Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный** **исследовательский политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

направление подготовки: 09.03.01– «Информатика и вычислительная техника»

**Лабораторная работа**

**по дисциплине**

**«Теория алгоритмов и структуры данных»**

**на тему**

**«Бинарные деревья»**

**Вариант 6**

Выполнил студент гр. ИВТ-23-1б

Бушмаков Кирилл Дмитриевич

Проверил:

Доцент каф. ИТАС

Яруллин Денис Владимирович

(оценка) (подпись)

(дата)

г. Пермь, 2024

**Цель и задачи работы**

Целью данной работы является получение навыков работы с бинарными деревьями

**Вариант 6:** Тип информационного поля int. Найти среднее арифметическое в дереве

**UML диаграмма**

На рисунке 1 изображена диаграмма класса

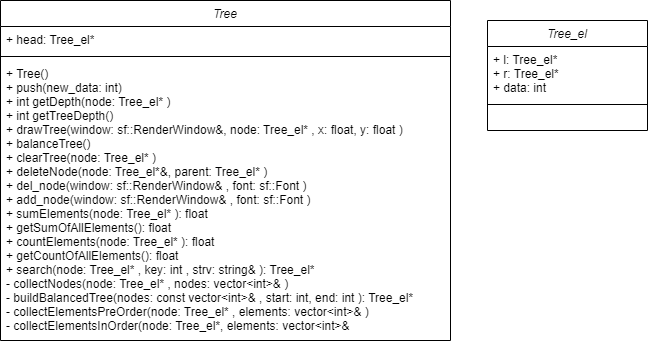


Рисунок 1

**Код программы**

**Файл tree.cpp**

#include <SFML/Graphics.hpp>

#include <iostream>

#include <vector>

#include <algorithm>

#include <thread>

#include <sstream>

#include <string>

#include <chrono>

using namespace std;

struct Tree\_el

{

Tree\_el\* l;

Tree\_el\* r;

int data;

};

class Tree

{

public:

Tree\_el\* head;

Tree()

{

head = new Tree\_el;

head->l = nullptr;

head->r = nullptr;

}

void push(int new\_data)

{

Tree\_el\* tmp = this->head;

while (tmp != nullptr)

{

if (tmp->data <= new\_data && tmp->r == nullptr)

{

Tree\_el\* tmp2 = new Tree\_el;

tmp2->data = new\_data;

tmp2->r = nullptr;

tmp2->l = nullptr;

tmp->r = tmp2;

break;

}

else if (tmp->data > new\_data && tmp->l == nullptr)

{

Tree\_el\* tmp2 = new Tree\_el;

tmp2->r = nullptr;

tmp2->l = nullptr;

tmp2->data = new\_data;

tmp->l = tmp2;

break;

}

else if (tmp->data <= new\_data && tmp->r != nullptr)

{

tmp = tmp->r;

}

else if (tmp->data > new\_data && tmp->l != nullptr)

{

tmp = tmp->l;

}

}

}

int getDepth(Tree\_el\* node)

{

if (node == nullptr)

{

return 0;

}

int leftDepth = getDepth(node->l);

int rightDepth = getDepth(node->r);

return 1 + max(leftDepth, rightDepth);

}

int getTreeDepth()

{

return getDepth(head);

}

void drawTree(sf::RenderWindow& window, Tree\_el\* node, float x, float y)

{

if (node == nullptr)

{

return;

}

sf::Color greyColor(128, 128, 128);

sf::CircleShape circle(20);

circle.setFillColor(greyColor);

circle.setPosition(x, y);

window.draw(circle);

int offsetX\_l = 30 \* pow(2, getDepth(node->l)-1);

int offsetX\_r = 30 \* pow(2, getDepth(node->r) - 1);

int offsetY = 70;

sf::Font font;

font.loadFromFile("C:/Windows/Fonts/arial.ttf");

sf::Text text(to\_string(node->data), font, 20);

text.setFillColor(sf::Color::Green);

if(abs(node->data)<10){ text.setPosition(x+13, y + 7); }

else if (abs(node->data) >= 100) { text.setPosition(x + 4, y + 7); }

else { text.setPosition(x+7, y + 7); }

window.draw(text);

if (node->l != nullptr)

{

sf::Vertex line[] =

{

sf::Vertex(sf::Vector2f(x, y + 20), sf::Color(128, 128, 128)),

sf::Vertex(sf::Vector2f(x - offsetX\_l + 20, y + offsetY), sf::Color(128, 128, 128))

};

window.draw(line, 2, sf::Lines);

drawTree(window, node->l, x - offsetX\_l, y + offsetY);

}

if (node->r != nullptr)

{

sf::Vertex line[] =

{

sf::Vertex(sf::Vector2f(x + 40, y + 20), sf::Color(128, 128, 128)),

sf::Vertex(sf::Vector2f(x + offsetX\_r + 20, y + offsetY), sf::Color(128, 128, 128))

};

window.draw(line, 2, sf::Lines);

drawTree(window, node->r, x + offsetX\_r, y + offsetY);

}

}

void balanceTree()

{

std::vector<int> nodes;

collectNodes(head, nodes);

clearTree(head);

std::sort(nodes.begin(), nodes.end());

head = buildBalancedTree(nodes, 0, nodes.size() - 1);

}

void clearTree(Tree\_el\* node)

{

if (node == nullptr)

{

return;

}

clearTree(node->l);

clearTree(node->r);

delete node;

}

void deleteNode(Tree\_el\*& node, Tree\_el\* parent)

{

if (node == nullptr)

return;

deleteNode(node->l, node);

deleteNode(node->r, node);

if (parent != nullptr)

{

if (parent->l == node)

parent->l = nullptr;

else if (parent->r == node)

parent->r = nullptr;

}

delete node;

node = nullptr;

}

void del\_node(sf::RenderWindow& window, sf::Font font)

{

Tree\_el\* ntd = head;

Tree\_el\* parent = nullptr;

bool delte = true;

sf::String unicodeString = L"Введите путь к узлу который будет удален, путь должен состоять из симловл 'R' и 'L'\nR - переход к правому лементу\nL - переход к левому элементу";

sf::Text text(unicodeString, font, 30);

text.setFont(font);

text.setFillColor(sf::Color::White);

text.setPosition(200, 100);

sf::Text inputText("", font, 30);

inputText.setFillColor(sf::Color::White);

inputText.setPosition(200, 215);

sf::String uunicodeString = L"Некорректный путь к узлу";

sf::Text erroText(uunicodeString, font, 30);

erroText.setFont(font);

erroText.setFillColor(sf::Color::White);

erroText.setPosition(200, 100);

string inputString = "";

bool isInputting = true;

while (isInputting)

{

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::TextEntered) {

if (event.text.unicode < 128) {

if (event.text.unicode == 'L' || event.text.unicode == 'R') {

inputString += static\_cast<char>(event.text.unicode);

}

else if (event.text.unicode == '\b' && inputString.size() > 0) {

inputString.pop\_back();

}

inputText.setString(inputString);

}

}

if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Enter) {

isInputting = false;

if (inputString == "") { inputString = "0"; }

}

}

window.clear();

window.draw(inputText);

window.draw(text);

window.display();

}

string s = inputString;

for (int i = 0; i < s.size(); i++)

{

if (ntd == nullptr)

{

cout << "некоректная позиция1\n";

window.clear();

window.draw(erroText);

window.display();

delte = false;

this\_thread::sleep\_for(std::chrono::seconds(3));

break;

}

parent = ntd;

if (s[i] == 'L') {

ntd = ntd->l;

}

else if (s[i] == 'R') {

ntd = ntd->r;

}

else if (s[i] != 'L' and s[i] != 'R')

{

cout << "некоректная позиция2\n";

window.clear();

window.draw(erroText);

window.display();

delte = false;

this\_thread::sleep\_for(std::chrono::seconds(3));

break;

}

}

if (delte)

{

deleteNode(ntd, parent);

}

}

void add\_node(sf::RenderWindow& window, sf::Font font)

{

string bcanW[61];

std::string inputString1 = "";

for (int i = 0; i < 61; ++i) {

bcanW[i] = std::to\_string(-30 + i);

}

bool isInputting11 = true;

sf::Text inputText1("", font, 30);

inputText1.setFillColor(sf::Color::White);

inputText1.setPosition(999, 150);

sf::String unicodeString1 = L"Введите количество элементов в новом дереве\n(не менее одного)\nесли не будет введено ничего, будет задано 7 элементов";

sf::Text outText1(unicodeString1, font, 30);

outText1.setFillColor(sf::Color::White);

outText1.setPosition(300, 150);

while (isInputting11)

{

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::TextEntered) {

if (event.text.unicode < 128) {

if (event.text.unicode == '\b' && inputString1.size() > 0) {

inputString1.pop\_back();

}

else if (inputString1.size() == 0) {

if (event.text.unicode == '1' || event.text.unicode == '2' || event.text.unicode == '3' || event.text.unicode == '4' || event.text.unicode == '5' || event.text.unicode == '6' || event.text.unicode == '7' || event.text.unicode == '8' || event.text.unicode == '9') {

inputString1 += static\_cast<char>(event.text.unicode);

}

}

else if (inputString1.size() > 0 && inputString1.size() <= 8) {

if (event.text.unicode == '1' || event.text.unicode == '2' || event.text.unicode == '3' || event.text.unicode == '4' || event.text.unicode == '5' || event.text.unicode == '6' || event.text.unicode == '7' || event.text.unicode == '8' || event.text.unicode == '9' || event.text.unicode == '0') {

inputString1 += static\_cast<char>(event.text.unicode);

}

}

inputText1.setString(inputString1);

}

}

if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Enter) {

isInputting11 = false;

if (inputString1 == "") { inputString1 = "8"; }

}

}

window.clear();

window.draw(inputText1);

window.draw(outText1);

window.display();

}

int n = stoi(inputString1);

std::string inputString2 = "";

bool isInputting12 = true;

sf::Text inputText2("", font, 30);

inputText2.setFillColor(sf::Color::White);

inputText2.setPosition(970, 150);

sf::String unicodeString3 = L"Введите \"корневой\" элемент нового дерева\nесли не будет введено значение, оно будет задано случайно от -30 до 30";

sf::Text outText3(unicodeString3, font, 30);

outText3.setFillColor(sf::Color::White);

outText3.setPosition(300, 150);

while (isInputting12)

{

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::TextEntered) {

if (event.text.unicode < 128) {

if (event.text.unicode == '\b' && inputString2.size() > 0) {

inputString2.pop\_back();

}

else if (inputString2.size() == 0) {

if (event.text.unicode == '1' || event.text.unicode == '2' || event.text.unicode == '3' || event.text.unicode == '4' || event.text.unicode == '5' || event.text.unicode == '6' || event.text.unicode == '7' || event.text.unicode == '8' || event.text.unicode == '9' || event.text.unicode == '0' || event.text.unicode == '-') {

inputString2 += static\_cast<char>(event.text.unicode); }

}

else if (inputString2.size() > 0 && inputString2.size() <= 8) {

if (event.text.unicode == '1' || event.text.unicode == '2' || event.text.unicode == '3' || event.text.unicode == '4' || event.text.unicode == '5' || event.text.unicode == '6' || event.text.unicode == '7' || event.text.unicode == '8' || event.text.unicode == '9' || event.text.unicode == '0') {

inputString2 += static\_cast<char>(event.text.unicode);

}

}

inputText2.setString(inputString2);

}

}

if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Enter) {

isInputting12 = false;

if (inputString2 == "" or inputString2 == "-") { int ara\_ara = rand() % 61; inputString2 = bcanW[ara\_ara]; }

}

}

window.clear();

window.draw(inputText2);

window.draw(outText3);

window.display();

}

int a1 = stoi(inputString2);

Tree tra;

tra.head->data = a1;

for (int i = 2; i <= n; i++)

{

std::string inputString3 = "";

bool isInputting13 = true;

sf::Text inputText3("", font, 30);

inputText3.setFillColor(sf::Color::White);

inputText3.setPosition(650, 150);

string isp = to\_string(i);

sf::String is = sf::String::fromUtf8(isp.begin(), isp.end());

sf::String unicodeString3 = L"Введите элемент №" + is+L"\nесли не будет введено значение, оно будет задано случайно от -30 до 30";

sf::Text outText4(unicodeString3, font, 30);

outText4.setFillColor(sf::Color::White);

outText4.setPosition(300, 150);

while (isInputting13)

{

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::TextEntered) {

if (event.text.unicode < 128) {

if (event.text.unicode == '\b' && inputString3.size() > 0) {

inputString3.pop\_back();

}

else if (inputString3.size() == 0) {

if (event.text.unicode == '1' || event.text.unicode == '2' || event.text.unicode == '3' || event.text.unicode == '4' || event.text.unicode == '5' || event.text.unicode == '6' || event.text.unicode == '7' || event.text.unicode == '8' || event.text.unicode == '9' || event.text.unicode == '0' || event.text.unicode == '-') {

inputString3 += static\_cast<char>(event.text.unicode); }

}

else if (inputString3.size() > 0 && inputString3.size() <= 8) {

if (event.text.unicode == '1' || event.text.unicode == '2' || event.text.unicode == '3' || event.text.unicode == '4' || event.text.unicode == '5' || event.text.unicode == '6' || event.text.unicode == '7' || event.text.unicode == '8' || event.text.unicode == '9' || event.text.unicode == '0') {

inputString3 += static\_cast<char>(event.text.unicode); }

}

inputText3.setString(inputString3);

}

}

if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Enter) {

isInputting13 = false;

if (inputString3 == "" or inputString3 == "-") { int ara\_ara = rand() % 61; inputString3 = bcanW[ara\_ara]; }

}

}

window.clear();

window.draw(inputText3);

window.draw(outText4);

window.display();

}

int a = stoi(inputString3);

tra.push(a);

}

Tree\_el\* tmp = head;

int t\_data = tra.head->data;

while (true)

{

if (tmp->data >= t\_data)

{

if (tmp->l == nullptr)

{

tmp->l = tra.head;

break;

}

tmp = tmp->l;

}

else if (tmp->data < t\_data)

{

if (tmp->r == nullptr)

{

tmp->r = tra.head;

break;

}

tmp = tmp->r;

}

}

}

float sumElements(Tree\_el\* node)

{

if (node == nullptr)

return 0;

// Рекурсивно считаем суммы левого и правого поддеревьев

int sumLeft = sumElements(node->l);

int sumRight = sumElements(node->r);

// Суммируем значение текущего узла с суммами левого и правого поддеревьев

return node->data + sumLeft + sumRight;

}

float getSumOfAllElements()

{

return sumElements(head);

}

float countElements(Tree\_el\* node)

{

if (node == nullptr)

return 0;

int countLeft = countElements(node->l);

int countRight = countElements(node->r);

return 1 + countLeft + countRight;

}

float getCountOfAllElements()

{

return countElements(head);

}

Tree\_el\* search(Tree\_el\* node, int key, string& strv)

{

if (node == nullptr || node->data == key)

{

return node;

}

if (key < node->data)

{

strv += "L";

return search(node->l, key, strv);

}

else

{

strv += "R";

return search(node->r, key, strv);

}

}

private:

void collectNodes(Tree\_el\* node, std::vector<int>& nodes)

{

if (node == nullptr)

{

return;

}

collectNodes(node->l, nodes);

nodes.push\_back(node->data);

collectNodes(node->r, nodes);

}

Tree\_el\* buildBalancedTree(const std::vector<int>& nodes, int start, int end)

{

if (start > end)

{

return nullptr;

}

int mid = (start + end) / 2;

Tree\_el\* newNode = new Tree\_el;

newNode->data = nodes[mid];

newNode->l = buildBalancedTree(nodes, start, mid - 1);

newNode->r = buildBalancedTree(nodes, mid + 1, end);

return newNode;

}

void collectElementsPreOrder(Tree\_el\* node, std::vector<int>& elements)

{

if (node == nullptr)

return;

elements.push\_back(node->data);

collectElementsPreOrder(node->l, elements);

collectElementsPreOrder(node->r, elements);

}

void collectElementsInOrder(Tree\_el\* node, std::vector<int>& elements)

{

if (node == nullptr)

return;

collectElementsInOrder(node->l, elements);

elements.push\_back(node->data);

collectElementsInOrder(node->r, elements);

}

};

**Файл trees.cpp**

#include <SFML/Graphics.hpp>

#include <iostream>

#include "Tree.cpp"

#include <thread>

#include <sstream>

#include <string>

#include <chrono>

using namespace std;

string bcan[61];

void reverserss(string& str)

{

int left = 0;

int right = str.length() - 1;

while (left < right)

{

swap(str[left], str[right]);

left++;

right--;

}

}

int main()

{

sf::RenderWindow window(sf::VideoMode(1920, 800), "Binary Tree Visualization");

for (int i = 0; i < 61; ++i) {

bcan[i] = std::to\_string(-30 + i);

}

sf::Font font;

font.loadFromFile("C:/Windows/Fonts/arial.ttf");

setlocale(LC\_ALL, "ru\_RU");

std::string inputString1 = "";

bool isInputting11 = true;

sf::Text inputText1("", font, 30);

inputText1.setFillColor(sf::Color::White);

inputText1.setPosition(900, 150);

sf::String unicodeString1 = L"Введите количество элементов в дереве\n(не менее одного)\nесли не будет введено ничего, будет задано 7 элементов";

sf::Text outText1(unicodeString1, font, 30);

outText1.setFillColor(sf::Color::White);

outText1.setPosition(300, 150);

while (isInputting11)

{

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::TextEntered) {

if (event.text.unicode < 128) {

if (event.text.unicode == '\b' && inputString1.size() > 0) {

inputString1.pop\_back();

}

else if (inputString1.size() == 0) {

if (event.text.unicode >= '1' && event.text.unicode <= '9') {

inputString1 += static\_cast<char>(event.text.unicode);

}

}

else if (inputString1.size() > 0 && inputString1.size()<=8) {

if ((event.text.unicode >= '0' && event.text.unicode <= '9')) {

inputString1 += static\_cast<char>(event.text.unicode);

}

}

inputText1.setString(inputString1);

}

}

if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Enter) {

isInputting11 = false;

if (inputString1.empty()) {

inputString1 = "7";

}

}

}

window.clear();

window.draw(inputText1);

window.draw(outText1);

window.display();

}

int n = std::stoi(inputString1);

std::string inputString2 = "";

bool isInputting12 = true;

sf::Text inputText2("", font, 30);

inputText2.setFillColor(sf::Color::White);

inputText2.setPosition(850, 150);

sf::String unicodeString3 = L"Введите \"корневой\" элемент дерева\nесли не будет введено значение, оно будет задано случайно от -30 до 30";

sf::Text outText3(unicodeString3, font, 30);

outText3.setFillColor(sf::Color::White);

outText3.setPosition(300, 150);

while (isInputting12)

{

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::TextEntered) {

if (event.text.unicode < 128) {

if (event.text.unicode == '\b' && inputString2.size() > 0) {

inputString2.pop\_back();

}

else if (inputString2.size() == 0) {

if (event.text.unicode == '1' || event.text.unicode == '2' || event.text.unicode == '3' || event.text.unicode == '4' || event.text.unicode == '5' || event.text.unicode == '6' || event.text.unicode == '7' || event.text.unicode == '8' || event.text.unicode == '9' || event.text.unicode == '0' || event.text.unicode == '-') {

inputString2 += static\_cast<char>(event.text.unicode); }

}

else if (inputString2.size() > 0 && inputString2.size() <= 8) {

if (event.text.unicode == '1' || event.text.unicode == '2' || event.text.unicode == '3' || event.text.unicode == '4' || event.text.unicode == '5' || event.text.unicode == '6' || event.text.unicode == '7' || event.text.unicode == '8' || event.text.unicode == '9' || event.text.unicode == '0') {

inputString2 += static\_cast<char>(event.text.unicode);

}

}

inputText2.setString(inputString2);

}

}

if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Enter) {

isInputting12 = false;

if (inputString2 == "" or inputString2 == "-") { int ara\_ara = rand() % 61; inputString2 = bcan[ara\_ara]; }

}

}

window.clear();

window.draw(inputText2);

window.draw(outText3);

window.display();

}

int a1 = stoi(inputString2);

Tree tr;

tr.head->data=a1;

for (int i = 2; i <= n; i++)

{

std::string inputString3 = "";

bool isInputting13 = true;

sf::Text inputText3("", font, 30);

inputText3.setFillColor(sf::Color::White);

inputText3.setPosition(650, 150);

string isp = to\_string(i);

sf::String is = sf::String::fromUtf8(isp.begin(), isp.end());

sf::String unicodeString3 = L"Введите элемент №"+is+L"\nесли не будет введено значение, оно будет задано случайно от -30 до 30";

sf::Text outText4(unicodeString3, font, 30);

outText4.setFillColor(sf::Color::White);

outText4.setPosition(300, 150);

while (isInputting13)

{

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::TextEntered) {

if (event.text.unicode < 128) {

if (event.text.unicode == '\b' && inputString3.size() > 0) {

inputString3.pop\_back();

}

else if (inputString3.size() == 0) {

if (event.text.unicode == '1' || event.text.unicode == '2' || event.text.unicode == '3' || event.text.unicode == '4' || event.text.unicode == '5' || event.text.unicode == '6' || event.text.unicode == '7' || event.text.unicode == '8' || event.text.unicode == '9' || event.text.unicode == '0' || event.text.unicode == '-') {

inputString3 += static\_cast<char>(event.text.unicode); }

}

else if (inputString3.size() > 0 && inputString3.size()<=8) {

if (event.text.unicode == '1' || event.text.unicode == '2' || event.text.unicode == '3' || event.text.unicode == '4' || event.text.unicode == '5' || event.text.unicode == '6' || event.text.unicode == '7' || event.text.unicode == '8' || event.text.unicode == '9' || event.text.unicode == '0') {

inputString3 += static\_cast<char>(event.text.unicode);

}

}

inputText3.setString(inputString3);

if (inputString3 == "" or inputString3 == "-") { int ara\_ara = rand() % 61; inputString3 = bcan[ara\_ara]; }

}

}

if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Enter) {

isInputting13 = false;

}

}

window.clear();

window.draw(inputText3);

window.draw(outText4);

window.display();

}

int a = stoi(inputString3);

tr.push(a);

}

sf::Text buttonText;

buttonText.setFont(font);

buttonText.setString("Balance Tree");

buttonText.setCharacterSize(24);

buttonText.setFillColor(sf::Color::White);

buttonText.setPosition(20, 20);

sf::Text button\_td\_Text;

button\_td\_Text.setFont(font);

button\_td\_Text.setString("delete mode");

button\_td\_Text.setCharacterSize(24);

button\_td\_Text.setFillColor(sf::Color::White);

button\_td\_Text.setPosition(20, 90);

sf::Text button\_addd\_Text;

button\_addd\_Text.setFont(font);

button\_addd\_Text.setString("add mode");

button\_addd\_Text.setCharacterSize(24);

button\_addd\_Text.setFillColor(sf::Color::White);

button\_addd\_Text.setPosition(20, 160);

sf::Text button\_sum\_Text;

button\_sum\_Text.setFont(font);

button\_sum\_Text.setString("get average value");

button\_sum\_Text.setCharacterSize(24);

button\_sum\_Text.setFillColor(sf::Color::White);

button\_sum\_Text.setPosition(20, 230);

sf::Text button\_search\_Text;

button\_search\_Text.setFont(font);

button\_search\_Text.setString("search");

button\_search\_Text.setCharacterSize(24);

button\_search\_Text.setFillColor(sf::Color::White);

button\_search\_Text.setPosition(20, 300);

bool balanced = false;

bool deleted = false;

bool isMouseOverAddButton = false;

bool isMouseOverSrArButton = false;

bool isMouseOverButton = false;

bool isMouseOverDeleteButton = false;

bool isMouseOverSearchButton = false;

sf::FloatRect buttonBounds;

sf::FloatRect deleteButtonBounds;

sf::FloatRect addButtonBounds;

sf::FloatRect midButtonBounds;

sf::FloatRect searchButtonBounds;

float aswl = 0;

while (window.isOpen())

{

sf::Event event;

while (window.pollEvent(event))

{

if (event.type == sf::Event::Closed)

window.close();

if (event.type == sf::Event::MouseButtonPressed)

{

sf::Vector2i mousePos = sf::Mouse::getPosition(window);

if (buttonBounds.contains(static\_cast<sf::Vector2f>(mousePos)))

{

tr.balanceTree();

balanced = true;

deleted = false;

}

if (deleteButtonBounds.contains(static\_cast<sf::Vector2f>(mousePos)))

{

sf::String unicodeString = L"Введите путь к узлу который будет удален, путь должен состоять из симловл 'r' и 'l'\nr - переход к правом улементу\nl - переход к левому элементу";

sf::Text text(unicodeString, font, 30);

text.setFont(font);

text.setFillColor(sf::Color::White);

text.setPosition(200, 100);

window.clear();

window.draw(text);

window.display();

tr.del\_node(window, font);

deleted = true;

balanced = false;

}

if (addButtonBounds.contains(static\_cast<sf::Vector2f>(mousePos)))

{

tr.add\_node(window, font);

balanced = false;

deleted = false;

}

if (midButtonBounds.contains(static\_cast<sf::Vector2f>(mousePos)))

{

float mid = tr.getSumOfAllElements() / tr.getCountOfAllElements();

sf::String unicodeString = L"Среднее арифмитическое дерева = " + std::to\_wstring(mid);

sf::Text text(unicodeString, font, 30);

text.setFont(font);

text.setFillColor(sf::Color::White);

text.setPosition(200, 100);

window.clear();

window.draw(text);

window.display();

this\_thread::sleep\_for(std::chrono::seconds(3));

}

if (searchButtonBounds.contains(static\_cast<sf::Vector2f>(mousePos)))

{

sf::Text inputText("", font, 30);

inputText.setFillColor(sf::Color::White);

inputText.setPosition(865, 150);

sf::String unicodeString = L"Введите значение которое надо найти";

sf::Text outText(unicodeString, font, 30);

outText.setFillColor(sf::Color::White);

outText.setPosition(300, 150);

std::string inputString;

bool isInputting = true;

while (isInputting)

{

sf::Event event;

while (window.pollEvent(event)) {

if (event.type == sf::Event::TextEntered) {

if (event.text.unicode < 128) {

if (event.text.unicode == '\b' && inputString.size() > 0) {

inputString.pop\_back();

}

else if (inputString.size() == 0) {

if (event.text.unicode == '1' || event.text.unicode == '2' || event.text.unicode == '3' || event.text.unicode == '4' || event.text.unicode == '5' || event.text.unicode == '6' || event.text.unicode == '7' || event.text.unicode == '8' || event.text.unicode == '9' || event.text.unicode == '0' || event.text.unicode == '-') {

inputString += static\_cast<char>(event.text.unicode);

}

}

else if (inputString.size() > 0 && inputString.size() <= 8) {

if (event.text.unicode == '1' || event.text.unicode == '2' || event.text.unicode == '3' || event.text.unicode == '4' || event.text.unicode == '5' || event.text.unicode == '6' || event.text.unicode == '7' || event.text.unicode == '8' || event.text.unicode == '9' || event.text.unicode == '0') {

inputString += static\_cast<char>(event.text.unicode);

}

}

inputText.setString(inputString);

}

}

if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Enter) {

isInputting = false;

if (inputString == "") { inputString = "err"; }

}

}

window.clear();

window.draw(inputText);

window.draw(outText);

window.display();

}

if (inputString != "err")

{

int key = stoi(inputString);

string strv = "";

reverserss(strv);

Tree\_el\* node\_searched = tr.search(tr.head, key, strv);

if (node\_searched != nullptr && node\_searched != tr.head)

{

sf::String sfStrv = sf::String::fromUtf8(strv.begin(), strv.end());

sf::String sunicodeString = L"Элемент найден на пути " + sfStrv;

sf::Text textt(sunicodeString, font, 30);

textt.setFont(font);

textt.setFillColor(sf::Color::White);

textt.setPosition(200, 100);

window.clear();

window.draw(textt);

window.display();

this\_thread::sleep\_for(std::chrono::seconds(3));

}

else if (node\_searched == tr.head)

{

sf::String sunicodeString = L"Элемент является корневым";

sf::Text textt(sunicodeString, font, 30);

textt.setFont(font);

textt.setFillColor(sf::Color::White);

textt.setPosition(200, 100);

window.clear();

window.draw(textt);

window.display();

this\_thread::sleep\_for(std::chrono::seconds(3));

}

else

{

sf::String sunicodeString = L"Элемент не найден";

sf::Text textt(sunicodeString, font, 30);

textt.setFont(font);

textt.setFillColor(sf::Color::White);

textt.setPosition(200, 100);

window.clear();

window.draw(textt);

window.display();

this\_thread::sleep\_for(std::chrono::seconds(3));

}

}

else

{

sf::String sunicodeString = L"Не введено значения для поиска";

sf::Text textt(sunicodeString, font, 30);

textt.setFont(font);

textt.setFillColor(sf::Color::White);

textt.setPosition(200, 100);

window.clear();

window.draw(textt);

window.display();

this\_thread::sleep\_for(std::chrono::seconds(2));

}

}

}

if (event.type == sf::Event::MouseMoved)

{

sf::Vector2i mousePos = sf::Mouse::getPosition(window);

buttonBounds = buttonText.getGlobalBounds();

buttonBounds.left -= 20;

buttonBounds.top -= 20;

buttonBounds.width += 40;

buttonBounds.height += 40;

if (buttonBounds.contains(static\_cast<sf::Vector2f>(mousePos)))

{

isMouseOverButton = true;

}

else

{

isMouseOverButton = false;

}

deleteButtonBounds = button\_td\_Text.getGlobalBounds();

deleteButtonBounds.left -= 18;

deleteButtonBounds.top -= 20;

deleteButtonBounds.width += 38;

deleteButtonBounds.height += 40;

if (deleteButtonBounds.contains(static\_cast<sf::Vector2f>(mousePos)))

{

isMouseOverDeleteButton = true;

}

else

{

isMouseOverDeleteButton = false;

}

addButtonBounds = button\_addd\_Text.getGlobalBounds();

addButtonBounds.left -= 18;

addButtonBounds.top -= 20;

addButtonBounds.width += 38;

addButtonBounds.height += 40;

if (addButtonBounds.contains(static\_cast<sf::Vector2f>(mousePos)))

{

isMouseOverAddButton = true;

}

else

{

isMouseOverAddButton = false;

}

midButtonBounds = button\_sum\_Text.getGlobalBounds();

midButtonBounds.left -= 18;

midButtonBounds.top -= 20;

midButtonBounds.width += 38;

midButtonBounds.height += 40;

if (midButtonBounds.contains(static\_cast<sf::Vector2f>(mousePos)))

{

isMouseOverSrArButton = true;

}

else

{

isMouseOverSrArButton = false;

}

searchButtonBounds = button\_search\_Text.getGlobalBounds();

searchButtonBounds.left -= 15;

searchButtonBounds.top -= 20;

searchButtonBounds.width += 30;

searchButtonBounds.height += 40;

if (searchButtonBounds.contains(static\_cast<sf::Vector2f>(mousePos)))

{

isMouseOverSearchButton = true;

}

else

{

isMouseOverSearchButton = false;

}

}

}

window.clear(sf::Color::Black);

if (balanced)

tr.drawTree(window, tr.head, 1100, 50);

if (deleted)

tr.drawTree(window, tr.head, 1100, 50);

else

tr.drawTree(window, tr.head, 1100, 50);

if (isMouseOverSrArButton)

{

sf::RectangleShape midButtonOutline(sf::Vector2f(midButtonBounds.width, midButtonBounds.height));

midButtonOutline.setPosition(midButtonBounds.left, midButtonBounds.top);

midButtonOutline.setFillColor(sf::Color::Transparent);

midButtonOutline.setOutlineColor(sf::Color::Red);

midButtonOutline.setOutlineThickness(2.f);

window.draw(midButtonOutline);

}

if (isMouseOverDeleteButton)

{

sf::RectangleShape deleteButtonOutline(sf::Vector2f(deleteButtonBounds.width, deleteButtonBounds.height));

deleteButtonOutline.setPosition(deleteButtonBounds.left, deleteButtonBounds.top);

deleteButtonOutline.setFillColor(sf::Color::Transparent);

deleteButtonOutline.setOutlineColor(sf::Color::Red);

deleteButtonOutline.setOutlineThickness(2.f);

window.draw(deleteButtonOutline);

}

if (isMouseOverAddButton)

{

sf::RectangleShape deleteButtonOutline(sf::Vector2f(addButtonBounds.width, addButtonBounds.height));

deleteButtonOutline.setPosition(addButtonBounds.left, addButtonBounds.top);

deleteButtonOutline.setFillColor(sf::Color::Transparent);

deleteButtonOutline.setOutlineColor(sf::Color::Red);

deleteButtonOutline.setOutlineThickness(2.f);

window.draw(deleteButtonOutline);

}

if (isMouseOverButton)

{

sf::RectangleShape deleteButtonOutline(sf::Vector2f(buttonBounds.width, buttonBounds.height));

deleteButtonOutline.setPosition(buttonBounds.left, buttonBounds.top);

deleteButtonOutline.setFillColor(sf::Color::Transparent);

deleteButtonOutline.setOutlineColor(sf::Color::Red);

deleteButtonOutline.setOutlineThickness(2.f);

window.draw(deleteButtonOutline);

}

if (isMouseOverSearchButton)

{

sf::RectangleShape searchButtonOutline(sf::Vector2f(searchButtonBounds.width, searchButtonBounds.height));

searchButtonOutline.setPosition(searchButtonBounds.left, searchButtonBounds.top);

searchButtonOutline.setFillColor(sf::Color::Transparent);

searchButtonOutline.setOutlineColor(sf::Color::Red);

searchButtonOutline.setOutlineThickness(2.f);

window.draw(searchButtonOutline);

}

window.draw(buttonText);

window.draw(button\_td\_Text);

window.draw(button\_addd\_Text);

window.draw(button\_sum\_Text);

window.draw(button\_search\_Text);

window.display();

}

system("pause");

return 0;

}

**Демонстрация работы программы:**

https://www.youtube.com/watch?v=RmjYu3dRx7E