Final Project: Hash Table Collision Algorithm Analysis

Purpose:

The purpose of this project was to construct hash tables using two different algorithms, chaining and open-addressing, for collision resolution, and to conduct tests on these algorithms to learn more about how well they perform. These hash tables were used to store data on every Major-League Baseball player on every team since 1985, which includes information such as their names, height, weight, teams played on, salary, etc. The goal was to successfully build the hash tables with the baseball player data and to evaluate the performance of the two algorithms used.

Procedure:

The hash table size is determined by user input in the command line and two tables are initialized with pointers to an empty node at each index of both tables. Two tables are created to test two different collision resolution algorithms. The algorithms used in this project to resolve collisions when constructing the hash tables were chaining and open-addressing. The chaining algorithm resolves collisions during the construction of the hash table by creating a linked list of data (a chain) at the index of the table where the collisions are occurring. Then when the user wants to find a baseball player stored in the table, the program finds the index where the player should be stored according the hash sum algorithm and then traverses through the linked list until that player is found. The other collision resolution algorithm, open-addressing, stores only one baseball player’s data at each index of the hash table. It works by first attempting to store the player’s data at the index of the table that is calculated by the hash sum algorithm and if that spot in the table is full, the index is incremented until the next open spot is found and then that player is stored there. To test how the two sorting algorithms performed when constructing the hash tables, various counters were implemented into the code to record the number of collisions that occurred when attempting to store the data in the table and to record the number of searches that took place to find an open spot for each player’s data to be stored. The collision resolution algorithms were also tested with hash table sizes of 5147, 10000, and 15000.

Data:

The data set used for this project was information on every Major-League Baseball player on every team since 1985. There was a total of 1547 unique players in this data set which was determined by using a unique player counter in the code that checked whether the player had been seen previously or not by comparing their first and last names, player ID, and birth country with players that have already been stored in the table. The information for each player included their name, birth year, birth country, height, weight, teams played on, salaries, and which arm they use to throw and bat with. To store each player’s data, their statistics were placed into a player struct where the teams played on each year and salary variables in the struct are vectors so that they may be updated if that player is seen again playing on a team during a different year, with a different salary. The key for each player was created by combining the first and last name. For example, the key for the player Len Barker would be “LenBarker”. To generate the hash table index, the hash sum algorithm that was taught during lecture was used. In this algorithm, the asci values for every letter in the key are added up and then divided by the table size and the remainder (index = sum % table size) becomes the index of the table where the program will attempt to store the player. There are other hash sum algorithms that could have been used to generate the hash tables, but I chose to utilize the one taught in lecture for my project.

Results:

After testing the collision algorithms with the use of the various counters and by adjusting the size of the hash tables, data was collected to compare the performance of chaining and open-addressing. For all three of the table sizes used for testing (5147, 10000, and 15000), the number of collisions recorded from chaining were the exact same. This is because of the hash sum algorithm that was used. The program always attempted to put the players at the same index that they would have been put at for all table sizes which resulted in the same number of collisions for all table sizes. In addition, the number of searches performed to store player data was the same for the previously mentioned reason. When a collision occurred, the linked list at that index was traversed until the next open spot was found or if the player was recognized and needed to have its information updated. In order to improve the performance of the chaining algorithm, a different hash sum algorithm could be used that finds more unique indexes to store the data in. For the open-addressing algorithm, its performance was the best when the table size was smallest, then it stayed the same for the very large hash tables (10000 and 15000). This is because the open-addressing algorithm resolves collisions by traversing the hash table until the next open spot is found to place the player’s data in. When the table is smaller, there are less indexes to traverse through, but once again the reason for the results comes from the hash sum algorithm. The indexes that were calculated to store the data in were the same for all test cases so the number of collisions were the same and the program needed to traverse through the table to find the next open spot. Both algorithms’ performance is heavily influenced by the hash sum algorithm that determines what index to place the data into. The results show that chaining is a better algorithm for this data set than open-addressing, this is because with open-addressing much more searches need to be conducted to traverse through the length of the hash table where chaining stays at the index it was assigned from the hash sum algorithm and just traverses through the length of the linked list which is a much smaller list than the table itself.

Pictures of Test Cases:





