**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-12 Йолкін Даніїл Сергійович*

**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

## Програмна реалізація

### Вихідний код

import random  
import linecache  
import tkinter  
  
DB\_FILE\_PATH = "db.txt"  
INDEX\_FILE\_PATH = "dense\_index.txt"  
LZ = 178 # Bytes in record(approximation)  
LK = 4 # Bytes in int(in task)  
KZ = 1000 # Number of rows(in task)  
LB = 39\*100 # Bytes in a block  
number\_of\_lines = 1000  
window = tkinter.Tk()  
window.title("My app")  
result\_field = tkinter.Label(text="Results will be displayed there!")  
result\_field.grid(row=6, column=0)  
  
  
def build\_index\_file(db\_file\_path=DB\_FILE\_PATH):  
 # Index file row = key\_value; db\_row\_id; \n  
 with open(INDEX\_FILE\_PATH, "w") as index\_file:  
 for i in range(0, KZ):  
 with open(DB\_FILE\_PATH, "r") as file:  
 for j, content in enumerate(file):  
 value = int(content.split(" ")[0])  
 if i == value:  
 index\_file.write(str(i) + " " + str(j+1) + "\n")  
 return i  
  
  
def search(row\_id=None):  
 number\_of\_reads = 0  
 try:  
 if not row\_id:  
 primary\_key = int(entry\_input\_field.get())  
 else:  
 primary\_key = row\_id  
 except:  
 result\_field.configure(text=f"Incorrect format given!")  
 return  
 # Using binary search to find item  
 upper\_index = number\_of\_lines  
 lower\_index = 0  
 while lower\_index <= upper\_index:  
 middle\_index = (upper\_index + lower\_index) // 2  
 value = linecache.getline(INDEX\_FILE\_PATH, middle\_index + 1)  
 number\_of\_reads += 1  
 if not value:  
 result\_field.configure(text=f"Item was not found! \n Number of reads: {str(number\_of\_reads)}")  
 return "Item not found"  
 if int(value.split(" ")[0]) == primary\_key:  
 result\_field.configure(text=f"Row in DB was found! \n Row details: {linecache.getline(DB\_FILE\_PATH, int(value.split(' ')[1]))} \n Number of reads: {str(number\_of\_reads)}")  
 return value  
 elif middle\_index < primary\_key:  
 lower\_index = middle\_index + 1  
 else:  
 upper\_index = middle\_index - 1  
 result\_field.configure(text=f"Item was not found! \n Number of reads: {str(number\_of\_reads)}")  
 return "Item not found"  
  
  
def add():  
 global number\_of\_lines  
 row = str(number\_of\_lines) + " " + " ".join(entry\_input\_field.get().split(" ")) + "\n"  
 with open(DB\_FILE\_PATH, "a+") as file:  
 file.write(row)  
 file.close()  
 with open(INDEX\_FILE\_PATH, "a+") as file1:  
 file1.write(row.split(" ")[0] + " " + str(number\_of\_lines + 1) + "\n")  
 file1.close()  
 linecache.clearcache()  
 number\_of\_lines += 1  
 result\_field.configure(text=f"Data was successfully added!")  
 return  
  
  
def update():  
 try:  
 global number\_of\_lines  
 data = entry\_input\_field.get().split(" ")  
 row\_id = int(data[0])  
 value = str(data[1])  
 index\_value = int(search(row\_id).split(" ")[1].replace("\n", ""))  
 with open(DB\_FILE\_PATH, "r+") as file1:  
 lines = []  
 for index, line in enumerate(file1):  
 if index == index\_value-1:  
 lines.append(str(row\_id) + " " + value + "\n")  
 else:  
 lines.append(line)  
 file1.write("")  
 file1.close()  
 with open(DB\_FILE\_PATH, "w") as file1:  
 file1.writelines(lines)  
 print("DB file was successfully updated!")  
 result\_field.configure(text="DB file was successfully updated!")  
 linecache.clearcache()  
 return "Success!"  
 except:  
 result\_field.configure(text="Incorrect format")  
  
  
def delete():  
 global number\_of\_lines  
 try:  
 row\_id = int(entry\_input\_field.get())  
 search\_result = search(row\_id).split(" ")  
 index\_value1 = int(search\_result[0])  
 index\_value2 = int(search\_result[1].replace("\n", ""))  
 with open(INDEX\_FILE\_PATH, "r+") as file:  
 lines = []  
 for index, line in enumerate(file):  
 if int(line.split(" ")[0]) == index\_value1:  
 lines.append("")  
 else:  
 lines.append(line)  
 file.write("")  
 with open(INDEX\_FILE\_PATH, "w") as file:  
 file.writelines(lines)  
 with open(DB\_FILE\_PATH, "r+") as file1:  
 lines = []  
 for index, line in enumerate(file1):  
 if index == index\_value2-1:  
 lines.append("\n")  
 else:  
 lines.append(line)  
 file1.write("")  
 with open(DB\_FILE\_PATH, "w") as file1:  
 file1.writelines(lines)  
 print("Row was successfully deleted!")  
 result\_field.configure(text=f"Row was successfully deleted!")  
 linecache.clearcache()  
 number\_of\_lines -= 1  
 return "Success!"  
 except:  
 result\_field.configure(text=f"Incorrect format given or incorrect row!")  
 return  
  
  
# LZ - length of a record in file  
# LK - size of a key  
# KZ - number of records in file  
# LB - size of a block  
def generate\_db\_file():  
 random\_data = []  
 letters = "abcdefghijklmnopqrstuvwxyz"  
 for i in range(0, KZ):  
 random\_data.append((str(i)  
 + " "  
 + "".join([letters[random.randint(0, 25)] for j in range(0, LB//100 + 1)]))[0:(LB//100)-1]  
 + "\n")  
 random.shuffle(random\_data)  
  
 with open(DB\_FILE\_PATH, "w") as file:  
 for record in random\_data:  
 file.write(record)  
  
  
def print\_hi(name):  
 # Use a breakpoint in the code line below to debug your script.  
 print(f'Hi, {name}') # Press ⌘F8 to toggle the breakpoint.  
  
  
# Press the green button in the gutter to run the script.  
if \_\_name\_\_ == '\_\_main\_\_':  
 generate\_db\_file()  
 number\_of\_lines = build\_index\_file() + 1  
 add\_label = tkinter.Label(text="Add element(format: string data without key)")  
 delete\_label = tkinter.Label(text="Delete element(format: key\_value of element)")  
 search\_label = tkinter.Label(text="Search element(format: key\_value of element)")  
 update\_label = tkinter.Label(text="Update element(format: key\_value + string updated data separated by space)")  
 entry\_input\_field = tkinter.Entry()  
 add\_button = tkinter.Button(text="Add", command=add)  
 delete\_button = tkinter.Button(text="Delete", command=delete)  
 search\_button = tkinter.Button(text="Search", command=search)  
 update\_button = tkinter.Button(text="Update", command=update)  
 add\_button.grid(row=1, column=1)  
 add\_label.grid(row=1, column=0)  
 delete\_button.grid(row=2, column=1)  
 delete\_label.grid(row=2, column=0)  
 search\_button.grid(row=3, column=1)  
 search\_label.grid(row=3, column=0)  
 update\_button.grid(row=4, column=1)  
 update\_label.grid(row=4, column=0)  
 entry\_input\_field.grid(row=5, column=0)  
 window.mainloop()

### Приклади роботи

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

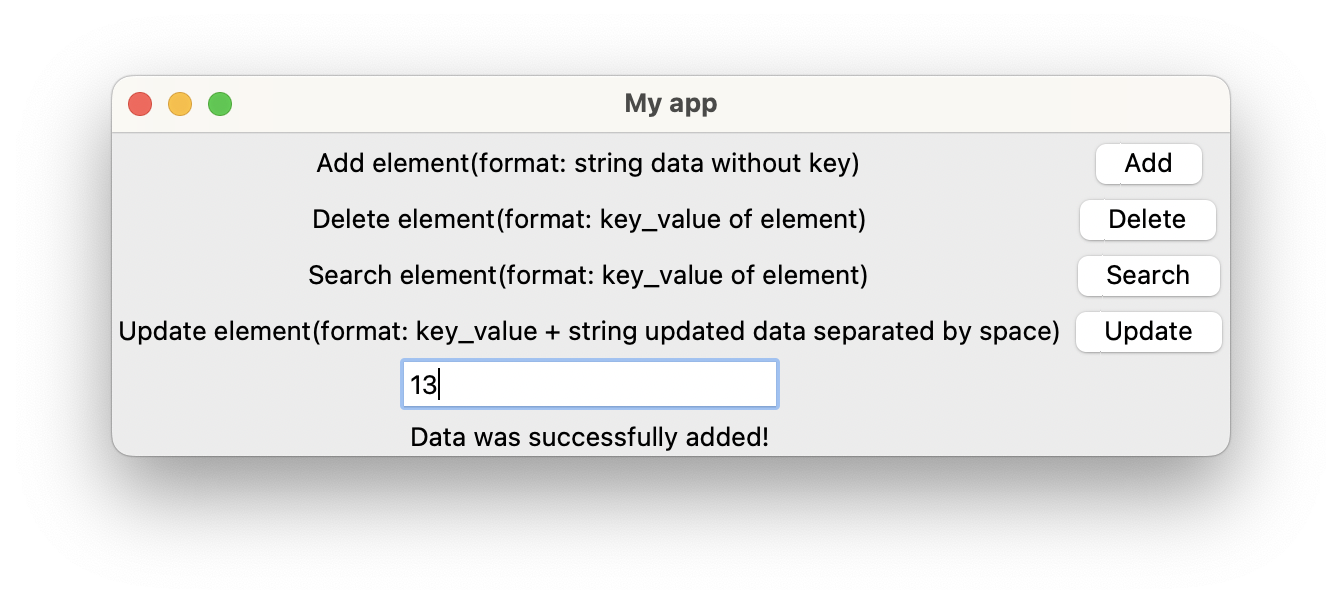


Рисунок 3.1 –Додавання запису

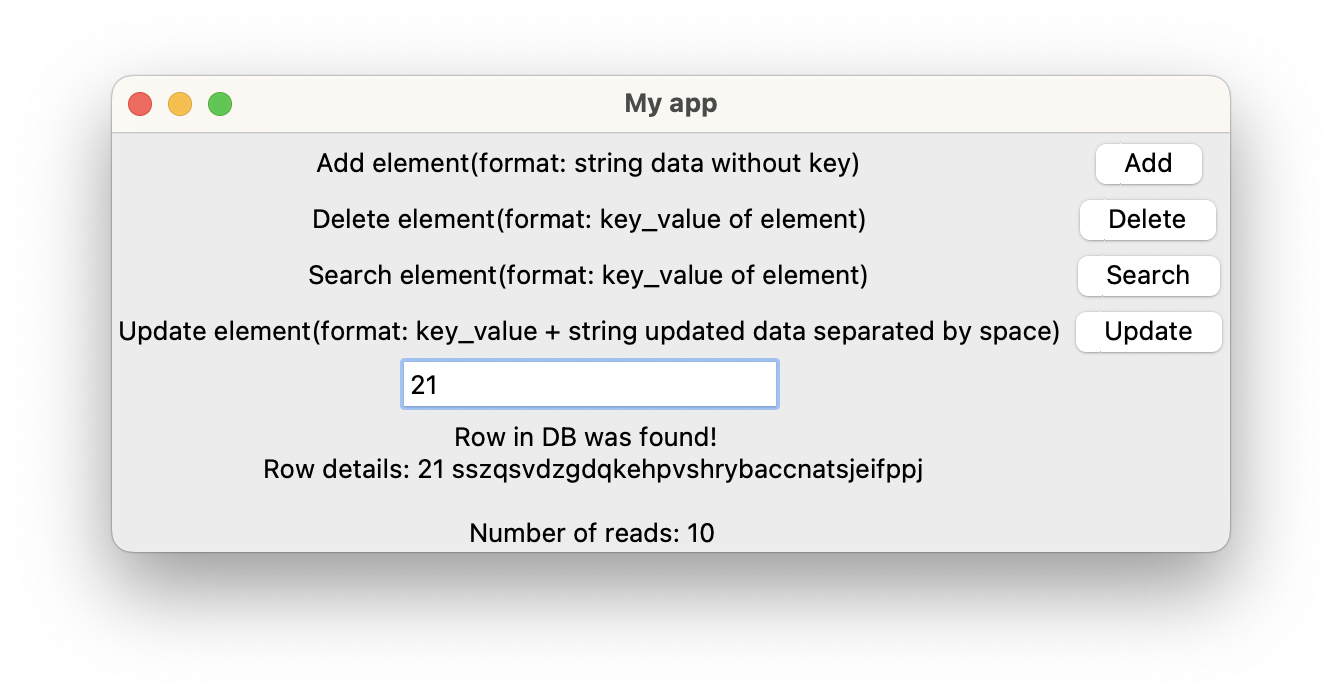


Рисунок 3.2 – Пошук запису

## 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 (обраний ключ 64)

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

Висновок

В рамках лабораторної роботи було програмно реалізовано файл з щільним індексом та методи для роботи з цим файлом.

Було проведено тестування з часовими оцінками пошуку. Можемо бачити, що алгоритм працює за логарифмічний час доступу до диску, що є оптимальним часом.

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.