Adam Ten Hoeve
106105239
Section 110, Thursday 11:00 am
TA: Mridula Natrajan

Hash Table Collision Algorithm Analysis Write-Up

Purpose:

The purpose of this project is to compare the chaining and open addressing methods of storing data in a hash table. We did this with a set of 25,000 lines of data from baseball players since 1985 which we sorted by the player's full name. As the players were added to the hash table, different players would be added to the same position causing collisions. We used two methods, chaining and open addressing, to resolve the collision issues and can compare the algorithms by seeing how many total collisions occurred with it. The algorithm with fewer collisions performs better.

Procedure:

Chaining is the process of adding all of the elements with the same hash value to a linked list that can then be traversed to get all of the values. If a collision were to occur at some index then the algorithm would start at the head of the linked list and work its way down until it reached the end or it finds a value greater than the added value, in which it would insert the value there. This way the linked list is sorted as it is being built. We used a command line to set the hash table size, which was 5147 for the most part for testing because that is the number of unique players. However, that number could have been smaller because multiple players were being stored at each index so the total number of positions used was less than 5147. In fact, only 895 positions in the hash table were used.

Open addressing was the other algorithm that was used to correct collisions. If a collision occurs, then the algorithm works by incrementing through the hash table until an empty (NULL) position is found and it stores the added value there instead. This method does not use any other data structure other than the hash table. Unlike chaining, for open addressing, the size of the hash table depends on the amount of data being stored. If there are not enough positions for all of the elements being added, then things will be overwritten because it will be full. Because there were 5147 unique players, the hash table had to have at least 5147 positions or else overflow.

Data:

The data for this project was 26,400 lines of statistics on baseball players. Each line involved a player's player ID, first name, last name, birth year, birth country, weight, height, batting arm, and throwing arm, as well as the year, team, league and salary for that season. As some players have played in multiple seasons, instead of adding the player multiple times, we created arrays within the player's struct that holds the years playing, teams played on, league, and salary for each season they played. Each player was then sorted into the hash table with their full name, as in the first name + last name, being the key. All of this information on the player was stored in a Player struct which was then inserted into the hash table.

Results:

The results of the analysis was that, for hash tables of size 5147, the chaining algorithm performed better with 4251 collisions than the open addressing algorithm which had 4819 collisions. The chaining algorithm also performed better in the number of searches performed, with chaining only having 94,451 searches to open addressing having 46,129,771 searches. When searching for individual players, the numbers can vary but are primarily in favor of chaining. When searching for Len Barker, open addressing goes through 0 searches while chaining goes through 6 searches. This is because Len is the first player added to the hash table so he is guaranteed to be in that location for open addressing while chaining sorts the linked list so he gets moved further back. Although this may look good for open addressing, it quickly breaks down as more players with the same hash number are added. If Nick Green, who has the same hash number as Len Barker, is searched for then chaining returns 9 search operations but open addressing returns 3221 because he is stored so much farther down in the hash table. Because chaining had to perform less operations for collisions and searches, chaining is overall the better algorithm. The reason chaining performs better is because the chaining algorithm "goes long" and stores multiple things in a single index location, whereas open addressing goes wide and stores values with the same hash sum in different locations. This causes a large buildup within the hash table of values taking the place of other values which cause more collisions than storing multiple in one location. When searching the hash table, it would take a lot longer to find the value with open addressing because the value could be stored in any position in the hash table because it does not have to be at its hash sum location. Whereas with chaining, you know it has to be at a single position and just have to traverse a linked list to find the desired value. This conclusion is supported by the data above, that chaining took much fewer operations than open addressing.