**Практическая работа №10**

**Бинарное дерево поиска. AVL дерево**

**Автор: Николаев-Аксенов И. С.**

**Группа: ИКБО-20-19**

**Код программы:**

*RedBlackTree.py:*

1. **import** sys

4. **class** Node():
5. **def** \_\_init\_\_(self, data):
6. self.data = data
7. self.parent = None
8. self.left = None
9. self.right = None
10. self.color = 1

13. **class** RedBlackTree():
14. **def** \_\_init\_\_(self):
15. self.TNULL = Node(0)
16. self.TNULL.color = 0
17. self.TNULL.left = None
18. self.TNULL.right = None
19. self.root = self.TNULL
21. **def** \_\_pre\_order\_helper(self, node):
22. **if** node != TNULL:
23. sys.stdout.write(node.data + " ")
24. self.\_\_pre\_order\_helper(node.left)
25. self.\_\_pre\_order\_helper(node.right)
27. **def** \_\_in\_order\_helper(self, node):
28. **if** node != TNULL:
29. self.\_\_in\_order\_helper(node.left)
30. sys.stdout.write(node.data + " ")
31. self.\_\_in\_order\_helper(node.right)
33. **def** \_\_post\_order\_helper(self, node):
34. **if** node != TNULL:
35. self.\_\_post\_order\_helper(node.left)
36. self.\_\_post\_order\_helper(node.right)
37. sys.stdout.write(node.data + " ")
39. **def** \_\_search\_tree\_helper(self, node, key):
40. **if** node == TNULL **or** key == node.data:
41. **return** node
43. **if** key < node.data:
44. **return** self.\_\_search\_tree\_helper(node.left, key)
45. **return** self.\_\_search\_tree\_helper(node.right, key)
47. **def** \_\_fix\_delete(self, x):
48. **while** x != self.root **and** x.color == 0:
49. **if** x == x.parent.left:
50. s = x.parent.right
51. **if** s.color == 1:
52. s.color = 0
53. x.parent.color = 1
54. self.left\_rotate(x.parent)
55. s = x.parent.right
57. **if** s.left.color == 0 **and** s.right.color == 0:
58. s.color = 1
59. x = x.parent
60. **else**:
61. **if** s.right.color == 0:
62. s.left.color = 0
63. s.color = 1
64. self.right\_rotate(s)
65. s = x.parent.right
67. s.color = x.parent.color
68. x.parent.color = 0
69. s.right.color = 0
70. self.left\_rotate(x.parent)
71. x = self.root
72. **else**:
73. s = x.parent.left
74. **if** s.color == 1:
75. s.color = 0
76. x.parent.color = 1
77. self.right\_rotate(x.parent)
78. s = x.parent.left
80. **if** s.left.color == 0 **and** s.right.color == 0:
81. s.color = 1
82. x = x.parent
83. **else**:
84. **if** s.left.color == 0:
85. s.right.color = 0
86. s.color = 1
87. self.left\_rotate(s)
88. s = x.parent.left
90. s.color = x.parent.color
91. x.parent.color = 0
92. s.left.color = 0
93. self.right\_rotate(x.parent)
94. x = self.root
95. x.color = 0
97. **def** \_\_rb\_transplant(self, u, v):
98. **if** u.parent == None:
99. self.root = v
100. **elif** u == u.parent.left:
101. u.parent.left = v
102. **else**:
103. u.parent.right = v
104. v.parent = u.parent
106. **def** \_\_delete\_node\_helper(self, node, key):
107. z = self.TNULL
108. **while** node != self.TNULL:
109. **if** node.data == key:
110. z = node
112. **if** node.data <= key:
113. node = node.right
114. **else**:
115. node = node.left
117. **if** z == self.TNULL:
118. **print**("Данный ключ не найден на дереве")
119. **return**
121. y = z
122. y\_original\_color = y.color
123. **if** z.left == self.TNULL:
124. x = z.right
125. self.\_\_rb\_transplant(z, z.right)
126. **elif** (z.right == self.TNULL):
127. x = z.left
128. self.\_\_rb\_transplant(z, z.left)
129. **else**:
130. y = self.minimum(z.right)
131. y\_original\_color = y.color
132. x = y.right
133. **if** y.parent == z:
134. x.parent = y
135. **else**:
136. self.\_\_rb\_transplant(y, y.right)
137. y.right = z.right
138. y.right.parent = y
140. self.\_\_rb\_transplant(z, y)
141. y.left = z.left
142. y.left.parent = y
143. y.color = z.color
144. **if** y\_original\_color == 0:
145. self.\_\_fix\_delete(x)
147. **def** \_\_fix\_insert(self, k):
148. **while** k.parent.color == 1:
149. **if** k.parent == k.parent.parent.right:
150. u = k.parent.parent.left  *# uncle*
151. **if** u.color == 1:
152. u.color = 0
153. k.parent.color = 0
154. k.parent.parent.color = 1
155. k = k.parent.parent
156. **else**:
157. **if** k == k.parent.left:
158. k = k.parent
159. self.right\_rotate(k)
160. k.parent.color = 0
161. k.parent.parent.color = 1
162. self.left\_rotate(k.parent.parent)
163. **else**:
164. u = k.parent.parent.right
166. **if** u.color == 1:
167. u.color = 0
168. k.parent.color = 0
169. k.parent.parent.color = 1
170. k = k.parent.parent
171. **else**:
172. **if** k == k.parent.right:
173. k = k.parent
174. self.left\_rotate(k)
175. k.parent.color = 0
176. k.parent.parent.color = 1
177. self.right\_rotate(k.parent.parent)
178. **if** k == self.root:
179. **break**
180. self.root.color = 0
182. **def** \_\_print\_helper(self, node, indent, last):
183. **if** node != self.TNULL:
184. sys.stdout.write(indent)
185. **if** last:
186. sys.stdout.write("R----")
187. indent += "     "
188. **else**:
189. sys.stdout.write("L----")
190. indent += "|    "
192. 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)
197. **def** preorder(self):
198. self.\_\_pre\_order\_helper(self.root)
200. **def** inorder(self):
201. self.\_\_in\_order\_helper(self.root)
203. **def** postorder(self):
204. self.\_\_post\_order\_helper(self.root)
206. **def** searchTree(self, k):
207. **return** self.\_\_search\_tree\_helper(self.root, k)
209. **def** minimum(self, node):
210. **while** node.left != self.TNULL:
211. node = node.left
212. **return** node
214. **def** maximum(self, node):
215. **while** node.right != self.TNULL:
216. node = node.right
217. **return** node
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
228. **def** predecessor(self,  x):
229. **if** (x.left != self.TNULL):
230. **return** self.maximum(x.left)
232. y = x.parent
233. **while** y != self.TNULL **and** x == y.left:
234. x = y
235. y = y.parent
237. **return** y
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
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
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
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
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
279. y = None
280. x = self.root
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
296. **if** node.parent == None:
297. node.color = 0
298. **return**
300. **if** node.parent.parent == None:
301. **return**
303. self.\_\_fix\_insert(node)
305. **def** get\_root(self):
306. **return** self.root
308. **def** delete\_node(self, data):
309. self.\_\_delete\_node\_helper(self.root, data)
311. **def** pretty\_print(self):
312. self.\_\_print\_helper(self.root, "", True)

*startRBT.py:*

1. **from** RedBlackTree **import** RedBlackTree
2. rbt = RedBlackTree()

5. **def** menu():
6. x = int(input("**\n**Выберите действие с деревом:**\n**1 - Добавить элемент**\n**2 - Удалить элемент**\n**3 - Печать дерева**\n**Ввод: "))
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(
13. int(input("Введите число которое вы хотите удалить: ")))
14. **print**("Число успешно удалено из дерева!**\n**")
15. menu()
16. **elif** (x == 3):
17. **print**("**\n**R - right, L - left**\n**")
18. rbt.pretty\_print()
19. menu()
20. **else**:
21. **print**("Действие не найдено! Повторите ввод.")
22. menu()

25. **def** main():
26. numbers = list(
27. map(int, input("Введите числа для добавления их на дерево: ").split()))
29. **for** i **in** numbers:
30. rbt.insert(i)
32. menu()

35. **if** \_\_name\_\_ == "\_\_main\_\_":
36. main()

**Результат выполнения программы:**

