CS 2302 Data Structures

Fall 2019

Lab Report #1

Due: September 10, 2019

Professor: Olac Fuentes

TA: Anindita Nath

**Introduction**

In this lab the task was to find the anagrams of any word given to our program and have it pull those anagrams from all valid anagrams from the english language. The method prefered to find these anagrams was recursion was required to make it an intrical part of our program. Recursion was the main focus of this lab as a refresher on what we had learned in the previous class as well as get our “feet wet” in the possible optimizations to be made to our program..

**Proposed Solution Design and Implementation**

**Part #1:**

In part one I needed to find the anagrams of any word given. I managed to achieve this without catastrophic failure. Using our textbook as a guide I made a method that would find every possible permutation of Any Given word using recursion. After this was completed, using pythons internal syntax I was able to compare the permutations I had just found with the text file containing all English words.In doing this I was able to find all of valid anagrams of the given word.

**Part #2:**

Part 2 provided a few more challenges. part one was concerned with just getting the program to work but in part 2 I was required to optimize the program. we were given two methods of optimization to implement. the first optimization was to check for duplicate letters as the recursion was taking place. And if there was duplicate letters we were to immediately stop the recursion which would save us some time.I made a second method in the program which Ren very similar to the part 1 method. the only thing that was added to meet the optimization requirements was adding a check right before the recursive call Witch wood scan through the Given word and determine if it was a duplicate letter.

**Part #2 - second bullet:**

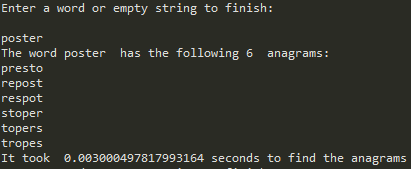
For the second part of part 2 the optimization was to actually compare the recursive call at any point to the prefix list of all the words in the English language. this was completed during the actual file reading of the English words which were then immediately converted into the prefix list then compared against anagrams found using methods very similar to part one and part two methods.

**Experimental Results**

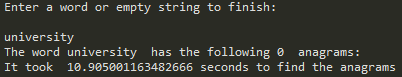
**Part #1:**

This part of the lab was meant to just make a working code even if it was slow so after finding the permutations of the given word and finding all valid anagrams i gave it some test words to ensure that the code works.

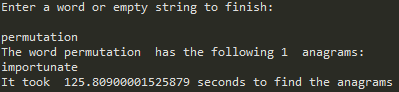
TEST 1 - Poster



TEST 2 - University



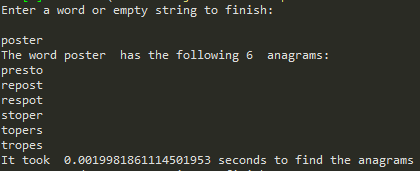
TEST 3 - Permutation



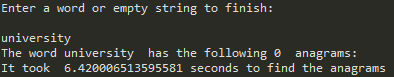
**Part #2:**

Now that the code has been verified to work i implement the first optimization and these are my test cases.

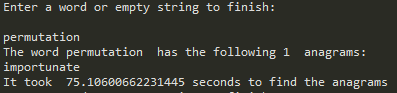
TEST 1 - Poster



TEST 2 - university



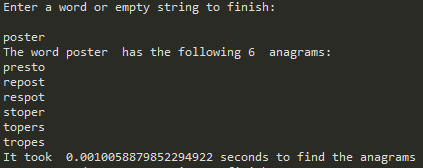
TEST 3 - permutation



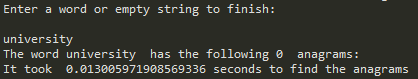
**Part #2 - second bullet:**

The second bullet designating how to optimize was ment to be the best optimization. These are the test case and their results.

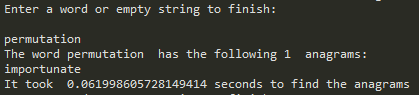
TEST 1 - Poster



TEST 2 - University



TEST 3 - Permutation



**Overall Lab Results:**

In this lab the optimizations was a success. While raw time may not always be the best form to represent optimization with do to that being a variable based on different hardware, the time differences between the different optimizations were large enough to confidently say that they worked. Below is a graph that summarizes this labs results.

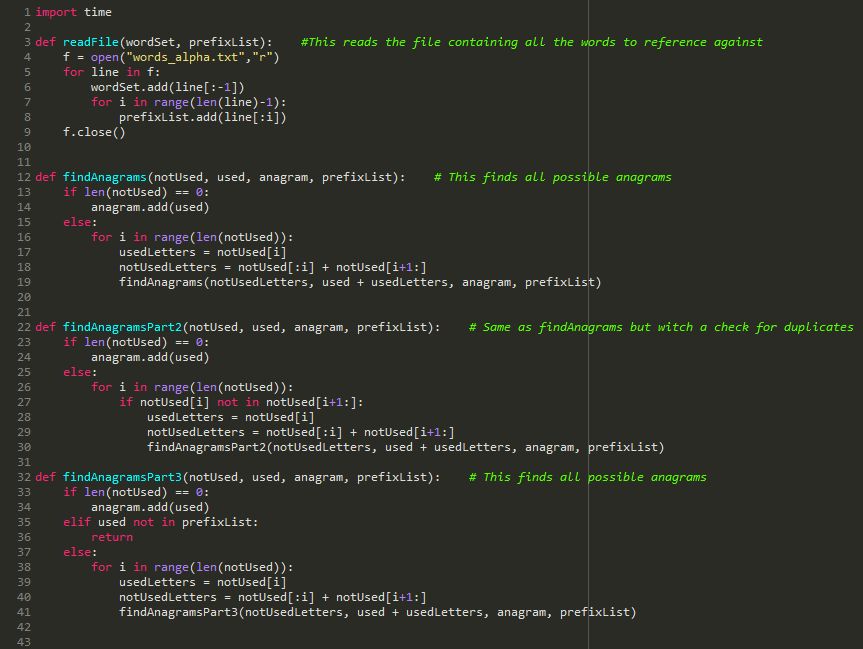
|  |  |  |  |
| --- | --- | --- | --- |
| Time in seconds | poster | university | permutation |
| No optimization - Part 1 | 0.003000497817993164 | 10.905001163482666 | 125.80900001525879 |
| Duplicate letters optimization - Part 2 | 0.0019981861114501953 | 6.420006513595581 | 75.10600662231445 |
| Prefix set optimization - part 2(Second Bullet) | 0.0010058879852294922 | 0.013005971908569336 | 0.061998605728149414 |

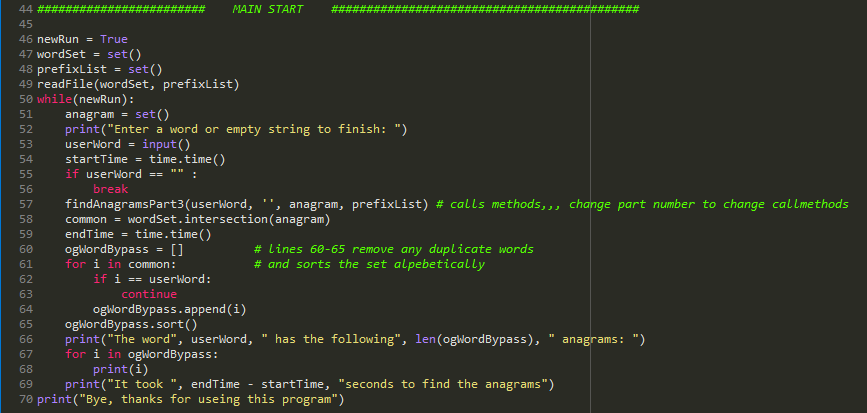
While the poster remained a short run time it did reduce the time it took to run as the optimizations were added. University had a 10 second decrease in time reducing it to a near instant runtime. Permutation saw the biggest time difference with a 125 second decrease also turning into a near instant runtime.

**Conclusion**

This was a nice refresher and prelude to the rest of the course. There was a considerable amount of recursion that was required which did cause me trouble. Overall i was able to take away from this lab a better understanding of recursion as well as making my code look presentable.

**Appendix**

****

****

*I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class*