27096661_A1

Task1:

First read the file into python and store it as list. Because we want the largest group of anagrams so it is a good idea to put anagrams together. Therefore, we need to find one way to group anagrams by one property and this property should be the same for same group of anagrams and different from others. Therefore, consider the prime numbers, because prime numbers have no common divisor except 1. If we assign prime numbers for every letter in a word and multiply them. Then the result must be only according what letters they have and this is the definition of anagram. Therefore, we use the result as a signature to identify anagrams. Now, we can identify which words are anagrams to each other and we want to group them. There are many ways to get the largest group but for the use of task2 we need to sort them and because the signatures are numbers now it should be easy using radix sort. Because radix sort must be used with same digits, we add 0 before numbers to make them have equal digits. Then we sort the list and anagrams should be altogether. To get the largest group we initialize the max group to 0 and go through the list. Always compare one element with the one before it so we can know if this element is in a new anagram group or it is still in the same. Every time we reach a new word, we check if current anagram group is larger than the max group if so we change the max group to the current one. Finishing this we should get the start index of the max group and how many anagrams in it. Easily, we have the largest group.

Time complexity: in the implementation, we use the result of multiplying prime numbers instead of strings so first we need to prove they are no difference. (*Proof*: suppose all M letters are "z", then this must give us a largest result after calculation. Then the result should be 101^M. then because it is base 10, the digits of this number is log (10) (101^M) = M log (10) (101) which is a constant times M and in the big O notation, this will not make difference Q.E.D.) We go through every letter in every word, which need O(MN) time. Then we find the max length of the signature which need O(M+N) < O(MN) time. Equal digits and radix sort the list need O(MN) time because we both go through all the characters. Then we change signature back into integer need O(MN) time as well for the same reason. Find largest group need O(MN) time because we go through the list and compare integers every time. Then we ignore signatures and return words as output need O(N) time. Overall, the largest time complexity is O(MN) and there is no loop for one function call another. Therefore, the time complexity of task 1 is O(MN)

Space complexity: the implementation read the file into python and store as a list which is O(MN) space. For the working after this, we only manipulate on this list and multiply the space by constant numbers. No loop with variables are used, therefore the space complexity of task 1 is also O(MN).

Task2:

The question of task2 need us to search in the list. Because the list from task1 is sorted, we use binary search. First, we calculate the signature for the input query using same method from task1. Then the searching will give us an index or not found. However, we cannot know if the index returned is the start index of the anagram group. Therefore, we go forward toward the start to get the start of the group. Then we go backward to the end to generate all element in the group.

Time complexity: first we calculate the signature of the query which need O(k) time. Then we do a binary search which need O (log N) time if we treat comparing integers need O (1) time. However, if we move the calculation into binary search and treat comparing integer as linear time. We still get O (k log (N)) time. Then we go forwards and backwards in the list the get anagram group. Because we already know that digit of number is same of number of characters and we at most go through output list twice. Therefore, the time needed is O(W). Overall, because we don't know O(W) and O (k log (N)) which one is bigger, the time complexity is O (k log (N) + W).

Space complexity: Because we use the list from task1 instead of creating a new one. The only list we create is used to store output anagrams. Therefore, the space complexity is just O(W).

Task3:

Task3 is basically the same to task2, instead of using query string, we loop for 26 times to add 26 letters to the query string and do the same job as task2. Then we just print the total output.

Time complexity: just repeat task2 for 26 times so time complexity is O (K log (N) + W) as well.

Space complexity: the only list we create is to store output which is O(W) where W may be bigger than the W in task2.