Report of First Week Assignment of Step Program

Xiaoxue Zang

06/02/2016

I have tried 3 ways to search for the longest anagrams.
Except for the first c code, I sort words first and I think the sort function is implemented by quicksort or mergesort.

1　Anagram1.c

In this code, every alphabet is assigned to a prime number and the product of all the letters in a given word represents the value of that word.
Example:
a=2, b =3,g = 5 -> bag =30
I calculated the value of each word in the dictionary first and store them in an array, which I could look up later. I do not need to sort the word in this method.
Secondly, I calculate the value of the input word (set this value as A) in the same way. I search for the whole dictionary to find all its devisors and select the longest word through comparison.
I think that searching for an input word cost constant time and it equals to
O(n) (n is the size of the dictionary). The comparison takes O(2**m)(m is the length of the word) at worst case, which occurs with little possibility, because the comparison occurs only between words that actually exist in the dictionary.
Thus, I roughly estimate the searching time in this algorithms as O(n).
This method is quick and easy to implement especially when the length of the word is bigger than 17 because 2**18 > the size of the dictionary.
However, this algorithm has its own problem that when the length of the given word is long, the calculation miss would occur, for the error of calculation will accumulate and false answer is given when the value of a word is too large.
Example:
Good Input: Moon Stareraaa -> astronomer [correct answer]
However,
Bad Input: Moon Starerzz -> abama mamba mamma [wrong answer]
(for prime number for z is 101 and it is too big)

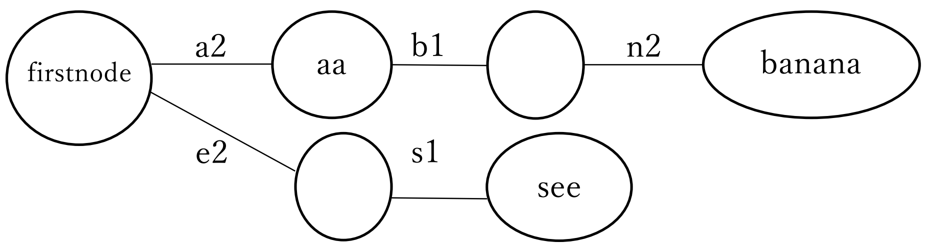
The value calculated for Moon Starerzz deviate from its correct value.
Although I could adjust by making the condition of deciding whether one integer is devisable by another not so restrict, I still think this problem is tricky to handle. Thus, I think this makes this method not a good one.

2 Anagram.py

I think the quickest way to find the word is using hash table. If it is not wrong, Python dictionaries are implemented using hash tables. It means that searching for each word cost O(1) time, however the memory cost could be high.
I list all the possible combinations of the letters of the given word and check whether such combination exists in the dictionary from the longest to the shortest combination, I would get the longest anagram. However, to get all the possible combinations of the letters from the given word is an NP problem.
The combinations of the sorted word are at worst 2**m (m is the length of the word), for some letters may appear more than once in that word.
So the running time is theoretically O(2**m). It is quick when the length of the input word is small.

3 anagramTree.cpp

This program is to build a dictionary tree, that word could be found by the letters appeared in the word and letters' frequency. It works as follows. Searching for each word would cost O(the number of letters that appear in the word). Though building the tree from dictionary is costly in time, it is quite quick even when the length of input is quite long. I think this method is the best of the 3.
Example:
Input: bromochloromethane -> chronothermal, chloromethane, monochromator
Indian lettuce -> unlicentiated


firstnode
a2
aa
b1
n2
banana
e2
s1
see