Ahmed Yasser Ahmed 10
Hossam ElDin AbdElGhany 24

RB Tree Report

Red-Black Tree is a self-balancing Binary Search Tree where every node follows following rules: every node has a color either red or black, root of tree is always black, there are no two adjacent red nodes (A red node cannot have a red parent or red child), and every path from a node to any of its descendant NIL node has the same number of black nodes.

Time complexity of the methods in the RB Tree Class:

1) Contains/ Search: The contains method in similar in its structure to the search method, the search method in the red black tree is O (log n) since the red black tree is nearly a balanced tree.
2) Insert: The insert method consists of three steps:
   a)looking for the suitable place for the insertion which takes O(log n) because it is in essence a search in the tree
   b)inserting the node in its right place O (1)
   c)fixing up the tree it has a worst case of O (log n)
   which In total equates to O (log n).
3) Delete: similar to the insert method it consists of three parts finding the element, deleting it (or exchanging it with its predecessor and deleting it), and fixing up the tree their total equates to O (log n).

Time complexity of the methods in the Tree Map Class:

As the tree map is implemented using the red black tree above, many of its methods has the same complexity as the methods in the RB Tree class.

1) Size: It only return a value of a field named (size), so it is O (1).
2) Clear: It just calls the clear method in the RB Tree class, which delete all the nodes, so it is O (n * log n)
3) Put/ Remove: They only call insert or delete methods in the RB Tree class and increment or decrement the size by 1, which make them O (log n) as well.
4) ContainsValue/ Get: They only call contains or search methods in RB Tree class, which make them O (log n) as well.
5) All method related to first or last entry: As they find the greatest or the smallest element in the tree, they traverse along the height of the tree, which equals to (Log n) because the red black tree is nearly balanced. So these methods are also O (log n).
6) All methods related to ceil or floor entry: In the worst case scenario, these methods will traverse along the height several times (c * h = c * log (n)). Which make these methods O (log n).
7) EntrySet/ KeySet/ Values/ HeadMap: These methods visit each node in the tree, so they are O (n).
8) PutAll: This method performs the insertion operation n times, so it is O (n * log n).

Tests

UnitTest (eg.edu.alexu.csd.filestructure.r	22 s 576 ms
testContainsAbsentKey	135 ms
testGetElementInTreemap	285 ms
testDeleteAbsentElementsInTree	270 ms
testfloorEntryWithNull	5 ms
testputAll	9 ms
testLastEntry	40 ms
testStressContains	8 s 143 ms
testContanisValueNormal	6 ms
testpollFirstEntry	9 ms
testEntrySet	10 ms
testputWithNullValue	3 ms
testRemoveNotFound	7 ms
testUpdateValue	5 ms
testSearchEmpty	4 ms
testCeilingEntryWithNull	4 ms
testpollLastEntry	8 ms
testGetElementInTreemapNotFound	22 ms
testputAllWithNullValue	3 ms
testInsertionWithNullKey	6 ms
testIsEmptyTrue	3 ms
testFirstEntry	15 ms
testNormalInsertion	3 ms
testClearTree	6 ms
testContanisValueWithNullparameter	4 ms
testGetRoot	9 ms
testCeilingKey1	7 ms
testCeilingKey2	58 ms
testLastKey	16 ms
testDeleteWhileInsertingInTree	103 ms

testContaninKeyNotFound	4 ms
testSearchAbsentKey	11 ms
testContanisValueNotFound	4 ms
testCeilingKeyWithNull	3 ms
testContainsEmpty	2 ms
testGetElementInTreemapWithNullParamert	2 ms
testSearchWithNull	2 ms
testNormalContains	4 ms
testRemoveNoraml	4 ms
testDeleteWithNull	3 ms
testHeadMapWithNullparameter	5 ms
testInsertionWithNullValue	5 ms
testFirstKey	6 ms
testCeilingEntry1	6 ms
testCeilingEntry2	30 ms
testHeadMap	62 ms
testContaninKeyNormal	5 ms
testRootNull	5 ms
testRemoveWithNullparameter	3 ms
testfloorEntry1	5 ms
testfloorEntry2	31 ms
testHeadMapInclusiveWithNullparameter	5 ms
testNormalSearch	4 ms
testfloorKey1	4 ms
testfloorKey2	25 ms
testHeadMapInclusive	42 ms
testContaninKeyWithNullparameter	2 ms
testKeySet	44 ms
testStressDelete	4 s 780 ms
testputWithNullKey	3 ms

testClearElementsInTreeMap	17 ms
testNormalInsertionWithRandomData	4 ms
testfloorKeyWithNull	2 ms
testDeleteAllElementsInTree	74 ms
testValues	14 ms
testContainsWithNull	4 ms
testDeleteRandomElementsInTree	83 ms
testStressSearch	8 s 65 ms
testIsEmptyFalse	4 ms

IntegrationTest (eg.edu.alexu.csd.filestructure	53 ms
testCreationRedBlackTree	48 ms
testCreationTreeMap	5 ms

```
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\Geek> cd C:\Users\Geek\git\RBTreeAndTreeMap\RBTreeAndTreeMap\bin
PS C:\Users\Geek\git\RBTreeAndTreeMap\RBTreeAndTreeMap\bin> java -jar RedBlackTreeTester.jar
Total tests passed: 70/70
PS C:\Users\Geek\git\RBTreeAndTreeMap\RBTreeAndTreeMap\bin>
```