Kweku’s Portfolio

Data Structures Project

Ashesi University

Dr Ayorkor Korsah

**The readfile Method:**

public HashMap<String, Integer> readfile(String inputFile) throws IOException {  
 HashMap<String, Integer> mapStrings = new HashMap<> ();  
 FileReader file;  
 file = new FileReader (inputFile);  
 BufferedReader buffRead = new BufferedReader (file);  
 String line = null;  
 String data = "";

while ((line = buffRead.readLine ()) != null) {  
 data = line;  
 if (line != " ") {words = data.split ("");}  
 for (String i : words) {if (mapStrings.containsKey (i)) {  
 mapStrings.replace (i, mapStrings.get (i) + 1);}  
 else {mapStrings.put (i, 1);}  
 }  
 }  
 return mapStrings;  
}

For our team project, I was tasked with writing the readfile method. The method takes a file as a parameter; it reads the file line by line and returns the total count of each character in the file. The goal of the method was achieved by using a HashMap as the underlying data structure to store the characters and their corresponding total frequencies. Even though a HashMap was used the data returned was in an unsorted order. Also, I used the FileReader and BufferedReader classes from the Java io package.FileReader reads streams of characters and BufferedReader reads text from a character-input stream, buffering characters to provide for the efficient reading of characters, and lines. I did this because FileReader class reads bytes from the file and converts them to characters which is sometimes inefficient.

**The huffmanTree Method:**

public void huffmanTree() throws IOException {  
 PriorityQueue<huffmanNode> treeQueue = hufQueue ();  
 while (treeQueue.size () > 1) {  
 huffmanNode left = treeQueue.poll ();  
 huffmanNode right = treeQueue.poll ();  
 huffmanNode internalNode = new huffmanNode ();  
 internalNode.freq = (Integer) left.freq + (Integer) right.freq;  
 internalNode.data = "node";  
 internalNode.left = left;  
 internalNode.right = right;  
 root = internalNode;  
 treeQueue.add (internalNode);  
 }  
}

Next, I was tasked with writing the huffmanTree method. The goal of this method is to build a Huffman tree using the binary tree format where the internal nodes are created from the frequencies of two characters, and the leaves are the characters and their frequencies which is a child of an internal node. I created an instance variable of the hufQueue method written by Leonette. I extracted the first two nodes with the highest priority from the queue, created a new node called the internal node. The frequency of this node is the sum of frequencies of the nodes extracted. I set left, and right child as one of the nodes and I make the internal node the root of that tree. I do this iteratively until my queue is empty, then I have my Huffman tree. This was how our team was able to build the Huffman tree.

**The huffmanCode Method:**

HashMap<String, String> youTire = new HashMap<> ();  
  
public HashMap<String, String> huffmanCode() {  
 return huffmanCode (root, "");  
}  
  
private HashMap<String, String> huffmanCode(huffmanNode sroot, String store) {  
 if (sroot != null) {  
 if (sroot.right != null) {huffmanCode (sroot.left, store + "0");}  
 if (sroot.left != null) {  
 huffmanCode (sroot.right, store + "1");  
 }

else {youTire.put ((String) sroot.data, store);}  
 }  
 return youTire;  
}

Finally, the last achievement for this project is when I was able to write the huffmanCode method for my team. This method generates the Huffman encoding for each character, where the character with the highest frequency gets the minimum bits, and the lowest frequency gets the highest bit encoding. The data is stored in a HashMap. This method was written recursively by traversing the Huffman tree built by me. The base case was if the node has no children then I put that node and its corresponding encoding in a HashMap as key-value pairs because it is a leave node. If the node is an internal node, I do a left traversal of the left child, and I add 0 to the sting which stores the encoding. Then I go on to do a right traversal of the right child, and I add 1 to the string which stores the encoding. The map is returned at the end after traversing the whole Huffman tree.