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 студент: ІП-13 Паламарчук Олександр Олександрович

Київ 2022

ЗМІСТ

ВАРІАНТ 21

МЕТА

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.

ВИКОНАННЯ

- Псевдокод алгоритмів

– Метод Пошуку

Алгоритм search(key)

N <- довжина впорядковаї множини

ЯКЩО length == 0, ТО повернути індикатор невдачі

k <- [log2(N)]

i <- pow(2, k) -1

IndexBlock <- зчитуємо індексний блок з індексом i

itr <- indexBlock.calculateInticator(key)

ЯКЩО itr == 0,

ТО повернути IndexBlock

ЯКЩО itr < 0,

ТО повернути homogeneousBinarySearch(key, i - pow(2, k) / 2, pow(2, k) / 2)

ЯКЩО N > pow(2, k),

ТО L <- log2(N - pow(2, k) + 1) повернути homogeneousBinarySearch(key, N - pow(2, L), pow(2, L))

Повернути індикатор невдачі

Кінець алгоритма

Алгоритм homogeneousBinarySearch(key, i, step)

```
        Повторювати
        IndexBlock <- зчитати індексний блок з індексом
        itr <- indexBlock.calculateInticator(key)
        ЯКЩО itr == 0,
                ТО повернути IndexBlock
                step /= 2
                i = itr < 0 ? -step : step
        Поки step > 0
        Повернути індикатор невдачі
Кінець алгоритма

Алгоритм calculateInticator(key)
        records <- записи в індексному блоці
        first  <- перший запис в records
        ЯКЩО records.size == 1,
                ТО ЯКЩО first.pk == key,
                        ТО повернути 0
                ЯКЩО first.pk > key,
                        ТО повернути -1
                повернути 1
        last <- останній запис в records
```

ЯКЩО id >= first.pk && id <= last.pk,

ТО повернути 0

ЯКЩО id < first.pk,

ТО повернути -1

повернути 1

Кінець алгоритма

– Метод додавання

Алгоритм insert(record)

ЯКЩО globalArea.isEmpty(),

ТО додати перший запис в глобальну область і індексну область

ІНАКШЕ

Записати record в кінець глобальної області

NumberOfIndexBlocks <- кількість блоків в індексній області

IndexBlock <- блок, в який потрібно здійснити запис

ЯКЩО record має ключ, що вже присутній в індексній області,

ТО повернути індикатор невдачі

IsOvercrowded <- додати record в IndexBlock

ЯКЩО isOvercrowded,

ТО реконструювати індексний простір

ІНАКШЕ записати IndexBlock на його місце

Повернути 1

Кінець алгоритма

- Метод редагування

Алгоритм update(updatedRecord)

IndexBlock <- знайти блок в якому знаходиться запис з ключем updatedRecord.id

ЯКЩО блок знайдено,

ТО знайти запис в блоці

ЯКЩО запис знайдено,

ТО редагувати дані

І записати оновлений запис в блок

Записати блок

Кінець алгоритму

- Метод видалення

Алгоритм delete(key)

IndexBlock <- знайти блок в якому знаходиться запис з ключем key

ЯКЩО блок знайдений,

ТО видалити в ньому запис з ключем key

ЯКЩО запис був видалений

ТО ЯКЩО IndexBlock.isEmpty()

ТО здвинути усі блоки, що стоять після нашого на одну позицію вверх

Повернути 1

ІНАКШЕ записати оновлений блок

Повернути 1

Повернути 0

Кінець алгоритму

- Часова складність пошуку

Часова складність для даної задачі визначається у кількость звертань до диксу. Таким чином використовуючи метод пошуку Шарра можемо дійти висновку, що для пошуку запису нам потрібно log2(N) + 1 звернень, де N - це кількість індексних блоків, а + 1 це звертання до глобального простору з даними.

- Програмна реалізація

```
package ua.algorithms.accessor;

import java.io.FileNotFoundException;
```

```java
import java.io.IOException;
import java.io.RandomAccessFile;

public abstract class FileAccessor {
    protected final String fileName;
    protected final RandomAccessFile access;

    public FileAccessor(RandomAccessFile access, String fileName)
{
        this.access = access;
        this.fileName = fileName;
    }

    public static FileAccessor of(String fileName, String
accessor) {
        RandomAccessFile raf;
        try {
            raf = new RandomAccessFile(fileName, "rw");
        } catch (FileNotFoundException e) {
            throw new RuntimeException("File [%s] does not
exist".formatted(fileName), e);
        }

        if (accessor.equals("GLOBAL"))
            return new GlobalFileAccessor(raf, fileName);
        else if (accessor.equals("INDEX"))
            return new IndexFileAccessor(raf, fileName);
        else
            throw new IllegalArgumentException("Bad file
accessor");
    }

    protected byte[] read(int length) {
```

```
            byte[] bytes = new byte[length];
            try {
                access.read(bytes);
            } catch (IOException e) {
                throw new RuntimeException("Failed while reading from
file [%s]".formatted(fileName), e);
            }
            return bytes;
        }

        protected void write(byte[] bytes) {
            try {
                access.write(bytes, 0, bytes.length);
            } catch (IOException e) {
                throw new RuntimeException("Failed while writing to
file [%s]".formatted(fileName), e);
            }
        }

        protected void movePtr(long offset) {
            try {
                access.seek(offset);
            } catch (IOException e) {
                throw new RuntimeException("Failed while moving
pointer in file [%s]".formatted(fileName), e);
            }
        }

        public long getSizeOfFile() {
            try {
                return access.length();
            } catch (IOException e) {
                throw new RuntimeException("Can not get length of
```

```
file %s".formatted(fileName), e);
        }
    }

    public void clearFile() {
        try {
            access.setLength(0);
        } catch (IOException e) {
            throw new RuntimeException("Can not set length of
file [%s]".formatted(fileName), e);
        }
    }

    public boolean isEmpty() {
        return getSizeOfFile() == 0;
    }
}
package ua.algorithms.accessor;

import ua.algorithms.serializer.DatumRecordSerializer;
import ua.algorithms.structure.DatumRecord;

import java.io.RandomAccessFile;

public class GlobalFileAccessor extends FileAccessor {
    public GlobalFileAccessor(RandomAccessFile access, String
fileName) {
        super(access, fileName);
    }

    public long write(DatumRecord datumRecord) {
        long offset = getSizeOfFile();
        movePtr(offset);
```

```
        write(DatumRecordSerializer.serialize(datumRecord));
        return offset;
    }

    public void update(DatumRecord datumRecord, long offset) {
        movePtr(offset);
        write(DatumRecordSerializer.serialize(datumRecord));
    }
    public DatumRecord read(long offset) {
        movePtr(offset);
        return
DatumRecordSerializer.deserialize(read(DatumRecord.BYTES));
    }

    public boolean isEmpty() {
        return getSizeOfFile() == 0;
    }
}
package ua.algorithms.accessor;

import ua.algorithms.serializer.IndexBlockSerializer;
import ua.algorithms.structure.IndexBlock;

import java.io.IOException;
import java.io.RandomAccessFile;

import static ua.algorithms.structure.IndexBlock.BYTES;

public class IndexFileAccessor extends FileAccessor {
    public IndexFileAccessor(RandomAccessFile raf, String
fileName) {
        super(raf, fileName);
    }
```

```
    public void write(IndexBlock indexBlock, int blockNumber) {
        movePtr((long) blockNumber * BYTES);
        write(IndexBlockSerializer.serialize(indexBlock));
    }

    public IndexBlock readBlock(int blockNumber) {
        movePtr((long) blockNumber * BYTES);
        return IndexBlockSerializer.deserialize(read(BYTES));
    }

    public int countNumberOfBlocks() {
        return (int) getSizeOfFile() / BYTES;
    }

    public void setLength(long bytes) {
        try {
            super.access.setLength(bytes);
        } catch (IOException e) {
            throw new RuntimeException("Cannot set length
[%s]".formatted(super.fileName));
        }
    }
}
package ua.algorithms.exception;

public class RecordAlreadyExistsException extends Exception {

    public RecordAlreadyExistsException(String message) {
        super(message);
    }
}
package ua.algorithms.mvc.controller;
```

```
import ua.algorithms.exception.RecordAlreadyExistsException;
import ua.algorithms.mvc.Controller;
import ua.algorithms.mvc.Model;
import ua.algorithms.structure.DatumRecord;

import java.util.Objects;
import java.util.Optional;

public class SimpleController implements Controller {

    private final Model model;

    public SimpleController(Model model) {
        this.model = model;
    }

    private static final String INVALID_INPUT = "Invalid input";
    private static final String VIOLATION_OF_LENGTH_CONSTRAINT =
"Violation of constraint. Value [%s] length must be <= [%d] but
actually [%d]";
    private static final String VIOLATION_OF_EMPTY_VALUE_CONSTANT
= "Violation of constraint. Value [%s] length must be present";
    private static final String RECORDS_AFFECTED = "%d records
was affected";
    private static final String FIRST_NAME_FIELD = "first name";
    private static final String LAST_NAME_FIELD = "last name";
    private static final String EMAIL = "email";

    @Override
    public String select(String pk) {
        long id;
        try {
```

```
            id = Long.parseLong(pk);
        } catch (NumberFormatException e) {
            return INVALID_INPUT;
        }

        Optional<DatumRecord> datumRecordOptional =
model.findById(id);

        if (datumRecordOptional.isPresent())
            return datumRecordOptional.get().toString();

        return "Record with pk [%d] does not
exist".formatted(id);
    }

    @Override
    public String insert(String pk, String firstName, String
lastName, String email) {
        long id;
        try {
            id = Long.parseLong(pk);
        } catch (NumberFormatException e) {
            return INVALID_INPUT;
        }

        if (firstName.length() == 0)
            return
VIOLATION_OF_EMPTY_VALUE_CONSTANT.formatted(FIRST_NAME_FIELD);

        if (lastName.length() == 0)
            return
VIOLATION_OF_EMPTY_VALUE_CONSTANT.formatted(LAST_NAME_FIELD);
```

```
        if (email.length() == 0)
            return
VIOLATION_OF_EMPTY_VALUE_CONSTANT.formatted(EMAIL);

        String check = checkLengthConstraint(firstName, lastName,
email);

        if (Objects.nonNull(check))
            return check;

        int insert;
        try {
            insert = model.insert(new DatumRecord(id, firstName,
lastName, email));
        } catch (RecordAlreadyExistsException e) {
            return e.getMessage();
        }

        return RECORDS_AFFECTED.formatted(insert);
    }

    @Override
    public String update(String pk, String firstName, String
lastName, String email) {
        long id;
        try {
            id = Long.parseLong(pk);
        } catch (NumberFormatException e) {
            return INVALID_INPUT;
        }

        String check = checkLengthConstraint(firstName, lastName,
email);
```

```
        if (Objects.nonNull(check))
            return check;

        int update = model.update(new DatumRecord(id, firstName,
lastName, email));

        return RECORDS_AFFECTED.formatted(update);
    }

    @Override
    public String delete(String pk) {
        long id;
        try {
            id = Long.parseLong(pk);
        } catch (NumberFormatException e) {
            return INVALID_INPUT;
        }

        int update = model.delete(id);

        return RECORDS_AFFECTED.formatted(update);
    }

    private String checkLengthConstraint(String firstName, String
lastName, String email) {
        if (firstName.length() > DatumRecord.FIRST_NAME_LENGTH)
            return VIOLATION_OF_LENGTH_CONSTRAINT.formatted(
                    FIRST_NAME_FIELD,
DatumRecord.FIRST_NAME_LENGTH, firstName.length());

        if (lastName.length() > DatumRecord.LAST_NAME_LENGTH)
            return VIOLATION_OF_LENGTH_CONSTRAINT.formatted(
```

```
                    LAST_NAME_FIELD,
DatumRecord.LAST_NAME_LENGTH, lastName.length());

        if (email.length() > DatumRecord.EMAIL_LENGTH)
            return
VIOLATION_OF_LENGTH_CONSTRAINT.formatted(EMAIL,
                    DatumRecord.EMAIL_LENGTH, email.length());

        return null;
    }

}
package ua.algorithms.mvc.model;

import ua.algorithms.accessor.GlobalFileAccessor;
import ua.algorithms.accessor.IndexFileAccessor;
import ua.algorithms.exception.RecordAlreadyExistsException;
import ua.algorithms.mvc.Model;
import ua.algorithms.structure.DatumRecord;
import ua.algorithms.structure.IndexBlock;
import ua.algorithms.structure.IndexRecord;

import java.math.RoundingMode;
import java.util.List;
import java.util.Optional;
import java.util.TreeMap;

import static com.google.common.math.LongMath.log2;
import static java.lang.Math.pow;

public class SimpleRepository implements Model {
    private final IndexFileAccessor indexArea;
    private final GlobalFileAccessor globalArea;
```

```
    public SimpleRepository(IndexFileAccessor indexArea,
GlobalFileAccessor globalArea) {
        this.indexArea = indexArea;
        this.globalArea = globalArea;
    }

    public Optional<DatumRecord> findById(long id) {
        Optional<IndexBlock> indexBlockOptional =
searchIndexBlock(id);

        if (indexBlockOptional.isPresent()) {
            IndexBlock indexBlock = indexBlockOptional.get();
            Optional<IndexRecord> indexRecordOptional =
indexBlock.findById(id);

            if (indexRecordOptional.isPresent())
                return
Optional.of(globalArea.read(indexRecordOptional.get().ptr()));
        }

        return Optional.empty();
    }

    public int insert(DatumRecord datumRecord) throws
RecordAlreadyExistsException {
        if (globalArea.isEmpty()) {
            addFirst(datumRecord);
        } else {
            IndexRecord indexRecord =
writeNewDatumRecord(datumRecord);
            int numberOfIndexBlocks =
indexArea.countNumberOfBlocks();
```

```
            int blockIdx = 0;
            IndexBlock indexBlock = null;
            for (; blockIdx < numberOfIndexBlocks; blockIdx++) {
                IndexBlock temp = indexArea.readBlock(blockIdx);
                List<IndexRecord> records = temp.getRecords();
                if (datumRecord.getId() <
records.get(records.size() - 1).pk() || blockIdx ==
numberOfIndexBlocks - 1) {
                    indexBlock = temp;
                    break;
                }
            }

            assert indexBlock != null;
            if (indexBlock.findById((int)
datumRecord.getId()).isPresent())
                throw new RecordAlreadyExistsException("Record
with id [%d] already exists".formatted(datumRecord.getId()));

            boolean isOvercrowded =
indexBlock.addRecord(indexRecord);

            if (isOvercrowded)
                reconstructIndexArea(indexBlock,
numberOfIndexBlocks);
            else
                indexArea.write(indexBlock,
indexBlock.getIndex());

        }

        return 1;
```

```
    }

    public int update(DatumRecord updatedDatumRecord) {
        long id = updatedDatumRecord.getId();
        Optional<IndexBlock> optionalIndexBlock =
searchIndexBlock(id);

        if (optionalIndexBlock.isPresent()) {
            IndexBlock indexBlock = optionalIndexBlock.get();
            Optional<IndexRecord> optionalIndexRecord =
indexBlock.findById(id);
            if (optionalIndexRecord.isPresent()) {
                IndexRecord indexRecord =
optionalIndexRecord.get();
                DatumRecord datumRecord =
globalArea.read(indexRecord.ptr());

                if (!updatedDatumRecord.getFirstName().isBlank())

datumRecord.setFirstName(updatedDatumRecord.getFirstName());

                if (!updatedDatumRecord.getLastName().isBlank())

datumRecord.setLastName(updatedDatumRecord.getLastName());

                if (!updatedDatumRecord.getEmail().isBlank())

datumRecord.setEmail(updatedDatumRecord.getEmail());

                globalArea.update(datumRecord,
indexRecord.ptr());
                return 1;
            }
```

```
        }

        return 0;
    }

    public int delete(long id) {
        Optional<IndexBlock> optionalIndexBlock =
searchIndexBlock(id);

        if (optionalIndexBlock.isPresent()) {
            IndexBlock curr = optionalIndexBlock.get();
            int number = curr.delete(id);

            if (curr.isEmpty()) {
                int numberOfBlocks =
indexArea.countNumberOfBlocks();

                IndexBlock next;
                while (true) {
                    if (curr.getIndex() == numberOfBlocks - 1) {

indexArea.setLength(indexArea.getSizeOfFile() -
IndexBlock.BYTES);
                        return number;
                    }

                    next = indexArea.readBlock(curr.getIndex() +
1);
                    next.setIndex(curr.getIndex());
                    indexArea.write(next, curr.getIndex());
                    curr = next;
                }
            } else {
```

```
                if (number > 0) {
                    indexArea.write(curr, curr.getIndex());
                    return number;
                }
            }
        }

        return 0;
    }

    private void reconstructIndexArea(IndexBlock curr, int
numberOfBlocks) {
        IndexBlock temp = curr.separate();

        if (curr.getIndex() == numberOfBlocks - 1) {
            indexArea.write(curr, curr.getIndex());
            indexArea.write(temp, temp.getIndex());
            return;
        }

        IndexBlock next = indexArea.readBlock(curr.getIndex() +
1);
        while (true) {
            if (next.getIndex() == numberOfBlocks - 1) {
                temp.setIndex(next.getIndex());
                indexArea.write(temp, next.getIndex());
                next.setIndex(next.getIndex() + 1);
                indexArea.write(next, next.getIndex());
                return;
            }

            temp.setIndex(next.getIndex());
            indexArea.write(temp, next.getIndex());
```

```
            temp = next;
            next = indexArea.readBlock(temp.getIndex() + 1);
        }
    }

    private void addFirst(DatumRecord datumRecord) {
        IndexRecord indexRecord =
writeNewDatumRecord(datumRecord);
        IndexBlock indexBlock = new IndexBlock(0, 0, new
TreeMap<>());
        indexBlock.addRecord(indexRecord);
        indexArea.write(indexBlock, indexBlock.getIndex());
    }

    private IndexRecord writeNewDatumRecord(DatumRecord
datumRecord) {
        long ptr = globalArea.write(datumRecord);
        return new IndexRecord(datumRecord.getId(), ptr);
    }

    public Optional<IndexBlock> searchIndexBlock(long key) {
        int length = indexArea.countNumberOfBlocks();

        if (length == 0)
            return Optional.empty();

        int k = log2(length, RoundingMode.DOWN), i = (int) pow(2,
k) - 1;

        IndexBlock indexBlock = indexArea.readBlock(i);
        int itr = indexBlock.calculateIndicator(key, indexBlock);

        if (itr == 0)
```

```
            return Optional.of(indexBlock);

        if (itr < 0)
            return homogeneousBinarySearch(
                    key, i - ((int) pow(2, k) / 2), (int) pow(2,
k) / 2);

        if (length > (int) pow(2, k)) {
            int l = log2(length - (int) pow(2, k) + 1,
RoundingMode.DOWN);
            return homogeneousBinarySearch(key, length - (int)
pow(2, l), (int) pow(2, l));
        }

        return Optional.empty();
    }

    private Optional<IndexBlock> homogeneousBinarySearch(long
key, int i, int step) {
        do {
            IndexBlock indexBlock = indexArea.readBlock(i);
            int itr = indexBlock.calculateIndicator(key,
indexBlock);

            if (itr == 0)
                return Optional.of(indexBlock);

            step /= 2;

            i += itr < 0 ? -step : step;

        } while (step > 0);
```

```
            return Optional.empty();
        }
    }
package ua.algorithms.mvc.view;

public enum BasicAction {

    SELECT("SELECT"),
    INSERT("INSERT"),
    UPDATE("UPDATE"),
    DELETE("DELETE");

    private final String actionName;
    BasicAction(String actionName) {
        this.actionName = actionName;
    }

    public String getActionName() {
        return actionName;
    }
}
package ua.algorithms.mvc.view;

import ua.algorithms.mvc.Controller;

import javax.swing.*;
import java.awt.*;

import static ua.algorithms.mvc.view.BasicAction.*;

public class SimpleGUI extends JFrame {

    private JButton actionButton;
```

```
private JRadioButton selectRadio;
private JRadioButton insertRadio;
private JRadioButton deleteRadio;
private JRadioButton updateRadio;
private JTextPane outputArea;
private JTextField firstNameInputField;
private JTextField lastNameInputField;
private JTextField emailInputField;
private JTextField pkInputField;
private final Controller controller;

public SimpleGUI(Controller controller) {
    super("DBMS");
    this.controller = controller;
}

public void init() {
    setBounds(270, 50, 1000, 300);
    setDefaultCloseOperation(EXIT_ON_CLOSE);

    selectRadio = new JRadioButton(SELECT.getActionName());
    insertRadio = new JRadioButton(INSERT.getActionName());
    deleteRadio = new JRadioButton(DELETE.getActionName());
    updateRadio = new JRadioButton(UPDATE.getActionName());
    selectRadio.setSelected(true);

    ButtonGroup radioGroup = new ButtonGroup();
    radioGroup.add(selectRadio);
    radioGroup.add(insertRadio);
    radioGroup.add(deleteRadio);
    radioGroup.add(updateRadio);

    Panel radioPanel = new Panel();
```

```
radioPanel.add(selectRadio);
radioPanel.add(insertRadio);
radioPanel.add(deleteRadio);
radioPanel.add(updateRadio);
radioPanel.setLayout(new GridLayout(4, 1, 10, 10));

outputArea = new JTextPane();
outputArea.setToolTipText("Output area");


String primaryKeyText = "Primary key";
pkInputField = new JTextField(primaryKeyText);
pkInputField.setToolTipText(primaryKeyText);

String firstNameText = "First name";
firstNameInputField = new JTextField(firstNameText);
firstNameInputField.setToolTipText(firstNameText);

String lastNameText = "Last name";
lastNameInputField = new JTextField(lastNameText);
lastNameInputField.setToolTipText(lastNameText);

String emailText = "Email";
emailInputField = new JTextField(emailText);
emailInputField.setToolTipText(emailText);

actionButton = new JButton(SELECT.getActionName());
actionButton.setToolTipText("Action button");

firstNameInputField.setEditable(false);
lastNameInputField.setEditable(false);
emailInputField.setEditable(false);
```

```
Panel actionPanel = new Panel();
actionPanel.add(pkInputField);
actionPanel.add(firstNameInputField);
actionPanel.add(lastNameInputField);
actionPanel.add(emailInputField);
actionPanel.add(actionButton);
actionPanel.setLayout(new GridLayout(5, 1, 10, 10));

Container contentPane = getContentPane();
contentPane.setLayout(new GridLayout(1, 3, 10, 10));
contentPane.add(radioPanel);
contentPane.add(actionPanel);
contentPane.add(outputArea);

selectRadio.addActionListener(e -> {
    actionButton.setText(SELECT.getActionName());
    firstNameInputField.setEditable(false);
    lastNameInputField.setEditable(false);
    emailInputField.setEditable(false);
});

insertRadio.addActionListener(e -> {
    actionButton.setText(INSERT.getActionName());
    firstNameInputField.setEditable(true);
    lastNameInputField.setEditable(true);
    emailInputField.setEditable(true);
});

updateRadio.addActionListener(e -> {
    actionButton.setText(UPDATE.getActionName());
    firstNameInputField.setEditable(true);
    lastNameInputField.setEditable(true);
    emailInputField.setEditable(true);
```

```
        });

        deleteRadio.addActionListener(e -> {
            actionButton.setText(DELETE.getActionName());
            firstNameInputField.setEditable(false);
            lastNameInputField.setEditable(false);
            emailInputField.setEditable(false);
        });

        actionButton.addActionListener(e -> {
            String pk = pkInputField.getText();
            if (selectRadio.isSelected()) {
                String message = controller.select(pk);
                outputArea.setText(message);
            } else if (insertRadio.isSelected()) {
                String firstName = firstNameInputField.getText();
                String lastName = lastNameInputField.getText();
                String email = emailInputField.getText();
                String message = controller.insert(pk, firstName,
lastName, email);
                outputArea.setText(message);
            } else if (updateRadio.isSelected()) {
                String firstName = firstNameInputField.getText();
                String lastName = lastNameInputField.getText();
                String email = emailInputField.getText();
                String message = controller.update(pk, firstName,
lastName, email);
                outputArea.setText(message);
            } else {
                String message = controller.delete(pk);
                outputArea.setText(message);
            }
        });
```

```
            setVisible(true);
        }
    }
package ua.algorithms.mvc;

public interface Controller {

    String select(String pk);
    String insert(String pk, String firstName, String lastName,
String email);
    String update(String pk, String firstName, String lastName,
String email);
    String delete(String pk);

}
package ua.algorithms.mvc;

import ua.algorithms.exception.RecordAlreadyExistsException;
import ua.algorithms.structure.DatumRecord;

import java.util.Optional;

public interface Model {

    Optional<DatumRecord> findById(long id);
    int insert(DatumRecord newDatumRecord) throws
RecordAlreadyExistsException;
    int update(DatumRecord updatedDatumRecord);
    int delete(long id);

}
package ua.algorithms.serializer;
```

```
import ua.algorithms.structure.DatumRecord;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;

public class DatumRecordSerializer {
    public static byte[] serialize(DatumRecord datumRecord) {
        return ByteBuffer.allocate(DatumRecord.BYTES)
                .putLong(DatumRecord.ID_OFFSET,
datumRecord.getId())
                .put(DatumRecord.FIRST_NAME_OFFSET,
datumRecord.getFirstName().getBytes())
                .put(DatumRecord.LAST_NAME_OFFSET,
datumRecord.getLastName().getBytes())
                .put(DatumRecord.EMAIL_OFFSET,
datumRecord.getEmail().getBytes())
                .array();
    }

    public static DatumRecord deserialize(byte[] bytes) {
        ByteBuffer wrap = ByteBuffer.wrap(bytes);
        long id = wrap.getLong(DatumRecord.ID_OFFSET);

        byte[] firstNameBytes = new
byte[DatumRecord.FIRST_NAME_BYTES];
        wrap.get(DatumRecord.FIRST_NAME_OFFSET, firstNameBytes);
        String firstName = new String(firstNameBytes,
StandardCharsets.UTF_8);

        byte[] lastNameBytes = new
byte[DatumRecord.LAST_NAME_BYTES];
        wrap.get(DatumRecord.LAST_NAME_OFFSET, lastNameBytes);
```

```
        String lastName = new String(lastNameBytes,
StandardCharsets.UTF_8);

        byte[] emailBytes = new byte[DatumRecord.EMAIL_BYTES];
        wrap.get(DatumRecord.EMAIL_OFFSET, emailBytes);
        String email = new String(emailBytes,
StandardCharsets.UTF_8);

        return new DatumRecord(id, firstName.trim(),
lastName.trim(), email.trim());
    }
}
package ua.algorithms.serializer;

import ua.algorithms.structure.IndexBlock;
import ua.algorithms.structure.IndexRecord;

import java.nio.ByteBuffer;
import java.util.TreeMap;

public class IndexBlockSerializer {
    public static byte[] serialize(IndexBlock indexBlock) {
        ByteBuffer buffer = ByteBuffer.allocate(IndexBlock.BYTES)
                .putInt(indexBlock.getIndex())
                .putInt(indexBlock.getSize());
        for (IndexRecord record : indexBlock.getRecords())
            buffer.put(IndexRecordSerializer.serialize(record));
        return buffer.array();
    }

    public static IndexBlock deserialize(byte[] bytes) {
        ByteBuffer wrap = ByteBuffer.wrap(bytes);
        int index = wrap.getInt(IndexBlock.INDEX_OFFSET);
```

```
        int size = wrap.getInt(IndexBlock.SIZE_OFFSET);
        TreeMap<Long, IndexRecord> records = new TreeMap<>();
        for (int i = 0; i < size; i++) {
            byte[] indexRecordBytes = new
byte[IndexRecord.BYTES];
            wrap.get(IndexBlock.RECORDS_OFFSET + i *
IndexRecord.BYTES, indexRecordBytes);
            IndexRecord indexRecord =
IndexRecordSerializer.deserialize(indexRecordBytes);
            records.put(indexRecord.pk(), indexRecord);
        }
        return new IndexBlock(size, index, records);
    }
}
```

```
package ua.algorithms.serializer;

import ua.algorithms.structure.IndexRecord;

import java.nio.ByteBuffer;

public class IndexRecordSerializer {
    public static byte[] serialize(IndexRecord record) {
        return ByteBuffer
                .allocate(IndexRecord.BYTES)
                .putLong(record.pk())
                .putLong(record.ptr())
                .array();
    }

    public static IndexRecord deserialize(byte[] bytes) {
        ByteBuffer wrap = ByteBuffer.wrap(bytes);
        long pk = wrap.getLong(IndexRecord.PRIMARY_KEY_OFFSET);
        long ptr = wrap.getLong(IndexRecord.POINTER_OFFSET);
```

```
            return new IndexRecord(pk, ptr);
        }
    }
    package ua.algorithms.structure;

    import lombok.AllArgsConstructor;
    import lombok.Getter;
    import lombok.NoArgsConstructor;

    @Getter
    @AllArgsConstructor
    @NoArgsConstructor
    public class DatumRecord implements Comparable<DatumRecord> {

        private long id;
        private String firstName;
        private String lastName;
        private String email;
        public static final int ID_OFFSET = 0;
        public static final int ID_BYTES = Long.BYTES;
        public static final int FIRST_NAME_OFFSET = ID_OFFSET +
    ID_BYTES;
        public static final int FIRST_NAME_LENGTH = 60;
        public static final int FIRST_NAME_BYTES = FIRST_NAME_LENGTH
    * 2;

        public static final int LAST_NAME_OFFSET = FIRST_NAME_OFFSET
    + FIRST_NAME_BYTES;
        public static final int LAST_NAME_LENGTH = 60;
        public static final int LAST_NAME_BYTES = LAST_NAME_LENGTH *
    2;

        public static final int EMAIL_OFFSET = LAST_NAME_OFFSET +
```

```
LAST_NAME_BYTES;
    public static final int EMAIL_LENGTH = 60;
    public static final int EMAIL_BYTES = EMAIL_LENGTH * 2;
    public static final int BYTES = ID_BYTES + FIRST_NAME_BYTES +
LAST_NAME_BYTES + EMAIL_BYTES;

    public DatumRecord(long id) {
        this.id = id;
    }

    public void setFirstName(String firstName) {
        if (firstName.length() > FIRST_NAME_LENGTH)
            throw new IllegalArgumentException();

        this.firstName = firstName;
    }

    public void setLastName(String lastName) {
        if (lastName.length() > LAST_NAME_LENGTH)
            throw new IllegalArgumentException();

        this.lastName = lastName;
    }

    public void setEmail(String email) {
        if (email.length() > EMAIL_LENGTH)
            throw new IllegalArgumentException();

        this.email = email;
    }

    @Override
    public String toString() {
```

```
        return """
                {
                    id: %d,
                    firstName: %s,
                    lastName: %s,
                    email: %s
                }
                """.formatted(id, firstName, lastName, email);
    }

    @Override
    public int compareTo(DatumRecord o) {
        return Long.compare(this.id, o.id);
    }
}
package ua.algorithms.structure;

import lombok.AllArgsConstructor;
import lombok.Getter;

import java.util.*;

@Getter
@AllArgsConstructor
public class IndexBlock {

    private int size;
    private int index;
    private Map<Long, IndexRecord> records;
    public static final int INDEX_OFFSET = 0;
    private static final int INDEX_BYTES = Integer.BYTES;
    public static final int SIZE_OFFSET = INDEX_OFFSET +
INDEX_BYTES;
```

```
    public static final int SIZE_BYTES = Integer.BYTES;
    public static final int RECORDS_OFFSET = SIZE_OFFSET +
SIZE_BYTES;
    public static final int RECORDS_BYTES = 1024;
    public static final int BYTES = INDEX_BYTES + SIZE_BYTES +
RECORDS_BYTES; // size of block in bytes

    public boolean addRecord(IndexRecord indexRecord) {
        records.put(indexRecord.pk(), indexRecord);
        return ++size > RECORDS_BYTES / IndexRecord.BYTES;
    }

    public Optional<IndexRecord> findById(long id) {
        return Optional.ofNullable(records.get(id));
    }

    public int delete(long id) {
        IndexRecord i = records.remove(id);

        if (Objects.nonNull(i)) {
            size--;
            return 1;
        }

        return 0;
    }

    public int calculateIndicator(long id, IndexBlock block) {
        List<IndexRecord> records = block.getRecords();
        IndexRecord first = records.get(0);
        if (records.size() == 1) {
            return Long.compare(id, first.pk());
        }
```

```
        IndexRecord last = records.get(records.size() - 1);
        if (id >= first.pk() && id <= last.pk())
            return 0;

        if (id < first.pk())
            return -1;

        return 1;
    }

    public IndexBlock separate() {
        TreeMap<Long, IndexRecord> partOfRecords = new
TreeMap<>();
        List<IndexRecord> values = valueOf(records.values());

        int length = values.size() / 2;

        records.clear();

        for (int i = 0; i < length; i++) {
            IndexRecord indexRecord = values.get(i);
            records.put(indexRecord.pk(), indexRecord);
        }

        for (int i = length; i < values.size(); i++) {
            IndexRecord indexRecord = values.get(i);
            partOfRecords.put(indexRecord.pk(), indexRecord);
        }

        size -= partOfRecords.size();

        return new IndexBlock(partOfRecords.size(), index + 1,
```

```
partOfRecords);
    }

    private List<IndexRecord> valueOf(Collection<IndexRecord>
values) {
        return new ArrayList<>(values);
    }

    public boolean isEmpty() {
        return records.isEmpty();
    }

    public void setIndex(int index) {
        this.index = index;
    }

    public List<IndexRecord> getRecords() {
        return valueOf(records.values());
    }

}
package ua.algorithms.structure;

public record IndexRecord(long pk, long ptr) implements
Comparable<IndexRecord> {

    public static final int PRIMARY_KEY_OFFSET = 0;
    public static final int PRIMARY_KEY_BYTES = Long.BYTES;
    public static final int POINTER_OFFSET = PRIMARY_KEY_OFFSET +
PRIMARY_KEY_BYTES;
    public static final int POINTER_BYTES = Long.BYTES;
    public static final int BYTES = PRIMARY_KEY_BYTES +
POINTER_BYTES;
```

```
    @Override
    public int compareTo(IndexRecord o) {
        return Long.compare(this.pk, o.pk);
    }
}
package ua.algorithms;

import lombok.SneakyThrows;
import ua.algorithms.accessor.FileAccessor;
import ua.algorithms.accessor.GlobalFileAccessor;
import ua.algorithms.accessor.IndexFileAccessor;
import ua.algorithms.mvc.Controller;
import ua.algorithms.mvc.Model;
import ua.algorithms.mvc.controller.SimpleController;
import ua.algorithms.mvc.model.SimpleRepository;
import ua.algorithms.mvc.view.SimpleGUI;


public class Main {
    @SneakyThrows
    public static void main(String[] args) {
        IndexFileAccessor indexFileAccessor =
                (IndexFileAccessor)
FileAccessor.of("src/main/resources/index.bin", "INDEX");
        GlobalFileAccessor globalFileAccessor =
                (GlobalFileAccessor)
FileAccessor.of("src/main/resources/global.bin", "GLOBAL");

        Model model = new SimpleRepository(indexFileAccessor,
globalFileAccessor);
        Controller controller = new SimpleController(model);
        SimpleGUI simpleGUI = new SimpleGUI(controller);
```

```
            simpleGUI.init();
        }
    }
```

- Приклад роботи

На рисунках 3.1, 3.2, 3.3, 3.4 показані приклади роботи програми для додавання, пошуку, зміни і видалення запису.

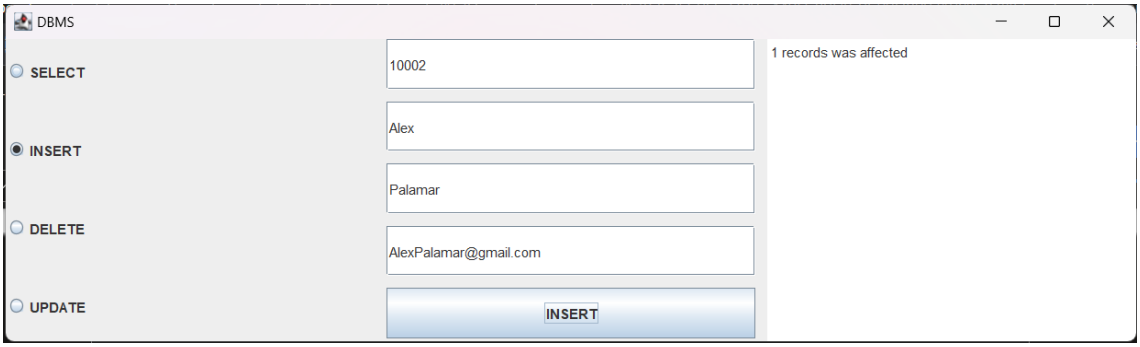

Рисунок 3.1 –Додавання запису

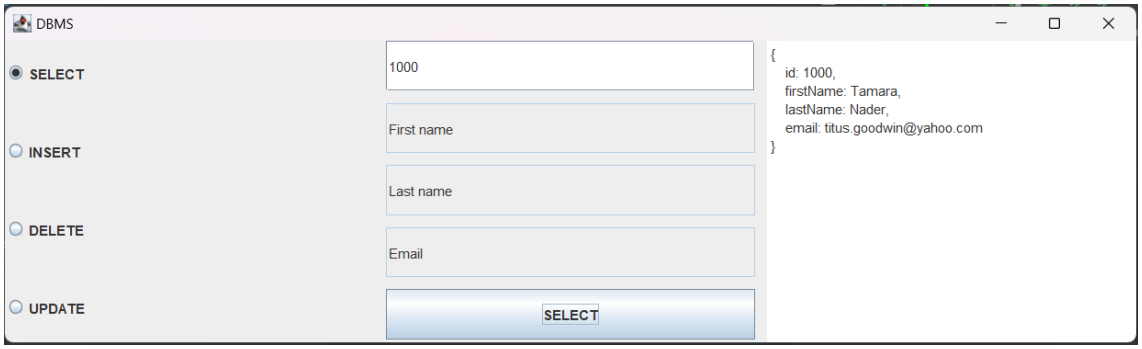

Рисунок 3.2 – Пошук запису

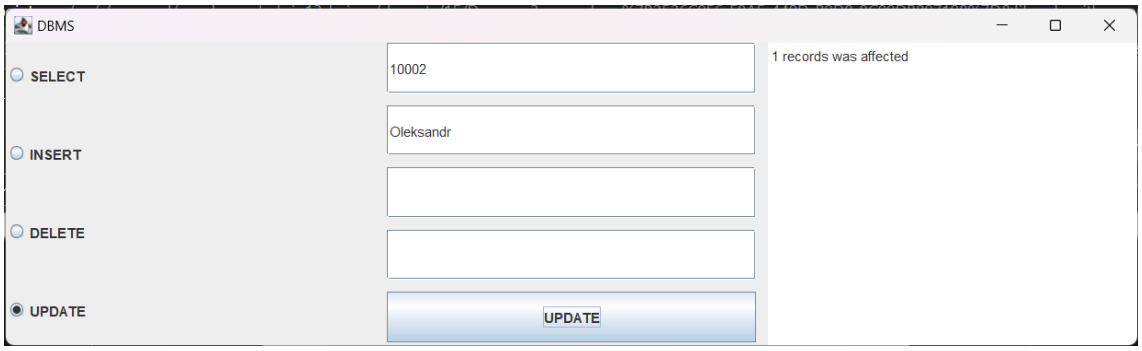

Рисунок 3.3 – Зміна запису

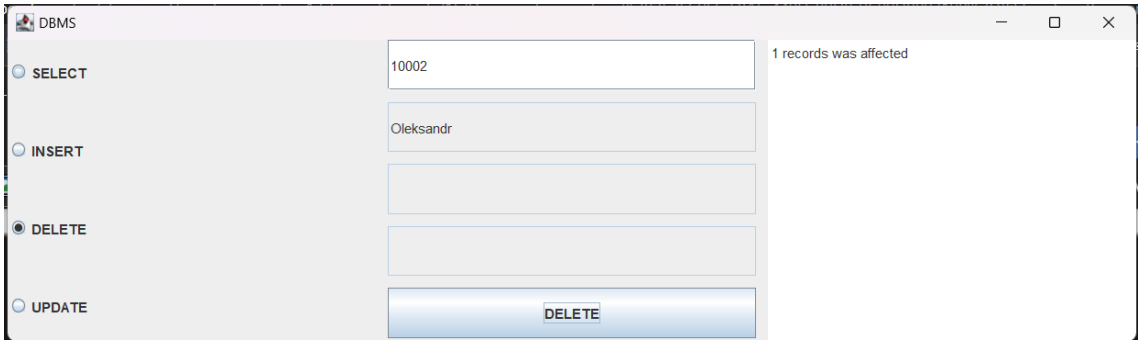

Рисунок 3.4 – Видалення запису

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

Номер спроби пошуку	Число порівнянь
1	8
2	4
3	6

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

ВИСНОВОК

На лабораторній роботі було вивчено основні підходи проектування та обробки складних структур даних, було реалізовано програмне забезпечення, що дає змогу здійснювати пошук, додавання, редагування і видалення даних, в основі якого лежить файл із щільним індексом та з перебудовою індексної області, а також пошук методом Шарра. У ході роботи було досліджено алгоритм пошуку і ми дійшли висновку, що для того, щоб знайти запис нам потрібно виконати не більше як log2(N) + 1 зчитавань з файлу. Середня кількість порівнянь при пошуку запису 6.73.