CSCI 2270 - Data Structures and Algorithms
Final Project

Jason A. Popich, Matthew Januszewski
CSCI 2270, 04/24/2020

Contents

I. Introduction 3
II. Discussion 3
III. Results 4
IV. Conclusions 9

Abstract

This report examines the performance of insertion and search operations in relation to the number of elements stored in various data structures. Between a linked list, binary search tree, and various hash table implementations it was found that a chaining hash table achieved the greatest performance as a function of number of elements stored. Thus the Chaining Hash Table Data Structure would be the appropriate choice for USPS package tracking application.

I. Introduction

This project examines the performance of insert and search operations on various data structures as the number of elements being stored increases, with the goal of determining the data structure with the best insertion and search performance (lowest time per operation) given a random data set. The team was provided two data sets of random key values to be inserted into a linked list, binary search tree (BST), and several different implementations of a hash table. 100 values were inserted into the data structure at a time, then 100 random keys were searched for and these operations were timed. The resulting timing data was then graphed to show the relationship between performance and number of data points for each data structure.

II. Discussion

Methodology

Duplicate keys were handled by checking in the insert functions whether or not a key was already stored in the structure. If a matching duplicate key was found the insert function returned without adding an additional item to the data structure. This results in each key stored being unique and maximizes search performance.

Collisions in the hash table implementations were counted according to the following rules: If a given key's hash representation caused a collision this counted as a collision, as did all subsequent increments (either linear or quadratic) of the key's hash representation until an empty index was found. In the case of the chained hash implementation if the index of the hash table given by the key's hash representation already has a node stored in it this was considered a collision, as was each comparison to nodes stored at that index (i.e. if an index of the hash table had 6 nodes stored in it then this was counted as 6 separate collisions).

All trials were run on the same virtual machine to ensure consistent performance. Outlying data that was determined over several trials to be random interruptions from the virtual machine was removed using MATLAB.

III. Results

The following graphs show the time to perform search and insert operations on each data structure as the number of keys stored increases. For the three hash implementations insert and search collisions per 100 operations are also shown.

Datasets

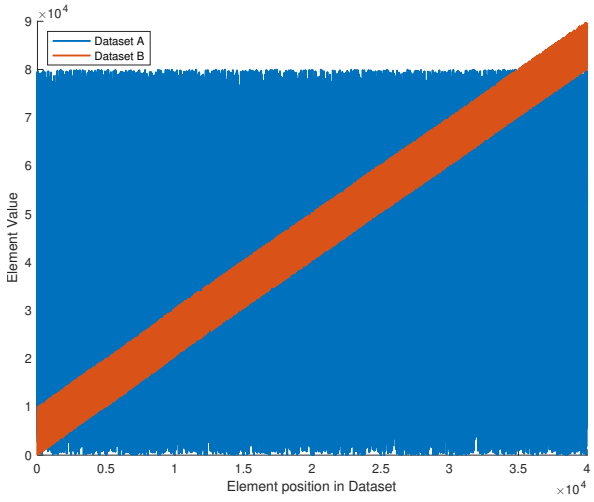

Figure 0.1. Datasets Plotted for info

The linked list results from both data sets A and B show $O(n)$ performance for both insert and search operations, which makes sense given that each insert operation performed on the linked list has to traverse n nodes and the average search operation has to traverse a number of nodes directly proportional to n.

Linked List

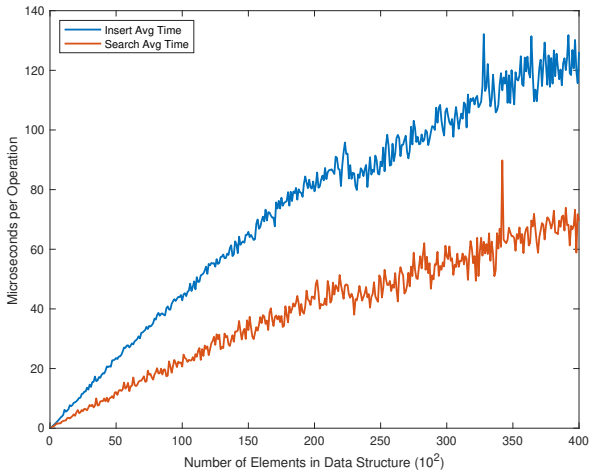

Figure 0.2. Linked List (Data set A)

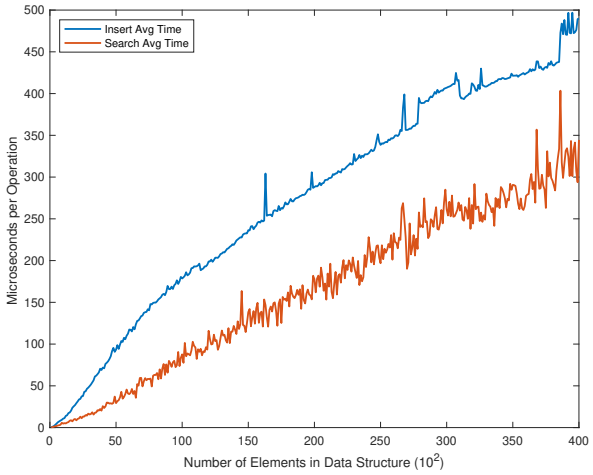

Figure 0.3. Linked List (Data set B)

The linked list results from both data sets A and B show $O(n)$ performance for both insert and search operations, which makes sense given that each insert operation performed on the linked list has to traverse n nodes and the average search operation has to traverse a number of nodes directly proportional to n.

BST

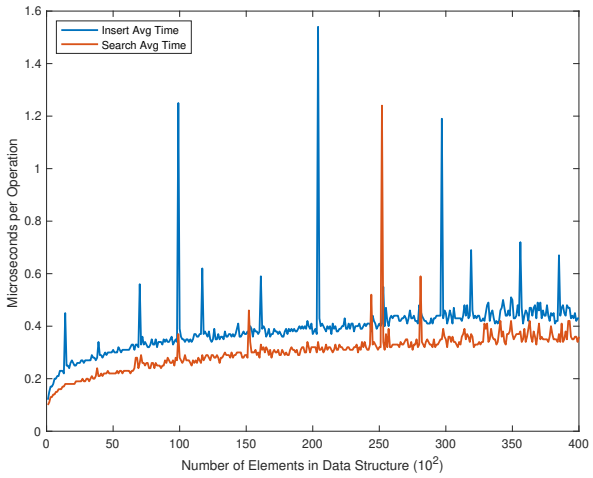

Figure 0.4. Binary Search Tree (Data set A)

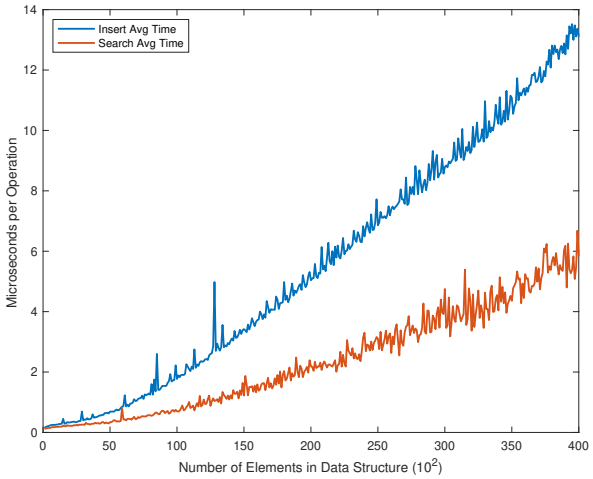

Figure 0.5. Binary Search Tree (Data set B)

The BST results from from data set A clearly show the $O(log(n))$ performance of both the insert and search operations of a well balanced BST. However, a BST without some invariant that ensures a balanced structure (such as a red-black tree) has a worst case of $O(n)$ performance for both insert and search if the data inserted into it is highly unbalanced. For example an ascending series of keys would result in the BST essentially acting like a linked list with $O(n)$ performance. This appears to be the case in data set B, which shows distinctly $O(n)$ behavior compared to data set A.

Chaining Hash Table

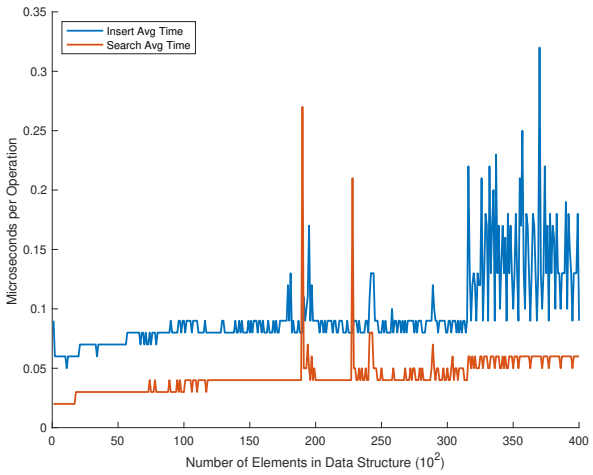

Figure 0.6. Chaining Hash Table Insert/Search (Dataset A)

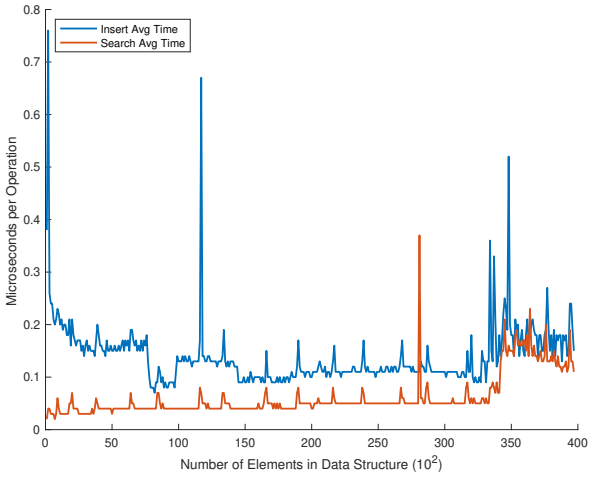

Figure 0.7. Chaining Hash Table Insert/Search (Dataset B)

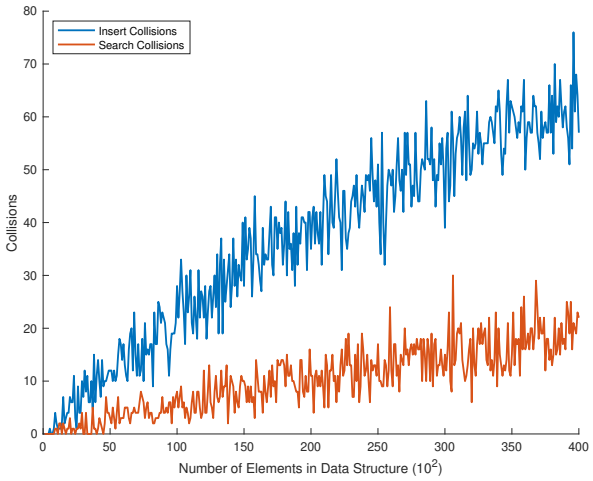

Figure 0.8. Chaining Hash Table Collisions (Dataset A)

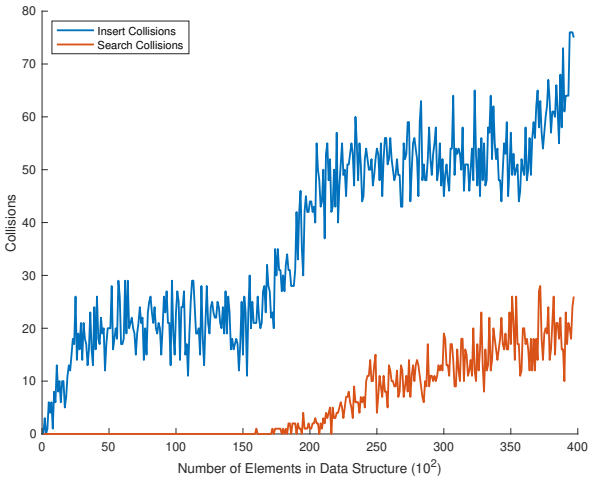

Figure 0.9. Chaining Hash Table Collisions (Dataset B)

The chaining hash table results show that the vast majority of search and insert operations performed for both A and B were completed in close to $O(1)$ time because the time required to insert and search for a key does not significantly increase as more keys are added. The number of collisions shows that some indices of the hash table were more frequently hit than others, however the number of nodes stored at any individual index of the hash table remained small and thus the time required to search the linked list stored at each index of the hash table grew slowly compared to n. The rapid increase in collisions for data set B between 15000 and 20000 keys indicates that the keys in this range had a high incidence of repeated hash representations.

Linear Probing Hash Table

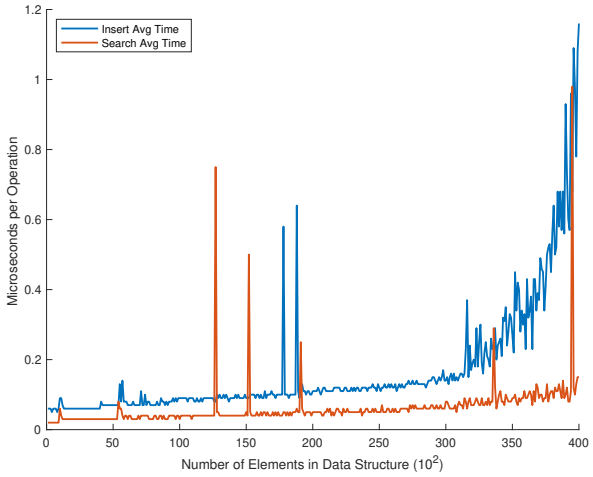

Figure 0.10. Linear Probing Hash Table Insert/Search (Dataset A)

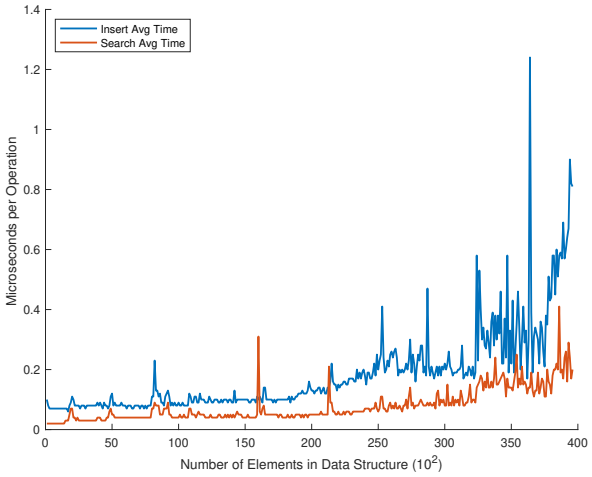

Figure 0.11. Linear Probing Hash Insert/Search (Dataset B)

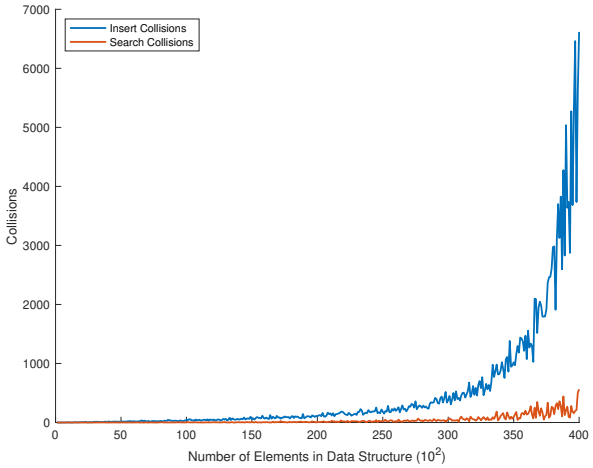

Figure 0.12. Linear Probing Hash Table Collisions (Dataset A)

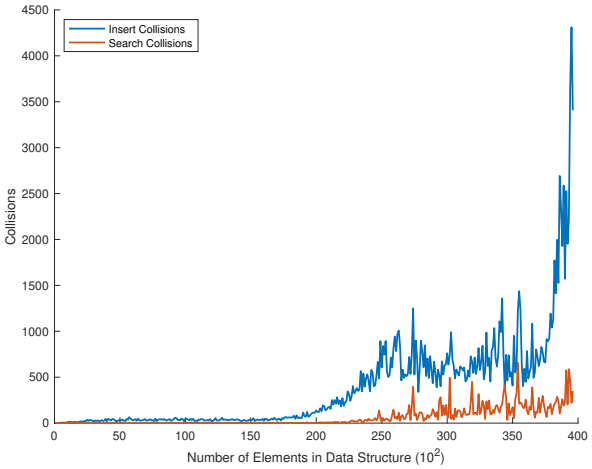

Figure 0.13. Linear Probing Hash Table Collisions (Dataset B)

The linear probing hash table results, particularly for data set A, show relatively constant performance until 30000 elements had been added, after which the time required to insert a key increased rapidly. The search time did not increase as rapidly due to the duplicate key policy explained in section II. The number of insert and search collisions increased in a very similar way compared to the insert and search times, which makes sense because the main factor in increasing search/insert time is the number of collisions encountered. The hash table only stored 40000 elements which meant that as it was filled more and more collisions occurred which is reflected in the graph.

Quadratic Probing Hash Table

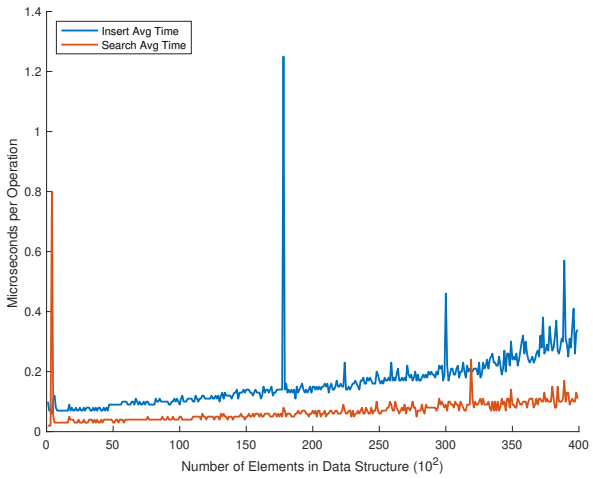

Figure 0.14. Quadratic Probing Hash Insert/Search (Dataset A)

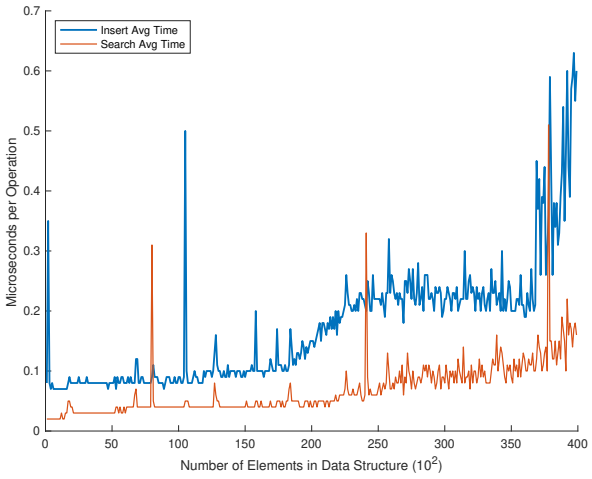

Figure 0.15. Quadratic Probing Hash Insert/Search (Dataset B)

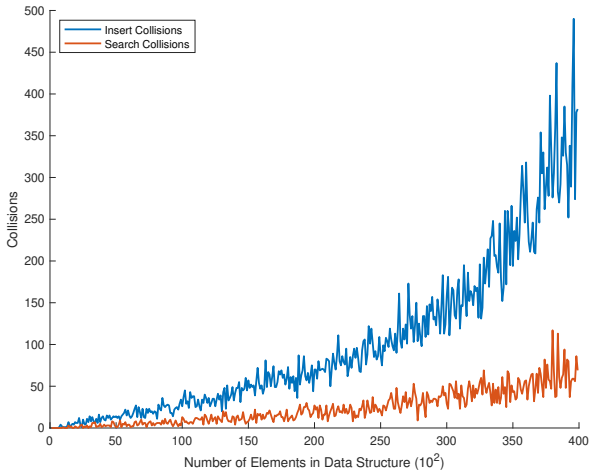

Figure 0.16. Quadratic Probing Hash Table Collisions (Dataset A)

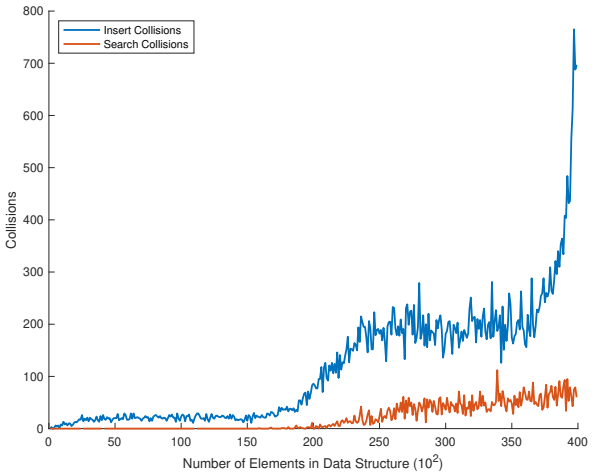

Figure 0.17. Quadratic Probing Hash Table Collisions (Dataset B)

The quadratic probing hash table suffered from a less severe increase in insert times past 30000 elements compared to the linear hash table, which corresponds to a smaller increase in search collisions compared to the linear hash table. This due to the quadratic form of hashing, however, given enough elements the Quadratic probing hash would run into the same performance issues as the Linear hash table.

IV. Conclusions

Summary Graphs

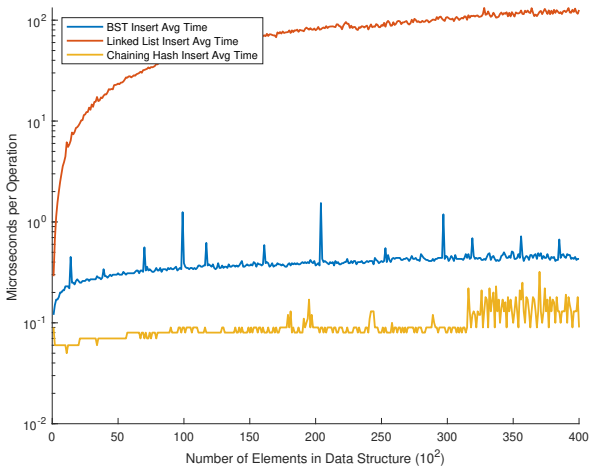

Figure 0.18. Insert Summary

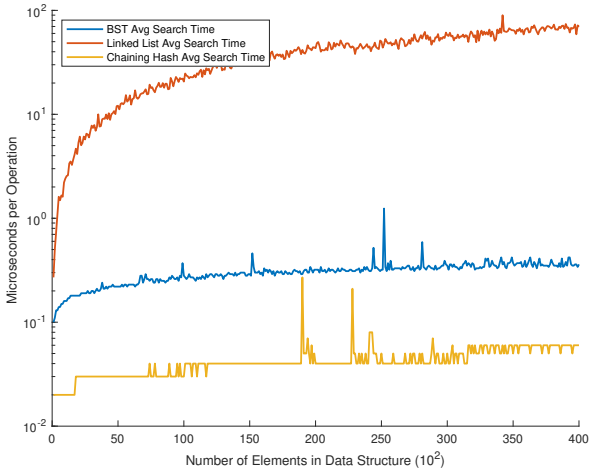

Figure 0.19. Search Summary

The summary graphs show the performance of the linked list, BST, and chaining hash structures relative to the number of elements stored in each structure. The graphs above use data set A. The Y axis of both graphs is logarithmic due to the extremely large insertion and deletion times of the linked list.

Performance Analysis

The results clearly show the complexity of inserting and searching each data structure as a function of the number of elements in the data structure (n). Based on the summary figures, the chaining hash table performed the best compared to the linked list and BST. This makes sense considering that the average complexity of the hash table is $O(1)$ for both insert and search, whereas the BST is on average $O(log(n)$ and the linked list is $O(n)$. The chaining hash table achieves this high level of performance because it avoids searching through most or all of the stored data in order to perform each operation. The linked list in comparison has such poor performance because every insert and search operation has to linearly search through each node, meaning a single insert or search call could involve tens of thousands of comparisons.