Практическая работа №10

Бинарное дерево поиска. AVL дерево

Автор: Николаев-Аксенов И. С.

Группа: ИКБО-20-19

Код программы:

RedBlackTree.py:

```
import sys


class Node():
    def __init__(self, data):
        self.data = data
        self.parent = None
        self.left = None
        self.right = None
        self.color = 1


class RedBlackTree():
    def __init__(self):
        self.TNULL = Node(0)
        self.TNULL.color = 0
        self.TNULL.left = None
        self.TNULL.right = None
        self.root = self.TNULL

    def __pre_order_helper(self, node):
        if node != TNULL:
            sys.stdout.write(node.data + " ")
            self.__pre_order_helper(node.left)
            self.__pre_order_helper(node.right)

    def __in_order_helper(self, node):
        if node != TNULL:
            self.__in_order_helper(node.left)
            sys.stdout.write(node.data + " ")
            self.__in_order_helper(node.right)

    def __post_order_helper(self, node):
        if node != TNULL:
            self.__post_order_helper(node.left)
            self.__post_order_helper(node.right)
            sys.stdout.write(node.data + " ")

    def __search_tree_helper(self, node, key):
        if node == TNULL or key == node.data:
            return node

        if key < node.data:
            return self.__search_tree_helper(node.left, key)
        return self.__search_tree_helper(node.right, key)

    def __fix_delete(self, x):
        while x != self.root and x.color == 0:
            if x == x.parent.left:
                s = x.parent.right
                if s.color == 1:
                    s.color = 0
                    x.parent.color = 1
                    self.left_rotate(x.parent)
```

```
                    s = x.parent.right

                if s.left.color == 0 and s.right.color == 0:
                    s.color = 1
                    x = x.parent
                else:
                    if s.right.color == 0:
                        s.left.color = 0
                        s.color = 1
                        self.right_rotate(s)
                        s = x.parent.right

                    s.color = x.parent.color
                    x.parent.color = 0
                    s.right.color = 0
                    self.left_rotate(x.parent)
                    x = self.root
            else:
                s = x.parent.left
                if s.color == 1:
                    s.color = 0
                    x.parent.color = 1
                    self.right_rotate(x.parent)
                    s = x.parent.left

                if s.left.color == 0 and s.right.color == 0:
                    s.color = 1
                    x = x.parent
                else:
                    if s.left.color == 0:
                        s.right.color = 0
                        s.color = 1
                        self.left_rotate(s)
                        s = x.parent.left

                    s.color = x.parent.color
                    x.parent.color = 0
                    s.left.color = 0
                    self.right_rotate(x.parent)
                    x = self.root
        x.color = 0

    def __rb_transplant(self, u, v):
        if u.parent == None:
            self.root = v
        elif u == u.parent.left:
            u.parent.left = v
        else:
            u.parent.right = v
        v.parent = u.parent

    def __delete_node_helper(self, node, key):
        z = self.TNULL
        while node != self.TNULL:
            if node.data == key:
                z = node

            if node.data <= key:
                node = node.right
            else:
                node = node.left

        if z == self.TNULL:
            print("Данный ключ не найден на дереве")
            return

        y = z
        y_original_color = y.color
        if z.left == self.TNULL:
```

```
            x = z.right
            self.__rb_transplant(z, z.right)
        elif (z.right == self.TNULL):
            x = z.left
            self.__rb_transplant(z, z.left)
        else:
            y = self.minimum(z.right)
            y_original_color = y.color
            x = y.right
            if y.parent == z:
                x.parent = y
            else:
                self.__rb_transplant(y, y.right)
                y.right = z.right
                y.right.parent = y

            self.__rb_transplant(z, y)
            y.left = z.left
            y.left.parent = y
            y.color = z.color
        if y_original_color == 0:
            self.__fix_delete(x)

    def __fix_insert(self, k):
        while k.parent.color == 1:
            if k.parent == k.parent.parent.right:
                u = k.parent.parent.left  # uncle
                if u.color == 1:
                    u.color = 0
                    k.parent.color = 0
                    k.parent.parent.color = 1
                    k = k.parent.parent
                else:
                    if k == k.parent.left:
                        k = k.parent
                        self.right_rotate(k)
                    k.parent.color = 0
                    k.parent.parent.color = 1
                    self.left_rotate(k.parent.parent)
            else:
                u = k.parent.parent.right

                if u.color == 1:
                    u.color = 0
                    k.parent.color = 0
                    k.parent.parent.color = 1
                    k = k.parent.parent
                else:
                    if k == k.parent.right:
                        k = k.parent
                        self.left_rotate(k)
                    k.parent.color = 0
                    k.parent.parent.color = 1
                    self.right_rotate(k.parent.parent)
            if k == self.root:
                break
        self.root.color = 0

    def __print_helper(self, node, indent, last):
        if node != self.TNULL:
            sys.stdout.write(indent)
            if last:
                sys.stdout.write("R----")
                indent += "     "
            else:
                sys.stdout.write("L----")
                indent += "|    "

            s_color = "RED" if node.color == 1 else "BLACK"
```

```
193.             print(str(node.data) + "(" + s_color + ")")
194.             self.__print_helper(node.left, indent, False)
195.             self.__print_helper(node.right, indent, True)
196.
197.     def preorder(self):
198.         self.__pre_order_helper(self.root)
199.
200.     def inorder(self):
201.         self.__in_order_helper(self.root)
202.
203.     def postorder(self):
204.         self.__post_order_helper(self.root)
205.
206.     def searchTree(self, k):
207.         return self.__search_tree_helper(self.root, k)
208.
209.     def minimum(self, node):
210.         while node.left != self.TNULL:
211.             node = node.left
212.         return node
213.
214.     def maximum(self, node):
215.         while node.right != self.TNULL:
216.             node = node.right
217.         return node
218.
219.     def successor(self, x):
220.         if x.right != self.TNULL:
221.             return self.minimum(x.right)
222.         y = x.parent
223.         while y != self.TNULL and x == y.right:
224.             x = y
225.             y = y.parent
226.         return y
227.
228.     def predecessor(self,  x):
229.         if (x.left != self.TNULL):
230.             return self.maximum(x.left)
231.
232.         y = x.parent
233.         while y != self.TNULL and x == y.left:
234.             x = y
235.             y = y.parent
236.
237.         return y
238.
239.     def left_rotate(self, x):
240.         y = x.right
241.         x.right = y.left
242.         if y.left != self.TNULL:
243.             y.left.parent = x
244.
245.         y.parent = x.parent
246.         if x.parent == None:
247.             self.root = y
248.         elif x == x.parent.left:
249.             x.parent.left = y
250.         else:
251.             x.parent.right = y
252.         y.left = x
253.         x.parent = y
254.
255.     def right_rotate(self, x):
256.         y = x.left
257.         x.left = y.right
258.         if y.right != self.TNULL:
259.             y.right.parent = x
260.
261.         y.parent = x.parent
```

```
262.             if x.parent == None:
263.                 self.root = y
264.             elif x == x.parent.right:
265.                 x.parent.right = y
266.             else:
267.                 x.parent.left = y
268.             y.right = x
269.             x.parent = y
270.
271.         def insert(self, key):
272.             node = Node(key)
273.             node.parent = None
274.             node.data = key
275.             node.left = self.TNULL
276.             node.right = self.TNULL
277.             node.color = 1
278.
279.             y = None
280.             x = self.root
281.
282.             while x != self.TNULL:
283.                 y = x
284.                 if node.data < x.data:
285.                     x = x.left
286.                 else:
287.                     x = x.right
288.             node.parent = y
289.             if y == None:
290.                 self.root = node
291.             elif node.data < y.data:
292.                 y.left = node
293.             else:
294.                 y.right = node
295.
296.             if node.parent == None:
297.                 node.color = 0
298.                 return
299.
300.             if node.parent.parent == None:
301.                 return
302.
303.             self.__fix_insert(node)
304.
305.         def get_root(self):
306.             return self.root
307.
308.         def delete_node(self, data):
309.             self.__delete_node_helper(self.root, data)
310.
311.         def pretty_print(self):
312.             self.__print_helper(self.root, "", True)
```

startRBT.py:

```
1.  from RedBlackTree import RedBlackTree
2.  rbt = RedBlackTree()
3.
4.
5.  def menu():
6.      x = int(input("\nВыберите действие с деревом:\n1 - Добавить элемент\n2 - Удалить
    элемент\n3 - Печать дерева\nВвод: "))
7.      if (x == 1):
8.          rbt.insert(int(input("Введите число для добавления его на дерево: ")))
9.          print("Число успешно добавлено на дерево!\n")
10.         menu()
11.     elif (x == 2):
12.         rbt.delete_node(
```

```
            int(input("Введите число которое вы хотите удалить: ")))
        print("Число успешно удалено из дерева!\n")
        menu()
    elif (x == 3):
        print("\nR - right, L - left\n")
        rbt.pretty_print()
        menu()
    else:
        print("Действие не найдено! Повторите ввод.")
        menu()


def main():
    numbers = list(
        map(int, input("Введите числа для добавления их на дерево: ").split()))

    for i in numbers:
        rbt.insert(i)

    menu()


if __name__ == "__main__":
    main()
```

Результат выполнения программы: