**Отчет по лабораторной работе № 23 по курсу**

**«Практикум программирования»**

Студент группы М8О-109Б-22 Концебалов Олег Сергеевич

Контакты: telegram @baronpipistron

Работа выполнена: 27.04.2023

Преподаватель: каф.806 Сысоев Максим Алексеевич

Отчет сдан «25» июня 2023г., итоговая оценка \_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_

**1. Тема:** Динамические структуры данных. Обработка деревьев

**2. Цель работы:** Составить программу на C++ для построения и обработки дерева, а так же для выполнения специального действия

**3. Задание (вариант № 19):** Определить ширину двоичного дерева

**4. Оборудование (студента):**

Процессор AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz, ОП 16,0 Гб, SSD 512 Гб. Монитор 1920x1080 144 Hz

**5. Программное обеспечение (студента):**

Операционная система семейства Linux, наименование Ubuntu, версия 18.10

Интерпретатор команд: bash, версия 4.4.19

Система программирования – версия --, редактор текстов Emacs, версия 25.2.2

Утилиты операционной системы –

Прикладные системы и программы –

Местонахождение и имена файлов программ и данных на домашнем компьютере –

**6. Идея, метод, алгоритм решения задачи** *(в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)*

Строем дерево, используя DFS находим ширину каждого уровня и в конце выбираем максимальную

**7. Сценарий выполнения работы** *(план работы, первоначальный текст программы в черновике [можно на отдельном листе] и тесты, либо соображения по тестированию)*

1. Читаю про деревья
2. Делаю свою реализацию дерева на плюсах
3. Пишу код специального действия

**8. Распечатка протокола** *(подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем)*

***Node.hpp***

#ifndef NODEHPP

#define NODEHPP

#include <iostream>

template <typename T>

class BinaryTree;

template <typename T>

class Node{

    friend class BinaryTree<T>;

private:

    T data;

    Node<T>\* left;

    Node<T>\* right;

public:

    Node();

    Node(const T& value);

    Node(const T& value, Node<T>\* left, Node<T>\* right);

};

#endif

***Node.cpp***

#include "Node.hpp"

template <typename T>

Node<T>::Node(): data(0), left(nullptr), right(nullptr){}

template <typename T>

Node<T>::Node(const T& value): data(value), left(nullptr), right(nullptr){}

template <typename T>

Node<T>::Node(const T& value, Node<T>\* left, Node<T>\* right):

        data(value), left(left), right(right){}

***BinaryTree.hpp***

#ifndef BINARYTREEHPP

#define BINARYTREEHPP

#include "Node.hpp"

#include "MyVector.hpp"

template <typename T>

class BinaryTree{

private:

    Node<T>\* root;

    Node<T>\* findMin(Node<T>\* root);

public:

    BinaryTree();

    BinaryTree(const T& value);

    ~BinaryTree();

    Node<T>\* getRoot() const;

    void deleteTree(Node<T>\* node);

    void insertNode(const T& value, Node<T>\* root);

    Node<T>\* removeNode(const T& value, Node<T>\* root);

    void visualize(Node<T>\* root, const int height = 0) const;

    void inOrderBypass(Node<T>\* root);

    bool searchNode(const T& value, Node<T>\* root);

    int getMaxWidth(Node<T>\* root);

    int getWidth(Node<T>\* root, int level, myVector<T>& levels);

    int getHeight(Node<T>\* root);

};

#endif

***BinaryTree.cpp***

#include "BinaryTree.hpp"

#include "MyVector.hpp"

#include "MyVector.cpp"

#include <iostream>

template <typename T>

BinaryTree<T>::BinaryTree(): root(nullptr){}

template <typename T>

BinaryTree<T>::BinaryTree(const T& value){

    this->root = new Node<T>(value);

}

template <typename T>

BinaryTree<T>::~BinaryTree(){

    deleteTree(root);

}

template <typename T>

Node<T>\* BinaryTree<T>::getRoot() const{

    return this->root;

}

template <typename T>

void BinaryTree<T>::deleteTree(Node<T>\* node){

    if (node == nullptr) return;

    deleteTree(node->left);

    deleteTree(node->right);

    delete node;

}

template <typename T>

void BinaryTree<T>::insertNode(const T& value, Node<T>\* root){

    if (value < root->data){

        if (root->left == nullptr){

            root->left = new Node<T>(value);

        } else{

            insertNode(value, root->left);

        }

    } else{

        if (root->right == nullptr){

            root->right = new Node<T>(value);

        } else{

            insertNode(value, root->right);

        }

    }

    return;

}

template <typename T>

Node<T>\* BinaryTree<T>::removeNode(const T& value, Node<T>\* root){

    if (root == nullptr) return root;

    if (value < root->data){

        root->left = removeNode(value, root->left);

    } else if (value > root->data){

        root->right = removeNode(value, root->right);

    } else{

        if (root->left == nullptr && root->right == nullptr){

            delete root;

            return nullptr;

        } else if(root->left == nullptr){

            Node<T>\* tmp = root->right;

            delete root;

            return tmp;

        } else if (root->right == nullptr){

            Node<T>\* tmp = root->left;

            delete root;

            return tmp;

        }

        Node<T>\* tmp = findMin(root->right);

        root->data = tmp->data;

        root->right = removeNode(tmp->data, root->right);

    }

    return root;

}

template <typename T>

Node<T>\* BinaryTree<T>::findMin(Node<T>\* root){

    while (root->left != nullptr){

        root = root->left;

    }

    return root;

}

template <typename T>

void BinaryTree<T>::visualize(Node<T>\* root, const int height) const{

    if (root != nullptr){

        visualize(root->right, height + 1);

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

            std::cout << "\t";

        }

        std::cout << root->data << "\n";

        visualize(root->left, height + 1);

    }

}

template <typename T>

void BinaryTree<T>::inOrderBypass(Node<T>\* root){

    if (root == nullptr) return;

    inOrderBypass(root->left);

    std::cout << root->data << " ";

    inOrderBypass(root->right);

}

template <typename T>

bool BinaryTree<T>::searchNode(const T& value, Node<T>\* root){

    if (root == nullptr) return false;

    if (value == root->data) return true;

    if (value < root->data){

        searchNode(value, root->left);

    } else{

        searchNode(value, root->right);

    }

}

template <typename T>

int BinaryTree<T>::getMaxWidth(Node<T>\* root){

    int height = getHeight(root);

    int max\_width = 0;

    myVector<T> levels;

    levels.resize(height);

    getWidth(root, 0, levels);

    for (size\_t i = 0; i != height; ++i){

        if (levels[i] > max\_width) max\_width = levels[i];

    }

    return max\_width;

}

template <typename T>

int BinaryTree<T>::getWidth(Node<T>\* root, int level, myVector<T>& levels){

    if (root == nullptr) return 0;

    ++levels[level];

    getWidth(root->left, level + 1, levels);

    getWidth(root->right, level + 1, levels);

}

template <typename T>

int BinaryTree<T>::getHeight(Node<T>\* root){

    if (root == nullptr) return 0;

    int left\_height = getHeight(root->left);

    int right\_height = getHeight(root->right);

    return ((left\_height > right\_height) ? left\_height : right\_height) + 1;

}

***Run.cpp***

#include "Node.hpp"

#include "Node.cpp"

#include "BinaryTree.hpp"

#include "BinaryTree.cpp"

#include <iostream>

int main(){

    BinaryTree<int> bin\_tree(10);

    bin\_tree.insertNode(15, bin\_tree.getRoot());

    bin\_tree.insertNode(8, bin\_tree.getRoot());

    bin\_tree.insertNode(5, bin\_tree.getRoot());

    bin\_tree.insertNode(4, bin\_tree.getRoot());

    bin\_tree.insertNode(3, bin\_tree.getRoot());

    bin\_tree.insertNode(17, bin\_tree.getRoot());

    bin\_tree.insertNode(14, bin\_tree.getRoot());

    bin\_tree.insertNode(13, bin\_tree.getRoot());

    bin\_tree.insertNode(16, bin\_tree.getRoot());

    bin\_tree.insertNode(18, bin\_tree.getRoot());

    bin\_tree.insertNode(6, bin\_tree.getRoot());

    bin\_tree.removeNode(15, bin\_tree.getRoot());

    bin\_tree.visualize(bin\_tree.getRoot());

    std::cout << "\nWidth of tree: " << bin\_tree.getMaxWidth(bin\_tree.getRoot()) << "\n";

    return 0;

}

**9. Дневник отладки** *(дата и время сеансов отладки и основные события [ошибки в сценарии и программе, нестандартные ситуации] и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы)*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *№* | *Лаб. или дом* | *Дата* | *Время* | *Событие* | *Действие по исправлению* | *Примечания* |

Особых проблем при выполнении лабы не возникло

**10. Замечания автора** *(по существу работы)*

Замечания отсутствуют

**11. Вывод**

Данная лабораторная дает хорошую базу в понимании деревьев, учит работать с ними. Благодаря тому, что пришлось самостоятельно реализовать дерево, получилось глубже разобраться в этой структуре данных, которая еще не раз встретится в моей программистской жизни. Пришлось немного помучаться с функцией нахождения ширины, так как не сразу смог придумать хороший и оптимальный алгоритм

Работа на 10/10

Подпись студента \_\_\_\_\_\_\_\_\_\_