Veet Zaveri

Lab 3 Analysis

605.202 Data Structures

04/06/2022

The data structures I used is trees which is a nonlinear data structure type that allows the implementation host of algorithms faster than linear data structures. With trees the relationships are hierarchical with certain objects “above” and certain objects “below” the other ones. Trees are recursive data structures since they are defined recursively. Each tree has a set of elements or nodes since each node is linked to its successors. Huffman trees represent Huffman codes for characters that could be in a text file. Huffman code uses different numbers of bits to encode letters. Huffman code uses variable length encoding. Huffman code uses short code-word strings to encode high-frequency characters and long code-word strings to encode low frequency characters so that it saves space compared to fixed-length encoding.

I used trees because to build the Huffman encoding tree so that a preorder traversal can be implemented, and ties can be resolved by giving single letter groups precedence over multiple letter groups.

There was useful data compression achieved with this method because since it reduces the size of the data using variable bit length for different ASCII characters. To compare this to conventional encoding methods we can use an example. Lets consider "HELLO" as a sample text to compare. which has 5 ASCII characters each contain 8 bits. Then conventional encoding will require 5*8 = 40 bits of bandwidth to transfer "HELLO" word. but in the case of Huffman Encoding method only 21 bits are required for this word, instead of 40. If a different scheme is used to break ties it will result in different output code but does not matter if you are using the same scheme for both encoding and decoding. Other data structures I used to implement the Huffman encoding algorithm are Priority Queue, Binary Tree and Node Structures. I took the frequency table as dictionary parameter to perform Huffman coding and a while loop to perform the merging of the nodes.

The time and space efficiency for this is O(nlogn). A heap is used to store the weight of each tree, so this means that for each iteration to find out the cheapest weight it needs O(logn) time. There are also O(n) iterations since there is one for each one, so the time and space efficiency turns out to be O(nlogn).

I learned how Huffman coding technique works which is through making a binary tree of nodes which contain the character and the weight of the character. A node can be a leaf or internal node and to begin with all nodes are leaf nodes. Huffman trees should get rid of unused characters in the text so that the code length is optimal. Also, that a priority queue can be used for building the Huffman tree since the node with the lowest frequency has higher priority over others.

Although there is not much I would do differently next time, I would want to explore how to encode and decode strings by using tkinter or a GUI where I would be able to manually select if I

want to encode or decode a string. However, when decoding the same key has to be used for encoding.