Paul Diaz

CS 146

Prof. Potika

RedBlackTree

In this project, I created a data structure called Red Black Tree, which is a balanced binary search tree. Red Black Tree is the same as Binary Search Tree, except that we incorporate another field in the node; which is color.

Red Black Tree properties are: 1. Every node is either red or black, 2. Every leaf (null pointer) is black, 3. If a node is red, both children are black, 4. Every path from node to descendant leaf contains the same number of black nodes, and finally, 5. The root is always black.

**Tree Creation**

|  |  |
| --- | --- |
| Tree Creation | System.currentTimeMillis() |
| Total of 30 Entries of Insertion | 6 |

**Dictionary Look Up**

|  |  |
| --- | --- |
| Dictionary Look up | System.nanoTime() |
| Total of 30 Entries to Visit (Successful look up) | 9912 |

|  |  |
| --- | --- |
| Dictionary Look up | System.nanoTime() |
| Total of 30 Entries to Visit (Unsuccessful look up) | 9248 |

**Note:**

1. Using System.nanoTime() for Dictionary look up to actually get a number result; using System.currentTimeMills() always gives me 0 time.
2. I did a little bit of modification in JUnit, inside makeStringDetails(RedBlackTree t) inside if(!(n.data == null)) to make the test work.