Pavittar Singh and Benjamin Stone
Professor Porquet-Lupine
ECS36C
19 May 2022

Homework 4 Report

PQueue:

In making PQueue, not much needed to change from the priority queue made in class. Because the PQueue has to grow and shrink as elements are pushed and popped, we edited those functions so that cur_size would stay equal to the size of the internal vector. Thus for Push(), we can just push_back the new item into the vector, and for Pop(), we can simply pop_back the end of the vector after making the root equal to the vector's end. For the Percolate functions, the only change needed was to replace direct node comparisons with calls to CompareNodes(), which simply uses the comparator passed through the template. In testing PQueue, we added more tests for both std::less and std::greater in order to make sure they both worked properly. To test a PQueue of pointers to a class, we made a struct that compares pointers and passed it through the PQueue template as a comparator.

BStream:

For the BinaryInputStream, we know that a char is made up of 8 bits and therefore we need to get 8 bits to read one char. That's exactly what GetChar does, it does eight iterations and gets 8 consecutive bits and returns the char. An int is made up of 32 bits and therefore GetInt gets 32 consecutive bits and returns the int. For the BinaryOutputStream, we again know that 8 bits make up a char and therefore to output a char param, we need to get the 8 bits that make up the char and output them from highest to lowest. This is what GetChar does. GetInt does the same thing except it needs to get the 32 bits that make up the int param. For testing, all the input functions were tested alongside all the output functions. They were then tested together to make sure they can stand alone without hard coded constants.

Huffman:

To compress a file, we start by making a map of each unique char and its frequency. Then we push the elements of the map into a PQueue for the purpose of being able to pull them out of the PQueue in the order of their frequencies and their characters. Constructing the tree simply follows the algorithm presented in the homework assignment. Similarly, outputting the tree follows the simple algorithm of preorder traversal. We subsequently make a map containing each character and its path by traversing the tree in the same way. Lastly, we output the paths by referencing this map we just made. To decompress a file, we start by rebuilding the tree, using a recursive function that follows the logic of preorder traversal. Lastly we output the characters by iterating once for each character and then following the paths presented in the input stream.