**LAB 5: HASH AND SEARCHING**

**Name: Trần Gia Bảo**

**ID: 22127034**

**EXERCISE 7.5: Refer back Lab 3 Sorting and search for detail description. In Lab 3, you do a simple spell checker with Linear Search and Binary search. In Lab 5, you continue that project by adding a new search – Search with Hash Table. Now in this Lab 7, you will do search with Binary Tree and AVL Tree**

**Compare the result from different methods. Analyze and discussion.**

|  |  |
| --- | --- |
| **Methods** | **Run time (ms)** |
| **Linear Search** | 3184 |
| **Binary Search** | 20 |
| **Chaining Approach** | 2 |
| **Linear Probing** | 2 |
| **Quadratic Probing** | 2 |
| **Double Hash** | 2 |
| **Binary Tree** | 32 |
| **AVL Tree** | 24 |

- From the experiment results, it appears that linear search and binary search are included in the comparison, which are not directly related to the collision resolution techniques. However, we will focus on analyzing the collision resolution techniques mentioned (Linear Probing, Chaining Approach, Quadratic Probing, and Double Hashing) based on the time taken for each method.

- Linear Probing (2 milliseconds): Linear probing is a basic open addressing technique that sequentially searches for the next available slot in case of a collision. The result of 2 milliseconds indicates that the performance of linear probing was reasonably efficient for the given dataset and load factor.

- Chaining Approach (2 milliseconds): The chaining approach, where collisions are resolved by creating linked lists at each bucket, also took 2 milliseconds. This result suggests that chaining was able to handle collisions efficiently for the dataset and provided similar performance to linear probing.

- Quadratic Probing (2 milliseconds): Quadratic probing, another open addressing technique, uses a quadratic function to find the next available slot after a collision. The result of 2 milliseconds indicates that quadratic probing performed well and achieved the same level of efficiency as linear probing and chaining in this particular scenario.

- Double Hashing (2 milliseconds): Double hashing uses two hash functions to calculate the step size for probing after a collision. The result of 2 milliseconds suggests that double hashing, with the chosen second hash function, was effective in handling collisions and achieved similar performance to the other probing techniques.

- Binary Tree (3910 milliseconds): The execution time of 3910 ms for the binary tree suggests that the tree might not be well-balanced, and the search operation is slower than expected. The performance of binary trees heavily depends on their structure, and if the tree is not balanced, it may lead to inefficient search times, approaching linear search in some cases. Balancing issues can arise if elements are inserted in a way that creates a skewed tree.

- AVL Tree (24 milliseconds): AVL trees are balanced binary search trees where the difference in height between the left and right subtrees of any node is at most 1. The execution time of 24 suggests that the AVL tree performs better than the basic binary tree in this case. The self-balancing property of AVL trees ensures efficient operations, such as insertion, deletion, and search, with a time complexity of O (log n).

**What is average depth, longest depth of nodes in these Binary and AVL Tree?**

Average Depth in Binary Tree: The average depth of nodes in a binary tree can be calculated as the sum of the depths of all nodes divided by the total number of nodes in the tree. The depth of a node is the number of edges from the root to that node.

Average Depth BIN Tree = (Sum of Depths of all Nodes) / (Total Number of Nodes)

Longest Depth in Binary Tree: The longest depth in a binary tree refers to the depth of the deepest node in the tree. It is the maximum number of edges from the root to any node in the tree.

Average Depth and Longest Depth in AVL Tree: In AVL trees, the balance factor of each node (the difference in height between the left and right subtrees) is kept balanced, typically within the range of -1 to 1. Due to this property, AVL trees maintain a height that is approximately log base 2 of the number of nodes in the tree, resulting in a more balanced structure.

The average depth and longest depth in an AVL tree depend on the number of nodes and the specific structure of the tree. However, in an ideal scenario with a balanced AVL tree, the average depth can be close to log base 2 of the number of nodes, and the longest depth can be log base 2 of the number of nodes.