VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi-560014

File Structure

Mini Project Report

On

"FILE COMPRESSOR"

Submitted in partial fulfillment of the requirement of VI semester File Structure Laboratory

Submitted by,

RONIT ASAWA	**1DT19IS115**
SAPNIL DUTTA	**1DT19IS119**

Under the guidance of

Mrs. Neha Jadhav
Asst. Professor
Dept. of ISE
DSATM, Bangalore.

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT
(Affiliated to Visvesvaraya Technological University, Belagavi & Approved by AICTE, New Delhi)
(Accredited by NBA, New Delhi)
Opp. Art of Living, Udayapura, Kanakapura Road, Bangalore- 560082

2021-22

DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT

(Affiliated to Visvesvaraya Technological University, Belagavi & Approved by AICTE, New Delhi)
(Accredited by NBA, New Delhi)
Opp. Art of Living, Udayapura, Kanakapura Road, Bangalore- 560082

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

Certificate

This is to certify that the mini-project work entitled **"FILE COMPRESSOR"** is carried out by **RONIT ASAWA(1DT19IS115) and SAPNIL DUTTA(1DT19IS119)** in partial fulfillment of the requirement of VI semester File Structure Laboratory in **Information Science and Engineering** of the **Visvesvaraya Technological University, Belagavi** during the year 2022-2023. It is certified that all the corrections/ suggestions indicated for the given internal assessment have been incorporated in the report. This report has been approved as it satisfies the academic requirements with respect to the mini-project work.

Signature of the Guide

Mrs. Neha Jadhav
Asst. Professor, Dept. of ISE
DSATM, Bangalore.

Signature of the HOD

Dr. Sumithra Devi K A
Dean Academics, Professor & Head
Dept. of ISE
DSATM, Bangalore.

External Viva

Name of the Examiners **Signature with date**

1.

2.

ACKNOWLEDGEMENT

I/We express my/our profound gratitude to **Dr. Ravishankar.M ,Principal, DSATM, Bangalore,** for providing the necessary facilities and an ambient environment to work.

I/We am/are grateful to **Dr. Sumithra Devi K A, Dean Academics, Professor & Head, Department of Information Science and Engineering, DSATM, Bangalore,** for her valuable suggestions and advice throughout my/our work period.

I would like to express my deepest gratitude and sincere thanks to our guide **Mrs. Neha Jadhav, Assistant Professor, Department of Information Science and Engineering, DSATM, Bangalore,** for her keen interest and encouragement in the project whose guidance made the project into reality.

I/We would like to thank all the staff members of **Department of Information Science and Engineering** for their support and encouragement during the course of this project.

Definitely most, I/We would like to thank my/our parents, all my family members and friends, without whose help and encouragement this project would have been impossible

Ronit Asawa 1DT19IS115
Sapnil Dutta 1DT19IS119

ABSTRACT

The Project “Data Compression Techniques is aimed at developing programs that transform a string of characters in some representation (such as ASCII) into a new string (of bits, for example) which contains the same information but whose length is as small as possible. Compression is useful because it helps reduce the consumption of resources such as data space or transmission capacity. The design of data compression schemes involves trade-off s among various factors, including the degree of compression, the amount of distortion introduced (e.g., when using lossy data compression), and the computational resources required to compress and uncompressed the data.

Many data processing applications require storage of large volumes of data, and the number of such applications is constantly increasing as the use of computers extends to new disciplines. Compressing data to be stored or transmitted reduces storage and/or communication costs. When the amount of data to be transmitted is reduced, the effect is that of increasing the capacity of the communication channel. Similarly, compressing a file to half of its original size is equivalent to doubling the capacity of the storage medium. It may then become feasible to store the data at a higher, thus faster, level of the storage hierarchy and reduce the load on the input/output channels of the computer system.

\\

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The Project "Data Compression Techniques is aimed at developing programs that transform a string of characters in some representation (such as ASCII) into a new string (of bits, for example) which contains the same information but whose length is as small as possible. Compression is useful because it helps reduce the consumption of resources such as data space or transmission capacity. The design of data compression schemes involves trade-offs among various factors, including the degree of compression, the amount of distortion introduced (e.g., when using lossy data compression), and the computational resources required to compress and uncompress the data.

Most compression techniques work by reducing the space redundant information in a file takes up. The more redundancy the compression algorithm detects, the smaller the compressed file becomes. Text files, for example, may have many repeated words or letter combinations that can produce significant compression–as much as 80%, in some cases.

Databases and spreadsheets often also make good candidates for file compression because they, too, typically have repeated content. Conversely, files that have already been compressed, such as MP3s and JPEGs, have low redundancy. Compressing them further yields results only a few percent smaller than the originals–in some cases, they may become slightly larger when compressed, since the compression can add a small amount of management data to the file.

It is frequently convenient to package many files and/or folders into a single compressed file, such as for emailing a collection of files or distributing a complex software application. This packaged collection of files is called an archive. Some compression programs also let you combine multiple files together, providing the dual benefit of smaller space and archival packaging. Other programs, particularly in the Linux/Unix domain, only handle compression of one file at a time.

CHAPTER 2

REQUIREMENTS ANALYSIS

The requirement analysis specifies the requirements needed to develop a graphic project. In this phase, we collect the requirements needed for designing the project. The requirements collected are then analyzed and carried to the next phase.

2.1 SOFTWARE REQUIREMENTS:

1. Operating System: Windows 7 or above
2. Programming Language: Python 3.8
3. Front-end Development: Python – Tkinter and Turtle Graphics
4. Back-end Development: File Handling using Python.
5. IDE used: Visual Studio Code

2.2 HARDWARE REQUIREMENTS

1. Processor – Pentium IV or above
2. RAM – 2 GB or more
3. Hard disk – 3 GB or more

CHAPTER 3

DESIGN

3.1 Use Case Diagram

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well.

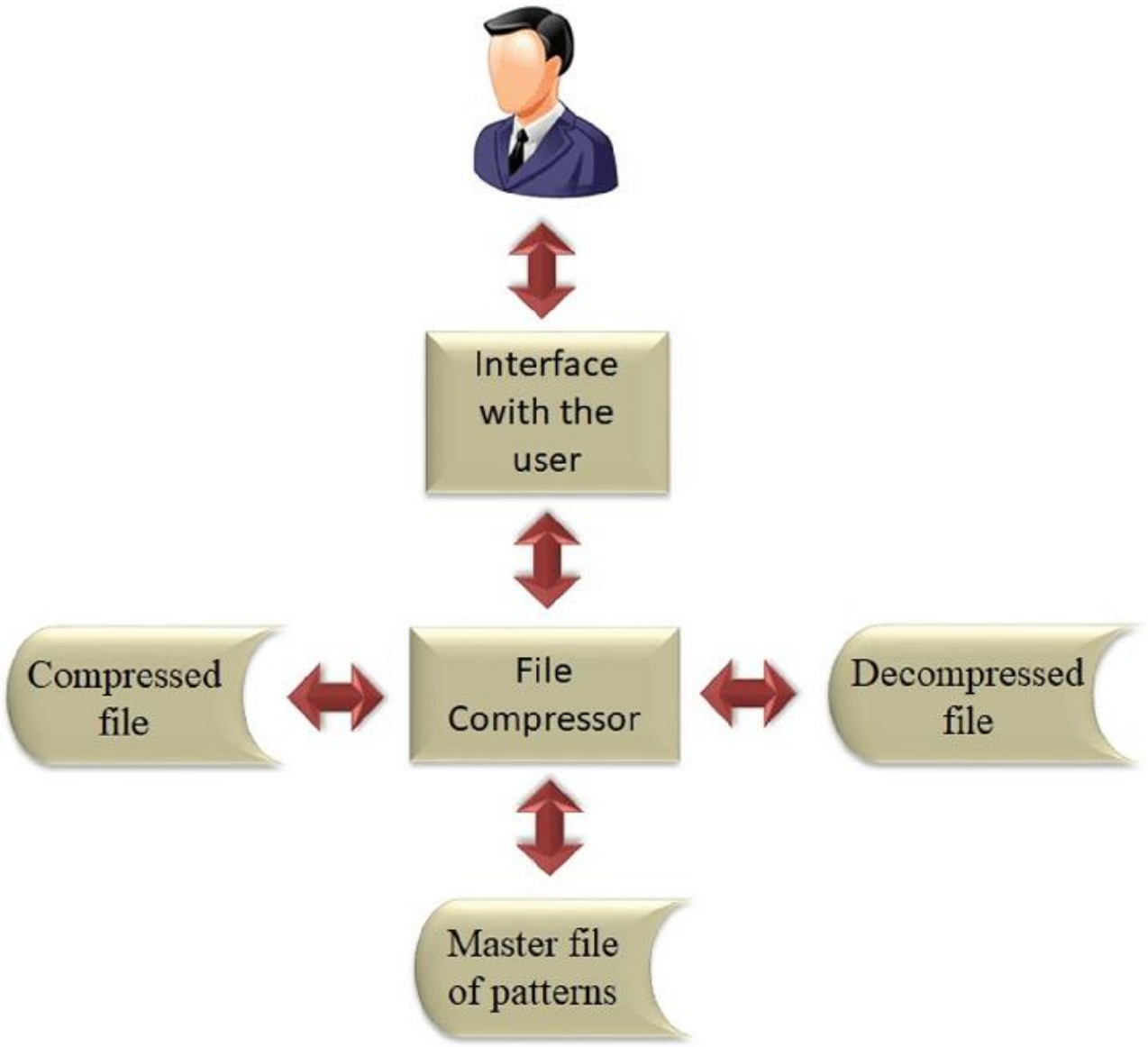


Figure 1: Use case diagram of File Compressor

CHAPTER 4

IMPLEMENTATION

4.1 INTRODUCTION TO FRONT END TOOL

Front End Development Tool is a software application which helps developers to build attractive website layouts and apps with ease. Those tools help to accelerate the web development process by providing drag and drop elements and various built-in features to create a more attractive web design layout

4.1.1 HTML 5

HTML stands for Hyper Text Markup Language. It is used to design web pages using markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. Markup language is used to define the text document within tag which defines the structure of web pages. HTML 5 is the fifth and current version of HTML.

4.1.2 CSS 3

Cascading Style Sheets, fondly referred to as CSS, is a simply designed language intended to simplify the process of making web pages presentable. CSS allows you to apply styles to web pages. More importantly, CSS enables you to do this independent of the HTML that makes each web page.

4.1.3 JAVASCRIPT

JavaScript, often abbreviated JS, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. Over 97% of websites use JavaScript on the client side for web page behavior, often incorporating third-party libraries. All major web browsers have a dedicated JavaScript engine to execute the code on users' devices.

4.2 FILE STRUCTURE CONCEPT USED

4.2.1 Huffman Encoding:

This is the basic idea behind Huffman coding: to use fewer bits for more frequently occurring characters. We'll see how this is done using a tree that stores characters at the leaves, and whose root-to-leaf paths provide the bit sequence used to encode the characters. We'll use Huffman's algorithm to construct a tree that is used for data compression. We'll assume that each character has an associated weight equal to the number of times the character occurs in a file, for example. In the" go go gophers" example, the characters 'g' and 'o' have weight 3, the space has weight 2, and the other characters have weight 1. When compressing a file we'll need to calculate these weights, we'll ignore this step for now and assume that all character weights have been calculated.

4.2.2 Building The Huffman Tree

Huffman's algorithm assumes that we're building a single tree from a group (or forest) of trees. Initially, all the trees have a single node with a character and the character's weight. Trees are combined by picking two trees, and making a new tree from the two trees. This decreases the number of trees by one at each step since two trees are combined into one tree. The algorithm is as follows:

1. Begin with a forest of trees. All trees are one node, with the weight of the tree equal to the weight of the character in the node. Characters that occur most frequently have the highest weights. Characters that occur least frequently have the smallest weights.
2. Repeat this step until there is only one tree.
3. Choose two trees with the smallest weights, call these trees T1 and T2.
4. Create new tree whose node had a weight equal to the sum of the weight of T1 and T2 and whose left subtree is T1 and right subtree is T2
5. The single tree left after the previous step is an optional encoding tree,

4.3 MODULES

style.css index.html

index.html > html > body > nav.navbar.navbar-light.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
        integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">

    <link rel="stylesheet" href="style.css">
    <script src="heap.js"></script>
    <script src="huffman.js" type="module"></script>
    <script src="script.js" type="module"></script>
</head>

<body>
    <nav class="navbar navbar-light " style="font-size: 25px; font-family: sans-serif; background-color: whitesmoke;">


    </nav>
    <div id="container">
        <div class="text_box" style="overflow-y: scroll">
            <span id="treearea" style="width: 100%; text-align: center; font-size: medium;">
                Tree Structure Will Be Displayed Here !!
            </span>
        </div>
        <div class="text_box" style="overflow-y: scroll">
            <span id="temptext" style="width: 100%; text-align: center; font-size: x-large;">
                Operation info will be shown here !!
            </span>
        </div>
    </div>

    <div>
        <form method="post" enctype="multipart/form-data" style="display: inline-block;">
            <input type="file" id="uploadedFile" />
        </form>
        <br>
        <button type="button" class="btn btn-success center_buttons" id="encode">  Encode  </button>
        <br>
        <button type="button" class="btn btn-danger center_buttons" id="decode">  Decode  </button>
    </div>
</body>

</html>
```

style.css | index.html | huffman.js | heap.js | index.php

huffman.js > HuffmanCoder > display

```
/**
 * Created by aarnavjindal on 25/04/20.
 */

import { BinaryHeap } from './heap.js';

export { HuffmanCoder }

class HuffmanCoder{

    stringify(node){
        if(typeof(node[1])==="string"){
            return '\''+node[1];
        }

        return '0' + this.stringify(node[1][0]) + '1' + this.stringify(node[1][1]);
    }

    display(node, modify, index=1){
        if(modify){
            node = ['',node];
            if(node[1].length===1)
                node[1] = node[1][0];
        }

        if(typeof(node[1])==="string"){
            return String(index) + " = " + node[1];
        }

        let left = this.display(node[1][0], modify, index*2);
        let right = this.display(node[1][1], modify, index*2+1);
        let res = String(index*2)+" <= "+index+" => "+String(index*2+1);
        return res + '\n' + left + '\n' + right;
    }

    destringify(data){
        let node = [];
        if(data[this.ind]==='\''){
            this.ind++;
            node.push(data[this.ind]);
            this.ind++;
            return node;
        }

        this.ind++;
        let left = this.destringify(data);
        node.push(left);
```

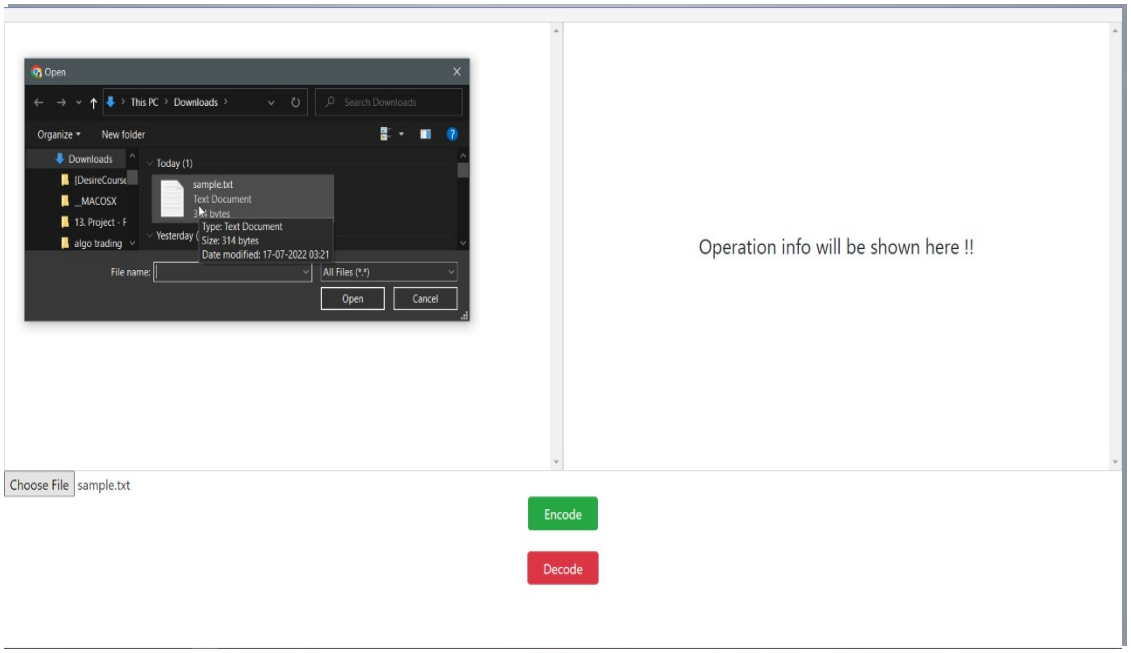
```
            data[2] = data[3];
            data.pop();
        }

        this.ind = 0;
        const huffman_decoder = this.destringify(data[0]);
        const text = data[2];

        let binary_string = "";
        for(let i=0;i<text.length;i++){
            let num = text[i].charCodeAt(0);
            let bin = "";
            for(let j=0;j<8;j++){
                bin = num%2 + bin;
                num = Math.floor(num/2);
            }
            binary_string = binary_string + bin;
        }
        binary_string = binary_string.substring(0,binary_string.length-data[1]);

        console.log(binary_string.length);

        let res = "";
        let node = huffman_decoder;
        for(let i=0;i<binary_string.length;i++){
            if(binary_string[i]==='0'){
                node = node[0];
            } else{
                node = node[1];
            }

            if(typeof(node[0])==="string"){
                res += node[0];
                node = huffman_decoder;
            }
        }
        let info = "Decompression complete and file sent for download";
        return [res, this.display(huffman_decoder, true), info];
    }
}
```

```
            data[2] = data[3];
            data.pop();
        }

        this.ind = 0;
        const huffman_decoder = this.destringify(data[0]);
        const text = data[2];

        let binary_string = "";
        for(let i=0;i<text.length;i++){
            let num = text[i].charCodeAt(0);
            let bin = "";
            for(let j=0;j<8;j++){
                bin = num%2 + bin;
                num = Math.floor(num/2);
            }
            binary_string = binary_string + bin;
        }
        binary_string = binary_string.substring(0,binary_string.length-data[1]);

        console.log(binary_string.length);

        let res = "";
        let node = huffman_decoder;
        for(let i=0;i<binary_string.length;i++){
            if(binary_string[i]==='0'){
                node = node[0];
            } else{
                node = node[1];
            }

            if(typeof(node[0])==="string"){
                res += node[0];
                node = huffman_decoder;
            }
        }
        let info = "Decompression complete and file sent for download";
        return [res, this.display(huffman_decoder, true), info];
    }
}
```

CHAPTER 5

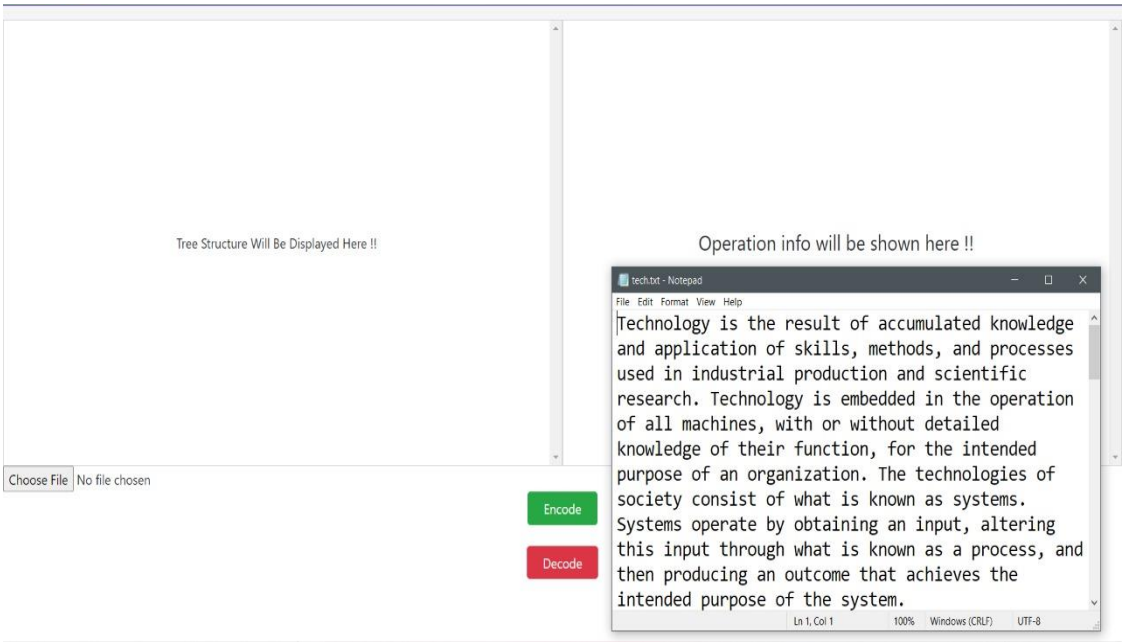
SNAPSHOTS

5.1 HOME PAGE:



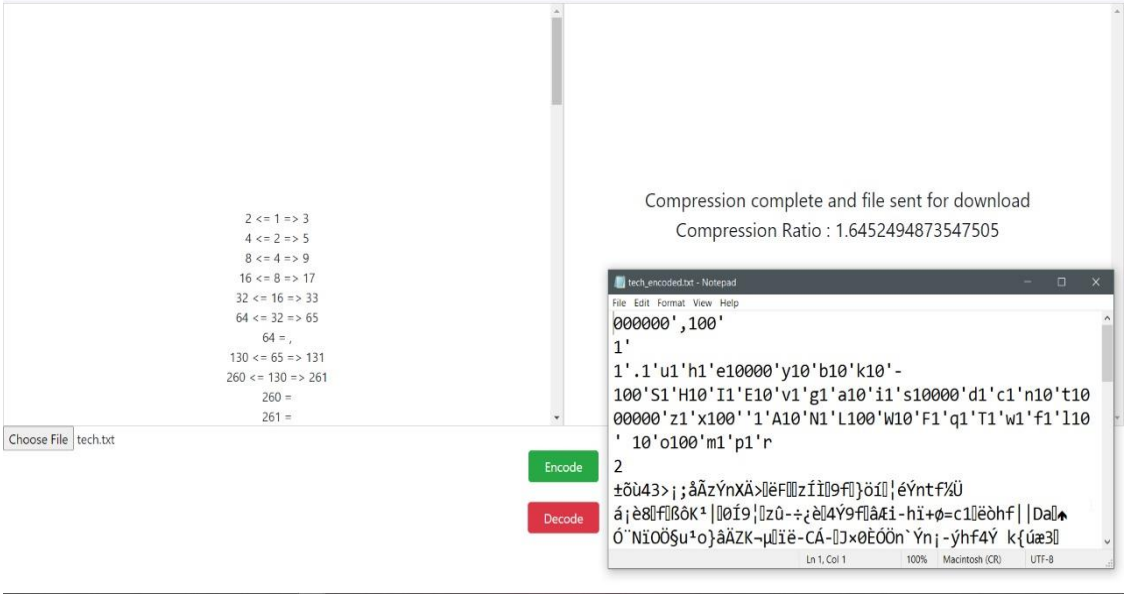
Snapshot 1: "Home page"
User can upload text file here.

5.2 Before Encoding/Compression:



Snapshot 2: "operation"
User can see the content of the text file before compression.

5.3 After Encoding/Compression:



Snapshot 3: "Successful Encoding"

Compression Successful. User can see the encoded text.

CONCLUSION AND FUTURE ENHANCEMENTS

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 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. The above discussed algorithms use lossless compression technique.JPEG technique which is used mostly for image compression is a lossy compression technique.JPEG 2000is advancement in JPEG standard which uses wavelets.

After computing and comparing the compression ratio, average code length and standard deviation for Huffman Coding, Repeated Huffman Coding, Run-Length Coding and Modified Run-Length Coding, an idea is generated about how much compression can be obtained by each technique. So, now the most effective algorithm can be used based on the input text file size, content type, available memory and execution time to get the best result. A new approach for data compression “Modified Run Length algorithm” is also proposed here and which gives a lot better compression than the existing Run-Length algorithm. Future works can be carried on an efficient and optimal coding technique using mixture of two or more coding techniques for image file, exe file etc.to improve compression ratio and reduce average code length.

REFERENCES

1) The complete reference -JavaScript for everybody by dr. Chuck:

2). Google.

3) File Structure: An Object Oriented Approach with C++ by M.Folk

4) Data Structures and Algorithm by R.S. Salaria