Noah French (njf5cu)
Lab 10
11/26/17
Filename: postlab10.pdf
File Description: Lab 10 Time and Space Complexity Report

Compression

1. Read the source file and determine the frequencies of the characters in the file.

Within the provided loop that saves each character as a char named g, I first cast the char as a string named s. Then, if the string was " ", I changed it to "space". I stored the chars in a vector and the counts in a second vector with corresponding indexes. The worst case runtime would be n + n!, where n is the number of characters being looked at. Each character is handled once, and within that handling there is a for loop that could, at worst, loop through all the previously handled characters. The worst-case space complexity is 2m, where m is the space needed to store the chars, assuming the space to hold a char/string is equal to the space to hold an int. Each char is saved with its corresponding count. No extra space is allocated.

2. Store the character frequencies in a heap (priority queue).

I loop through the vector of characters, create a HuffNode for each character and its corresponding count, and then insert it into the heap. Insert is performed n times, and insert is worst-case log n, so this step is worst case n(log n) runtime. The worst-case situation is n(h + v), where n is the number of characters, h is the space required to make one HuffNode, and v is the space required to hold one vector index.

3. Build a tree of prefix codes (a Huffman code) that determines the unique bit codes for each character.

I create a dummy HuffNode, pop off the two minimums from the top (a constant time operation), set the two minimums to be the children of the dummy, then insert the dummy into the heap. The worst case runtime for this step is n, where n is the number of characters being handled. The worst-case space situation is n(h + v) from the previous step plus the new dummy nodes being created.

4. Write the prefix codes to the output file, following the file format above.

Outputting the prefix codes involves reading every node on the prefix code tree and storing the prefix code and character in vectors with corresponding indexes. Reading the tree is worst case n + d time, where n is the number of characters, and d is the number of dummy nodes, because each node is visited once. The additional space required is just the memory needed to store every character and the space to store every prefix code.

5. Re-read the source file and for each character read, write its prefix code to the output, following the file format described herein.

This step is simply linear time, based on the number of characters. Some additional space is used to store temporary calculations for output.

Decompression

1. Read in the prefix code structure from the compressed file. You can NOT assume that you can re-use the tree currently in memory, as we will be testing your in-lab code on files that you have not encoded.

The worst case runtime for this step is linear, depending on the length of the file being read. The characters are stored in a vector and their corresponding indexes are stored in a second vector. Thus, the worst case space usage is linearly dependent on the number of characters being read in.

2. Re-create the Huffman tree from the code structure read in from the file.

The worst case runtime for this step is n, where n is the number of nodes that needs to be added. The worst case for space is simply the space allocated by the HuffNode leaf nodes and dummy HuffNodes.

3. Read in one bit at a time from the compressed file and move through the prefix code tree until a leaf node is reached.

4. Output the character stored at the leaf node.

I accomplished both of these steps with a recursive “readTree” method. The “find node” operation is log n time, and it is carried out n times. Thus, the overall process’s worst case runtime is n(log n). The function I wrote to complete this step printed out the character every time a leaf node was reached and then continued to follow the prefix instructions at the base of the tree.