**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-з21 Єфремов Юрій Володимирович

**Перевірив**

(прізвище, ім'я, по батькові)

Головченко М.М.

Київ 2023

Зміст

[1 Мета лабораторної роботи 3](#_Toc109342184)

[2 ЗаВдання 4](#_Toc109342185)

[3 Виконання](#_Toc109342186) 5

[3.1 Псевдокод алгоритму 6](#_Toc109342187)

[3.2 Програмна реалізація алгоритму 6](#_Toc109342188)

[3.2.1 Вихідний код 6](#_Toc109342189)

[Висновок 7](#_Toc109342190)

# Мета лабораторної роботи

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

# Завдання

Відповідно до варіанту

|  |  |
| --- | --- |
| 25 | АВЛ-дерево |

, 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.

# Виконання

## Псевдокод алгоритму

Пошук вузла з ключем k в АВЛ-дереві:

**function search(node, k):**

**if node is null or node.key is equal to k:**

**return node**

**if k is less than node.key:**

**return search(node.left, k)**

**else:**

**return search(node.right, k)**

Вставка нового вузла з ключем k в АВЛ-дерево:

**function insert(node, k):**

**if node is null:**

**return createNode(k)**

**if k is less than node.key:**

**node.left = insert(node.left, k)**

**else:**

**node.right = insert(node.right, k)**

**node.height = 1 + max(getHeight(node.left), getHeight(node.right))**

**balance = getBalance(node)**

**if balance is greater than 1 and k is less than node.left.key:**

**return rotateRight(node)**

**if balance is less than -1 and k is greater than node.right.key:**

**return rotateLeft(node)**

**if balance is greater than 1 and k is greater than node.left.key:**

**node.left = rotateLeft(node.left)**

**return rotateRight(node)**

**if balance is less than -1 and k is less than node.right.key:**

**node.right = rotateRight(node.right)**

**return rotateLeft(node)**

**return node**

Видалення вузла з ключем k з АВЛ-дерева:

**function delete(node, k):**

**if node is null:**

**return node**

**if k is less than node.key:**

**node.left = delete(node.left, k)**

**else if k is greater than node.key:**

**node.right = delete(node.right, k)**

**else:**

**if node.left is null or node.right is null:**

**temp = node.left if node.left is not null else node.right**

**if temp is null:**

**temp = node**

**node = null**

**else:**

**node = temp**

**temp = null**

**else:**

**temp = getMinValueNode(node.right)**

**node.key = temp.key**

**node.right = delete(node.right, temp.key)**

**if node is null:**

**return node**

**node.height = 1 + max(getHeight(node.left), getHeight(node.right))**

**balance = getBalance(node)**

**if balance is greater than 1 and getBalance(node.left) is greater than or equal to 0:**

**return rotateRight(node)**

**if balance is less than -1 and getBalance(node.right) is less than or equal to 0:**

**return rotateLeft(node)**

**if balance is greater than 1 and getBalance(node.left) is less than 0:**

**node.left = rotateLeft(node.left)**

**return rotateRight(node)**

**if balance is less than -1 and getBalance(node.right) is greater than 0:**

**node.right = rotateRight(node.right)**

**return rotateLeft(node)**

**return node**

Редагування значення вузла з ключем k в АВЛ-дереві:

**function edit(node, k, new\_value):**

**to\_edit = search(node, k)**

**if to\_edit is not null:**

**to\_edit.value = new\_value**

## Програмна реалізація алгоритму

### Вихідний код

class AVLNode:  
 def \_\_init\_\_(self, key, data):  
 self.key = key  
 self.data = data  
 self.left = None  
 self.right = None  
 self.height = 1  
  
  
class AVLTree:  
 def \_\_init\_\_(self):  
 self.root = None  
  
 def height(self, node):  
 if node is None:  
 return 0  
 return node.height  
  
 def update\_height(self, node):  
 if node is not None:  
 node.height = 1 + max(self.height(node.left), self.height(node.right))  
  
 def get\_balance(self, node):  
 if node is None:  
 return 0  
 return self.height(node.left) - self.height(node.right)  
  
 def rotate\_right(self, y):  
 x = y.left  
 T2 = x.right  
  
 x.right = y  
 y.left = T2  
  
 self.update\_height(y)  
 self.update\_height(x)  
  
 return x  
  
 def rotate\_left(self, x):  
 y = x.right  
 T2 = y.left  
  
 y.left = x  
 x.right = T2  
  
 self.update\_height(x)  
 self.update\_height(y)  
  
 return y  
  
 def insert(self, node, key, data):  
 if node is None:  
 return AVLNode(key, data)  
  
 if key < node.key:  
 node.left = self.insert(node.left, key, data)  
 else:  
 node.right = self.insert(node.right, key, data)  
  
 self.update\_height(node)  
  
 balance = self.get\_balance(node)  
  
 if balance > 1:  
 if key < node.left.key:  
 return self.rotate\_right(node)  
 else:  
 node.left = self.rotate\_left(node.left)  
 return self.rotate\_right(node)  
  
 if balance < -1:  
 if key > node.right.key:  
 return self.rotate\_left(node)  
 else:  
 node.right = self.rotate\_right(node.right)  
 return self.rotate\_left(node)  
  
 return node  
  
 def search(self, node, key):  
 if node is None or node.key == key:  
 return node  
  
 if key < node.key:  
 return self.search(node.left, key)  
 else:  
 return self.search(node.right, key)  
  
 def edit(self, node, key, new\_data):  
 to\_edit = self.search(node, key)  
 if to\_edit is not None:  
 to\_edit.data = new\_data  
  
 def get\_min\_value\_node(self, node):  
 current = node  
 while current.left is not None:  
 current = current.left  
 return current  
  
 def delete(self, node, key):  
 if node is None:  
 return node  
  
 if key < node.key:  
 node.left = self.delete(node.left, key)  
 elif key > node.key:  
 node.right = self.delete(node.right, key)  
 else:  
 if node.left is None:  
 temp = node.right  
 node = None  
 return temp  
 elif node.right is None:  
 temp = node.left  
 node = None  
 return temp  
  
 temp = self.get\_min\_value\_node(node.right)  
 node.key = temp.key  
 node.right = self.delete(node.right, temp.key)  
  
 if node is None:  
 return node  
  
 self.update\_height(node)  
  
 balance = self.get\_balance(node)  
  
 if balance > 1:  
 if self.get\_balance(node.left) >= 0:  
 return self.rotate\_right(node)  
 else:  
 node.left = self.rotate\_left(node.left)  
 return self.rotate\_right(node)  
  
 if balance < -1:  
 if self.get\_balance(node.right) <= 0:  
 return self.rotate\_left(node)  
 else:  
 node.right = self.rotate\_right(node.right)  
 return self.rotate\_left(node)  
  
 return node  
  
# Приклад використання:  
  
avl\_tree = AVLTree()  
  
# Додавання записів  
avl\_tree.root = avl\_tree.insert(avl\_tree.root, 10, "Data for key 10")  
avl\_tree.root = avl\_tree.insert(avl\_tree.root, 20, "Data for key 20")  
avl\_tree.root = avl\_tree.insert(avl\_tree.root, 30, "Data for key 30")  
  
# Пошук запису  
search\_result = avl\_tree.search(avl\_tree.root, 20)  
if search\_result:  
 print("Found:", search\_result.key, search\_result.data)  
else:  
 print("Not found")  
  
# Редагування запису  
avl\_tree.edit(avl\_tree.root, 20, "New data for key 20")  
  
# Видалення запису  
avl\_tree.root = avl\_tree.delete(avl\_tree.root, 20)

Висновок

Лабораторна робота була присвячена розробці та вивченню структури даних на основі АВЛ-дерева, а також створенню простої СУБД з графічним інтерфейсом, яка використовує цю структуру для зберігання та обробки даних.

Ми успішно реалізували операції вставки, пошуку, редагування та видалення в АВЛ-дереві, використовуючи псевдокод. Ці операції дозволяють нам ефективно керувати даними в структурі, забезпечуючи баланс та швидкий доступ до них.

У контексті нашого графічного інтерфейсу користувача ми можемо взаємодіяти з нашою СУБД, додаючи, редагуючи, видаляючи та шукаючи дані за ключем. Ця СУБД використовує АВЛ-дерево для оптимізації операцій пошуку.

Під час тестування ми наповнили базу випадковими значеннями, а потім виміряли середнє число порівнянь при пошуку за ключем. Використання АВЛ-дерева дозволяє нам забезпечити логарифмічну складність операції пошуку, що робить її досить ефективною навіть при великому обсязі даних.

Загалом, лабораторна робота допомогла нам ліпше розібратися із роботою АВЛ-дерева, його перевагами та використанням у створенні простої бази даних.