Jesse Narkmanee

805190133

CS 32 Project 3 Report

**Obstacles and Difficulties**

My project overall consisted of three assigned functions that was to find all possible anagrams of a target word from a dictionary text document. While the first and third assigned functions were largely trivial and not too hard to implement, the second function to me was extremely difficult as it was not easy for me to satisfy the requirements listed in the specs that asked for specific return values while not surpassing global constants only using recursion.

For example, while I was able to follow the Spec FAQ pretty well in developing a recursive function that creates all possible string permutations, I then had to figure out how to return how many elements were in the result through recursion, copy it into result array, and also make sure to not include duplicate permutations in the result array.

To achieve this task I had to additionally create an insertResult() function that inserts anagram matches of the dictionary to the result array while also returning the count or element size of that same array and a Boolean function isRepeat() that evaluates whether or not a string permutation has repeated and already been inserted in the results[] array. Much of the difficulty in figuring out the algorithm was trying to think which way (head or tail recursion) to satisfy these requirements and the order I had to implement these functions. Overall, from my tests as shown below to work with the spec requirements.

**Test Cases**

Test Cases I ran were words similar to the word “rat” and “regardless” on the g32 linux servers with an 8mb limit. While regardless did take a long time, they all passed and matched the user prompt as shown in the spec which was my initial goal. Next, I wanted to test mainly five things and combine them together to see how they react with each other and get varied test cases. A very small word (like a), a medium sized word (4-5 characters), and a long word (9 characters), decreasing the global constants to check if my results and dictionary arrays would change from their original constants and still compile (to satisfy the requirements), then change the list size to something smaller, and choose words in different alphabetical order like starting with a, m, and z which represent first, middle, and last in alphabetical order.

Some examples of my test cases included evolution which would be considered a nine letter very long word that is located around the early middle of the list. Another test case I ran was the very first word which is 10th and the very last word which is zygote, which all ran successful at this point. After that, I looked on Google for popular anagrams and searched flesh which brought back flesh and shelf. Then I inserted three anagrams (auctioned, cautioned, education) into the words.txt file in places they should usually be. I chose these words since they are nine letter words and three are anagrams. Lastly, I went back to some of my previous tests and changed global constants to lesser values to see if my dict[] array and results[] array would also be accordingly truncated. For example, for rat I truncated MAXRESULTS to 2 which meant it only showed two results, and then created a cerr to print the dict[] array after I changed the max size to something as small as 20 words. Overall, all these array elements would be printed to their respective global constant size, deeming all these tests in being a success.