Lab 4 BTrees

Introduction

The learning objective for this lab is to experiment the speed of inserting a large amount of data into a Btree data structure. Unlike binary trees, where each node contains one key and two pointers to the next data points. Btrees contain a list of keys and a list of nodes, each key is mapped to a node with the same exact setup, the amount of keys is up to the user. Along with inserting during this lab we'll be searching the Tree, we'll be testing what type of setup works for different situations.

Design

BTree

Our professor gave us a simple Btree implementation to work with, however, the data we're reading contains both a key along with corresponding data. So I created a wrapper class to hold the data, this has two attributes, a key and data. Since python is awesome we can use simple types at the key, and whatever we want as the data as this will not be processed by my new Btree implementation. My new Btree implementation only needed slight editing, this came when finding a key and sorting the keys as well. My Btree is simply a child of the original Btree implementation given.

File reading came

Part 1

I used the same old implementation as the last lab to read the data from the given file, the only difference was first creating the object wrapper type, inputting the word as the key and the embedding as the data. Take this type and input it into the Btree.

Part 2

The only change that came from this function was renaming the function call. Other than that there was no difference.

Part 3

A. I just added an extra attribute to count the amount of nodes as they were being inserted. This to me seemed like the simplest and most time efficient way to add data.
B. This function was given to us by the original implementation.
C. Used recursion and a for loop to get through every element in the tree.

D. Used the same implementation as the previous lab, except I used a for loop to access every child node.

Results

For these results I read the entire file. To compare both I also ran the entire file for both Red and Black and AVL binary trees.

Time Complexity

N = The entire file.

Amount of keys: 5

- Part 1: 21.767024993896484
- Part 2 (search): 0.002048015594482422
- Total time: 22.320160150527954

Full results below:

- Time to read the file: 21.767024993896484
- Time to search: 0.002048015594482422
- Time to find height: 1.4066696166992188e-05
- Time to go through entire tree: 0.5491948127746582
- Time go get to desired depth: 0.0018062591552734375
- Total Time for job to run: 22.320160150527954

Amount of keys: 50

- Part 1: 51.37858295440674
- Part 2 (Search): 0.0015001296997070312
- Total Time: 51.84529900550842

Full Results Below:

- Time to read the file: 51.37858295440674
- Time to search: 0.0015001296997070312
- Time to find height: 9.059906005859375e-06
- Time to go through entire tree: 0.45098376274108887
- Time go get to desired depth: 0.014132022857666016
- Total Time for job to run: 51.84529900550842

Amount of keys: 100

- Part 1: 119.53724598884583
- Part 2 Search: 0.0015530586242675781
- Total Time: 120.43460297584534

Full Results Below:

- Time to read the file: 119.53724598884583
- Time to search: 0.0015530586242675781
- Time to find height: 6.198883056640625e-06
- Time to go through entire tree: 0.46491098403930664
- Time go get to desired depth: 0.43079400062561035
- Total Time for job to run: 120.43460297584534

Amount of keys: 500

- Part 1: 2084.056919813156
- Part 2: 0.0034339427947998047
- Part 3: 2084.938626766205

Full Results Below:

- Time to read the file: 2084.056919813156
- Time to search: 0.0034339427947998047
- Time to find height: 5.9604644775390625e-06
- Time to go through entire tree: 0.4467909336090088
- Time go get to desired depth: 0.4313969612121582
- Total Time for job to run: 2084.938626766205

Conclusion

The fastest insert time was easily 5 keys per node. However, the fastest search time was between 100 and 50 keys per node, these suffer when it comes to inserting. As the amount of keys grows so does the amount of time it takes to insert, the amount of time to search increases though.