Assignment 2 report KPPDEA001 23 March 2022

**Goal:**

The goal of this experiment was to demonstrate the performance of AVL trees given different permutations of input data. The experiment displays the difference in number of operations taken to complete the search and insert algorithms using varied degrees of randomised data.

In order to execute the experiment, I have created a programme called AVLExperiment. The main class consists of four methods, a ReadFile method, a shuffle method, a load tree method and a HowRandom method.

The ReadFile method simply reads the data from the given CSV file into an array. The Shuffle method then n items in the array based on a given parameter. Once the array is shuffled to the specified degree, the data is loaded into an AVL tree using the LoadTree method and the find algorithm is performed on every item of data.

The operations are counted in the AVLTree class using integer variables which are incremented per operation. The results of the experiment are loaded into a separate array as a string. The string contains the number of operations taken to perform the search and find tasks as well as how random the data was that these tasks were performed on. This is given by the number of items that are out of order which is received from the HowRandom method.

**Randomisation:**

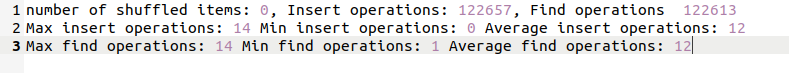
In order to randomise the data, I have created a method called shuffle. The method uses an algorithm which operates on the given number of items. The algorithm loops through the array n times and performs the following operations.

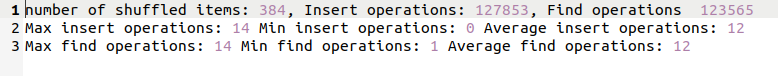
1. Generates a “random” number using the Math.random function which will be used as an index.
2. Stores the current index value in a temporary variable.
3. Copies the value at the index of the “random” number that was generated and stores it in the current index.
4. Stores the item of the temporary variable in the index of the “random” number.

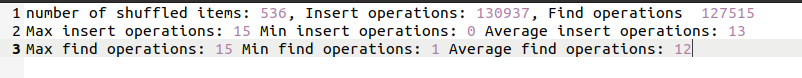
In summary, these steps take the current index and swap its value with a value in a random index in order to shuffle the items in the array.

This algorithm is good due to its simplicity. The code is only several lines, and it is easily understandable by anyone. The algorithm is a very efficient way to make sure most, if not all items in the array ends up in a different position.

**Results:**

**No randomisation (level 0)**

**Level 10 of randomisation**

**Level 20 of randomisation**

The results of the experiment demonstrate the efficiency of AVL trees. Due to the properties of an AVL tree of having the difference in height between sub trees less than or equal to one, the tree is constantly rebalanced and therefor will never become a linear data structure. Due to this fact it does not matter what order the items are inserted or searched for as the average and worst case time complexities will be O(log n), this is far more efficient then a regular binary search tree which has worst case time complexities of O(n). The graphs above show the small amount of change between number of operations for the insert algorithm with different permutations.

**Creativity:**

For this assignment I struggled to find a way in which to be creative however I have I feel as though I have performed the required tasks and I have used simple and easy to understand algorithms to do so and not over complicate the process. my experiment results or printed neatly to a text file for later review as well as to the terminal for instant viewing.

Table

Description automatically generated **Git log:**

Text

Description automatically generated