Jhf5my - John Fishbein Tuesday, April 16, 2019

Post-Lab 10 Report - Huffman Encoding/Decoding

The first step in a Huffman encoding of a string of text is to determine the frequency of each character that appears in the text. This step is always linear ($\Theta(n)$) time because in order to determine the frequencies, every character in the string must be processed. These frequencies are then used to build a min heap of the characters in which the characters are prioritized by their frequencies. This step is worst case $\Theta(n\log n)$ because each insert in to the heap takes worst case $\Theta(\log n)$ and n elements are inserted. After the min heap of character frequencies has been created, the next step is to create the Huffman prefix tree. This step involves two calls to the heap's deleteMin() method and then another call to heap insert(). These three method call's must happen n times until there is only one node containing the entire prefix tree. Therefore this step also takes $\Theta(n\log n)$ worst case time. Once the Huffman tree has been created, I chose to use the stl::unordered_map data structure to map each unique character to its specific Huffman encoding. The insert into this data structure occurs in constant time. However, the operation to build the path to each unique character in the Huffman tree is worst case $\Theta(n)$ time because in the worst case the entire tree must be traversed to locate a specific character. Therefore, the overall procedure of inserting all of the characters into the unordered_map takes worst case $\Theta(n^2)$ time. Once this mapping is constructed, the original string can be encoded in a linear process. Each character must be read and converted to its Huffman encoding using the map taking $\Theta(n)$ time. This finally results in a fully encoded string.

The first step in decoding is to read in the map of character, encoding pairs. This step takes linear time because each character must be processed individually. Once each encoding was read in, the Huffman tree was recreated. The process of inserting a new character into the Huffman tree takes worst case $\Theta(\log n)$ because in the worst case, it is inserted at the very bottom of one of the subtrees. Thus, the overall process of building the tree is a worst case $\Theta(n\log n)$ operation. Once the prefix tree has been recreated, each bit is processed and the tree is traversed based on that bit. If a leaf node is reached, that character is added to the output and then the process continues starting back at the root node. This process in always $\Theta(n)$ because each bit is processed individually.

In terms of space complexity, the minHeap used in encoding holds n huffNode

pointers. This means that the minHeap requires 8*n bytes. Each huffNode contains two huffNode pointers along with char and int fields. Therefore each huffNode requires 21 bytes (16 for the two pointers, 1 for the char, and 4 for the int).