|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | | |  | | | | | |
|  | | |
|  | Институт информационных технологий (ИТ) |
|  | Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ) |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4** | |
| **по дисциплине** | |
| **«Структуры и алгоритмы обработки данных»**  **По теме: «Структура данных - дерево»**  **Вариант 2** | |
|  | |
| Выполнил студент группы ИВБО-06-17 | Бикеев А. И. |
| Принял преподаватель | Скворцова Л.А. |

Москва 2019

**Оглавление**

[1. Задание 1. Разработать класс бинарное дерево, включив в него методы: конструктор по умолчанию и вывод бинарного дерева, используя прямой обход дерева, разворачивая дерево горизонтально. 3](#_Toc26230334)

[1.1. Вариант задания 3](#_Toc26230335)

[1.2. Абстрактный тип данных (далее АТД) для варианта задания, включая список общих функций из задания. 3](#_Toc26230336)

[1.3. Реализация АТД 3](#_Toc26230337)

[Список модулей реализации АТД (или описать где расположена 4](#_Toc26230338)

[реализация АТД) 4](#_Toc26230339)

[1.4. Таблица тестов 4](#_Toc26230340)

[1.5. Текст исходного кода (листинг) программы 4](#_Toc26230341)

[2. Задание 2. Разработать производный класс для красно-черного дерева 6](#_Toc26230342)

[2.1. Вариант задания 6](#_Toc26230343)

[2.2. Реализация АТД 7](#_Toc26230344)

[Список модулей реализации АТД (или описать где расположена 7](#_Toc26230345)

[реализация АТД) 7](#_Toc26230346)

[2.3. Таблица тестов 8](#_Toc26230347)

[2.4. Текст исходного кода (листинг) программы 8](#_Toc26230348)

[3. Дополнительные задания 12](#_Toc26230349)

[3.1. Вариант задания 12](#_Toc26230350)

[3.2. Вариант задания 12](#_Toc26230351)

[4. Общие контрольные прогоны программы 12](#_Toc26230352)

[Выводы 14](#_Toc26230353)

1. Задание 1. Разработать класс бинарное дерево, включив в него методы: конструктор по умолчанию и вывод бинарного дерева, используя прямой обход дерева, разворачивая дерево горизонтально.

1.1. Вариант задания

Информационная часть узла содержит символьное значение.

* Создать дерево из n узлов.
* Определить сумму цифр, содержащихся в дереве, применив алгоритм симметричного обхода
* Найти первое вхождение заданного символа и вернуть указатель на узел
* Найти максимальное значение среди листьев дерева

## 1.2. Абстрактный тип данных (далее АТД) для варианта задания, включая список общих функций из задания.

Абстрактный тип данных BinaryTreeNode

**left** – указатель на левую ветку дерева текущей вершины

**right** – указатель на правую ветку дерева текущей вершины

**char** – информационная часть вершины

Операции:

// Добавляет новый узел с заданным значением

BinaryTreeNode\* InsertNode(char value);

// Используя прямой обход – разворачивает дерево горизонтально

void PrintTree(BinaryTreeNode\* p, int level);

## 1.3. Реализация АТД

АТД реализуется на одномерном динамическом массиве.

typedef struct Node {

/// <summary>

/// Левый узел

/// </summary>

struct Node\* left;

/// <summary>

/// Правый ухел

/// </summary>

struct Node\* right;

/// <summary>

/// Информационная часть вершины parent

/// </summary>

char data;

Node() {

this->color = NodeColor::Red;

this->left = 0;

this->parent = 0;

this->right = 0;

this->data = ' ';

}

} BinaryTreeNode;

Список модулей реализации АТД (или описать где расположена

реализация АТД)

Реализация АТД находится в файле BinaryTreeNode.h

## 1.4. Таблица тестов

|  |  |  |
| --- | --- | --- |
| Номер теста | Исходные данные | Эталон результата |
| 1 | Узлы дерева  49=’1’, 50=’2’ | Сумма значений в узлов дерева: 99 |
| 2 | Узлы дерева  49=’1’, 50=’2’  Ищем узел со значением ‘2’ = 50 | Узел найден, его адрес: 000000FB1B30F848 его значение: 2 |
| 3 | Узлы дерева  49=’1’, 50=’2’  Ищем узел со значением ‘2’ = 50 | Максимальный элемент дерева имеет значение: 2 и численно равен: 50 |

## 1.5. Текст исходного кода (листинг) программы

#include <iostream>

#include <windows.h>

#include "Operation.h"

using namespace std;

int main()

{

    SetConsoleCP(1251);

    SetConsoleOutputCP(1251);

    short answer = 10;

    RedBlackTree\* rbTree = nullptr;

    int n = 0;

    while (answer != 0) {

        system("cls");

        cout << "Лабораторная работа №4 Бикеева А. И. Вариант 2\n\n";

        cout << "Использовать  сбалансированное бинарное дерево (Использую Красно-Черные деревья)\n";

        cout << "Информационная часть узла содержит символьное значение\n";

        cout << "\t• Создать дерево из n узлов.\n";

        cout << "\t• Определить сумму цифр, содержащихся в дереве, применив алгоритм симметричного обхода\n";

        cout << "\t• Найти первое вхождение заданного символа и вернуть указатель на узел\n";

        cout << "\t• Найти максимальное значение среди листьев дерева\n";

        cout << "\nМеню\n";

        cout << "1) Создать новый список из n узлов\n";

        cout << "2) Определить сумму цифр, содержащихся в дереве, применив алгоритм симметричного обхода\n";

        cout << "3) Найти первое вхождение заданного символа и вернуть указатель на узел \n";

        cout << "4) Найти максимальное значение среди листьев дерева\n";

        cout << "0) Выход\n";

        cout << "\nВаш выбор: ";

        while ((!(cin >> answer)) || cin.get() != '\n') {     //делаем проверку на ввод букв

            cout << "ERROR\n";

            cin.clear(); // сбрасывает все биты потока, тоесть поток становится "хорошим"

            cin.sync();//Удалим данные из буффера

            cout << "Повторите ввод : ";

        }

        system("cls");

        cout << "Лабораторная работа №4 Бикеева А. И. Вариант 2\n\n";

        switch (answer)

        {

            case 0: {

                system("pause");

                break;

            }

            case 1: {

                cout << "Введите кол-во узлов в дереве: ";

                cin >> n;

                if (rbTree != nullptr)

                    free(rbTree);

                rbTree = new RedBlackTree();

                CreateTree(rbTree, n);

                PrintTree(rbTree->GetRoot(), n);

                system("pause");

                break;

            }

            case 2: {

                if (rbTree)

                {

                    PrintTree(rbTree->GetRoot(), n);

                    cout << "Сумма значений в узлов дерева: " << InOrderTravelRL(rbTree->GetRoot(), 0) << endl;

                }

                else {

                    cout << "Для начало задайте не пустое дерево\n";

                }

                system("pause");

                break;

            }

            case 3: {

                if (rbTree)

                {

                    cout << "Введите значение для поиска в дереве: ";

                    char c;

                    cin >> c;

                    Node\* node = FindNode(rbTree->GetRoot(), c);

                    if (node && node->data != ' ') {

                        cout << "Узел найден, его адрес: " << &node << " его значение: " << node->data << endl;

                    }

                    else {

                        cout << "Данного узла нет в дереве\n";

                    }

                }

                else {

                    cout << "Для начало задайте не пустое дерево\n";

                }

                system("pause");

                break;

            }

            case 4: {

                if (rbTree) {

                    BinaryTreeNode\* max\_node = FindMax(rbTree->GetRoot());

                    cout << "Максимальный элемент дерева имеет значение: " << max\_node->data << " и численно равен: " << (int)max\_node->data << endl;

                }else {

                    cout << "Для начало задайте не пустое дерево\n";

                }

                system("pause");

                break;

            }

            default:

                break;

        }

    }

    return 0;

}

2. Задание 2. Разработать производный класс для красно-черного дерева

2.1. Вариант задания

Требования

В класс включить методы:

• Конструктор с параметром (для сбалансированного дерева) – число узлов.

• Копирующий конструктор

• Деструктор

• Методы для операций, определенных в варианте.

## 2.2. Реализация АТД

АТД реализуется на однонаправленном списке.

Абстрактный тип данных RedBlackTree

**left** – указатель на левую ветку дерева текущей вершины

**right** – указатель на правую ветку дерева текущей вершины

**color –** цвет узла дерева

**char** – информационная часть вершины

Операции:

// Добавляет новый узел с заданным значением

BinaryTreeNode\* InsertNode(char value);

// Создает пустой узел дерева

Node();

// Создает узел NIL

Node(struct Node\* left, struct Node\* right, struct Node\* parent, NodeColor color, char data);

// Копирущий конструктор

Node(const Node\*& root);

Список модулей реализации АТД (или описать где расположена

реализация АТД)

Реализация АТД расположена в трех файлах: RedBlackTree.cpp и RedBlackTree.h

#pragma once

#include "BinaryTreeNode.h"

class RedBlackTree: BinaryTreeNode

{

private:

/// <summary>

/// Вращение дерева влево

/// </summary>

/// <param name="node">Узел дерева, ветви которого нужно вращать</param>

/// <returns></returns>

BinaryTreeNode\* RotateToLeft(BinaryTreeNode\* node);

/// <summary>

/// Балансировка дерева на основе вставляемого узла

/// </summary>

/// <param name="root">Новый узел дерева</param>

void BalanceInsert(BinaryTreeNode\*\* root);

/// <summary>

/// Добавляет новый узел с заданным значением

/// </summary>

/// <param name="value">Значение узла</param>

/// <returns></returns>

BinaryTreeNode\* InsertNode(char value);

public:

/// <summary>

/// Инициализирует пустой узел дерева

/// </summary>

RedBlackTree();

~RedBlackTree();

/// <summary>

/// Рекурсивно вставка узла.

/// Возвращает true если изменений не было или если балансировка в вершине не нужна

/// </summary>

/// <param name="value">Значение добавляемого узла</param>

/// <param name="root">Указатель на головной узел дерева/поддерева</param>

/// <returns></returns>

bool Insert(char value, BinaryTreeNode\*\* root);

};

## 2.3. Таблица тестов

Тесты аналогичны тестам из задания 1.

## 2.4. Текст исходного кода (листинг) программы

#include "RedBlackTree.h"

#define NIL sentinel

Node\* sentinel = new Node(NIL, NIL, 0, NodeColor::Black, ' ');

RedBlackTree::RedBlackTree()

{

    this->root = NIL;

}

RedBlackTree::~RedBlackTree()

{

}

void RedBlackTree::BalanceInsert(Node\* x)

{

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

        \*  maintain Red-Black tree balance  \*

        \*  after inserting node x           \*

        \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

        /\* check Red-Black properties \*/

    while (x != this->root && x->parent->color == NodeColor::Red) {

        /\* we have a violation \*/

        if (x->parent == x->parent->parent->left) {

            Node\* y = x->parent->parent->right;

            if (y->color == NodeColor::Red) {

                /\* uncle is RED \*/

                x->parent->color = NodeColor::Black;

                y->color = NodeColor::Black;

                x->parent->parent->color = NodeColor::Red;

                x = x->parent->parent;

            }

            else {

                /\* uncle is BLACK \*/

                if (x == x->parent->right) {

                    /\* make x a left child \*/

                    x = x->parent;

                    RotateLeft(x);

                }

                /\* recolor and rotate \*/

                x->parent->color = NodeColor::Black;

                x->parent->parent->color = NodeColor::Red;

                RotateRight(x->parent->parent);

            }

        }

        else {

            /\* mirror image of above code \*/

            Node\* y = x->parent->parent->left;

            if (y->color == NodeColor::Red) {

                /\* uncle is RED \*/

                x->parent->color = NodeColor::Black;

                y->color = NodeColor::Black;

                x->parent->parent->color = NodeColor::Red;

                x = x->parent->parent;

            }

            else {

                /\* uncle is BLACK \*/

                if (x == x->parent->left) {

                    x = x->parent;

                    RotateRight(x);

                }

                x->parent->color = NodeColor::Black;

                x->parent->parent->color = NodeColor::Red;

                RotateLeft(x->parent->parent);

            }

        }

    }

    root->color = NodeColor::Black;

}

Node\* RedBlackTree::InsertNode(char data)

{

    Node\* current, \* parent, \* x;

    current = root;

    parent = 0;

    // Перебираем узлы дерева, вплоть до листьев

    // или пока не встретим элемент с теми эе данными

    while (current != NIL)

    {

        if (data == current->data)

            return current;

        parent = current;

        current = data > current->data ? current->right : current->left;

    }

    // Для нового узла создаем объект вершины

    x = new Node();

    x->data = data;

    x->parent = parent;

    x->left = NIL;

    x->right = NIL;

    x->color = NodeColor::Red;

    // Если дерево имеет хотяб 1 вершину

    if (parent) {

        // Определяем с какой стороны вставить новый узел

        if (data < parent->data) {

            parent->left = x;

        }

        else {

            parent->right = x;

        }

        //x->parent = parent;

    }

    else {

        // Иначе делаем вершину главной

        root = x;

    }

    BalanceInsert(x);

    return x;

}

void RedBlackTree::RotateRight(Node\* x)

{

    Node\* y = x->left;

    x->left = y->right;

    if (y->right != NIL) y->right->parent = x;

    if (y != NIL) y->parent = x->parent;

    if (x->parent) {

        if (x == x->parent->right)

            x->parent->right = y;

        else

            x->parent->left = y;

    }

    else {

        this->root = y;

    }

    y->right = x;

    if (x != NIL) x->parent = y;

}

void RedBlackTree::RotateLeft(Node\* x)

{

    Node\* y = x->right;

    x->right = y->left;

    if (y->left != NIL) y->left->parent = x;

    if (y != NIL) y->parent = x->parent;

    if (x->parent) {

        if (x == x->parent->left)

            x->parent->left = y;

        else

            x->parent->right = y;

    }

    else {

        this->root = y;

    }

    y->left = x;

    if (x != NIL) x->parent = y;

}

Node\* RedBlackTree::GetRoot()

{

    return this->root;

}

3. Дополнительные задания

3.1. Вариант задания

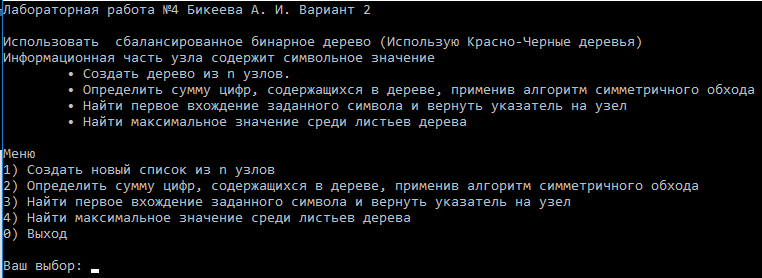
Использовать сбалансированное бинарное дерево (не AVL) (ЕСЛИ ВАРИАНТ НЕ ПРЕДУСМАТРИВАЕТ СВОЮ СТРУКТУРУ).

3.2. Вариант задания

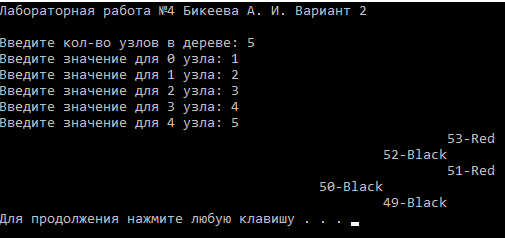
Реализовать задачу с использованием красно-черного дерева. Добавить тестирование в созданную программу.

4. Общие контрольные прогоны программы

Меню

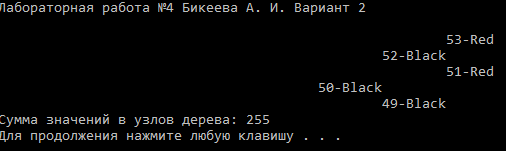


1 пункт

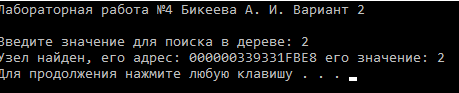


1 пункт

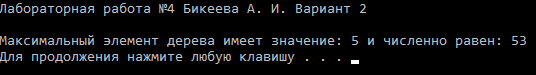
2 пункт



3 пункт



4 пункт



Выводы

В результате выполнения лабораторной работы были получены навыки по созданию структуры данных бинарное дерево и ее применению.

Бинарное дерево было разработано сбалансированным используя такое представление дерева как «красно-черное дерево», используя свойства которой и удается оставаться дереву сбалансированным.