**CS 2302 Data Structures**

**Lab Report #1**

**Recursion**

Carlos Cardenas

June 21, 2019

**Introduction**

Recursion is when a method “calls itself” to execute over and over again until a base case is reached. Recursion can be identified when a base case is written. A base case is made so the method knows when to stop. Recursion is useful to simplify bigger problems that require the use of for loops or similar algorithms.

The purpose of this lab is to successfully execute a recursion algorithm to find all the valid permutations of a given word. Recursion will simplify the problem by generating a string with all possible combinations of the word. The program also needs to compare the given permutations to a set containing over 466000 words, so that it returns valid words and not just all possible permutations. The program should also display the time it took to execute.

**Implementation**

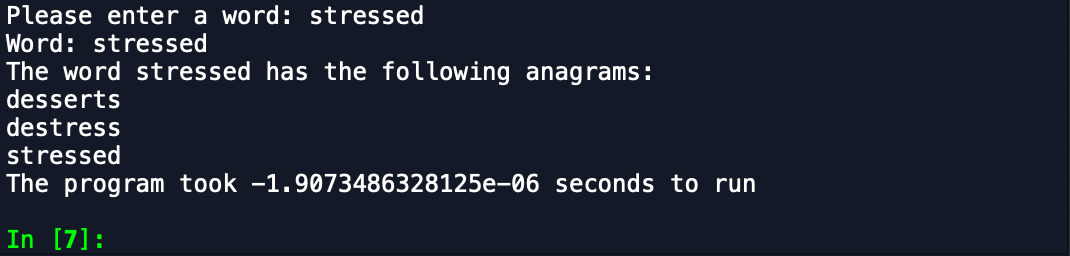
I solved this problem by first reading the text file containing the valid words. This was done by using the rstrip() function that will remove all spaces in between the words and arrange them side by side in a set. The next step is to ask the user for the word they want to look up for anagrams. This was easily done by using the input function. Then the recursive method was implemented. The base case was implemented to be an empty string, so if there is no word, the program will end automatically. An extra base case was made so if the user inputs a single letter, it would only return that single letter. For the permutations, I first created an empty string to store all permutations as they were being made. Then, a for loop determined by the length of the word that obtained one of the characters of the word and added the rest of the characters. After this, another for loop that appended every output of both variables using a recursive call to obtain every possible permutation. After this, the function returns the string containing the permutations. The hardest part about this approach was to declare a function using only the input word as a parameter, but this was done by creating the for loop with the recursive call to append the outputs.

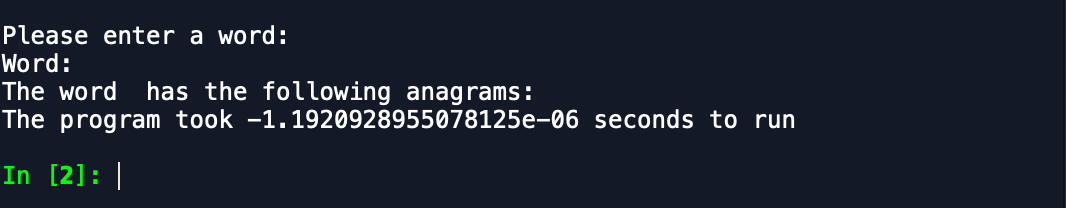
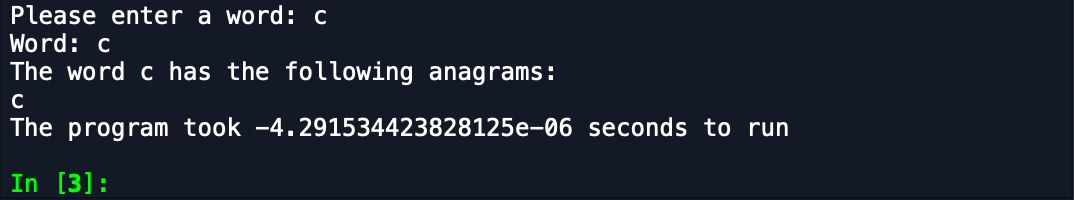
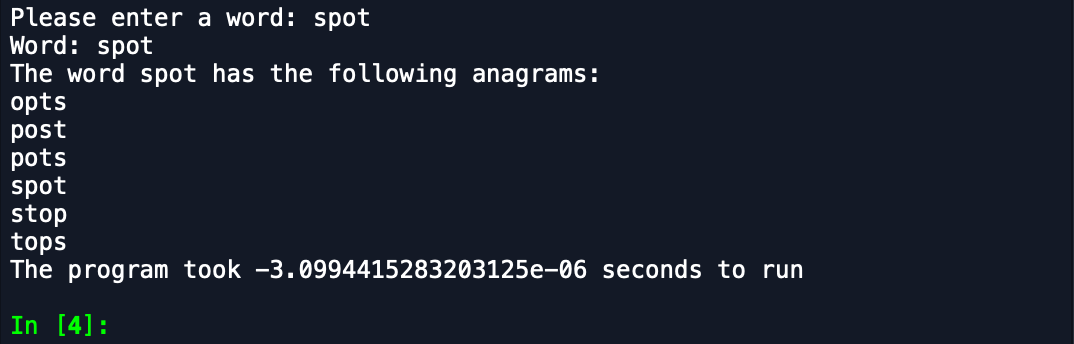
After that, I used the function intersection() that compares two sets and returns the words that matched in both sets. This is how I made the program to obtain only valid words and not “made up” words resulting from the permutations. After this, the sort() function was used to sort the matches in alphabetical order. Then a simple print statement to print out all anagrams in a separate line. Finally, to count the runtime of the program, the time library was imported to use the time() function that serves as a stopwatch to calculate the program’s time to execute.

**Experimental Results**

To test if my code works, I first started by inputting the base cases to see if they worked correctly. The empty string input terminated the program, while the single letter input returned just the same letter. To further test my program, I tried the example given to us in the instructions: the word spot. Sure enough, my output was identical to the example. I followed more examples by increasing the number of letters in each word. However, when an eleven or more-letter word is entered, the program takes an immense amount of time to compute the anagrams. This is because the amount of comparisons it has to make grows in a factorial manner. I tried to see the runtime of my program from one of these words but unfortunately my computer started to overheat, and I had to stop the program from running to avoid damage. After testing all of these cases, I noticed that the amount of time does increase with a bigger word, however the difference is not very significant until I enter an eleven-letter word.

Below are the outputs of my test cases:

A screenshot of a cell phone

Description automatically generated

**Conclusion**

After being able to successfully solve the problem, and as a beginner Python programmer, the first thing that I learned is that Python is an easier and simpler program to use than Java. I also really learned how to implement base cases and how to identify them. Recursion is a useful tool in a programmer’s code that simplifies programs. I am now more confident in my recursion methods, and though I’m still not a “master” in recursion, I definitely learned a lot from this lab.

**Appendix**

**A screenshot of a cell phone

Description automatically generated**

“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.”