Project 2: Recursion on Tree

Problem Statement:

1 Motivation:

Here is an interesting problem. Suppose we have a lot of words, and want to find all words that share the same prefix, what's a good way to do it?

If you aren't familiar with this problem, consider yourself lucky. Take a break now, go for a walk and think about it before resuming.

One way is to create a hash-table (a dictionary, which we'll look at next quarter) which maps every possible prefix into a set of strings that can follow it. But that would be awfully wasteful, yes?

The **trie** presents an elegant way. In a **trie**, the data you want to store is not stored explicitly under a data label, but is instead implicit in the presence or absence of a particular child.

You can think of the **trie** as an abstraction over a general tree (the koala quest) with a fixed number of children. In this spec, I'm using the vector of children technique to represent this particular kind of general tree.

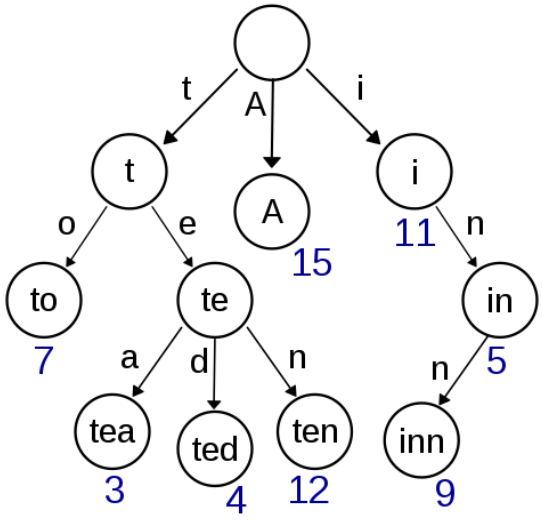

Figure 1: An example **trie**

Reference Materials: https://en.wikipedia.org/wiki/Trie

https://www.geeksforgeeks.org/trie-insert-and-search/

2. Problem Statement:

Let's assume that we want to count the occurrence of each word from a text file "declare.txt". In this file, it has the content of the article, ***Declaration of Independence***, which was written in 1776. We want to count each word in this file has how many occurrences.

Instead of using hashtable, hashmap, or dictionary, we would like to use this tree data structure, **trie**.

The initial trie will have

[0, []] # which means no occurrence **0**, and no sub-trie **[]** on the root node.

No occurrence, no sub-trie. There is a trie node of no occurrence and no sub-trie.

Then, if a word 'a' is added, the trie will become

[0, [**[1, []]**]]

Then, a sub-trie of [1, []] is added. This means 'a' letter occurs once.

Then, if another word 'ab' is added.

The trie will become

[0, **[** [**1**, [**None**, **[1, []]**]] **]]**

The black 1, means 'a' occurs once.

The red 1, means 'ab' occurs once.

The None, means 'aa' never been added to the **trie**.

In summary,

Each trie node is a list with 2 elements

number_of_occurrence: int, which denotes number of occurrence times for the string which terminated at current node.

list_for_sub-tries: list of sub-tries. If this list is an empty list. Then, there is no sub-trie.

If it is not an empty list, then, it will have many sub-tries up to the character which has sub-trie.

Say, 'aa' has no sub-trie, then a None will be store.

'ab' has a sub-trie, then, we store [1, []]

There is no 'ac' or any string with leading 'a' and trailing letter after 'b'. There is no need to put more sub-trie into it. Using the length of the list, we know how many letters are on this sub-trie list.

To write this program,

(1) Use class for Object-Oriented Programming.
(2) Numbers, symbols, and punctuation marks are not counted.
(3) **apple** and **apples** are treated as different words.
(4) it's and its considered one same word. (our purpose is to exercise on recursion and tree, not to development tokenization algorithm in this project).
(5) we're and we are considered different words.

(6) Only consider lowercase, all text should be converted to lowercase first. Therefore, a trie node may have a minimum of 0 sub-trie node and a maximum of 26 sub-trie nodes
(7) Read in the "declare.txt". **Click this link to download**.
(8) Enter each word into the trie. If a word has never been added, then, create a new path for it. If the word has been added to the trie, then increate the trie's corresponding occurrence number by 1.
(9) Write a __str__ function for the trie can be converted into a string with
word – 10
anotherword – 20
(10) Print out your listing for the number of occurrence for each word in the trie.

Have fun!!!

Example Program:

Starter Program:

```
class Trie{
    private:
        trienode *root;
    public:
        Trie();
        int size();
        bool is_empty();
        void add(string word);
        int get(string word);
        void clear(string word);
        string to_string();
};
```

Expected Results:

Click here for sample result (too long for listing).

Grading Rubric:

Components	Points Possible	Points Earned
Comments include name, date, purpose of the program	20	
Correctly implement the constructor for the **trie** class	20	
Correctly implements add("word") function	40	
Correctly implements get("word") function	40	
Correctly implements size() function	20	
Correctly is_empty() function	20	
Correctly clear() function	20	

Correctly __str__() or __repr__() function	40	
Fully documented report	20	
Successfully print out all words in the article with the right number of occurrences	60	
Total (Extra-points may be given to students)	300	