Algorithms Hoffman Project

Prepared by omar badr bas these n words didn’t help me

**Code and Files**

1. **main.java**

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.OutputStreamWriter;

import java.util.ArrayList;

import java.util.Comparator;

import java.util.HashMap;

import java.util.PriorityQueue;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        Utilities utilities = new Utilities();

        String inputFile = new String();

        int option;

        System.out.println("\*\*\*\*\*\* Huffman Project \*\*\*\*\*\*");

        System.out.println("Press 1 for Compression OR press 2 for Decompression");

        option = scanner.nextInt();

        scanner.nextLine();

        if(option == 1){

            System.out.print("File name : ");

            inputFile = scanner.nextLine();

            utilities.readInputFile(inputFile);

            utilities.setCharactersAndFrequencies();

            utilities.buildHuffmanQueue();

            utilities.buildHuffmanTree();

            try {

                utilities.writeEncodedFile();

            } catch (IOException e) {

                // TODO Auto-generated catch block

                e.printStackTrace();

            }

            System.out.println("File is compressed succesfully");

        }

        else if(option == 2){

            System.out.println("File name : ");

            inputFile = scanner.nextLine();

            utilities.readEncodedFile(inputFile);

            try {

                utilities.writeDecodedFile();

            } catch (IOException e) {

                // TODO Auto-generated catch block

                e.printStackTrace();

            }

            System.out.println("\nFile is decompressed succesfully");

        }

        else{

            System.out.println("Please pick 1 or 2");

        }

    }

}

This is our main function, were we simply ask the user to pick an option between 1 to compress and 2 to decompress. Providing a file name.

1. **HuffmanNode.java**

public class HuffmanNode {

    private int freq;

    private char character;

    private HuffmanNode left;

    private HuffmanNode right;

    private String code;

    public HuffmanNode(Integer freq, Character character) {

        this.freq = freq;

        this.character = character;

        left = null;

        right = null;

        code = new String();

    }

    public int getFreq() {

        return freq;

    }

    public char getCharacter() {

        return character;

    }

    public String getCode() {

        return code;

    }

    public HuffmanNode getLeft() {

        return left;

    }

    public void setLeft(HuffmanNode left) {

        this.left = left;

    }

    public HuffmanNode getRight() {

        return right;

    }

    public void setRight(HuffmanNode right) {

        this.right = right;

    }

    public void appendBit(String string){

        code += string;

    }

}

Class is responsible for the creation of the nodes we use in the making of the Huffman tree. Each node is given a frequency, a character, a left and a right node and a code (given it is a leaf node). We assign a frequency 1 and a special character to every node that isn’t a leaf node so it will not be used when decompressing and finding the character.

1. **Utilities.java**
2. import java.io.BufferedWriter;
3. import java.io.File;
4. import java.io.FileInputStream;
5. import java.io.FileNotFoundException;
6. import java.io.FileOutputStream;
7. import java.io.IOException;
8. import java.io.InputStream;
9. import java.io.OutputStream;
10. import java.io.OutputStreamWriter;
11. import java.util.ArrayList;
12. import java.util.Comparator;
13. import java.util.HashMap;
14. import java.util.PriorityQueue;
15. import java.util.Scanner;
16. public class Utilities {
17. String compressedLine = "", fileString;
18. StringBuilder decompressedFile ;
19. HashMap<Character, String> huffmanEncodingMap = new HashMap<>();
20. HashMap<String, String> huffmanDecodingMap = new HashMap<>();
21. ArrayList<Character> fileCharacters = new ArrayList<>();
22. ArrayList<Integer> fileCharactersFrequency = new ArrayList<>();
23. Comparator<HuffmanNode> huffmanComparator = new HuffmanNodeComparator();
24. PriorityQueue<HuffmanNode> huffmanQueue = new PriorityQueue<HuffmanNode>(5, huffmanComparator);
25. HuffmanDecodedTree huffmanDecodedTree = new HuffmanDecodedTree();
26. int fileStringLength;
27. public void readBinaryFile(String filename) throws IOException {
28. try (InputStream inputStream = new FileInputStream(filename);
29. ) {
30. long fileSize = new File(filename).length();
31. byte[] allBytes = new byte[(int) fileSize];
32. fileString = "";
33. inputStream.read(allBytes);
34. String inputBinary = new String(allBytes);
35. System.out.println(inputBinary);
36. String test;
37. for (int i = 0; i < inputBinary.length();) {
38. if (i + 8 > inputBinary.length()) {
39. test = inputBinary.substring(i, inputBinary.length());
40. System.out.println(test);
41. i = inputBinary.length();
42. } else {
43. test = inputBinary.substring(i, i + 8);
44. i += 8;
45. }
46. char x = (char) Integer.parseInt(test, 2);
47. System.out.println(String.valueOf(x));
48. fileString += x;
49. System.out.println(fileString);
50. }
51. } catch (IOException ex) {
52. ex.printStackTrace();
53. }
54. }
55. public void readEncodedFile(String fileName) {
56. File outputFile = new File(fileName);
57. Scanner scanner;
58. try {
59. scanner = new Scanner(outputFile);
60. while (scanner.hasNextLine()) {
61. String keyCharacter = scanner.next();
62. if(keyCharacter.equals("^"))
63. keyCharacter = " ";
64. if (keyCharacter.equals("�"))
65. break;
66. String codeValue = scanner.next();
67. scanner.nextLine();
68. huffmanDecodedTree.build(keyCharacter, codeValue);
69. }
70. scanner.nextLine();
71. while (scanner.hasNextLine())
72. compressedLine += scanner.nextLine();
73. } catch (FileNotFoundException e) {
74. // TODO Auto-generated catch block
75. e.printStackTrace();
76. }
77. }
78. public String getCompressedLine() {
79. return compressedLine;
80. }
81. public HashMap<Character, String> getHuffmanEncodingMap() {
82. return huffmanEncodingMap;
83. }
84. public void readInputFile(String fileName) {
85. fileString = new String();
86. File inputFile = new File(fileName);
87. Scanner scanner;
88. try {
89. scanner = new Scanner(inputFile);
90. while (scanner.hasNextLine()) {
91. fileString += (scanner.nextLine());
92. fileString += "{";
93. }
94. fileString = fileString.substring(0, fileString.length() - 1);
95. // System.out.println(fileString);
96. } catch (FileNotFoundException e) {
97. // TODO Auto-generated catch block
98. e.printStackTrace();
99. }
100. }
101. public String getFileString() {
102. return fileString;
103. }
104. public void setCharactersAndFrequencies() {
105. fileStringLength = fileString.length();
106. for (int i = 0; i < fileStringLength; i++) {
107. if (!fileCharacters.contains(fileString.charAt(i))) {
108. fileCharacters.add(fileString.charAt(i));
109. fileCharactersFrequency.add(1);
110. } else {
111. int index = fileCharacters.indexOf(fileString.charAt(i));
112. int characterFrequency = fileCharactersFrequency.get(index);
113. characterFrequency++;
114. fileCharactersFrequency.set(index, characterFrequency);
115. }
116. }
117. }
118. public ArrayList<Character> getFileCharacters() {
119. return fileCharacters;
120. }
121. public ArrayList<Integer> getFileCharactersFrequency() {
122. return fileCharactersFrequency;
123. }
124. public void buildHuffmanQueue() {
125. int fileCharactersSize = fileCharacters.size();
126. for (int i = 0; i < fileCharactersSize; i++) {
127. huffmanQueue.add(new HuffmanNode(fileCharactersFrequency.get(i), fileCharacters.get(i)));
128. }
129. }
130. public void buildHuffmanTree() {
131. HuffmanTree huffmanTreeBuilder = new HuffmanTree(huffmanQueue);
132. huffmanEncodingMap = huffmanTreeBuilder.build();
133. }
134. public void writeEncodedFile() throws IOException {
135. String encodedOutput = new String();
136. String bitsSubString;
137. char correspondingCharacter;
138. // get the input file as sequence of 0s and 1s
139. for (int i = 0; i < fileStringLength; i++)
140. encodedOutput += huffmanEncodingMap.get(fileString.charAt(i));
141. BufferedWriter outputFile = new BufferedWriter(new OutputStreamWriter(new FileOutputStream("compressedFile.txt")));
142. outputFile.write("");
143. // write each character and its code in the output file
144. for (int i = 0; i < huffmanEncodingMap.size(); i++) {
145. char mapCharacter = fileCharacters.get(i);
146. if (mapCharacter == ' ')
147. outputFile.append("^" + " " + huffmanEncodingMap.get(mapCharacter));
148. else
149. outputFile.append(mapCharacter + " " + huffmanEncodingMap.get(mapCharacter));
150. outputFile.newLine();
151. }
152. outputFile.append('�');
153. outputFile.newLine();
154. // convert each 6 digits to their corresponding asci character and write
155. // it in the output file
156. // if the corresponding asci character is \n, then write � in the output
157. // file
158. int encodedOutputLength = encodedOutput.length();
159. for (int i = 0; i < encodedOutputLength;) {
160. if (i + 6 > encodedOutputLength) {
161. bitsSubString = encodedOutput.substring(i, encodedOutputLength);
162. // System.out.println(test);
163. i = encodedOutputLength;
164. } else {
165. bitsSubString = encodedOutput.substring(i, i + 6);
166. i += 6;
167. }
168. correspondingCharacter = (char) (Integer.parseInt(bitsSubString, 2) + 33);
169. outputFile.append(correspondingCharacter);
170. }
171. outputFile.close();
172. }
173. public void writeDecodedFile() throws IOException {
174. decompressedFile = huffmanDecodedTree.decodeFile(compressedLine);
175. writeDecompressedFile(decompressedFile);
176. }
177. private void writeDecompressedFile(StringBuilder decompressedFile) throws IOException {
178. BufferedWriter outputFile = new BufferedWriter(new OutputStreamWriter(new FileOutputStream("decompressedFile.txt")));
179. outputFile.write("");
180. outputFile.write(decompressedFile.toString());
181. outputFile.close();
182. }
183. }

**Utilities.java contains most of the logic used in compressing and decompressing.**

ReadEncodedFile is the function that reads the compressed file and singles out the key character and its corresponding binary code, and sends them to another function to build a tree using only the traversals of the specific binary code it read from the file. It also is responsible for the text written in compressedFile.

ReadInputFile is the function responsible for reading the input file which probably will be a text file containing the text to be compressed.

BuildHuffmanQueue and BuildHuffmanTree is mainly responsible for creating the priority queue we use to store the characters and their frequencies and building the Huffman tree we use to assign the binary equivalent.

WriteEncodedFile is used to start writing the compressed file. We write each character with its equivalent binary equivalent. Then we have an empty string that we store every binary equivalent from the given characters, we turn every 6 bits in this string to its ASCII equivalent and write the results under the keys and its code.

There is a test file with input

CHAPTER I.

YOU don't know about me without you have read a book by the name of The

Adventures of Tom Sawyer; but that ain't no matter. That book was made

by Mr. Mark Twain, and he told the truth, mainly. There was things which

he stretched, but mainly he told the truth. That is nothing. I never

seen anybody but lied one time or another, without it was Aunt Polly, or

the widow, or maybe Mary. Aunt Polly--Tom's Aunt Polly, she is--and

Mary, and the Widow Douglas is all told about in that book, which is

mostly a true book, with some stretchers, as I said before.

{ 110000

C 1111001010

H 011111000

A 1110000

P 0111101

T 011000

E 011111011

R 1111001011

^ 00

I 11100010

. 1111111

Y 011111001

O 011111010

U 111100100

d 01011

o 1010

n 11101

' 11110011

t 1101

k 1111110

w 111101

a 1000

b 111110

u 01101

m 111001

e 1001

i 11001

h 0100

y 01010

v 11100011

r 10111

f 11110000

s 10110

S 011001100

; 011001101

M 0111111

, 110001

l 01110

g 11110001

c 0110010

- 0111100

W 011001110

D 011001111

Ë

Q]J]=(L$\]MYP\_'$Z@8E&VXTU`>LUD[G[=S?ZV+<IKGC2RS8G&E(V6\_0F'J\*W/:+]"B\*QY,YZ\L<T9LQ'+Z':(KJOT4ZOGJ'E:\Z[>IZD>G`]'%D4[K`(M7/98/(SB`@\_0YP\_'09TX%DKR\*.JST5EW\>4%ZD08+`AQEXE^CRK:\YW05SS4"\*,<?>:\*+]4ZOH-:[Z1EGUZ:K3<>OJ`"B)UT9\7JHPD`$C/TRT@#UTUDKPK7I\_<I\-L+\\*O?:+O2WN3<YHO>3G[:U^CRQ<X2\4H+R+PBK3?Z88O\*6TG&@30YOL\_("P>(N/=JY]9LHTM='^U?UZSM3S3:M]?2V?$\_,KR)[MV\*'>F>?D0JPC[,'<%/=<4FE@5X4045D4[K`9HK::):NBT6N=IAW\:0KL^C?ZV#V=S7W][S3<WR)M=2;-L0J]+P00

4. HuffmanTree.java

import java.util.HashMap;

import java.util.PriorityQueue;

public class HuffmanTree {

    PriorityQueue<HuffmanNode> huffmanQueue;

    HashMap<Character, String> huffmanEncodingMap = new HashMap<Character, String>();

    public HuffmanTree(PriorityQueue<HuffmanNode> huffmanQueue) {

        this.huffmanQueue = huffmanQueue;

    }

    public HashMap build() {

        while (huffmanQueue.size() != 1) {

            HuffmanNode leftChildNode = huffmanQueue.remove();

            HuffmanNode rightChildNode = huffmanQueue.remove();

            HuffmanNode parentNode = new HuffmanNode(leftChildNode.getFreq() + rightChildNode.getFreq(), '�');

            parentNode.setLeft(leftChildNode);

            parentNode.setRight(rightChildNode);

            huffmanQueue.add(parentNode);

        }

        printHuffmanTree(huffmanQueue.remove());

        return huffmanEncodingMap;

    }

    private void printHuffmanTree(HuffmanNode root) {

        //System.out.println(root.getFreq());

        if (root.getLeft() != null){

            root.getLeft().appendBit(root.getCode());

            root.getLeft().appendBit("0");

            printHuffmanTree(root.getLeft());

        }

        if (root.getRight() != null){

            root.getRight().appendBit(root.getCode());

            root.getRight().appendBit("1");

            printHuffmanTree(root.getRight());

        }

//      System.out.println(root.getCharacter() + " " + root.getCode());

        if(root.getCharacter()!='�'){

        huffmanEncodingMap.put(root.getCharacter(), root.getCode());

    }}

}

**Function is responsible for building the tree. Moving left in the tree we give the node a 0 and we give it 1 if we move right.**