Liam Kolber (code partner: Cameron Heppe)
Linda Jacobson
CSCI 2270
5 May 2017

Hash Table Collision Algorithm Analysis

Purpose:

This project focused around two separate collision resolution techniques (open addressing and chaining) used to properly distribute data in hash tables. Students were provided with a set of data covering Major League Baseball players and a few statistics about them starting from 1985 to 2016. With thousands of data points and multiple repeated players due to continuity from year to year, this type of data is a great example for the usefulness of something like a hash table. The algorithms each organize and traverse through the data differently; fortunately, the size of the provided data set allows for a decent assessment of the efficiency and speed of each of the methods.

Procedure:

Open addressing is a hashing method that adds data to the set in the first open location following it's expected index should that location be full. There are multiple methods for open addressing — linear, quadratic, and double — of which linear probing was used. Linear probing the process in which locating a new location for a datapoint is done progresses at a constant search interval. This places the datapoint at the closest possible index to the expected index given by the hash function.

Chaining is a process in which a collision is resolved by creating a linked list at each index. Should a collision occur, the new datapoint is added to a specified location within the associated linked list. In the case of this project, the datapoint was just added to the end of the list as no organization (i.e. alphabetical order) was necessary for the report. This method can differ from the open addressing method as it allows for more indices within the hash table to be unused which can prove less efficient at smaller hash table sizes.

As with any list of algorithms that achieve the same result, there are benefits and drawbacks to each method. With open addressing, less memory is necessary as the only stored information is the datapoint and a location. In chaining this is not the case as each index contains a linked list, so each data point also needs to include pointer information in addition to the already stated information. It also results in some empty indices which is simply wasted space.

A benefit to chaining, however, is that similarly hashed datapoints (depending on the chosen key) in certain situations could provide a better organized hash table as everything within a given linked list could generally be related whereas an open addressing hash table has information more scattered.

Data:

The data used in this project focused on information covering ever Major League Baseball player from 1985 to 2016. The total number of data sets was just over 26,000 which goes to show why an organize and search algorithm that are used in this project can be so useful. Each piece of data had 13 different fields of information. These fields provided in order were the year, the ID code for the team, the ID code for the league, the ID specific to the player, their salary for that year, their first name, their last name, their birth year, birth country, weight, height, the direction in which the player bats, and the arm with which the player throws. When adding the player the to hash table, their first and last name were concatenated and the resulting string was used in the hashing function to determine their index. As expected though, it wasn't out of the question that some players would have the same name especially considering that the data covers 30 years of names. When these issue came up, a check in the algorithm was implemented to compare their unique player ID's to determine which one the new data set was referring to.

Results:

As stated previously, both algorithms have their benefits and their drawbacks. After running the code and simply building the hash tables, it's clear that open addressing algorithm experienced significantly less collisions than the chaining method. The open addressing algorithm had on average about 1000 collisions across the three test sizes (5147, 15000, and 20000) with a general decreasing trend as the table size increased. The chaining method, however, had a generally increasing trend in the number of collisions as the table size increased, but the 20000 table size trial reported slightly less collisions than the 15000 table size trial. This suggests that the table size generally doesn't carry a lot of weight when determining the distribution with the chaining algorithm.

When it came to traversing through the data to find the player that the user desired, a similar result occurred. At the lower table sizes the chaining method appeared to be the quicker of the two as the open addressing method often showed hundreds and even thousands more search operations than the chaining method. As the table size increased, the number of operations in the chaining method was pretty static, and the number of operations in the open addressing method decreased significantly to show lower numbers than the chaining in many cases. This shows that chaining is in general a pretty decent algorithm as it is consistent for varying table size. A reason behind this is because of the use of linked lists, there are typically only a handful of steps needed to be taken after determining the index of the datapoint whereas with open addressing, if a datapoint toward the end of the datafile is added with the same index as something that showed up early on, then it's possible that two points with the same index could be on completely opposite ends of the hash table which can negatively affect search operations and time need to find the information.