Navid Hariri
David Quines
Computer Science 282
May 1st, 2019

Status

Status:

The current project compiles and runs, and the program produces the correct results. We tested the program under many different circumstances, and it worked correctly. We event tested the program with extreme inputs to be sure that the project works under any circumstance, and it still produced the correct output. The driver class reads through both the key and the text file stripping away any punctuations or spaces that could be attached to the program. The driver then takes the text file and uses it word by word and removes any punctuation or space that could be attached to the word using regular expressions. Then takes that word and turns it lower case to compare only the words and not allow the capitalization to affect the comparison of the word.

Development Process:

Firstly, we needed to find how to populate the keys in the correct order to be hashed into the hash table. Our process first started with finding the frequencies of the first and last letters of each key. This was to find the total value of each word. We did this by finding the said letters and converting them to their ascii value. We then stored this in an array of 26 elements for each letter of the alphabet. Since character "a" had a value of 97, we just subtracted each character's value to fit into the correct slot within the array. For example, "b" would be indexed at 1 because its ascii value is 98. 98 subtracted by 97 is 1. We did this for every single first and last value the keys. With this, we calculated the value of each word based on first and last letter value. We were able to sort them in descending order by having our Key class extend comparable and our compareTo method comparing the key's total value. Then, with the final order of the keys done, we were able to run Cichelli's method to properly insert into the hash table.

Difficulties:

Our biggest difficulty was figuring out how to sort the keys. That was solved and explained in our development process. Another decision that we had to make was where to place the methods for creating the order of keys within the hashtable. We were not sure whether to place them within the driver or within the hashtable. This was in parts due to not being sure at first how we wanted our HashTable class to be written. It was either only going to take the already sorted list of keys, or it would take care of everything the moment the HashTable was created with the unsort list of keys. We solved this by putting all the methods necessary within the HashTable class. One of the reasons we made this decision was because we realized that we need objects such as the g two-dimensional array within the HashTable class to be able to find the frequencies of keys in the text file. Also, if we put everything within driver class and passed it into the constructor of the HashTable class, we ran the risk of data misplacement. HashTable would have to carry parameters of the g-values and occurrences of letters. We could not find a way to keep them when creating the hashtable.