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) ІП-35 Адаменко Арсен Богданович _____
(шифр, прізвище, ім'я, по батькові)

Перевірив Головченко М.М. _____
(прізвище, ім'я, по батькові)

Київ 2024

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

2 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

3 ВИКОНАННЯ

3.1 Псевдокод алгоритмів

```
ФУНКЦІЯ Бінарний_пошук_по_індексним_блокам(Ключ)
	Кількість_блоків = Розмір_файла() // Розмір_блоку
	Від = 0
	До = Кількість_блоків — 1

	ЗАВЖДИ ПОВТОРЮВАТИ
		Сер = (Від + До) / 2
		СБЗ = Блок(Сер)
		НСБЗ = Блок(Сер)

		ЯКЩО Ключ < СБЗ.Мін ТО
			До = Сер - 1
		ІНАКШЕ_ЯКЩО СБЗ.Макс < Ключ ТО
			Від = Сер + 1
		ІНАКШЕ
			ЯКЩО СБЗ.Мін < Ключ < СБЗ.Макс ТО
				ПОВЕРНУТИ Сер
			ІНАКШЕ
				ПОВЕРНУТИ НУЛЛ
			КІНЕЦЬ
		КІНЕЦЬ
	КІНЕЦЬ
ЦІНЕЦЬ

ФУНКЦІЯ Бінарний_пошук_по_індексному_блоці(Ключ, Блок)
	Кількість_записів = Кількість_записів()
	Від = 0
	До = Кількість_записів - 1

	ЗАВЖДИ ПОВТОРЮВАТИ
		Сер = (Від + До) / 2
		СБЗ = Запис_по_блоку(Сер)

		ЯКЩО Ключ < СБЗ ТО
			До = Сер - 1
		ІНАКШЕ_ЯКЩО СБЗ < Ключ ТО
			Від = Сер + 1
		ІНАКШЕ
			ЯКЩО Ключ = СБЗ ТО
```

```
                        ПОВЕРНУТИ Сер + Змищенння_блоку(Ключ, Блок)
                  ІНАКШЕ
                        ПОВЕРНУТИ НУЛЛ
                  КІНЕЦЬ
            КІНЕЦЬ
      КІНЕЦЬ
ЦІНЕЦЬ

ФУНКЦІЯ Отримати_запис(Індекс)
      Індекс_блоку = Бінарний_пошук_по_індексним_блокам(Індекс)
      Індекс_запису =  Бінарний_пошук_по_індексному_блоці(Індекс_блоку, Індекс)
КІНЕЦЬ

ФУНКЦІЯ Додати_запис(Індекс, Значення)
      Індекс_блоку = Бінарний_пошук_по_індексним_блокам(Індекс)

      ЯКЩО Блок_не_повний(Індекс_блоку) ТО
            Додати_індекс_в_початок(Індекс_блоку, Індекс, Значення)
      ІНАКШЕ
            Створити_новий_файл(«Index.bin.tmp»)
            ДЛЯ Кожний_індекс В Індекси() РОБИТИ
                  Записати_індекс_в_кінець_файла(«Index.bin.tmp», Кожний_індекс)
            КІНЕЦЬ

            Встановити_коефіцієнт_заповнення(«Index.bin», Коефіцієнт_заповнення)

            ДЛЯ Кожний_індекс В Індекси_нового_файлу() РОБИТИ
                  Записати_індекс_в_кінець_файла(«Index.bin», Кожний_індекс)
            КІНЕЦЬ
      КІНЦЕЬ
КІНЕЦЬ

ФУНКЦІЯ Видалити_запис(Індекс)
      Індекс_блоку = Бінарний_пошук_по_індексним_блокам(Індекс)
      Індекс_запису =  Бінарний_пошук_по_індексному_блоці(Індекс_блоку)

      Перемістити_індекси_в_блоці_лівіше(Індекс_запису, Індекс_блоку)

      ЯКЩО Індексний_блок_пустий(Індекс_блоку) ТО
            Перерозподілити_індекси_в_блоці_з_сусідніми(Індекс_блоку)
      КІНЕЦЬ
КІНЕЦЬ
```

3.2 Часова складність пошуку

Кількість пар індекс значення: n. Кількість індексних записів в одному індексному блоці: 64. Кількість блоків з індексними записами пар індекс-ключ: $\frac{n}{64}$.

Кількість ітерацій пошуку індексного блоку з потрібним діапазом значень ключів: $O(f(n)\log n)$, де $f(n)$ — функція часової складності пошуку діапазону значень в одному блоці, і вона рівна $\log_2 64 = 6 = O(1)$ (процедура знаходження діапазону та кількості індексів в блоці використовує бінарний пошук або обійтися без нього, бо кількість індексних записів не перевищує розмір блоку, який є константою в цій лабораторній роботі та лекційному матеріалі). Тому складність знаходження індексного блоку: $O(\log n)$.

Час знаходження потрібного запису в індексному блоці відбувається за $O(\log n)$ тому що так як записи по індексу відсортовані, то можна використати бінарний пошук для знаходження ключа в індексному блоці.

Остаточна часова складність знаходження потрібного індексного запису та ключа рівна $O(\log n)+O(\log n)=O(\log n)$.

Так як основна частина з записами даних має табличний формат, а також первинний ключ є прямим знначенням індекса в масиві з записами даних в основній частині з даними, то час пошуку в основній частині займає рівно $O(1)$.

3.3 Програмна реалізація

3.3.1 Вихідний код

Main.py:

```
import tkinter as tk
import IndexIO
import RecordIO

load_factor = 0.7

db_path = 'db'

idx_filename = 'index.bin'
rec_filename = 'record.bin'
```

```
idx_io = None
rec_io = None

def IntToBytes(i: int, size: int):
    return i.to_bytes(size, byteorder='little', signed=False)

def BytesToInt(b: bytes):
    return int.from_bytes(b, byteorder='little', signed=False)

if 0:
    #idx_io = IndexIO.IndexIO(db_path + '/' + idx_filename)
    #rec_io = RecordIO.RecordIO(db_path + '/' + rec_filename)

    #idx_io.Add('0000'.encode('ascii'), '000'.encode('ascii'))

    #for i in range(1, 81):
    #    idx_io.Add(f'{i * 100:0>5}'.encode('ascii'), f'{hex(i)
[2:]:0>2}'.encode('ascii'))

    #for i in range(10000):
    #    idx_io.Add(f'{i:0>5}'.encode('ascii'), f'{hex(i)
[2:]:0>3}'.encode('ascii'))

    exit(1)

def clrmsg_btn_handler():
    global messages_console

    messages_console.delete(1.0, tk.END)

def messages_add_line(line):
    global messages_console

    messages_console.insert(tk.END, line + '\n')

def load_db_btn_handler():
    global db_name_entry
    global db_in_use_value

    global db_path, idx_io, rec_io

    db_path = db_name_entry.get()

    db_in_use_value.config(text=db_path + '/*')

    idx_io = IndexIO.IndexIO(db_path + '/' + idx_filename)
    rec_io = RecordIO.RecordIO(db_path + '/' + rec_filename)

    messages_add_line(f'Було завантажено БД по шляху {db_path}')

def wipe_db_btn_handler():
    global idx_io, rec_io

    if not (idx_io != None):
        messages_add_line('Немає індексного файлу')
        return

    if not (rec_io != None):
        messages_add_line('Немає записного файлу')
        return

    idx_io.Wipe()
    rec_io.Wipe()
```

```
    messages_add_line('БД була повністю очищена')

def win_on_exit():
    global idx_io, rec_io

    if idx_io != None:
        idx_io.Sync()

    if rec_io != None:
        rec_io.Sync()

    win.destroy()

def get_btn_handler():
    global idx_io, rec_io
    global key_value_entry

    if not (idx_io != None):
        messages_add_line('Немає індексного файлу')
        return

    if not (rec_io != None):
        messages_add_line('Немає записного файлу')
        return

    key = key_value_entry.get()
    raw_id_info = idx_io.Get(key.encode('ascii'))

    if not (raw_id_info != None):
        messages_add_line(f'Немає такого ключа як <{key}>')
        return

    iters = raw_id_info['iters']
    id_ = BytesToInt(raw_id_info['id'])

    messages_add_line(f'По ключу <{key}> було отримано значення
<{rec_io.Get(id_)}> за <{iters}> ітерацій')

def add_btn_handler():
    global idx_io, rec_io
    global key_value_entry
    global content_value_value

    if not (idx_io != None):
        messages_add_line('Немає індексного файлу')
        return

    if not (rec_io != None):
        messages_add_line('Немає записного файлу')
        return

    key  = key_value_entry.get()
    raw_key = key.encode('ascii')

    if not (idx_io.Get(raw_key) == None):
        messages_add_line(f'Такий ключ як <{key}> вже існує')
        return

    data = content_value_value.get()

    id_ = rec_io.Add(data)

    raw_id = IntToBytes(id_, 4)
```

```
    idx_io.Add(raw_key, raw_id)

    messages_add_line(f'Пара ключ-значення <{key}>/<{data}> були додані')

def remove_btn_handler():
    global idx_io, rec_io
    global key_value_entry

    if not (idx_io != None):
        messages_add_line('Немає індексного файлу')
        return

    if not (rec_io != None):
        messages_add_line('Немає записного файлу')
        return

    key  = key_value_entry.get()
    raw_key = key.encode('ascii')

    if not (idx_io.Get(raw_key) != None):
        messages_add_line(f'Такий ключ як <{key}> не існує')
        return

    idx_io.Remove(raw_key)

    messages_add_line(f'Ключ <{key}> та його значення було видалено')

def add_10000_btn_handler():
    global idx_io, rec_io

    if not (idx_io != None):
        messages_add_line('Немає індексного файлу')
        return

    if not (rec_io != None):
        messages_add_line('Немає записного файлу')
        return

    messages_add_line('Початок процесу зарахування 10000 абітурієнтів')

    for i in range(10000):
        key  = f'{i:0>6}'.encode('ascii')
        id_  = IntToBytes(i, 4)
        data = str(i) + '=' + hex(i)

        idx_io.Add(key, id_)
        rec_io.Add(data)

    messages_add_line('Було зараховано 10000 абітурієнтів')

win = tk.Tk()
win.title('Лабораторна робота №3')

win_padding = tk.Frame(win, padx=8, pady=8)
win_padding.grid(row=0, column=0, sticky='nsew')

win.protocol("WM_DELETE_WINDOW", win_on_exit)

title_label = tk.Label(
    win_padding,
    text='ДИСКЛЕЙМЕР, ЦЯ дисципліна створена, щоб студенти страдали. Ця
дисципліна це негуманний експеримент'.upper(),
    justify='center'
)
```

```
title_label.grid(row=0, column=0, columnspan=3, sticky='nsew')

load_db_btn = tk.Button(
    win_padding,
    text='Зайти до ВУЗ',
    justify='center',
    command=load_db_btn_handler
)
load_db_btn.grid(row=1, column=0, sticky='nsew')

db_name_entry = tk.Entry(
    win_padding,
    justify='center'
)
db_name_entry.grid(row=1, column=1, sticky='nsew')

db_in_use_desc = tk.Label(
    win_padding,
    text='Обраний ВУЗ:',
    justify='center'
)
db_in_use_desc.grid(row=2, column=0, sticky='nsew')

db_in_use_value = tk.Label(
    win_padding,
    text='<БД не відкрита>',
    justify='center'
)
db_in_use_value.grid(row=2, column=1, sticky='nsew')

# Massive operations

wipe_db_btn = tk.Button(
    win_padding,
    text='ЗНИЩИТИ ВСІ ДИПЛОМИ',
    justify='center',
    command=wipe_db_btn_handler
)
wipe_db_btn.grid(row=3, column=0, sticky='nsew')

add_10000_btn = tk.Button(
    win_padding,
    text='Зарахувати 10000 абітурів',
    justify='center',
    command=add_10000_btn_handler
)
add_10000_btn.grid(row=4, column=0, sticky='nsew')

# Inputs

key_value_desc = tk.Label(
    win_padding,
    text='Студент:',
    justify='center'
)
key_value_desc.grid(row=5, column=0, sticky='nsew')

key_value_entry = tk.Entry(
    win_padding,
    justify='center'
)
key_value_entry.grid(row=5, column=1, sticky='nsew')

# ---
```

```
content_value_desc = tk.Label(
    win_padding,
    text='Успішсть:',
    justify='center'
)
content_value_desc.grid(row=6, column=0, sticky='nsew')

content_value_value = tk.Entry(
    win_padding,
    justify='center'
)
content_value_value.grid(row=6, column=1, sticky='nsew')

# Manipluate DB

get_btn = tk.Button(
    win_padding,
    text='Стеження 24/7',
    justify='center',
    command=get_btn_handler
)
get_btn.grid(row=7, column=0, sticky='nsew')

add_btn = tk.Button(
    win_padding,
    text='Зарахувати',
    justify='center',
    command=add_btn_handler
)
add_btn.grid(row=8, column=0, sticky='nsew')

remove_btn = tk.Button(
    win_padding,
    text='ВІДРАХУВАТИ',
    justify='center',
    command=remove_btn_handler
)
remove_btn.grid(row=9, column=0, sticky='nsew')

clrmsg_btn = tk.Button(
    win_padding,
    text='ЗНИЩИТИ ВСЮ ПАЛЬ',
    justify='center',
    command=clrmsg_btn_handler
)
clrmsg_btn.grid(row=10, column=0, sticky='nsew')

# Messages

messages_console = tk.Text(
    win_padding
)
messages_console.grid(row=1, column=2, rowspan=10, sticky='nsew')

tk.mainloop()
```

IndexIO.py:

```
import FileMapper
import ZTStrUtils

import math
```

```
def FitTo(bytes_, size):
    ln = len(bytes_)

    if not (ln < size):
        return bytes_[:size]

    return bytes_ + bytes([0]) * (size - ln)

def MaxRecordsWhenAppending(rpb, loadFactor):
    return min(math.ceil(rpb * loadFactor), rpb - 1)

def DistributeTwoBlocksCounts(recordsCount):
    first = math.ceil(recordsCount / 2)
    second = recordsCount - first

    return {'first': first, 'second': second}

# NO ONE EVEN SATAN should not see this OVERJUNKED and OVERROTTEN
# and FIFTHLY TIME DIGESTED PIECE OF SHIT
# ****** **********, YOU ARE A PIECE OF SHIT SHATTEN BY MUHA IRYNA PAVLIVNA

class IndexIO:
    indexSize = 12
    idSize = 4
    recordSize = 16

    def __init__(self, filename, *, blockSize=1024, loadFactor=0.75):
        self.fm = FileMapper.FileMapper(filename, cache_size=1024)
        self.rpb = blockSize // self.recordSize
        self.blockSize = blockSize
        self.loadFactor = loadFactor

    def Get(self, key):
        iters = [0]

        index = self._get_index_by_key(key, iters)

        return {'id': self._get_record(index)['id'], 'iters': iters[0]} if
index != None else None

    def Remove(self, key):
        index = self._get_index_by_key(key)

        if not (index != None):
            return

        block_index = self._get_block_index_by_record_index(index)
        block_index_range = self._get_block_index_range(block_index)

        for i in range(index, block_index_range['end']):
            new_record = self._get_record(i + 1)

            key = new_record['key']
            id_ = new_record['id']

            self._set_record(i, key, id_)

        # since it is moved lefter, remove an existance mark
        self._set_record(block_index_range['end'], bytes([]), bytes([]))

        self._redistribute_after_remove(block_index)

    def Add(self, key, id_):
        if not (self._get_index_by_key(key) == None):
```

```
            return

        if self._add_append_impl_and_return_status(key, id_):
            return

        self._add_insert_into_block(key, id_)

    def _add_insert_into_block(self, key, id_):
        #-----------------------------------------
        #-----------------------------------------

        blocks_range = self._get_all_blocks_index_range()

        l = 0

        # in a case for next mvr var validness
        h = blocks_range['end'] - 1

        for _ in range(10):
            m = (l + h) // 2

            mvr = self._get_block_value_range(m)
            nmvr = self._get_block_value_range(m + 1)

            if key < mvr['min']:
                h = m
            elif nmvr['max'] < key:
                l = m
            else:
                break

        if nmvr['min'] <= key and key <= nmvr['max']:
            m += 1

        block_index = m

        #-----------------------------------------
        #-----------------------------------------

        if self._is_block_filled_out(block_index):
            self._copy_blocks_to('temp.bin')

            self.fm.WipeData()

            self._copy_back_and_insert('temp.bin', key, id_)

        # OH SHIT, HERE WE GO AGAIN! I HATE MY ALGORITHMS DESIGNING TEACHER

        #-----------------------------------------
        #-----------------------------------------

        blocks_range = self._get_all_blocks_index_range()

        l = 0

        # in a case for next mvr var validness
        h = blocks_range['end'] - 1

        for _ in range(10):
            m = (l + h) // 2

            mvr = self._get_block_value_range(m)
            nmvr = self._get_block_value_range(m + 1)
```

```
            if key < mvr['min']:
                h = m
            elif nmvr['max'] < key:
                l = m
            else:
                break

        if nmvr['min'] <= key and key <= nmvr['max']:
            m += 1

        block_index = m

        #----------------------------------------
        #----------------------------------------

        block_records_count = self._get_block_records_count(block_index)

        # there is off by 1 trick!
        self._set_record(self._get_record_index_by_block(block_index) +
block_records_count, key, id_)

        # including a new record
        # in this order exactly
        block_index_range = self._get_block_index_range(block_index)

        # there is off by 1 trick!
        for i in range(block_index_range['end'] - 1,
block_index_range['start'] - 1, -1):
            key_1 = self._get_record(i)['key']
            key_2 = self._get_record(i + 1)['key']

            if not (key_1 > key_2):
                break

            self._swap_records(i, i + 1)

    def Wipe(self):
        self.fm.WipeData()

    def Sync(self):
        self.fm._write_cache()

    def _copy_back_and_insert(self, filename, key, id_):
        tmp_record_id = 0

        tmp_fm = FileMapper.FileMapper(filename, cache_size=1024)

        records_count = tmp_fm.Size() // self.recordSize

        block_id = 0
        record_id = 0

        records_in_block = 0

        while tmp_record_id < records_count:
            tmp_record = tmp_fm.ReadMany(tmp_record_id * self.recordSize,
self.recordSize)

            self._set_record_raw(self._get_record_index_by_block(block_id) +
record_id, tmp_record)

            tmp_record_id += 1
            record_id += 1
            records_in_block += 1
```

```
            if MaxRecordsWhenAppending(self.rpb, self.loadFactor) <=
records_in_block and tmp_record_id != records_count - 1:
                records_in_block = 0
                record_id = 0

                block_id += 1

                for i in range(self.rpb):

self._set_record(self._get_record_index_by_block(block_id) + i, bytes([]),
bytes([]))

    def _copy_blocks_to(self, filename):
        with open(filename, 'w'):
            pass

        tmp_fm = FileMapper.FileMapper(filename, cache_size=1024)

        count = 0

        # there is off by 1 trick!
        for block_i in range(self._get_all_blocks_index_range()['end'] + 1):
            for record_i in range(self._get_block_index_range(block_i)['end']
+ 1):
                record = self._get_record_raw(record_i)
                tmp_fm.WriteMany(record_i * self.recordSize, record)

                count += 1

    def _swap_records(self, index_1, index_2):
        record_1 = self.fm.ReadMany(index_1 * self.recordSize,
self.recordSize)
        record_2 = self.fm.ReadMany(index_2 * self.recordSize,
self.recordSize)

        self.fm.WriteMany(index_1 * self.recordSize, record_2)
        self.fm.WriteMany(index_2 * self.recordSize, record_1)

    def _move_record_src_dst(self, src, dst):
        record = self.fm.ReadMany(src * self.recordSize, self.recordSize)

        self.fm.WriteMany(dst * self.recordSize, record)

    def _add_append_impl_and_return_status(self, key, id_):
        blocks_range = self._get_all_blocks_index_range()

        last_block_index = blocks_range['end']

        if last_block_index == -1:
            for i in range(self.rpb):
                self._set_record(i, bytes([]), bytes([]))

            last_block_index = 0

        max_value = self._get_block_value_range(last_block_index)['max']

        if key > max_value:
            block_fillness = self._get_block_fillness(last_block_index)

            if block_fillness < self.loadFactor:
                last_block_end_index =
self._get_block_index_range(last_block_index)['end']
                self._set_record(last_block_end_index + 1, key, id_)
```

```
            else:
                next_last_block_first_record_index =
self._get_record_index_by_block(last_block_index + 1)

                self._fill_block(last_block_index + 1)

                self._set_record(next_last_block_first_record_index, key,
id_)

            return True

        return False

    def _fill_block(self, index):
        for i in range(self.rpb):
            self._set_record(self._get_record_index_by_block(index) + i,
bytes([]), bytes([]))

    # get record index by key

    def _get_index_by_key(self, key, iters=None):
        if self.fm.Size() == 0:
            return None

        l = 0
        h = self._get_all_blocks_index_range()['end']

        for _ in range(30):
            if iters != None:
                iters[0] += 1

            m = (l + h) // 2

            mvr = self._get_block_value_range(m)

            if key < mvr['min']:
                h = m - 1
            elif     mvr['max'] < key:
                l = m + 1
            else:
                if mvr['min'] <= key <= mvr['max']:
                    break
                else:
                    return None

        block_range = self._get_block_index_range(m)

        l = block_range['start']
        h = block_range['end']

        for _ in range(20):
            m = (l + h) // 2

            mv = self._get_record(m)['key']

            if key < mv:
                h = m - 1
            elif     mv < key:
                l = m + 1
            else:
                if mv == key:
                    return m
                else:
                    return None
```

```
    # basic record input/output

    def _get_record(self, index):
        if not (0 <= index and index * self.recordSize + self.recordSize <=
self.fm.Size()):
            return {
                'key': bytes([]),
                'id':  bytes([])
            }

        record = self.fm.ReadMany(index * self.recordSize, self.recordSize)

        return {
            'key': ZTStrUtils.GetFromBytesZT(record[:self.indexSize]),
            'id':  record[self.indexSize:]
        }

    def _set_record(self, index, key, id_):
        self.fm.WriteMany(
            index * self.recordSize,
            FitTo(key, self.indexSize) + FitTo(id_, self.idSize)
        )

    def _get_record_raw(self, index):
        return self.fm.ReadMany(index * self.recordSize, self.recordSize)

    def _set_record_raw(self, index, record):
        self.fm.WriteMany(index * self.recordSize, record)

    # is record/block used

    def _is_record_used(self, index):
        key = self._get_record(index)['key']

        return key != bytes()

    def _is_block_used(self, index):
        key = self._get_record(
            self._get_block_index_range(index)['start']
        )['key']

        return key != bytes()

    def _is_block_filled_out(self, index):
        return self._get_block_records_count(index) == self.rpb

    # block records/all blocks ranges

    def _get_block_index_by_record_index(self, index):
        return index // self.rpb * self.rpb

    def _get_record_index_by_block(self, index):
        return index * self.rpb

    def _get_block_value_range(self, index):
        range_ = self._get_block_index_range(index)

        if self._get_block_records_count(index) == 0:
            return {
                'min': bytes([]),
                'max': bytes([])
            }
```

```
        return {
            'min': self._get_record(range_['start'])['key'],
            'max': self._get_record(range_['end'  ])['key']
        }

    def _get_block_index_range(self, index):
        start = index * self.rpb

        l = start
        h = start + self.rpb - 1

        while True:
            if l in (h - 1, h):
                break

            m = (l + h) // 2

            if not self._is_record_used(m):
                h = m - 1
            else:
                l = m

        end = h if self._is_record_used(h) else (l if self._is_record_used(l)
else -1)

        return {'start': start, 'end': end}

    def _get_block_records_count(self, index):
        if index < 0:
            return 0

        range_ = self._get_block_index_range(index)

        return range_['end'] - range_['start'] + 1

    def _get_all_blocks_index_range(self):
        l = 0
        h = self.fm.Size() // (self.recordSize * self.rpb) - 1

        if h == -1:
            return {'start': 0, 'end': -1}

        while True:
            if l in (h - 1, h):
                break

            m = (l + h) // 2

            if not self._is_block_used(m):
                h = m - 1
            else:
                l = m

        end = h if self._is_block_used(h) else l

        return {'start': 0, 'end': end}

    def _blocks_count(self):
        end_index = self._get_all_blocks_index_range['end']

        return end_index + 1

    def _get_block_fillness(self, index):
        block_records_count = self._get_block_records_count(index)
```

```
        return block_records_count / self.rpb

    # advanced operations

    def _redistribute_after_remove(self, index):
        all_blocks_range = self._get_all_blocks_index_range()

        if not (index < all_blocks_range['end']):
            return

        if not (self._get_block_records_count(index) == 0):
            return

        next_block_index = index + 1
        while next_block_index <= all_blocks_range['end']:
            next_block_range = self._get_block_index_range(next_block_index)
            next_block_records = []

            for i in range(next_block_range['start'], next_block_range['end']
+ 1):
                record = self._get_record(i)

                key = record['key']
                id_ = record['id']

                next_block_records += [{'key': key, 'id': id_}]

            two_blocks_records_count =
DistributeTwoBlocksCounts(len(next_block_records))

            block_new_length = two_blocks_records_count['first']
            next_block_new_length = two_blocks_records_count['second']

            block_record_index = self._get_record_index_by_block(index)
            next_block_record_index =
self._get_record_index_by_block(next_block_index)

            for i in range(64):
                self._set_record(next_block_record_index + i, bytes([]),
bytes([]))

            i = 0
            for j in range(block_new_length):
                record = next_block_records[i]
                self._set_record(block_record_index + j, record['key'],
record['id'])

                i += 1

            for j in range(next_block_new_length):
                record = next_block_records[i]
                self._set_record(next_block_record_index + j, record['key'],
record['id'])

                i += 1

            next_block_index += 1
            index += 1
```

RecordIO.py:

```
import ZTStrUtils
```

```
import os

class RecordIO:
    recordSize = 128

    def __init__(self, filename):
        self.file = open(filename, 'r+b')

    def Add(self, data):
        end = self.getFileSize()

        self.file.write(bytes(self.recordSize))
        self.file.seek(end, os.SEEK_SET)
        self.file.write((data + '\x00').encode('ascii'))

        return end // self.recordSize

    def Get(self, id_):
        if not (id_ <= self.getFileSize() // self.recordSize):
            return None

        self.file.seek(self.recordSize * id_, os.SEEK_SET)

        return ZTStrUtils.GetFromBytesZT(self.file.read(128)).decode('ascii')

    def Set(self, id_, data):
        if not (id_ <= self.getFileSize() // self.recordSize):
            return None

        self.file.seek(self.recordSize * id_, os.SEEK_SET)
        self.file.write(
            (data[:self.recordSize - 1] + '\x00')
                .encode('ascii')
        )

    def Wipe(self):
        self.file.seek(0, os.SEEK_SET)
        self.file.truncate()

    def Sync(self):
        pass

    def getFileSize(self):
        self.file.seek(0, os.SEEK_END)
        return self.file.tell()
```

FileMapper.py:

```
import os

class FileMapper:
    def __init__(self, filename, cache_size=1024):
        self.file = open(filename, 'r+b')

        self.cache_valid = False
        self.cache_start = -1
        self.cache_end = -1
        self.cache_data = bytearray(cache_size)
        self.cache_size = cache_size

        self.file.seek(0, os.SEEK_END)
        self.length = self.file.tell()

    def _load_cache(self, index):
```

```
        self._write_cache()

        cache_size = self.cache_size

        aligned_index = self._aligned_index(index)

        self.cache_start = aligned_index
        self.cache_end = aligned_index + cache_size

        size_for_read = min(self.length, self.cache_end) - self.cache_start
        self.file.seek(aligned_index, os.SEEK_SET)
        read_data = self.file.read(size_for_read)

        for i in range(self.cache_size):
            self.cache_data[i] = 0

        for i, c in enumerate(read_data):
            self.cache_data[i] = c

        self.cache_valid = True

    def _write_cache(self):
        if not self.cache_valid:
            return

        self.file.seek(self.cache_start, os.SEEK_SET)
        self.file.write(
            self.cache_data[:min(self.cache_end, self.length) -
self.cache_start]
        )

    def _aligned_index(self, index):
        return index // self.cache_size * self.cache_size

    def WriteByte(self, index, value):
        self.length = max(self.length, index + 1)

        if not (self.cache_valid and self.cache_start <= index <
self.cache_end):
            self._load_cache(index)

        self.cache_data[index - self._aligned_index(index)] = value

    def ReadByte(self, index):
        if not (
                    self.cache_valid
                and self.cache_start <= index
                and (index < self.cache_end and index < self.length)
            ):
            self._load_cache(index)

        return self.cache_data[index - self._aligned_index(index)]

    def WriteMany(self, index, data):
        for offset, char in enumerate(data):
            self.WriteByte(index + offset, char)

    def ReadMany(self, index, size):
        data = bytearray(size)

        for data_index in range(size):
            data[data_index] = self.ReadByte(index + data_index)

        return data
```

```
    def WipeData(self):
        self.cache_valid = False
        self.length = 0

        self.file.seek(0, os.SEEK_SET)
        self.file.truncate()

    def Size(self):
        return self.length

    def __del__(self):
        if self.cache_valid:
            self._write_cache()

        self.file.close()
```

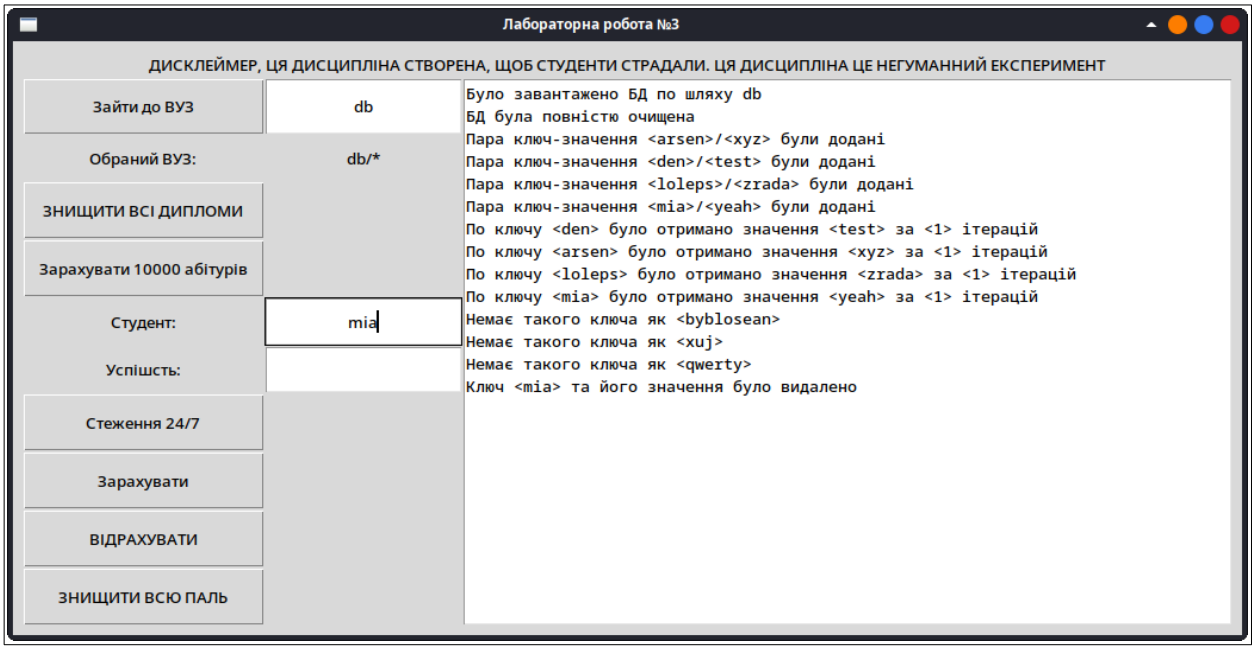
ZTStrUtils.py:

```
def GetToStrZT(bytes_, size):
    ln = len(bytes_)

    if not (ln < size - 1):
        return bytes_[:size]

    return bytes_ + bytes([0])

def GetFromBytesZT(bytes_):
    for i, c in enumerate(bytes_):
        if c == 0:
            break

    return bytes_[:i]
```

3.3.2 Приклади роботи

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

Рисунок 3.1 –Додавання запису



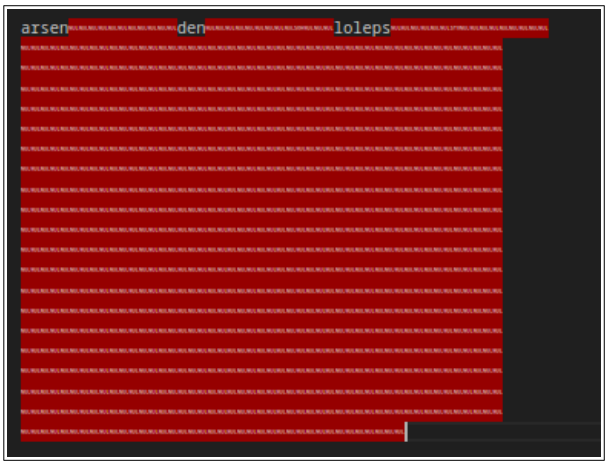
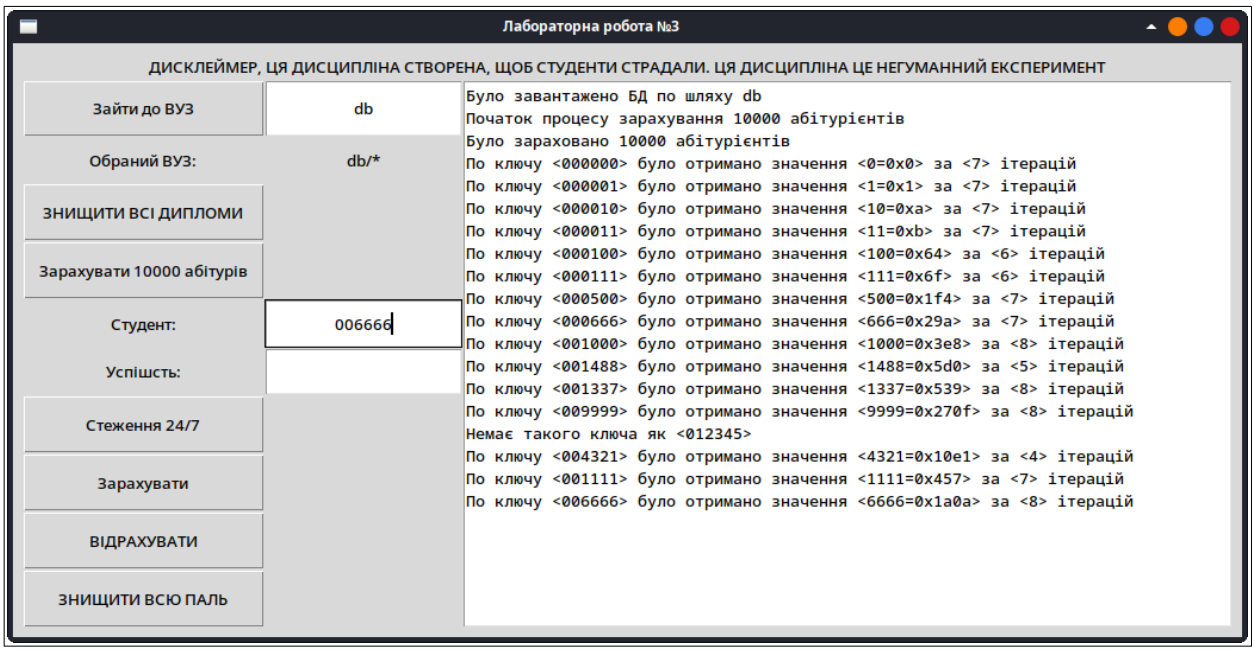


Рисунок 3.2 – Пошук запису


Лабораторна робота №3
ДИСКЛЕЙМЕР, ЦЯ ДИСЦИПЛІНА СТВОРЕНА, ЩОБ СТУДЕНТИ СТРАДАЛИ. ЦЯ ДИСЦИПЛІНА ЦЕ НЕГУМАННИЙ ЕКСПЕРИМЕНТ
Зайти до ВУЗ
db
Обраний ВУЗ:
db/*
ЗНИЩИТИ ВСІ ДИПЛОМИ
Зарахувати 10000 абітурів
Студент:
006666
Успішість:
Стеження 24/7
Зарахувати
ВІДРАХУВАТИ
ЗНИЩИТИ ВСЮ ПАЛЬ
Було завантажено БД по шляху db
Початок процесу зарахування 10000 абітурієнтів
Було зараховано 10000 абітурієнтів
По ключу <000000> було отримано значення <0=0x0> за <7> ітерацій
По ключу <000001> було отримано значення <1=0x1> за <7> ітерацій
По ключу <000010> було отримано значення <10=0xa> за <7> ітерацій
По ключу <000011> було отримано значення <11=0xb> за <7> ітерацій
По ключу <000100> було отримано значення <100=0x64> за <6> ітерацій
По ключу <000111> було отримано значення <111=0x6f> за <6> ітерацій
По ключу <000500> було отримано значення <500=0x1f4> за <7> ітерацій
По ключу <000666> було отримано значення <666=0x29a> за <7> ітерацій
По ключу <001000> було отримано значення <1000=0x3e8> за <8> ітерацій
По ключу <001488> було отримано значення <1488=0x5d0> за <5> ітерацій
По ключу <001337> було отримано значення <1337=0x539> за <8> ітерацій
По ключу <009999> було отримано значення <9999=0x270f> за <8> ітерацій
Немає такого ключа як <012345>
По ключу <004321> було отримано значення <4321=0x10e1> за <4> ітерацій
По ключу <001111> було отримано значення <1111=0x457> за <7> ітерацій
По ключу <006666> було отримано значення <6666=0x1a0a> за <8> ітерацій

3.4 Тестування алгоритму

3.4.1 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	Значення ключа	Число порівнянь
1	000000	7
2	000001	7
3	000010	7
4	000011	7
5	000100	6
6	000111	6
7	000500	7
8	000666	7
9	001000	8
10	001488	5
11	001337	8
12	009999	8
13	004321	4
14	001111	7
15	006666	8

Як бачимо, на пошук первинного ключа в індексному файлі витрачається лише від 4 до 8 в середньому ітерацій пошуку по диску та файлу. Це є достатньо ефективною кількістю ітерацій для 10000 елементів в базі данних.

Висновок

В рамках лабораторної роботи я нарешті зміг за весь цей час реалізувати програмну реалізацію, алгоритмувати, визначити часову складність файлів з індексами з перебудовою індексу з використанням бінарного пошуку для знаходження первинних ключів для знаходження даних в основному файлі з даними для цієї лабораторної роботи.

Я ще більш тісно познайомився з опитмізаціями в роботі з файлами, оптимізаціями доступу до диску. Було використано дуже ефективні алгоритми для доступу та запису до диску без лишніх опреацій вводу-виводу, а також залишити код чистим, високоструктурованим, малозв’язним і легким для читання програмістами.

Ця лабораторна робота заставила задуматися над тим, який багаж знань та зусиль були вкладені розробниками баз даних для їхньої ефективної роботи. Також ці оптимізації та алгоритми дають змогу зрозуміти, коли індекси є дуже ефективним інструментом в руках девелоперів та в яких ситуаціях.

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.