Program #1 Review

Written by Alexander DuPree

Data Structure Performance

Program #2 utilized a hash table data strucute to implement a Table ADT. The hash table Performs admirably up to a certain point. When the load factor of the table reached a certain threshold the number of collisions increased, worsening the runtime performance of the table.

To accurately asses the performance of the hash table I utilized a data set of 1002 unique entries. The following are the results of the hash table
with different table sizes:

Table Size	Load Factor	Longest Chain	Average Chain	Chains Used	Prime?
101	992.70%	17 Items	9 Items	100%	Yes
128	782.81%	27 Items	7 Items	100%	No
541	185.21%	7 Items	2 Items	82.07%	Yes
512	195.70%	13 Items	2 Items	81.05%	No
1013	98.91%	5 Items	1 Items	61.50%	Yes
1024	97.85%	8 Items	1 Items	58.59%	No
1223	81.94%	5 Items	1 Items	55.36%	Yes
2048	48.93%	5 Items	1 Items	36.81%	No

As the table above shows, utilizing powers of 2 as the table size results in reduced performance. Using a prime number for the table size had the greatest performance; However, that performance tapered off despite the load factor being greater. It seems my hash function would cluster certain data items into groups of two. Utilizing 100% of the available chains only happened when the load factor was significantly greater than the table size.

Data Structure Recommendation

The hash table's biggest drawback is how it handled collisions. For this assignment we used chaining. This lead to the problem that as the load factor increased and collisions occured, we had to traverse multiple nodes to retrieve data. One solution to this problem would to have been to treat collisions as overwrites, replacing data at the hashed index. This would have ensured constant time operation but would have destroyed any data that hashed to the same index. A better alternative would have to implement a hash table of binary search trees. This would mean that at the worst case we would still be able to access our data in logarithmic time.

Design

As far as application design, I feel that the data structures were sufficiently abstracted. A client using the application would have difficulty determining exactly which data structures were being used. The main, and other classes had no knowledge of the specifics of the Table ADT's implementation. Instead, they relied on a public interface to accomplish the goals of the program.

Inefficiences

Despite many tweaks, it was still difficult to achieve an even dispersion of data with my hash function. Even with a load factor of 185% the hash table only used 85% of the available indexes. With more time and research into the subject of hashing, I would have liked to have implemented a more robust hashing algorithm.