CS 2302 - Data Structures
Fall 2018
Project 3 - Option B

Overview

An anagram is a permutation of the letters of a word to produce another word. The method print_anagrams shown below (adapted from Carl Burch's book Programming via Java) prints all the anagrams of a given word. To do this, it generates all permutations of the letters in the word and, for each permutation, it checks if it is a valid word in the English language. For example, if you call print_anagrams("spot"), the method should print the following words: spot, stop, post, pots, opts, and tops

```
def print_anagrams(word, prefix=""):
  if len(word) <= 1:
     str = prefix + word

     if str in engish_words:
        print(prefix + word)
  else:
     for i in range(len(word)):
        cur = word[i: i + 1]
        before = word[0: i] # letters before cur
        after = word[i + 1:] # letters after cur

        if cur not in before: # Check if permutations of cur have not been generated.
           print_anagrams(before + after, prefix + cur)
```

The method uses a data structure called engish_words to determine if a given anagram is a valid word in the English language. You can think of engish_words as a container of all the words in the English language. We will implement this data structure using a binary search tree. To populate engish_words, we will use a text file called words.txt that contains 354,984 English words. You can download words.txt from the following URL: https://github.com/dwyl/english-words/. Once you have downloaded words.txt, write a function that reads the file and populates the binary search tree with all the English words contained in the file. Ask the user what type of binary search tree he/she wants to use (AVL Tree or Red-Black Tree). You are free to use the implementation provided in your zyBook for these two types of trees. Adapt zyBook's code to include the word and make it the key.

Write another function called count_anagrams that does not produce output, but returns the number of anagrams that a given word has. For example, count_anagrams("spot") should return 6. Finally, write another function that reads another file that contains words (feel free to create it yourself) and finds the word in the file that has the greatest number of anagrams.

What you need to do

Part 1 - Due Friday, November 2, 2018

Implement the program described above, and upload your code to GitHub.

Part 2 - Due Tuesday, November 6, 2018

Add your team members as collaborators to your GitHub repo. They will add you to their projects as a collaborator as well. Read their code and give them feedback. Use *pull requests* and/or the *Issues* section to do so .

Extra Credit

Re-do the lab using a B-Tree, run multiple experiments using different values for the degree of the tree, and include in your report how performance changes as the degree varies. Finally, use tables and graphs to compare B-Trees with AVL and Red-Black trees.

Rubric

Criteria	**Proficient**	**Neutral**	**Unsatisfactory**
Correctness	The code compiles, runs, and solves the problem.	The code compiles, runs, but does not solve the problem (partial implementation).	The code does not compile/run, or little progress was made.
Space and Time complexity	Appropriate for the problem.	Can be greatly improved.	Space and time complexity not analyzed
Problem Decomposition	Operations are broken down into loosely coupled, highly cohesive methods	Operations are broken down into methods, but they are not loosely coupled/highly cohesive	Most of the logic is inside a couple of big methods
Style	Variables and methods have meaningful/appropriate names	Only a subset of the variables and methods have meaningful/appropriate names	Few or none of the variables and methods have meaningful/appropriate names

Robustness	Program handles erroneous or unexpected input gracefully	Program handles some erroneous or unexpected input gracefully	Program does not handle erroneous or unexpected input gracefully
Documentation	Non-obvious code segments are well documented	Some non-obvious code segments are documented	Few or none non-obvious segments are documented
Code Review	Useful feedback was provided to team members. Feedback received from team members was used to improve the code.	Feedback was provided to team members, but it was not very useful. Feedback received from team mates was partially used to improve the code	Little to no feedback was provided to team mates. Received feedback was not used to improve the code.
Report	Covers all required material in a concise and clear way with proper grammar and spelling.	Covers a subset of the required material in a concise and clear way with proper grammar and spelling.	Does not cover enough material and/or the material is not presented in a concise and clear way with proper grammar and spelling.