Stephen Chew (ssc6ae)
Post-lab, 11:00am

Time Complexity

Compression

During my compression process, I first inserted every element in the text file into a map. The worst case big-theta performance for this is log(n). From there I iterated through the map and inserted each element into my min heap. To retrieve data from a map is also log(n) time and the worst-case runtime to insert into a minheap is log(n) because each element will be inserted at the heap and will percolate up if needed. However, the expected runtime to insert is constant. After the tree is built, to write the prefix codes, I created another map to hold the characters as a key and prefix codes as a value. To remove the minimum element from a min heap takes log(n) time because the tree needs to percolate to balance itself. Map operations are log(n) time. From there, I printed out the key value pair. This is also log(n) time.

Decompression

In my decompression code, I first read in all the prefix codes and used a recursive function to build a Huffman tree. I believe the time complexity to do this is logarithmic because each element that is inserted will be placed at a height that is less equal to or less than $\log_2(\text{nodes}+1) - 1$. After the Huffman tree is created, I read in each bit from the file and traversed the Huffman tree until the pointer reached a leaf node. To traverse the tree takes log(n) time. This process is repeated for each set of bits (character), therefore it will actually take (n)log(n) time.

Space Complexity

Data structures

The primary data structures I used are map and my min heap object.

Map: I used two different maps during the encoding phase of my code. The first map contained a char as the key and an integer as the value, the second map contained a char as the key and a string as the value. An integer contains 4 bytes of memory; a char contains 1 byte of memory, and a string contains one byte more than the actual number of characters in the string. Since the first map only had a key-value pair for each unique element in the text file, the amount of memory required for this data structure is (88 bytes for the map structure + 1 byte for each char + 4 bytes for each integer). The amount of memory required for the second map is (88 bytes for the map structure + 1 byte for each char + an unknown number of bytes for the string).

Min heap: my min heap data structure contains Huffman nodes. Each Huffman node contains an integer for the frequency of the character, a char to represent a character, and left and right pointers to either null or other nulls. Since a pointer takes up 4 bytes of memory, each Huffman node created will take up 13 bytes of memory, and the amount of memory my min heap will require is (13 bytes) * (number of nodes in the min heap).

Sources:
http://jsteemann.github.io/blog/2016/06/14/how-much-memory-does-an-stl-container-use/