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»
Варіант 11
„ Проектування структур даних”

Виконав(ла) *ІП-15 Кондрацької Соні* _____
(шифр, прізвище, ім'я, по батькові)

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

Київ 2023

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи проектування 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 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

FUNCTION binary_search(search_key):

OPEN index_file in read mode and assign it to i_file

mas=i_file.readlines()

CUR_IND=int(size//2) +1

SHIFT =int(size / 2)

WHILE SHIFT!= 0

SEEK to CUR_IND in i_file and read the next line

SPLIT the current line into key and offset

CONVERT key to integer and assign it to key

CONVERT offset to integer and assign it to offset

IF key is equal to search_key

RETURN offset

ELSE IF key is greater than search_key

CUR_IND=CUR_IND-(int(SHIFT/2)+1)

SHIFT = int(SHIFT/2)

ELSE

CUR_IND = CUR_IND+ (int(SHIFT/2)+1)

SHIFT = int(SHIFT/2)

RETURN None

FUNCTION insert(record):

SPLIT the record into search_key and data by ', '

CONVERT search_key to integer

ASSIGN the result of binary_search(search_key) to offset

IF offset is None

OPEN data_file in read mode and assign it to d_file
SEEK to the end of d_file

ASSIGN the current position of d_file to offset

OPEN data_file in append mode and assign it to d_file_w
OPEN index_file in append mode and assign it to i_file_w
WRITE a new line and record to d_file_w
SEEK to the end of i_file_w
CREATE index_entry as search_key + ', ' + offset
WRITE index_entry to i_file_w
FLUSH the buffer of i_file_w

FUNCTION delete(search_key):
ASSIGN the result of binary_search(search_key) to offset
IF offset is not None:
OPEN data_file in write mode and assign it to d_file
OPEN index_file in read mode and assign it to i_file
SEEK to the offset in d_file
OPEN data_file in read mode and assign it to d_file_r
WRITE '*' with the length of the first line of d_file_r to d_file
SEEK to the beginning of i_file
READ ALL lines of i_file and assign it to index_data
SEEK to the beginning of i_file
OPEN index_file in write mode and assign it to i_file_w
TRUNCATE i_file_w
FOR EACH line in index_data
IF search_key is not in line
OPEN index_file in write mode and assign it to i_file_w
WRITE line to i_file_w
FLUSH the buffer of i_file

FUNCTION update(search_key, new_data):

ASSIGN the result of binary_search(search_key) to offset

OPEN data_file in read and write mode and assign it to d_file IF offset is not None:

SEEK to the offset in d_file

READ the first line of d_file and assign it to current_line

SPLIT current_line into key and data by ', '

CREATE new_line as key + ', ' + new_data + '\n'

SEEK to the offset in d_file

OPEN data_file in read and write mode and assign it to d_file

FOR EACH line in fileinput.input(data_file)

WRITE line.replace(current_line, new_line) to d_file

3.2 Часова складність пошуку

Однорідний бінарний пошук є різновидом/модифікацією до звичайного бінарного. Часова складність наданого алгоритму становить O(log n). Його називають бінарним пошуком, оскільки він багаторазово ділить вхідні дані методом золотого перетину(відношення довжини більшого відрізка до довжини всього інтервалу = відношенню довжини меншого відрізка до довжини більшого відрізка) і перевіряє середній елемент. У кожній ітерації він порівнює середній елемент із ключем пошуку. Якщо ключ дорівнює середньому елементу, він повертає індекс середнього елемента. Якщо ключ менший за середній елемент, він повторює процес у лівій частині вхідних даних, інакше він повторює процес у правій частині. Він продовжує цей процес, доки ключ не буде знайдено або пошуковий простір не стане порожнім. Оскільки вхідні дані зменшуються ± вдвічі під час кожної ітерації, максимальна кількість ітерацій, необхідних для пошуку ключа, становить log n, де n — розмір вхідних даних. Таким чином, часова складність алгоритму бінарного пошуку становить O(log n). Варто зазначити, що ця складність передбачає, що дані були попередньо відсортовані, а операція порівняння між середнім елементом і ключем пошуку є операцією постійного часу O(1).

3.3 Програмна реалізація

3.3.1 Вихідний код

main.py

```
import tkinter
import customtkinter
from DenseIndexes import *

customtkinter.set_appearance_mode("System")
customtkinter.set_default_color_theme("dark-blue")


class App(customtkinter.CTk):
    def __init__(self):
        super().__init__()

        # configure window
        self.title("Algo | Lab 3")
        self.geometry(f"{300}x{500}")
        self.resizable(False, False)

        self.action = tkinter.StringVar(value="")

        self.pack_widgets()

        self.di = DenseIndex("data.txt", "index.txt")
        self.di.build_index()

    def pack_widgets(self):
        self.key_value_frame = customtkinter.CTkFrame(self, corner_radius=50)
        self.key_value_frame.grid(row=0, column=0, columnspan=2, sticky="ew")

        self.key_label = customtkinter.CTkLabel(self.key_value_frame, text="Your KEY")
        self.key_label.grid(row=0, column=0, padx=40)

        self.value_label = customtkinter.CTkLabel(self.key_value_frame, text="Your VALUE")
        self.value_label.grid(row=0, column=1, padx=40)

        self.key_value_entry_frame = customtkinter.CTkFrame(self, corner_radius=50)
        self.key_value_entry_frame.grid(row=1, column=0, columnspan=2, sticky="ew")

        self.key_entry = customtkinter.CTkEntry(self.key_value_entry_frame, placeholder_text="input your key here...")
        self.key_entry.grid(row=1, column=0)

        self.value_entry = customtkinter.CTkEntry(self.key_value_entry_frame,
                                                  placeholder_text="input your value here...")
        self.value_entry.grid(row=1, column=1)

        self.action_frame = customtkinter.CTkFrame(self)
        self.action_frame.grid(row=2, column=0, padx=10, pady=5, columnspan=2, sticky="nsew")

        self.radio_insert = customtkinter.CTkRadioButton(self.action_frame, variable=self.action,
                                                         value="insert", text="insert")
        self.radio_insert.grid(row=0, column=0, columnspan=2, pady=10, padx=20, sticky="we")

        self.radio_delete = customtkinter.CTkRadioButton(self.action_frame, variable=self.action,
                                                         value="delete", text="delete")
        self.radio_delete.grid(row=1, column=0, columnspan=2, pady=10, padx=20, sticky="we")

        self.radio_update = customtkinter.CTkRadioButton(self.action_frame, variable=self.action,
                                                         value="update", text="update")
```

```
        self.radio_update.grid(row=2, column=0, columnspan=2, pady=10, padx=20, sticky="we")

        self.radio_find = customtkinter.CTkRadioButton(self.action_frame, variable=self.action,
                                                       value="find", text="find")
        self.radio_find.grid(row=3, column=0, columnspan=2, pady=10, padx=20, sticky="we")

        self.execute_button = customtkinter.CTkButton(self, text="Execute", command=self.execute)
        self.execute_button.grid(row=3, column=0, columnspan=2, sticky="we")

        self.info_area = customtkinter.CTkTextbox(self)
        self.info_area.insert("0.0", "Loading...")
        self.info_area.configure(state="disabled")
        self.info_area.grid(row=4, column=0, columnspan=2, padx=5, pady=5, sticky="snwe")

    @staticmethod
    def clear_area(area):
        area.configure(state="normal")
        area.delete("1.0", customtkinter.END)
        area.configure(state="disabled")

    def insert_area(self, area, to_insert):
        self.clear_area(area)
        area.configure(state="normal")

        area.insert("0.0", to_insert)
        area.configure(state="disabled")

    def execute(self):
        action_value = self.action.get()
        self.clear_area(self.info_area)
        if action_value:
            if action_value == "insert":
                key_to_insert = self.key_entry.get()
                value_to_insert = self.value_entry.get()

                record_to_insert = f"{key_to_insert}, {value_to_insert}"
                res=self.di.insert(record_to_insert)
                self.di.build_index()
                if res ==1:
                    self.insert_area(self.info_area,
                                     f"Record with key {key_to_insert} was inserted with value {value_to_insert}")
                else:
                    self.insert_area(self.info_area,
                                     f"Record with key {key_to_insert} is already exist")
            elif action_value == "delete":
                key_to_delete = int(self.key_entry.get())
                res=self.di.delete(key_to_delete)

                self.di.build_index()
                if res ==1:
                    self.insert_area(self.info_area,
                                     f"Founded record with key {key_to_delete} was deleted")
                else:
                    self.insert_area(self.info_area,
                                     f"Not founded record with key {key_to_delete}")
            elif action_value == "update":
                key_to_update = int(self.key_entry.get())
                value_to_update = self.value_entry.get()

                res=self.di.update(key_to_update, value_to_update)
                self.di.build_index()

                if res ==1:
                    self.insert_area(self.info_area,
                                     f"The value with {key_to_update} was updated by new value {value_to_update}")
                else:
```

```
                self.insert_area(self.info_area,
                                 f"Not founded record with key {key_to_update}")

        elif action_value == "find":
            key_to_find = int(self.key_entry.get())
            print(key_to_find)

            found_value = self.di.search(key_to_find)
            print(found_value)
            if found_value:
                self.insert_area(self.info_area, "The founded record is:\n"
                                                 f"{found_value}")
            else:
                self.insert_area(self.info_area,
                                 f"Not founded record with key {key_to_find}")
        else:
            self.insert_area(self.info_area, "Pick the option firstly!")


if __name__ == "__main__":
    app = App()
    app.mainloop()
```

DenseIndex.py

```
import fileinput
import os

class DenseIndex:
    def __init__(self, data_file, index_file):
        self.data_file = data_file
        self.index_file = index_file

    def build_index(self):
        mas=[]
        with open(self.index_file, 'w') as i_file:
            with open(self.data_file, 'r') as d_file:
                d_file.seek(0)
                i_file.seek(0)
                i_file.truncate()
                current_offset = 0
                for line in d_file:
                    search_key, data = line.strip().split(',')
                    search_key = int(search_key)

                    index_entry = f"{search_key}, {current_offset}\n"
                    current_offset += len(line) + 1
                    mas.append(index_entry)
                    key = index_entry.split(',')[0]
                mas.sort(key=lambda x: x.split(',')[0])
                for i in mas: i_file.write(i)

                i_file.flush()

    def binary_search(self, search_key):
        with open(self.index_file, 'r') as i_file:
            i_file.seek(0)
            mas = i_file.readlines()
            left = 0
            right = len(mas) - 1

        while left <= right:
            mid =int((right - left)/1.619031)+left

            current_line = mas[mid]
            key, offset = current_line.split(',')

            key = int(key)
            offset = int(offset)
```

```
        print(f"current pointer at", offset)
        print(f"our key is", key)

        if key == search_key:
            return offset

        elif key > search_key:
            right = mid - 1

        else:
            left = mid + 1
    return None


def search(self, search_key):
    offset = self.binary_search(search_key)

    with open(self.data_file, 'r') as d_file:
        if offset is not None:
            d_file.seek(offset)
            return d_file.readline()
        else:
            return None

def insert(self, record):
    search_key, data = record.strip().split(',')
    search_key = int(search_key)
    offset = self.binary_search(search_key)

    if offset is None:
        with open(self.data_file, 'r') as d_file:
            d_file.seek(0, 2)
            offset = d_file.tell()

        with open(self.data_file, 'a') as d_file_w:
            with open(self.index_file, 'a') as i_file_w:
                d_file_w.write(record+'\n')
                i_file_w.seek(0, 2)

                index_entry = f"{search_key},{offset}\n"
                i_file_w.write(index_entry)
                i_file_w.flush()
        return 1
    else:
        print(f"Record with key {search_key} is exist.")
        return 0

def delete(self, search_key):
    offset = self.binary_search(search_key)
    if offset is not None:
        with open(self.data_file, 'r+') as d_file:
            d_file.seek(offset)
            del_line = d_file.readline()
        with fileinput.FileInput(self.data_file, inplace=True, backup='.bak') as d_file:
            for line in d_file:
                if line != del_line:
                    print(line, end='')
        os.unlink(self.data_file + '.bak')
        return 1
    else:
        print(f"Record with key {search_key} not found.")
        return 0

def update(self, search_key, new_data):
    offset = self.binary_search(search_key)
    if offset is not None:
        with open(self.data_file, 'r+') as d_file:
```

```
            d_file.seek(offset)
            old_line = d_file.readline()
            key, data = old_line.strip().split(',')
            new_line = f"{key}, {new_data}\n"
        with fileinput.FileInput(self.data_file, inplace=True, backup='.bak') as d_file:
            for line in d_file:
                if line == old_line:
                    print(new_line, end='')
                else:
                    print(line, end='')
        os.unlink(self.data_file + '.bak')
        return 1
    else:
        print(f"Record with key {search_key} not found.")
        return 0
```

3.3.2 Приклади роботи

На рисунках 3.1 - 3.4 показані приклади роботи програми для додавання і пошуку запису.

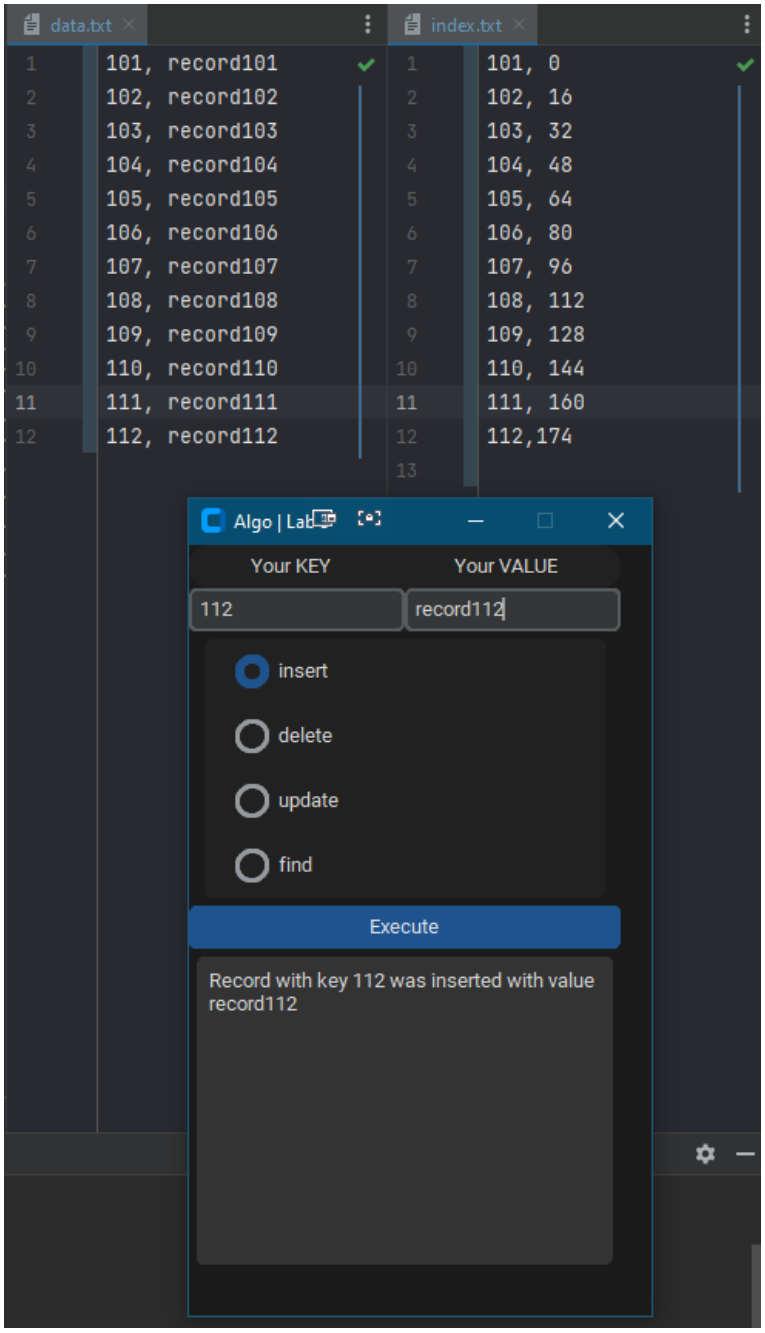

Рисунок 3.1 –Додавання запису

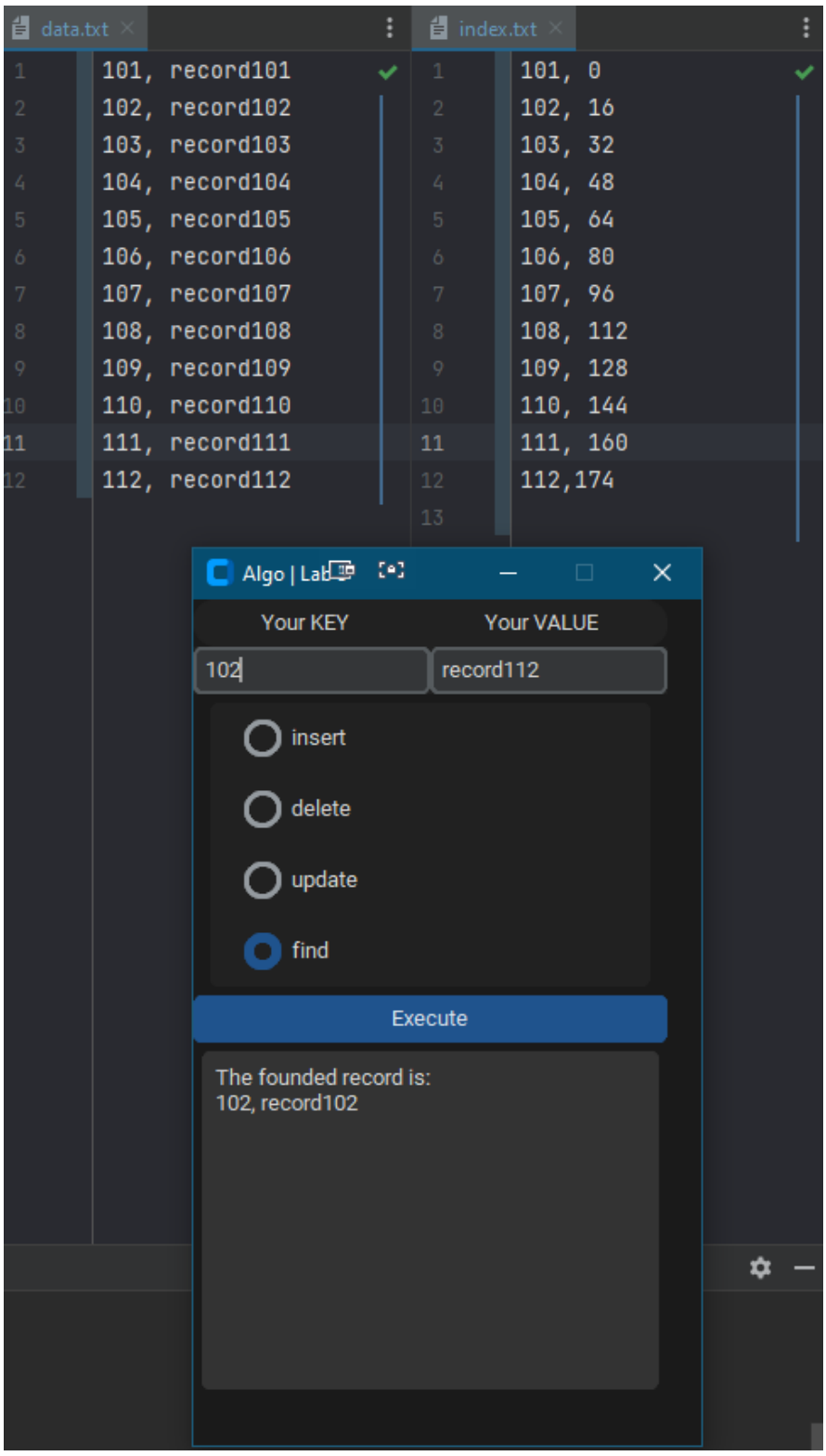

Рисунок 3.2 – Пошук запису

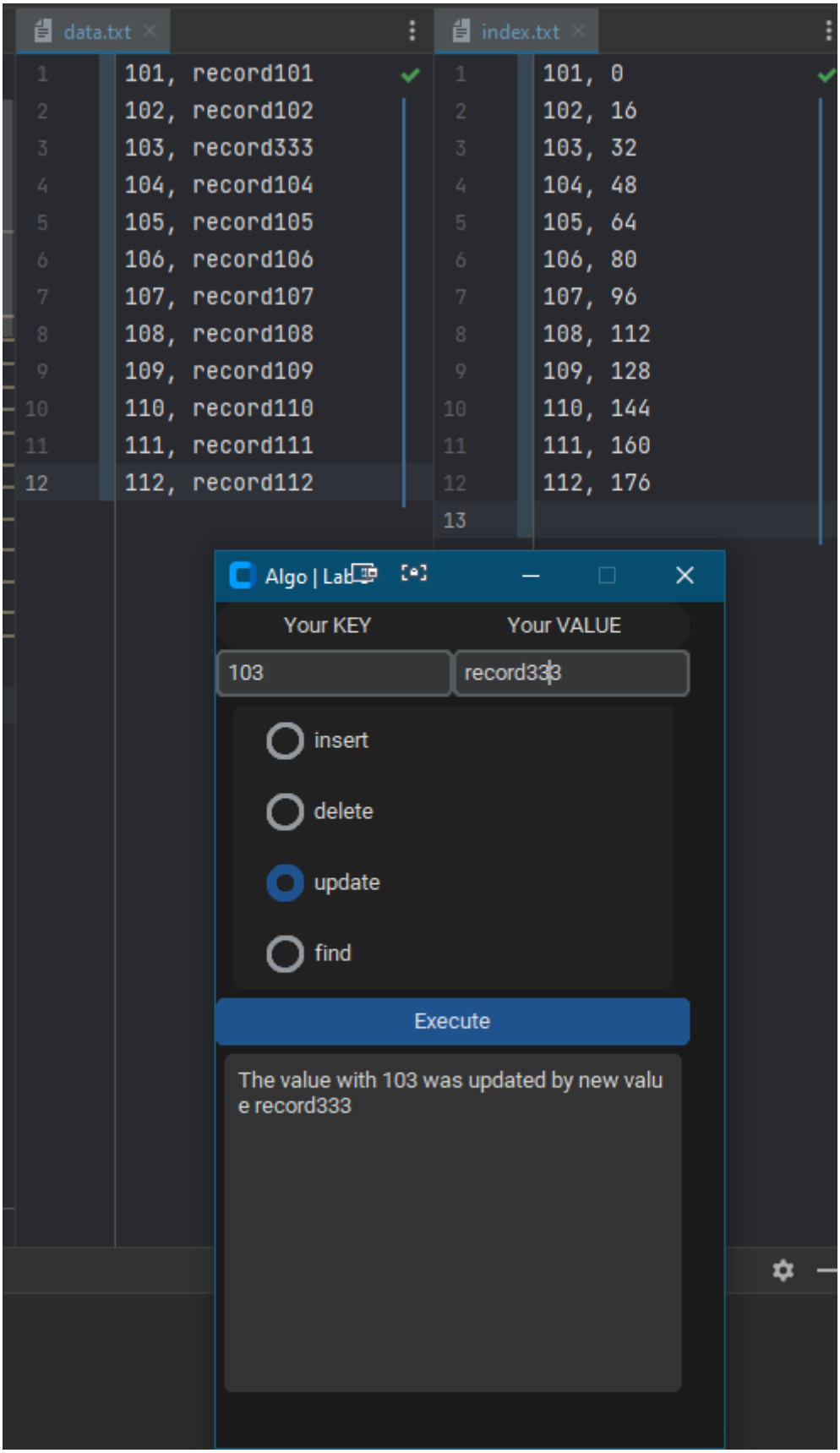

Рисунок 3.3 – Оновлення запису

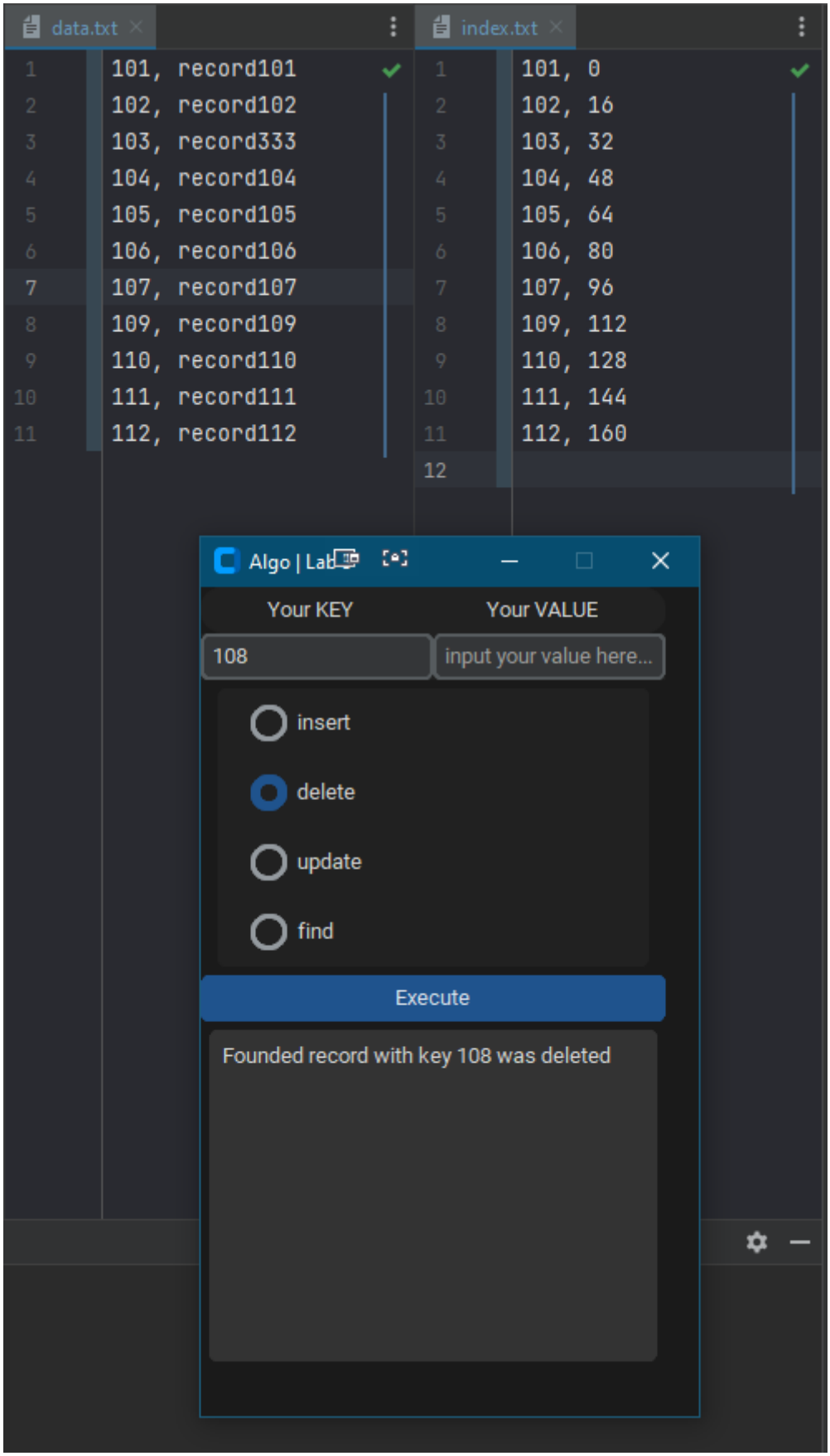

Рисунок 3.4 – Видалення запису

3.4 Тестування алгоритму

3.4.1 Часові характеристики оцінювання

Максимальна кількість порівнянь може бути представлена як степінь двійки, але так як даних в межах роботи мало знаходиться, то макс $2^3 = 8$.

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

ВИСНОВОК

В рамках лабораторної роботи було реалізовано структуру даних файли зі щільним індексом.

Підсумовуючи, метод файлу щільного індексування є ефективним способом пошуку даних у великих файлах даних. Завдяки створенню файлу індексу, який містить запис для кожного значення ключа пошуку у файлі даних, пошук даних стає швидшим.

Індексний файл містить ключ пошуку та вказівник на фактичний запис у файлі даних. Однак цей метод вимагає більше місця для зберігання записів індексу. У цій лабораторній роботі ми реалізували метод файлу щільного індексування в Python, створивши клас DenseIndex, який включає такі функції, як пошук, вставка, оновлення та видалення. Усі ці функції використовують бінарний алгоритм пошуку для ефективного пошуку записів у файлі даних.

Часова складність алгоритму бінарного пошуку становить O(log n), що є ефективним при роботі з великими файлами даних.

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.