Juhwan Lee

CS-163

August 2nd 2020

Program #3 Efficiency Write Up

Program #3 was about working with Table Abstraction. A Table ADT allows us to search on the “value” of the data without requiring a particular location to be known. It allows us to store the data using non-linear techniques. There are two non-linear pointer based data structures and those are hash table and binary search tree. In Program #3, we used hash table. A hash table comes to mind as a good alternative for programs that require working by “value” but that does not require that the data be sorted. A hash table is a data structure that maps keys to values. This uses a hash function to compute indexes for a key. Based on the hash table index, we can store the value at the appropriate location. The whole benefit of using a hash table is due to it’s very fast access time. However, the hash function may lead to collision that is two or more keys are mapped to same value. Chaining avoids collision. The idea is to make each cell of hash table point to a linear linked list of datas that have same function value. Our goal was to build a program that keep track of superhero comics that we are interested in. We used an external data files for this program. The information that we keep track of in this program includes:

1. The name of the comic (Amazing Spider-Man)
2. The publisher (e.g. DC Comics, Marvel, Dark Horse, etc.)
3. The superhero (e.g. Spider Man)
4. The issue (Issue #2 of the Amazing Spider-Man)
5. A review (from you or someone else)
6. Other items you are interested in

The data file’s name is “Comic.txt” and the delimiter is a ‘ | ‘. And the Table ADT have these functions:

1. Add new comic into the table
2. Retrieve information about the comics entered by the superhero name
3. Remove a comic based on the name of the comic book
4. Display all comics featuring a superhero
5. Display all (not ordered)

After testing this program, I have discovered that if the table size is bigger, the number of collision reduces which results better runtime performance when accessing the data. If the table size is small, there will be more collision which makes the linear linked list longer. In terms of Big O, in the best case scenario, the runtime performance is O(1) and in the worst case scenario, the runtime performance is O(length of linear linked list). So reducing the number of collision is important. Moreover, while I was working on this program, I have noticed that the number created by hash function needs to be a big number so that the mod operation helps distributing the data. Everything became clear after working on this program however, there were some parts that I didn't quite understand. When I use key value to access the data, it goes through a hash function which creates the index so that it takes me where the data is, however, I needed to implement a function that remove a comic based on the name of the comic book and the name of the comic book was not used for key value, all the data was stored with the index created by the superhero name. So when I implement the remove a comic based on the name of the comic book, I used the technique where it checks every single array index with every single linear linked list node. Therefore, I assume that its performance is significantly slower than the retrieve function where it uses superhero name which creates the appropriate index. I am not sure if it was meant to be like this or if there was some other way to use hash function to create an appropriate index for that data.