**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-13 Крупосій Вадим Сергійович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О.О.*

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

## Псевдокод алгоритмів

1. **Binary Search**

binarySearch(arr\_sparse, x)

{

arr = make\_sparse\_dense()

repeat till left <= high

mid = left + (right - left)/2

if (x == arr[mid])

return arr[mid]

else if (x > arr[mid])

left = mid + 1

else

right = mid – 1

}

1. **Sparse Array**

Make\_dense\_sparse()

{

indexes\_dense = get\_all\_indexes(dense\_array);

maximum = find\_max(indexes\_dense);

for (int i =0; i<=maximum;i++){

if(indexes\_dense.contains(i)) {

sparse\_array.add(dense\_array.get(indexes\_dense.indexOf(i)));

int amount\_of\_i = find\_amount\_of\_value(indexes\_dense,i);

if(amount\_of\_i>1)

{

for(int j =1; j<amount\_of\_i;j++) {

dense\_array.set(value more than max);

indexes\_dense = get\_all\_indexes(dense\_array);

overflowing\_bucket.add(value); } } }

else

sparse\_array.add(null); }}

1. **Main.java**

initialise frame

amount\_of\_records = 100

write file with random input

get input from file as dense array

make sparse array from dense

create\_frame()

## Часова складність пошуку

O(log n)

## Програмна реалізація

### Вихідний код

**BinarySearch.java**

|  |
| --- |
| package db; |
|  |  |
|  | import java.util.ArrayList; |
|  |  |
|  |  |
|  | public class BinarySearch { |
|  |  |
|  | public static db\_input binarySearch(ArrayList<db\_input> arr\_base, int x) |
|  | { |
|  | int counter = 0; |
|  | ArrayList<db\_input> arr = SparseArray.make\_sparse\_dense(arr\_base); |
|  | int l = 0, r = arr.size() - 1; |
|  | while (l <= r) { |
|  | int index = l + (r - l) / 2; |
|  |  |
|  | // Check if x is present at mid |
|  | if (arr.get(index).index == x) { |
|  | System.out.println("Found with " + counter + " steps"); |
|  | return arr.get(index); |
|  | } |
|  |  |
|  | // If x greater, ignore left half |
|  | if (arr.get(index).index < x) { |
|  | l = index + 1; |
|  | } |
|  |  |
|  | // If x is smaller, ignore right half |
|  | else { |
|  | r = index - 1; |
|  | } |
|  | counter++; |
|  | } |
|  |  |
|  | // if we reach here, then element was |
|  | // not present |
|  | return null; |
|  | } |
|  | } |

**SparseArray.java**

|  |
| --- |
| package db; |
|  |  |
|  | import java.util.ArrayList; |
|  |  |
|  | public class SparseArray { |
|  |  |
|  | static ArrayList<db\_input> answer = new ArrayList<>(); |
|  |  |
|  | static ArrayList<db\_input> dense\_array = new ArrayList<>(); |
|  | public static ArrayList<db\_input> sparse\_array = new ArrayList<>(); |
|  |  |
|  | static int amount\_of\_records = 100; |
|  |  |
|  | public SparseArray(ArrayList<db\_input> dense\_array, ArrayList<db\_input> sparse\_array) |
|  | { |
|  | SparseArray.dense\_array = dense\_array; |
|  | SparseArray.sparse\_array = sparse\_array; |
|  | } |
|  |  |
|  | public static void setAnswer(ArrayList<db\_input>arr) |
|  | { |
|  | answer = new ArrayList<>(); |
|  | answer.addAll(arr); |
|  | } |
|  |  |
|  | public static ArrayList<db\_input> getSparseArray() |
|  | { |
|  | return sparse\_array; |
|  | } |
|  |  |
|  | public static ArrayList<db\_input> getOverflowBucket() |
|  | { |
|  | return answer; |
|  | } |
|  |  |
|  | public static void make\_dense\_sparse() |
|  | { |
|  | ArrayList<Integer> indexes\_dense = get\_all\_indexes(dense\_array); |
|  | int maximum = arr\_work.find\_max\_arr\_list(indexes\_dense); |
|  | for (int i =0; i<=maximum;i++) { |
|  | if(indexes\_dense.contains(i)) { |
|  | if (sparse\_array.size() >= amount\_of\_records) { |
|  | answer.add(dense\_array.get(indexes\_dense.indexOf(i))); |
|  | } else sparse\_array.add(dense\_array.get(indexes\_dense.indexOf(i))); |
|  | } |
|  | } |
|  |  |
|  |  |
|  | } |
|  |  |
|  | public static ArrayList<db\_input> getAnswer() |
|  | { |
|  | return answer; |
|  | } |
|  |  |
|  | public static ArrayList<Integer> get\_all\_indexes(ArrayList<db\_input> dense\_array) |
|  | { |
|  | ArrayList<Integer>answer = new ArrayList<>(); |
|  | for (db.db\_input db\_input : dense\_array) { |
|  | answer.add(db\_input.index); |
|  | } |
|  |  |
|  | return answer; |
|  |  |
|  | } |
|  |  |
|  | public static ArrayList<db\_input> make\_sparse\_dense(ArrayList<db\_input>sparse) |
|  | { |
|  | ArrayList<db\_input> dense = new ArrayList<>(); |
|  | for(db\_input i : sparse) |
|  | if(i != null) |
|  | dense.add(i); |
|  | return dense; |
|  | } |
|  | } |

**WriterReader.java**

|  |
| --- |
| package db; |
|  |  |
|  | import java.io.\*; |
|  | import java.util.ArrayList; |
|  | import java.util.Random; |
|  |  |
|  | public class WriterReader { |
|  | static final String pathname = "lab3\_db.obj"; |
|  | static int amount\_of\_records; |
|  |  |
|  | public static void setAmount\_of\_records(int am) throws IOException { |
|  | amount\_of\_records = am; |
|  | save\_size(); |
|  | } |
|  |  |
|  | public static int get\_size\_of\_file() throws IOException { |
|  | BufferedReader reader = new BufferedReader(new FileReader("sz\_o\_file.txt")); |
|  | String line = reader.readLine(); |
|  | int size = Integer.parseInt(line); |
|  | reader.close(); |
|  | return size; |
|  | } |
|  |  |
|  | public static void save\_size () throws IOException { |
|  | File sz = new File("sz\_o\_file.txt"); |
|  | FileWriter fileWriter = new FileWriter(sz); |
|  | fileWriter.write(Integer.toString(amount\_of\_records)); |
|  | fileWriter.close(); |
|  |  |
|  | } |
|  |  |
|  | public static void main(int amount\_input) { |
|  | try { |
|  | FileOutputStream f = new FileOutputStream(pathname); |
|  | ObjectOutputStream o = new ObjectOutputStream(f); |
|  | ArrayList<Integer> get\_indexes = new ArrayList<>(); |
|  | int counter =0; |
|  | for (int i = 0; i < amount\_input\*2; i++) { |
|  | Random value = new Random(); |
|  | int rand\_index = value.nextInt(amount\_input + i); |
|  | if(!get\_indexes.contains(rand\_index)) { |
|  | db\_input input = new db\_input(rand\_index, value.nextInt(10000)); |
|  | get\_indexes.add(rand\_index); |
|  | // Write objects to file |
|  | o.writeObject(input); |
|  | counter++; |
|  | } |
|  | } |
|  | setAmount\_of\_records(counter); |
|  | save\_size(); |
|  | o.close(); |
|  | f.close(); |
|  | } catch (FileNotFoundException e) { |
|  | System.out.println("File not found"); |
|  | } catch (IOException e) { |
|  | System.out.println("Error initializing stream"); |
|  | } |
|  | } |
|  |  |
|  | public static ArrayList<db\_input> read\_file() throws IOException, ClassNotFoundException { |
|  | setAmount\_of\_records(get\_size\_of\_file()); |
|  | FileInputStream fi = new FileInputStream(pathname); |
|  | ObjectInputStream oi = new ObjectInputStream(fi); |
|  | ArrayList<db\_input> get\_input = new ArrayList<>(); |
|  | for(int i =0; i<amount\_of\_records;i++) get\_input.add(i, (db\_input) oi.readObject()); |
|  | oi.close(); |
|  | fi.close(); |
|  | return get\_input; |
|  | } |
|  |  |
|  | public static void write\_to\_db(ArrayList<db\_input> new\_input) throws IOException { |
|  | new FileWriter(pathname, false).close(); |
|  | try { |
|  | FileOutputStream f = new FileOutputStream(pathname); |
|  | ObjectOutputStream o = new ObjectOutputStream(f); |
|  | for (db\_input input : new\_input) o.writeObject(input); |
|  | o.close(); |
|  | f.close(); |
|  | } catch (FileNotFoundException e) { |
|  | System.out.println("File not found"); |
|  | } catch (IOException e) { |
|  | System.out.println("Error initializing stream"); |
|  | } |
|  | } |
|  | } |

**Input\_for\_db.java**

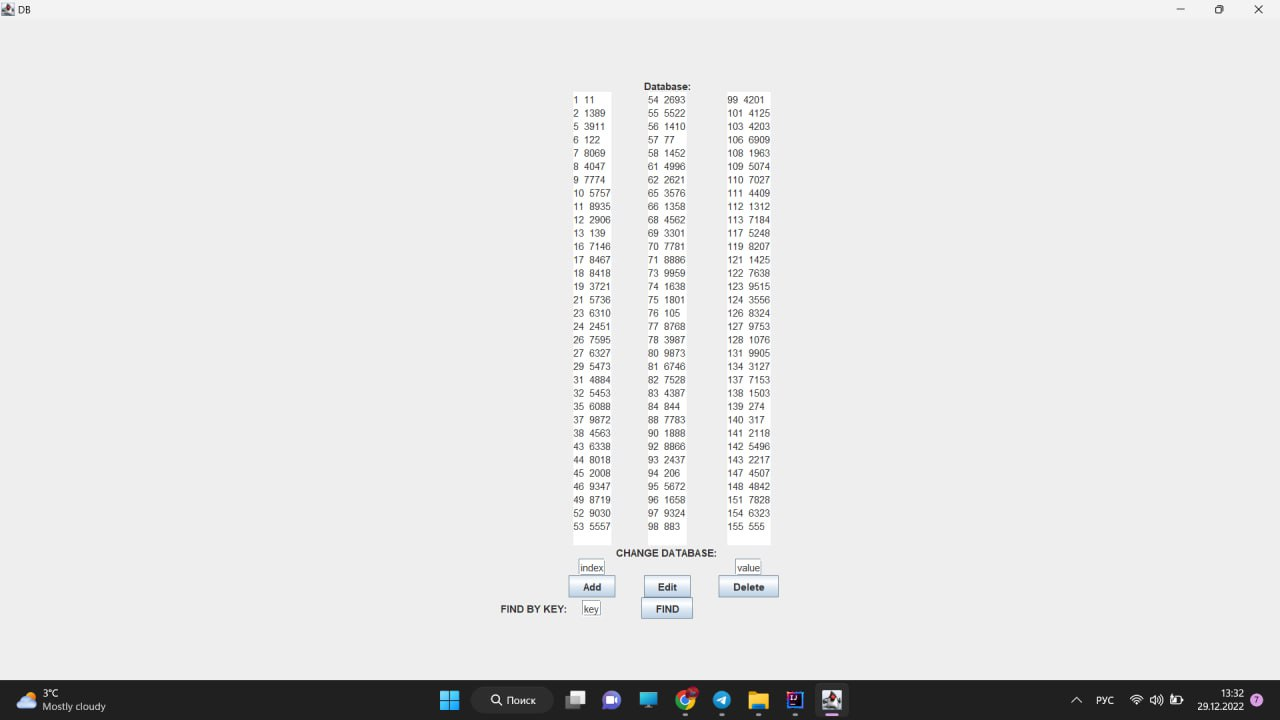
|  |
| --- |
| package db; |
|  |  |
|  | import java.io.Serializable; |
|  |  |
|  | public class db\_input implements Serializable { |
|  |  |
|  | int index; |
|  | int value; |
|  |  |
|  | public db\_input(int index, int value) |
|  | { |
|  | this.index = index; |
|  | this.value = value; |
|  | } |
|  |  |
|  | public String toString() { |
|  | return index + " " + value; |
|  | } |
|  |  |
|  | public int getIndex() |
|  | { |
|  | return index; |
|  | } |
|  |  |
|  | public int getValue() |
|  | { |
|  | return value; |
|  | } |
|  |  |
|  | } |

**Work\_with\_array.java**

|  |
| --- |
| Package db; |
|  |  |
|  | import java.util.ArrayList; |
|  |  |
|  | public class arr\_work { |
|  |  |
|  | public static void print\_arr\_list(ArrayList<db\_input> arr) |
|  | { |
|  | for (db.db\_input db\_input : arr) |
|  | { |
|  | if(db\_input!=null) System.out.println(db\_input); |
|  | } |
|  | } |
|  |  |
|  | public static ArrayList<Integer> get\_indexes(ArrayList<db\_input>arr) |
|  | { |
|  | ArrayList<Integer>indexes = new ArrayList<>(); |
|  | for (db.db\_input db\_input : arr) if (db\_input != null) indexes.add(db\_input.getIndex()); |
|  | return indexes; |
|  | } |
|  |  |
|  | public static ArrayList<Integer> get\_values(ArrayList<db\_input>arr) |
|  | { |
|  | ArrayList<Integer> indexes = new ArrayList<>(); |
|  | for (db.db\_input db\_input : arr) if (db\_input != null) indexes.add(db\_input.getValue()); |
|  | return indexes; |
|  | } |
|  |  |
|  | public static int find\_max\_arr\_list(ArrayList<Integer> arr) |
|  | { |
|  | int maximum = arr.get(0); |
|  | for (int i = 1; i < arr.size(); i++) { |
|  | if (maximum < arr.get(i)) |
|  | maximum = arr.get(i); |
|  | } |
|  | return maximum; |
|  | } |
|  |  |
|  | public static String make\_array\_text (ArrayList<db\_input>arr) |
|  | { |
|  | StringBuilder answer = new StringBuilder(); |
|  | for(db\_input i:arr) { |
|  | if(i!=null) { |
|  | answer.append(i); |
|  | answer.append(System.lineSeparator()); |
|  | } |
|  | } |
|  | return answer.toString(); |
|  | } |
|  |  |
|  |  |
|  | public static ArrayList<db\_input> divide\_array\_into\_three(ArrayList<db\_input>arr,int part) |
|  | { |
|  | arr = SparseArray.make\_sparse\_dense(arr); |
|  | ArrayList<db\_input> answer = new ArrayList<>(); |
|  | if(part == 1) |
|  | { |
|  | for(int i =0;i<arr.size()/3;i++) |
|  | { |
|  | answer.add(arr.get(i)); |
|  | } |
|  | } |
|  | else if(part==2) { |
|  | for (int i = arr.size() / 3; i < 2 \* arr.size() / 3; i++) { |
|  | answer.add(arr.get(i)); |
|  | } |
|  |  |
|  | } |
|  | else{ |
|  | for (int i = 2\* arr.size() / 3; i < arr.size(); i++) { |
|  | answer.add(arr.get(i)); |
|  | } |
|  | } |
|  | return answer; |
|  | } |
|  |  |
|  | } |

### Приклади роботи

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



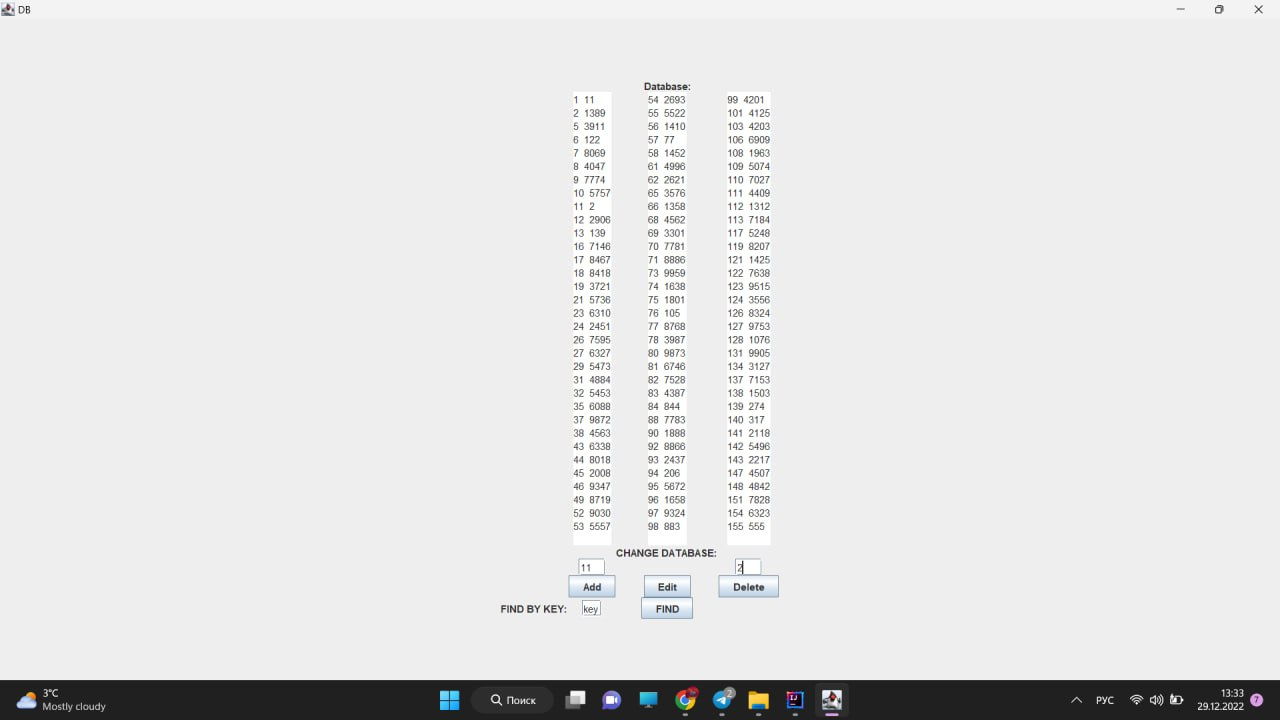


Рисунок 3.1 –Додавання запису

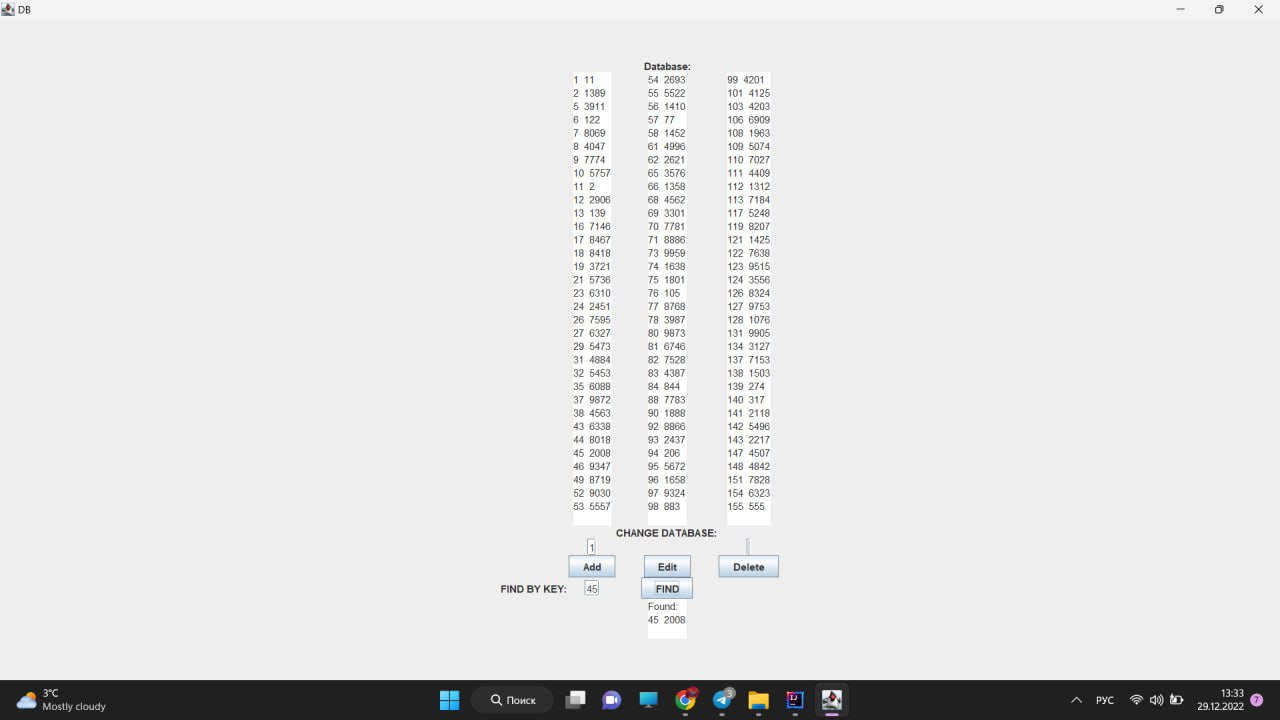


Рисунок 3.2 – Пошук запису

## Тестування алгоритму

### Часові характеристики оцінювання

В таблиці 3.1 наведено кількість порівнянь для 5 спроб пошуку запису по ключу.

Таблиця 3.1 – Число порівнянь при спробі пошуку запису по ключу

|  |  |
| --- | --- |
| Номер спроби пошуку | Число порівнянь |
| 1 | 5 |
| 2 | 3 |
| 3 | 3 |
| 4 | 5 |
| 5 | 5 |

Висновок

В ході даної лабораторної роботи було вивчено основні підходи проектування та обробки складних структур даних. Було реалізовано операції додавання, видалення та пошуку для бази даних. Було використано однорідний бінарний пошук. Створено файл з НЕ щільним індексом (sparse), який в даній реалізації забезпечує швидке та правильне сортування, що полегшує роботу з базою даних. Вихідні дані записуються у файл з об’єктами класу.

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.