МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

Институт информационных технологий (ИТ)

Кафедра математического обеспечения и стандартизации информационных

технологий (МОСИТ)

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 7**

**по дисциплине «***Структуры и алгоритмы обработки данных***»**

**тема «**Двоичное бинарное дерево**»**

Выполнил: студент группы ИВБО-02-15 Прохоров А.В.

Приняла: старший преподаватель кафедры Скворцова Л.А.

**1. Условие задания**

Цель: получение навыков по созданию иерархических структур.

**Задания:**

1. Разработать класс дерево, включив в него методы: конструктор по умолчанию и вывод бинарного дерева.
2. Разработать класс бинарное дерево. В класс включить методы:
   * Конструктор с параметром
   * Копирующий конструктор
   * Деструктор
   * Методы для операций, определенных в варианте.

3. В нечетном варианте использовать бинарное дерево поиска, а в четном – сбалансированное бинарное дерево (ЕСЛИ ВАРИАНТ НЕ ПРЕДУСМАТРИВАЕТ СВОЮ СТРУКТУРУ).

**Вариант 20 (6):**

Информационная часть узла целое число.

* Заменить в дереве все отрицательные значения на их модули
* Найти минимальное значение, хранящееся в дереве.
* Обойти дерево “в ширину”.

**2. Определение структуры представления данных и операций над данными**

**Двоичное дерево** — иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей). Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками.

**АВЛ-дерево** — сбалансированное по высоте двоичное дерево: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

**Обход в ширину —** такой обход, когда узлы посещаются уровень за уровнем (N-й уровень дерева - множество узлов с высотой N). Каждый уровень обходится слева направо.Для реализации используется структура queue — очередь.

**3. Разработка программы**

**3.1. Декомпозиция**

Список подзадач:

1) Добавление элемента в дерево

2) Обход дерева «в ширину»

3) Балансировка дерева

4) Удаление элемента дерева

5) Замена отрицательных элементов на их модули

6) Поиск минимального элемента

**3.2. Определение функций**

void addToTree(TreeNode \*\*Ptr, int value) — добавление элемента

void levelOrder()—обход «в шиирну»

TreeNode\* balance(TreeNode \*Ptr)—балансировка узла

void deleteTree(TreeNode \*Ptr, int value)—удаление элемента

void outputTreePlus(TreeNode \*Ptr, int totalSpace)—замена отрицательных элементов на их модули.

void MinValue(TreeNode \*Ptr)—поиск минимального элемента

**3.3. Код разработанного приложения**

#include <iostream>

#include <iomanip>

using namespace std;

//узел дерева

struct TreeNode

{

int data;

short bal;

TreeNode \*left, \*right;

};

//очередь узла

struct QueueNode

{

TreeNode \*trans;

QueueNode \*next;

};

class Queue

{

private:

QueueNode \*Head;

public:

Queue() { Head = NULL; }

int isEmpty();

void addToQueue(TreeNode \*tmp);

TreeNode \*delQueue()

{

QueueNode \*cur = Head->next;

TreeNode \*tmp\_c = Head->trans;

if (Head != NULL)

{

delete Head;

Head = cur;

return tmp\_c;

}

else

return NULL;

}

};

int Queue::isEmpty()

{

if (Head == NULL)

return 1;

else

return 0;

}

void Queue::addToQueue(TreeNode \*tmp)

{

QueueNode \*Ptr = new QueueNode;

Ptr->trans = tmp;

if (Head == NULL)

{

Head = Ptr;

Ptr->next = NULL;

}

else

{

QueueNode \*tmp\_c = Head;

while (tmp\_c->next != NULL)

tmp\_c = tmp\_c->next;

tmp\_c->next = Ptr;

Ptr->next = NULL;

}

}

class Tree

{

private:

TreeNode \*root, \*next;

public:

Tree() { root = NULL; } //конструктор

Tree(int TreeNode); //конструктор с параметром

Tree(const Tree&); //конструктор копии

~Tree() { //деструктор

delete[] root;

}

//обход в ширину

void levelOrder()

{

Queue Universal;

Universal.addToQueue(root);

TreeNode \*tmp;

while (!Universal.isEmpty())

{

tmp = Universal.delQueue();

cout << tmp->data << " ";

if (tmp->left != NULL)

Universal.addToQueue(tmp->left);

if (tmp->right != NULL)

Universal.addToQueue(tmp->right);

}

}

void addToTreeHelper(int value)

{

addToTree(&root, value);

}

//добавление элемента

void addToTree(TreeNode \*\*Ptr, int value)

{

if ((\*Ptr) == NULL)

{

(\*Ptr) = new TreeNode;

(\*Ptr)->data = value;

(\*Ptr)->left = (\*Ptr)->right = NULL;

}

else

{

if (value < (\*Ptr)->data)

addToTree(&(\*Ptr)->left, value);

else if (value >(\*Ptr)->data)

addToTree(&(\*Ptr)->right, value);

else

cout << "Дубликат!" << endl;

}

balanceHelper();

}

void deleteTreeHelp(int value)

{

deleteTree(root, value);

}

//удаление элемента

void deleteTree(TreeNode \*Ptr, int value)

{

TreeNode \*parent = NULL;

TreeNode \*removed = NULL;

while (Ptr != NULL && Ptr->data != value)

{

parent = Ptr;

if (Ptr->data > value)

Ptr = Ptr->left;

else

Ptr = Ptr->right;

}

int peremen = Ptr->data;

if (root != NULL)

{

TreeNode \*child = NULL;

if (Ptr->left == NULL || Ptr->right == NULL)

{

/\*случай когда один ребенок\*/

if (Ptr->left == NULL)

child = Ptr->right;

else

child = Ptr->left;

/\*усли удаляем корень с одним потомком\*/

if (root->data == value)

root = child;

else

{

if (parent->left == Ptr)

parent->left = child;

else

parent->right = child;

}

removed = Ptr;

}

else /\*if(Ptr -> left != NULL || Ptr -> right != NULL)\*/

{

/\*случай когда 2 ребенка\*/

TreeNode \*mostLeft = Ptr->right;

while (mostLeft->left != NULL)

{

parent = mostLeft;

mostLeft = mostLeft->left;

}

if (root->data == value) {

removed = mostLeft;

if (parent->left == mostLeft)

parent->left = NULL;

else

parent->right = NULL;

root->data = mostLeft->data;

}

else

{

removed = mostLeft;

if (Ptr->left == mostLeft)

Ptr->left = NULL;

else

Ptr->right = NULL;

Ptr->data = mostLeft->data;

}

}

}

delete removed;

}

void outputHelper(int totalSpace)

{

outputTree(root, totalSpace);

}

//вывод дерева

void outputTree(TreeNode \*Ptr, int totalSpace)

{

while (Ptr != NULL)

{

outputTree(Ptr->right, totalSpace + 5);

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

cout << " ";

cout << Ptr->data << "\n";

Ptr = Ptr->left;

totalSpace += 5;

}

}

void outputHelperPlus(int totalSpace)

{

outputTreePlus(root, totalSpace);

}

//замена отрицательных элементов

void outputTreePlus(TreeNode \*Ptr, int totalSpace)

{

while (Ptr != NULL)

{

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

cout << " ";

if ((Ptr->data)<0) {

(Ptr->data) = (Ptr->data)\*(-1);

}

Ptr = Ptr->left;

totalSpace += 5;

}

}

void MinValueHelper() {

MinValue(root);

}

//минимальный элемент дерева

void MinValue(TreeNode \*Ptr) {

while (Ptr->left != NULL ) {

Ptr = Ptr->left;

}

cout << ( Ptr->data);

}

// вычисление фактора баланса

short factor(TreeNode \*Ptr) {

short fr = (Ptr->right != NULL) ? Ptr->right->bal : 0;

short fl = (Ptr->left != NULL) ? Ptr->left->bal : 0;

return (fr - fl);

}

void balanceHelper() {

balance(root);

}

// балансировка узла

TreeNode\* balance(TreeNode \*Ptr) {

short lh = (Ptr->left != NULL) ? Ptr->left->bal : 0;

short rh = (Ptr->right != NULL) ? Ptr->right->bal : 0;

Ptr->bal = ((lh > rh) ? lh : rh) + 1;

short fac = factor(Ptr);

if (fac == 2) {

if (factor(Ptr->right) < 0)

Ptr->right = rotate(Ptr->right, false);

return rotate(Ptr, true);

}

else if (fac == -2) {

if (factor(Ptr->left) > 0)

Ptr->left = rotate(Ptr->left, true);

return rotate(Ptr, false);

}

return Ptr;

}

// ротация влево или вправо

TreeNode\* rotate(TreeNode \*Ptr, bool left) {

short lh, rh;

TreeNode\* t;

if (left) {

t = Ptr->right;

Ptr->right = t->left;

t->left = Ptr;

}

else {

t = Ptr->left;

Ptr->left = t->right;

t->right = Ptr;

}

lh = (Ptr->left != NULL) ? Ptr->left->bal : 0;

rh = (Ptr->right != NULL) ? Ptr->right->bal : 0;

Ptr->bal = ((lh > rh) ? lh : rh) + 1;

lh = (t->left != NULL) ? t->left->bal : 0;

rh = (t->right != NULL) ? t->right->bal : 0;

t->bal = ((lh > rh) ? lh : rh) + 1;

return t;

}

};

void menu()

{

cout << "1 - Добавить элемент." << endl;

cout << "2 - Вывод дерева" << endl;

cout << "3 - Удалить элемент" << endl;

cout << "4 - Заменить отрицательные элементы положительными" << endl;

cout << "5 - Найти минимальный элемент дерева" << endl;

cout << "6 - Выход." << endl;

}

int main()

{

setlocale(LC\_ALL, "Russian");

Tree Universal;

int c, value, space = 0;

do {

menu();

cout << endl;

cin >> c;

switch (c)

{

case 1:

cout << "Введите число: ";

cin >> value;

Universal.addToTreeHelper(value);

Universal.levelOrder();

break;

case 2:

cout << "Вывод дерева:\n\n";

Universal.outputHelper(space);

Universal.levelOrder();

break;

case 3:

cout << "Введите значение удаляемого элемента: ";

cin >> value;

Universal.deleteTreeHelp(value);

Universal.levelOrder();

break;

case 4:

cout << "Замена отрицательных элементов дерева произведена." << endl;

Universal.outputHelperPlus(space);

Universal.levelOrder();

cout << endl;

break;

case 5:

cout << "Наименьший элемент дерева: ";

Universal.MinValueHelper();

cout << endl;

break;

case 6:

exit(0);

default: cout << "Неверный пункт меню!" << endl;

}

system("Pause");

system("cls");

} while (c != 6);

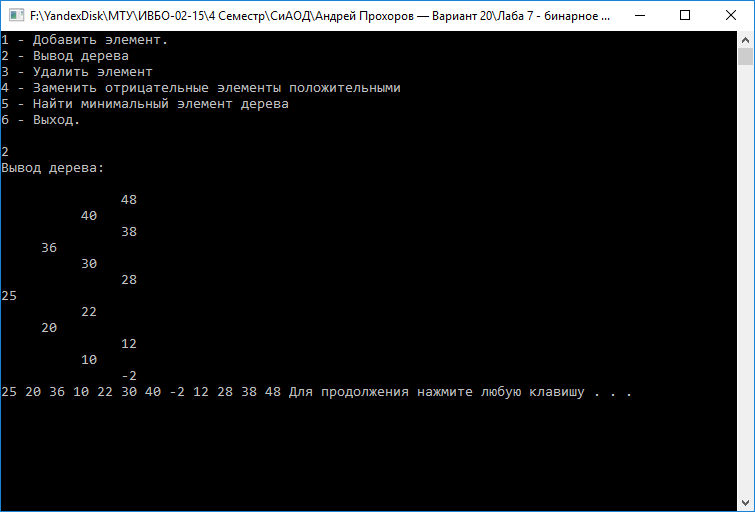
return 0;

}

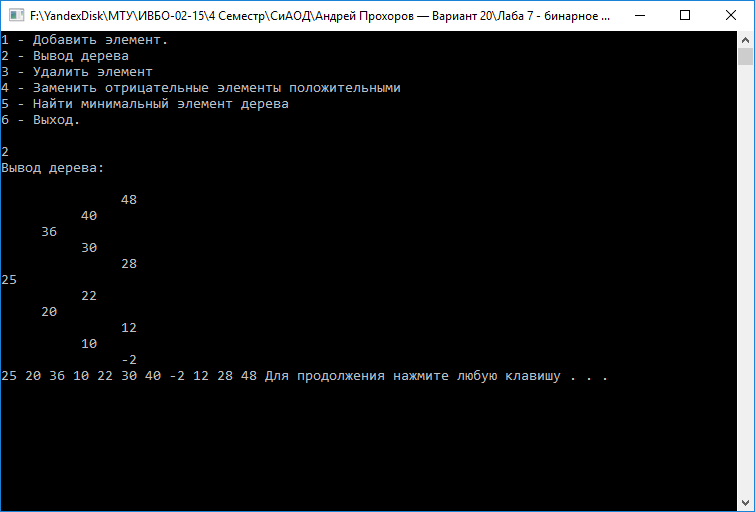
**4. Тестирование программы**

**Контрольные прогоны программы**

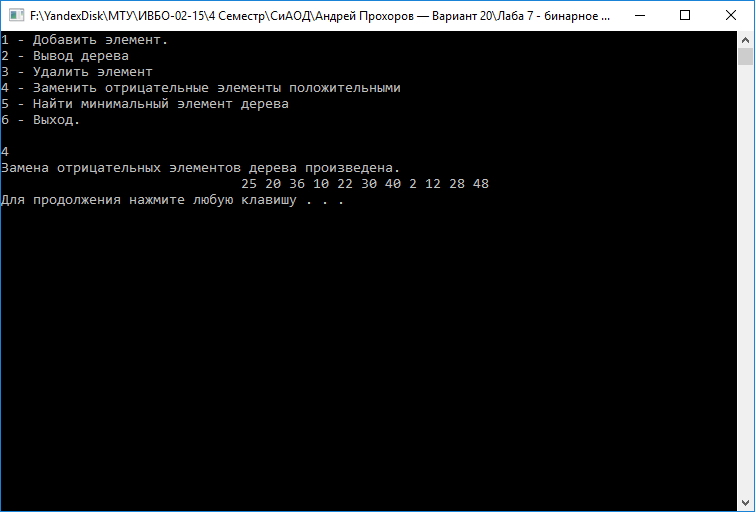
Тест №1—ввод и вывод сбалансированного бинарного дерева



Тест №2—удаление элемента 38



Тест №3—замена отрицательных элементов (-2) его модулем



Тест №4—вывод наименьший элемент дерева

