Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

КУРСОВОЙ ПРОЕКТ

Дисциплина: Алгоритмы и структуры данных

Тема: Разработка GUI приложения для реализации АВЛ дерева в виде класса на Java с авторегулировкой.

Выполнила

студентка гр. №3530903/80003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фам Т. Т. Бинь

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Глухих М. И.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019 г.

Санкт-Петербург

2019

Оглавление

Техническое задание …………………………………………………………3

Метод решения ……………………………………………………………..4

Листинг ………………………………………………………………………...5

[**AVLNode.java**](https://github.com/PhamBinh98/AVLTree/blob/master/src/AVLNode.java) ………………………………………………………………5

**AvlTree.java** ……………………………………………………………………5

**Board.java** ……………………………………………………………………..18

**MainFraim.java** ……………………………………………………………….23

**RunAVL.java** …………………………………………………………………29

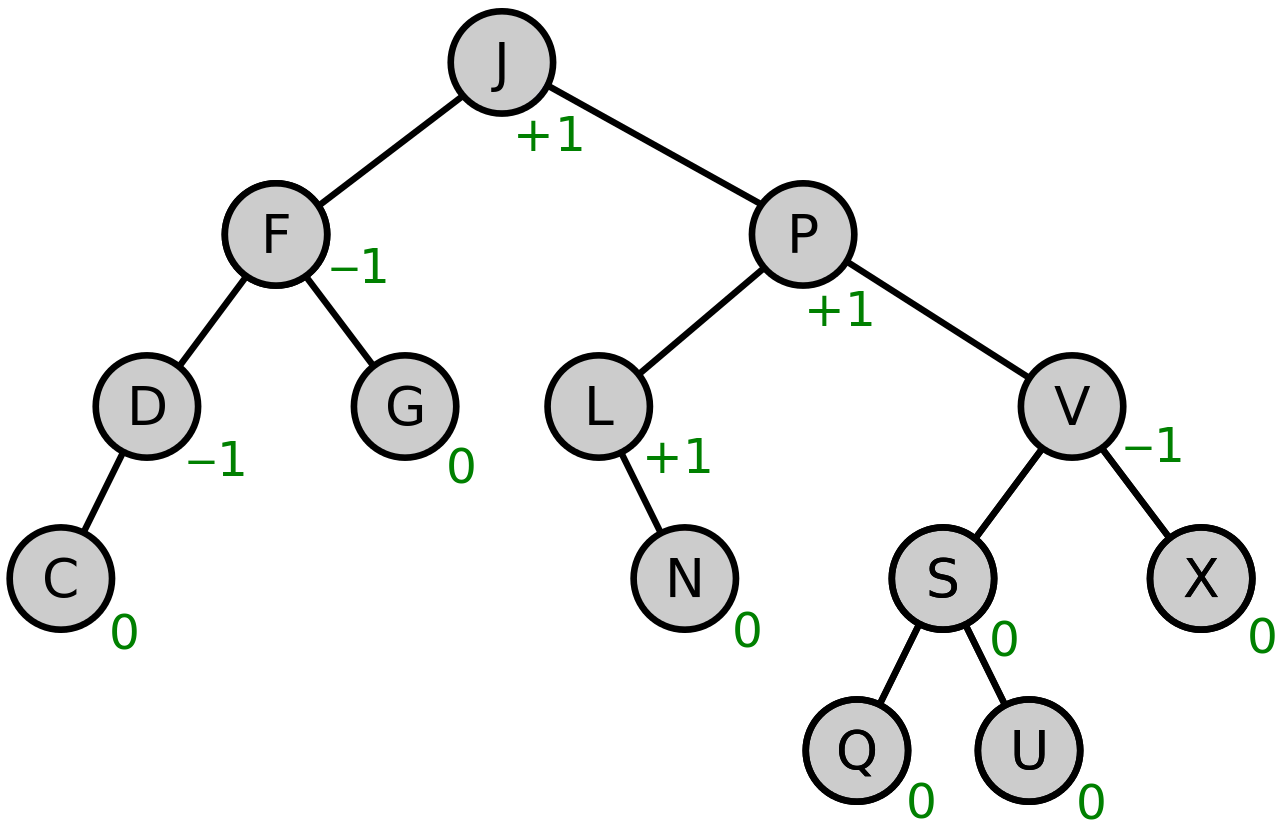
Тестирование ………………………………………………………………....30

Скриншоты работы программы ……………………………………………37

Техническое задание

Реализация в виде класса на Java одного из видов бинарного дерева поиска с авторегулировкой. Класс дерева должен реализовывать соответствующий интерфейс Java в обобщённом виде Set.

АВЛ-дерево - Сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.



GitHub репозиторий проекта: <https://github.com/PhamBinh98/AVLTree>.

Метод решения

В проекте использована концепция MVC (Model-View-Controller) для структурирования кода и отделения логики от визуализации.

**1.Класс “AVLNode”:** Реализация объекта “AVLNode”. В этом классе есть следующие данные:

* + **data :** некоторые данные, привязанные к узлу.
  + **left и right:** ссылки на узлы, являющиеся детьми данного узла — левый и правый сыновья соответственно.
  + **height:** высота узла.

**2. Класс “AvlTree”:** я реализовал этот класс в обобщенном виде (Set). В этом классе есть следующие данные:

- **FindNode ():** Найти node, значение которого (data) равно или близко к данному значению.

- **Contains ():** Есть ли данное значение в дереве?

- **insertNode ():** Добавлять node в дерево и осуществлять баланс.

- **height ():** Выводить высоту данного узла и определить высоты пустого узла.

- **calculateTreeHeight ():** Вычислить высоту данного узла.

- **getBalanceValue ():** На сколько различается высота двух поддеревьев.

- **isRightHeavy () и isLeftHeavy ():** проверка правой (левой) разбалансировки.

- **balanceTree ():** Расстановка балансов.

+) При добавления: тип разбалансировки: right-right; left-left; right-left; left-right.

+) При удалении: левая разбалансировка, правая разбалансировка.

- **rightRotate () и leftRotate ():** Методы поворота вокруг данного узла (лево и право).

- **removeNode ():** удаление узла.

**3.Класс “Board”:** в поле board мы рисуем дерево. В этом классе есть следующие данные:

- **insertNode ():** Добавлять node в дерево.

- **clearBoard ():** Убрать данное дерево.

- **removeNode ():** Убрать node.

- **findNode ():** Найти node в дерево.

Если мы нашли узла, она изменит цвет и показать уведомление: “You found node”

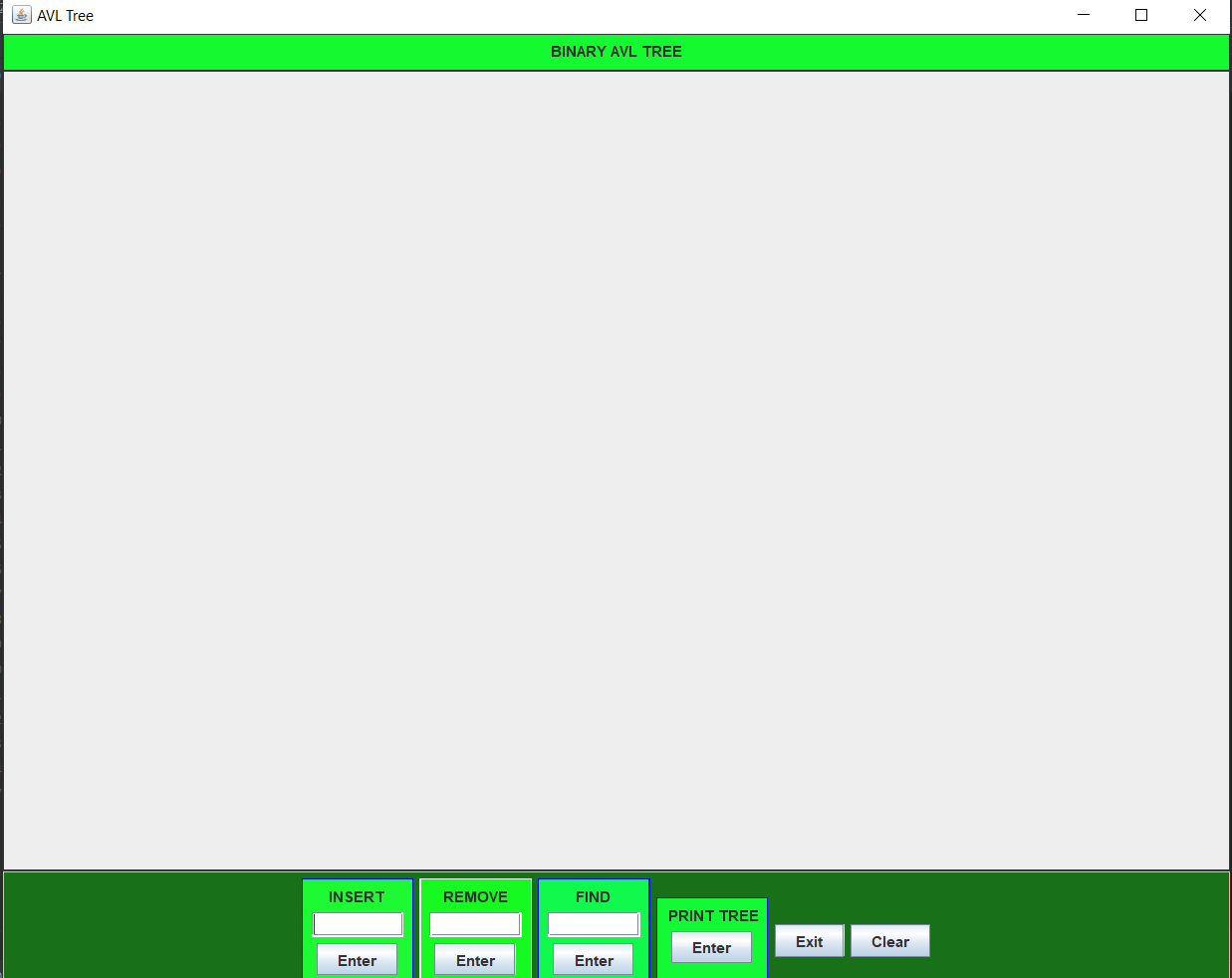
- **patinOver ():** Рисовать все куги и значения данного дерева.

- **patinGeneralPath ():** Рисовать круг и значение данного узла.

**4.Класс “MainFraim”:** Отображение графика дерева. В этом классе есть следующие данные:

- initHeadPanel(), initMainPanel(), initOptionPanel(), initRadioPanel():

методы распределения компонентов.

****

**initHeadPanel()**

**initMainPanel()**

**initRadioPanel()**

**initOptionPanel()**

**5.Класс “RunAVL”:** Проверьте правильность работы дерева.

Листинг

[**AVLNode.java**](https://github.com/PhamBinh98/AVLTree/blob/master/src/AVLNode.java)

public class AVLNode<T extends Comparable<T>> {

T data;

int height = 0;

AVLNode<T> left = null;

AVLNode<T> right = null;

AVLNode(T data) {

this.data = data;

}

@Override

public String toString() {

return "AVLNode(data=" + this.data + ")";

}

}

**AvlTree.java**

import java.security.AlgorithmConstraints;

import java.util.\*;

public class AvlTree <T extends Comparable<T>> implements Set<T> {

private AVLNode<T> root = null;

AVLNode<T> getRoot() {

return root;

}

private int size = 0;

@Override

public boolean contains(Object o) {

@SuppressWarnings("unchecked")

T t = (T) o;

AVLNode<T> closest = find(t);

return closest != null && closest.data.compareTo(t) == 0;

}

AVLNode<T> find(T data) {

if (root == null) return null;

return findNode(root, data);

}

// Найти node, значение которого (data) равно Т или близко к Т

private AVLNode<T> findNode(AVLNode<T> start, T data) {

int comparison = data.compareTo(start.data);

if (comparison == 0) {

return start;

}

else if (comparison < 0) {

if (start.left == null) return start;

return findNode(start.left, data);

}

else {

if (start.right == null) return start;

return findNode(start.right, data);

}

}

@Override

public boolean add(T t) {

if (contains(t)) return false;

root = insertNode(root, t);

size ++;

return true;

}

@Override

public int size() {

return size;

}

@Override

public boolean isEmpty() {

return size == 0;

}

@Override

public boolean remove(Object o) {

@SuppressWarnings("unchecked")

T t = (T) o;

if (!contains(o)) return false;

root = removeNode(root, t);

size--;

return true;

}

// Добавлять node в дерево и осуществовать баланс

private AVLNode<T> insertNode(AVLNode<T> currentNode, T dataToInsert) {

if (currentNode == null) {

return new AVLNode<>(dataToInsert);

}

if (dataToInsert.compareTo(currentNode.data) < 0) {

currentNode.left = insertNode(currentNode.left, dataToInsert);

} else {

currentNode.right = insertNode(currentNode.right,dataToInsert);

}

currentNode = balanceTree(currentNode, dataToInsert); //осуществовать баланс

currentNode.height = calculateTreeHeight(currentNode); // вычислить рост нового узла

return currentNode;

}

private int height(AVLNode<T> currentNode) {

if (currentNode == null) {

return -1; // определение пустого роста

}

return currentNode.height;

}

//вычислить роста currentNode

private int calculateTreeHeight(AVLNode<T> currentNode) {

return Math.max(height(currentNode.left),height(currentNode.right)) + 1;

}

// С помошью этого методы мы получим целое число, из которого мы узнали различие между сынами

private int getBalanceValue(AVLNode<T> currentNode) {

if (currentNode == null) {

return 0;

}

return height(currentNode.left) - height(currentNode.right);

}

// проверка правой разбалансировки

private boolean isRightHeavy(AVLNode<T> currentNode) {

return getBalanceValue(currentNode) < -1;

}

// проверка левой разбалансировки

private boolean isLeftHeavy(AVLNode<T> currentNode) {

return getBalanceValue(currentNode) > 1;

}

// Расстановка балансов при добавления

private AVLNode<T> balanceTree(AVLNode<T> currentNode, T dataToInsert) {

// Тип разбалансировки - это right - right

if (isRightHeavy(currentNode) && dataToInsert.compareTo(currentNode.right.data) > 0) {

return leftRotate(currentNode);

}

// Тип разбалансировки - это left - left

if (isLeftHeavy(currentNode) && dataToInsert.compareTo(currentNode.left.data) < 0) {

return rightRotate(currentNode);

}

// Тип разбалансировки - это right - left

if (isRightHeavy(currentNode) && dataToInsert.compareTo(currentNode.right.data) < 0) {

currentNode.right = rightRotate(currentNode.right);

return leftRotate(currentNode);

}

// Тип разбалансировки - это left - right

if (isLeftHeavy(currentNode) && dataToInsert.compareTo(currentNode.left.data) > 0) {

currentNode.left = leftRotate(currentNode.left);

return rightRotate(currentNode);

}

return currentNode;

}

//Расстановка балансов при удалении

private AVLNode<T> balanceTree(AVLNode<T> current) {

int balance = getBalanceValue(current);

// левая разбалансировка

if (balance > 1) {

if (getBalanceValue(current.left) < 0) {

current.left = leftRotate(current.left);

}

return rightRotate(current);

}

//правая разбалансировка

if (balance < -1) {

if (getBalanceValue(current.right) > 0) {

current.right = rightRotate(current.right);

}

return leftRotate(current);

}

return current;

}

//Метод правого поворота вокруг currentNode

private AVLNode<T> rightRotate(AVLNode<T> currentNode) {

AVLNode<T> newRootNode = currentNode.left;

currentNode.left = newRootNode.right;

newRootNode.right = currentNode;

newRootNode.height = calculateTreeHeight(newRootNode);

currentNode.height = calculateTreeHeight(currentNode);

return newRootNode;

}

// Метод левого поворота вокруг currentNode

private AVLNode<T> leftRotate(AVLNode<T> currentNode) {

AVLNode<T> newRootNode = currentNode.right;

currentNode.right = newRootNode.left;

newRootNode.left = currentNode;

newRootNode.height = calculateTreeHeight(newRootNode);

currentNode.height = calculateTreeHeight(currentNode);

return newRootNode;

}

private AVLNode<T> getMaxNode(AVLNode<T> currentNode) {

while(currentNode.right != null) {

currentNode = currentNode.right;

}

return currentNode;

}

private AVLNode<T> removeNode(AVLNode<T> current, T dataToRemove) {

if (current == null) {

return null;

}

AVLNode<T> leftChild = current.left;

AVLNode<T> rightChild = current.right;

if (dataToRemove.compareTo(current.data) == 0) {

if (leftChild == null && rightChild == null) {

return null;

} else if (leftChild == null) {

return rightChild;

} else if (rightChild == null) {

return leftChild;

} else {

AVLNode<T> successor = getMaxNode(leftChild);

current.data = successor.data;

current.left = removeNode(current.left, successor.data);

}

} else if (dataToRemove.compareTo(current.data) < 0) {

current.left = removeNode(leftChild, dataToRemove);

} else {

current.right = removeNode(rightChild, dataToRemove);

}

current.height = calculateTreeHeight(current);

return balanceTree(current);

}

public class AvlTreeIterator implements Iterator {

private AVLNode<T> current = null;

private int location = 0;

private List<AVLNode<T>> list;

private AvlTreeIterator() {

list = new ArrayList<>();

addToList(root);

}

private void addToList(AVLNode<T> node) {

if (node != null) {

addToList(node.left);

list.add(node);

addToList(node.right);

}

}

private AVLNode<T> findNext() {

return list.get(location++);

}

@Override

public boolean hasNext() {

return location <list.size();

}

@Override

public Object next() {

current = findNext();

if (current == null) throw new NoSuchElementException();

return current.data;

}

@Override

public void remove() {

AvlTree.this.remove(current.data);

list.remove(current);

location--;

}

}

@Override

public Iterator<T> iterator() {

return new AvlTreeIterator();

}

@Override

public Object[] toArray() {

Object[] r = new Object[size()];

Iterator<T> it = this.iterator();

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

if (it.hasNext()) {

r[i] = it.next();

}

}

return r;

}

@SuppressWarnings("uncheked")

@Override

public <T1> T1[] toArray(T1[] a) {

T1[] r = a.length >= size() ? a :

(T1[]) java.lang.reflect.Array

.newInstance(a.getClass().getComponentType(), this.size());

Iterator<T> it = iterator();

for (int i = 0; i < r.length; i++){

if (!it.hasNext()) {

r[i] = null;

return r;

}

r[i] = (T1) it.next();

}

return r;

}

@Override

public boolean containsAll(Collection<?> c) {

for(Object e : c) {

if (!contains(e))

return false;

}

return true;

}

@Override

public boolean addAll(Collection<? extends T> c) {

boolean b = false;

for ( T e : c) {

if (this.add(e)) b = true;

}

return b;

}

@Override

public boolean retainAll(Collection<?> c) {

boolean b = false;

Iterator<T> it = this.iterator();

while (it.hasNext()) {

if (!c.contains(it.next())) {

it.remove();

b = true;

}

}

return b;

}

@Override

public boolean removeAll(Collection<?> c) {

boolean b = false;

for (Object e : c) {

if (this.contains(e)) {

this.remove(e);

b = true;

}

}

return b;

}

@Override

public void clear() {

Iterator<T> iterator = this.iterator();

while (iterator.hasNext()) {

iterator.next();

iterator.remove();

}

}

@Override

public boolean equals(Object obj) {

if (obj == this) return true;

if (obj instanceof Set) {

Set<?> o = (AvlTree<?>) obj;

return o.size() == this.size() && o.containsAll(this) && this.containsAll(o);

}

return false;

}

@Override

public String toString() {

if (isEmpty()) {

return "AVLTree is empty";

} else {

StringBuilder result = new StringBuilder("This is AVLTree. It contains elements:");

for (T t : this){

result.append(" ").append(t);

}

return result.toString();

}

}

}

**Board.java**

import javax.swing.\*;

import java.awt.\*;

import java.awt.geom.Ellipse2D;

import java.awt.geom.GeneralPath;

public class Board extends JPanel {

private AvlTree<Integer> avlTree;

private int fistX;

private int fistY;

private boolean isSearch = false;

private String toSearch = null;

Board(int fromX, int fromY) {

avlTree = new AvlTree<>();

this.fistX = fromX;

this.fistY = fromY;

}

void insertNode(int data) {

avlTree.add(data);

this.repaint();

}

void clearBoard() {

avlTree.clear();

repaint();

}

void removeNode(int data) {

avlTree.remove(data);

repaint();

}

void findNode(int data) {

AVLNode<Integer> node = avlTree.find(data);

if (node != null) {

isSearch = true;

toSearch = String.valueOf(node.data);

}

repaint();

System.out.println();

}

private void patinGeneralPath(int x, int y, Graphics2D gr2d, String data, int height) {

Ellipse2D ellipse2D = new Ellipse2D.Double(x, y, 35.0, 35.0);

gr2d.setColor(Color.black);

gr2d.draw(ellipse2D);

if (isSearch && data.equals(toSearch)) {

gr2d.setColor(new Color(24, 229, 47));

gr2d.fill(ellipse2D);

gr2d.setColor(Color.black);

gr2d.drawString(data, x + 10, y + 20);

gr2d.drawString(String.valueOf(height + 1), x + 40, (int) (y + 17.5));

isSearch = false;

}

else{

gr2d.setColor(new Color(249, 229, 47));

gr2d.fill(ellipse2D);

gr2d.setColor(Color.black);

gr2d.drawString(data, x + 10, y + 20);

gr2d.drawString(String.valueOf(height + 1), x + 40, (int) (y + 17.5));

}

private Graphics2D graphics2D;

private int count = 0;

private void paintOver(AVLNode<Integer> current, int x, int y, boolean isLeft) {

Graphics2D g2 = graphics2D;

int currentX;

int currentY;

if (current == null) {

return;

}

if (current == avlTree.getRoot()) {

currentX = x;

currentY = y;

patinGeneralPath(x, y, g2, String.valueOf(current.data), current.height);

} else if (isLeft) {

currentX = x - 150 + count;

currentY = y + 60 + count;

patinGeneralPath(currentX, currentY, g2, String.valueOf(current.data), current.height);

GeneralPath generalPath = new GeneralPath();

generalPath.moveTo(x + 17.5, y + 35);

generalPath.lineTo(currentX + 17.5, currentY);

generalPath.closePath();

g2.draw(generalPath);

} else {

currentX = x + 150 - count;

currentY = y + 60 + count;

patinGeneralPath(currentX, currentY, g2, String.valueOf(current.data), current.height);

GeneralPath generalPath = new GeneralPath();

generalPath.moveTo(x + 17.5, y + 35);

generalPath.lineTo(currentX + 17.5, currentY);

generalPath.closePath();

g2.draw(generalPath);

}

count += 30;

paintOver(current.left, currentX, currentY, true);

paintOver(current.right, currentX, currentY, false);

count -= 30;

}

@Override

protected void paintComponent(Graphics g) {

if (avlTree.getRoot() == null) return;

graphics2D = (Graphics2D) g;

paintOver(avlTree.getRoot(), fistX, fistY, true);

}

@Override

public String toString() {

return avlTree.toString();

}

}

**MainFraim.java**

import javax.swing.\*;

import javax.swing.border.LineBorder;

import java.awt.\*;

import java.awt.event.ActionListener;

public class MainFrame extends JFrame {

private JPanel optionPanel;

private Board board;

private String elements = "BINARY AVL TREE";

private boolean isSearch = false;

private void initMainPanel() {

board = new Board((int) ((this.getWidth() - 200) / 2 - 17.5), 15);

board.setBackground(new Color(180, 180, 180));

board.setBorder(new LineBorder(new Color(48, 56, 58), 1));

this.add(board, BorderLayout.CENTER);

}

private void initHeadPanel() {

JPanel headPanel = new JPanel();

headPanel.setPreferredSize(new Dimension(100, 30));

headPanel.setBorder(new LineBorder(new Color(48, 56, 58), 1));

headPanel.setBackground(new Color(21, 249, 48));

JLabel headLabel = new JLabel("BINARY AVL TREE");

headLabel.setHorizontalAlignment(SwingConstants.CENTER);

headPanel.add(headLabel);

this.add(headPanel, BorderLayout.NORTH);

}

private void showWarring() {

JOptionPane.showOptionDialog(this,

elements, "WARRING", JOptionPane.OK\_CANCEL\_OPTION,

JOptionPane.INFORMATION\_MESSAGE, null, null, "default");

}

private void initRadioPanel() {

JPanel radioPanel = new JPanel();

radioPanel.setBorder(new LineBorder(Color.blue, 1));

radioPanel.setPreferredSize(new Dimension(90, 100));

radioPanel.setBackground(new Color(29, 249, 50));

radioPanel.setLayout(new FlowLayout(FlowLayout.CENTER));

JLabel head = new JLabel("INSERT");

radioPanel.add(head);

JTextField textInsert = new JTextField(7);

radioPanel.add(textInsert);

ActionListener actionListener = e -> {

if (!textInsert.getText().matches("-?[0-9]+")) {

elements = "You can only enter integers";

this.showWarring();

return;

}

board.insertNode(Integer.parseInt(textInsert.getText()));

textInsert.setText("");

repaint();

};

JButton buttonInsert = new JButton("Enter");

buttonInsert.addActionListener(actionListener);

radioPanel.add(buttonInsert);

optionPanel.add(radioPanel);

JPanel removePanel = new JPanel();

removePanel.setBorder(new LineBorder(Color.WHITE, 1));

removePanel.setPreferredSize(new Dimension(90, 100));

removePanel.setBackground(new Color(22, 249, 33));

removePanel.setLayout(new FlowLayout(FlowLayout.CENTER));

JLabel remove = new JLabel("REMOVE");

removePanel.add(remove);

JTextField textRemove = new JTextField(7);

removePanel.add(textRemove);

ActionListener listener = e -> {

if (!textRemove.getText().matches("-?[0-9]+")) {

elements = "You can only enter integers";

this.showWarring();

return;

}

board.removeNode(Integer.parseInt(textRemove.getText()));

textRemove.setText("");

repaint();

};

JButton buttonRemove = new JButton("Enter");

buttonRemove.addActionListener(listener);

removePanel.add(buttonRemove);

optionPanel.add(removePanel);

JPanel findPanel = new JPanel();

findPanel.setBorder(new LineBorder(Color.blue, 1));

findPanel.setPreferredSize(new Dimension(90, 100));

findPanel.setBackground(new Color(18, 249, 78));

findPanel.setLayout(new FlowLayout(FlowLayout.CENTER));

JLabel find = new JLabel(" FIND ");

findPanel.add(find);

JTextField textFind = new JTextField(7);

findPanel.add(textFind);

ActionListener listener1 = e -> {

if (!textFind.getText().matches("-?[0-9]+")) {

elements = "You can only enter integers";

this.showWarring();

return;

}

board.findNode(Integer.parseInt(textFind.getText()));

textFind.setText("");

this.repaint();

if (!isSearch) {

elements = " You found node ";

this.showWarring();

}

};

JButton buttonFind = new JButton("Enter");

buttonFind.addActionListener(listener1);

findPanel.add(buttonFind);

optionPanel.add(findPanel);

JPanel travelPanel = new JPanel();

travelPanel.setBorder(new LineBorder(Color.blue, 1));

travelPanel.setPreferredSize(new Dimension(90, 70));

travelPanel.setBackground(new Color(19, 249, 54));

travelPanel.setLayout(new FlowLayout(FlowLayout.CENTER));

JLabel travel = new JLabel(" PRINT TREE ");

travelPanel.add(travel);

ActionListener aL = e -> {

elements = board.toString();

this.showWarring();

};

JButton buttonTravel = new JButton("Enter");

buttonTravel.addActionListener(aL);

travelPanel.add(buttonTravel);

optionPanel.add(travelPanel);

}

private void initOptionPanel() {

optionPanel = new JPanel();

optionPanel.setPreferredSize(new Dimension(600, 90));

optionPanel.setBorder(new LineBorder(Color.PINK, 1));

optionPanel.setBackground(new Color(24, 112, 24, 255));

initRadioPanel();

JButton exitButton = new JButton("Exit");

exitButton.addActionListener(e -> System.exit(0));

optionPanel.add(exitButton);

ActionListener actionListener = e -> {

board.clearBoard();

repaint();

};

JButton clear = new JButton("Clear");

clear.addActionListener(actionListener);

optionPanel.add(clear);

this.add(optionPanel, BorderLayout.SOUTH);

}

private MainFrame(String s) {

super(s);

setSize(1000, 800);

this.setLayout(new BorderLayout());

initMainPanel();

initHeadPanel();

initOptionPanel();

setVisible(true);

this.setLocationRelativeTo(null);

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

}

public static void main(String[] args) {

MainFrame mainFrame = new MainFrame("AVL Tree");

}

}

**RunAVL.java**

public class RunAVL {

public static void main(String[] args) {

AvlTree<Integer> avlTree = new AvlTree<>();

avlTree.add(40);

avlTree.add(41);

avlTree.add(42);

avlTree.add(43);

avlTree.add(44);

avlTree.add(45);

System.out.println(avlTree.toString());

}

}

Тестирование

Древовидные тесты написаны в классе “AvlTreeTest.java”.

import org.junit.jupiter.api.Test;

import java.util.\*;

import static org.junit.jupiter.api.Assertions.\*;

class AvlTreeTest {

private Set<Integer> avlTree = new AvlTree<>();

private void insert() {

avlTree.add(1);

avlTree.add(2);

avlTree.add(3);

avlTree.add(4);

avlTree.add(5);

avlTree.add(6);

avlTree.add(7);

avlTree.add(8);

avlTree.add(9);

avlTree.add(10);

}

@Test

void contains() {

insert();

for (int i = 0; i < 10; i++)

assertTrue(avlTree.contains(i + 1));

}

@Test

void add() {

insert();

for (int i = 0; i < 10; i++)

assertTrue(avlTree.contains(i + 1));

}

@Test

void size() {

assertEquals(0, avlTree.size());

insert();

assertEquals(10, avlTree.size());

}

@Test

void isEmpty() {

assertTrue(avlTree.isEmpty());

insert();

assertFalse(avlTree.isEmpty());

}

@Test

void remove() {

insert();

avlTree.remove(5);

avlTree.remove(7);

assertFalse(avlTree.contains(5));

assertFalse(avlTree.contains(7));

}

@Test

void iterator() {

Set<Integer> tree = new AvlTree<>();

tree.add(5);

tree.add(1);

tree.add(2);

tree.add(7);

tree.add(9);

tree.add(10);

tree.add(8);

tree.add(4);

tree.add(3);

tree.add(6);

Iterator<Integer> it = tree.iterator();

for (int i = 1; i < 11; i++) {

assertEquals(i, (int) it.next());

}

Iterator<Integer> int2 = tree.iterator();

while (int2.hasNext()) {

int2.next();

int2.remove();

}

assertEquals(0, tree.size());

}

@Test

void toArray() {

Set<String> tree = new AvlTree<>();

tree.add("don");

tree.add("doc");

tree.add("se");

tree.add("giup");

tree.add("ban");

tree.add("chin");

tree.add("chan");

tree.add("hon");

tree.add("nghin");

tree.add("lan");

String[] strings = new String[10];

strings[0] = "ban";

strings[1] = "chan";

strings[2] = "chin";

strings[3] = "doc";

strings[4] = "don";

strings[5] = "giup";

strings[6] = "hon";

strings[7] = "lan";

strings[8] = "nghin";

strings[9] = "se";

assertTrue(Arrays.equals(tree.toArray(), strings));

}

@Test

void containsAll() {

insert();

List<Integer> list = new ArrayList<>();

list.add(3);

list.add(4);

list.add(5);

list.add(6);

assertTrue(avlTree.containsAll(list));

list.add(23);

assertFalse(avlTree.containsAll(list));

}

@Test

void addAll() {

insert();

List<Integer> list = new ArrayList<>();

list.add(3);

list.add(4);

list.add(5);

list.add(6);

assertFalse(avlTree.addAll(list));

list.add(12);

list.add(13);

list.add(14);

list.add(15);

assertTrue(avlTree.addAll(list));

for (int i = 12; i < 16; i++) {

assertTrue(avlTree.contains(i));

}

}

@Test

void retainAll() {

insert();

List<Integer> list = new ArrayList<>();

list.add(3);

list.add(4);

list.add(5);

list.add(6);

avlTree.retainAll(list);

assertTrue(avlTree.containsAll(list));

assertFalse(avlTree.contains(1));

assertFalse(avlTree.contains(2));

assertFalse(avlTree.contains(7));

assertFalse(avlTree.contains(8));

assertFalse(avlTree.contains(9));

assertFalse(avlTree.contains(10));

}

@Test

void removeAll() {

insert();

List<Integer> list = new ArrayList<>();

list.add(1);

list.add(2);

list.add(3);

list.add(4);

avlTree.removeAll(list);

assertFalse(avlTree.containsAll(list ));

}

@Test

void clear() {

insert();

avlTree.clear();

assertTrue(avlTree.isEmpty());

}

}

Скриншоты программы:

- Создать новое дерево и добавлять поочередно элементы : 40, 41, 42, 43, 44, 45, 46

По определению мы получим такое дерево:

A screenshot of a video game

Description automatically generated

Рис 1: Создание нового дерева

- Теперь если мы удалим 43 и 45, то получим такое дерево :

A picture containing screenshot, green

Description automatically generated

Рис 2: Удаления узла

Теперь мы найдем узел 45:

A screenshot of a video game

Description automatically generated

Рис 3: Поиск узла