CS5310 Assignment 3

Huffman Trees

Jonah Kubath

In this assignment, we are to implement a min-heap that is used to build Huffman trees. The min-heap that I chose to implement was the build in Java implementation of a Priority Queue. The Java documentation states that a PriorityQueue will implement an array-based queue. The inserted values are inserted based on a natural ordering or a comparator can be used upon construction of the PriorityQueue.

The Java documentation also states that enqueing and dequeing methods offer O(log(n)) time. Linear time for removing a certain object and checking for a certain object. Constant time is given for retrieval methods (peek, element, and size).

As my results show, the heap is built in linear time. There is almost no change in time when building smaller heaps, such as 10 – 100 nodes. The time change is easier to see when the number nodes has a larger step between tests. The 1000 – 10000 shows a steady increase in build time for the heap. There is one outlier in my data. The last tree to be built in the 100 -1000 test has a steep increase in build time. This can probably be explained by a spike in system processing while the application was running.

In my trials, the time for creating the Huffman tree takes longer than building the heap. When the Huffman tree is made, the two lowest frequency nodes are combined into one node and this node is then inserted back into the min heap. Getting the first lowest node is done in constant time as it is the root of the min heap, but once this node is removed it takes O(log (n)) to remake the min heap before the next lowest can be removed. The two nodes are then combined in constant time and adding the combined node back to the min heap is done in O(log (n)) just like the removal.

The building of the Huffman codes can require each node to be iterated over before finding the wanted node. The reason Huffman trees are used in practice is that the highly used codes ( high frequencies) are created in the lower levels of the tree. This requires less traversing for the more frequent codes and a longer traversal for lesser used codes.