Computing Red and Black Binary Tree using Python

Homework #10

By

Mario Pendleton

CS 303 [Algorithms and Data Structures](https://uab.instructure.com/courses/1507655)

November 1, 2019

### Problem Specification

Implement the binary tree algorithm and analyze the results using the give input files.



**Problem**

1. Implement Left Rotation and Right Rotation methods in the RBNode class. The Left-Rotate algorithm is given in the next page. Following that algorithm, devise the Right-Rotate algorithm and implement the methods.

2. Implement a method RBInsert(K key, V value) in RBTree.java to insert a node to a Red Black Tree (this method uses the insert method of BinarySearchTree, look at the algorithm next page).

(i) Implement the case for which parent of x is the left child of grandparent of x.

(ii) Implement the case for which parent of x is the right child of grandparent of x.

3. Write a driver program to test the RBTree class. Make sure you use all the methods implemented above.

4. Use the RBTree class to build a search tree using the given input file that consists of two fields: a UPC key and the corresponding description. Use the search tree created to find the description associated with a given set of UPC keys. The input file UPC-random.csv provides the key and corresponding descriptions in a comma separated file and the various search keys are provided in the file input.dat. First test the program by entering couple of keys manually and print the description. Once you are convinced the program is working correctly, test the program for the given search keys and determine the total time taken to complete the search.

5. Compare the times for searching the keys using the RBTree with the corresponding performance of given function, linear probing, quadratic probing with hashmap that was implemented in your last homework.

### Program Design

This program requires an array of data that will be sorted using the binary tree. The core methods were design after the pseudo code below.

R B - l NSERT ( T, *z)*

*y* = *T.nil x* = *T.root*

**while** *x =/- T. nil*

*y=x*

**if** *z . key* < *x.key x* = *x.left*

**else** *x* = *x . right*

*z.p* = *Y*

**if** *y* == *T.nil*

*T.root* = *z*

**elseif** *z . key* < *y. key y.left* = *z*

**else** *y . right* = *z z.left* = *T.nil*

*z. right* = *T. nil*

*z. color* = RED

R B -l NSERT-F IXUP( T, *z)*

L EFT-R OTATE( T, *x)*

*y* = *x .right*

*x. right* = *y. left* **if** *y. left =I- T. nil y.left.p* = *x*

*y.p* = *x.p*

**if** *x.p* == *T.nil*

*T.root* = *y*

**elseif** *x* == *x .p. left x . p.l eft* = *y*

**else** *x . p.ri ght* = *y*

*y .l eft* = *x x . p* = *y*

*II* set *y*

*II* tum *y* 's left subtree into *x* 's right subtree

*II* link *x* 's parent to *y*

*II* put *x* on *y* 's left

R B - l NSERT -F IXUP ( T, *z)*

**while** *z.p.color* == RED

**if** *z . p* == *z . p. p. left*

*y* = *z.p.p.right*

**if** *y. color* == RED

*z . p. color* = BLACK

*y.color* = BLACK *z . p. p.color* = RED *z* = *z . p. p*

**else if** *z* == *z . p. right*

*z* = *z . p*

L EFT-R OTATE( T, z)

*z . p.color* = BLACK

*z . p. p. color* = RED

R IGHT- R OTATE( T, *z.p.p)*

**else** (same as **then** clause with "rig ht" and "left" exchanged)

*T.root.color* = BLACK

*II* case 1 *II* case I *II* case 1 *II* case I

*II* case 2

*II* case 2

*II* case 3

*II* case 3

*II* case 3

The following steps were required to develop this program:

1. Write a node class in python.

Class Node:

\_\_init\_\_(self, key, data)

Methods within the class.

printNode(self) – Prints a node stats.(key, data, left, right, parent,color)

1. Write a binary tree class in python.

Class Tree

\_\_init\_\_(self)

Methods with the class.

insert(self,node) – Inserts a node into a binary tree

inOrderTree(self) – Checks the root of a tree to see if it is empty then prints.

inorderTree(self, node) – Prints a tree in order.

searchTree(self,node,k) – Searches a tree for a given node.

isEmpty(self) – Checks the root of a tree to see if it is null.

emptyTree(self) – Deletes the tree.

load(self,path) – Loads data from a file and places it into a binary tree.

createNodes(self, path) – Loads data from a file and creates a list of nodes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | Input Values | Output Values | Actual Output | Reverse Bubble Sort |
| (a) | Taz,2  brock,1  dude,3  diamond,4 | Brock, 1  Taz, 2  Dude, 3  Diamond,4 | Brock, 1  Taz, 2  Dude, 3  Diamond,4 | 0.000690 |
| (f) | load(path) path = input.dat.txt | 79,,INDIANA LOTTO  93,,treo 700w  123,,Wrsi Riversound cafe cd  161,,Dillons/Kroger Employee Coupon ($1.25 credit)  2140000070,,Rhinestone Watch  2140118461,,"""V"": Breakout/The Deception VHS Tape"  2144209103,VHS,Tintorera - Tiger Shark  2144622711,,Taxi : The Collector's Edition VHS  2147483647,,Toshiba 2805 DVD player  2158242769,288/1.12Z,GREEN SUGAR COOKIES4276  2158561631,,HOT COCOA W/BKMK  2158769549,njhjhn,gjfhjbgkj  2160500567,2.25 oz (64)g,Dollar Bar Rich Raspberry  2172307284,,Mixed seasonal flower bouquet  2177000074,,4 way 13 AMP Extension Lead (Wilkinson UK)  2184000098,21 oz,Christopher's Assorted Fruit Jellies  2187682888,,fairway | 79,,INDIANA LOTTO  93,,treo 700w  123,,Wrsi Riversound cafe cd  161,,Dillons/Kroger Employee Coupon ($1.25 credit)  2140000070,,Rhinestone Watch  2140118461,,"""V"": Breakout/The Deception VHS Tape"  2144209103,VHS,Tintorera - Tiger Shark  2144622711,,Taxi : The Collector's Edition VHS  2147483647,,Toshiba 2805 DVD player  2158242769,288/1.12Z,GREEN SUGAR COOKIES4276  2158561631,,HOT COCOA W/BKMK  2158769549,njhjhn,gjfhjbgkj  2160500567,2.25 oz (64)g,Dollar Bar Rich Raspberry  2172307284,,Mixed seasonal flower bouquet  2177000074,,4 way 13 AMP Extension Lead (Wilkinson UK)  2184000098,21 oz,Christopher's Assorted Fruit Jellies  2187682888,,fairway | 0.000832 |
| (f) | load(path) path = UPC.csv |  | 2.384816 | 13.4758 |
| (f) | load(path) path = UPCunsorted.cvs |  | 9.721682 | 0.9346 |

1. Use the a method to read the following txt files and covert them into arrays to be sorted

* UPC.csv
* UPCunsorted.csv
* Input.dat.txt

### Testing Plan

Sample nodes were created and printed (taz, brock, dude, diamond, love, joe). New instance of Tree was created to house these nodes. The nodes were then placed into the new tree with the exception of diamond. Once the nodes were placed in the tree they were then printed to see the updated values. I then tested the search node function for nodes that did not exist and nodes that were improperly keyed. Once the core functions were working I created 2 import functions to load from .csv and .txt files.

### Test Cases

The test cases are shown in the table below using a MacBook Pro 16GB, 8 core 2.3 GHz Intel Core i9:

### Analysis and Conclusions

### References

Textbook, python.org, and examples provided in the assignment.

**Screen Shots**

