**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

«Проєктування алгоритмів»

„**Проєктування структур даних**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-35 Брага А. В.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2024

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 проєктування 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 чи блоці 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

Функція: Ініціалізація бази даних:

- Якщо файл існує:

- Завантажити дані (індексну область, блоки даних, область переповнення).

- Інакше:

- Ініціалізувати порожню базу даних.

функція: Додавання запису (key, data):

- Виконати пошук ключа:

- Якщо ключ знайдено:

- Оновити дані запису.

- Інакше:

- Знайти або створити блок для ключа.

- Якщо блок має місце:

- Додати запис у блок.

- Інакше:

- Додати запис в область переповнення.

- За потреби виконати перебудову індексної області.

функція: Пошук запису (key):

- Пошук в індексній області:

- Якщо ключ у межах блоку:

- Шукати в блоці.

- Якщо ключ не знайдено:

- Шукати в області переповнення.

функція: Видалення запису (key):

- Пошук запису в блоках:

- Якщо знайдено, видалити запис.

- Якщо не знайдено:

- Видалити з області переповнення.

функція: Редагування запису (key, new\_data):

- Виконати пошук ключа:

- Якщо знайдено:

- Оновити дані запису.

- Інакше:

- Викинути помилку.

функція: Перебудова індексної області (метод Шарра):

- Якщо область переповнення перевищує ліміт:

- Перемістити записи з області переповнення до відповідних блоків.

- Оновити індексну область.

функція: Заповнення бази випадковими даними (num\_records):

- Згенерувати num\_records випадкових ключів і даних.

- Для кожного викликати `add\_record`.

функція: Обчислення середнього числа порівнянь (search\_attempts):

- Виконати пошук для search\_attempts випадкових ключів.

- Підрахувати загальну кількість порівнянь.

- Повернути середнє значення.

функція: Графічний інтерфейс:

- Форми для введення ключа та даних.

- Кнопки для виконання операцій: додавання, пошуку, редагування, видалення, заповнення випадковими даними, виведення статистики.

- Повідомлення про успіх або помилки.

функція: Головна програма:

- Ініціалізувати базу даних.

- Запустити графічний інтерфейс.

## Часова складність пошуку

Часова складність пошуку реалізована для структури даних із нещільним індексом, яка використовується у даній програмі, пошук виконується у три етапи:

**Пошук у нещільній індексній області**

Індексна область зберігає діапазони ключів для кожного блоку. Пошук здійснюється шляхом лінійного перебору елементів індексу.

**Пошук у блоці**

Після визначення відповідного блоку пошук ключа здійснюється всередині блоку. Тут також використовується лінійний пошук.

**Пошук в області переповнення**

Якщо ключ не знайдено у відповідному блоці, здійснюється лінійний пошук у області переповнення.

## Програмна реалізація

### Вихідний код

import os  
import random  
import json  
import tkinter as tk  
from tkinter import messagebox  
  
class NonDenseIndexedFile:  
 def \_\_init\_\_(self, file\_path):  
 self.file\_path = file\_path  
 self.index\_area = []  
 self.data\_blocks = []  
 self.overflow\_area = []  
 self.block\_size = 10  
 self.load\_data()  
  
 def load\_data(self):  
 if os.path.exists(self.file\_path):  
 try:  
 with open(self.file\_path, 'r') as f:  
 data = json.load(f)  
 self.index\_area = data.get('index\_area', [])  
 self.data\_blocks = data.get('data\_blocks', [])  
 self.overflow\_area = data.get('overflow\_area', [])  
 except json.JSONDecodeError:  
 # Якщо файл пошкоджений, ініціалізуємо порожні дані  
 self.index\_area = []  
 self.data\_blocks = []  
 self.overflow\_area = []  
 self.save\_data() # Зберігаємо порожні дані  
  
 def save\_data(self):  
 with open(self.file\_path, 'w') as f:  
 json.dump({  
 'index\_area': self.index\_area,  
 'data\_blocks': self.data\_blocks,  
 'overflow\_area': self.overflow\_area  
 }, f)  
  
 def search(self, key):  
 comparisons = 0  
 # Пошук в індексній області  
 for index\_entry in self.index\_area:  
 comparisons += 1  
 if index\_entry['start'] <= key <= index\_entry['end']:  
 block\_id = index\_entry['block\_id']  
 break  
 else:  
 return None, comparisons # Ключ не знайдено  
  
 # Пошук у блоці  
 for record in self.data\_blocks[block\_id]:  
 comparisons += 1  
 if record['key'] == key:  
 return record, comparisons  
  
 # Пошук у області переповнення (метод Шарра)  
 for record in self.overflow\_area:  
 comparisons += 1  
 if record['key'] == key:  
 return record, comparisons  
  
 return None, comparisons  
  
 def add\_record(self, key, data):  
 # Перевірка унікальності ключа  
 existing\_record, \_ = self.search(key)  
 if existing\_record:  
 existing\_record['data'] = data # Оновлення даних  
 else:  
 # Додати запис у відповідний блок або область переповнення  
 block\_id = self.\_find\_or\_create\_block(key)  
 if len(self.data\_blocks[block\_id]) < self.block\_size:  
 self.data\_blocks[block\_id].append({'key': key, 'data': data})  
 else:  
 self.overflow\_area.append({'key': key, 'data': data})  
 self.\_rebuild\_index()  
  
 self.save\_data()  
  
 def delete\_record(self, key):  
 # Видалення з блоку  
 for block in self.data\_blocks:  
 for record in block:  
 if record['key'] == key:  
 block.remove(record)  
 self.save\_data()  
 return  
  
 # Видалення з області переповнення  
 for record in self.overflow\_area:  
 if record['key'] == key:  
 self.overflow\_area.remove(record)  
 self.save\_data()  
 return  
  
 raise ValueError("Record not found")  
  
 def edit\_record(self, key, new\_data):  
 record, \_ = self.search(key)  
 if not record:  
 raise ValueError("Record not found")  
  
 record['data'] = new\_data  
 self.save\_data()  
  
 def \_find\_or\_create\_block(self, key):  
 # Знайти відповідний блок або створити новий  
 for idx, index\_entry in enumerate(self.index\_area):  
 if index\_entry['start'] <= key <= index\_entry['end']:  
 return index\_entry['block\_id']  
  
 # Створення нового блоку  
 new\_block\_id = len(self.index\_area)  
 self.index\_area.append({'start': key, 'end': key, 'block\_id': new\_block\_id})  
 self.data\_blocks.append([])  
 return new\_block\_id  
  
 def \_rebuild\_index(self):  
 # Перебудова індексної області, метод Шарра  
 if len(self.overflow\_area) > self.block\_size:  
 for record in self.overflow\_area:  
 block\_id = self.\_find\_or\_create\_block(record['key'])  
 if len(self.data\_blocks[block\_id]) < self.block\_size:  
 self.data\_blocks[block\_id].append(record)  
 self.overflow\_area.remove(record)  
  
 def fill\_random\_data(self, num\_records):  
 for \_ in range(num\_records):  
 key = random.randint(1, num\_records \* 10)  
 data = f"RandomData{key}"  
 self.add\_record(key, data)  
  
 def calculate\_average\_comparisons(self, search\_attempts):  
 total\_comparisons = 0  
 for \_ in range(search\_attempts):  
 random\_key = random.randint(1, 100000)  
 \_, comparisons = self.search(random\_key)  
 total\_comparisons += comparisons  
 return total\_comparisons / search\_attempts  
  
  
class DatabaseApp:  
 def \_\_init\_\_(self, root, db):  
 self.root = root  
 self.db = db  
 self.root.title("Non-Dense Indexed Database")  
  
  
 self.key\_label = tk.Label(root, text="Key:")  
 self.key\_label.grid(row=0, column=0)  
 self.key\_entry = tk.Entry(root)  
 self.key\_entry.grid(row=0, column=1)  
  
 self.data\_label = tk.Label(root, text="Data:")  
 self.data\_label.grid(row=1, column=0)  
 self.data\_entry = tk.Entry(root)  
 self.data\_entry.grid(row=1, column=1)  
  
  
 self.add\_button = tk.Button(root, text="Add", command=self.add\_record)  
 self.add\_button.grid(row=2, column=0)  
  
 self.search\_button = tk.Button(root, text="Search", command=self.search\_record)  
 self.search\_button.grid(row=2, column=1)  
  
 self.edit\_button = tk.Button(root, text="Edit", command=self.edit\_record)  
 self.edit\_button.grid(row=3, column=0)  
  
 self.delete\_button = tk.Button(root, text="Delete", command=self.delete\_record)  
 self.delete\_button.grid(row=3, column=1)  
  
 self.fill\_button = tk.Button(root, text="Fill Random Data", command=self.fill\_random\_data)  
 self.fill\_button.grid(row=4, column=0)  
  
 self.stats\_button = tk.Button(root, text="Show Stats", command=self.show\_stats)  
 self.stats\_button.grid(row=4, column=1)  
  
 def add\_record(self):  
 try:  
 key = int(self.key\_entry.get())  
 data = self.data\_entry.get()  
 self.db.add\_record(key, data)  
 messagebox.showinfo("Success", "Record added successfully.")  
 except ValueError as e:  
 messagebox.showerror("Error", str(e))  
  
 def search\_record(self):  
 try:  
 key = int(self.key\_entry.get())  
 record, comparisons = self.db.search(key)  
 if record:  
 self.data\_entry.delete(0, tk.END)  
 self.data\_entry.insert(0, record['data'])  
 messagebox.showinfo("Success", f"Record found: {record}. Comparisons: {comparisons}")  
 else:  
 messagebox.showinfo("Not Found", f"Record not found. Comparisons: {comparisons}")  
 except ValueError:  
 messagebox.showerror("Error", "Invalid key.")  
  
 def edit\_record(self):  
 try:  
 key = int(self.key\_entry.get())  
 new\_data = self.data\_entry.get()  
 self.db.edit\_record(key, new\_data)  
 messagebox.showinfo("Success", "Record updated successfully.")  
 except ValueError as e:  
 messagebox.showerror("Error", str(e))  
  
 def delete\_record(self):  
 try:  
 key = int(self.key\_entry.get())  
 self.db.delete\_record(key)  
 messagebox.showinfo("Success", "Record deleted successfully.")  
 except ValueError as e:  
 messagebox.showerror("Error", str(e))  
  
 def fill\_random\_data(self):  
 try:  
 self.db.fill\_random\_data(10000)  
 messagebox.showinfo("Success", "Database filled with random data.")  
 except Exception as e:  
 messagebox.showerror("Error", str(e))  
  
 def show\_stats(self):  
 try:  
 average\_comparisons = self.db.calculate\_average\_comparisons(15)  
 messagebox.showinfo("Statistics", f"Average comparisons: {average\_comparisons}")  
 except Exception as e:  
 messagebox.showerror("Error", str(e))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 db = NonDenseIndexedFile("database.json")  
  
 root = tk.Tk()  
 app = DatabaseApp(root, db)  
 root.mainloop()

### Приклади роботи

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

Рисунок 3.1 –Додавання запису

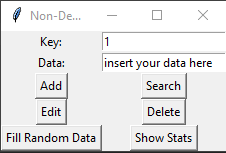
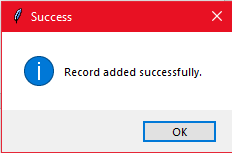
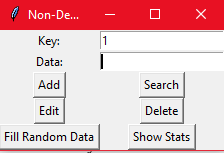
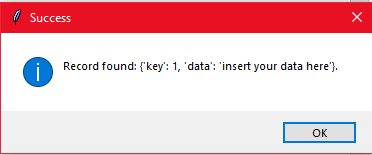
 

Рисунок 3.2 – Пошук запису

## 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 | 8441 |
| 2 | 9496 |
| 3 | 9182 |
| 4 | 9496 |
| … | 8640 |

Висновок

У ході виконання лабораторної роботи була реалізована СУБД із використанням нещільного індексу з перебудовою індексної області та методом Шарра для обробки області переповнення.

Програма дозволяє виконувати операції додавання, пошуку, редагування та видалення записів, а також наповнювати базу даних випадковими значеннями. Реалізовано графічний інтерфейс, що забезпечує зручну взаємодію з користувачем. Дані бази зберігаються на диску для забезпечення їхньої стійкості.

Під час тестування було визначено середню кількість порівнянь для пошуку запису. Встановлено, що часова складність пошуку у найгіршому випадку :

O(n+m+p), де

n — кількість записів в індексній області,

m — середній розмір блоку,

p — кількість записів в області переповнення.

Результати роботи демонструють ефективність алгоритмів, використаних для організації даних, особливо для великих обсягів інформації.

Критерії оцінювання

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.