**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-11 Трикош Іван*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2022

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

class TreeNode:

public key

public data

public left

public right

public height

method TreeNode(Key, Data):

key = Key

data = Data

left = None

right = None

height = 1

return

end method TreeNode

end class TreeNode

function insert\_node(node, key, data):

if node == None:

return TreeNode(key, data)

else if key < node.key:

node.left = insert\_node(node.left, key, data)

else if key > node.key:

node.right = insert\_node(node.right, key, data)

else:

return node

end if

node.height = 1 + max(get\_height(node.left), get\_height(node.right))

node = balance(node)

return node, is\_insert

end function insert\_node

function balance(node):

Balance = get\_balance(node)

if Balance < -1 and self.get\_balance(node.right) <= 0:

return left\_rotate(node)

end if

if Balance > 1 and get\_balance(node.left) >= 0:

return right\_rotate(node)

end if

if Balance > 1 and get\_balance(node.left) < 0:

node.left = left\_rotate(node.left)

return right\_rotate(node)

end if

if Balance < -1 and get\_balance(node.right) > 0:

node.right = right\_rotate(node.right)

return left\_rotate(node)

end if

return node

end function balance

function left\_rotate(node):

node\_right = node.right

node\_right\_\_left = node\_right.left

node\_right.left = node

node.right = node\_right\_\_left

node.height = 1 + max(get\_height(node.left), get\_height(node.right))

node\_right.height = 1 + max(get\_height(node\_right.left), get\_height(node\_right.right))

return node\_right

end function left\_rotate

function right\_rotate(node):

node\_left = node.left

node\_left\_\_right = node\_left.right

node\_left.right = node

node.left = node\_left\_\_right

node.height = 1 + max(get\_height(node.left), get\_height(node.right)) node\_ left.height = 1 + max(get\_height(node\_ left.left), get\_height(node\_ left.right))

return node\_left

end function right\_rotate

function get\_height(node):

if node == None:

return 0

end if

return node.height

end function get\_height

function get\_balance(node):

if node == None:

return 0

end if

return get\_height(node.left) - get\_height(node.right)

end function get\_balance

function get\_min\_node(node):

if node == None or node.left == None:

return node

end if

return get\_min\_node(node.left)

end function get\_min\_node

function get\_max\_node(node):

if node == None or node.right == None:

return node

end if

return get\_max\_node(node.right)

end function get\_max\_node

function delete\_node(node, key):

data = None

if node == None:

return node, data

else if key < node.key:

node.left, data = delete\_node(node.left, key)

else if key > node.key:

node.right, data = delete\_node(node.right, key)

else:

if node.left == None:

temp = node.right

data = node.data

node = None

return temp, data

else if node.right == None:

temp = node.left

node = None

return temp, data

end if

if get\_balance(node) < 0:

temp = get\_min\_node(node.right)

node.key, node.data = temp.key, temp.data

node.right, data = delete\_node(node.right, node.key)

else if get\_balance(node) > 0:

temp = get\_max\_node(node.left)

node.key, node.data = temp.key, temp.data

node.left, data = delete\_node(node.left, node.key)

else:

if node.key - node.left.key < node.right.key - node.key:

temp = get\_max\_node(node.left)

node.key, node.data = temp.key, temp.data

node.left, data = delete\_node(node.left, node.key)

else:

temp = get\_min\_node(node.right)

node.key, node.data = temp.key, temp.data

node.right, data = delete\_node(node.right, node.key)

end if

end if

end if

if node == None:

return node, data

end if

node.height = 1 + max(get\_height(node.left), get\_height(node.right))

node = balance(node)

return node, data

end function delete\_node

function change\_node(node, key, new\_data):

if not node:

return node

else if key < node.key:

node.left = change\_node(node.left, key, new\_data)

else if key > node.key:

node.right = change\_node(node.right, key, new\_data)

else:

node.data = new\_data

end if

return node

end function change\_node

function search\_node(node, key):

data = None

if node == None:

return data

else if key < node.key:

data = search\_node(node.left, key)

else if key > node.key:

data = search\_node(node.right, key)

else:

return node.data

return data

end function search\_node

## Часова складність пошуку

Оскільки висота АВЛ-дерева , де – число вузлів, то складність пошуку у найгіршому випадку (як і в середньому) буде , тобто асимптотична складність пошуку в АВЛ-дереві .

## Програмна реалізація

### Вихідний код

**AVL\_Tree.py**

class TreeNode:

"""Вузол дерева"""

def \_\_init\_\_(self, key, data):

"""Конструктор"""

self.key = key

self.data = data

self.left = None

self.right = None

self.height = 1

return

class AVL\_Tree:

"""АВЛ дерево"""

def insert\_node(self, node, key, data):

"""Вставка вузла в дерево"""

is\_insert = True

if not node:

return TreeNode(key, data), is\_insert

elif key < node.key:

node.left, is\_insert = self.insert\_node(node.left, key, data)

elif key > node.key:

node.right, is\_insert = self.insert\_node(node.right, key, data)

else:

is\_insert = False

return node, is\_insert

node.height = 1 + (self.get\_height(node.left) if self.get\_height(node.left) > self.get\_height(node.right) else self.get\_height(node.right))

node = self.balance(node)

return node, is\_insert

def balance(self, node):

"""Балансування дерева"""

Balance = self.get\_balance(node)

if Balance < -1 and self.get\_balance(node.right) <= 0:

return self.left\_rotate(node)

if Balance > 1 and self.get\_balance(node.left) >= 0:

return self.right\_rotate(node)

if Balance > 1 and self.get\_balance(node.left) < 0:

node.left = self.left\_rotate(node.left)

return self.right\_rotate(node)

if Balance < -1 and self.get\_balance(node.right) > 0:

node.right = self.right\_rotate(node.right)

return self.left\_rotate(node)

return node

def left\_rotate(self, node):

"""Поворот ліворуч"""

node\_right = node.right

node\_right\_\_left = node\_right.left

node\_right.left = node

node.right = node\_right\_\_left

node.height = 1 + (self.get\_height(node.left) if self.get\_height(node.left) > self.get\_height(node.right) else self.get\_height(node.right))

node\_right.height = 1 + (self.get\_height(node\_right.left) if self.get\_height(node\_right.left) > self.get\_height(node\_right.right) else self.get\_height(node\_right.right))

return node\_right

def right\_rotate(self, node):

"""Поворот праворуч"""

node\_left = node.left

node\_left\_\_right = node\_left.right

node\_left.right = node

node.left = node\_left\_\_right

node.height = 1 + (self.get\_height(node.left) if self.get\_height(node.left) > self.get\_height(node.right) else self.get\_height(node.right))

node\_left.height = 1 + (self.get\_height(node\_left.left) if self.get\_height(node\_left.left) > self.get\_height(node\_left.right) else self.get\_height(node\_left.right))

return node\_left

def get\_height(self, node):

"""Висота вузла"""

if not node:

return 0

return node.height

def get\_balance(self, node):

"""Баланс вузла"""

if not node:

return 0

return self.get\_height(node.left) - self.get\_height(node.right)

def get\_min\_node(self, node):

"""Мінімальне значення в піддереві"""

if not node or not node.left:

return node

return self.get\_min\_node(node.left)

def get\_max\_node(self, node):

"""Максимальне значення в піддереві"""

if not node or not node.right:

return node

return self.get\_max\_node(node.right)

def delete\_node(self, node, key):

"""Видалення елемента за ключем"""

data = None

if not node:

return node, data

elif key < node.key:

node.left, data = self.delete\_node(node.left, key)

elif key > node.key:

node.right, data = self.delete\_node(node.right, key)

else:

if not node.left:

temp = node.right

data = node.data

node = None

return temp, data

elif not node.right:

temp = node.left

node = None

return temp, data

if self.get\_balance(node) < 0:

temp = self.get\_min\_node(node.right)

node.key, node.data = temp.key, temp.data

node.right, data = self.delete\_node(node.right, node.key)

elif self.get\_balance(node) > 0:

temp = self.get\_max\_node(node.left)

node.key, node.data = temp.key, temp.data

node.left, data = self.delete\_node(node.left, node.key)

else:

if node.key - node.left.key < node.right.key - node.key:

temp = self.get\_max\_node(node.left)

node.key, node.data = temp.key, temp.data

node.left, data = self.delete\_node(node.left, node.key)

else:

temp = self.get\_min\_node(node.right)

node.key, node.data = temp.key, temp.data

node.right, data = self.delete\_node(node.right, node.key)

if not node:

return node, data

node.height = 1 + (self.get\_height(node.left) if self.get\_height(node.left) > self.get\_height(node.right) else self.get\_height(node.right))

node = self.balance(node)

return node, data

def change\_node(self, node, key, new\_data):

"""Зміна даних за ключем"""

is\_change = True

if not node:

is\_change = False

return node, is\_change

elif key < node.key:

node.left, is\_change = self.change\_node(node.left, key, new\_data)

elif key > node.key:

node.right, is\_change = self.change\_node(node.right, key, new\_data)

else:

node.data = new\_data

return node, is\_change

def search\_node(self, node, key):

"""Пошук даних за ключем"""

data = None

if not node:

return data

elif key < node.key:

data = self.search\_node(node.left, key)

elif key > node.key:

data = self.search\_node(node.right, key)

else:

return node.data

return data

def Print(self, node):

"""Вивід дерева"""

if not node:

return

print(node.key, node.data, end="\n")

self.Print(node.left)

self.Print(node.right)

return

def read\_data(self, node, path):

"""Читання даних з файлу"""

with open(path, "r") as file:

for line in file:

line = line[:-1] if line[-1] == "\n" else line

data = line.split()

data[0] = int(data[0])

node, temp = self.insert\_node(node, data[0], data[1])

return node

def write\_data(self, node, path):

"""Запис даних у файл"""

with open(path, "w") as file:

self.traversal(node, file)

return

def traversal(self, node, file):

"""Обхід дерева і запис даних у файл"""

if not node:

return

file.write(str(node.key) + " " + node.data + "\n")

self.traversal(node.left, file)

self.traversal(node.right, file)

return

### Приклади роботи

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

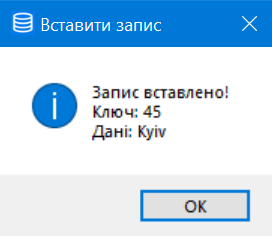


Рисунок 3.1 – Додавання запису

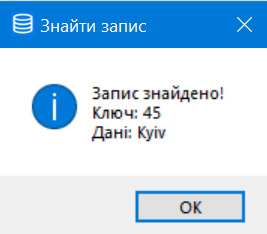


Рисунок 3.2 – Пошук запису

## 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 | 13 |
| 2 | 13 |
| 3 | 7 |
| 4 | 14 |
| 5 | 8 |
| 6 | 12 |
| 7 | 12 |
| 8 | 9 |
| 9 | 13 |
| 10 | 13 |
| 11 | 12 |
| 12 | 12 |
| 13 | 13 |
| 14 | 14 |
| 15 | 14 |

Висновок

В рамках лабораторної роботи я вивчив основні підходи проєктування та обробки складних структур даних. Я удосконалив свої знання про АВЛ дерева, про їхнє балансування, вставку, видалення, пошук, заміну його вузлів.

Я дослідив асимптотичну складність алгоритму пошуку в АВЛ дереві. Вона дорівнює . При тестуванні алгоритму я визначив, що мій алгоритм пошуку має таку саму асимптотичну складність.

Я навчився додавати, видаляти, шукати та змінювати записи в АВЛ дереві.

Окрім цього, я покращив свої знання з мови Python та її графічної бібліотеки tkinter.

Критерії оцінювання

За умови здачі лабораторної роботи до 13.11.2022 включно максимальний бал дорівнює – 5. Після 13.11.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 15%;
* аналіз часової складності – 5%;
* програмна реалізація алгоритму – 65%;
* тестування алгоритму – 10%;
* висновок – 5%.

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