Бюджетное учреждение высшего образования Ханты-Мансийского автономного округа – Югры «СУРГУТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт

Кафедра автоматики и компьютерных систем

Отчет

по лабораторной работе № 6

по дисциплине «Алгоритмы и структуры данных»

Проверила: старший преподаватель кафедры АиКС,

Карамышев. Э. Р

Выполнил: студент группы 605-31,

Хайитов Ш. Д.

Сургут 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc194868567)

[ОСНОВНАЯ ЧАСТЬ 4](#_Toc194868568)

[ЗАКЛЮЧЕНИЕ 11](#_Toc194868569)

# **ВВЕДЕНИЕ**

**Цель:** изучить основные принципы построения сбалансированных и цифровых деревьев, их свойства и назначение, закрепить навыки структурного программирования.

Вариант данный преподавателем представлен на рисунке 1.



Рисунок 1 – Варинат 3

# **ОСНОВНАЯ ЧАСТЬ**

Код написан на js, html и css, так как важна визуализация. Логика написана на js, а отображение уже на html и css.

**Описание функций скрипта:**

1. **Node(key)** - Конструктор узла дерева, создает узел с заданным ключом, цветом RED, инициализирует потомков как NIL.
2. **RBTree()** - Конструктор красно-черного дерева, инициализирует корень как NIL-узел.
3. **insert(key)** - Вставляет новый узел с заданным ключом в дерево, сохраняя свойства BST.
4. **fixInsert(node)** - Восстанавливает свойства красно-черного дерева после вставки нового узла.
5. **rotateLeft(x)** - Выполняет левый поворот вокруг узла x для балансировки дерева.
6. **rotateRight(x)** - Выполняет правый поворот вокруг узла x для балансировки дерева.
7. **calculatePositions(node, x, y, levelWidth)** - Рекурсивно вычисляет координаты для отрисовки всех узлов дерева.
8. **drawConnectors(node, container)** - Рисует соединительные линии между узлами дерева.
9. **drawNodes(node, container)** - Создает и размещает DOM-элементы для всех узлов дерева.
10. **insertKey()** - Обрабатывает ввод пользователя, добавляет ключ в дерево и обновляет отрисовку.
11. **clearTree()** - Очищает дерево, сбрасывая корень в NIL и обновляя отображение.
12. **drawTree()** - Координирует процесс отрисовки дерева (позиции, связи, узлы).

Интерфейс готовой программы представлен на рисунке 2.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 2 – Интерфейс программы.

**Листинг программы:**

**<!**DOCTYPE html**>**

**<**html**>**

**<**head**>**

**<**title**>**Красно**-**черное дерево**</**title**>**

**<**style**>**

body **{**

font**-**family**:** Arial**,** sans**-**serif**;**

margin**:** 20px**;**

**}**

**.**tree**-**container **{**

position**:** relative**;**

height**:** 600px**;**

width**:** 800px**;**

margin**:** 20px auto**;**

border**:** 1px solid #eee**;**

overflow**:** auto**;**

**}**

**.**node **{**

position**:** absolute**;**

width**:** 40px**;**

height**:** 40px**;**

border**-**radius**:** 50**%;**

display**:** flex**;**

align**-**items**:** center**;**

justify**-**content**:** center**;**

color**:** white**;**

font**-**weight**:** bold**;**

z**-**index**:** 2**;**

**}**

**.**red **{**

background**:** #e74c3c**;**

border**:** 2px solid #c0392b**;**

**}**

**.**black **{**

background**:** #2c3e50**;**

border**:** 2px solid #1a1a1a**;**

**}**

**.**nil **{**

width**:** 30px**;**

height**:** 30px**;**

border**:** 1px dashed #95a5a6**;**

color**:** #95a5a6**;**

font**-**size**:** 0.8em**;**

**}**

**.**connector **{**

position**:** absolute**;**

background**:** #7f8c8d**;**

z**-**index**:** 1**;**

**}**

**.**controls **{**

margin**:** 20px**;**

padding**:** 15px**;**

background**:** #ecf0f1**;**

border**-**radius**:** 5px**;**

text**-**align**:** center**;**

**}**

input**,** button **{**

padding**:** 8px 12px**;**

margin**:** 0 5px**;**

font**-**size**:** 16px**;**

**}**

button **{**

cursor**:** pointer**;**

**}**

**</**style**>**

**</**head**>**

**<**body**>**

**<**div class**=**"controls"**>**

**<**input id**=**"keyInput" type**=**"number" placeholder**=**"Введите число"**>**

**<**button onclick**=**"insertKey()"**>**Добавить**</**button**>**

**<**button onclick**=**"clearTree()"**>**Очистить**</**button**>**

**</**div**>**

**<**div id**=**"tree" class**=**"tree-container"**></**div**>**

**<**script**>**

const Color **=** **{** RED**:** 'RED'**,** BLACK**:** 'BLACK' **};**

const nil **=** **{** key**:** 'NIL'**,** color**:** Color**.**BLACK**,** isNil**:** **true** **};**

const NODE\_SIZE **=** 40**;**

const LEVEL\_HEIGHT **=** 80**;**

const HORIZONTAL\_SPACING **=** 60**;**

class Node **{**

constructor**(**key**)** **{**

this**.**key **=** key**;**

this**.**color **=** Color**.**RED**;**

this**.**left **=** nil**;**

this**.**right **=** nil**;**

this**.**parent **=** null**;**

this**.**x **=** 0**;**

this**.**y **=** 0**;**

**}**

**}**

class RBTree **{**

constructor**()** **{**

this**.**root **=** nil**;**

**}**

insert**(**key**)** **{**

const node **=** new Node**(**key**);**

let parent **=** null**;**

let current **=** this**.**root**;**

**while** **(**current **!==** nil**)** **{**

parent **=** current**;**

current **=** key **<** current**.**key **?** current**.**left **:** current**.**right**;**

**}**

node**.**parent **=** parent**;**

**if** **(!**parent**)** this**.**root **=** node**;**

**else** **if** **(**key **<** parent**.**key**)** parent**.**left **=** node**;**

**else** parent**.**right **=** node**;**

**if** **(!**node**.**parent**)** **{**

node**.**color **=** Color**.**BLACK**;**

**return;**

**}**

**if** **(!**node**.**parent**.**parent**)** **return;**

this**.**fixInsert**(**node**);**

**}**

fixInsert**(**node**)** **{**

**while** **(**node**.**parent**.**color **===** Color**.**RED**)** **{**

const parentIsLeft **=** node**.**parent **===** node**.**parent**.**parent**.**left**;**

const uncle **=** parentIsLeft

**?** node**.**parent**.**parent**.**right

**:** node**.**parent**.**parent**.**left**;**

**if** **(**uncle**.**color **===** Color**.**RED**)** **{**

node**.**parent**.**color **=** Color**.**BLACK**;**

uncle**.**color **=** Color**.**BLACK**;**

node**.**parent**.**parent**.**color **=** Color**.**RED**;**

node **=** node**.**parent**.**parent**;**

**}** **else** **{**

**if** **((**node **===** node**.**parent**.**right**)** **===** parentIsLeft**)** **{**

node **=** node**.**parent**;**

parentIsLeft **?** this**.**rotateLeft**(**node**)** **:** this**.**rotateRight**(**node**);**

**}**

node**.**parent**.**color **=** Color**.**BLACK**;**

node**.**parent**.**parent**.**color **=** Color**.**RED**;**

parentIsLeft **?** this**.**rotateRight**(**node**.**parent**.**parent**)** **:** this**.**rotateLeft**(**node**.**parent**.**parent**);**

**}**

**if** **(**node **===** this**.**root**)** **break;**

**}**

this**.**root**.**color **=** Color**.**BLACK**;**

**}**

rotateLeft**(**x**)** **{**

const y **=** x**.**right**;**

x**.**right **=** y**.**left**;**

**if** **(**y**.**left **!==** nil**)** y**.**left**.**parent **=** x**;**

y**.**parent **=** x**.**parent**;**

**if** **(!**x**.**parent**)** this**.**root **=** y**;**

**else** **if** **(**x **===** x**.**parent**.**left**)** x**.**parent**.**left **=** y**;**

**else** x**.**parent**.**right **=** y**;**

y**.**left **=** x**;**

x**.**parent **=** y**;**

**}**

rotateRight**(**x**)** **{**

const y **=** x**.**left**;**

x**.**left **=** y**.**right**;**

**if** **(**y**.**right **!==** nil**)** y**.**right**.**parent **=** x**;**

y**.**parent **=** x**.**parent**;**

**if** **(!**x**.**parent**)** this**.**root **=** y**;**

**else** **if** **(**x **===** x**.**parent**.**right**)** x**.**parent**.**right **=** y**;**

**else** x**.**parent**.**left **=** y**;**

y**.**right **=** x**;**

x**.**parent **=** y**;**

**}**

**}**

const tree **=** new RBTree**();**

function calculatePositions**(**node**,** x**,** y**,** levelWidth**)** **{**

**if** **(**node **===** nil**)** **return;**

node**.**x **=** x**;**

node**.**y **=** y**;**

**if** **(**node**.**left **!==** nil**)** **{**

calculatePositions**(**node**.**left**,** x **-** levelWidth**/**2**,** y **+** LEVEL\_HEIGHT**,** levelWidth**/**2**);**

**}**

**if** **(**node**.**right **!==** nil**)** **{**

calculatePositions**(**node**.**right**,** x **+** levelWidth**/**2**,** y **+** LEVEL\_HEIGHT**,** levelWidth**/**2**);**

**}**

**}**

function drawTree**()** **{**

const container **=** document**.**getElementById**(**'tree'**);**

container**.**innerHTML **=** ''**;**

**if** **(**tree**.**root **===** nil**)** **{**

container**.**innerHTML **=** '<p style="text-align: center; margin-top: 20px;">Дерево пусто</p>'**;**

**return;**

**}**

calculatePositions**(**tree**.**root**,** container**.**offsetWidth**/**2**,** 30**,** container**.**offsetWidth**/**3**);**

drawConnectors**(**tree**.**root**,** container**);**

drawNodes**(**tree**.**root**,** container**);**

**}**

function drawConnectors**(**node**,** container**)** **{**

**if** **(**node **===** nil**)** **return;**

**if** **(**node**.**left **!==** nil**)** **{**

const connector **=** document**.**createElement**(**'div'**);**

connector**.**className **=** 'connector'**;**

const length **=** Math**.**sqrt**(**

Math**.**pow**(**node**.**left**.**x **-** node**.**x**,** 2**)** **+**

Math**.**pow**(**node**.**left**.**y **-** node**.**y**,** 2**)**

**);**

const angle **=** Math**.**atan2**(**

node**.**left**.**y **-** node**.**y**,**

node**.**left**.**x **-** node**.**x

**)** **\*** 180 **/** Math**.**PI**;**

connector**.**style**.**width **=** `$**{**length**}**px`**;**

connector**.**style**.**height **=** '2px'**;**

connector**.**style**.**left **=** `$**{**node**.**x **+** NODE\_SIZE**/**2**}**px`**;**

connector**.**style**.**top **=** `$**{**node**.**y **+** NODE\_SIZE**/**2**}**px`**;**

connector**.**style**.**transformOrigin **=** '0 0'**;**

connector**.**style**.**transform **=** `rotate**(**$**{**angle**}**deg**)**`**;**

container**.**appendChild**(**connector**);**

drawConnectors**(**node**.**left**,** container**);**

**}**

**if** **(**node**.**right **!==** nil**)** **{**

const connector **=** document**.**createElement**(**'div'**);**

connector**.**className **=** 'connector'**;**

const length **=** Math**.**sqrt**(**

Math**.**pow**(**node**.**right**.**x **-** node**.**x**,** 2**)** **+**

Math**.**pow**(**node**.**right**.**y **-** node**.**y**,** 2**)**

**);**

const angle **=** Math**.**atan2**(**

node**.**right**.**y **-** node**.**y**,**

node**.**right**.**x **-** node**.**x

**)** **\*** 180 **/** Math**.**PI**;**

connector**.**style**.**width **=** `$**{**length**}**px`**;**

connector**.**style**.**height **=** '2px'**;**

connector**.**style**.**left **=** `$**{**node**.**x **+** NODE\_SIZE**/**2**}**px`**;**

connector**.**style**.**top **=** `$**{**node**.**y **+** NODE\_SIZE**/**2**}**px`**;**

connector**.**style**.**transformOrigin **=** '0 0'**;**

connector**.**style**.**transform **=** `rotate**(**$**{**angle**}**deg**)**`**;**

container**.**appendChild**(**connector**);**

drawConnectors**(**node**.**right**,** container**);**

**}**

**}**

function drawNodes**(**node**,** container**)** **{**

**if** **(**node **===** nil**)** **return;**

const nodeElement **=** document**.**createElement**(**'div'**);**

nodeElement**.**className **=** `node $**{**node**.**color**.**toLowerCase**()}**`**;**

nodeElement**.**textContent **=** node**.**key**;**

nodeElement**.**style**.**left **=** `$**{**node**.**x**}**px`**;**

nodeElement**.**style**.**top **=** `$**{**node**.**y**}**px`**;**

container**.**appendChild**(**nodeElement**);**

**if** **(**node**.**left **===** nil**)** **{**

const nilNode **=** document**.**createElement**(**'div'**);**

nilNode**.**className **=** 'node nil'**;**

nilNode**.**textContent **=** 'NIL'**;**

nilNode**.**style**.**left **=** `$**{**node**.**x **-** NODE\_SIZE**}**px`**;**

nilNode**.**style**.**top **=** `$**{**node**.**y **+** LEVEL\_HEIGHT**}**px`**;**

container**.**appendChild**(**nilNode**);**

**}** **else** **{**

drawNodes**(**node**.**left**,** container**);**

**}**

**if** **(**node**.**right **===** nil**)** **{**

const nilNode **=** document**.**createElement**(**'div'**);**

nilNode**.**className **=** 'node nil'**;**

nilNode**.**textContent **=** 'NIL'**;**

nilNode**.**style**.**left **=** `$**{**node**.**x **+** NODE\_SIZE**}**px`**;**

nilNode**.**style**.**top **=** `$**{**node**.**y **+** LEVEL\_HEIGHT**}**px`**;**

container**.**appendChild**(**nilNode**);**

**}** **else** **{**

drawNodes**(**node**.**right**,** container**);**

**}**

**}**

function insertKey**()** **{**

const key **=** parseInt**(**document**.**getElementById**(**'keyInput'**).**value**);**

**if** **(!**isNaN**(**key**))** **{**

tree**.**insert**(**key**);**

drawTree**();**

document**.**getElementById**(**'keyInput'**).**value **=** ''**;**

**}** **else** **{**

alert**(**'Пожалуйста, введите число'**);**

**}**

**}**

function clearTree**()** **{**

tree**.**root **=** nil**;**

drawTree**();**

**}**

// Инициализация при загрузке

window**.**onload **=** function**()** **{**

drawTree**();**

document**.**getElementById**(**'keyInput'**).**focus**();**

**};**

**</**script**>**

**</**body**>**

**</**html**>**

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения работы была успешно реализована структура данных "красно-черное дерево" на языке JavaScript с визуализацией в браузере. Разработанная программа позволяет наглядно демонстрировать основные операции вставки узлов с автоматической балансировкой дерева, сохраняя все свойства красно-черного дерева