Luke Jones

Inseon Kim

ECS36C HW4 Report

In order to compress the file, we first get the frequencies of every character from the input file using an array of 128 ASCII characters. Then we build a priority queue of Huffman Nodes representing each character and its frequency. We then create a Huffman Tree out of the priority queue so that we can create the code table by traversing the Huffman Tree and deleting the nodes as we go. Using the code table we write every character with its new bit code into the output file.

In order to decompress the file, we rebuild the Huffman Tree stopping when every internal Huffman Node has two children. As we rebuild the Huffman Tree, we create the code table and delete the Huffman Nodes. We then get the total number of characters from the original file by getting the next 32 bits representing an integer. Using the code table and the total number of characters, we decrypt the rest of the bits into their corresponding characters and put them into the output file.

To implement our priority queue class, we percolate when items are added or removed in order to keep the structure of the user-defined priority queue. After implementing the priority queue, we tried to test every function for edge cases and exception throws along with its functionality. We tested whether calling pop or top on an empty priority queue would throw an underflow error. Then we tested whether the priority queue could handle custom objects and comparators by inputting a custom object and comparator into the priority queue.

To implement our binary stream class, we either get or put the number of bits corresponding to the type size. After implementing the binary stream class we tried to test for every edge case and exception throw. For the exception throw, we check whether attempting to read bits when there isn't a full byte in the file causes an underflow error. To test the edge cases, we inputted a random assortment of integers, characters, and bits. We then checked whether the inputted bit string matched what was expected.

When deciding which std exception to throw in the priority queue class when users attempt to call pop or top on an empty priority queue, we chose underflow error because underflow applies when trying to access elements from an empty container.