for (int j = 0; j < text\_.length(); j++) {

            if (ASCII[text\_.charAt(j)] == 0) {

                bflag = false;

                break;

            }

        }

        return bflag;

    }

}