NEW HORIZON
COLLEGE OF ENGINEERING
Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade, Accredited by NBA

TEXT COMPRESSION USING HUFFMAN CODING

A MINI PROJECT REPORT

Submitted by

J KARTHIK SURYA

1NH19ME047

In partial fulfillment for the award of the degree of

DEGREE OF BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

ABSTRACT

Data reduction is one of the data preprocessing techniques which can be applied to obtain a reduced representation of the data set that is much smaller in volume, yet closely maintains the integrity of the original data.

That is, mining on the reduced data set should be more efficient yet produce the same analytical results. Data compression is useful, where encoding mechanisms are used to reduce the data set size.

In data compression, data encoding or transformations are applied so as to obtain a reduced or compressed representation of the original data.

Huffman coding is a successful compression method used originally for text compression. Huffman's idea is, instead of using a fixed-length code such as 8 bit extended ASCII or DBCDIC for each symbol, to represent a frequently occurring character in a source with a shorter codeword and to represent a less frequently occurring one with a longer codeword.

Huffman coding is a very simple method to compress the memory that has been allocated for the text specified by us. It uses a binary trees and linked list along with arrays for general usage.

We first enter the variables along with their frequency of occurrence in the increasing order. Taking these values, we create a binary tree. We then sort the elements from a variable having the least frequency to the greatest frequency.

This forms the tree and then we add the frequencies of 2 elements having the least frequency and add them back to the tree. We sort the tree again until there is a complete binary tree.

Traversing tree is simple, the left edge traversal get the value 0 and the right edge traversal gets the value 1. The path to a particular variable is the encoded/compressed value of the variable.

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be impossible without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

I have great pleasure in expressing gratitude to Dr. Mohan Manghnani, Chairman of New Horizon Educational Institutions for providing necessary infrastructure and creating good environment.

I take this opportunity to express my profound gratitude to Dr. Manjunatha, Principal NHCE, for his constant support and encouragement.

I would also like to thank Dr. B. Rajalakshmi, Professor and Head, Department of Computer Science and Engineering, for her constant support.

I express my gratitude to Mr./Ms./Dr. Faculty Name, Designation, my project guide, for constantly monitoring the development of the project and setting up precise deadlines. Her valuable suggestions were the motivating factors in completing the work.

J KARTHIK SURYA (1NH19ME047)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 PROBLEM DEFINITION

Compression refers to reducing the quantity of data used to represent a file, image or video content without excessively reducing the quality of the original data. It also reduces the number of bits required to store and/or transmit digital media. To compress something means that you have a piece of data and you decrease its size.

There are different techniques who to do that and they all have their own advantages and disadvantages. One trick is to reduce redundant information, meaning saving sometimes once instead of 6 times. Another one is to find out which parts of the data are not really important and just leave those away.

Arithmetic coding is a technique for lossless data compression. It is a form of variable-length entropy encoding. Arithmetic coding encodes the entire message into a single number, a fraction n where ($0.0 \le n < 1.0$) but where other entropy encoding techniques separate the input message into its component symbols and replace each symbol with a code word.

Run length encoding method is frequently applied to images or pixels in a scan line. It is a small compression component used in JPEG compression Huffman coding is a loseless data compression technique.

Huffman coding is based on the frequency of occurrence of a data item i.e. pixel in images. The technique is to use a lower number of bits to encode the data in to binary codes that occurs more frequently. It is used in JPEG files.

Data compression is one of the most widespread applications in computer technology. Algorithms and methods that are used depend strongly on the type of data, i.e. whether the data is static or dynamic, and on the content that can be any combination of text, images, numeric data or unrestricted binary data.

The compression and decompression techniques are playing main role in the data transmission process. The best compression techniques among the three algorithms have to be analyzed to handle text data file. This analysis may be performed by comparing the measures of the compression and decompression

1.2 OBJECTIVES

Huffman coding is extensively used to compress bit strings representing text and it also plays an important role in compressing audio and image files. Based on the symbols and their frequencies, the goal is to construct a rooted binary tree where the symbols are the labels of the leaves.

1.3 METHODOLOGY TO BE FOLLOWED

Huffman coding is based on the frequency of occurrence of a data item (pixel in images). The principle is to use a lower number of bits to encode the data that occurs more frequently. Codes are stored in a Code Book which may be constructed for each image or a set of images. In all cases the code book plus encoded data must be transmitted to enable decoding.

1.4 EXPECTED OUTCOMES

Huffman coding is an entropy encoding algorithm used for lossless data compression in computer science and information theory. The term refers to the use of a variable-length code table for encoding a source symbol (such as a character in a file) where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol.

Huffman coding uses a specific method for choosing the representation for each symbol, resulting in a prefix-free code (that is, the bit string representing some particular symbol is never a prefix of the bit string representing any other symbol) that expresses the most common characters using shorter strings of bits than are used for less common source symbols.

Huffman was able to design the most efficient compression method of this type: no other mapping of individual source symbols to unique strings of bits will produce a smaller average output size when the actual symbol frequencies agree with those used to create the code. A method was later found to do this in linear time if input probabilities (also known as weights) are sorted.

1.5 HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements

- Processor : X86 Compatible processor with 1.7 GHz Clock speed
- RAM : 512 MB or more
- Hard disk : 500 GB

Software Requirements

- Visual Studio Community 2017
- Windows 10 Operating System
- Data structures like Stack and Hash Table

CHAPTER 2

DATASTRUCTURES

2.1. Linked list:

For implementation of dictionary which makes note of all the characters present and its frequency which will be used to build the tree.

2.2. Binary Tree:

For obtaining the Huffman Code for each character and to the build Huffman table for a particular text.

2.3. Arrays :

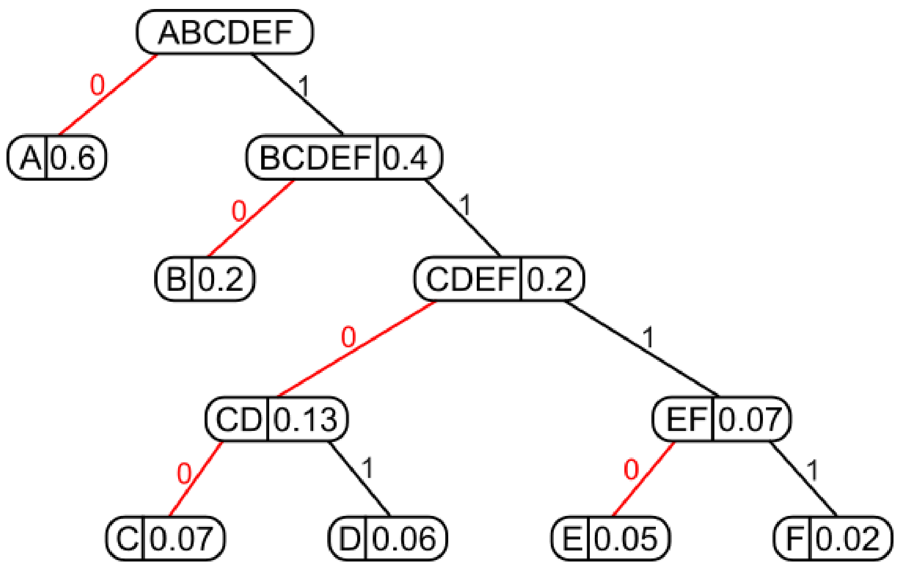
General usage….

2.4. GRAPHS:

Specifically, given a graph G = (V, E), where V is the set of vertices and E is the set of edges, and |V| = n, we propose methods to reduce the space requirements of the graph by compressing the adjacency representation of the same.

The proposed methods show up to 80% reduction is the space required to store the graphs as compared to using the adjacency matrix. The methods can also be applied to other representations as well.

The proposed techniques help solve the issues related to computing on the graphs on resources limited compute nodes, as well as reduce the latency for transfer of data over the network in case of distributed computing.



CHAPTER 3

DESIGN

3.1. DESIGN GOALS

Our main objective here is to reduce the space allocated for the storage of the variable. To achieve this, we must first create a sorted binary tree.

Add the 2 elements bearing the least frequency and add to the tree and sort them again and finally get the binary tree for the given variables.

Path value traversing to the left is given as 0 and path value traversing to the right is given as 1 and hence the path values to reach the variables are the encoded /compressed code for the variable.

3.2 ALGORITHM / PSEUDOCODE

```
Input()

Initialize Hsize to 0

Step 1: START

Step 2: increment Hsize by 1

Step 3: assign Hsize to now.

Step 4: BEGIN while

        check if H[now/2]->freq > element->freq

        If yes, swap the elements

        If no, Input the element to the next node.

Step 5: END while

Step 6: END


Print()

Step 1: BEGIN

Step 2: BEGIN if
Step 3: checks if the llink and the rlink of temp = NULL Step

4: If yes, print the character and its coded value.

Step 5: If no, assigns left traversal as ‘0’ and right traversal as ‘1’

Step 6: END if


Main()

Step 1: allocates the value of H[0]

H[0]=(node *)malloc(sizeof(node));

Step 2: We enter the number of char

Step 3: We enter the char along with their freq

Step 4: BEGIN for loop

Step 5: we send these values to Input() to be sorted and inserted in to the H

Step 6: after sorting we take the first 2 elements of the tree and add their

freq and it back to the H to be sorted and stored.

Step 7: End for loop

Step 8: END
```

CHAPTER 4

IMPLEMENTATION

```
void Input(node*element)

{

   Hsize++;

   H[Hsize]=element;

   int now=Hsize;

   while(H[now/2]->freq>element->freq)

   {

      H[now]=H[now/2];

      now/=2;

   }

   H[now]=element;

}
```

The above function takes the value and from various other function and sorts them and places them in the H

```
node*RemoveMin()

{

      node * minElement,*lastElement;

      int child,now;

      minElement = H[1];
```

```
lastElement = H[Hsize--];

for(now = 1; now*2 <= Hsize;now = child)

{

    child = now*2;

       if(child != Hsize && H[child+1]->freq < H[child] -> freq )

      {

            child++;

      }

      if(lastElement -> freq > H[child] -> freq)

      {

            H[now] = H[child];
      }

      Else

      {

            break;

      }

}

H[now] = lastElement;

return minElement;
```

```
}

void print(node *temp,char *code)

{

        if(temp->left==NULL && temp->right==NULL)

        {
                printf("char %c code %s\n",temp->ch,code);
                return;

        }

        int length = strlen(code);

        char leftcode[10],rightcode[10];

        strcpy(leftcode,code);

        strcpy(rightcode,code);

        leftcode[length] = '0';

        leftcode[length+1] = '\0';

        rightcode[length] = '1';

        rightcode[length+1] = '\0';

        print(temp->left,leftcode);

        print(temp->right,rightcode);

}
```

```
int main()
{
       H[0] = (node *)malloc(sizeof(node));
        H[0]->freq = 0;
        int n ;
        printf("Enter the no of char: ");
        scanf("%d",&n);
        printf("Enter the characters and their freq:  ");
        char ch;
        int freq,i;
        for(i=0;i<n;i++)
        {
                scanf(" %c",&ch);
                scanf("%d",&freq);
                node * temp = (node  *) malloc(sizeof(node));
                temp -> ch  = ch;
                temp -> freq = freq;
                temp -> left  = temp -> right = NULL;
```

```
    Input(temp);
}

        if(n==1)
    {
        printf("char %c code 0\n",ch);

        return 0;

    }

    for(i=0;i<n-1 ;i++)

    {
        node * left = RemoveMin();

        node * right = RemoveMin();

        node * temp = (node *) malloc(sizeof(node));

        temp -> ch = 0;

        temp -> left = left;

        temp -> right = right;

        temp -> freq = left->freq + right -> freq;

        Input(temp);
    }

    node *tree = RemoveMin();

    char code[10];

    code[0] = '\0';
```

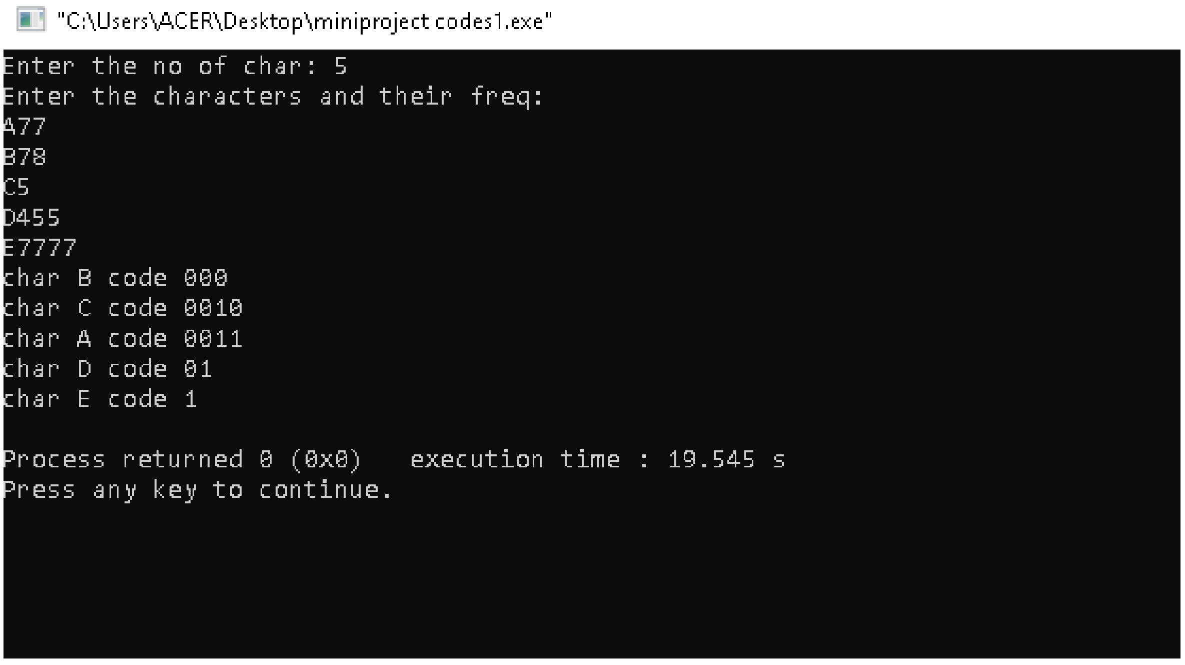
```
print(tree,code);

}
```

This function allocates the right and left traversal of the sorted binary tree to 1 and 0 respectively.

CHAPTER 5

RESULTS

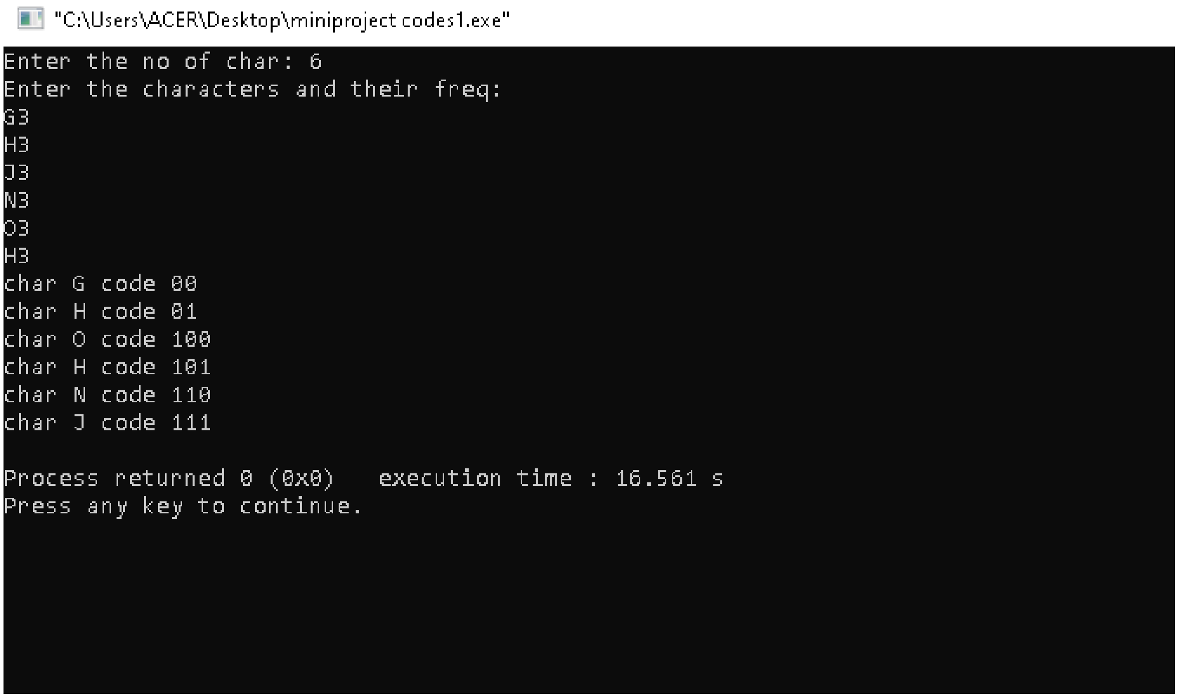
Sample output 1



"C:\Users\ACER\Desktop\miniproject codes1.exe"

```
Enter the no of char: 5
Enter the characters and their freq:
A77
B78
C5
D455
E7777
char B code 000
char C code 0010
char A code 0011
char D code 01
char E code 1

Process returned 0 (0x0)   execution time : 19.545 s
Press any key to continue.
```

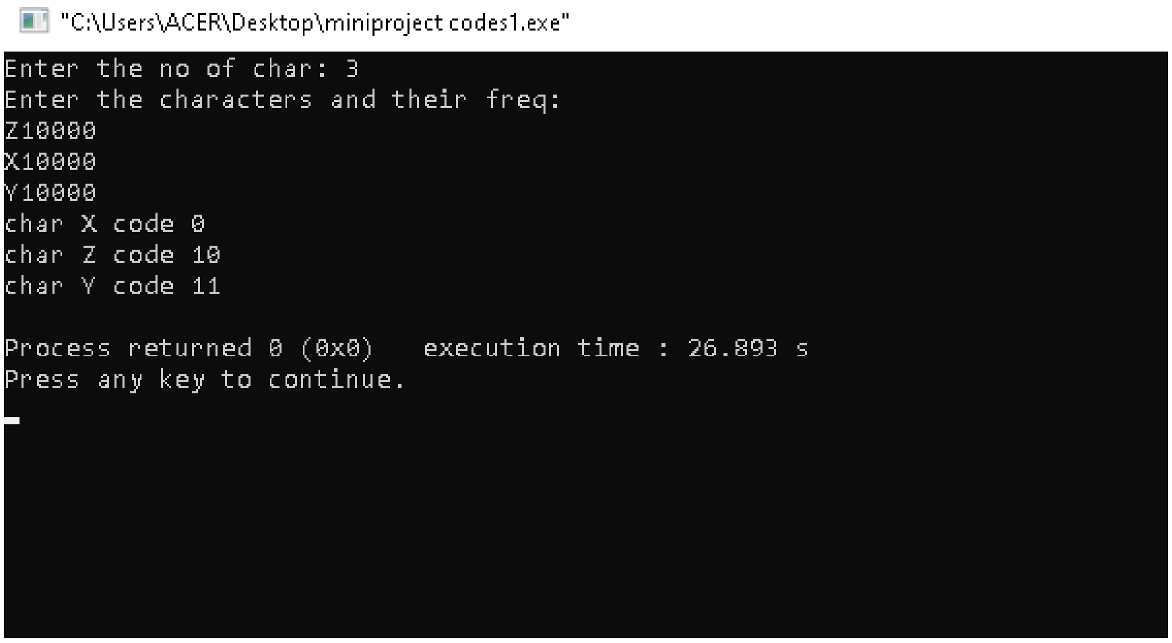
Sample output 2



"C:\Users\ACER\Desktop\miniproject codes1.exe"

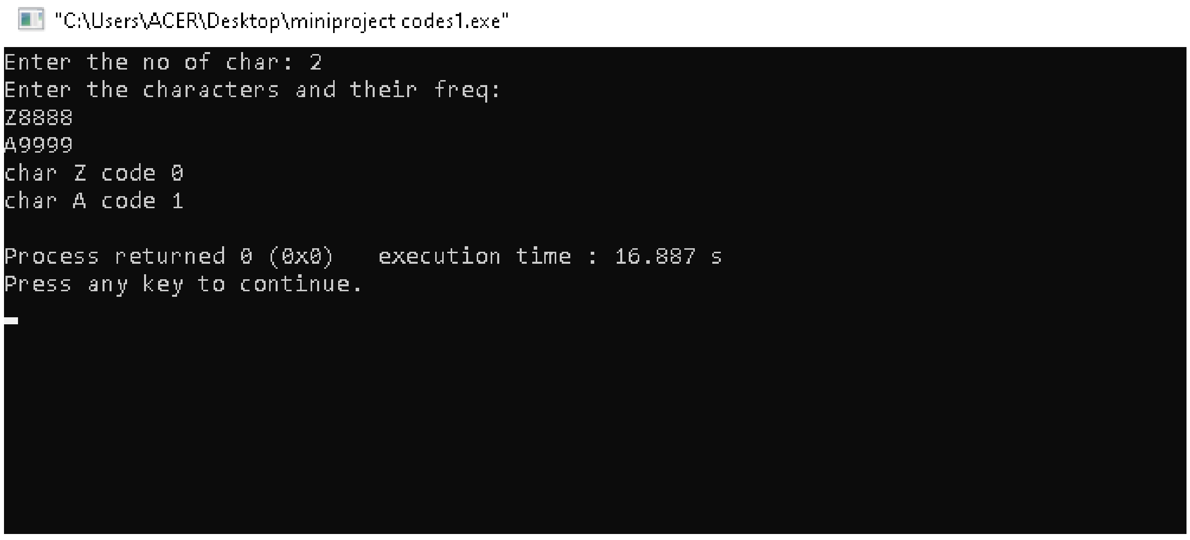
```
Enter the no of char: 6
Enter the characters and their freq:
G3
H3
J3
N3
O3
H3
char G code 00
char H code 01
char O code 100
char H code 101
char N code 110
char J code 111

Process returned 0 (0x0)   execution time : 16.561 s
Press any key to continue.
```

SAMPLE OUTPUT 3


"C:\Users\ACER\Desktop\miniproject codes1.exe"

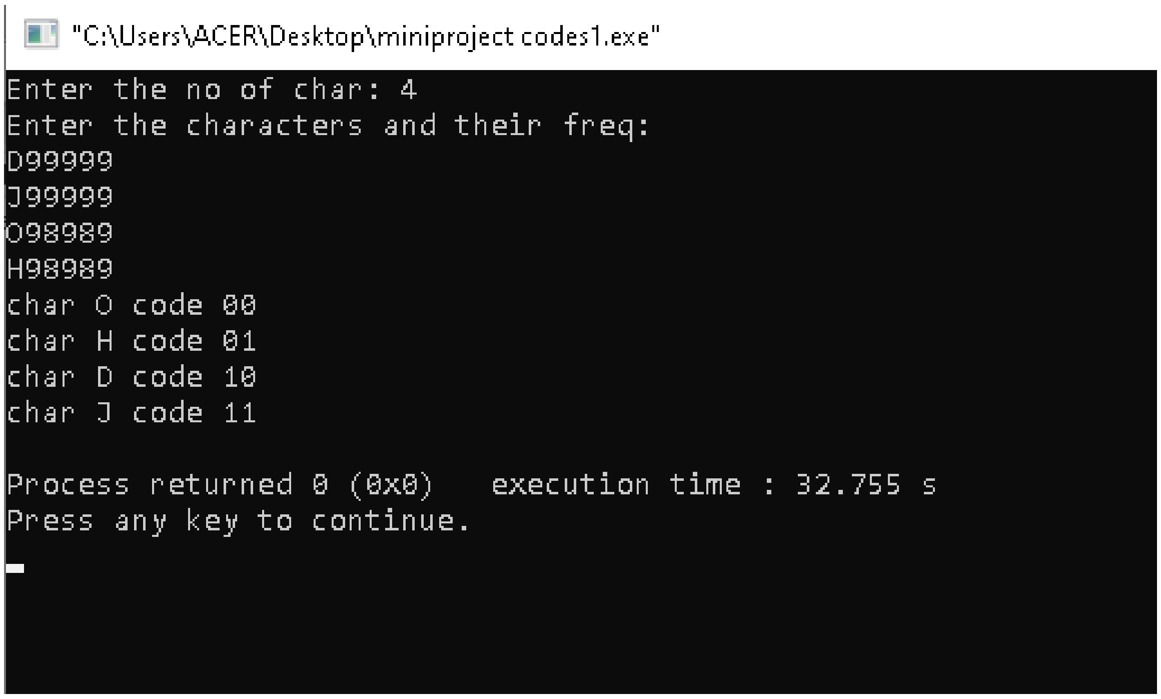
```
Enter the no of char: 3
Enter the characters and their freq:
Z10000
X10000
Y10000
char X code 0
char Z code 10
char Y code 11

Process returned 0 (0x0)   execution time : 26.893 s
Press any key to continue.
```

SAMPLE OUTPUT 4


"C:\Users\ACER\Desktop\miniproject codes1.exe"

```
Enter the no of char: 2
Enter the characters and their freq:
Z8888
A9999
char Z code 0
char A code 1

Process returned 0 (0x0)   execution time : 16.887 s
Press any key to continue.
```

SAMPLE OUTPUT 5



"C:\Users\ACER\Desktop\miniproject codes1.exe"

```
Enter the no of char: 4
Enter the characters and their freq:
D99999
J99999
O98989
H98989
char O code 00
char H code 01
char D code 10
char J code 11

Process returned 0 (0x0)   execution time : 32.755 s
Press any key to continue.
```

.

CHAPTER 6

CONCLUSION

I have studied various techniques for compression and compare them on the basis of their use in different applications and their advantages and disadvantages. I have concluded that arithmetic coding is very efficient for more frequently occurring sequences of pixels with fewer bits and reduces the file size dramatically.

RLE is simple to implement and fast o execute. LZW algorithm is better to use for TIFF, GIF and Textual Files. It is easy to implement, fast and lossless algorithm whereas Huffman algorithm is used in JPEG compression. It produces optimal and compact code but relatively slow. Huffman algorithm is based on statistical model which adds to overhead.

REFERENCES

[1] Darrel Hankersson, Greg A. Harris, and Peter D. Johnson Jr.Introduction to Information Theory and Data Compression. CRC Press, 1997.

[2] Gilbert Held and Thomas R. Marshall. Data and Image Compression: Tools and Techniques

[3] http://en.wikipedia.org/wiki/Discrete_Fourier_transform

[4] Terry Welch, "A Technique for High-Performance Data Compression", Computer, June 1984

[5] http://www.The LZW compression algorithm.htm

[6] Dzung Tien Hoang and Jeffery Scott Vitter .Fast and Efficient Algorithms for video Compression and Rate Control,June 20,1998.

[7] http://www.Howstuffworks How File Compression Works.htm.

[8] http://www .H_261 - Wikipedia, the free encyclopedia.htm

[9] http://en.wikipedia.org/wiki/Entropy_encoding

[10] http://www .JPEG - Wikipedia, the free encyclopedia.htm

[11] http://www.rz.go.dlr.de:8081/info/faqs/graphics/jpeg1.html

[12] http://www.amara.com/IEEEwave/IEEEwavelet.html

[13] http://cm.bell-labs.com/who/jelena/Wavelet/w_applets.html

[14] http://en.wikipedia.org/wiki/Lossless_JPEG

[15] http://en.wikipedia.org/wiki/Arithmetic_coding

[16] http://fly.cc.fer.hr/~eddie/Sem_DPCM.htm

[17] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, May 1977

[18] Rudy Rucker, "Mind Tools", Houghton Mifflin Company, 1987

[19] http://www.dogma.net/markn/index.html

[20] Huffman's original article: D.A. Huffman, "[3]" (PDF), Proceedings of the I.R.E., sept 1952, pp 1098-1102 [21] Background story: Profile: David A. Huffman, Scientific American, Sept. 1991, pp. 54-58

[22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 16.3, pp.385–392.

[23] http://www.Huffman coding - Wikipedia, the free encyclopedia.htm

[24] http://en.wikipedia.org/wiki/Huffman_coding

[25] http://en.wikipedia.org/wiki/Adaptive_Huffman_coding [26] Compressed Image File Formats: JPEG, PNG, GIF,XBM, BMP, John Miano, August 1999 [27] Introduction to Data Compression, Khalid Sayood, Ed Fox (Editor), March 2000