//Student number jamorran(17699381) Joel Andrew Morran

Section 1

I implemented the WordMatch, AVLTree and AVLNode classes. For the WordMatch class my methods where Read, readFiles, writeFile, patternMatching and match. For Read I would read in the file containing the names of the files to be read using the ability to read in from the command line using args i.e (java test file1.txt), for readFiles I would read the files that where provided by the read method and read and split them and add them to a unique word string array, for writeFile I would take the unique word and there neighbours and write the words to the lexicon, for patternMatching I would take my provided patterns read them from there file and search for them in the arrayList of patterns, for match I would take pattern and check that it matched another pattern.

For AVLNode I used the basic AVLNode file provided in the labs but added a addCount, getCount, getBorder(neighbouring words), getBorder2 to remove the , and white space from the end of the array of neighbours (I.e. hs, bd, = hs, gb), setBorder(neighbouring words) and toString method.

For the AVLTree I used the basic AVLTree files provided in the lab but added the findingBorders and isBorder methods, findingBorders takes in data and calls isBorder to search the AVLTree to find If it is a neighbouring word or not which is more efficient than using selection sort like I did at the AVLTree automatically sort all data it takes in in a tree structure, I also use fixBorder I have and public and private method to get past the second method being private this is used to remove the , and white space from the end of the array of neighbours (I.e. hs, bd, = hs, gb), which splits the string and the private and public sortBorder methods which is also used to get past the second method being private this method uses selection sort to sort the borders(neighbours) in alphabetical order but does don’t add additional time as it is still faster than the pattern matching which is taking place.

Section 2

The main issues in my effort to make the program run efficiently would be the searching as AVLTrees are more suited to making sorting efficiency and sorting of the text files which is why I have implemented AVLTrees to improve the execution time of my program over just using selection sort also I was using linked lists in areas but I found that replacing them and using array lists was more efficient for as the program would run in less time although this was only a very small gain in time compared with the implementation of AVLTrees, I also selection sort to sort the borders(neighbours) in alphabetical order but does don’t add additional time as it is still faster than the pattern matching which is taking place.

Section 3

I tested the functional correctness of my program by checking the time it takes to do run as well as Reading the output files by making sure that the output that is being written to my files is as expected and correctly formatted.

Section 4

I tested the efficiency of my program by running an unoptimized version that was only using selection sort and 5-6 sets of patterns, but all 5 of the test files and I compared the amount of time this took to run which was so long that I stopped it after 4 hours, to the amount of time my optimised program took to run under these same conditions using AVLTrees which was around 19 minutes which is much better considering the near indefinite nature of the unoptimized program.