Assignment 3: Searching with Trees
ADVIK BAHADUR

Task 2

Choice of Unique Id

To obtain a balanced tree that allowed for good performance when searching with trees, there seemed to be two distinct paths to go:

1) Use of an algorithm that balances trees as more items are added. For example, the use of algorithms like Red Trees black trees, or AVL.
2) Implementation of a function that puts out nodes in an order that results in a more balanced tree.

Given the context of this assignment, it seemed clear that the latter option is what is required.

I decided to make use of random numbers, to ensure a more balanced tree. I decided on this since it was easy to implement and shouldn't take much computational effort.

When I did implement the random number key Unique ID generator, while I was able to get a well balanced tree, I would get multiple nodes with the same Unique ID. An easier yet more computationally heavy way to handle this issue would be to generate random numbers for a much larger range of integers. For example, for having 100000 nodes in the database, to have 10 to 100 times for the range the random numbers can be. So, we can allow the Unique ID to be between 0 to 10 million.

Yet problems with this include high run times, the probability of having the same unique id, is unlikely yet still possible. Hence this solution isn't the best.

I ended up using the **Fisher-Yates shuffle** to generate my random numbers. This method has 3 steps:

1) Generate an array of all numbers going from 0 to no of books. (0,1,2,3,4,5….)
2) Then using Fisher-Yates shuffle to swap out each integer with an integer at a random index of the array
3) Finally, when we have an array of integers, with no duplicates and in randomised order, we can now call each integer one by one from the array, in order.

This method eliminated any errors that were coming up, where the Unique Id for the node were the same.

Extra testing

To ensure the correct workings of my system I decided to print out the following things:

1) The overall time taken for the entire system to work. From initiation to insertion to searching.
   a. I did this, since I did a lot of computation in my initiation function *bstdb_init.* The timers in the profiling code didn't account for this and didn't time this computation. Hence I though it ll be a good indication of performance to return this time aswell.
2) Time Taken for initiation function:
3) Total leaves nodes pass through before insertion.
4) Average no of leaves passed through.

Printing the results of these allowed me to look at the numbers to gather if my tree actually was balanced or not.

Clearly by the outputs printed below, we can see how quick the algorithm is and how quickly it can perform all computations.

Student STDOUT.txt

```
Generating 104977 books... OK

Profiling listdb
-------------------------------------------

Total Inserts                :      104977
Num Insert Errors            :           0
Avg Insert Time              :   0.000003 s
Var Insert Time              :   0.000002 s
Total Insert Time            :   0.552822 s

Total Title Searches         :       10497
Num Title Search Errors      :           0
Avg Title Search Time        :   0.001168 s
Var Title Search Time        :   0.005984 s
Total Title Search Time      :  12.302623 s

Total Word Count Searches    :       10497
Num Word Count Search Errors :           0
Avg Word Count Search Time   :   0.001133 s
Var Word Count Search Time   :   0.005908 s
Total Word Count Search Time :  11.935442 s

STAT
Avg comparisons per search  -> 52391.386301
List size matches expected? -> Y

Profiling bstdb
-------------------------------------------

Total Inserts                :      104977
Num Insert Errors            :           0
Avg Insert Time              :   0.000005 s
Var Insert Time              :   0.000026 s
Total Insert Time            :   0.746808 s

Total Title Searches         :       10497
Num Title Search Errors      :           0
Avg Title Search Time        :   0.000006 s
Var Title Search Time        :   0.000004 s
Total Title Search Time      :   0.084426 s

Total Word Count Searches    :       10497
Num Word Count Search Errors :           0
Avg Word Count Search Time   :   0.000005 s
Var Word Count Search Time   :   0.000000 s
Total Word Count Search Time :   0.074988 s

STATS:
Total leaves nodes pass through before insertion: 2241407
Average no of leaves passed through: 21.000000
Time Taken for initiation function: 0.024008 seconds
Time Taken for total program to run: 0.927795 seconds
Press Enter to quit...

```