**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-14 Радзівіло Валерія Артемівна*

**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

|  |  |
| --- | --- |
| 14 | Файли з не щільним індексом з областю переповнення, однорідний бінарний пошук |

# 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

Main.java

import db.\*;  
  
import javax.swing.\*;  
import java.io.IOException;  
import java.util.ArrayList;  
  
import static interface\_for\_db.db\_interface.*create\_frame*;  
  
public class Main {  
 public static void main(String[] args) throws IOException, ClassNotFoundException {  
 JFrame frame= new JFrame("DB");  
 int amount\_of\_records = 100;  
 WriterReader.*main*(amount\_of\_records);  
 ArrayList<db\_input> get\_from\_db = WriterReader.*read\_file*(amount\_of\_records);  
 ArrayList<db\_input> sparse\_get\_from\_db = new ArrayList<>();  
  
 SparseArray sparseArray = new SparseArray(get\_from\_db,sparse\_get\_from\_db);  
  
 SparseArray.*make\_dense\_sparse*();  
 System.*out*.println("DB:");  
 arr\_work.*print\_arr\_list*(sparse\_get\_from\_db);  
  
 System.*out*.println("Overflow:");  
 arr\_work.*print\_arr\_list*(sparseArray.*getOverflowing\_bucket*());  
  
 ArrayList<Integer> indexes = arr\_work.*get\_indexes*(sparse\_get\_from\_db);  
 ArrayList<Integer> values = arr\_work.*get\_values*(sparse\_get\_from\_db);  
 *create\_frame*(frame,indexes,values,sparseArray);  
  
  
  
 }  
  
  
}

arr\_work.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.toString());  
 }  
 }  
  
 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 int find\_amount\_of\_value(ArrayList<Integer>arr, int value)  
 {  
 int count = 0;  
 for(int i =0;i <arr.size();i++)  
 {  
 if(arr.get(i)==value)  
 count++;  
 }  
 return count;  
 }  
  
 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.toString());  
 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;  
 }  
  
}

BinarySearch.java

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

db\_input.java

package db;  
  
import java.io.Serializable;  
import java.util.ArrayList;  
  
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;  
 }  
  
}

SparseArray.java

package db;  
  
import java.util.ArrayList;  
  
public class SparseArray {  
  
 static ArrayList<db\_input> *overflowing\_bucket* = new ArrayList<>();  
  
 static ArrayList<db\_input> *dense\_array* = new ArrayList<>();  
 static ArrayList<db\_input> *sparse\_array* = new ArrayList<>();  
  
  
 public SparseArray(ArrayList<db\_input> dense\_array, ArrayList<db\_input> sparse\_array)  
 {  
 SparseArray.*dense\_array* = dense\_array;  
 SparseArray.*sparse\_array* = sparse\_array;  
 }  
  
 public SparseArray(){ }  
  
 public static void setOverflowing\_bucket(ArrayList<db\_input>arr)  
 {  
 *overflowing\_bucket* = new ArrayList<>();  
 *overflowing\_bucket*.addAll(arr);  
 }  
  
 public static ArrayList<db\_input> getSparseArray()  
 {  
 return *sparse\_array*;  
 }  
  
 public static ArrayList<db\_input> getOverflowBucket()  
 {  
 return *overflowing\_bucket*;  
 }  
  
 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)) {  
 *sparse\_array*.add(*dense\_array*.get(indexes\_dense.indexOf(i)));  
 int amount\_of\_i =arr\_work.*find\_amount\_of\_value*(indexes\_dense,i);  
 if(amount\_of\_i>1)  
 {  
 for(int j =1; j<amount\_of\_i;j++) {  
 *dense\_array*.set(indexes\_dense.indexOf(i),new db\_input(maximum+1,maximum+1));  
 indexes\_dense = *get\_all\_indexes*(*dense\_array*);  
 *overflowing\_bucket*.add(*dense\_array*.get(indexes\_dense.indexOf(i)));  
 }  
 }  
  
 }  
 else  
 *sparse\_array*.add(null);  
  
 }  
 }  
  
 public static ArrayList<db\_input> getOverflowing\_bucket()  
 {  
 return *overflowing\_bucket*;  
 }  
  
 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";  
  
 public static String get\_filepath()  
 {  
 return *pathname*;  
 }  
  
  
 public static void main(int amount\_input) {  
  
  
 try {  
 FileOutputStream f = new FileOutputStream(new File(*pathname*));  
 ObjectOutputStream o = new ObjectOutputStream(f);  
  
  
 for(int i =0; i<amount\_input;i++)  
 {  
 Random value = new Random();  
 db\_input input = new db\_input(value.nextInt(amount\_input+i), value.nextInt(10000));  
 // Write objects to file  
 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");  
 }  
  
 }  
  
 public static ArrayList<db\_input> read\_file(int amount) throws IOException, ClassNotFoundException {  
 FileInputStream fi = new FileInputStream(new File(*pathname*));  
 ObjectInputStream oi = new ObjectInputStream(fi);  
  
 // Read objects  
 ArrayList<db\_input> get\_input = new ArrayList<>();  
  
  
 for(int i =0; i< amount;i++)  
 {  
 get\_input.add(i,(db\_input) oi.readObject());  
  
 }  
  
  
 oi.close();  
 fi.close();  
  
 return get\_input;  
 }  
  
 public static ArrayList<db\_input> deserealised\_read()  
 {  
 ArrayList<db\_input> emp = new ArrayList<>();  
 try  
 {  
 FileInputStream fileIn =new FileInputStream(*pathname*);  
 ObjectInputStream in = new ObjectInputStream(fileIn);  
 int i = 0;  
 while (in.readObject()!=null) {  
 emp.add(i,(db\_input)in.readObject());  
 i++;  
  
 }  
 in.close();  
 fileIn.close();  
 }catch(IOException | ClassNotFoundException i)  
 {  
 i.printStackTrace();  
 }  
  
 return emp;  
 }  
  
 public static void add\_to\_db(db\_input additional\_input)  
 {  
 try {  
 FileOutputStream f = new FileOutputStream(new File(*pathname*));  
 ObjectOutputStream o = new ObjectOutputStream(f);  
  
  
 db\_input input = additional\_input;  
 // Write objects to file  
 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");  
 }  
 }  
  
 static void clean\_file() throws IOException {  
 new FileWriter(*pathname*, false).close();  
 }  
  
  
 public static void write\_to\_db(ArrayList<db\_input> new\_input) throws IOException {  
 *clean\_file*();  
 try {  
 FileOutputStream f = new FileOutputStream(new File(*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");  
 }  
 }  
}

### Приклади роботи

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

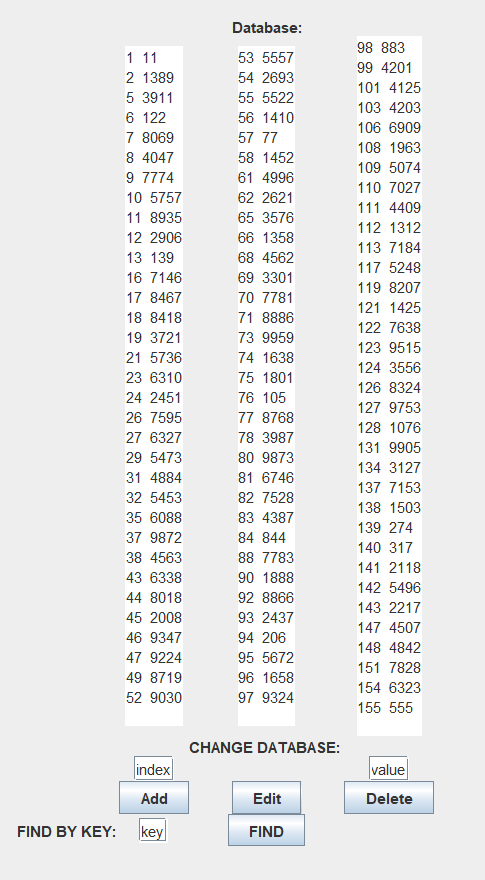
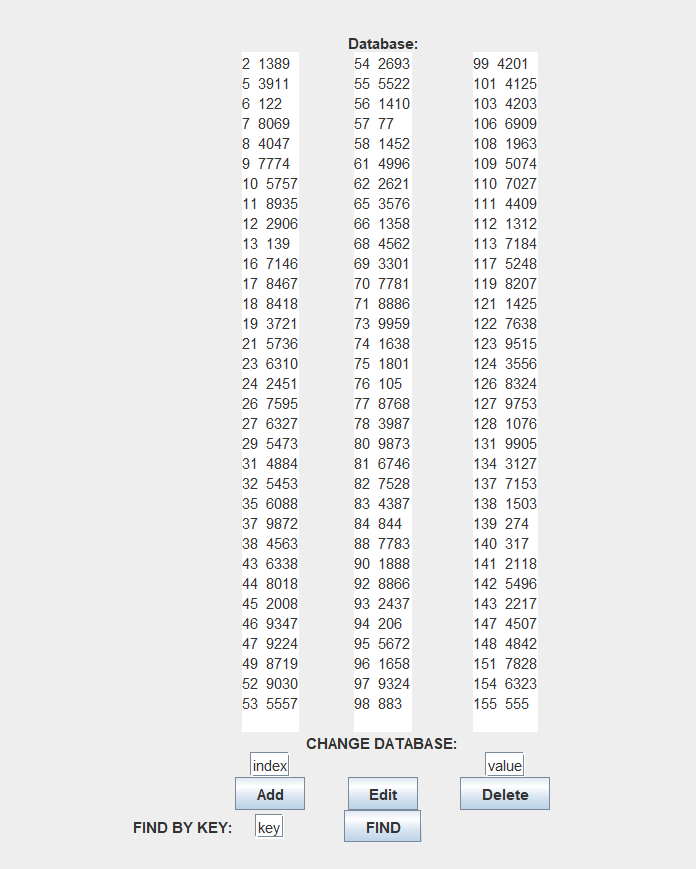


Рисунок 3.1 –Додавання запису (1,11)

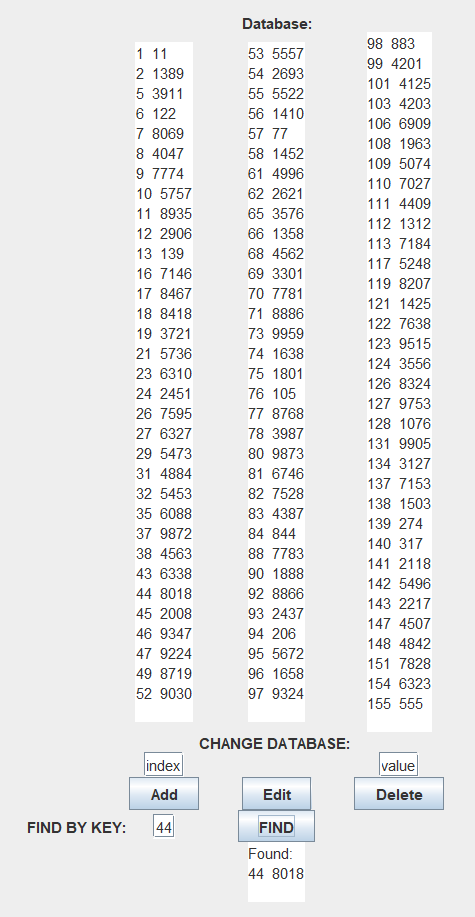


Рисунок 3.2 – Пошук запису

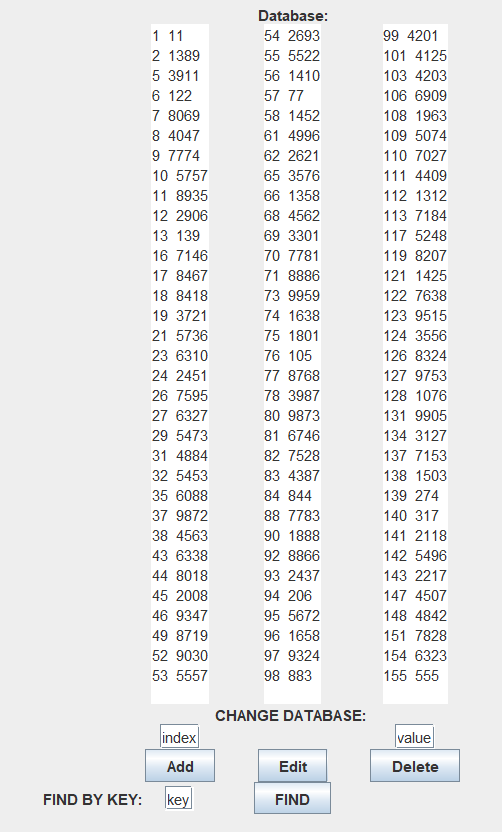
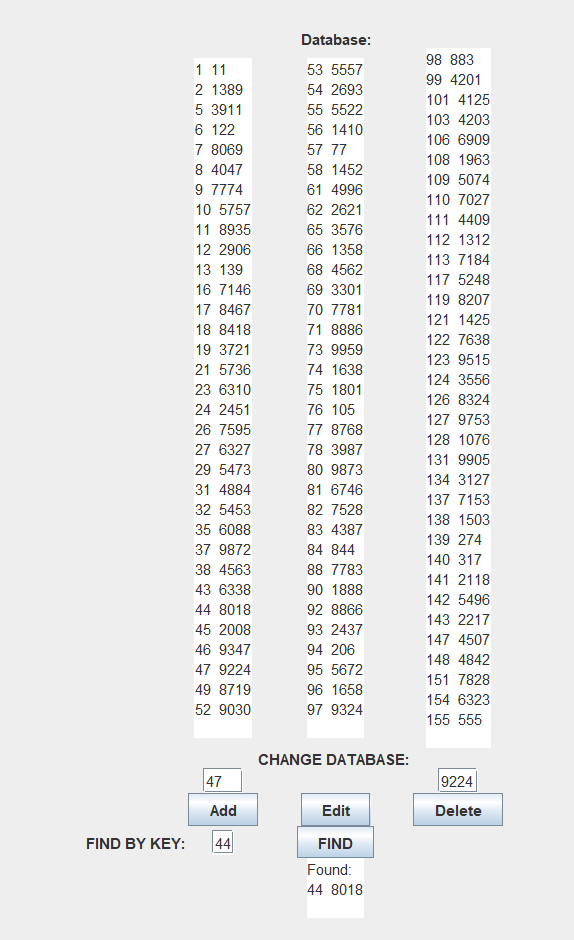


Рисунок 3.3 Видалення запису

## Тестування алгоритму

### Часові характеристики оцінювання

В таблиці 3.1 наведено кількість порівнянь для 11 спроб пошуку запису по ключу.

Таблиця 3.1 – Число порівнянь при спробі пошуку запису по ключу

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

Висновок

В рамках лабораторної роботи було вивчено основні підходи проектування та обробки складних структур даних. Було створено реалізацію простої бази даних з класичними операціями, такі як додавання, видалення та редагування. Графічно реалізація забезпечується javax.swing з простим, але зрозумілим інтерфейсом. Також реалізований пошук у масиві однорідним бінарним пошуком. Та створений файл з не щільним індексом (sparse), який в даній реалізації забезпечує швидке та правильне сортування, що полегшує роботу з бд. Все це записується у файлі з об’єктами класу, створеного для цієї лабораторної роботи.

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.