Class Functionality

Jake Osselaer &

Quincy Schurr

**AVL2**

This class serves as an AVL Tree that is specifically for Word\* objects. We had a hard time getting Word\*’s to work with our templated AVL tree so we created a new one specifically for Word\* objects.

*Quincy worked on this class.*

**AVLNODE.h**

This class is just a struct for the AVL node that is used in the AVL2 class.

*Jake worked on this class.*

**AVLTREE.h**

This class is a templated class for the AVL Tree. The functionality includes inserting, rotating to keep the height, printing and returning and searching for objects.

*Quincy worked on this class*

**DOCUMENT PARSER**

The Document Parser class’ main functionality is that it brings in the info from the xml file using RapidXML. The getInfo function parses that data first by removing all punctuation, digits, and non-ASCII characters from the code. Then the word is transformed to lowercase, tested to see if it is a stop word and then stemmed. If the word is stemmed then the word is added to a Word object. We also keep track of pages that word appears on and then frequency in this function as well. After all the words are put in either an AVL tree or a Hash table of AVL trees the data structure prints out the data to output files.

*Jake and Quincy both worked on this class.*

**HASH TABLE**

The hash table functionality creates buckets of AVL trees that the words are stored in based on a certain hash function. Since the hash table is a hash of AVL Trees the functionality needed to be able to get the root from the AVLTree class but we can also insert words, search the tree and return a Word\* object.

*Jake worked on this class.*

**INDEX HANDLER**

Index Handler just determines how we want to parse the data using either an AVL Tree or a Hash Table.

*Jake and Quincy.*

**PAGE**

A simple class with getters and setters for the Page object.

*Quincy*

**QUERY**

This class has a lot of functionality. The first thing this class does is bring back in all the words that were indexed out to an output file and rebuilds the Words from the output file. Once the words have been added to the hash table data structure the user can input their query. Those arguments are then used to take the set\_union or difference or intersection of the final pages that the query appears on. Then we take the frequency just by adding up each frequency of each word on each page. If the user chooses to view a page then the page text is brought in and the printed to the terminal.

*Jake worked on the query itself and Quincy worked on re-indexing the words and pages.*

**USER INTERFACE**

User interface gave the users the option of things to do when the program first starts and calls different functions based on the input argument.

*Jake and Quincy.*

**WORD**

Getters and setters for the Word object as well as inserting into Word and printing.

*Quincy*