

**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)**

**ІП-13 Крупосій Вадим Сергійович**  
(шифр, прізвище, ім'я, по батькові)

**Перевірів**

**Сопов О.О.**  
(прізвище, ім'я, по батькові)

Київ 2022

## ЗМІСТ

<b>1</b>	<b>МЕТА ЛАБОРАТОРНОЇ РОБОТИ .....</b>	<b>3</b>
<b>2</b>	<b>ЗАВДАННЯ .....</b>	<b>4</b>
<b>3</b>	<b>ВИКОНАННЯ.....</b>	<b>7</b>
	3.1 ПСЕВДОКОД АЛГОРИТМІВ.....	7
	3.2 ЧАСОВА СКЛАДНІСТЬ ПОШУКУ .....	7
	3.3 ПРОГРАМНА РЕАЛІЗАЦІЯ .....	8
	3.3.1 Вихідний код .....	8
	3.3.2 Приклади роботи .....	14
	3.4 ТЕСТУВАННЯ АЛГОРИТМУ .....	17
	3.4.1 Часові характеристики оцінювання.....	17
	<b>ВИСНОВОК .....</b>	<b>20</b>
	<b>КРИТЕРІЇ ОЦІНЮВАННЯ .....</b>	<b>21</b>

## 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	В-дерево $t=10$ , бінарний пошук
8	В-дерево $t=25$ , бінарний пошук
9	В-дерево $t=50$ , бінарний пошук
10	В-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	В-дерево $t=10$ , однорідний бінарний пошук
18	В-дерево $t=25$ , однорідний бінарний пошук
19	В-дерево $t=50$ , однорідний бінарний пошук
20	В-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	В-дерево $t=10$ , метод Шарра
28	В-дерево $t=25$ , метод Шарра

29	В-дерево $t=50$ , метод Шарра
30	В-дерево $t=100$ , метод Шарра
31	АВЛ-дерево
32	Червоно-чорне дерево
33	В-дерево $t=250$ , бінарний пошук
34	В-дерево $t=250$ , однорідний бінарний пошук
35	В-дерево $t=250$ , метод Шарра

## 3.1 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
}

```

**2. 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);  }}

```

### 3. 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()

```

#### 3.2 Часова складність пошуку

$O(\log n)$

#### 3.3 Програмна реалізація

##### 3.3.1 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;
}
}

```

### 3.3.2 Приклади роботи

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

D8

111

21389

53911

6122

78069

84047

97774

105757

118835

122906

13139

167146

178467

188418

193721

215736

236310

242451

267595

276327

295473

314884

325453

356088

379872

384563

436338

448018

452008

469347

498719

529030

535557

Database:

542693

555522

561410

5777

581452

614996

622621

653576

661358

684562

693301

707781

718886

739859

741638

751801

76105

778768

783887

809873

816746

827528

834387

84844

887783

901888

928866

932437

94206

955672

961658

979324

98883

994201

1014125

1034203

1065909

1081963

1095074

1107027

1114409

1121312

1137184

1175248

1198207

1211425

1227638

1239515

1243556

1268324

1279753

1281076

1319905

1343127

1377153

1381503

139274

140317

1412118

1425496

1432217

1474507

1484842

1517828

1546323

155555

CHANGE DATABASE:

Index

value

Add

Edit

Delete

FIND BY KEY:

key

FIND

3°C

Mostly cloudy

Почк

29.12.2022

13:32

D8

111

21389

53911

6122

78069

84047

97774

105757

112

122906

13139

167146

178467

188418

193721

215736

236310

242451

267595

276327

295473

314884

325453

356088

379872

384563

436338

448018

452008

469347

498719

529030

535557

Database:

542693

555522

561410

5777

581452

614996

622621

653576

661358

684562

693301

707781

718886

739859

741638

751801

76105

778768

783887

809873

816746

827528

834387

84844

887783

901888

928866

932437

94206

955672

961658

979324

98883

994201

1014125

1034203

1065909

1081963

1095074

1107027

1114409

1121312

1137184

1175248

1198207

1211425

1227638

1239515

1243556

1268324

1279753

1281076

1319905

1343127

1377153

1381503

139274

140317

1412118

1425496

1432217

1474507

1484842

1517828

1546323

155555

CHANGE DATABASE:

11

Add

Edit

Delete

FIND BY KEY:

key

FIND

3°C

Mostly cloudy

Почк

29.12.2022

13:33

Рисунок 3.1 – Додавання запису

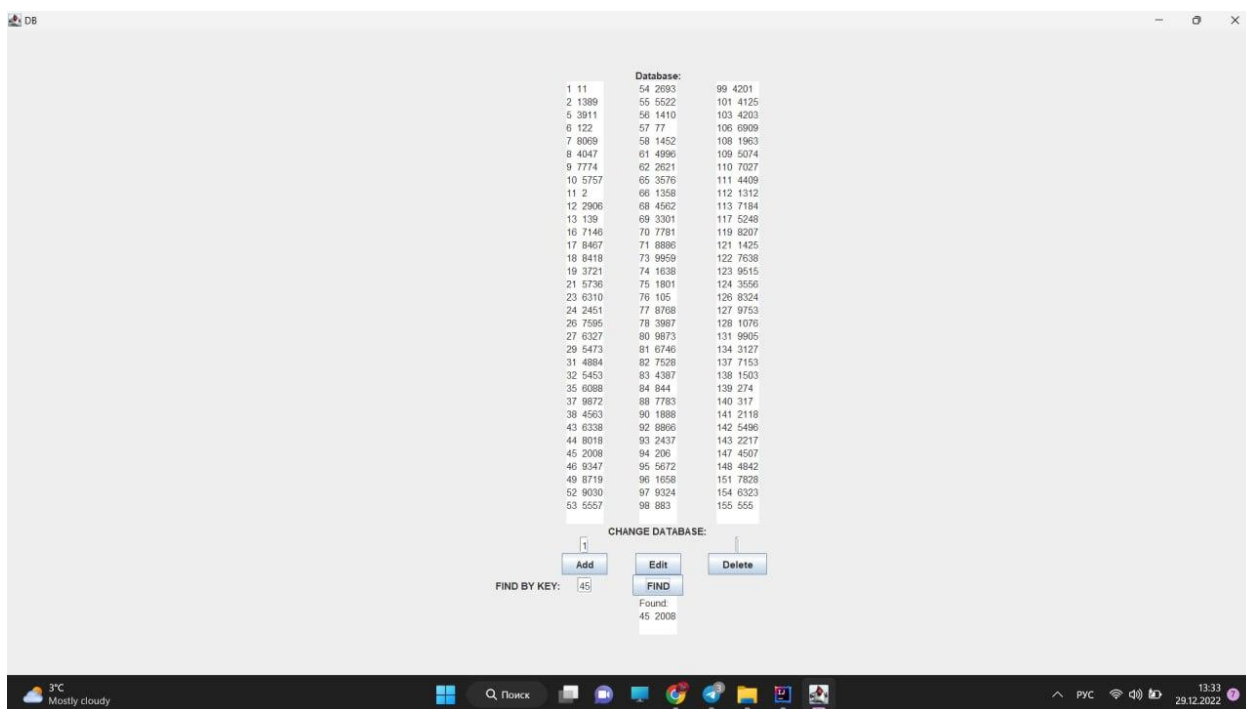


Рисунок 3.2 – Пошук запису

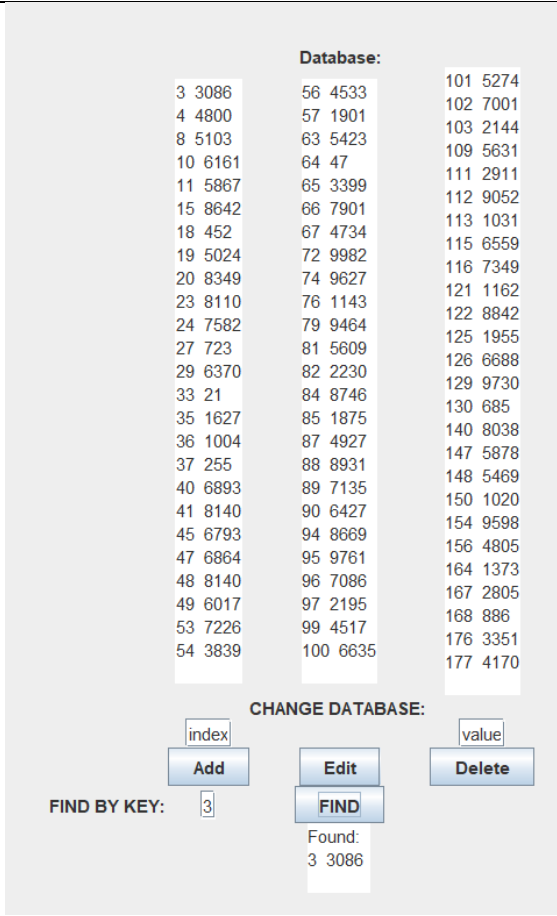


### 3.4 Тестування алгоритму

#### 3.4.1 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
<div>1</div>  <p>The screenshot shows a database application window. At the top, it says 'Database:'. Below this is a list of 177 records, each with an index and a value. The records are arranged in three columns. The first column contains indices from 3 to 54, the second from 56 to 100, and the third from 101 to 177. Below the list, there is a section titled 'CHANGE DATABASE:' with three buttons: 'Add', 'Edit', and 'Delete'. To the left of these buttons is a 'FIND BY KEY:' label and a text input field containing the number '3'. To the right of the buttons is a 'value' label and a text input field. Below the 'FIND' button, it says 'Found: 3 3086'.</p>	5

2

Database:

3 3086	56 4533	101 5274
4 4800	57 1901	102 7001
8 5103	63 5423	103 2144
10 6161	64 47	109 5631
11 5867	65 3399	111 2911
15 8642	66 7901	112 9052
18 452	67 4734	113 1031
19 5024	72 9982	115 6559
20 8349	74 9627	116 7349
23 8110	76 1143	121 1162
24 7582	79 9464	122 8842
27 723	81 5609	125 1955
29 6370	82 2230	126 6688
33 21	84 8746	129 9730
35 1627	85 1875	130 685
36 1004	87 4927	140 8038
37 255	88 8931	147 5878
40 6893	89 7135	148 5469
41 8140	90 6427	150 1020
45 6793	94 8669	154 9598
47 6864	95 9761	156 4805
48 8140	96 7086	164 1373
49 6017	97 2195	167 2805
53 7226	99 4517	168 886
54 3839	100 6635	176 3351
		177 4170

CHANGE DATABASE:

index

Add

Edit

FIND

value

Delete

FIND BY KEY: 125

Found:  
125 1955

3

3

Database:

3 3086	56 4533	101 5274
4 4800	57 1901	102 7001
8 5103	63 5423	103 2144
10 6161	64 47	109 5631
11 5867	65 3399	111 2911
15 8642	66 7901	112 9052
18 452	67 4734	113 1031
19 5024	72 9982	115 6559
20 8349	74 9627	116 7349
23 8110	76 1143	121 1162
24 7582	79 9464	122 8842
27 723	81 5609	125 1955
29 6370	82 2230	126 6688
33 21	84 8746	129 9730
35 1627	85 1875	130 685
36 1004	87 4927	140 8038
37 255	88 8931	147 5878
40 6893	89 7135	148 5469
41 8140	90 6427	150 1020
45 6793	94 8669	154 9598
47 6864	95 9761	156 4805
48 8140	96 7086	164 1373
49 6017	97 2195	167 2805
53 7226	99 4517	168 886
54 3839	100 6635	176 3351
		177 4170

CHANGE DATABASE:

index

Add

Edit

FIND

value

Delete

FIND BY KEY: 72

Found:  
72 9982

3

Database:

3 3086

4 4800

8 5103

10 6161

11 5867

15 8642

18 452

19 5024

20 8349

23 8110

24 7582

27 723

29 6370

33 21

35 1627

36 1004

37 255

40 6893

41 8140

45 6793

47 6864

48 8140

49 6017

53 7226

54 3839

56 4533

57 1901

63 5423

64 47

65 3399

66 7901

67 4734

72 9982

74 9627

76 1143

79 9464

81 5609

82 2230

84 8746

85 1875

87 4927

88 8931

89 7135

90 6427

94 8669

95 9761

96 7086

97 2195

99 4517

100 6635

101 5274

102 7001

103 2144

109 5631

111 2911

112 9052

113 1031

115 6559

116 7349

121 1162

122 8842

125 1955

126 6688

129 9730

130 685

140 8038

147 5878

148 5469

150 1020

154 9598

156 4805

164 1373

167 2805

168 886

176 3351

177 4170

CHANGE DATABASE:

index

Add

Edit

Delete

value

FIND BY KEY:

103

FIND

Found:

103 2144

4

5

Database:

3 3086

4 4800

8 5103

10 6161

11 5867

15 8642

18 452

19 5024

20 8349

23 8110

24 7582

27 723

29 6370

33 21

35 1627

36 1004

37 255

40 6893

41 8140

45 6793

47 6864

48 8140

49 6017

53 7226

54 3839

56 4533

57 1901

63 5423

64 47

65 3399

66 7901

67 4734

72 9982

74 9627

76 1143

79 9464

81 5609

82 2230

84 8746

85 1875

87 4927

88 8931

89 7135

90 6427

94 8669

95 9761

96 7086

97 2195

99 4517

100 6635

101 5274

102 7001

103 2144

109 5631

111 2911

112 9052

113 1031

115 6559

116 7349

121 1162

122 8842

125 1955

126 6688

129 9730

130 685

140 8038

147 5878

148 5469

150 1020

154 9598

156 4805

164 1373

167 2805

168 886

176 3351

177 4170

CHANGE DATABASE:

index

Add

Edit

Delete

value

ID BY KEY:

56

FIND

Found:

56 4533

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.