

Mapua Malayan Colleges Mindanao

“Boneman Compression System:

Text File Compression System Implemented with Huffman Coding “

In Partial Fulfillment of the Requirements in:

CS106P Data Structures & Algorithms

Submitted to:

Mr. Martzel P. Baste

Submitted by:

Kenji Azriel S. Mende

Introduction

Company Profile



Figure 1: Boneman Technologies Logo

Boneman Technologies is a leading provider of innovative app solutions that address a wide range of tech challenges. With a team of experienced developers and a deep understanding of the latest technologies, Boneman Technologies is committed to delivering cutting-edge apps that empower businesses and individuals to thrive in the ever-evolving tech landscape.

Statement of the Problem

All companies in the modern world utilize computers, and all computers require storage for data. That data will be used for all the functions of the company. Here are some of the issues that affect computers without a proper compression system:

1. **Storage is finite:** Since large amounts of data are transferred and processed around all computers, it is crucial to optimize the use of storage by minimizing storage consumption as much as possible. This is why data compression is crucial for all computers.
2. **Lack of storage efficiency:** It requires a lot of time investment to process large amounts of data at a time, especially when the file sizes are large or a large number of data is being processed simultaneously. This is why it is important to optimize the efficiency of file processing by decreasing the file size.
3. **Better security:** Data compression is not only about reducing and optimizing file size but also encrypting files for better security and smooth data transfer. Compression achieves the level of security needed while keeping storage costs low by significantly reducing the size of encrypted files.

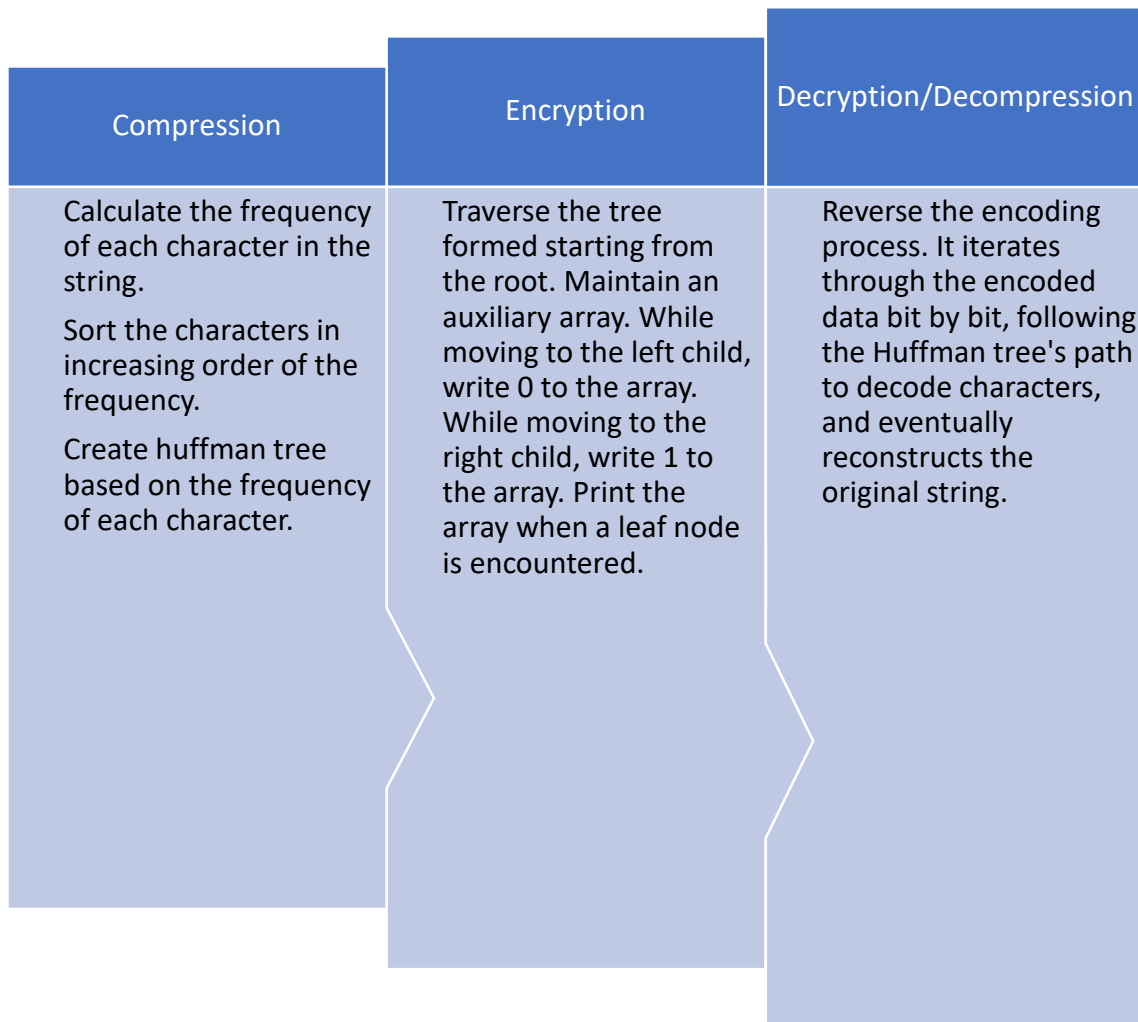
It is detrimental to resolve these issues as it could exponentially benefit the company's efficiency. This article supports this claim: "Beyond cost savings in storage and smaller files that lead to faster transmission, there are so many valuable features of file compression software, like OCR (optical character recognition), encryption, advanced archiving and cataloging. These features greatly affect your business operations and workflows across departments." Mattingly (2020).

Objectives

Data Compression is a process of reducing the size of a computer file by encoding, restructuring, or modifying data to reduce its size. Compression is done by a program that uses functions or an algorithm to effectively discover how to reduce the data size. We will utilize the implementation of Huffman encoding to achieve the objectives of data compression. Huffman encoding is a fundamental algorithm used in many data compression applications. It offers efficient compression, especially for text-based data, by assigning shorter codes to frequently occurring characters, reducing data size while preserving the original content. The objectives of the implementation of Huffman encoding are as follows:

1. **Data Compression:** The primary objective is to implement Huffman encoding to compress text files efficiently by encoding the data into a smaller size.
2. **Decompression:** Implement the corresponding decompression functionality to recover the original data from the compressed format. Ensure that the decompression process is lossless, meaning the original data can be fully recovered from the compressed version.
3. **Encryption Integration:** Integrate a robust encryption mechanism into the data compression to enhance data security during storage and transmission.

Conceptual Framework



Scope and Limitations

The main purpose of the Boneman Compression System is to implement Huffman Encoding for the data compression and decompression of text files. This article states that "Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters; lengths of the assigned codes are based on the frequencies of corresponding characters." (GeeksforGeeks, 2023). The program is designed to compress and decompress files only using Huffman Encoding and will not focus on other compression algorithms and using different file formats other than text files.

Project Definition

The Boneman File Compression System utilizes the concepts and principles of Data Structures and Algorithms, the backbone of the project's functions. It is crucial to the concepts of Data Structures and Algorithms because the project utilizes those concepts to efficiently encode, compress, decode, and decompress text files. The following are the Data Structures used in the project:

Priority Queues: A Priority Queue is a specialized type of queue that always returns the highest priority element. In this code, the Priority Queue stores the nodes of the Huffman tree, with the nodes having the lowest frequency being the highest priority. This ensures that the nodes are merged in the correct order to create the Huffman tree.

Hash Maps: A Hash Map is a type of map that stores key-value pairs. In this code, the HashMap stores the lookup table for characters and their Huffman codes. The lookup table is used to encode and decode the data.

Time and Space Complexity

The Boneman File Compression System utilizes Huffman Coding to encode and compress data. For Huffman Coding, The time complexity for encoding each unique character based on its frequency is $O(n \log n)$. N is the number of symbols being coded.

Huffman Coding is a greedy algorithm that tries to complete its task in the shortest time possible.

Project Solution

Project Prototyping

```
// Debugging test cases
Run | Debug
public static void main(String[] args) {
    testEncodingAndDecoding(input:"Lorem ipsum dolor sit amet");
    testEncodingAndDecoding(input:"Hello, world!");
    testEncodingAndDecoding(input:"AABBBBC");
    testEncodingAndDecoding(input:"This is a test message for Huffman encoding.");
}

private static void testEncodingAndDecoding(String input) {
    HuffmanEncoder encoder = new HuffmanEncoder();
    HuffmanEncoder.HuffmanEncodedResult result = encoder.compress(input);

    System.out.println("Original Data: " + input);
    System.out.println("Encoded Data: " + result.getEncodedData());

    String decodedData = encoder.decompress(result);

    System.out.println("Decoded Data: " + decodedData);

    if (input.equals(decodedData)) {
        System.out.println(x:"Test Passed: Data matches after compression and decompression.");
    } else {
        System.out.println(x:"Test Failed: Data does not match after compression and decompression.");
    }

    System.out.println(x:"-----");
}
```

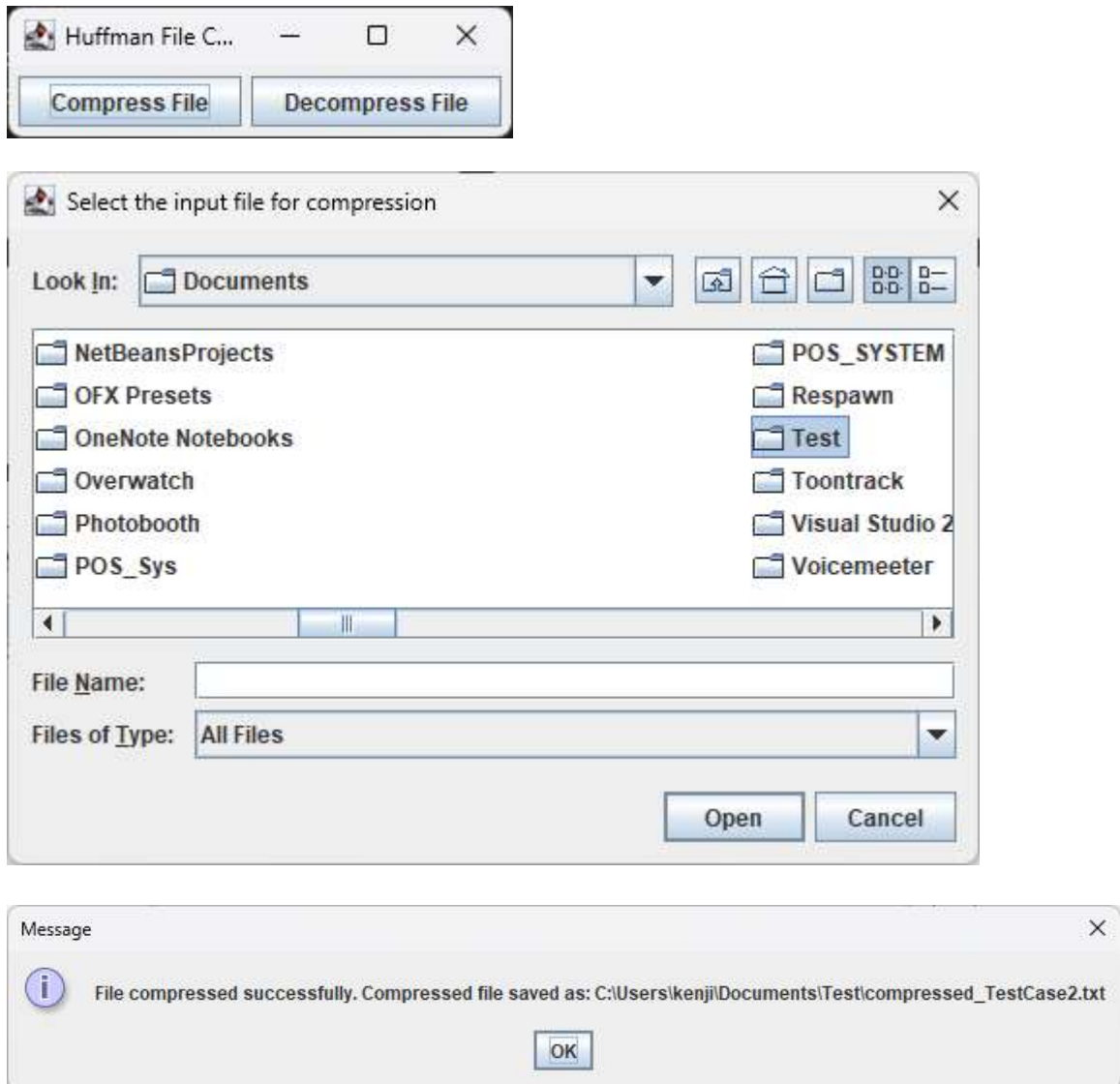
Figure 2: Debugging Test Cases for the compression system.

```
Original Data: Lorem ipsum dolor sit amet
Encoded Data: 10100011100110110100010001101011101101000111100101110010011110001111001010101011111
Decoded Data: Lorem ipsum dolor sit amet
Test Passed: Data matches after compression and decompression.
-----
Original Data: Hello, world!
Encoded Data: 1001111101001000110010110010100111101
Decoded Data: Hello, world!
Test Passed: Data matches after compression and decompression.
-----
Original Data: AABBBBC
Encoded Data: 010111100
Decoded Data: AABBBBC
Test Passed: Data matches after compression and decompression.
-----
Original Data: This is a test message for Huffman encoding.
Encoded Data: 0010000111100001111010001111010101110111010000101011101111011111110111011101111110110001101001110010100100101110000110100010010000111100
Decoded Data: This is a test message for Huffman encoding.
Test Passed: Data matches after compression and decompression.
```

Figure 3: Test case results

Sample Input/Output

Figure 4: UI of the compression system.



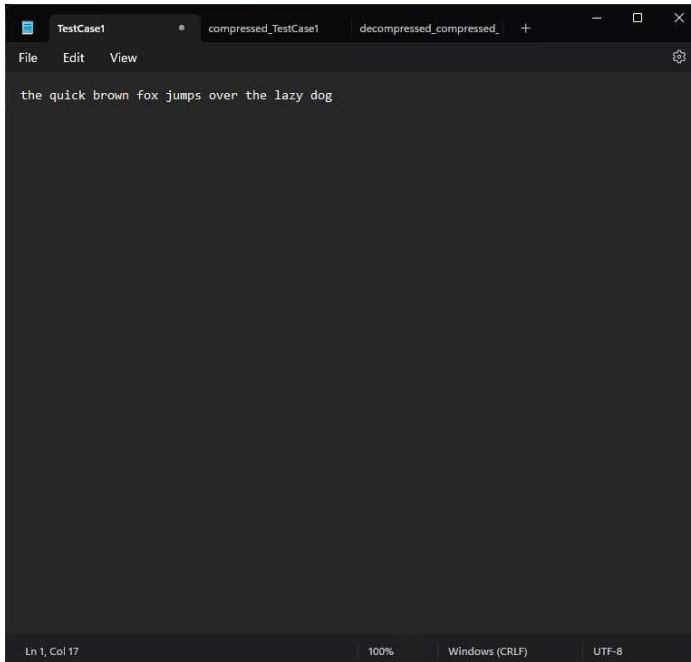


Figure 5: Test Case 1 uncompressed input

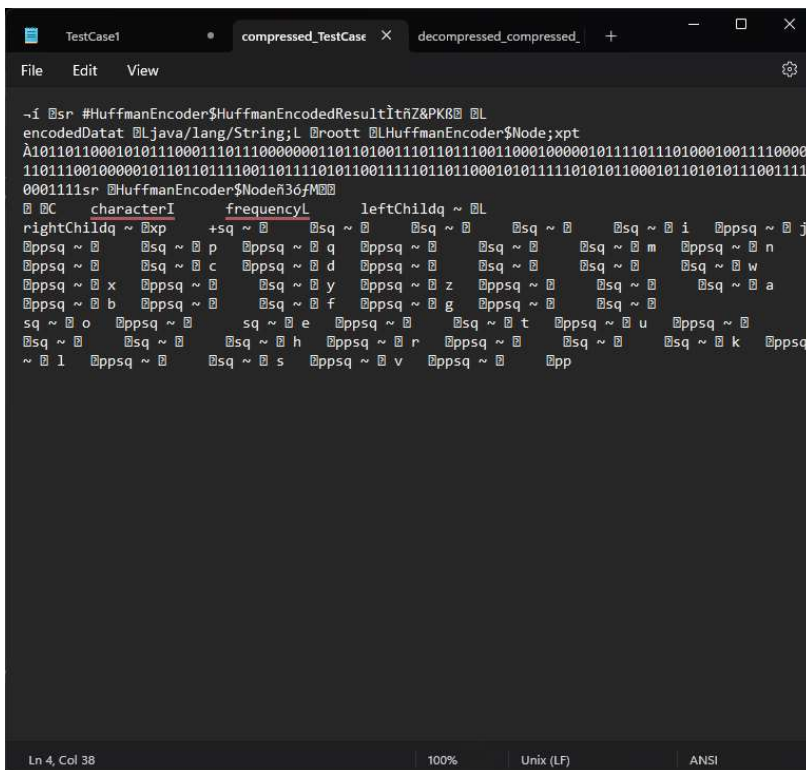


Figure 6: Test Case 1 Compressed Input

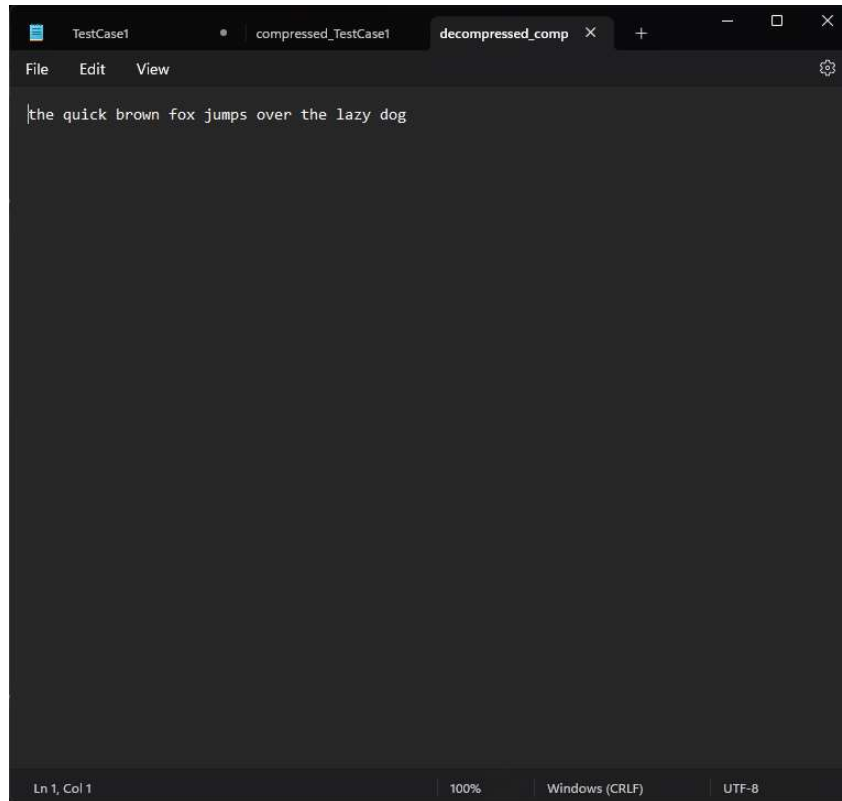


Figure 7: Test Case 1 decompressed output.




 compressed_TestCase1	11/5/2023 9:57 PM	Text Document	2 KB
 decompressed_compressed_TestCase1	11/5/2023 9:58 PM	Text Document	1 KB
 TestCase1	10/22/2023 8:29 PM	Text Document	1 KB

Figure 8: Test Case Uncompressed, Compressed, and Decompressed Input

Source Code

HuffmanEncoder.Java

```
import java.io.Serializable;
import java.util.HashMap;
import java.util.Map;
import java.util.PriorityQueue;

public class HuffmanEncoder {
    // Define the size of the character alphabet (ASCII characters)
    private static final int ALPHABET_SIZE = 256;

    // Compress a given string and return the result
    public HuffmanEncodedResult compress(final String data) {
        // Build a frequency table for characters in the data
        final int[] freq = buildFrequencyTable(data);

        // Build a Huffman tree from the frequency table
        final Node root = buildHuffmanTree(freq);

        // Build a lookup table for characters and their Huffman codes
        final Map<Character, String> lookupTable = buildLookupTable(root);

        // Generate the encoded data using the lookup table
        final String encodedData = generateEncodedData(data, lookupTable);

        // Return the result, including the encoded data and the Huffman tree's root
        return new HuffmanEncodedResult(encodedData, root);
    }
}
```

```

// Generate encoded data based on the lookup table

private static String generateEncodedData(String data, Map<Character,
String> lookupTable) {
    final StringBuilder builder = new StringBuilder();
    for (final char character : data.toCharArray()) {
        builder.append(lookupTable.get(character));
    }
    return builder.toString();
}

// Build a lookup table for characters and their Huffman codes

private static Map<Character, String> buildLookupTable(final Node root) {
    final Map<Character, String> lookupTable = new HashMap<>();
    buildLookupTableImpl(root, "", lookupTable);
    return lookupTable;
}

// Helper method to recursively build the lookup table

private static void buildLookupTableImpl(final Node node, String s,
    Map<Character, String> lookupTable) {
    if (!node.isLeaf()) {
        buildLookupTableImpl(node.leftChild, s + '0', lookupTable);
        buildLookupTableImpl(node.rightChild, s + '1', lookupTable);
    } else {
        lookupTable.put(node.character, s);
    }
}

```

```

// Build a frequency table for characters in the data
private static int[] buildFrequencyTable(final String data) {
    final int[] freq = new int[ALPHABET_SIZE];
    for (final char character : data.toCharArray()) {
        freq[character]++;
    }
    return freq;
}

```

```

// Decompress a HuffmanEncodedResult and return the original data
public String decompress(final HuffmanEncodedResult result) {
    final StringBuilder resultBuilder = new StringBuilder();
    Node current = result.getRoot();
    int i = 0;
    while (i < result.getEncodedData().length()) {
        while (!current.isLeaf()) {
            char bit = result.getEncodedData().charAt(i);
            if (bit == '1') {
                current = current.rightChild;
            } else if (bit == '0') {
                current = current.leftChild;
            } else {
                throw new IllegalArgumentException("Invalid bit in message" + bit);
            }
            i++;
        }
        resultBuilder.append(current.character);
        current = result.getRoot();
    }
}

```

```

    }
    return resultBuilder.toString();
}

```

// Define a helper class to store Huffman encoding results

```
static class HuffmanEncodedResult implements Serializable {
```

```
    final Node root;
```

```
    final String encodedData;
```

```
    HuffmanEncodedResult(final String encodedData, final Node root) {
```

```
        this.encodedData = encodedData;
```

```
        this.root = root;
```

```
    }
```

```
    public String getEncodedData() {
```

```
        return this.encodedData;
```

```
    }
```

```
    public Node getRoot() {
```

```
        return this.root;
```

```
    }
```

```
}
```

// Build a Huffman tree from a frequency table

```
private static Node buildHuffmanTree(int[] freq) {
```

```
    final PriorityQueue<Node> priorityQueue = new PriorityQueue<>();
```

```
    for (int i = 0; i < ALPHABET_SIZE; i++) {
```

```
        if (freq[i] > 0) {
```

```

        priorityQueue.add(new Node((char) i, freq[i], null, null));
    }
}
while (priorityQueue.size() > 1) {
    final Node left = priorityQueue.poll();
    final Node right = priorityQueue.poll();
    final Node parent = new Node('\0', left.frequency + right.frequency, left,
right);
    priorityQueue.add(parent);
}
return priorityQueue.poll();
}

```

```

// Define a helper class to represent nodes in the Huffman tree
static class Node implements Comparable<Node>, Serializable {
    private final char character;
    private final int frequency;
    private final Node leftChild;
    private final Node rightChild;

    private Node(final char character, final int frequency, final Node leftChild,
final Node rightChild) {
        this.character = character;
        this.frequency = frequency;
        this.leftChild = leftChild;
        this.rightChild = rightChild;
    }
}

```

```

// Check if the node is a leaf node

```

```

    boolean isLeaf() {
        return this.leftChild == null && this.rightChild == null;
    }

    @Override
    public int compareTo(final Node that) {
        // Compare nodes based on frequency and character
        final int frequencyComparison = Integer.compare(this.frequency,
that.frequency);
        if (frequencyComparison != 0) {
            return frequencyComparison;
        }
        return Character.compare(this.character, that.character);
    }
}

// Debugging test cases
public static void main(String[] args) {
    testEncodingAndDecoding("Lorem ipsum dolor sit amet");
    testEncodingAndDecoding("Hello, world!");
    testEncodingAndDecoding("AABBBC");
    testEncodingAndDecoding("This is a test message for Huffman encoding.");
}

private static void testEncodingAndDecoding(String input) {
    HuffmanEncoder encoder = new HuffmanEncoder();
    HuffmanEncoder.HuffmanEncodedResult result = encoder.compress(input);

    System.out.println("Original Data: " + input);
}

```



```

        System.out.println("Encoded Data: " + result.getEncodedData());

        String decodedData = encoder.decompress(result);

        System.out.println("Decoded Data: " + decodedData);

        if (input.equals(decodedData)) {
            System.out.println("Test Passed: Data matches after compression and
            decompression.");
        } else {
            System.out.println("Test Failed: Data does not match after compression
            and decompression.");
        }

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

```

FileCompressor.java:

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;

public class FileCompressor {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            createAndShowGUI();
        });
    }
}

```

```
}
```

```
private static void createAndShowGUI() {  
    JFrame frame = new JFrame("Huffman File Compressor");  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
    JPanel panel = new JPanel();  
    frame.getContentPane().add(panel);  
    panel.setLayout(new FlowLayout());  
  
    JButton compressButton = new JButton("Compress File");  
    JButton decompressButton = new JButton("Decompress File");  
  
    panel.add(compressButton);  
    panel.add(decompressButton);  
  
    compressButton.addActionListener(new ActionListener() {  
        @Override  
        public void actionPerformed(ActionEvent e) {  
            File inputFilePath = selectFile("Select the input file for compression");  
            if (inputFilePath != null) {  
                try {  
                    String data = readFile(inputFilePath);  
                    HuffmanEncoder encoder = new HuffmanEncoder();  
                    HuffmanEncoder.HuffmanEncodedResult result =  
encoder.compress(data);  
  
                    // Create the compressed file in the same folder as the input file
```

```
        File outputFilePath = new File(inputFilePath.getParentFile(),
"compressed_" + inputFilePath.getName());
```

```
        writeEncodedFile(outputFilePath, result);
```

```
        JOptionPane.showMessageDialog(null, "File compressed
successfully. Compressed file saved as: " + outputFilePath);
```

```
    } catch (IOException ex) {
```

```
        JOptionPane.showMessageDialog(null, "Error: " +
ex.getMessage());
```

```
    }
```

```
}
```

```
}
```

```
});
```

```
decompressButton.addActionListener(new ActionListener() {
```

```
    @Override
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        File inputFilePath = selectFile("Select the compressed file for
decompression");
```

```
        if (inputFilePath != null) {
```

```
            try {
```

```
                HuffmanEncoder encoder = new HuffmanEncoder();
```

```
                HuffmanEncoder.HuffmanEncodedResult result =
readEncodedFile(inputFilePath);
```

```
                // Create the decompressed file in the same folder as the input file
```

```
                File outputFilePath = new File(inputFilePath.getParentFile(),
"decompressed_" + inputFilePath.getName());
```

```

        String decodedData = encoder.decompress(result);

        writeFile(outputFilePath, decodedData);

        JOptionPane.showMessageDialog(null, "File decompressed
successfully. Decompressed file saved as: " + outputFilePath);
    } catch (IOException | ClassNotFoundException ex) {
        JOptionPane.showMessageDialog(null, "Error: " +
ex.getMessage());
    }
}

});

frame.pack();
frame.setVisible(true);
}

private static File selectFile(String dialogTitle) {
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setDialogTitle(dialogTitle);
    int returnValue = fileChooser.showDialog(null, "Select");

    if (returnValue == JFileChooser.APPROVE_OPTION) {
        return fileChooser.getSelectedFile();
    } else {
        return null;
    }
}
}

```

```

private static String readFile(File file) throws IOException {
    try (FileInputStream inputStream = new FileInputStream(file)) {
        ByteArrayOutputStream buffer = new ByteArrayOutputStream();
        byte[] data = new byte[128];
        int bytesRead;

        while ((bytesRead = inputStream.read(data, 0, data.length)) != -1) {
            buffer.write(data, 0, bytesRead);
        }

        return new String(buffer.toByteArray());
    }
}

```

```

private static void writeEncodedFile(File file,
HuffmanEncoder.HuffmanEncodedResult result) throws IOException {
    try (ObjectOutputStream outputStream = new ObjectOutputStream(new
FileOutputStream(file))) {
        outputStream.writeObject(result);
    }
}

```

```

private static HuffmanEncoder.HuffmanEncodedResult readEncodedFile(File
file) throws IOException, ClassNotFoundException {
    try (ObjectInputStream inputStream = new ObjectInputStream(new
FileInputStream(file))) {
        return (HuffmanEncoder.HuffmanEncodedResult)
inputStream.readObject();
    }
}

```

```
}
```

```
private static void writeFile(File file, String data) throws IOException {  
    try (FileOutputStream outputStream = new FileOutputStream(file)) {  
        outputStream.write(data.getBytes());
```

```
    }
```

```
}
```

```
}
```

References

1. GeeksforGeeks. (2023, September 11). *Huffman Coding Greedy Algo 3*.
<https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/>
2. Mattingly, C. W. (2020, March 10). *How file compression benefits business beyond storage and transmission*. <https://www.linkedin.com/pulse/how-file-compression-benefits-business-beyond-storage-mattingly/>

Curriculum Vitae



CONTACT

📞 09278123080
✉ kenji.mende@lpunetwork.edu.ph
📍 La Verna Hills, Davao City

OTHER INFORMATION

Religion Born Again Christian
Hobbies Video Editing, Video Games
Talents Video Editing, Programming
Place of Birth Davao City
Date of Birth 10/26/2003

KENJI AZRIEL S. MENDE

BSCS

EDUCATION HISTORY

2022 2020-2022 2016-2020 2007-2016

PHILIPPINE NIKKEI JIN KAI INTERNATIONAL SCHOOL
ELEMENTARY

PHILIPPINE NIKKEI JIN KAI INTERNATIONAL SCHOOL
JUNIOR HIGH SCHOOL

LYCEUM OF THE PHILIPPINES UNIVERSITY - DAVAO
SENIOR HIGH SCHOOL

MAPUA MALAYAN COLLEGES MINDANAO
COLLEGE - BACHELOR OF SCIENCE IN COMPUTER SCIENCE

PROGRAMMING EXPERIENCE

- Java Programming
- C# Programming
- Python Programming