**TEXT COMPRESSION USING HUFFMAN CODING**

**A MINI PROJECT REPORT**

***of***

**BACHELOR OF TECHNOLOGY**

***in***

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

***at***

**DAYANANDA SAGAR UNIVERSITY**

**SCHOOL OF ENGINEERING, BANGALORE-560068**

**IV SEMESTER**

**(Course Code:16CS209 )**

**(Algorithm Design and Applications)**

**DAYANANDA SAGAR UNIVERSITY**

**CERTIFICATE**

This is to certify that the Object Oriented Programming Mini-Project report entitled **“Text Compression”** being submitted by SAHANA M [ENG18CS0240] to Department of Computer Science and Engineering, School of Engineering, Dayananda Sagar University, Bangalore, for the 4th semester B.Tech C.S.E of this university during the academic year 2019-2020.

***Date***:\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Signature of the Faculty in Charge***

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Signature of the Chairman***

**DAYANANDA SAGAR UNIVERSITY**

**CERTIFICATE**

This is to certify that the Object Oriented Programming Mini-Project report entitled **“Text Compression”** being submitted by SANJANA A G [ENG18CS0244] to Department of Computer Science and Engineering, School of Engineering, Dayananda Sagar University, Bangalore, for the 4th semester B.Tech C.S.E of this university during the academic year 2019-2020.

***Date***:\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Signature of the Faculty in Charge***

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Signature of the Chairman***

**DAYANANDA SAGAR UNIVERSITY**

**CERTIFICATE**

This is to certify that the Object Oriented Programming Mini-Project report entitled **“Text compression”** being submitted by RACHIT P [ENG18CS0220] to Department of Computer Science and Engineering, School of Engineering, Dayananda Sagar University, Bangalore, for the 4th semester B.Tech C.S.E of this university during the academic year 2019-2020.

***Date***:\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Signature of the Faculty in Charge***

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Signature of the Chairman***

**DAYANANDA SAGAR UNIVERSITY**

**CERTIFICATE**

This is to certify that the Object Oriented Programming Mini-Project report entitled **“Text Compression”** being submitted by SAHANA H P [ENG18CS0239] to Department of Computer Science and Engineering, School of Engineering, Dayananda Sagar University, Bangalore, for the 4th semester B.Tech C.S.E of this university during the academic year 2019-2020.

***Date***:\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Signature of the Faculty in Charge***

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Signature of the Chairman***

**ACKNOWLEDGEMENT**

We are pleased to acknowledge the **Neeta M S**,**Asso.Prof** Department of Computer Science & Engineeringfor his invaluable guidance, support, motivation and patience during the course of this mini- project work.

We extend our sincere thanks to **Dr. Banga M.K**,**Chairman,** Department of Computer Science & Engineering who continuously helped us throughout the project and without his guidance, this project would have been an uphill task.

We have received a great deal of guidance and co-operation from our friends and we wish to thank one and all that have directly or indirectly helped us in the successful completion of this mini-project work.

**Team Members** **SAHANA H P[ENG18CS0239]**

**SANJANA A G[ENG18CS0244]**

**SAHANA M[ENG18CS240]**

**RACHIT P[ENG18CS0220]**

**ABSTRACT**

The human world is currently undergoing intense technological revolution. There is a need for maintaining a large volume of data or information and storing them. But maintaining such large data is not an easy task as the time and storage available is limited. Hence, storing and managing the data has to be done in an efficient manner. Therefore, data compression is one of the techniques used to store data in a limited space.Data compression can be viewed as a branch of information theory in which the primary objective is to minimize the amount of data. Data compression has important applications in the areas of data transmission and data storage. Data compression can be achieved through various data compression algorithms.

This project uses the Huffman code for data, specifically text compression. The Huffman algorithm is an encoding algorithm for lossless text compression under greedy technique. The term encoding refers to the use of a variable length code for every character that has been derived in a particular way based on the estimated probability of occurrences for each possible value of the source character.

**TABLE OF CONTENTS**

**CHAPTER NO.** **TITLE**   **PAGE NO.**

1. Introduction 1

1.1 Problem Statement

1.2 Objectives of the project

2. System Requirements 2

2.1 Functional Requirements

2.2 Software and Hardware Requirements

3. System Design 3

3.1 Architecture/Data Flow Diagrams

3.2 Modules 4

4. System Implementation 5

4.1 Module Description

4.2 Pseudocode 6

5. Output Screen Shots 9

6. Conclusion 11

Reference 12

**1.INTRODUCTION**

Data compression is known for reducing storage and communication costs. Usually many programming languages use ASCII (American Standard Code for Information Interchange) coding for characters. Huffman coding is one of the techniques that can be used for data compression and works by creating a binary tree of nodes and assigning the characters the binary code of 0’s and 1’s and significantly reducing the values of the characters and compressing the text data.

**1.1 PROBLEM STATEMENT**

The text compression problem using Huffman code aims to reduce the number of bits used by each character.

Each character having ASCII value uses fixed 8 bits which is not very efficient hence the text has to be compressed in such a way that it uses fewer bits of data. This is achieved using huffman codes obtained by prioritising the characters based on the occurrences of each character and encoding the character string which is compressed and gives the encoded compressed bits as an output.

**1.2 OBJECTIVE**

The principle is to use a lower number of bits to encode the data that occurs more frequently. Based on the symbols and their frequencies, the goal is to construct a rooted binary tree where the characters are the labels of the leaves and assign codes to characters. It provides lossless data compression which means there is no data lost. Huffman coding is extensively used to compress bit strings. The text after compression uses fewer bits when compared to the ASCII bits for each character.

1

**2.SYSTEM REQUIREMENTS**

**2.1 FUNCTIONAL REQUIREMENTS**

The input character strings taken from the user. The making a priority queue of the characters on the basis of the frequency keeping the highest frequency as lowest priority. A binary tree has to be built one by one considering the first two values in the priority queue. After the binary tree is built the accumulates are assigned 0’s and 1’s and the binary code for the character string is encoded. This string must contain fewer bits.

**2.2 SOFTWARE AND HARDWARE REQUIREMENTS**

**SOFTWARE REQUIREMENTS**

* Programming language : C programming
* Operating system : Windows 10
* Compiler used : GNU GCC Compiler

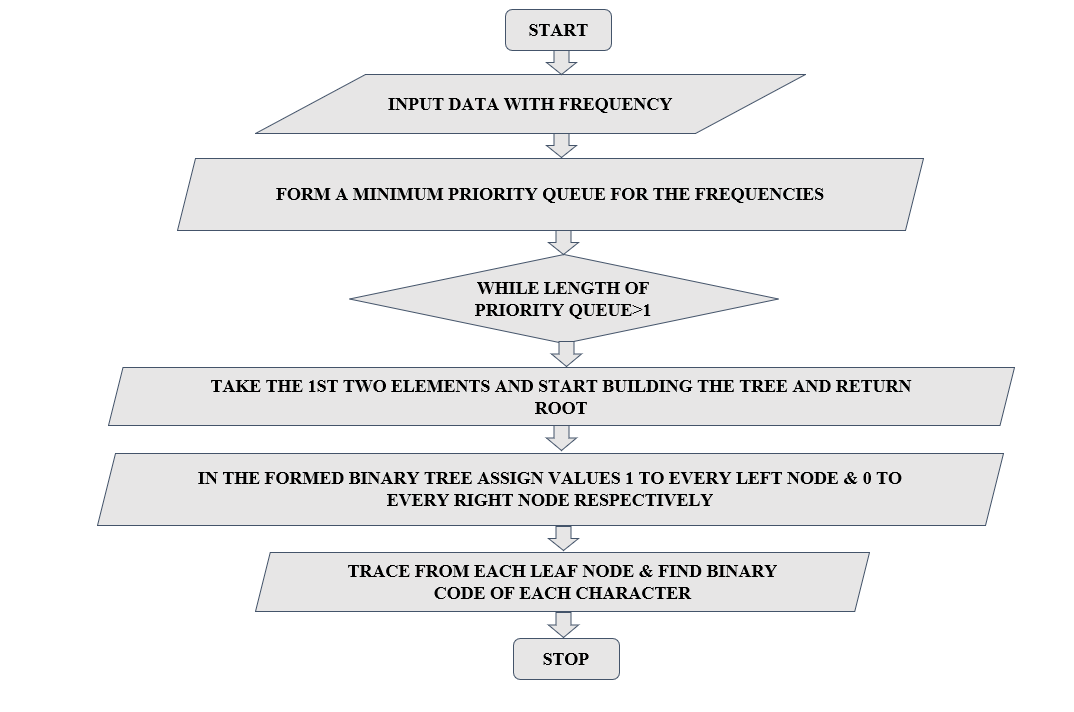
**HARDWARE REQUIREMENTS**

* Processor : intel(R) Core i5-8250U CPU
* RAM : 8.00GB

2

**3. SYSTEM DESIGN**

**3.1 ARCHITECTURE/ DATA FLOW DIAGRAM**

****

3

**3.2 MODULES**

* node new\_node(int freq, char c, node a, node b)
* void qinsert(node n)
* node qremove()
* void init(const char \*s)
* void build\_code(node n, char \*s, int len)
* void encode(const char \*s, char \*out)
* void decode(const char \*s, node t)

4

**4. SYSTEM IMPLEMENTATION**

**4.1 MODULE DESCRIPTION**

* node new\_node(int freq, char c, node a, node b) - Reads through the given input and forms list of characters and frequency.
* void qinsert(node n) - This function creates a priority queue. Higher frequency has lower priority move up lower frequency. Compare frequency of the new node with the parent's frequency.
* node qremove() - Remove the top element and move up other elements.
* void init(const char \*s) - count frequency for each character and completes the heap while merging nodes starting from the lower frequency nodes.This is done in the following steps.

(1) Remove top two nodes which have the highest priority (lowest frequency)

(2) Make a new one with the two removed nodes while adding the two freq.

(3) When we make the new node, it remembers its children as left/right nodes

(4) Keep merging the nodes until there is only one node left.

* void build\_code(node n, char \*s, int len) - This function traverses the tree and assigns the accumulates as 0’s to right nodes and 1’s to the left nodes vice versa also works well.
* void encode(const char \*s, char \*out) - encodes the final binary codes of each character.
* void decode(const char \*s, node t)-decodes the given binary code for the string and gives back the same string if the code is correct.

5

**4.2 PSEUDO CODE**

typedef struct node\_t{

struct node\_t \*left, \*right;

int freq;

char c;

} \*node;

node new\_node(int freq, char c, node a, node b)

{

node n = pool + n\_nodes++;

if (freq) n->c = c, n->freq = freq;

else

n->left = a, n->right = b;

n->freq = a->freq + b->freq;

return n;

}

void qinsert(node n)

{

int j, i = qend++;

while ((j = i / 2)) {

if (q[j]->freq <= n->freq)break;

[i] = q[j], i = j;

}

q[i] = n;

}

node qremove()

{

int i, l;

node n = q[i = 1];

if (qend < 2)

return 0;

qend--;

while ((l = i \* 2) < qend) {

if (l + 1 < qend && q[l + 1]->freq < q[l]->freq) l++;

q[i] = q[l], i = l;

}

q[i] = q[qend];

return n;

}

void build\_code(node n, char \*s, int len)

{

static char \*out = buf;

if (n->c)

s[len] = 0;

strcpy(out, s);

code[n->c] = out;

out += len; return;}

s[len] = '0'; build\_code(n->left, s, len + 1);

s[len] = '1'; build\_code(n->right, s, len + 1); 6

}

void init(const char \*s)

{

int i, freq[128] = {0};

char c[16];

while (\*s) freq[(int)\*s++]++;

for (i = 0; i < 128; i++) {

if (freq[i]) qinsert(new\_node(freq[i], i, 0, 0));

}

while (qend > 2) {

qinsert(new\_node(0, 0, qremove(), qremove()));

}

build\_code(q[1], c, 0);

}

void encode(const char \*s, char \*out)

{

while (\*s) {

strcpy(out, code[\*s]);

out += strlen(code[\*s++]);

}

}

void decode(const char \*s, node t)

{

node n = t;

while (\*s) {

if (\*s++ == '0') n = n->left;

else n = n->right;

if (n->c) putchar(n->c), n = t;

}

putchar('\n');

if (t != n) printf("garbage input\n");

}

int main(void)

{

char \*str;

printf("ENTER THE STRING - ");

scanf("%[^\n]%\*c",str);

printf("%s",str);

char buf[1024];

init(str);

for (int i = 0; i < 128; i++)

if code[i] ,printf("'%c': %s\n", i, code[i]);

7

encode(str, buf);

printf encoded string

printf("decoded: ");

print decoded string

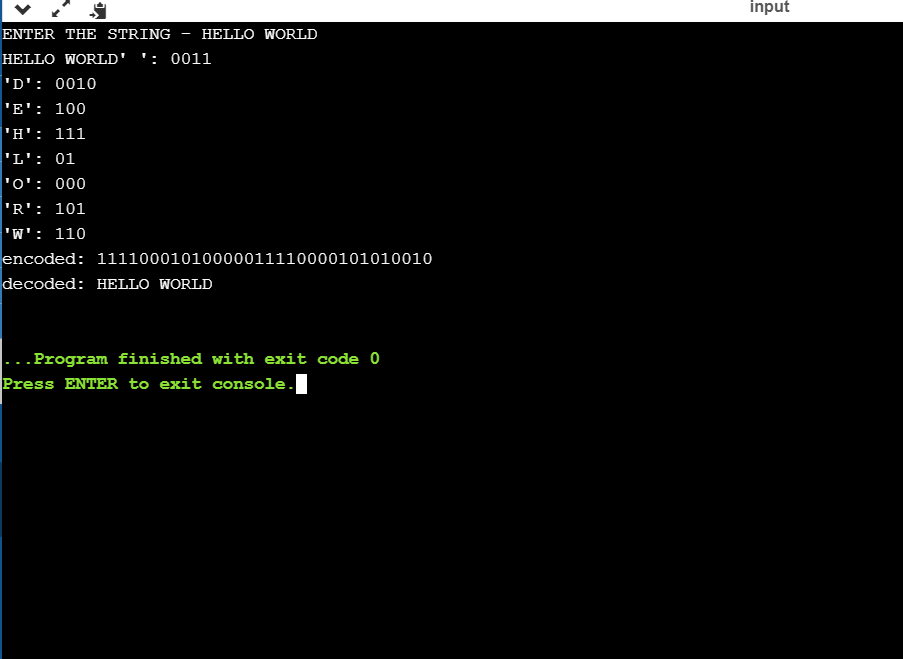
return 0;

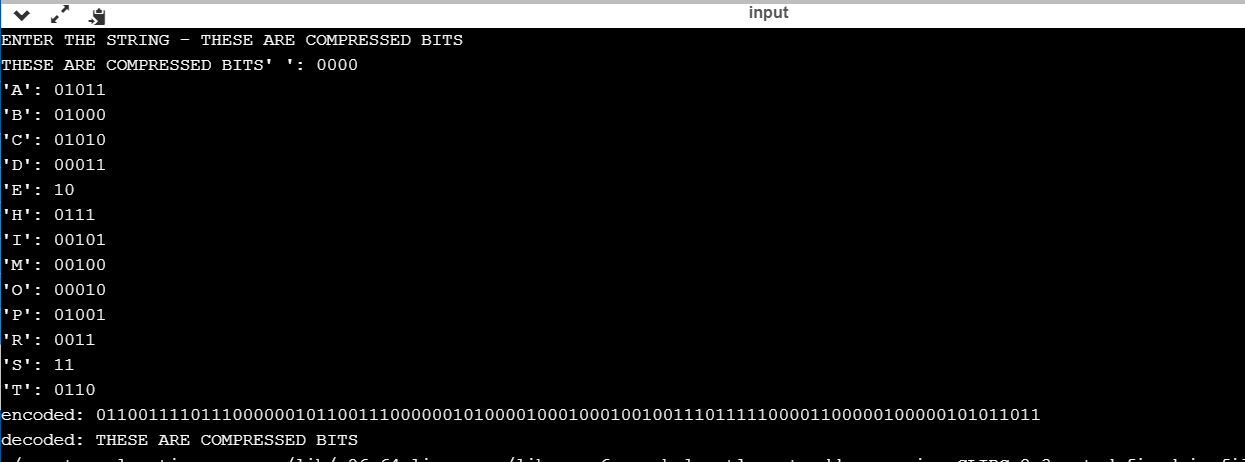
}

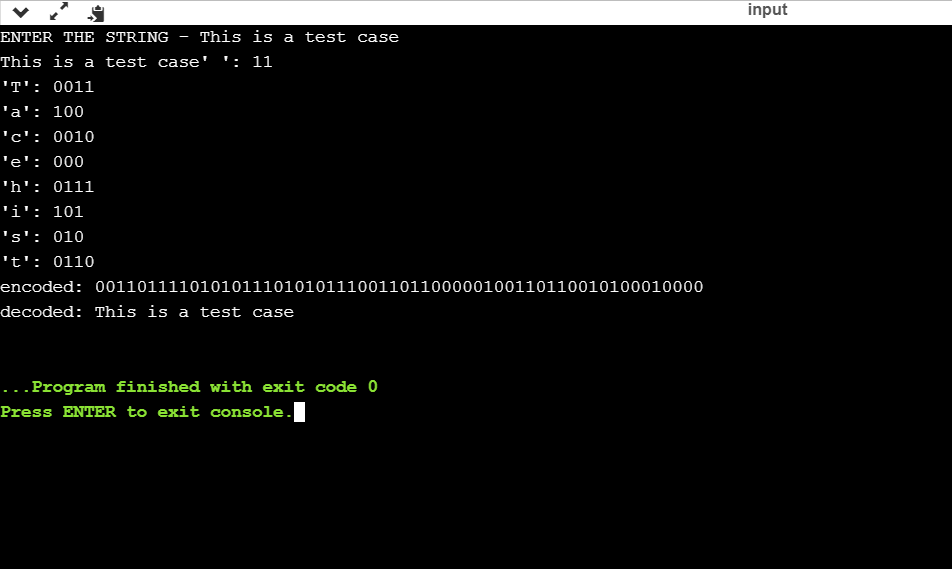
8

**5.OUTPUT SCREENSHOTS**

**EXAMPLE 1:**

****

**EXAMPLE 2:** 9

**EXAMPLE 3:**

10

**6. CONCLUSION**

Huffman coding is extensively used to compress bit strings representing text and it also plays an important role in compressing audio and image files. Our project focuses on reducing the number of bits used by the characters significantly with changing any data during the process. It gives an average code word length that is approximately near the entropy of the source. Lossless compression techniques reconstruct the original data from the compressed text without any loss of data. The frequently used symbols most likely to be in the data uses the least number of bits in the encoding, making the coding efficient unlike the ASCII values that consumes 8bits per character. The project can further improvise and can be implemented to compress large data files and image files making the system of storing and transferring data more efficient supporting the widely growing technological world.

11

**REFERENCES**

[1]-<https://courses.cs.washington.edu/courses/cse326/10wi/lectures/lec24/lec24-10wi-Huffman.pdf>

[2]-<https://www2.cs.duke.edu/csed/poop/huff/info/>

[3]-<https://freecontent.manning.com/data-compression-with-huffmans-algorithm/>

12