Philip Sloan

CS 240

July 17th , 2014

Project 2: Hashing

Specifications

The purpose of this project was to explore a hash table using data from files. By utilizing separate chaining, a hash table had to be created that would keep track of the average scores of the various NFL teams over the seasons. The program needed to be able to print out the number of collisions, the size of the table, as well as the highest and lowest averages in the table. Additionally, the program had to be able to take input in the form of a String from a user and display the average and the number of scores that had be entered for that team.

The hash table itself used separate chaining which essentially is an array of head nodes in a linked list. Whenever a collision occurs, the new value is added to the head of the linked list, resolving the collision without creating additional entries in the table array. The hash function I chose to use was a cyclic shift, as we discussed in class as being particularly good for strings.

Analysis

The initial table size was a size of 50. With that size, a total of 10 collisions were created. That is a pretty high collision rate. At a size of 58, the collisions were only 5. Increasing the size more reduced the collisions, and the first size where there were no collisions was a table of 244. In general, it seems that past 58 or so, the amount the size needed to be increased to reduce collisions went up considerably, resulting in a sort of diminishing return.

Another way to reduce the collisions was to play around with the values in the hash function. Initially, the values used in the class example (5, 27) were used. By changing that to 7 and 31 the size where zero collisions occurred was reduced to 158. It is obvious that changing the values has an effect, but 7 and 31 seemed to be one of the better combinations. Still, at a size of 50, 10 collisions remained.

Lessons Learned

In this project I learned that a node in a linked list can be a very versatile tool to expand the functionality of the program. By making each value in the table essentially a node object, the node could keep track of the team name as well as how many scores had been entered. The put method was one I struggled with a little bit, but eventually I figured out what I was doing wrong. Basically, after each entry I was checking if the new value changed the Minimum and Maximum average, but because the average was changing constantly as scores were added, this resulted in incorrect final averages. I eventually put a calculateMinMax function in that could be run once all of that data was entered in.