FALL 2019 – CS 303 Algorithms and Data Structures
Lab 10
Deadline: November 3, 2019 Sunday 11:59pm

Notes:
• Implement the algorithm and analyze the results using the give input files
• Deliverables: Report.pdf file and your code file (please do not send a zip file. If you have more than one class in your code, then submit each file separately through Canvas.)
• Homework report must follow the guidelines provided in the sample report uploaded in Canvas

Objectives:
- Insert elements in a Red-Black Tree
- Use the Red-Black Tree in a real application

Problems
From the assignment 8, extend the BSTNode class to form a new class RBNode to add a color field and a value field to each node (let the color field be of type *boolean* and the value field be a generic type). Also extend the BinarySearchTree class to create the RBTree class and include the following methods:

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.

```
RB-INSERT(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
  RB-INSERT-FIXUP(T, z)
```

```
LEFT-ROTATE(T, x)
  y = x.right              // set y
  x.right = y.left         // turn y's left subtree into x's right subtree
  if y.left ≠ T.nil
      y.left.p = x
  y.p = x.p                // link x's parent to y
  if x.p == T.nil
      T.root = y
  elseif x == x.p.left
      x.p.left = y
  else x.p.right = y
  y.left = x               // put x on y's left
  x.p = y
```

```
RB-INSERT-FIXUP(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                  // case 1
              y.color = BLACK                    // case 1
              z.p.p.color = RED                  // case 1
              z = z.p.p                          // case 1
          else if z == z.p.right
                  z = z.p                        // case 2
                  LEFT-ROTATE(T, z)              // case 2
              z.p.color = BLACK                  // case 3
              z.p.p.color = RED                  // case 3
              RIGHT-ROTATE(T, z.p.p)             // case 3
      else (same as then clause with "right" and "left" exchanged)
  T.root.color = BLACK
```