Assignment 5

Julius Ware

The Binary Search Tree is a good data structure to use because it is memory efficient. Unlike an array based data structure it does not need to reserve more memory than what is actually being used. On average the complexities of find, add, and remove are O(log n) when balanced. The code uses recursive calls which makes implementation very easy. Also great for ordered data.

The biggest disadvantage of a BST come when adding data in sequential order. If this phenomenon occurs the runtime of all functions becomes O(n) which is very inefficient; the datat from my test runs show this explicitly. It is not ideal when data is to be entered in numerical order. To avoid this inefficiency, data can be entered randomly to ensure a more balanced tree.

The Red Black Tree is a very fast very popular data structure. Its add and remove functions ensure for a worst case of O(log n) complexity making this one of the fastest data structures. A RBT is simulated 2-4tree which ensures that the tree is mostly balanced at all times. Because of its red and black node properties it allows for self-balancing thus eliminating the problem of the binary search tree.

A big disadvantage of the RBT is how complex it’s code it. The many different balancing function can be very complex and hard to implement and grasp. Although they are hard to implement the speed and efficiency of the RBT makes up for the implementation.

ChainedHashTables are great for unordered data and likewise implement the USet interface. They are special because of their hash function. This function ensures that data is entered in a randomized order and allows for balanced “buckets”. The backing array holds pointers to lists where the actual data is stored. This is also memory efficient when needing to store big data object.

Because CHT ‘s use a backing array they require a block of memory before entering data. This can become very memory inefficient when lists become very long and the backing array has to resize. A way to help curve this is to allow for a bigger load factor in the “buckets”. Also when entering data certain items might hash to the same list more than others. Because of this the add and remove functions have a O(n) worst case scenario. On average they perform in O(1) time which is shown in my data.

LinearHashTables are similar to CHT’s except they use linear probing rather than a modular function. They also place data directly in the backing array eliminating the need for “buckets”. LHT’s used three types of data: data, null, deletes. Null and delete values help control the expanding and contracting of the backing array.

The same problem occurs with linear hashing as with chained hashing. If data is entered in a certain order the backing array can become longer and longer thus pushing the run time to O(n) complexity.