**CS 2302 - Data Structures**

**Fall 2019**

**Project 3 - Option A**

**Overview**

[Natural Language Processing (NLP)](https://en.wikipedia.org/wiki/Natural_language_processing) is the sub-field of artificial intelligence that deals with designing algorithms, programs, and systems that can understand human languages in written and spoken forms.

[Word embeddings](https://en.wikipedia.org/wiki/Word_embedding) are a recent advance in NLP that consists of representing words by vectors in such a way that if two words have similar meanings, their embeddings are also similar. See <https://nlp.stanford.edu/projects/glove/> for an overview of this interesting research.

In order to work in real-time, NLP systems such as Siri and Alexa need to efficiently retrieve the embeddings given their corresponding words. In this lab, you will implement a simple version of this. The web page mentioned above contains links to files that contain word embeddings of various lengths for various vocabulary sizes. Use the file ”glove.6B.50d.txt” which contains word embeddings of length 50 for a very large number of words. Each line in the file starts with the word being described, followed by 50 floating point numbers that represent the word’s vector description (the embedding). The words are ordered by frequency of usage, so ”the” is the first word.

Your task for this lab is to write a program that does the following:

1. Read the file *glove.6B.50d.txt* and store each word and its embedding in a binary search tree. 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 its embedding and use the word as the key. Ignore the ”words” in the file that do not start with an alphabetic character (for example ”,” and ”.”).
2. Read another file containing pairs of words (two words per line) and for every pair of words find and display the ”similarity” of the words (see example in appendix). To find the similarity of words and , with embeddings and , we use the cosine distance, which ranges from -1 to 1, given by:

Where is the dot product of and and and are the magnitudes of and . Look-up these formulas online if necessary.

1. Write the following methods to extract information from the tree.

(a) Compute the number of nodes in the tree.

(b) Compute the height of the tree.

(c) Generate a file containing all the words stored in the tree, in ascending order, one per line.

(d) Given a desired depth, generate a file with all the keys that have that depth, in ascending order.

Recall that the root has depth zero, its children have depth one, and so on.

As usual, write a report describing your work. Determine the O () running times of your methods and show tables illustrating their actual running times and discuss disagreements between theoretical and experimental results.

**Appendix**

*Word similarities:*

barley shrimp 0.5352693296408768

barley oat 0.6695943991361065

federer baseball 0.28697851474308855

federer tennis 0.7167608209373065

harvard stanford 0.8466463229405593

harvard utep 0.06842559024883524

harvard ant -0.026703792920826475

raven crow 0.615012250504612

raven whale 0.32908915459219595

spain france 0.7909148906835685

spain mexico 0.7513763544646808

mexico france 0.5477963415949284

**What you need to do**

**Part 1 - Due Tuesday, October 14, 2019**

Upload the progress you have made. You need to have at least 50% of the lab done by this date.

**Part 2 - Due Friday, October 18, 2019**

Final due date (everything finished - code).

**Part 3- Due Wednesday, October 23, 2019**

Final due date (report)

**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 |
| **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. |