**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 3**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **«Деревья»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3311 | Баймухамедов Р. Р. |  |
| Преподаватель | Манирагена В. |  |

Санкт-Петербург

2024

**Цель работы**

Исследование алгоритмов для работы с двоичным деревом

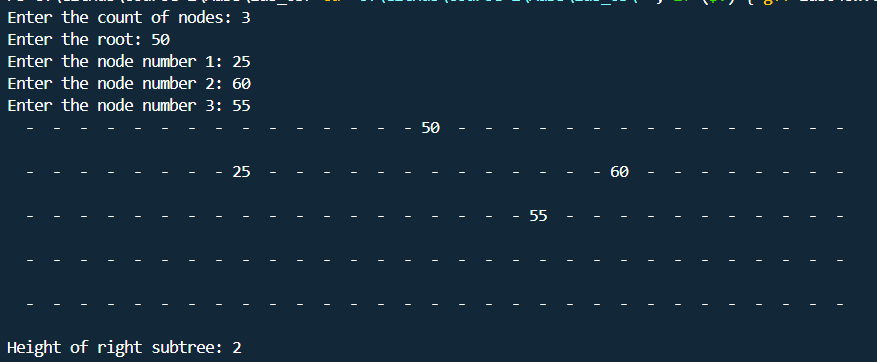
**Задание на работу с деревьями**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варианта | Вид дерева | Разметка | Способ обхода | Что надо вычислить |
| 21 | Двоичное | Прямая | Внутренний | Высоту правого поддерева для корня |

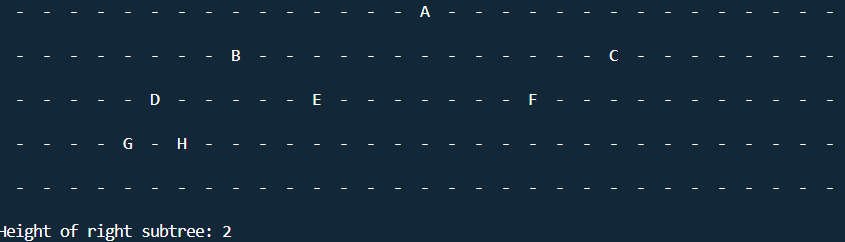
**Обоснование выбора способа представления деревьев в памяти ЭВМ**

По лабораторным работам №1 и №2 было принято решение о реализации лабораторной работе без использования класса по причине экономии времени выполнения программы

**Тестовый пример**



**Результаты прогона программы с генерацией случайного дерева**

****

Оценки временной сложности каждой функции обхода дерева, использованной в программе: создание дерева, обработка, вывод.

Создание дерева имеет сложность O(N)

Обработка дерева также имеет сложность O(N)

Вывод дерева аналогично имеет сложность O(N), т.к. при выводе мы проходим по каждому из узлов.

(где N — количество узлов в дереве)

**Выводы о результатах испытания алгоритмов обхода деревьев**

Программа работает корректно. Рекурсивные функции для обхода деревьев успешно применены.

**Приложение**

Программа с вводом вручную

#include <iostream>

#include <string>

#include <time.h>

#include <random>

#include <cmath>

#include <iomanip>

#include <queue>

struct *TreeNode* { *// structure of tree*

    int value; *// value of int data*

*TreeNode*\* left; *// left child*

*TreeNode*\* right; *// right child*

    std::*string* name;

    TreeNode(int val) : value(val), left(nullptr), right(nullptr), name("") {}

};

void fill\_levels(*TreeNode\** node, std::vector<std::vector<std::*string*>>*&* levels, int depth, int pos, int width) {

    if (node == nullptr) return;

    levels[depth][pos] = std::to\_string(node->value); *// добавляем значение узла*

*// levels[depth][pos] = node->name;*

    int offset = width / 2; *// Делим ширину пополам для следующего уровня*

    if (offset > 0) {

*// Рекурсивно добавляем дочерние узлы, сдвигая на половину оставшейся ширины*

        fill\_levels(node->left, levels, depth + 1, pos - offset, offset);

        fill\_levels(node->right, levels, depth + 1, pos + offset, offset);

    }

}

void print\_tree(*TreeNode\** root) {

    int depth = 5; *// Максимальная глубина (количество уровней)*

    int width = (1 << depth) - 1; *// Ширина для центрирования дерева*

    std::vector<std::vector<std::*string*>> levels(depth, std::vector<std::*string*>(width, "-")); *// Создаем пустую сетку*

    fill\_levels(root, levels, 0, (width - 1) / 2, (width - 1) / 2); *// Начинаем с ширины всего уровня*

*// Печатаем уровни*

    for (*const* auto& level : levels) {

        for (*const* auto& node : level) {

            std::cout << std::setw(3) << node;

        }

        std::cout << std::endl << std::endl;

    }

}

*TreeNode\** insert(*TreeNode\** root, int value) {

    if (root == nullptr) {

        return new *TreeNode*(value);

    }

    if (value < root->value) {

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

    } else if (value > root->value) {

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

    }

    return root;

}

*TreeNode\** create\_tree(*TreeNode\** root, int node\_count){

    int root\_value, value;

    std::cout << "Enter the root: ";

    std::cin >> root\_value;

    root = new *TreeNode*(root\_value);

    for(int i=0;i<node\_count;i++){

        std::cout << "Enter the node number " << i+1 << ": ";

        std::cin >> value;

        root = insert(root, value);

    }

    return root;

}

*// Функция для поуровневого присвоения имен узлам*

void assign\_names\_by\_level(*TreeNode\** root) {

    if (root == nullptr) return;

    std::queue<*TreeNode*\*> q;

    q.push(root);

    char current\_name = 'A';

*// Обходим дерево по уровням*

    while (!q.empty()) {

*TreeNode*\* node = q.front();

        q.pop();

        node->name = std::*string*(1, current\_name++); *// присваиваем текущий символ и инкрементируем*

        if (node->left) q.push(node->left);

        if (node->right) q.push(node->right);

    }

}

*// Рекурсивная функция для вычисления высоты дерева*

int calculate\_height(*TreeNode\** node) {

    if (node == nullptr) return 0;

    int left\_height = calculate\_height(node->left);

    int right\_height = calculate\_height(node->right);

    return 1 + std::max(left\_height, right\_height);

}

*// Функция для вычисления высоты правого поддерева корня*

int height\_of\_right\_subtree(*TreeNode\** root) {

    if (root == nullptr || root->right == nullptr) return 0;

    return calculate\_height(root->right);

}

int main(){

    int node\_count;

    srand(time(0));

    std::cout << "Enter the count of nodes: ";

    std::cin >> node\_count;

*TreeNode*\* root = nullptr;

    root = create\_tree(root ,node\_count);

    assign\_names\_by\_level(root);

    print\_tree(root);

    int height\_of\_r\_subtree = height\_of\_right\_subtree(root);

    std::cout << "Height of right subtree: " << height\_of\_r\_subtree << "\n" << std::endl;

    return 0;

}

Программа с генерацией случайного дерева

#include <iostream>

#include <string>

#include <time.h>

#include <random>

#include <cmath>

#include <iomanip>

#include <queue>

struct *TreeNode* { *// structure of tree*

    int value; *// value of int data*

*TreeNode*\* left; *// left child*

*TreeNode*\* right; *// right child*

    std::*string* name;

    TreeNode(int val) : value(val), left(nullptr), right(nullptr), name("") {}

};

void fill\_levels(*TreeNode\** node, std::vector<std::vector<std::*string*>>*&* levels, int depth, int pos, int width) {

    if (node == nullptr) return;

*// levels[depth][pos] = std::to\_string(node->value); // добавляем значение узла*

    levels[depth][pos] = node->name;

    int offset = width / 2; *// Делим ширину пополам для следующего уровня*

    if (offset > 0) {

*// Рекурсивно добавляем дочерние узлы, сдвигая на половину оставшейся ширины*

        fill\_levels(node->left, levels, depth + 1, pos - offset, offset);

        fill\_levels(node->right, levels, depth + 1, pos + offset, offset);

    }

}

void print\_tree(*TreeNode\** root) {

    int depth = 5; *// Максимальная глубина (количество уровней)*

    int width = (1 << depth) - 1; *// Ширина для центрирования дерева*

    std::vector<std::vector<std::*string*>> levels(depth, std::vector<std::*string*>(width, "-")); *// Создаем пустую сетку*

    fill\_levels(root, levels, 0, (width - 1) / 2, (width - 1) / 2); *// Начинаем с ширины всего уровня*

*// Печатаем уровни*

    for (*const* auto& level : levels) {

        for (*const* auto& node : level) {

            std::cout << std::setw(3) << node;

        }

        std::cout << std::endl << std::endl;

    }

}

*TreeNode\** insert(*TreeNode\** root, int value) {

    if (root == nullptr) {

        return new *TreeNode*(value);

    }

    if (value < root->value) {

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

    } else if (value > root->value) {

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

    }

    return root;

}

*TreeNode\** create\_tree(*TreeNode\** root, int node\_count){

    int root\_value;

    root\_value = rand()%100;

    root = new *TreeNode*(root\_value);

    for(int i=0; i<node\_count;i++){

        int value = rand()%100;

        root = insert(root, value);

    }

    return root;

}

*// Функция для поуровневого присвоения имен узлам*

void assign\_names\_by\_level(*TreeNode\** root) {

    if (root == nullptr) return;

    std::queue<*TreeNode*\*> q;

    q.push(root);

    char current\_name = 'A';

*// Обходим дерево по уровням*

    while (!q.empty()) {

*TreeNode*\* node = q.front();

        q.pop();

        node->name = std::*string*(1, current\_name++); *// присваиваем текущий символ и инкрементируем*

        if (node->left) q.push(node->left);

        if (node->right) q.push(node->right);

    }

}

*// Рекурсивная функция для вычисления высоты дерева*

int calculate\_height(*TreeNode\** node) {

    if (node == nullptr) return 0;

    int left\_height = calculate\_height(node->left);

    int right\_height = calculate\_height(node->right);

    return 1 + std::max(left\_height, right\_height);

}

*// Функция для вычисления высоты правого поддерева корня*

int height\_of\_right\_subtree(*TreeNode\** root) {

    if (root == nullptr || root->right == nullptr) return 0;

    return calculate\_height(root->right);

}

int main(){

    srand(time(0));

    int node\_count = 5 + rand() % 10;

*TreeNode*\* root = nullptr;

    root = create\_tree(root ,node\_count);

    assign\_names\_by\_level(root);

    print\_tree(root);

    int height\_of\_r\_subtree = height\_of\_right\_subtree(root);

    std::cout << "Height of right subtree: " << height\_of\_r\_subtree << "\n" << std::endl;

    return 0;

}