**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**



**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

# Отчёт

по лабораторной работе № 8

по дисциплине «Технологии и методы программирования»

**Тема: «Практическая оценка сложности алгоритмов»**

Вариант 4

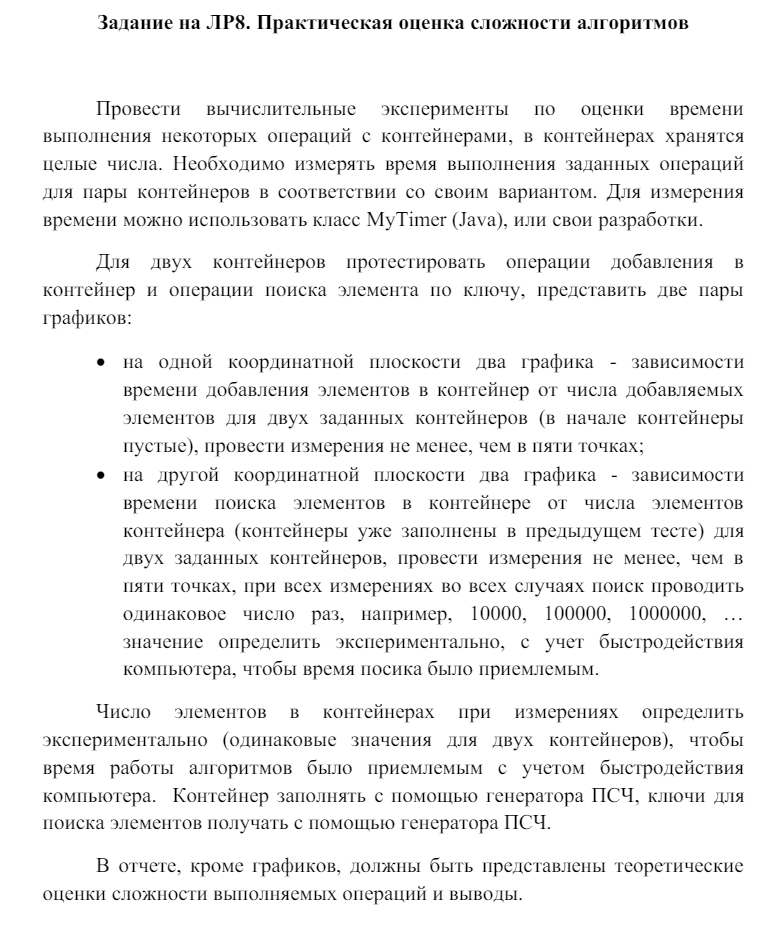
Выполнил:

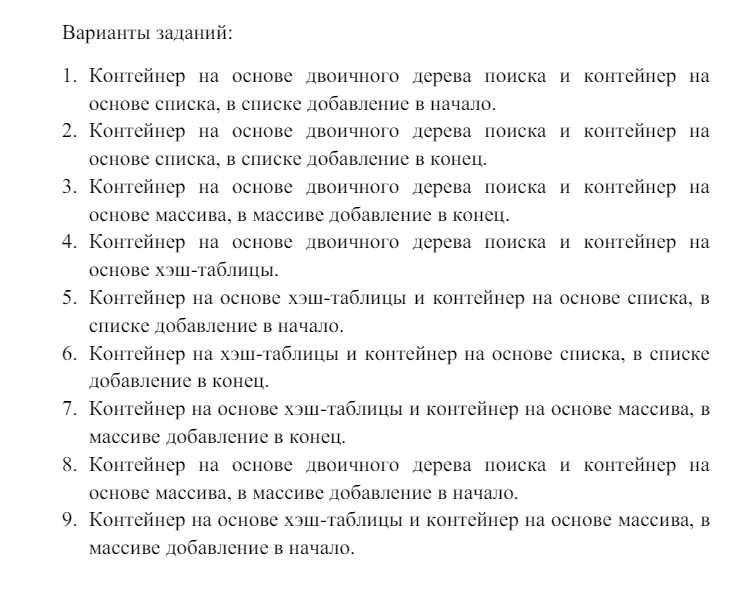
А. В. Куликова,

студент группы ИУ8-21М Проверил: А. Ю. Быков

Москва, 2024

## 1. Постановка задачи





**2. Ход работы**

Листинг 1 – Код программы main.java

import java.awt.BorderLayout;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.HashMap;

import java.util.Map;

import javax.swing.JFrame;

import javax.swing.SwingUtilities;

import org.jfree.chart.ChartFactory;

import org.jfree.chart.ChartPanel;

import org.jfree.chart.JFreeChart;

import org.jfree.chart.plot.PlotOrientation;

import org.jfree.data.xy.XYSeries;

import org.jfree.data.xy.XYSeriesCollection;

// Контейнер на основе двоичного дерева поиска

class BinarySearchTreeContainer {

    private Node root;

    private class Node {

        int key;

        Node left, right;

        public Node(int key) {

            this.key = key;

            left = null;

            right = null;

        }

    }

    public BinarySearchTreeContainer() {

        root = null;

    }

    public void insert(int key) {

        root = insertRec(root, key);

    }

    private Node insertRec(Node root, int key) {

        if (root == null) {

            root = new Node(key);

            return root;

        }

        if (key < root.key) {

            root.left = insertRec(root.left, key);

        } else if (key > root.key) {

            root.right = insertRec(root.right, key);

        }

        return root;

    }

    public boolean search(int key) {

        return searchRec(root, key);

    }

    private boolean searchRec(Node root, int key) {

        if (root == null) {

            return false;

        }

        if (root.key == key) {

            return true;

        }

        if (key < root.key) {

            return searchRec(root.left, key);

        } else {

            return searchRec(root.right, key);

        }

    }

}

// Абстрактный класс Hash с методом getHash

abstract class Hash {

    public abstract String getHash(String data);

}

// Класс для хеширования данных с использованием MD5

class MD5Hash extends Hash {

    @Override

    public String getHash(String data) {

        return digestJavaHexString("MD5", data);

    }

    private String digestJavaHexString(String algorithm, String message) {

        StringBuffer sb = new StringBuffer();

        try {

            MessageDigest md = MessageDigest.getInstance(algorithm);

            md.update(message.getBytes());

            byte[] digest = md.digest();

            for (byte b : digest) {

                sb.append(String.format("%02x", b & 0xff));

            }

        } catch (NoSuchAlgorithmException e) {

            e.printStackTrace();

        }

        return sb.toString();

    }

}

// Класс, использующий шаблонный метод и стратегию для получения хешей

class HashGenerator {

    public String getHash(String data, Hash hash) {

        // System.out.println(hash);

        return hash.getHash(data);

    }

}

// Контейнер на основе хэш-таблицы

public class HashMapContainer {

    private Map<Integer, String> map;

    public HashMapContainer() {

        map = new HashMap<>();

    }

    public void put(int key, String value) {

        // Использование MD5 для хеширования

        HashGenerator generator = new HashGenerator();

        Hash md5Hash = new MD5Hash();

        String md5Result = generator.getHash(value, md5Hash);

        map.put(key, md5Result);

    }

    public String get(int key) {

        return map.get(key);

    }

    public void printAllEntries() {

        // System.out.println("All entries in the HashMap:");

        for (Map.Entry<Integer, String> entry : map.entrySet()) {

            System.out.println(

                    "Ключ: " + entry.getKey() + ", код хэша: " + entry.getValue());

        }

    }

}

class Main {

    public static class MyTimer {

        private long startTime;

        public void start() {

            startTime = System.currentTimeMillis();

        }

        public long stop() {

            long endTime = System.currentTimeMillis();

            return endTime - startTime;

        }

        public MyTimer() {

            startTime = System.currentTimeMillis();

        }

        public void reset() {

            startTime = System.currentTimeMillis();

        }

        public long getTimeElapsed() {

            return System.currentTimeMillis() - startTime;

        }

    }

    private static void testAdditionAndSearch(Object container, String containerType) {

        MyTimer timer = new MyTimer();

        XYSeries addSeries = new XYSeries("Время добавления");

        XYSeries searchSeries = new XYSeries("Время поиска");

        for (int i = 1000; i <= 10000; i += 2000) {

            timer.start();

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

                if (container instanceof BinarySearchTreeContainer) {

                    ((BinarySearchTreeContainer) container).insert(j);

                } else if (container instanceof HashMapContainer) {

                    ((HashMapContainer) container).put(j, "значение" + j);

                }

            }

            long addTime = timer.stop();

            addSeries.add(i, addTime);

            timer.start();

            for (int j = 0; j < 10000; j++) {

                int keyToSearch = j % i;

                if (container instanceof BinarySearchTreeContainer) {

                    ((BinarySearchTreeContainer) container).search(keyToSearch);

                } else if (container instanceof HashMapContainer) {

                    ((HashMapContainer) container).get(keyToSearch);

                }

            }

            long searchTime = timer.stop();

            searchSeries.add(i, searchTime);

        }

        createChart(addSeries, "Время добавления для " + containerType);

        createChart(searchSeries, "Время поиска для " + containerType);

    }

    private static void createChart(XYSeries series, String title) {

        XYSeriesCollection dataset = new XYSeriesCollection(series);

        JFreeChart chart = ChartFactory.createXYLineChart(title, "Количество элементов",

                "Время (мс)", dataset,

                PlotOrientation.VERTICAL, true, true, false);

        SwingUtilities.invokeLater(() -> {

            JFrame frame = new JFrame(title);

            frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

            frame.setLayout(new BorderLayout());

            ChartPanel chartPanel = new ChartPanel(chart);

            frame.add(chartPanel, BorderLayout.CENTER);

            frame.pack();

            frame.setLocationRelativeTo(null);

            frame.setVisible(true);

        });

    }

    public static void main(String[] args) {

        BinarySearchTreeContainer bstContainer = new BinarySearchTreeContainer();

        testAdditionAndSearch(bstContainer, "BinarySearchTreeContainer");

        HashMapContainer hashMapContainer = new HashMapContainer();

        testAdditionAndSearch(hashMapContainer, "HashMapContainer");

    }

}

Результат с тестовым количеством [1000; 10000]:

