Ankush Wadhwa
G28234889
ankushwadhwa@gwu.edu

Project 2
Greedy Algorithm (Huffman Encoding)

2019/10/15

1 Problem Statement

Given a set of symbols and their frequency of usage, find a binary code for each symbol, such that: a. Binary code for any symbol is not the prefix of the binary code of another symbol. b. The weighted length of codes for all the symbols (weighted by the usage frequency) is minimized.

1.1 Theoretical Analysis

The given problem is an example of Huffman Encoding using Greedy approach. Given a set of distinct characters (n) with respective frequencies x_i, a symbol can be encoded into a set of binary values.

1. This can be achieved by using a binary tree with each unique symbol as the leaf node.
2. We can further make a min heap of the all the leaf nodes based on their given frequency of occurrence in the input.
3. Initialize the root node to null (empty tree).
4. Extract two minimum nodes from the heap
5. Add frequency values and store as new value in the heap.
6. Assign extracted values as left and right child of the root
7. Repeat until Heap is left with one node. Assign root node to this value

As mentioned above, the time complexity of a Huffman encoder depends on the number of unique symbols n in the input string. According to ASCII codes, 222 unique values of printable symbols exist (ASCII 35-ASCII 255). For the project, we will be taking into consideration all 255 possible distinct and printable symbols.
This is made possible by two methods, $randomCharListGenerator()$ and $randomFreqListGenerator()$ which generate a random list of arrays of varied sizes ranging from 0 to n for distinct symbols and their respective random frequency of usage/occurrence.
The $HuffmanEccoder$ class takes as input the symbol arraylist, the frequency arraylist and the symbol count upon initialization. The $encoder()$ method in the class encodes the the values using Huffman encoding technique and return the Huffman tree.
The $decoder()$ method decodes the binary tree into a hashmap containing the symbols and the respective encoding binary string, satisfying the required two conditions.
The $printHashmap()$ method prints the hashmap which it takes as an input parameter.
Priority Queues are used for the purpose of the project as a data structure to facilitate a min-heap
Time Complexity:
The $PriorityQueue.poll()$ method takes $logn$ time as it calls minHeapify(). For n unique nodes, the method is called 2(n-1) times. Therefore the time complexity of the problem is $\mathcal{O}(nlogn)$

2 Experimental Analysis

2.1 Program Listing

```
ArrayList<char[]> charArrayList = randomCharListGenerator();
ArrayList<int[]> charFreqList = randomFreqListGenerator();
long startTime = initTimeCal(csvWriter);
for (int n = 2; n < 222 ; n++) {
    HuffmanEncoder huffmanEncoder = new HuffmanEncoder(n, charArrayList.get(n
        -2), charFreqList.get(n-2)); // init input
    Node root = huffmanEncoder.encoder();          //Encode values
    huffmanEncoder.decoder(root, "");   //Decode Values
    printHashmap(huffmanEncoder.getHuffValues());   //Print values
    closeCSV(csvWriter, n, startTime);
}
```

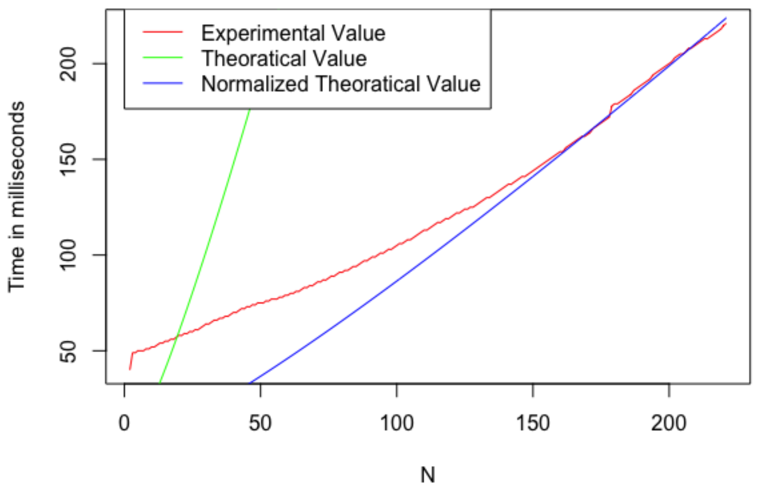
2.2 Data Normalization Notes

Following are the computed Experimental Values for the program with the Theoretical values and the normalized Theoretical Values.
For the purpose of normalization of theoretical values, a normalization constant of 0.187654 is used. The constant is derived by averaging the divided values of experimental values to theoretical values.

2.3 Output Numerical Data

N	Experimental Value	Theoretical Value	Normalized Value
2	40	1.386294	0.2601437
10	52	23.02585	4.320893
20	58	59.91465	11.24322
30	64	102.0359	19.14745
40	70	147.5552	27.68932
50	75	195.6012	36.70534
100	105	460.517	86.41786
150	144	751.5953	141.0399
175	169	903.8375	169.6087
200	200	1059.663	198.8501
222	223	1199.394	225.0712

2.4 Graph



Plotting the values for the problem.

2.5 Graph Observation

The experimental value gives an almost curved line with an offset on y axis. The theoretical value of the computed Time complexity (green) diverges with the experimental plot of values. After normalization, the newly plotted line (blue) converges with the experimental value of plots. The overall trend of the plot is satisfactory.

3 Conclusion

The project gives an insight into Huffman Coding for varied size binary variables.
For the given problem (problem 9), the time complexity is $\mathcal{O}(nlogn)$
The graph shows how the computed experimental values for the time elapsed converge with the normalized theoretical values of the asymptotic function.