Jessica Krynitsky jmk8vr

4/20/18

CS 2150 Postlab 10 Report

Encoding

To determine the frequencies of the characters in the file, I created an array with length 128. Each index represented the ASCII value of a character, and stored in the array were the frequencies of that character. This step had linear time complexity in terms of the number of characters in the file. The space complexity is constant because it is an array.

The next step was to create the heap. I did this by looping through the array of characters and frequencies, and if there was a frequency greater than 0 I created a new Huffman node with the values for character and frequency and added it to the heap. The expected running time for inserting into a heap is constant, so this operation is linear in terms of the number of distinct characters for both time and space complexity.

I created the tree by looping through the heap, deleting the two minimum values, and setting them as the left and right children of a new node. Once this process finished, I had one root node for the Huffman tree.

The Huffman codes were determined and printed in one recursive function. The base case was if it was a leaf node, add the character and its code to a map and print out the character and code. If the node had a left child, “0” was added to the code and the function was called on the left child. The right child was similar except a “1” was added to the code. This step was linear time complexity in terms of distinct characters in the file. It is also linear space complexity in terms of distinct characters because of the map.

To print the encoded message, I simply read the file one character at a time and outputted the code value in the map I had made. I chose to use a map so that this operation would be constant time. Therefore, this step is linear time complexity in terms of total characters in the file, and it is constant space complexity.

Decoding

I read the prefix code structure from the file and created the Huffman tree in one step. Every time a character was read in, a recursive function was called on the root node, the character, and its prefix code. If the first bit of the code was a 0 or 1 the tree went left or right respectively, creating a new node if necessary and calling the function on the subtree and the rest of the prefix code. The base case is that the prefix length is

0, and the node’s character is set to the corresponding character value. This operation is space/time linear in terms of the number of distinct characters in the file.

Next the encoded message was read in from the file and stored as a string of 1’s and 0’s. This step took linear time complexity in terms of the total number of characters in the file and it is constant space complexity.

Finally, I used a while loop to look at each character in the string one bit at a time. I set a node equal to the root of my Huffman tree, and then iteratively determined if the bit was a 0 to set that node equal to its left child, and the same for the right child, until a leaf node was found and the character at that node was printed. The node was then reset to the root of the tree and continued to read the bits. This step was linear time complexity in terms of the total number of bits in the encoded file and constant space complexity.