Name: Jonathan Henley (jth106)

Project partner: Raza Agha (raa117)

Design: the design of the project revolves around three separate data structures a binary search tree (Custom\_BST), hashtable (CustomHashtable), and array (FrequencyArray). Uses these three different classes, we can show the differences in times it takes to build the data structure, find the frequency of a word and find the word with the maximum frequency. We repeat this experiment with three files, each containing 1000, 10,000 and 100,000 words respectively to see how the running time of each data structures grows.

Custom\_BST:  
  
This is a Modified Binary Search Tree. Each Node contains a String Word and the frequency of the Word. The tree is created by using the ascii value of each word. If the word is not present in the tree then it is inserted in the appropriate place (by using its ascii value) otherwise the words frequency is incremented by one. Using the ascii value makes it easy for us to locate a word and print out its frequency. The find\_largest() function traverses the whole tree and prints out the word(s) with the greatest frequency.

CustomHashtable

This class extends the hashtable in java library util. This class has a modified put function that only takes a string as input in order to add to the hashtable new words being sorted with an automatic frequency of one and repeated words have their values increased by one. This class also contains the function to determine and print out the most frequently used word.

FrequencyArray

This class extends the ArrayList class from the java library. This class uses a modified add() and contains() methods. The add method takes a string and uses the contains method to determine if the word is already in the array. New methods created for the class include find() and findLargest(). Find takes an input of a string then iterates through the array while find largest just iterates to find the most frequently used word. The class uses a separate class called WordFrequency to keep the word and it’s frequency in the same place. This class is solely used for this array it contains simple getter methods, a constructor, comparator, and an increment frequency function which increases the frequency by one it is called whenever a word to be added is already in the array.

The main class contains two function main and read. The read function uses the Scanner class to read the file. It parses only words in the text file (ignoring numbers and punctuation). It then stores all the words in the array called token. Each word in token is then traversed to build the different data structures and then the relevant functions are applied on the data structures. The time is noted for each function and outputted. The main function passes the three files to the read function and produces the output which is given below.

Our output from main which uses three text documents with 1,000; 10,000; and 100,000 words respectively (contents include the plays Hamlet, Henry the VIII, and Rome and Juliet). What they show is that the Binary Search Tree is the fastest to create and store the frequency of words.

arraylist in milliseconds = 10; arraylist in milliseconds = 408; arraylist in milliseconds = 1503 Here it can be seen that with each increase of a factor of ten the time to create the ArrayList increases by a large multiple first forty then four making this by far our least efficient data structure. (it's also the slowest for small values)

Binary Search Tree in milliseconds = 10; Binary Search Tree in milliseconds = 8; Binary Search Tree in milliseconds = 20 (1000 words; 10,000 words; 100,000 words)

The binary search tree is the most efficient with a slight decrease on a factor of ten increase in size followed by slightly under a factor of three increase for the second word increase by a factor of ten.

Hashtable in milliseconds = 0; Hashtable in milliseconds = 8; Hashtable in milliseconds = 30

(1000 words; 10,000 words; 100,000 words)

The hash table starts off as the most efficient so if one needs to find the frequency of all the words in a short document this is the most efficient option. However it consistently increases at a faster rate than the Binary search tree making it slightly less efficient for larger text documents.

The following data is the time it takes each of these data structures to operate with 1000 words:

the most common word is: "the" which was written 48 times

time to create and find the most frequent word in a arraylist in milliseconds = 10

the frequency of "horatio" is 21

time to find and print a randomly chosen word = 0

The words with greatest frequency is: 'the' with frequency= 48

time to create and find the most frequent word in a Binary Search Tree in milliseconds = 10

the occurrence of the word horatio: 21

time to find and print a randomly chosen word 0

the most common word is: "the" which was written 48 times

time to create and find the most frequent word in a Hashtable in milliseconds = 0

the occurrence of the word horatio: 21

time to find and print a random word from the hashtable milliseconds = 0

The following data is the time it takes each of these data structures to operate with 10,000 words:

the most common word is: "and" which was written 329 times

time to create and find the most frequent word in a arraylist in milliseconds = 408

the frequency of "horatio" is 85

time to find and print a randomly chosen word = 1

The words with greatest frequency is: 'and' with frequency= 329

time to create and find the most frequent word in a Binary Search Tree in milliseconds = 8

the occurrence of the word horatio: 85

time to find and print a randomly chosen word 0

the most common word is: "and" which was written 329 times

time to create and find the most frequent word in a Hashtable in milliseconds = 8

the occurrence of the word horatio: 85

time to find and print a random word from the hashtable milliseconds = 0

The following data is the time it takes each of these data structures to operate with 100,000 words:

the most common word is: "the" which was written 3245 times

time to create and find the most frequent word in a arraylist in milliseconds = 1503

the frequency of "horatio" is 157

time to find and print a randomly chosen word = 0

The words with greatest frequency is: 'the' with frequency= 3245

time to create and find the most frequent word in a Binary Search Tree in milliseconds = 20

the occurrence of the word horatio: 157

time to find and print a randomly chosen word 0

the most common word is: "the" which was written 3245 times

time to create and find the most frequent word in a Hashtable in milliseconds = 30

the occurrence of the word horatio: 157

time to find and print a random word from the hashtable milliseconds = 0

Special thanks to william shakespeare for providing a never ending supply of words

Process finished with exit code 0