Huffman Coding Report

Alan O’Regan

# Project Description

This project is to implement the [Huffman Coding string compression algorithm](https://en.wikipedia.org/wiki/Huffman_coding). This implementation uses the List and Tree abstract data types (with the addtion of the comparable interface) implemented in the **ListArrayBased** and **TreeNode** classes from the Data Structures and Algorithms module along with custom data classes **HuffmanSymbol** and **HuffmanEncodedSymbol** to handle the data. This implementation includes a graphical user interface to interact with the program, developed using methods from the GUI Programming Module.

## Part 1: Generating the Huffman Tree

*Implementation:* [*HuffmanCoding.generateHuffmanTree Line:47*](src/HuffmanCoding.java#L47)

private void generateHuffmanTree() {  
  
 TreeNode first;  
 TreeNode second;  
 HuffmanSymbol newRootSymbol;  
 while (frequencyTable.size() > 1) {  
  
 // get the first two items from the table and cast them to a HuffmanSymbol  
 first = (TreeNode) frequencyTable.get(1);  
 second = (TreeNode) frequencyTable.get(2);  
  
 // create the new root symbol with a '\*' character  
 // and the sum of the first two items frequency  
 newRootSymbol = new HuffmanSymbol(  
 '\*',  
 ((HuffmanSymbol)first.getItem()).getFrequency()  
 + ((HuffmanSymbol)second.getItem()).getFrequency()  
 );  
  
 // replace the root node with the new root symbol node  
 // and use the first two items as the left and right children  
 rootNode = new TreeNode(newRootSymbol);  
 rootNode.setLeft(first);  
 rootNode.setRight(second);  
  
 // remove the first two items as they are now in a tree  
 frequencyTable.remove(1);  
 frequencyTable.remove(1);  
  
 frequencyTable.add(frequencyTable.size()+1, rootNode); // add new root symbol to the end of the table  
 frequencyTable.sort(); // sort in order to place new root symbol in its proper place  
 }  
}

## Part 2: Encode

*Implementation:* [*HuffmanCoding.encodeCharacters Line:115*](src/HuffmanCoding.java#L115)

/\*\*  
 \* encoding method based on binary search.  
 \* @param characters the string of characters to be encoded  
 \* @return the encoded value of given characters  
 \*/  
public String encodeCharacters(String characters) {  
 StringBuilder sb = new StringBuilder();  
 // validate input as only capital letters from A-Z  
 characters = characters.toUpperCase(Locale.ROOT).replaceAll("[^A-Z]","");  
  
 for (char ch : characters.toCharArray()) {  
  
 int low = 1, high = lookupTable.size(), mid;  
 while (low <= high) {  
 mid = (low + high) / 2;  
 HuffmanEncodedSymbol midItem = ((HuffmanEncodedSymbol)lookupTable.get(mid));  
  
 if (midItem.letter == ch) {  
 sb.append(midItem.binary);  
 break; // value found exit loop  
 }  
  
 if (ch > midItem.letter)  
 low = mid + 1;  
 else  
 high = mid - 1;  
 }  
 }  
  
 return sb.toString();  
}

## Part 3: Decode

Implementation: [*HuffmanCoding.decodeCharacters() Line:147*](src/HuffmanCoding.java#L147)

/\*\*  
 \* Decoding method based on binary search.  
 \* @param characters the string of characters to be decoded  
 \* @return the decoded value of given characters  
 \*/  
public String decodeCharacters(String characters) {  
 StringBuilder sb = new StringBuilder();  
 TreeNode currentNode = rootNode;  
 // validate characters as only numbers 0 and 1  
 characters = characters.toUpperCase(Locale.ROOT).replaceAll("[^01]","");  
  
 for (char ch : characters.toCharArray()) {  
  
 char character = ((HuffmanSymbol) currentNode.getItem()).getCharacter();  
  
 if (character != '\*') {  
 sb.append(character);  
 currentNode = rootNode;  
 }  
  
 currentNode = switch (ch) {  
 case '0' -> currentNode.getLeft();  
 case '1' -> currentNode.getRight();  
 default -> currentNode;  
 };  
 }  
 if (((HuffmanSymbol) currentNode.getItem()).getCharacter() != '\*')  
 sb.append(((HuffmanSymbol) currentNode.getItem()).getCharacter());  
  
 return sb.toString();  
}

## Part 4: Program Interface

|  |
| --- |
| Screenshot of Program Interface |

### Encode button logic

Implementation: [*HuffmanCodingGUI.actionPerformed() Line:188*](src/HuffmanCodingGUI.java#L88)

if (source == encode) {  
  
 if (!inputText.isBlank())  
 result = huffmanCoding.encodeCharacters(inputText);  
 else  
 resultTextArea.setText("Empty Input");  
  
  
 if (!result.isBlank()) {  
 inputTextArea.setText(result);  
 resultTextArea.setText("Encoded!");  
  
 inputText = inputText.toUpperCase(Locale.ROOT).replaceAll("[^A-Z]","");  
  
 if (inputText.length() > 0) {  
 resultTextArea.setText(String.format("Encoded! (Compression Ratio: %.2f%%)", (((float)result.length())/(inputText.length()\*7))\*100));  
 }  
 }  
}

### Decode button logic

Implementation: [*HuffmanCodingGUI.actionPerformed() Line:108*](src/HuffmanCodingGUI.java#L108)

if (source == decode) {  
  
 if (!inputText.isBlank())  
 result = huffmanCoding.decodeCharacters(inputText);  
 else  
 resultTextArea.setText("Empty Input");  
  
  
 if (!result.isBlank()) {  
 inputTextArea.setText(result);  
 resultTextArea.setText("Decoded!");  
 }  
}

# Testing

## Unit Testing

## Input Validation Testing

### Test Cases

# Academic Honesty

Any work that you submit for continuous assessment or assignments must be done by you. Failure to acknowledge the source of a significant idea or approach is considered plagiarism and not allowed. Academic dishonesty will be dealt with severely. At a minimum, you will receive a mark of zero for the assignment.

Signed: Alan O’Regan, Date: 22/11/2022