**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 3 з дисципліни

«Проектування алгоритмів»

„ **Проектування структур даних**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-21 Скрипець Ольга*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2023

Зміст

[1 Мета лабораторної роботи 3](#_Toc114359761)

[2 Завдання 4](#_Toc114359762)

[3 Виконання 7](#_Toc114359763)

[3.1 Псевдокод алгоритмів 7](#_Toc114359764)

[3.2 Часова складність пошуку 7](#_Toc114359765)

[3.3 Програмна реалізація 7](#_Toc114359766)

[3.3.1 Вихідний код 7](#_Toc114359767)

[3.3.2 Приклади роботи 7](#_Toc114359768)

[3.4 Тестування алгоритму 8](#_Toc114359769)

[3.4.1 Часові характеристики оцінювання 8](#_Toc114359770)

[Висновок 9](#_Toc114359771)

[Критерії оцінювання 10](#_Toc114359772)

# Мета лабораторної роботи

Мета роботи – вивчити основні підходи проектування та обробки складних структур даних.

# Завдання

Відповідно до варіанту (таблиця 2.1), записати алгоритми пошуку, додавання, видалення і редагування запису в структурі даних за допомогою псевдокоду (чи іншого способу по вибору).

Записати часову складність пошуку в структурі в асимптотичних оцінках.

Виконати програмну реалізацію невеликої СУБД з графічним (не консольним) інтерфейсом користувача (дані БД мають зберігатися на ПЗП), з функціями пошуку (алгоритм пошуку у вузлі структури згідно варіанту таблиця 2.1, за необхідності), додавання, видалення та редагування записів (запис складається із ключа і даних, ключі унікальні і цілочисельні, даних може бути декілька полів для одного ключа, але достатньо одного рядка фіксованої довжини). Для зберігання даних використовувати структуру даних згідно варіанту (таблиця 2.1).

Заповнити базу випадковими значеннями до 10000 і зафіксувати середнє (із 10-15 пошуків) число порівнянь для знаходження запису по ключу.

Зробити висновок з лабораторної роботи.

Таблиця 2.1 – Варіанти алгоритмів

|  |  |
| --- | --- |
| **№** | **Структура даних** |
| 1 | Файли з щільним індексом з перебудовою індексної області, бінарний пошук |
| 2 | Файли з щільним індексом з областю переповнення, бінарний пошук |
| 3 | Файли з не щільним індексом з перебудовою індексної області, бінарний пошук |
| 4 | Файли з не щільним індексом з областю переповнення, бінарний пошук |
| 5 | АВЛ-дерево |
| 6 | Червоно-чорне дерево |
| 7 | B-дерево t=10, бінарний пошук |
| 8 | B-дерево t=25, бінарний пошук |
| 9 | B-дерево t=50, бінарний пошук |
| 10 | B-дерево t=100, бінарний пошук |
| 11 | Файли з щільним індексом з перебудовою індексної області, однорідний бінарний пошук |
| 12 | Файли з щільним індексом з областю переповнення, однорідний бінарний пошук |
| 13 | Файли з не щільним індексом з перебудовою індексної області, однорідний бінарний пошук |
| 14 | Файли з не щільним індексом з областю переповнення, однорідний бінарний пошук |
| 15 | АВЛ-дерево |
| 16 | Червоно-чорне дерево |
| 17 | B-дерево t=10, однорідний бінарний пошук |
| 18 | B-дерево t=25, однорідний бінарний пошук |
| 19 | B-дерево t=50, однорідний бінарний пошук |
| 20 | B-дерево t=100, однорідний бінарний пошук |
| 21 | Файли з щільним індексом з перебудовою індексної області, метод Шарра |
| 22 | Файли з щільним індексом з областю переповнення, метод Шарра |
| 23 | Файли з не щільним індексом з перебудовою індексної області, метод Шарра |
| 24 | Файли з не щільним індексом з областю переповнення, метод Шарра |
| 25 | АВЛ-дерево |
| 26 | Червоно-чорне дерево |
| 27 | B-дерево t=10, метод Шарра |
| 28 | B-дерево t=25, метод Шарра |
| 29 | B-дерево t=50, метод Шарра |
| 30 | B-дерево t=100, метод Шарра |
| 31 | АВЛ-дерево |
| 32 | Червоно-чорне дерево |
| 33 | B-дерево t=250, бінарний пошук |
| 34 | B-дерево t=250, однорідний бінарний пошук |
| 35 | B-дерево t=250, метод Шарра |

# Виконання

## Псевдокод алгоритмів

**Клас** Node():  
 **функція** \_\_init\_\_(self, key, data):  
 self.key = key  
 self.data = data  
 self.parent = None  
 self.left = None  
 self.right = None  
 self.color = 1

**Клас** RedBlackTree():  
 number\_comparison = 0  
 **функція** \_\_init\_\_(self):  
 self.TNULL = Node(0, None)  
 self.TNULL.color = 0  
 self.TNULL.left = None  
 self.TNULL.right = None  
 self.root = self.TNULL  
  
 **функція** pre\_order\_helper(self, node):  
 **якщо** node != self.TNULL:  
 sys.stdout.write(node.key + " ")  
 self.pre\_order\_helper(node.left)  
 self.pre\_order\_helper(node.right)  
  
 **функція** in\_order\_helper(self, node):  
 **якщо** node != self.TNULL:  
 self.in\_order\_helper(node.left)  
 sys.stdout.write(node.key + " ")  
 self.in\_order\_helper(node.right)  
  
 **функція** post\_order\_helper(self, node):  
 **якщо** node != self.TNULL:  
 self.post\_order\_helper(node.left)  
 self.post\_order\_helper(node.right)  
 sys.stdout.write(node.key + " ")  
  
 **функція** search\_tree\_helper(self, node, key):  
 self.number\_comparison += 1  
 **якщо** node == self.TNULL or key == node.key:  
 **повернути** node  
 **якщо** key < node.key:  
 **повернути** self.search\_tree\_helper(node.left, key)  
 **повернути** self.search\_tree\_helper(node.right, key)  
  
 # Balancing the tree after deletion  
 **функція** delete\_fix(self, x):  
 **коли** x != self.root **і** x.color == 0:  
 **якщо** x == x.parent.left:  
 s = x.parent.right  
 **якщо** s.color == 1:  
 s.color = 0  
 x.parent.color = 1  
 self.left\_rotate(x.parent)  
 s = x.parent.right  
  
 **якщо** s.left.color == 0 **і** s.right.color == 0:  
 s.color = 1  
 x = x.parent  
 **інакше**:  
 **якщо** 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  
 **інакше**:  
 s = x.parent.left  
 **якщо** s.color == 1:  
 s.color = 0  
 x.parent.color = 1  
 self.right\_rotate(x.parent)  
 s = x.parent.left  
 **якщо** s.right.color == 0 and s.right.color == 0:  
 s.color = 1  
 x = x.parent  
 **інакше**:  
 **якщо** 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  
  
 **функція** \_\_rb\_transplant(self, u, v):  
 **якщо** u.parent == None:  
 self.root = v  
 **інакшеякщо** u == u.parent.left:  
 u.parent.left = v  
 **інакше**:  
 u.parent.right = v  
 v.parent = u.parent  
  
 # Node deletion  
 **функція** delete\_node\_helper(self, node, key):  
 z = self.TNULL  
 **коли** node != self.TNULL:  
 **якщо** node.key == key:  
 z = node  
 **якщо** node.key <= key:  
 node = node.right  
 **інакше**:  
 node = node.left  
 **якщо** z == self.TNULL:  
 print("Cannot find key in the tree")  
 **повернути**  
  
 y = z  
 y\_original\_color = y.color  
 **якщо** z.left == self.TNULL:  
 x = z.right  
 self.\_\_rb\_transplant(z, z.right)  
 **інакшеякщо** (z.right == self.TNULL):  
 x = z.left  
 self.\_\_rb\_transplant(z, z.left)  
 **інакше**:  
 y = self.minimum(z.right)  
 y\_original\_color = y.color  
 x = y.right  
 **якщо** y.parent == z:  
 x.parent = y  
 **інакше**:  
 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  
 **якщо** y\_original\_color == 0:  
 self.delete\_fix(x)  
  
 # Balance the tree after insertion  
 **функція** fix\_insert(self, k):  
 **коли** k.parent.color == 1:  
 **якщо** k.parent == k.parent.parent.right:  
 u = k.parent.parent.left  
 **якщо** u.color == 1:  
 u.color = 0  
 k.parent.color = 0  
 k.parent.parent.color = 1  
 k = k.parent.parent  
 **інакше**:  
 **якщо** 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)  
 **інакше**:  
 u = k.parent.parent.right  
 **якщо** u.color == 1:  
 u.color = 0  
 k.parent.color = 0  
 k.parent.parent.color = 1  
 k = k.parent.parent  
 **інакше**:  
 **якщо** 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)  
 **якщо** k == self.root:  
 **зупинти**  
 self.root.color = 0  
  
 # Printing the tree  
 **функція** \_\_print\_helper(self, node, indent, last):  
 **якщо** node != self.TNULL:  
 sys.stdout.write(indent)  
 **якщо** last:  
 sys.stdout.write("R----")  
 indent += " "  
 **інакше**:  
 sys.stdout.write("L----")  
 indent += "| "  
 s\_color = "RED" if node.color == 1 else "BLACK"  
 print(f"{node.key}({s\_color}): {node.data}")  
 self.\_\_print\_helper(node.left, indent, False)  
 self.\_\_print\_helper(node.right, indent, True)  
  
 **функція** preorder(self):  
 self.pre\_order\_helper(self.root)  
  
 **функція** inorder(self):  
 self.in\_order\_helper(self.root)  
  
 **функція** postorder(self):  
 self.post\_order\_helper(self.root)  
  
 **функція** searchTree(self, key):  
 self.number\_comparison =0  
 **повернути** self.search\_tree\_helper(self.root, key)  
  
 **функція** minimum(self, node):  
 **коли** node.left != self.TNULL:  
 node = node.left  
 **повернути** node  
  
 **функція** maximum(self, node):  
 **коли** node.right != self.TNULL:  
 node = node.right  
 **повернути** node  
  
 **функція** successor(self, x):  
 **якщо** x.right != self.TNULL:  
 **повернути** self.minimum(x.right)  
 y = x.parent  
 **коли** y != self.TNULL and x == y.right:  
 x = y  
 y = y.parent  
 **повернути** y  
  
 **функція** predecessor(self, x):  
 **якщо** (x.left != self.TNULL):  
 **повернути** self.maximum(x.left)  
 y = x.parent  
 **коли** y != self.TNULL and x == y.left:  
 x = y  
 y = y.parent  
 **повернути** y  
  
 **функція** left\_rotate(self, x):  
 y = x.right  
 x.right = y.left  
 **якщо** y.left != self.TNULL:  
 y.left.parent = x  
 y.parent = x.parent  
 **якщо** x.parent == None:  
 self.root = y  
 **інакшеякщо** x == x.parent.left:  
 x.parent.left = y  
 **інакше**:  
 x.parent.right = y  
 y.left = x  
 x.parent = y  
  
 **функція** right\_rotate(self, x):  
 y = x.left  
 x.left = y.right  
 **якщо** y.right != self.TNULL:  
 y.right.parent = x  
 y.parent = x.parent  
 **якщо** x.parent == None:  
 self.root = y  
 **інакшеякщо** x == x.parent.right:  
 x.parent.right = y  
 **інакше**:  
 x.parent.left = y  
 y.right = x  
 x.parent = y  
  
 **функція** insert(self, key, data):  
 node = Node(key, data)  
 node.parent = None  
 node.left = self.TNULL  
 node.right = self.TNULL  
 node.color = 1  
 is\_insert = True  
 y = None  
 x = self.root  
 **якщо** self.searchTree(key) != self.TNULL:  
 **повернути** node, False  
 **коли** x != self.TNULL:  
 y = x  
 **якщо** node.key < x.key:  
 x = x.left  
 **інакше**:  
 x = x.right  
 node.parent = y  
 **якщо** y == None:  
 self.root = node  
 **інакшеякщо** node.key < y.key:  
 y.left = node  
 **інакше**:  
 y.right = node  
 **якщо** node.parent == None:  
 node.color = 0  
 **повернути** node, is\_insert  
 **якщо** node.parent.parent == None:  
 **повернути** node, is\_insert  
 self.fix\_insert(node)  
 **повернути** node, is\_insert  
  
 **функція** get\_root(self):  
 **повернути** self.root  
  
 **функція** delete\_node(self, key):  
 node = self.searchTree(key)  
 self.delete\_node\_helper(self.root, key)  
 **повернути** node.data  
  
 **функція** print\_tree(self):  
 self.\_\_print\_helper(self.root, "", True)  
  
 **функція** update\_node\_data(self, key, new\_data):  
 **якщо** self.searchTree(key) == self.TNULL:  
 **повернути** False  
 self.delete\_node(key)  
 self.insert(key, new\_data)  
 **повернути** True  
  
 **функція** write\_data(self, file\_path):  
 with open(file\_path, "w") as file:  
 self.traversal(self.root, file)  
 **повернути**  
  
 **функція** traversal(self, node, file):  
 **якщо** node == self.TNULL:  
 **повернути**  
 file.write(str(node.key) + " " + node.data + "\n")  
 self.traversal(node.left, file)  
 self.traversal(node.right, file)  
 **повернути**  
  
  
 **функція** read\_tree\_from\_file(self, file\_path):  
 rb\_tree = RedBlackTree()  
 with open(file\_path, 'r') as file:  
 **для** line in file:  
 key, data = map(str.strip, line.split(' ', 1))  
 key = int(key)  
 rb\_tree.insert(key, data)

**повернути** rb\_tree

## Часова складність пошуку

Асимптотична складність алгоритмів червоно-чорних дерев визначається їх конструкцією та правилами балансування. Пошук в червоно-чорному дереві має складність O(log n).

Через те, що дерево є бінарним деревом пошуку і балансованим, час пошуку визначається глибиною дерева. Цей алгоритм забезпечує баланс, що гарантує стабільну асимптотичну складність операцій навіть у найгіршому випадку.

## Програмна реалізація

### Вихідний код

**Rbtree.py**

import sys  
  
  
class Node():  
 def \_\_init\_\_(self, key, data):  
 self.key = key  
 self.data = data  
 self.parent = None  
 self.left = None  
 self.right = None  
 self.color = 1  
  
  
class RedBlackTree():  
 number\_comparison = 0  
 def \_\_init\_\_(self):  
 self.TNULL = Node(0, None)  
 self.TNULL.color = 0  
 self.TNULL.left = None  
 self.TNULL.right = None  
 self.root = self.TNULL  
  
 def pre\_order\_helper(self, node):  
 if node != self.TNULL:  
 sys.stdout.write(node.key + " ")  
 self.pre\_order\_helper(node.left)  
 self.pre\_order\_helper(node.right)  
  
 def in\_order\_helper(self, node):  
 if node != self.TNULL:  
 self.in\_order\_helper(node.left)  
 sys.stdout.write(node.key + " ")  
 self.in\_order\_helper(node.right)  
  
  
 def post\_order\_helper(self, node):  
 if node != self.TNULL:  
 self.post\_order\_helper(node.left)  
 self.post\_order\_helper(node.right)  
 sys.stdout.write(node.key + " ")  
  
  
 def search\_tree\_helper(self, node, key):  
 self.number\_comparison += 1  
 if node == self.TNULL or key == node.key:  
 return node  
  
 if key < node.key:  
 return self.search\_tree\_helper(node.left, key)  
 return self.search\_tree\_helper(node.right, key)  
  
 # Balancing the tree after deletion  
 def delete\_fix(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.right.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  
  
 # Node deletion  
 def delete\_node\_helper(self, node, key):  
 z = self.TNULL  
 while node != self.TNULL:  
 if node.key == key:  
 z = node  
  
 if node.key <= key:  
 node = node.right  
 else:  
 node = node.left  
  
 if z == self.TNULL:  
 print("Cannot find key in the tree")  
 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.delete\_fix(x)  
  
 # Balance the tree after insertion  
 def fix\_insert(self, k):  
 while k.parent.color == 1:  
 if k.parent == k.parent.parent.right:  
 u = k.parent.parent.left  
 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  
  
 # Printing the tree  
 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"  
 print(f"{node.key}({s\_color}): {node.data}")  
 self.\_\_print\_helper(node.left, indent, False)  
 self.\_\_print\_helper(node.right, indent, True)  
  
 def preorder(self):  
 self.pre\_order\_helper(self.root)  
  
 def inorder(self):  
 self.in\_order\_helper(self.root)  
  
 def postorder(self):  
 self.post\_order\_helper(self.root)  
  
 def searchTree(self, key):  
 self.number\_comparison =0  
 return self.search\_tree\_helper(self.root, key)  
  
 def minimum(self, node):  
 while node.left != self.TNULL:  
 node = node.left  
 return node  
  
 def maximum(self, node):  
 while node.right != self.TNULL:  
 node = node.right  
 return node  
  
 def successor(self, x):  
 if x.right != self.TNULL:  
 return self.minimum(x.right)  
  
 y = x.parent  
 while y != self.TNULL and x == y.right:  
 x = y  
 y = y.parent  
 return y  
  
 def predecessor(self, x):  
 if (x.left != self.TNULL):  
 return self.maximum(x.left)  
  
 y = x.parent  
 while y != self.TNULL and x == y.left:  
 x = y  
 y = y.parent  
  
 return y  
  
 def left\_rotate(self, x):  
 y = x.right  
 x.right = y.left  
 if y.left != self.TNULL:  
 y.left.parent = x  
  
 y.parent = x.parent  
 if x.parent == None:  
 self.root = y  
 elif x == x.parent.left:  
 x.parent.left = y  
 else:  
 x.parent.right = y  
 y.left = x  
 x.parent = y  
  
 def right\_rotate(self, x):  
 y = x.left  
 x.left = y.right  
 if y.right != self.TNULL:  
 y.right.parent = x  
  
 y.parent = x.parent  
 if x.parent == None:  
 self.root = y  
 elif x == x.parent.right:  
 x.parent.right = y  
 else:  
 x.parent.left = y  
 y.right = x  
 x.parent = y  
  
 def insert(self, key, data):  
 node = Node(key, data)  
 node.parent = None  
 node.left = self.TNULL  
 node.right = self.TNULL  
 node.color = 1  
 is\_insert = True  
 y = None  
 x = self.root  
 if self.searchTree(key) != self.TNULL:  
 return node, False  
 while x != self.TNULL:  
 y = x  
 if node.key < x.key:  
 x = x.left  
 else:  
 x = x.right  
  
 node.parent = y  
 if y == None:  
 self.root = node  
 elif node.key < y.key:  
 y.left = node  
 else:  
 y.right = node  
  
 if node.parent == None:  
 node.color = 0  
 return node, is\_insert  
  
 if node.parent.parent == None:  
 return node, is\_insert  
 self.fix\_insert(node)  
 return node, is\_insert  
  
 def get\_root(self):  
 return self.root  
  
 def delete\_node(self, key):  
 node = self.searchTree(key)  
 self.delete\_node\_helper(self.root, key)  
 return node.data  
  
 def print\_tree(self):  
 self.\_\_print\_helper(self.root, "", True)  
  
 def update\_node\_data(self, key, new\_data):  
 if self.searchTree(key) == self.TNULL:  
 return False  
 self.delete\_node(key)  
 self.insert(key, new\_data)  
 return True  
  
 def write\_data(self, file\_path):  
 with open(file\_path, "w") as file:  
 self.traversal(self.root, file)  
 return  
  
 #Обхід дерева і запис даних у файл  
 def traversal(self, node, file):  
 if node == self.TNULL:  
 return  
 file.write(str(node.key) + " " + node.data + "\n")  
 self.traversal(node.left, file)  
 self.traversal(node.right, file)  
 return  
  
  
 def read\_tree\_from\_file(self, file\_path):  
 rb\_tree = RedBlackTree()  
 with open(file\_path, 'r') as file:  
 for line in file:  
 try:  
 key, data = map(str.strip, line.split(' ', 1))  
 key = int(key)  
 rb\_tree.insert(key, data)  
 except ValueError:  
 print(f"Error reading line: {line}. Skipping.")  
  
 return rb\_tree

**interface.py**

import tkinter as tk  
import tkinter.filedialog  
import tkinter.messagebox  
from rbtree import RedBlackTree  
  
  
class Window1:  
 #Вікно з функціями додавання, пошуку, видалення та зміни запису  
 def \_\_init\_\_(self, master):  
 self.master = master  
 self.master.title("СУБД")  
 self.master.geometry("300x200")  
 self.master.resizable(width=False, height=False)  
 self.master.iconbitmap("logo.ico")  
 self.Tree = None  
 self.Tree = RedBlackTree()  
 filepath="D:\programming\Java\pa-skrypets-olh\src\lab3\database.txt"  
  
 self.app = None  
 try:  
 self.Tree = RedBlackTree.read\_tree\_from\_file(self.Tree, filepath)  
 self.root = None  
 self.root = self.Tree.get\_root()  
 self.filepath = filepath  
  
 self.master.title("СУБД")  
 self.master.geometry("595x370")  
 self.master.resizable(width=False, height=False)  
 self.master.iconbitmap("logo.ico")  
 self.master.configure(bg="#FFC0CB")  
  
 self.label = tk.Label(self.master,text="Оберіть дію:", background="#FFC0CB", height=2, font=("Arial, 15"))  
 self.label.pack(pady=5)  
 self.button3 = tk.Button(self.master, text="Додати запис", font=("Arial, 10"), width=30, height=4, background="#FF69B4", command=self.insert\_record)  
 self.button3.place(x=15, y=70)  
 self.button2 = tk.Button(self.master, text="Знайти запис", font=("Arial, 10"), width=30, height=4, background="#FF69B4", command=self.search\_record)  
 self.button2.place(x=320, y=70)  
 self.button4 = tk.Button(self.master, text="Видалити запис", font=("Arial, 10"), width=30, height=4, background="#FF69B4", command=self.delete\_record)  
 self.button4.place(x=15, y=170)  
 self.button5 = tk.Button(self.master, text="Змінити запис", font=("Arial, 10"), width=30, height=4, background="#FF69B4", command=self.change\_record)  
 self.button5.place(x=320, y=170)  
 self.button6 = tk.Button(self.master, text="Зберегти", font=("Arial, 10"), width=68, height=4, background="#FF69B4", command=self.save\_DB)  
 self.button6.place(x=13, y=270)  
  
 self.master.protocol("WM\_DELETE\_WINDOW", self.close\_window)  
 except ValueError:  
 tkinter.messagebox.showerror(title="СУБД", message="Помилка при зчитуванні файлу!")  
 return  
  
 def close\_window(self):  
 if self.app:  
 self.Tree, self.root = self.app.get\_tree\_root()  
 is\_save = True  
 if tkinter.messagebox.askyesno(title="СУБД", message="Зберегти базу даних?"):  
 is\_save = self.save\_DB()  
 if is\_save:  
 self.master.destroy()  
 return  
  
 def save\_DB(self):  
 is\_save = True  
 self.root = self.Tree.get\_root()  
 tkinter.messagebox.showinfo(title="Збереження файлу", message="Файл успішно збережений!")  
 if self.filepath:  
 self.Tree.write\_data(self.filepath)  
 else:  
 self.filepath = tk.filedialog.asksaveasfile(title="Зберегти базу даних", initialdir="D:\programming\lab3\_pa", initialfile = "Untitled.txt", defaultextension=".txt", filetypes=[("Бази даних","\*.txt")])  
 if not self.filepath:  
 tkinter.messagebox.showerror(title="СУБД", message="Файл не вибрано!")  
 is\_save = False  
 tkinter.messagebox.showinfo(title="Збереження файлу", message="Файл успішно збережений!")  
 else:  
 self.filepath = self.filepath.name  
 self.Tree.write\_data(self.root, self.filepath)  
 return is\_save  
  
 def search\_record(self):  
 #Пошук даних за ключем  
 if self.app:  
 self.Tree, self.root = self.app.get\_tree\_root()  
 self.master.withdraw()  
 self.newWindow = tk.Toplevel(self.master)  
 self.app = Search\_record(self.newWindow, self.master, self.Tree, self.root)  
 return  
  
 def insert\_record(self):  
 #Додавання даних за ключем  
 if self.app:  
 self.Tree, self.root = self.app.get\_tree\_root()  
 self.master.withdraw()  
 self.newWindow = tk.Toplevel(self.master)  
 self.app = Insert\_record(self.newWindow, self.master, self.Tree, self.root)  
 return  
  
 def delete\_record(self):  
 #Видалення даних за ключем  
 if self.app:  
 self.Tree, self.root = self.app.get\_tree\_root()  
 self.master.withdraw()  
 self.newWindow = tk.Toplevel(self.master)  
 self.app = Delete\_record(self.newWindow, self.master, self.Tree, self.root)  
 return  
  
 def change\_record(self):  
 #Зміна даних за ключем  
 if self.app:  
 self.Tree, self.root = self.app.get\_tree\_root()  
 self.master.withdraw()  
 self.newWindow = tk.Toplevel(self.master)  
 self.app = Change\_record(self.newWindow, self.master, self.Tree, self.root)  
 return  
  
class Search\_record():  
 def \_\_init\_\_(self, master, previous\_window, Tree, root):  
 self.previous\_window = previous\_window  
 self.Tree = Tree  
 self.root = root  
  
 self.master = master  
 self.master.title("Знайти запис")  
 self.master.geometry("300x105")  
 self.master.resizable(width=False, height=False)  
 self.master.iconbitmap("icons/logo.ico")  
 self.master.configure(bg="#FFC0CB")  
  
 self.label = tk.Label(self.master, text="Введіть ключ:", background="#FFC0CB", height=1, width=15, font=("Arial, 10"), justify=tk.CENTER)  
 self.label.place(x=15, y=15)  
 self.entry = tk.Entry(self.master, font=("Arial, 10"), justify=tk.CENTER, width=18)  
 self.entry.place(x=155, y=15)  
 self.button1 = tk.Button(self.master, text="Знайти запис", font=("Arial, 10"), width=15, height=2, background="#FF69B4", command=self.search\_record)  
 self.button1.place(x=13, y=50)  
 self.button2 = tk.Button(self.master, text="Назад", font=("Arial, 10"), width=15, height=2, background="#FF69B4", command=self.exit)  
 self.button2.place(x=155, y=50)  
  
 self.master.protocol("WM\_DELETE\_WINDOW", self.exit)  
 return  
  
 def search\_record(self):  
 if not self.entry.get():  
 tkinter.messagebox.showerror(title="Знайти запис", message="Ключ не введений!")  
 else:  
 try:  
 key = int(self.entry.get())  
 data = self.Tree.searchTree(key).data  
 message = ("Запис знайдено!\nКлюч: " + str(key) + "\nДані: " + str(data)) if data else "Даних за ключем " + str(key) + " не знайдено!"  
 tkinter.messagebox.showinfo(title="Знайти запис", message=message)  
  
 print(f"Число порівнянь для знаходження запису за ключем {key} = {self.Tree.number\_comparison}")  
 self.Tree.number\_comparison = 0  
 except ValueError:  
 tkinter.messagebox.showerror(title="Знайти запис", message="Ключ повинен бути цілим числом!")  
 return  
  
 def exit(self):  
 self.previous\_window.deiconify()  
 self.master.destroy()  
 return  
  
 def get\_tree\_root(self):  
 return self.Tree, self.root  
  
class Insert\_record():  
 def \_\_init\_\_(self, master, previous\_window, Tree, root):  
 self.previous\_window = previous\_window  
 self.Tree = Tree  
 self.root = root  
  
 self.master = master  
 self.master.title("Додати запис")  
 self.master.geometry("300x140")  
 self.master.resizable(width=False, height=False)  
 self.master.iconbitmap("icons/logo.ico")  
 self.master.configure(bg="#FFC0CB")  
  
 self.label1 = tk.Label(self.master, text="Введіть ключ:", background="#FFC0CB", height=1, width=15, font=("Arial, 10"), justify=tk.CENTER)  
 self.label1.place(x=15, y=15)  
 self.entry1 = tk.Entry(self.master, font=("Arial, 10"), justify=tk.CENTER, width=18)  
 self.entry1.place(x=155, y=15)  
 self.label2 = tk.Label(self.master, text="Введіть дані:", background="#FFC0CB", height=1, width=15, font=("Arial, 10"), justify=tk.CENTER)  
 self.label2.place(x=15, y=50)  
 self.entry2 = tk.Entry(self.master, font=("Arial, 10"), justify=tk.CENTER, width=18)  
 self.entry2.place(x=155, y=50)  
 self.button1 = tk.Button(self.master, text="Вставити запис", font=("Arial, 10"), width=15, height=2, background="#FF69B4", command=self.insert\_record)  
 self.button1.place(x=13, y=85)  
 self.button2 = tk.Button(self.master, text="Назад", font=("Arial, 10"), width=15, height=2, background="#FF69B4", command=self.exit)  
 self.button2.place(x=155, y=85)  
  
 self.master.protocol("WM\_DELETE\_WINDOW", self.exit)  
 return  
  
 def insert\_record(self):  
 if not self.entry1.get():  
 tkinter.messagebox.showerror(title="Вставити запис", message="Ключ не введений!")  
 elif not self.entry2.get():  
 tkinter.messagebox.showerror(title="Вставити запис", message="Дані не введені!")  
 else:  
 try:  
 key = int(self.entry1.get())  
 data = self.entry2.get()  
 node, is\_insert = self.Tree.insert(key, data)  
 message = ("Запис вставлено!\nКлюч: " + str(key) + "\nДані: " + str(data)) if is\_insert else "Ключ " + str(key) + " вже є у базі даних!"  
 tkinter.messagebox.showinfo(title="Вставити запис", message=message)  
 except ValueError:  
 tkinter.messagebox.showerror(title="Вставити запис", message="Ключ повинен бути цілим числом!")  
 return  
  
 def exit(self):  
 self.previous\_window.deiconify()  
 self.master.destroy()  
 return  
  
 def get\_tree\_root(self):  
 return self.Tree, self.root  
  
class Delete\_record():  
 def \_\_init\_\_(self, master, previous\_window, Tree, root):  
 self.previous\_window = previous\_window  
 self.Tree = Tree  
 self.root = root  
  
 self.master = master  
 self.master.title("Видалити запис")  
 self.master.geometry("300x105")  
 self.master.resizable(width=False, height=False)  
 self.master.iconbitmap("icons/logo.ico")  
 self.master.configure(bg="#FFC0CB")  
  
 self.label = tk.Label(self.master, text="Введіть ключ:", background="#FFC0CB", height=1, width=15, font=("Arial, 10"), justify=tk.CENTER)  
 self.label.place(x=15, y=15)  
 self.entry = tk.Entry(self.master, font=("Arial, 10"), justify=tk.CENTER, width=18)  
 self.entry.place(x=155, y=15)  
 self.button1 = tk.Button(self.master, text="Видалити запис", font=("Arial, 10"), width=15, height=2, background="#FF69B4", command=self.delete\_record)  
 self.button1.place(x=13, y=50)  
 self.button2 = tk.Button(self.master, text="Назад", font=("Arial, 10"), width=15, height=2, background="#FF69B4", command=self.exit)  
 self.button2.place(x=155, y=50)  
  
 self.master.protocol("WM\_DELETE\_WINDOW", self.exit)  
 return  
  
 def delete\_record(self):  
 if not self.entry.get():  
 tkinter.messagebox.showerror(title="Видалення запису", message="Ключ не введений!")  
 else:  
 try:  
 key = int(self.entry.get())  
 data = self.Tree.delete\_node(key)  
 message = ("Запис видалено!\nКлюч: " + str(key) + "\nДані: " + str(data)) if data else "Даних за ключем " + str(key) + " немає!"  
 tkinter.messagebox.showinfo(title="Видалення запису", message=message)  
 except ValueError:  
 tkinter.messagebox.showerror(title="Видалення запису", message="Ключ повинен бути цілим числом!")  
 return  
  
 def exit(self):  
 self.previous\_window.deiconify()  
 self.master.destroy()  
 return  
  
 def get\_tree\_root(self):  
 return self.Tree, self.root  
  
class Change\_record():  
 def \_\_init\_\_(self, master, previous\_window, Tree, root):  
 self.previous\_window = previous\_window  
 self.Tree = Tree  
 self.root = root  
  
 self.master = master  
 self.master.title("Змінити запис")  
 self.master.geometry("300x140")  
 self.master.resizable(width=False, height=False)  
 self.master.iconbitmap("icons/logo.ico")  
 self.master.configure(bg="#FFC0CB")  
  
 self.label1 = tk.Label(self.master, text="Введіть ключ:", background="#FFC0CB", height=1, width=15, font=("Arial, 10"), justify=tk.CENTER)  
 self.label1.place(x=15, y=15)  
 self.entry1 = tk.Entry(self.master, font=("Arial, 10"), justify=tk.CENTER, width=18)  
 self.entry1.place(x=155, y=15)  
 self.label2 = tk.Label(self.master, text="Введіть нові дані:", background="#FFC0CB", height=1, width=15, font=("Arial, 10"), justify=tk.CENTER)  
 self.label2.place(x=15, y=50)  
 self.entry2 = tk.Entry(self.master, font=("Arial, 10"), justify=tk.CENTER, width=18)  
 self.entry2.place(x=155, y=50)  
 self.button1 = tk.Button(self.master, text="Змінити запис", font=("Arial, 10"), width=15, height=2, background="#FF69B4", command=self.change\_record)  
 self.button1.place(x=13, y=85)  
 self.button2 = tk.Button(self.master, text="Назад", font=("Arial, 10"), width=15, height=2, background="#FF69B4", command=self.exit)  
 self.button2.place(x=155, y=85)  
  
 self.master.protocol("WM\_DELETE\_WINDOW", self.exit)  
 return  
  
 def change\_record(self):  
 if not self.entry1.get():  
 tkinter.messagebox.showerror(title="Змінити запис", message="Ключ не введений!")  
 elif not self.entry2.get():  
 tkinter.messagebox.showerror(title="Змінити запис", message="Нові дані не введені!")  
 else:  
 try:  
 key = int(self.entry1.get())  
 new\_data = self.entry2.get()  
 is\_change = self.Tree.update\_node\_data(key, new\_data)  
 message = ("Запис змінено!\nКлюч: " + str(key) + "\nНові дані: " + str(new\_data)) if is\_change else "Ключа " + str(key) + " немає в базі даних!"  
 tkinter.messagebox.showinfo(title="Змінити запис", message=message)  
 except ValueError:  
 tkinter.messagebox.showerror(title="Змінити запис", message="Ключ повинен бути цілим числом!")  
 return  
  
 def exit(self):  
 self.previous\_window.deiconify()  
 self.master.destroy()  
 return  
  
 def get\_tree\_root(self):  
 return self.Tree, self.root  
  
def main():  
 window = tk.Tk()  
 app = Window1(window)  
 window.mainloop()  
 return  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

### Приклади роботи

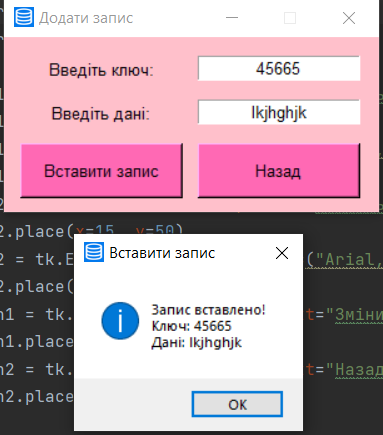
На рисунках 3.1 і 3.2 показані приклади роботи програми для додавання і пошуку запису.

Рисунок 3.1 –Додавання запису

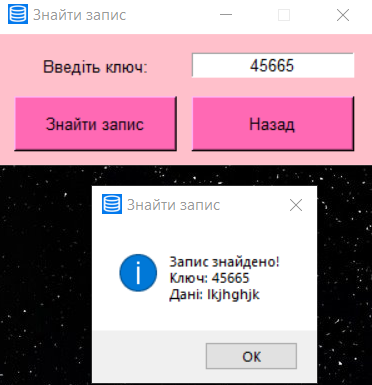


Рисунок 3.2 – Пошук запису

## Тестування алгоритму

### Часові характеристики оцінювання

В таблиці 3.1 наведено кількість порівнянь для 15 спроб пошуку запису по ключу.

Таблиця 3.1 – Число порівнянь при спробі пошуку запису по ключу

|  |  |
| --- | --- |
| Номер спроби пошуку | Число порівнянь |
| 1 | 11 |
| 2 | 12 |
| 3 | 11 |
| 4 | 13 |
| 5 | 16 |
| 6 | 9 |
| 7 | 16 |
| 8 | 10 |
| 9 | 10 |
| 10 | 12 |
| 11 | 14 |
| 12 | 11 |
| 13 | 12 |
| 14 | 12 |
| 15 | 13 |

Висновок

В ході лабораторної роботи я провела детальне дослідження червоно-чорного дерева, ознайомившись з основними підходами його проєктування та операціями з обробки складних структур даних. Основний фокус був приділений вивченню балансування червоно-чорного дерева, вставці, видаленню, пошуку, та заміні його вузлів.

Провівши аналіз асимптотичної складності алгоритму пошуку в червоно-чорному дереві, я визначила, що вона складає O(log n). Під час тестування алгоритму підтверджено, що моя реалізація пошуку також має аналогічну асимптотичну складність.

Результати мого дослідження представлені у Таблиці 3.1, де показано число порівнянь при спробах пошуку записів по ключу. Ці дані свідчать про ефективність реалізованого алгоритму, підтверджуючи його високу продуктивність при обробці данних у червоно-чорному дереві.

Критерії оцінювання

За умови здачі лабораторної роботи до 26.11.2023 включно максимальний бал дорівнює – 5. Після 26.11.2023 максимальний бал дорівнює – 4,5.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 10%;
* аналіз часової складності – 5%;
* програмна реалізація алгоритму – 50%;
* робота з гіт – 20%
* тестування алгоритму – 10%;
* висновок – 5%.

+1 додатковий бал можна отримати за реалізацію графічного відображення структури ключів.

+1 додатковий бал можна отримати за виконання та захист роботи до 19.11.2023.