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:

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 :

1) The frequency for all characters:

```
==================================
 Character      Frequency

    ' '          : 7049
    '!'          : 3
    '"'          : 2
    '''          : 20
    ','          : 436
    '-'          : 89
    '.'          : 414
    ':'          : 2
    ';'          : 26
    '?'          : 1
    'a'          : 2264
    'b'          : 484
    'c'          : 779
    'd'          : 1515
    'e'          : 3887
    'f'          : 794
    'g'          : 620
    'h'          : 2278
    'i'          : 1983
    'j'          : 20
    'k'          : 304
    'l'          : 1127
    'm'          : 678
    'n'          : 2077
    'o'          : 1971
    'p'          : 421
    'q'          : 17
    'r'          : 1481
    's'          : 1795
    't'          : 2937
    'u'          : 800
    'v'          : 179
    'w'          : 788
    'x'          : 34
    'y'          : 356
    'z'          : 61
```

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

Character	propability	Codeword	#codeword
' '	0.186946	111	3 bit
'!'	0.000080	00011111011011	14 bit
'"'	0.000053	00011111011001	14 bit
'''	0.000530	00011111010	11 bit
','	0.011563	000110	6 bit
'-'	0.002360	000111111	9 bit
'.'	0.010980	000100	6 bit
':'	0.000053	00011111011010	14 bit
';'	0.000690	0001111000	10 bit
'?'	0.000027	00011111011000	14 bit
'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
'g'	0.016443	100001	6 bit
'h'	0.060415	1010	4 bit
'i'	0.052591	0110	4 bit
'j'	0.000530	00011111001	11 bit
'k'	0.008062	1011000	7 bit
'l'	0.029889	10001	5 bit
'm'	0.017981	101101	6 bit
'n'	0.055084	0111	4 bit
'o'	0.052273	0011	4 bit
'p'	0.011165	000101	6 bit
'q'	0.000451	00011111000	11 bit
'r'	0.039278	10111	5 bit
's'	0.047605	0010	4 bit
't'	0.077892	1100	4 bit
'u'	0.021217	00001	5 bit
'v'	0.004747	0001110	7 bit
'w'	0.020899	110111	6 bit
'x'	0.000902	0001111001	10 bit
'y'	0.009441	1011001	7 bit
'z'	0.001618	000111101	9 bit

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.

3) 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

4) Other Calculations:

```
========================================================================
The Length of all story is (number of caracter with repeat ): 37692
-------------------------
The Number of characters (with out repeat): 36
-------------------------
The Entropy of alphabet is : 4.172
-------------------------
The Average Length Of CodeWord is : 4.214 bits/char
-------------------------
The Number of Bits CodeWords if useing Huffman is : 158895 bits
-------------------------
The Number Of Bits in Story if using ASCI is 301648 bits
-------------------------
The percentage of compression is 52.68%
========================================================================
```

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],pro
p,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;
    }

}
```