Chrystal Mingo

CSC 22000

Professor Lucci

Methodology of Project 1

In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. Huffman encoding has the capacity to decrease the size of bits a file takes up, and minimizes it. I accomplish this minimization through building a huffman tree that takes in the input from the frequency of character to character count in the file The Raven by Edgar Allen Poe. For this project, I used the Huffman Tree to assign codes to the characters from the file, in hopes of compressing the file and gathering codes that take up less space than the original file. This Huffman Tree has a O(nlogn) where n is the number of unique characters. If there are n nodes, extractMin() is called 2*(n – 1) times. extractMin() takes O(logn) time as it calls minHeapify(). So, overall complexity is O(nlogn).

To parse through my file (The Raven) I created a special Java class to find the frequency of each character:

```
package sample.sam;
import java.util.*;
import java.io.File;
import java.util.Map;
import java.io.IOException;

public class HistogramLetters {
  public static TreeMap<Character,Double> getMap() throws IOException {
    HashMap<Character, Double> chars = new HashMap<Character, Double>();
    Scanner scanner = null;
    try {
      scanner = new Scanner(new File("the_raven.txt")); //This is where I stored the text from The Raven poem by Edgar
//Allen Poe
      while (scanner.hasNext()) {
        char[] line = scanner.nextLine().toLowerCase().toCharArray();

        for (Character character : line) {
          if (Character.isLetter(character)) {
```

```
                if (chars.containsKey(character)) {
                    chars.put(character, chars.get(character) + 1.0);
                } else {
                    chars.put(character, 1.0);
                }
            }
        }
    }
  } finally {
    if (scanner != null) {
      scanner.close();
    }
  }
  //Finds the frequency of each character
  for (Map.Entry<Character, Double> entry : chars.entrySet()) {
    System.out.println(entry.getKey() + ": " + (entry.getValue()));
  }
  //Use a comparator to sort the frequency in decreasing order and store it into a new map.
  Comparator<Character> comparator = new ValueComparator<Character, Double>(chars);
  TreeMap<Character, Double> result = new TreeMap<Character, Double>(comparator);
  result.putAll(chars);
  //returns the new map with freq of each character from the text the_raven.txt
  return result;
 }
}
```

```
a: 345.0
b: 95.0
c: 74.0
d: 195.0
e: 627.0
f: 93.0
g: 123.0
h: 293.0
i: 323.0
j: 2.0
k: 31.0
l: 230.0
m: 163.0
n: 379.0
o: 372.0
p: 97.0
q: 9.0
r: 343.0
s: 278.0
t: 448.0
u: 122.0
v: 68.0
w: 79.0
x: 3.0
y: 104.0
```

For my Huffman Tree I followed the steps provided online by Geeks4Geeks

Steps to build Huffman Tree:

Input is an array of unique characters along with their frequency of occurrence which was collected by the HistogramLetters java class and output is Huffman Tree.

1. The first step is to create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)

2. Extract two nodes with the minimum frequency from the min heap. (Over time the characters with the least count keep going lower and lower in the min heap)

3. Then you need to create a new internal node with a frequency equal to the sum of the two nodes frequencies. (This is how we continue to compress the files) Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.

4. Repeat steps#2 and #3 until the heap contains only one node. (That one node represents the character with the max frequency count) The remaining node is the root node and the tree is complete.

```
//from Geeks4Geeks
package sample.sam;
//import HistogramLetters;
import java.io.File;
import java.io.IOException;
import java.util.*;
// node class is the basic structure
// of each node present in the Huffman - tree.
class HuffmanNode {

  int data;
  char c;

  HuffmanNode left;
  HuffmanNode right;
}
// comparator class helps to compare the node
// on the basis of one of its attribute.
// Here we will be compared
// on the basis of data values of the nodes.
class MyComparator implements Comparator<HuffmanNode> {
  public int compare(HuffmanNode x, HuffmanNode y)
  {

    return x.data - y.data;
  }
}
public class HuffmanEncoding{
  // recursive function to print the
  // huffman-code through the tree traversal.
  // Here s is the huffman - code generated.
  public static void printCode(HuffmanNode root, String s)
  {
    // base case; if the left and right are null
```

```
    // then its a leaf node and we print
    // the code s generated by traversing the tree.
    if (root.left
            == null
            && root.right
            == null
            && Character.isLetter(root.c)) {
        // c is the character in the node
        System.out.println(root.c + ":" + s);
        return;
    }
    // if we go to left then add "0" to the code.
    // if we go to the right add"1" to the code.
    // recursive calls for left and
    // right sub-tree of the generated tree.
    printCode(root.left, s + "0");
    printCode(root.right, s + "1");
}
// main function
public static void main(String[] args) throws IOException {
    Scanner s = new Scanner(System.in);
    // number of characters.
    int n = 25;
    char[] charArray = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x','y' };
    int[] charfreq = {345,95,74,195,627,93,123,293,323,2,31,230,163,379,372,97,9,343,278,448,122,68,79,3,104};
    // creating a priority queue q.
    // makes a min-priority queue(min-heap).
    PriorityQueue<HuffmanNode> q
            = new PriorityQueue<HuffmanNode>(n, new MyComparator());
    for (int i = 0; i < n; i++) {
        // creating a Huffman node object
        // and add it to the priority queue.
        HuffmanNode hn = new HuffmanNode();
        hn.c = charArray[i];
        hn.data = charfreq[i];
        hn.left = null;
        hn.right = null;
        // add functions adds
        // the huffman node to the queue.
```

```
            q.add(hn);
        }
        // create a root node
        HuffmanNode root = null;
        // Here we will extract the two minimum value
        // from the heap each time until
        // its size reduces to 1, extract until
        // all the nodes are extracted.
        while (q.size() > 1) {
            // first min extract.
            HuffmanNode x = q.peek();
            q.poll();
            // second min extarct.
            HuffmanNode y = q.peek();
            q.poll();
            // new node f which is equal
            HuffmanNode f = new HuffmanNode();
            // to the sum of the frequency of the two nodes
            // assigning values to the f node.
//making the internal nodes
            f.data = x.data + y.data;
            f.c = '-';
            // first extracted node as left child.
            f.left = x;
            // second extracted node as the right child.
            f.right = y;
            // marking the f node as the root node.
            root = f;
            // add this node to the priority-queue.
            q.add(f);
        }
        // print the codes by traversing the tree
        printCode(root, "");
    }
}
```

t:000
l:0010
j:001100000
x:001100001
q:00110001
k:0011001
v:001101
u:00111
g:01000
c:010010
w:010011
s:0101
h:0110
i:0111
e:100
r:1010
a:1011
m:11000
f:110010
b:110011
o:1101
n:1110
d:11110
p:111110
y:111111