Tommy Hua

Trivedi/Deshmukh

Final Project

Summary

I have found that the best data structure for the USPS package tracking application is a Hash Table with chaining. Compared to other data structures, the Hash Table utilizing chaining produced the lowest times for inserting and searching for data. The hash table with chaining, which utilizes the linked list data structure, is more efficient than a linked list because of its search times. In my functions for the linked list and Hash Table with chaining, I inserted data points at the head of the linked list regardless of value, resulting in very similar insertion times. However, search times differ drastically because of a Hash Table’s indices. A Hash Table allows values to be searched for directly, occasionally having to go through a linked list. Unlike the linked list, where values must always be searched by going through the linked list.

Compared to a Binary Search Tree, a Hash Table is also more efficient. A BST is great at ordering data in a specific way, such as least to greatest, but when it comes to just raw data it does not do so well. Every time new data is inserted, the BST insertion time goes up in a linear fashion. Of course, a data structure getting slower over the course of more data being inserted into it happens across the board, but it happens much faster for a BST. I believe this is because a BST must make sure the data’s value goes into the right place, which increases the insertion time. Same goes for search times. But unlike insertion, not every value is going to be at the end of the tree, so it can take less time. A Hash Table is more efficient than a BST because the insert and search times stay relatively the same up until the Hash Table comes near its limit.

Compared to Linear and Quadratic Probing, Chaining also comes out on top, mainly because of processing times when the table is almost full. For Linear and Quadratic Probing, inserting into an almost full hash table becomes a problem, as finding the next open spot equates about to searching through the entire table. However, for chaining, this is not a problem. Chaining does not require an empty index, and this decreases insertion times when a hash table is nearing its limit. Search times for both probing and chaining are relatively the same, but once again, when a table nears its limit open addressing suffers, producing higher search times than chaining.

Looking at the test data trends, we can see that Data Set A has no trend whatsoever, and Data Set B has a very linear trend. Looking at the graphs we can see that this does not significantly affect any of the data structures except for the BST. I believe this is because of a BST’s nature. For Data Set A, when trying to order data values that are random, the left and right trees can grow unevenly. This would make insertion and search times very random, which explains the graph of BST (A). For Data Set B, a flow of data values in a linear trend would create a very even tree. This would make insertion and search times grow in a linear fashion, which explains the constant slope of BST (B).

Graphs:

Linked List (A)

Linked List (B)

Binary Search Tree (A)

Binary Search Tree (B)

Hash Table (Chaining) (A)

Hash Table (Chaining) (B)

Hash Table (Linear Probing w/ collision) (A)

Hash Table (Linear Probing w/ collision) (B)

Hash Table (Quadratic Probing w/ collision) (A)

Hash Table (Quadratic Probing w/ collision) (B)

Insert Time Summary (Hash Table w/ Chaining)

Search Time Summary (Hash Table w/ Chaining)

Data Points Comparison