Министерство образования и науки Российской Федерации

Федеральное государственное образовательное автономное учреждение высшего образования

"Пермский национальный исследовательский политехнический университет"

Лабораторная работа.

«Бинарное дерево».

Выполнил: студент группы РИС-23-2б

Вековшинин Иван Николаевич

Проверила: доцент кафедры ИТАС

О.А. Полякова.

2024 г.

Постановка задачи:

1. Деревья

1.1 Требуется реализовать бинарное дерево поиска.

1.2 Реализовать допустимые операции для дерева:

- Вставка узла.

- Удаление узла.

- Поиск элемента по ключу.

1.3 Реализовать алгоритмы обхода дерева:

1) Прямой

2) Симметричный

3) Обратный

1.4 Реализовать алгоритм балансировки дерева.

1.5 Реализовать вертикальную и горизонтальную печать.

1.6 Визуализация дерева с использованием любой доступной графической библиотеки – SFML(Предпочтительно), SDL, OpenGL…

1.7 Пользовательский интерфейс – на усмотрение разработчика.

1.8 Выполнить отчет:

- постановка задачи;

-анализ задачи с разбором применения используемых структур данных, функций;

- код программы на C++ с подробными комментариями;

- скриншоты работы программы;

- визуализация решения;

1.9 Диаграмма классов.

Для решения задачи необходимо реализовать класс дерево. Далее нужно продумать логику и прописать реализацию методов.

Код программы:

Tree.h

#pragma once

#include <iostream>

#include <queue>

#include <stack>

#include <fstream>

#include <vector>

using namespace std;

template <typename T>

class Tree

{

private:

class Node //вложенный класс - ячейка памяти

{

public:

Node(T data = T(), Node\* pLeft = nullptr, Node\* pRight = nullptr) //используется параметр по умолчанию, то есть если мы не передаем указатель, то он автоматически присваивается nullptr

{ //для поля data также используется параметр по умолчанию, выглядит он как вызов конструктора по умолчанию

this->data = data;

this->pLeft = pLeft;

this->pRight = pRight;

}

Node\* pLeft; //хранит указатель на следующий элемент слева

Node\* pRight; //хранит указатель на следующий элемент справа

T data; //хранит наши данные

};

int \_size; //количество элементов в списке.

Node\* head; //указатель на первый элемент в списке

public:

Tree(); //Конструктор

~Tree(); //Деструктор

void insert(T data); //добавляет элемент в конец списка

int size() { return this->\_size; } //так как инкапсуляция не дает нам доступ к количеству элементов в списке, то нам необходимо происать геттер

bool erase(const T& data); //Удаление элемента

void clear(); //удаляет все элементы списка (логика как у pop\_front, деструктор основан на этой штуке)

Tree<T>\* find(T data);

void showWidth(); //обход в ширину

void showStraight(); //прямой обход

void showDepth(); //обход в глубину

void showDepthBackward(); //обход в глубину в другую сторону (элементы идут по возрастанию)

Tree<T>& operator=(const Tree<T>& other);

Tree<T>\* get\_data();

Tree<T>\* get\_left(); // Возвращает указатель на левое поддерево.

Tree<T>\* get\_right(); // Возвращает указатель на правое поддерево.

// методы обхода дерева

void direct\_way(Tree<T>\*); // Обход дерева в прямом порядке.

void symmetric\_way(Tree<T>\*); // Обход дерева в симметричном порядке.

void reverse\_way(Tree<T>\*); // Обход дерева в обратном порядке.

// функции для балансировки дерева

int getHeight(Node\* node);

int getBalanceFactor(Node\* node);

typename Node\* rotateLeft(Node\* node);

typename Node\* rotateRight(Node\* node);

typename Node\* balance(Node\* node);

private:

void HelperDepth(Node\* node);

void HelperDepthBackward(Node\* node);

void HelperClear(Node\* node);

int helpDepth;

};

template<typename T>

Tree<T>::Tree()

{

this->\_size = 0;

head = nullptr;

helpDepth = 0;

}

template<typename T>

Tree<T>::~Tree()

{

this->clear();

}

template<typename T>

void Tree<T>::insert(T data)

{

if (head == nullptr)

head = new Node(data);

else

{

Node\* temp = this->head;

while (true)

{

if (data < temp->data)

{

if (temp->pLeft == nullptr)

{

temp->pLeft = new Node(data);

break;

}

else

temp = temp->pLeft;

}

else if (data > temp->data)

if (temp->pRight == nullptr)

{

temp->pRight = new Node(data);

break;

}

else

temp = temp->pRight;

else return;

}

}

++\_size;

head = balance(head); //вызов балансировки

}

template<typename T>

bool Tree<T>::erase(const T& data)

{

Node\* prev = head;

Node\* iter = head;

while (true)

{

if (data == iter->data)

{

break;

}

else if (data < iter->data)

{

if (iter->pLeft == nullptr) return false;

prev = iter;

iter = iter->pLeft;

}

else

{

if (iter->pRight == nullptr) return false;

prev = iter;

iter = iter->pRight;

}

}

if (iter->pLeft == nullptr && iter->pRight == nullptr)

{

if (iter == head)

{

delete iter;

head == nullptr;

}

else

{

if (prev->pLeft == iter)

prev->pLeft = nullptr;

else

prev->pRight = nullptr;

delete iter;

}

}

else if (iter->pLeft == nullptr || iter->pRight == nullptr)

{

if (iter == head)

{

if (iter->pLeft != nullptr)

head = head->pLeft;

else

head = head->pRight;

}

else

{

if (iter->pLeft != nullptr)

{

if (prev->pLeft == iter)

prev->pLeft = iter->pLeft;

else

prev->pRight = iter->pLeft;

}

else

{

if (prev->pLeft == iter)

prev->pLeft = iter->pRight;

else

prev->pRight = iter->pRight;

}

}

delete iter;

}

else

{

Node\* temp = iter;

prev = iter;

iter = iter->pRight;

while (iter->pLeft != nullptr)

{

temp = iter;

iter = iter->pLeft;

}

prev->data = iter->data;

delete iter;

temp->pLeft = nullptr;

}

--this->\_size;

return true;

}

template<typename T>

void Tree<T>::showWidth()

{

if (this->head == nullptr) return;

int j = 0;

Node\* temp;

queue<Node\*> que;

que.push(this->head);

while (!que.empty())

{

temp = que.front();

cout << ++j << ". " << temp->data << endl;

que.pop();

if (temp->pLeft != nullptr)

que.push(temp->pLeft);

if (temp->pRight != nullptr)

que.push(temp->pRight);

}

}

template<typename T>

void Tree<T>::showStraight()

{

if (this->head == nullptr) return;

int j = 0;

Node\* temp;

stack<Node\*> st;

st.push(this->head);

while (!st.empty())

{

temp = st.top();

cout << ++j << ". " << temp->data << endl;

st.pop();

if (temp->pRight != nullptr)

st.push(temp->pRight);

if (temp->pLeft != nullptr)

st.push(temp->pLeft);

}

}

template<typename T>

inline void Tree<T>::HelperDepth(Node\* node)

{

if (node == nullptr) return;

HelperDepth(node->pLeft);

cout << ++helpDepth << ". " << node->data << endl;

HelperDepth(node->pRight);

}

template<typename T>

void Tree<T>::showDepth()

{

if (this->head != nullptr)

{

this->helpDepth = 0;

this->HelperDepth(this->head);

}

}

template<typename T>

void Tree<T>::HelperDepthBackward(Node\* node)

{

if (node == nullptr) return;

HelperDepthBackward(node->pRight);

cout << ++helpDepth << ". " << node->data << endl;

HelperDepthBackward(node->pLeft);

}

template<typename T>

void Tree<T>::showDepthBackward()

{

if (this->head != nullptr)

{

this->helpDepth = 0;

this->HelperDepthBackward(this->head);

}

}

template<typename T>

inline void Tree<T>::HelperClear(Node\* node)

{

if (node == nullptr) return;

HelperClear(node->pLeft);

HelperClear(node->pRight);

node->pLeft == nullptr;

node->pRight = nullptr;

delete node;

}

template<typename T>

void Tree<T>::clear()

{

if (head != nullptr)

{

this->HelperClear(this->head);

this->\_size = 0;

this->head = nullptr;

}

}

template<typename T>

Tree<T>& Tree<T>::operator=(const Tree<T>& other)

{

this->clear();

if (other.head == nullptr) return \*this;

Node\* temp;

stack<Node\*> st;

st.push(other.head);

while (!st.empty())

{

temp = st.top();

this->insert(temp->data);

st.pop();

if (temp->pRight != nullptr)

st.push(temp->pRight);

if (temp->pLeft != nullptr)

st.push(temp->pLeft);

}

return \*this;

}

template<typename T>

inline Tree<T>\* Tree<T>::find(T data)

{

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

{

return this;

}

else if (data > this->data)

{

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

}

else

{

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

}

}

template<typename T>

int Tree<T>::getHeight(Node\* node)

{

if (node == nullptr)

return 0;

return max(getHeight(node->pLeft), getHeight(node->pRight)) + 1;

}

template<typename T>

int Tree<T>::getBalanceFactor(Node\* node)

{

if (node == nullptr)

return 0;

return getHeight(node->pLeft) - getHeight(node->pRight);

}

template<typename T>

typename Tree<T>::Node\* Tree<T>::rotateLeft(Node\* node)

{

Node\* newRoot = node->pRight;

node->pRight = newRoot->pLeft;

newRoot->pLeft = node;

return newRoot;

}

template<typename T>

typename Tree<T>::Node\* Tree<T>::rotateRight(Node\* node)

{

Node\* newRoot = node->pLeft;

node->pLeft = newRoot->pRight;

newRoot->pRight = node;

return newRoot;

}

template<typename T>

typename Tree<T>::Node\* Tree<T>::balance(Node\* node)

{

if (node == nullptr)

return nullptr;

int balanceFactor = getBalanceFactor(node);

if (balanceFactor > 1) //если левое поддерево выше

{

if (getBalanceFactor(node->pLeft) < 0) //условтие балансировки

node->pLeft = rotateLeft(node->pLeft);

return rotateRight(node);

}

else if (balanceFactor < -1) //правое поддерево выше

{

if (getBalanceFactor(node->pRight) > 0)

node->pRight = rotateRight(node->pRight);

return rotateLeft(node);

}

return node;

}

// функции для получения элементов из дерева

template <class T>

Tree<T>\* Tree<T>::get\_data() {

return data;

}

template <class T>

Tree<T>\* Tree<T>::get\_left() {

return left;

}

template <class T>

Tree<T>\* Tree<T>::get\_right() {

return right;

}

// функции обхода дерева

template <class T>

void Tree<T>::direct\_way(Tree<T>\* current) {

if (current == nullptr) {

return;

}

else {

cout << current->get\_data() << " ";

direct\_way(current->get\_left());

direct\_way(current->get\_right());

}

}

template <class T>

void Tree<T>::symmetric\_way(Tree<T>\* tree) {

if (tree != nullptr) {

symmetric\_way(tree->left);

cout << tree->data << " ";

symmetric\_way(tree->right);

}

}

template <class T>

void Tree<T>::reverse\_way(Tree<T>\* tree) {

if (tree != nullptr) {

reverse\_way(tree->left);

reverse\_way(tree->right);

cout << tree->data << " ";

}

}

Tree.cpp

#include <iostream>

#include <queue>

#include <stack>

#include <fstream>

#include <vector>

using namespace std;

#include "tree.h"

#include <SFML/Graphics.hpp>

int main()

{

system("chcp 1251 >> null");

//system("color F0");

cout << "Создание дерева и вывод консоль." << endl;

cout << "Обход в ширину:" << endl;

Tree<double> tr, tr1;

tr.insert(14.5);

tr.insert(12.1);

tr.insert(10.8);

tr.insert(7.4);

tr.insert(3.4);

tr.insert(2.9);

tr.insert(1.1);

tr.insert(0.2);

tr.showWidth();

cout << endl << endl;

cout << "Удаление элемента и попытка добавления элемента со значением, которое существует в дереве:" << endl;

if (tr.erase(2.9)) cout << "Уделение успешно." << endl;

else cout << "Удаление невозможно." << endl;

tr.insert(7.4);

cout << "Прямой обход." << endl;

tr.showStraight();

cout << endl << endl;

cout << "Выполнение оператора присваивания" << endl;

tr1 = tr;

cout << "Вывод скопированного дерева." << endl;

cout << "Обход в глубину." << endl;

tr1.showDepth();

cout << endl << endl;

cout << "Отрисовка полученного дерева." << endl;

cout << "Внимание на окно." << endl;

sf::RenderWindow window(sf::VideoMode(1000, 800), "Binary Tree");

window.setFramerateLimit(10);

sf::CircleShape shape(40.f);

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

shape.setOutlineThickness(3.0f);

shape.setOutlineColor(sf::Color::Black);

shape.setOrigin(shape.getLocalBounds().width / 2, shape.getLocalBounds().height / 2);

shape.setPosition(500.f, 48.f);

sf::Font font;

if (!font.loadFromFile("arial.ttf"))

{

cout << "Ошибка" << endl;

}

sf::Text text;

text.setFont(font);

text.setString("7.4");

text.setCharacterSize(18);

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

text.setStyle(sf::Text::Bold);

text.setOrigin(text.getLocalBounds().width / 2, text.getLocalBounds().height / 2);

text.setPosition(shape.getPosition().x - 3, shape.getPosition().y - 3);

vector<pair<sf::CircleShape, sf::Text>> vec;

vec.resize(8);

vec[0] = make\_pair(shape, text);

shape.setPosition(300.f, 118.f);

text.setString("2.9");

text.setPosition(shape.getPosition().x - 3, shape.getPosition().y - 3);

vec[1] = make\_pair(shape, text);

shape.setPosition(700.f, 118.f);

text.setString("12.1");

text.setPosition(shape.getPosition().x - 3, shape.getPosition().y - 3);

vec[2] = make\_pair(shape, text);

shape.setPosition(200.f, 188.f);

text.setString("1.1");

text.setPosition(shape.getPosition().x - 3, shape.getPosition().y - 3);

vec[3] = make\_pair(shape, text);

shape.setPosition(400.f, 188.f);

text.setString("3.4");

text.setPosition(shape.getPosition().x - 3, shape.getPosition().y - 3);

vec[4] = make\_pair(shape, text);

shape.setPosition(600.f, 188.f);

text.setString("10.8");

text.setPosition(shape.getPosition().x - 3, shape.getPosition().y - 3);

vec[5] = make\_pair(shape, text);

shape.setPosition(800.f, 188.f);

text.setString("14.5");

text.setPosition(shape.getPosition().x - 3, shape.getPosition().y - 3);

vec[6] = make\_pair(shape, text);

shape.setPosition(130.f, 290.f);

text.setString("0.2");

text.setPosition(shape.getPosition().x - 3, shape.getPosition().y - 3);

vec[7] = make\_pair(shape, text);

sf::RectangleShape line(sf::Vector2f(211, 3));

vector<sf::RectangleShape> vec\_lines;

vec\_lines.resize(7);

vec\_lines[0].setSize(sf::Vector2f(211, 3));

vec\_lines[0].setFillColor(sf::Color::Black);

vec\_lines[0].setPosition(vec[0].first.getPosition());

vec\_lines[0].setRotation(160);

vec\_lines[1].setSize(sf::Vector2f(211, 3));

vec\_lines[1].setFillColor(sf::Color::Black);

vec\_lines[1].setPosition(vec[0].first.getPosition());

vec\_lines[1].setRotation(20);

vec\_lines[2].setSize(sf::Vector2f(122, 3));

vec\_lines[2].setFillColor(sf::Color::Black);

vec\_lines[2].setPosition(vec[1].first.getPosition());

vec\_lines[2].setRotation(145);

vec\_lines[3].setSize(sf::Vector2f(122, 3));

vec\_lines[3].setFillColor(sf::Color::Black);

vec\_lines[3].setPosition(vec[1].first.getPosition());

vec\_lines[3].setRotation(35);

vec\_lines[4].setSize(sf::Vector2f(122, 3));

vec\_lines[4].setFillColor(sf::Color::Black);

vec\_lines[4].setPosition(vec[2].first.getPosition());

vec\_lines[4].setRotation(145);

vec\_lines[5].setSize(sf::Vector2f(123, 3));

vec\_lines[5].setFillColor(sf::Color::Black);

vec\_lines[5].setPosition(vec[2].first.getPosition());

vec\_lines[5].setRotation(35);

vec\_lines[6].setSize(sf::Vector2f(122, 3));

vec\_lines[6].setFillColor(sf::Color::Black);

vec\_lines[6].setPosition(vec[3].first.getPosition());

vec\_lines[6].setRotation(125);

while (window.isOpen())

{

sf::Event event;

while (window.pollEvent(event))

{

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

window.close();

}

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

for (size\_t i = 0; i < vec\_lines.size(); i++)

{

window.draw(vec\_lines[i]);

}

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

{

window.draw(vec[i].first);

window.draw(vec[i].second);

}

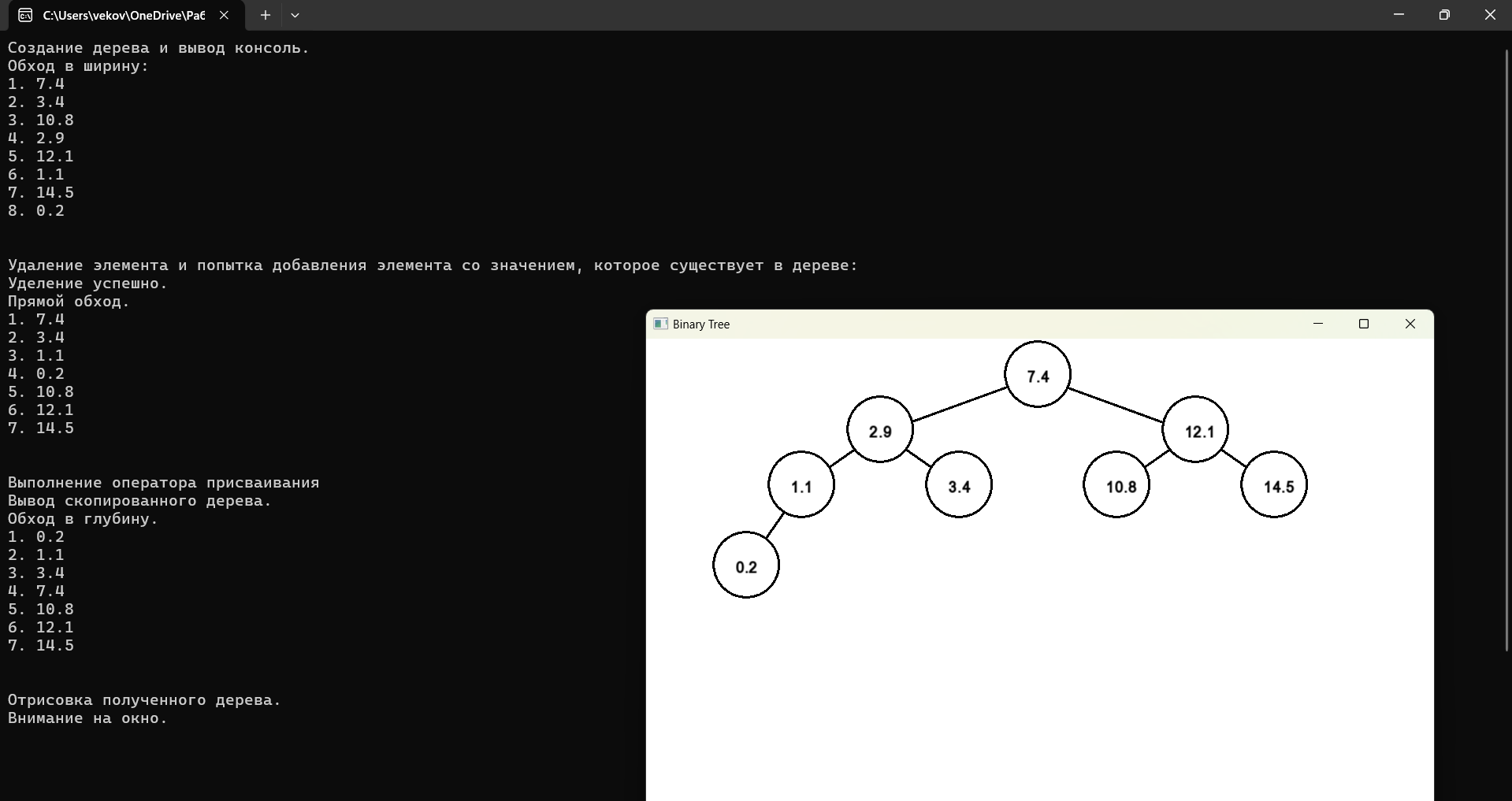
window.display();

}

return 0;

}

Результат работы программы:



Была разработана программа, которая представляет из себя структуру данных бинарное дерево, а также отрисовывает данную структуру. Полученная программа может в дальнейшем модифицироваться или использоваться в более сложных проектах, как отдельный компонент.

UML диаграмма классов:

