Dion Niazi dn3gy 19 04 2017 postlab10.pdf

I implemented quite a couple of data structures for my pre and in lab Huffman tree. For the encoding and decoding lab the data structures I used were vectors, map data structure, a heap, and heap nodes.

Encoding Lab

I first made my heapNode class to hold the information of the character to encode and the pointers to the left and right node. I then included getter subroutines to be able to get and set new data within the node. I then made a heap class that was much easier to read and understand and that helped to produce a min heap. After creating the heap class, I went and created the main file for the encoding part of the lab. I first read in the file if the file name was provided along with the executable name, then it the file is read character by character. I save each character into a vector<char> and I pop the last element, which is usually an unnecessary space. I also made another vector to save the original characters for later, when I want to output the original file contents. I then sort the vector to be able to count the frequency of each char that showed up in the file so I can build the Huffman tree. I also made another vector to hold the frequencies for each character. It works by going through and checking if the next element is equal to the current element and if so the frequency of the current element in the frequency vector increases by one else you put 1 into the last section of the frequency vector. I then make a heap object and place nodes with a frequency corresponding to each character. The heap automatically stores the nodes in a particular order of nodes from the lowest number of frequencies to the highest at the bottom. This helps with the Huffman

encoding by using the Huffman algorithm by storing the 2 lowest nodes into another node, but with the left and right nodes pointing to the two lowest nodes and the node holding those nodes are “empty”. I go through the created tree and store the prefix code into a string and once I reach a leaf node, I then put the prefix code into a map along with the character that it was encoded for. I believe the worse running time to be big theta n and for each vector the space complexity is also big theta n because its unknown how many characters will be in the vector.

Decoding Lab

For the decoding part of the lab I made the tree based on the frequencies of each node. So how I build the tree was by building the root node first and then looking at each character, I recursively went through, bit by bit to add the nodes and once the last bit is reached then place the node with the character in it. Once the tree was built then to decode the frequencies I go through each bit and once I receive to a last node I output the respective character and then go back to the top of the root node and continue until I finish decoding. I believe the tree to be big theta log n and its space complexity to be the same.