Course: CSE 274

Assignment 5

Jiyuan Xu

**===== Print report for BinarySearchTree =====**

**1. Average Time processing RandomAdd is: 17095**

**2. Average Time processing RandomFind is: 13551**

**3. Average Time processing RandomRemove is: 19925**

**4. Average Time processing DoSequentialAdd is: 7836750**

**5. Average Time processing DoSequentialFind is: 7824690**

**6. Average Time processing DoSequentialRemove is: 4974**

**===== Print report for RedBlackTree =====**

**1. Average Time processing RandomAdd is: 19352**

**2. Average Time processing RandomFind is: 12220**

**3. Average Time processing RandomRemove is: 23840**

**4. Average Time processing DoSequentialAdd is: 14311**

**5. Average Time processing DoSequentialFind is: 6583**

**6. Average Time processing DoSequentialRemove is: 15696**

**===== Print report for Linear Hash Table=====**

**1. Average Time processing RandomAdd is: 8555**

**2. Average Time processing RandomFind is: 2004**

**3. Average Time processing RandomRemove is: 6745**

**4. Average Time processing DoSequentialAdd is: 5606**

**5. Average Time processing DoSequentialFind is: 1452**

**6. Average Time processing DoSequentialRemove is: 4893**

**===== Print report for ChainedHashTable =====**

**1. Average Time processing RandomAdd is: 21014**

**2. Average Time processing RandomFind is: 975**

**3. Average Time processing RandomRemove is: 1808**

**4. Average Time processing DoSequentialAdd is: 21537**

**5. Average Time processing DoSequentialFind is: 4344**

**6. Average Time processing DoSequentialRemove is: 6750**

The most obvious disadvantage of a binary tree is that when adding elements in order, the tree will become unbalanced. We can simply prove it by looking at the time it take for sequential add and sequential find. The longest time compare to other data structure when adding same amount of elements. In this case, we are seeing worst or close to worst case O(n).

However there are advantages of binary search tree such as code is easier than other data structures, storing keys in the node so that searching, inserting and deletion can be done efficiently, nodes are dynamic and implementation is very easy.

I like red black tree because its always balanced which means the complexity is always O(logN). To compare this with unbalanced binary search tree, we can clearly see that the sequential add and find did take significantly less amount of time which means the process gets efficient when user is adding elements into the tree in order. It has relatively low constants in a wide variety of scenario and plus the fact that RB tree inherits all advantages of binary search tree, its very useful data structure.

However, using red black tree is not easy in terms of understanding the code. Since RB tree involves a lot of shifting and updating, when number of elements is big enough, we can notice deleting item from red black tree takes more time than binary search tree.

There are few advantages of using chained hash table. The most obvious one is hash table is in average faster than trees since the best case of complexity is O(1) and its hard to get the worst case O(n) using a good hash function. There are few other advantages of using chained hash table. It can be expended to infinite size, easy to implement and will never search inefficiently.

The disadvantage of hash table is that if hash function is not good enough to avoid collisions, elements can be saved in one single list and therefore cause the time of searching and deleting an item inefficient.

Linear Hash inherits all advantages of Chained hash table, including fast searching and deleting operation and low complexity of O(1) when hashing well.

However, Linear hash table has longer time processing add because it involves more steps for hashing a position to add item.

Red-Black Tree and 2-4 Tree

A red black tree is a binary version of 2-4 tree. It’s a simulation of 2-4 but more efficient. However the conversion from 2-4 trees to red black is sometimes very complicated. When changing, two methods are often used. Pushblack() takes as input a black node u that has two red children and color u red and its children red whereas pullBlack do the other way.