ADS Project - Fall 2012

Submitted by:
Bhimender Arora

Compilation Instructions : Compiler to use : 'javac'. All the '.java' files, including 'dictionary.java', are stored in one folder, 'arora_bhimender'.
In the directory,

javac *.java

compiles all the files.
To run the following commands are used:

java dictionary -r *s order*
java dictionary -u *filePath*

Please note here that, the program expects either full path for the input file, or the input file to be present in the same directory as the classes.

Function Prototypes

dictionary
- writeToFile(String, ArrayList<String>)
- main(String[])

AVL
- search(int, AVLNode)
- add(int, int, AVLNode)
- height(AVLNode)
- LL(AVLNode)
- RR(AVLNode)
- inOrder(AVLNode)
- inOrderWrite(AVLNode, ArrayList<String>)
- postOrder(AVLNode)
- postOrderWrite(AVLNode, ArrayList<String>)

AVLNode
- key : int
- value : int
- rc : AVLNode
- lc : AVLNode
- balanceFactor : int
- height : int
- AVLNode(int, int)
- setKey(int)
- getKey()
- setValue(int)
- getValue()
- setRc(AVLNode)
- getRc()
- setLc(AVLNode)
- getLc()
- setBalanceFator(int)
- getBalanceFator()

setHeight(int)
getHeight()

AVLHash
AVLNode(int, int)
setKey(int)
getKey()
setValue(int)
getValue()
setRc(AVLNode)
getRc()
setLc(AVLNode)
getLc()
setBalanceFator(int)
getBalanceFator()
setHeight(int)
getHeight()

BNode
key : ArrayList<Integer>
value : ArrayList<Integer>
pointers : ArrayList<BNode>
weight : int
order : int
BNode (int, int, int)
setKey(int, int)
getKey(int)
setValue(int, int)
getValue(int)
getFirstKey()
getLastKey()
getChild(int)
addChild(BNode, String)
getOrder()
getWeight()
insertIntoLeaf(int, int)
find(int)
split()
merge(BNode)

BTree
addRecurse(int, int, BNode)
add(int, int, BNode)
search(int, BNode)
sorted(BNode)
sortedWrite(BNode, ArrayList<String>)
level(BNode)
levelWrite(BNode, ArrayList<String>)

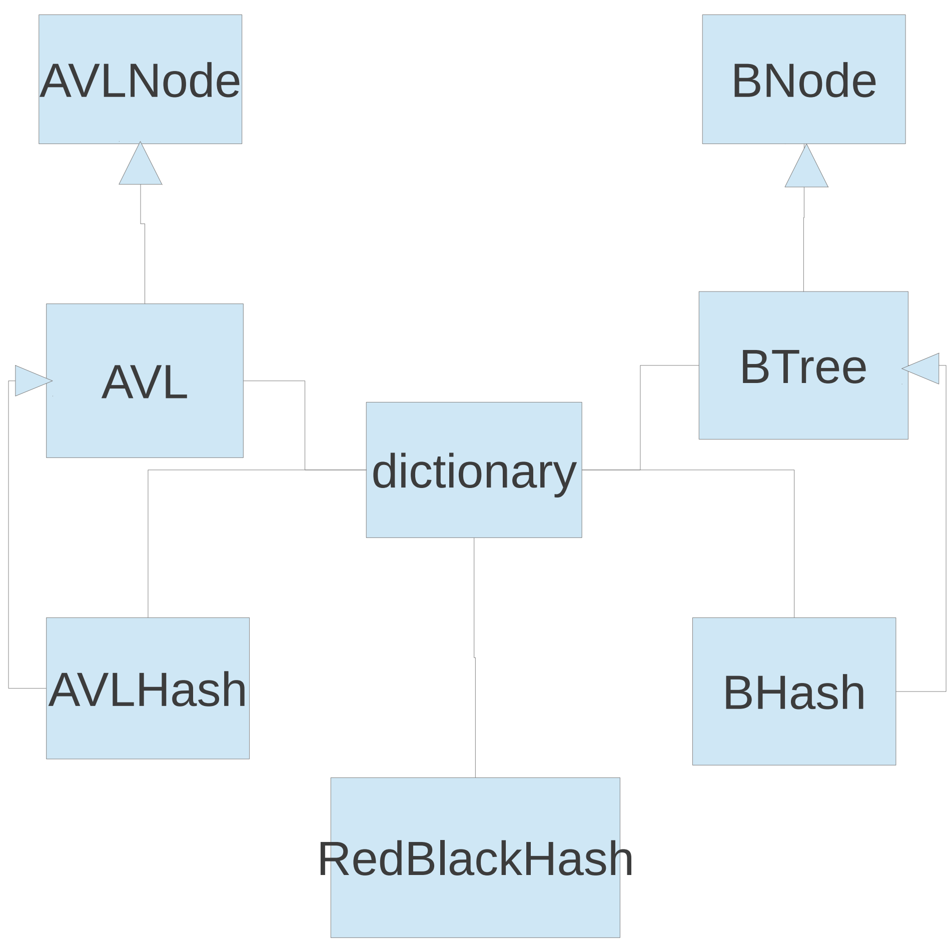
BHash

- s : int
- BArray : BNode[]
- Btree : BTree
- order : int
- BHash(int, int)
- add(int, int)
- search(int)
- level()
- levelWrite(ArrayList<String>)
- main(String[])

RedBlackHash

- s : int
- TMArray : TreeMap<Integer, Integer>[]
- RedBlackHash(int)
- add(int, int)
- search(int)

Class Diagram



Expectations:

The insert and search operations for all these tress, ie AVLTree, RedBlackTree andBTree should be completed in logarithmic times, as the the height of each of these tree is O(log n). More precisely, the heights are :

AVL – $\log_{(base2)}$ (n+2) to 1.44 *$\log_{(base2)}$ (n+2)
RedBlackTree - $\log_{(base2)}$ (n+1) to 2$\log_{(base2)}$ (n+1)
Btree – $\log_{(base\ m)}$((n+1)/2)

where n is the number of elements and m is order of Btree.

Btree best for searching as the height of Btree is considerable less thean the other two. One may also expect the search for AVL to perform faster than that of RedBlack, in the worst case.

For insertion, as the balacing of Balance factor and colors take the same amount o time, the time taken for insertion for AVL and RebBlack should be comparable in Average and AVL better in Worst case. Again, for Btree the insertion time should be less owing to the smaller height, notwithstanding a complex balancing mechanism.

Hashing improves the insert and search time for all the tree types. Higher value of s should mean better performance.

Result Comparison

To search for the **optimal B-Tree order**, a number of tests were done. These use number of keys entered, n=1000000, in the random mode. The keys were added to both Btree and BtreeHash and then searched in the same order. For each of the configuration, 10 iterations were done and the average time taken was reported.

s=Order	BTree Insert	BTree Search	BTreeHash Insert	BTreeHash Search
3	4311.8	3240.6	4243.4	1174.2
5	2737.4	2187.4	2578.7	1189.5
10	2302.8	1636.4	1955.1	719.7
20	2262.4	1418.1	1869.8	505.6
25	2255.7	1634.7	1909.7	1321.4
30	2283.1	1666.7	1900.9	555.3
40	2361.6	1696	1956.2	522.8
50	2462.4	1692.2	2065.9	557.2
100	3084.3	1817.6	2448.4	709.1
200	3707.4	2178.1	2925.9	747.5
500	4527.1	2890.6	3562.8	815.1

*All Values in ms. n=1000000

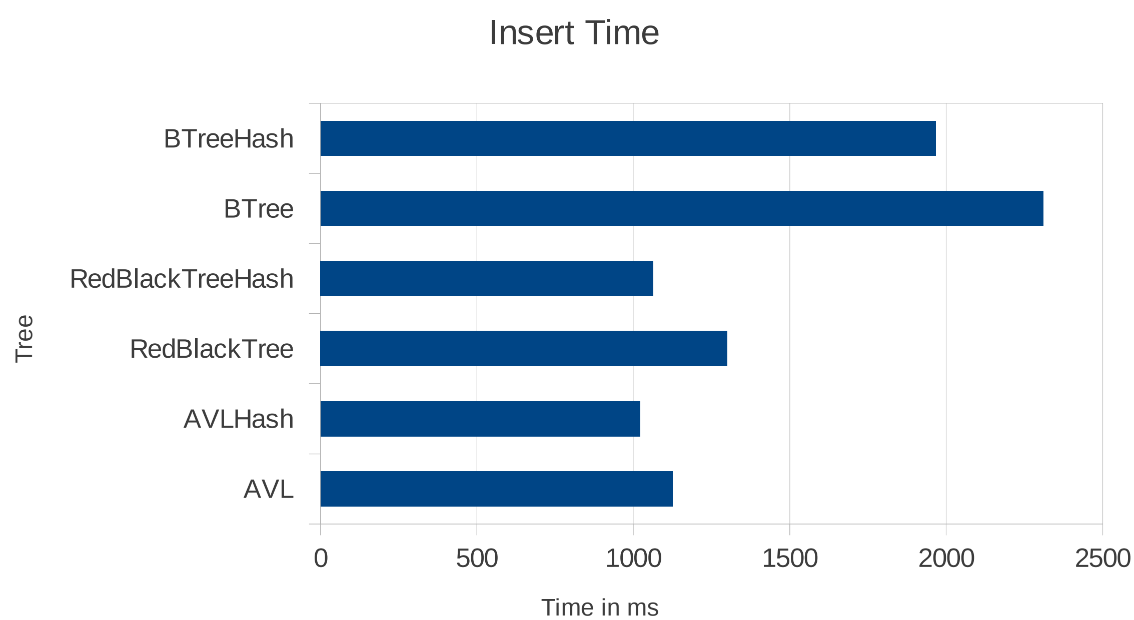
It can be seen that best performance is between order values 20 and 30. For further tests, order = 25 was used.

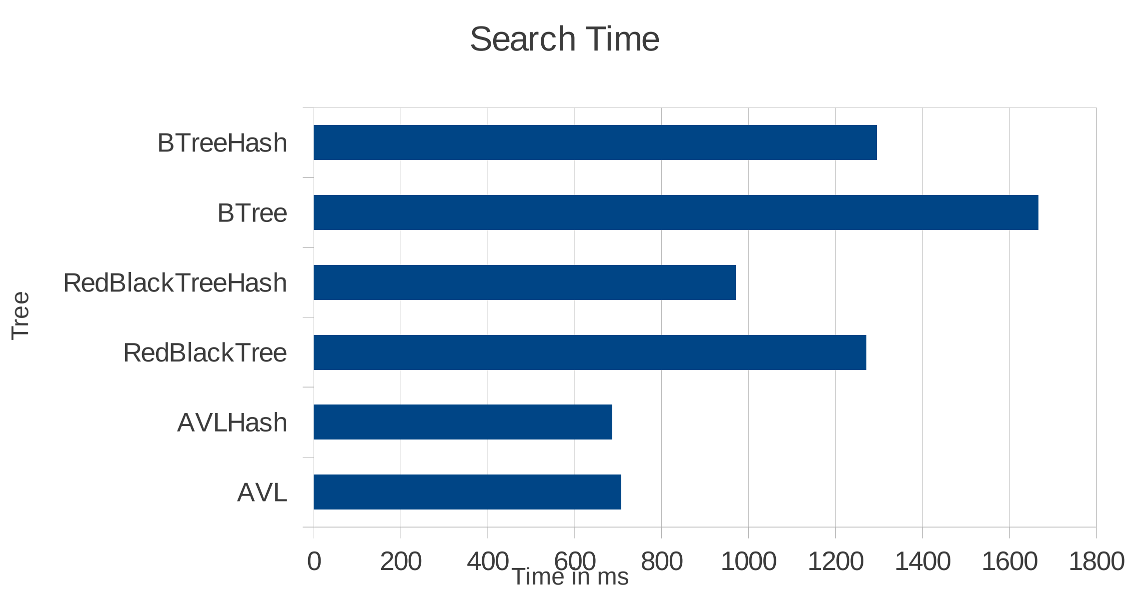
Next the values values were experimented with, using the same testing conditions.

s values →	3	11	101
AVL Insert	1112.1	1103.2	1125.6
AVL Search	753.1	750.2	707.2
AVLHash Insert	1104.4	1054.2	1021
AVLHash Search	736.9	706.8	686.4
RedBlackTree Insert	1308.3	1263.1	1300.6
RedBlackTree Search	1210	1167.6	1271.1
RedBlackTreeHash Insert	1085.1	1103.8	1063.4
RedBlackTreeHash Search	1038.2	981.2	970.8
BTree Insert	2134.1	2177.9	2310.3
BTree Search	1461.9	1489.1	1666.7
BTreeHash Insert	2162.5	2014.4	1966.8
BTreeHash Search	1556.8	1430.7	1295.5

*All Values in ms. n=1000000. Order = 25

These values can be presented as graphs:





It can seen here for hashed structures, the performance increases with s. The effect of increasing s is more noticeable in search times.