|  |  |
| --- | --- |
| Name: | Prerna Sunil Jadhav |
| Sap Id: | 60004220127 |
| Class: | T. Y. B. Tech (Computer Engineering) |
| Course: | Advance Algorithm Laboratory |
| Course Code: | DJ19CEL602 |
| Experiment No.: | 04-A |

**AIM: Implement Red-black Tree Operations.**

**04-A) INSERTION**

**CODE:**

# RB tree insertion

class Node:

  def \_\_init\_\_(self, val, color):

    self.val = val

    self.color = color

    self.left = None

    self.right = None

    self.parent = None

class RedBlackTree:

    def \_\_init\_\_(self):

        self.root = None

    def insert(self, val):

        new\_node = Node(val, "RED")

        if not self.root:

            self.root = new\_node

            new\_node.color = "BLACK"

            return

        curr = self.root

        parent = None

        while curr:

            parent = curr

            if val < curr.val:

                curr = curr.left

            else:

                curr = curr.right

        new\_node.parent = parent

        if val < parent.val:

            parent.left = new\_node

        else:

            parent.right = new\_node

        self.\_fix\_violations(new\_node)

    def \_fix\_violations(self, node):

        while node.parent and node.parent.color == "RED":

            if node.parent == node.parent.parent.left:

                uncle = node.parent.parent.right

                if uncle and uncle.color == "RED":

                    node.parent.color, uncle.color, node.parent.parent.color = "BLACK", "BLACK", "RED"

                    node = node.parent.parent

                else:

                    if node == node.parent.right:

                        node = node.parent

                        self.\_left\_rotate(node)

                    node.parent.color, node.parent.parent.color = "BLACK", "RED"

                    self.\_right\_rotate(node.parent.parent)

            else:

                uncle = node.parent.parent.left

                if uncle and uncle.color == "RED":

                    node.parent.color, uncle.color, node.parent.parent.color = "BLACK", "BLACK", "RED"

                    node = node.parent.parent

                else:

                    if node == node.parent.left:

                        node = node.parent

                        self.\_right\_rotate(node)

                    node.parent.color, node.parent.parent.color = "BLACK", "RED"

                    self.\_left\_rotate(node.parent.parent)

        self.root.color = "BLACK"

    def \_left\_rotate(self, node):

        right\_child = node.right

        node.right = right\_child.left

        if right\_child.left:

            right\_child.left.parent = node

        right\_child.parent = node.parent

        if not node.parent:

            self.root = right\_child

        elif node == node.parent.left:

            node.parent.left = right\_child

        else:

            node.parent.right = right\_child

        right\_child.left = node

        node.parent = right\_child

    def \_right\_rotate(self, node):

        left\_child = node.left

        node.left = left\_child.right

        if left\_child.right:

            left\_child.right.parent = node

        left\_child.parent = node.parent

        if not node.parent:

            self.root = left\_child

        elif node == node.parent.right:

            node.parent.right = left\_child

        else:

            node.parent.left = left\_child

        left\_child.right = node

        node.parent = left\_child

    def inorder\_traversal(self, node):

        if node:

            self.inorder\_traversal(node.left)

            print(f"{node.val} ({node.color})", end=" ")

            self.inorder\_traversal(node.right)

# Example usage

tree = RedBlackTree()

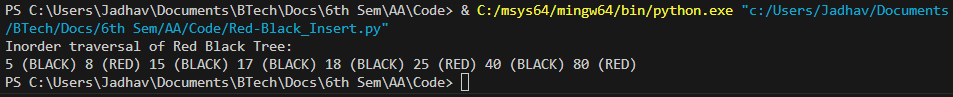
for val in [8,18,5,15,17,25,40,80]:

  tree.insert(val)

print("Inorder traversal of Red Black Tree:");

tree.inorder\_traversal(tree.root)

**OUTPUT:**

****