Matthew Parke

Description:

The way my algorithm works is by taking a word that is read in from the file and hashing it. In the hash function a hash index and a unique ID# are calculated, then those numbers and the original word are made into an AnagramClass object and put in the hash table. The hash index represents the index that class will be placed at in the hash table. The Unique ID# represents an anagram class, moreover, every word in an anagram class should have the same unique ID#. This is important because multiple different anagram classes are going to be hashed to the same index, which is handled with chaining. When a word is hashed to an index that already has anagram classes the algorithm will look for the class with the matching unique ID and place the word in that class. At the end of runtime all the anagram classes in the hash table should have all the words that belong to that class in a list stored in the respective class.

Correctness:

First let me start by saying there is an array of the first 26 prime numbers, each representing a letter in the alphabet, that is used by my hash function. My algorithm is correct in its sorting into the hash table because an anagram class's hash index is calculated by multiplying all the letter's prime numbers together. Ex) aba = 12 because a's prime is 2 and b's prime is 3(first and second prime for the first and second letters of the alphabet, respectively) 2*3*2 =12. However, since this number has to fit into a hash table of size 10000, whenever the hash index is larger than 10000 I mod by 10000. Modding takes away from the uniqueness of the prime multiplication, causing collisions at hash indices. That is why I created the unique ID#; when an index has a collision we can check the unique ID# to find the correct class for the hashed word. The unique ID# is calculated by adding all the letters' ascii values to that letter's prime number. Ex) aba = 299 = (97+2)+(98+3)+(97+2). This leaves very little room for error when placing a word into the correct anagram class because even if there is a collision then there would also have to be an class at that index with the same uniqueID#, but the ID# would represent a different class.

Runtime:

Below is very general pseudocode for my algorithm.

n = # of words   k = # of letters in a word

```
Algorithm()
    while(file still has words to read)     -------------- Θ(n)
        hash a word                         -------------- Θ(2k)
        place hashed word                   -------------- Θ(0.0001n) on average
                                            -------------- Θ(n) on worst case
        print out all classes               -------------- Θ(n)
```

Total average runtime = $\Theta(n(2k + 0.0001n) + n) = \Theta(0.0001n^2 + 2kn + n) = \Theta(n^2 + kn + n)$

Worst case runtime = $\Theta(n(2k + n) + n) = \Theta(n^2 + 2kn + n) = \Theta(n^2 + kn + n)$

**Worst case only happens when all words hash to the same index and are all different anagram classes, causing the linked list at that index to be n in length.