Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

# Лабораторная работ: Деревья

Выполнил работу

Студент группы РИС 22-2б

Вахрушева А.В.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи**

Реализовать бинарное дерево поиска и функции для работы с ним. Выполнить визуализацию при помощи библиотеки SFML.

**Диаграмма класса**



Рисунок 1 – диаграмма класса Tree

Код программы

#include <iostream>

#include <ctime>

#include <vector>

#include <SFML/Graphics.hpp>

#include <cmath>

#define PI 3.14159

int n;

using namespace std;

using namespace sf;

class tree

{

public:

tree();

tree(int data);

tree(int data, tree\* parent);

~tree();

void Add(int data, tree\*& branch);

void print\_tree(int level);

void pr\_obh(tree\* branch);

void sim\_obh(tree\* branch);

void obr\_obh(tree\* branch);

bool is\_empty(tree\*& branch);

void free(tree\* branch);

tree\* del\_elem(tree\*& branch, int data);

int getHeight(tree\* branch);

void updateHeight(tree\* branch);

tree\* find(int data);

tree\* find\_max();

tree\* find\_min();

void add\_left(tree\* temp) { left = temp; }

void add\_right(tree\* temp) { right = temp; }

int getData() { return data; }

int getX() { return x; }

int getY() { return y; }

void setX(int x) { this->x = x; }

void setY(int y) { this->y = y; }

void printTree(tree\* branch, RenderWindow &window, CircleShape &shape);

tree\* rotateRight(tree\* y);

tree\*rotateLeft(tree\* x);

tree\* insertBT(tree\* branch, int data, tree\* parent);

void updateLevel(tree\* branch, int level);

protected:

int data;

tree\* left;

tree\* right;

tree\* parent;

unsigned int level;

unsigned char height;

int x;

int y;

};

tree::tree()

{

data = 0;

left = nullptr;

right = nullptr;

parent = nullptr;

height = 0;

level = 1;

x = y = 0;

}

tree::tree(int data)

{

this->data = data;

left = right = parent = nullptr;

x = y = 0;

height = 1;

level = 1;

}

tree::tree(int data, tree\* parent)

{

this->data = data;

left = right = nullptr;

this->parent = parent;

x = y = 0;

height = 1;

level = 1;

}

tree::~tree()

{

}

void tree::Add(int data, tree\*& branch)

{

if (!branch)

{

branch = new tree;

branch->data = data;

branch->left = 0;

branch->right = 0;

branch->height = 1;

return;

}

else

{

if (branch->data > data)

{

Add(data, branch->left);

branch->left->parent = branch;

updateLevel(branch->left, level + 1);

}

else

{

Add(data, branch->right);

branch->right->parent = branch;

updateLevel(branch->right, level + 1);

}

}

updateHeight(branch);

}

void tree::print\_tree(int level = 1)

{

if (this != nullptr)

{

this->right->print\_tree(level + 1);

for (int i = 1; i < level; i++)

cout << " ";

cout << this->data << endl;

this->left->print\_tree(level + 1);

}

}

int tree::getHeight(tree\* branch)

{

if (branch == nullptr) {

return 0;

}

return branch->height;

}

void tree::updateLevel(tree\* branch, int level = 1)

{

if (branch == NULL)

return;

branch->level = level;

updateLevel(branch->left, level + 1);

updateLevel(branch->right, level + 1);

}

void tree::updateHeight(tree\* branch)

{

int leftHeight = getHeight(branch->left);

int rightHeight = getHeight(branch->right);

branch->height = 1 + max(leftHeight, rightHeight);

}

tree\* tree::rotateRight(tree\* y)

{

tree\* x = y->left;

tree\* T2 = x->right;

x->right = y;

y->left = T2;

x->parent = y->parent;

y->parent = x;

if (T2)

T2->parent = y;

y->height = max(getHeight(y->left), getHeight(y->right)) + 1;

x->height = max(getHeight(x->left), getHeight(x->right)) + 1;

return x;

}

tree\* tree::rotateLeft(tree\* x)

{

tree\* y = x->right;

tree\* T2 = y->left;

y->left = x;

x->right = T2;

y->parent = x->parent;

x->parent = y;

if (T2)

T2->parent = x;

x->height = max(getHeight(x->left), getHeight(x->right)) + 1;

y->height = max(getHeight(y->left), getHeight(y->right)) + 1;

return y;

}

tree\* tree::insertBT(tree\* branch, int data, tree\* parent)

{

if (branch == nullptr)

return new tree(data, parent);

if (data < branch->data)

branch->left = insertBT(branch->left, data, branch);

else if (data > branch->data)

branch->right = insertBT(branch->right, data, branch);

else

return branch;

branch->height = max(getHeight(branch->left), getHeight(branch->right)) + 1;

branch->updateLevel(branch);

branch->parent = parent;

int balance = getHeight(branch->left) - getHeight(branch->right);

if (balance > 1 && data < branch->left->data)

return rotateRight(branch);

if (balance < -1 && data > branch->right->data)

return rotateLeft(branch);

if (balance > 1 && data > branch->left->data)

{

branch->left = rotateLeft(branch->left);

return rotateRight(branch);

}

if (balance < -1 && data < branch->right->data)

{

branch->right = rotateRight(branch->right);

return rotateLeft(branch);

}

return branch;

}

void tree::pr\_obh(tree\* branch)

{

if (branch == NULL)

return;

cout << branch->data << endl;

pr\_obh(branch->left);

pr\_obh(branch->right);

}

void tree::sim\_obh(tree\* branch)

{

if (branch == NULL)

return;

sim\_obh(branch->left);

cout << branch->data << endl;

sim\_obh(branch->right);

}

void tree::obr\_obh(tree\* branch)

{

if (branch == NULL)

return;

obr\_obh(branch->left);

obr\_obh(branch->right);

cout << branch->data << endl;

}

bool tree::is\_empty(tree\*& branch)

{

if (!branch)

return 0;

else

return 1;

}

void tree::free(tree\* branch)

{

if (!branch)

return;

free(branch->left);

free(branch->right);

delete branch;

branch = nullptr;

}

tree\* tree::find(int data)

{

if (this == nullptr || this->data == data)

return this;

else if (data > this->data)

return this->right->find(data);

else

return this->left->find(data);

}

tree\* tree::find\_max()

{

if (this->right == nullptr)

return this;

return this->right->find\_max();

}

tree\* tree::find\_min()

{

if (this->left == nullptr)

return this;

return this->left->find\_min();

}

void tree::printTree(tree\* branch, RenderWindow &window, CircleShape &shape)

{

if (branch == NULL)

return;

Font fnt;

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

RectangleShape line;

line.setFillColor(Color(190, 107, 39));

Text txt(to\_string(branch->data), fnt, 35);

txt.setFillColor(Color(0, 0, 0));

txt.setOrigin(txt.getLocalBounds().width / 2.f, txt.getLocalBounds().height / 2.f);

shape.setFillColor(Color(0, 255, 64));

shape.setOutlineThickness(3);

shape.setOutlineColor(Color(255, 0, 128));

if (branch->parent == nullptr)

{

branch->x = window.getView().getCenter().x;

branch->y = 10;

}

else if (branch->parent->data > branch->data)

{

branch->x = branch->parent->x - sqrt(n) \* 175 / (pow(2, branch->parent->level));

branch->y = branch->parent->y + 130;

line.setPosition(branch->parent->x, branch->parent->y + shape.getLocalBounds().height - 3);

line.setSize(Vector2f(2, sqrt(pow(branch->parent->x - branch->x, 2) + pow(branch->parent->y + shape.getLocalBounds().height - branch->y, 2)) - 3));

line.rotate(90 - atan((130.f - shape.getLocalBounds().height) / abs((float)(branch->parent->x - branch->x))) \* 180 / PI);

}

else if (branch->parent->data < branch->data)

{

branch->x = branch->parent->x + sqrt(n) \* 175 / (pow(2, branch->parent->level));

branch->y = branch->parent->y + 130;

line.setPosition(branch->parent->x, branch->parent->y + shape.getLocalBounds().height - 2);

line.setSize(Vector2f(2, sqrt(pow(branch->parent->x - branch->x, 2) + pow(branch->parent->y + shape.getLocalBounds().height - branch->y, 2)) - 1));

line.rotate(-(90 - atan((130.f - shape.getLocalBounds().height) / abs((float)(branch->parent->x - branch->x))) \* 180 / PI));

}

shape.setPosition(branch->x, branch->y);

txt.setPosition(shape.getPosition().x - 2, shape.getPosition().y + (shape.getLocalBounds().height - txt.getLocalBounds().height) / 2.f + 2);

window.draw(shape);

window.draw(txt);

window.draw(line);

printTree(branch->left, window, shape);

printTree(branch->right, window, shape);

}

tree\* tree::del\_elem(tree\*& branch, int data)

{

if (branch == nullptr)

return branch;

if (data == branch->data)

{

tree\* tmp = nullptr;

tree\* tmpPar = nullptr;

if (branch->left == nullptr && branch->right != nullptr)

{

tmp = branch->right;

tmpPar = branch->parent;

delete branch;

branch = tmp;

branch->parent = tmpPar;

}

else if (branch->left != nullptr && branch->right == nullptr)

{

tmp = branch->left;

tmpPar = branch->parent;

delete branch;

branch = tmp;

branch->parent = tmpPar;

}

else if (branch->left != nullptr && branch->right != nullptr)

{

tmp = branch->right;

while (tmp->left != nullptr)

tmp = tmp->left;

if (tmp->right == nullptr)

{

branch->data = tmp->data;

if (tmp->parent->left->data >= branch->data)

tmp->parent->left = nullptr;

else

tmp->parent->right = nullptr;

}

else

{

branch->data = tmp->data;

if (tmp->parent->left->data >= branch->data)

tmp->parent->left = tmp->right;

else

tmp->parent->right = tmp->right;

tmp->right->parent = tmp->parent;

}

delete tmp;

tmp = branch;

}

else

{

delete branch;

branch = nullptr;

}

return tmp;

}

else if (data < branch->data)

branch->left = del\_elem(branch->left, data);

else

branch->right = del\_elem(branch->right, data);

return branch;

}

int main()

{

system("chcp 1251 >> null");

int x;

cout << "Введите количество узлов в сбалансированном дереве: "; cin >> n;

tree\* tr = nullptr;

for (size\_t i = 0; i < n; ++i)

{

int el;

cout << "Введите уникальный элемент узла: "; cin >> el;

if (tr->find(el) == nullptr)

tr = tr->insertBT(tr, el, nullptr);

else

{

cout << "\nОшибка! Число должно быть уникальным!\n\n";

--i;

}

}

tr->updateLevel(tr);

RenderWindow window(VideoMode(1366, 768), "TestPrintTree");

CircleShape shape(40);

shape.setOrigin(shape.getRadius(), 0);

while (window.isOpen())

{

Event event;

while(window.pollEvent(event))

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

window.close();

window.clear(Color(9, 196, 242));

tr->printTree(tr, window, shape);

window.display();

}

return 0;

}

Результат работы программы.

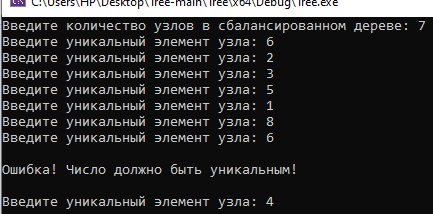


Рисунок 2 – ввод элементов дерева

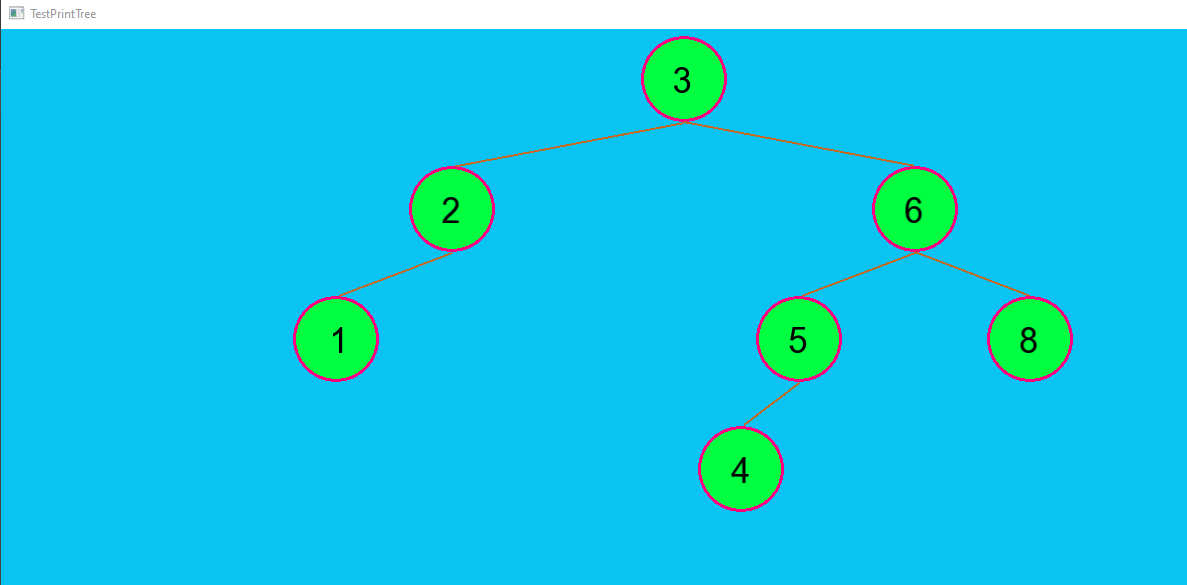


Рисунок 3 – вывод сбалансированного дерева