CS 2302 - Data Structures
Fall 2018
Project 4 - Option B

Overview

In the previous project (lab), you implemented an algorithm to find the anagrams of a word. For a vocabulary size v, it took O(v logv) to build a BST to store valid words in *english_words*. The purpose of this assignment is to improve the running time of the algorithm by using a hash table instead of a BST.

Your hash table must use chaining to solve collisions. For comparison purposes, you must write multiple hash functions. Also, write a method to determine the average number of comparisons required to perform a successful retrieve operation, and a method to compute the table's load factor. In the previous project, we used a BST, so on average, the retrieve operation needed to perform log (354,984) ≈ 18 comparisons. Your hash table (to be successful) needs to perform significantly fewer operation than that (in the ideal case, you'd need exactly 1 comparison per access).

Hint 1: The choice of table size and hashing function has a large impact in the number of collisions and thus running time.

Hint 2: There are 354,984 English words in words.txt

Hint 3: You can think of a word in the English language as a base-26 number.

What you need to do

Part 1 - Due Sunday, November 11, 2018

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

Part 2 - Due Wednesday, November 14, 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 .

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.