Makram Adaime 40033315

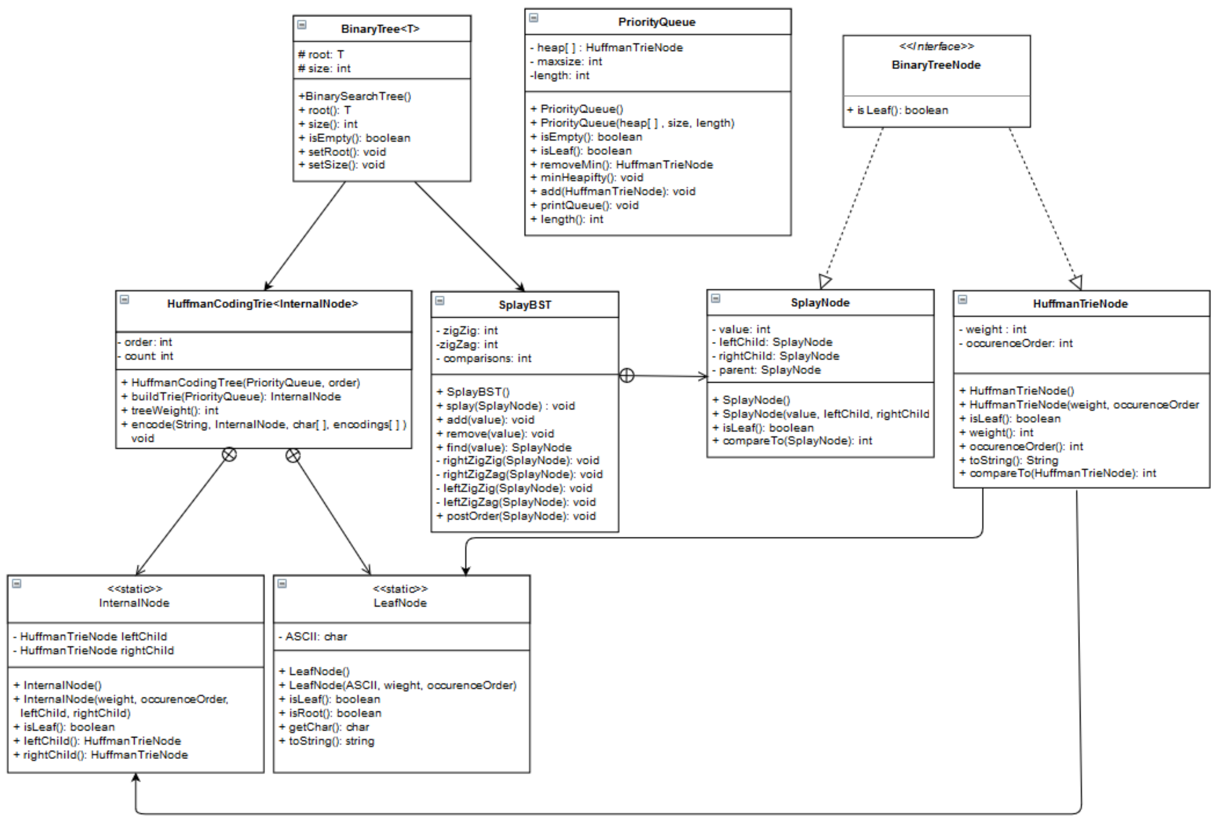
Comp 352 Assignment 3

1.

My general tree structure starts with the Binary Tree class. Given that a Splay Tree and a Huffman Trie don't have much in common in terms of how they operate (their only commonality being that they are both binary trees) not many core methods can be defined in the BinaryTree class. The class's flexibility comes from the generic type parameter which should represent any type of tree node. The BinaryTreeNode interface itself also is very limited given that I opted to separate the Huffman nodes into leaf and internal nodes. Because the leaf nodes do not point to any children, the BinaryTreeNode interface cannot enforce that all nodes implementing the interface possess left and right child nodes.

My Huffman structure is composed of 4 main classes, HuffmanCodingTrie which extends BinaryTreeNode, HuffmanTrieNode which implements the BinaryTreeNode interface and with InternalNode and LeafNode classes extending HuffmanTrieNode. I included the two subclasses as static inner classes simply to have a more logical grouping of the classes. Unlike the SplayNode inner class, there is no substantial advantage to doing so, it was simply done for convenience. I chose to implement the Huffman Coding Trie with a min-heap based priority queue. The queue itself is built by adding the nodes one by one using the add() method. It is initially composed of only Huffman leaf nodes which are initialized from the frequencies table and occurrence order table arrays. I only reheapify the queue once I pop its root to build the Huffman Trie. The max size of my heap array represents the final size of the Huffman coding trie. I initialize it this way to move recently popped nodes to the end of the heap array while having the newly formed Huffman internal nodes take their place in the queue.

My Splay Tree structure is composed of 2 classes, SplayBST which extends BinarySearchTree and its inner class SplayNode which implements the interface BinaryTreeNode. Unlike the Huffman nodes, the splay tree nodes are not separated into leaf and internal nodes. The node includes pointers to left and right children and a parent pointer as well. I use 4 methods for rotations (one for each of the 4 different kinds or rotations). The advantage of Splay over AVL is that I don't have to worry about keeping the tree balanced at all times. Moreover, a splay tree is advantageous here because there are a large number of operations to be done which amortizes the cost.



4. a)

The fixed-length ASCII encoding is 608 bits while the Huffman encoding length is 338 bits (a 44.4% decrease). I believe that the encoded string does indeed match up pretty well with the source text. Looking at the character frequencies, the top 2 most frequent characters are by far ‘space’ and ‘e’ in both texts. The rest of the characters are a wash. They are of lesser importance, especially given the length of the string.

4. b)

I learned a great deal on how to manipulate nodes in a tree. The decision to include a parent pointer was initially to make it a little easier to splay and rotate between the different nodes in the tree. However, while the increased upward flexibility is nice, the extra overhead did make me more prone to mistakes when linking the different subtrees back together. Again, I believe the Splay Tree is the correct choice here because of the large number of operations needed to be performed.