Практическая работа № 7

АВЛ - деревья

Вариант-22Ф

Постановка задачи

Составить программу создания двоичного дерева поиска и реализовать процедуры для работы с деревом согласно варианту.

Процедуры оформить в виде самостоятельных режимов работы созданного дерева. Выбор режимов производить с помощью пользовательского (иерархического ниспадающего) меню.

Провести полное тестирование программы на дереве размером n=10 элементов, сформированном вводом с клавиатуры. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.

Сделать выводы о проделанной работе, основанные на полученных результатах.

Оформить отчет с подробным описанием созданного дерева, принципов программной реализации алгоритмов работы с деревом, описанием текста исходного кода и проведенного тестирования программы.

1. Описание алгоритма

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

Относительно АВЛ-дерева балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев = 2, изменяет связи предок-потомок в поддереве данной вершины так, что разница становится <= 1, иначе ничего не меняет. Указанный результат получается вращениями поддерева данной вершины.

Используются 4 типа вращений:

Малое левое вращение

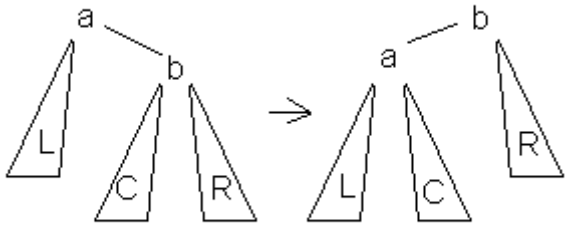

Рис.1 Малое левое вращение

Данное вращение используется тогда, когда (высота b-поддерева — высота L) = 2 и высота c-поддерева <= высота R.

Большое левое вращение

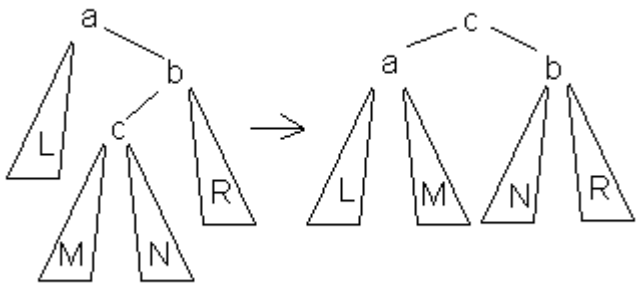

Рис.2 Большое левое вращение

Данное вращение используется тогда, когда (высота b-поддерева — высота L) = 2 и высота c-поддерева > высота R.

Малое правое вращение

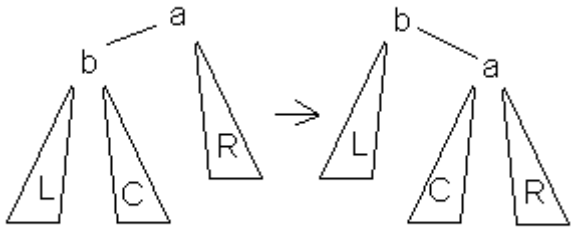

Рис.3 Малое правое вращение

Данное вращение используется тогда, когда (высота b-поддерева — высота R) = 2 и высота C <= высота L.

Большое правое вращение

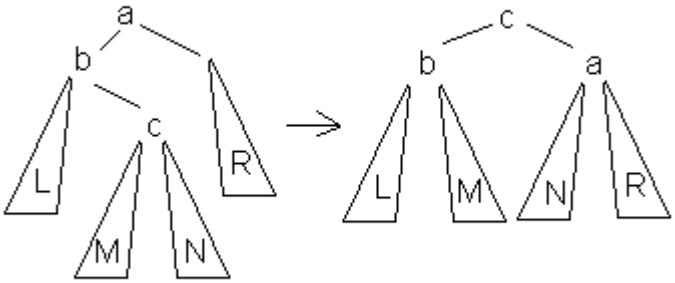

Рис.4 Большое правое вращение

Данное вращение используется тогда, когда (высота b-поддерева — высота R) = 2 и высота c-поддерева > высота L.

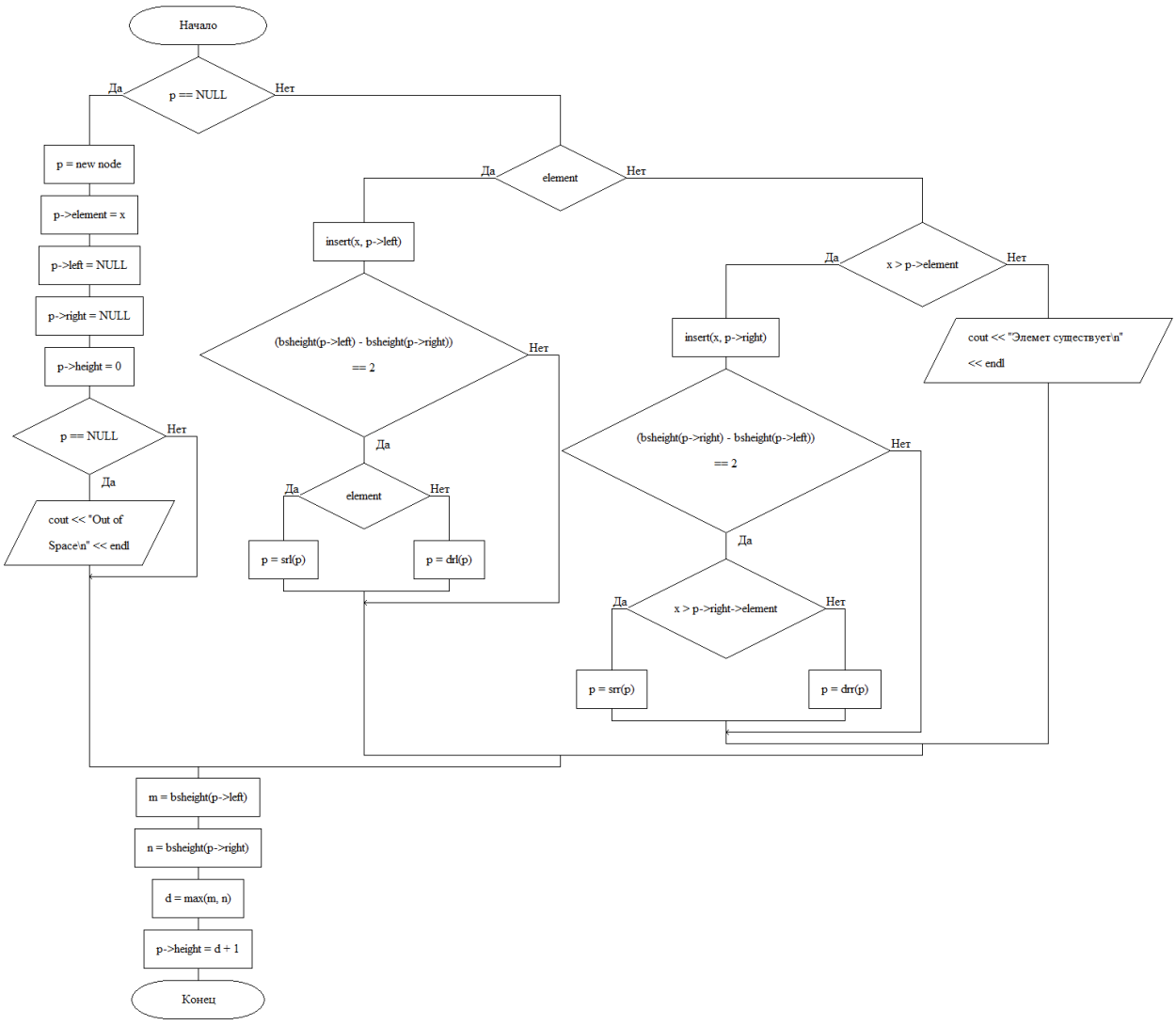

Рис.5 Схема алгоритма функции insert

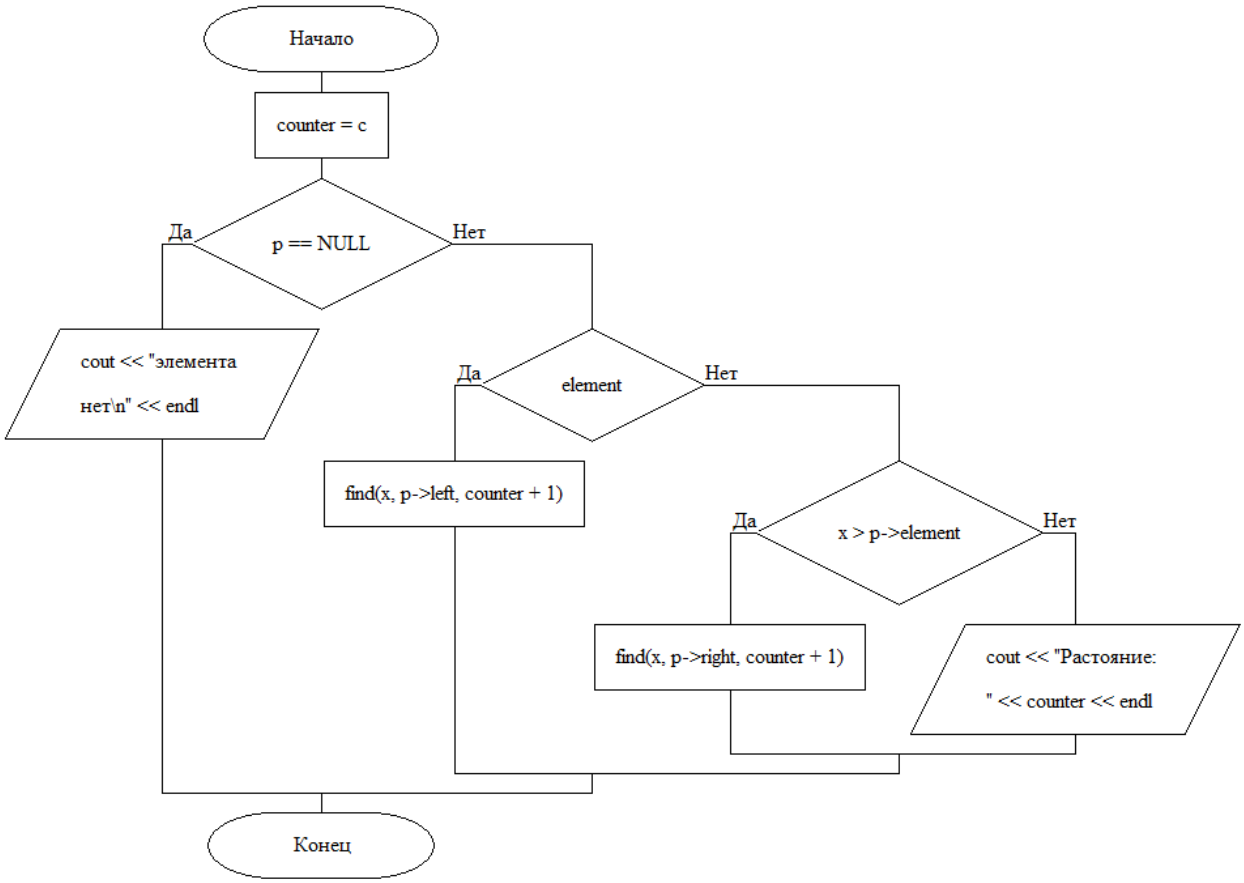

Рис.6 Схема алгоритма функции find

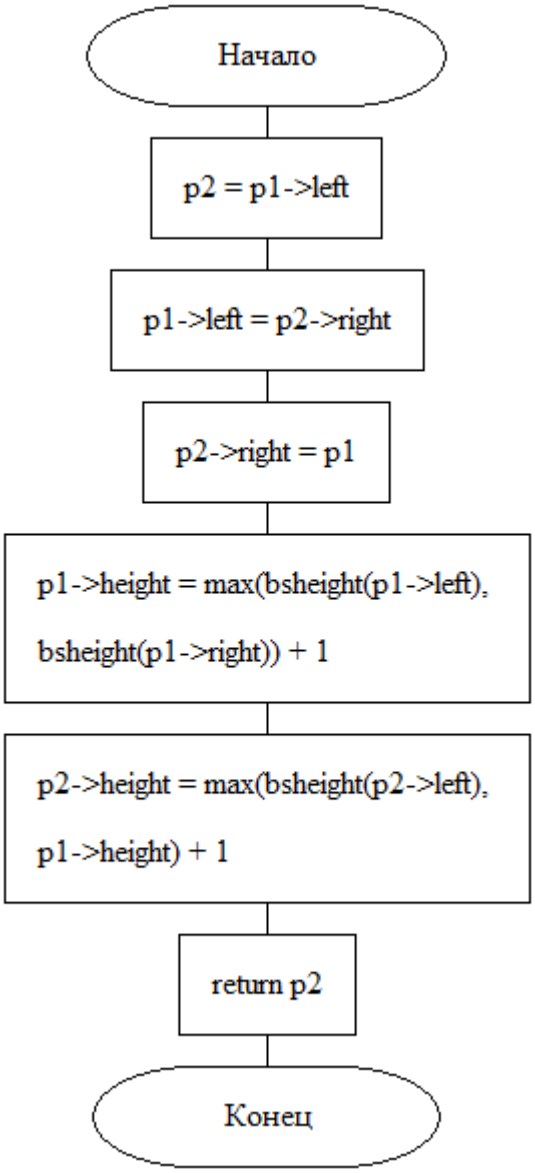

Рис.7 Схема алгоритма функции srl

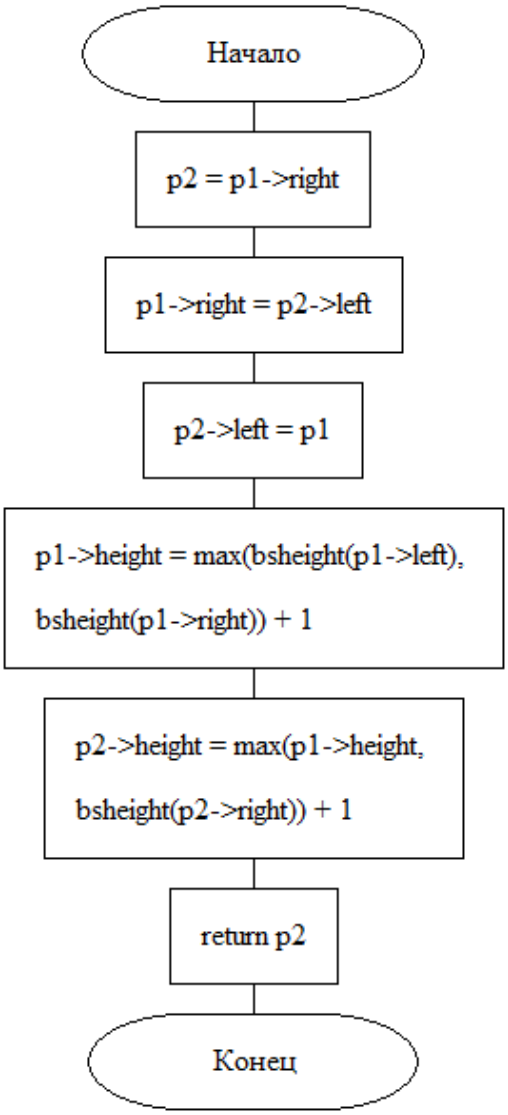

Рис.8 Схема алгоритма функции srr

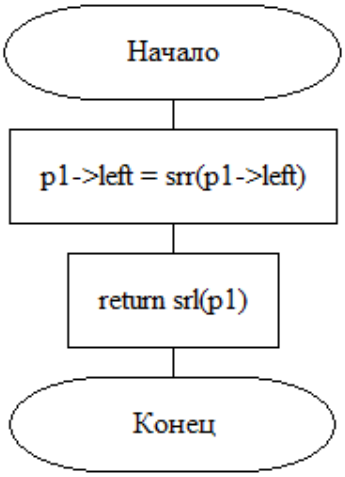

Рис.9 Схема алгоритма функции drl

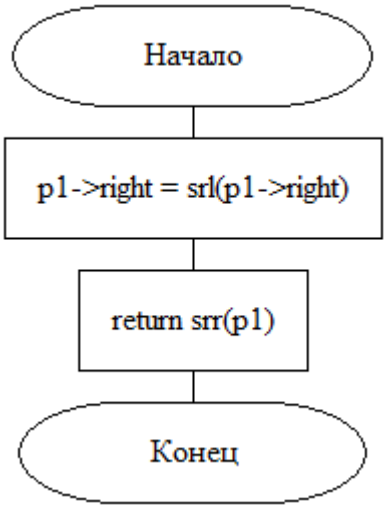

Рис.10 Схема алгоритма функции drr

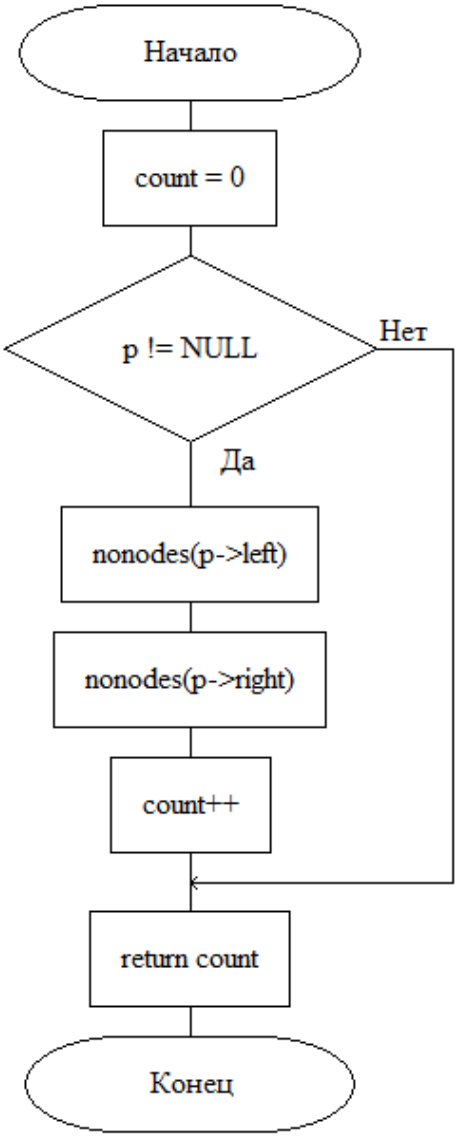

Рис.11 Схема алгоритма функции nonodes

Реализация алгоритма

Текст исходного кода программы

main.cpp

```
#include <iostream>
#include<ctype.h>
#include <stdlib.h>
#include <conio.h>

using namespace std;

struct node
{
	int element;
	node* left;
	node* right;
	int height;
```

```
};

typedef struct node* nodeptr;

class bstree
{
public:
	void insert(int, nodeptr&);
	void find(int, nodeptr&m, int);
	void makeempty(nodeptr&);
	void copy(nodeptr&, nodeptr&);
	nodeptr nodecopy(nodeptr&);
	void preorder(nodeptr);
	int bsheight(nodeptr);
	nodeptr srl(nodeptr&);
	nodeptr drl(nodeptr&);
	nodeptr srr(nodeptr&);
	nodeptr drr(nodeptr&);
	int max(int, int);
	int nonodes(nodeptr);
};

void bstree::insert(int x, nodeptr& p)
{
	if (p == NULL)
	{
		p = new node;
		p->element = x;
		p->left = NULL;
		p->right = NULL;
		p->height = 0;
		if (p == NULL)
		{
			cout << "Out of Space\n" << endl;
		}
	}
	else
	{
		if (x < p->element)
		{
			insert(x, p->left);
			if ((bsheight(p->left) - bsheight(p->right)) == 2)
			{
				if (x < p->left->element)
				{
					p = srl(p);
				}
				else
				{
					p = drl(p);
				}
			}
		}
		else if (x > p->element)
		{
			insert(x, p->right);
			if ((bsheight(p->right) - bsheight(p->left)) == 2)
			{
				if (x > p->right->element)
				{
					p = srr(p);
				}
				else
				{
					p = drr(p);
```

```
                        }
                    }
                }
                else
                {
                    cout << "Элемет существует\n" << endl;
                }
        }
        int m, n, d;
        m = bsheight(p->left);
        n = bsheight(p->right);
        d = max(m, n);
        p->height = d + 1;
}

void bstree::find(int x, nodeptr& p, int c)
{
        int counter = c;
        if (p == NULL)
        {
                cout << "элемента нет\n" << endl;
        }
        else
        {
                if (x < p->element)
                {
                    find(x, p->left, counter+1);
                }
                else
                {
                    if (x > p->element)
                    {
                        find(x, p->right, counter + 1);
                    }
                    else
                    {
                        cout << "Растояние: "<<counter<< endl;
                    }
                }
        }
}

void bstree::copy(nodeptr& p, nodeptr& p1)
{
        makeempty(p1);
        p1 = nodecopy(p);
}

void bstree::makeempty(nodeptr& p)
{
        nodeptr d;
        if (p != NULL)
        {
                makeempty(p->left);
                makeempty(p->right);
                d = p;
                free(d);
                p = NULL;
        }
}
nodeptr bstree::nodecopy(nodeptr& p)
{
        nodeptr temp;
        if (p == NULL)
        {
```

```
                return p;
        }
        else
        {
                temp = new node;
                temp->element = p->element;
                temp->left = nodecopy(p->left);
                temp->right = nodecopy(p->right);
                return temp;
        }
}


void bstree::preorder(nodeptr p)
{
        if (p != NULL)
        {
                cout << p->element << "\t";
                preorder(p->left);
                preorder(p->right);
        }
}


int bstree::max(int value1, int value2)
{
        return ((value1 > value2) ? value1 : value2);
}
int bstree::bsheight(nodeptr p)
{
        int t;
        if (p == NULL)
        {
                return -1;
        }
        else
        {
                t = p->height;
                return t;
        }
}

nodeptr bstree::srl(nodeptr& p1)
{
        nodeptr p2;
        p2 = p1->left;
        p1->left = p2->right;
        p2->right = p1;
        p1->height = max(bsheight(p1->left), bsheight(p1->right)) + 1;
        p2->height = max(bsheight(p2->left), p1->height) + 1;
        return p2;
}
nodeptr bstree::srr(nodeptr& p1)
{
        nodeptr p2;
        p2 = p1->right;
        p1->right = p2->left;
        p2->left = p1;
        p1->height = max(bsheight(p1->left), bsheight(p1->right)) + 1;
        p2->height = max(p1->height, bsheight(p2->right)) + 1;
        return p2;
}
nodeptr bstree::drl(nodeptr& p1)
{
        p1->left = srr(p1->left);
```

```
	return srl(p1);
}
nodeptr bstree::drr(nodeptr& p1)
{
	p1->right = srl(p1->right);
	return srr(p1);
}

int bstree::nonodes(nodeptr p)
{
	int count = 0;
	if (p != NULL)
	{
		nonodes(p->left);
		nonodes(p->right);
		count++;
	}
	return count;
}

int main()
{
	setlocale(LC_ALL, "Russian");
	nodeptr root, root1, min, max;
	int a, choice, findele, delele;
	char ch = 'y';
	bstree bst;

	root = NULL;
	root1 = NULL;

	do
	{
		cout << "[1] - Вставить новый узел" << endl;
		cout << "[2] - Растояние от головного объекта до искомого" << endl;
		cout << "[3] - Показать высоту дерева" << endl;
		cout << "[4] - Прямой обход дерева" << endl;
		cout << "[5] - Выход" << endl;
		cin >> choice;

		switch (choice)
		{
		case 1:
			cout << "Добавление нового узла" << endl;
			cout << "Введите элемент: ";
			cin >> a;
			bst.insert(a, root);
			cout << "Новый элемент добавлен успешно" << endl;
			break;
		case 2:
			cout << "Введите искомый элемент: ";
			cin >> findele;
			if (root != NULL)
			{
				bst.find(findele, root, 0);
			}
			break;
		case 4:
			cout << "Вывод дерева:" << endl;
			bst.preorder(root);
			cout << endl;
			break;
		case 3:
			cout << "Дерево имеет высоту: " << bst.bsheight(root)+1 << endl;
```

```
                        break;
                case 5:
                        cout << "Программа завершена" << endl;
                        break;
                default:

                        break;
                }
        } while (choice != 0);
        return 0;
}
```

2. Тестирование программы

Ниже представлен результат работы программы

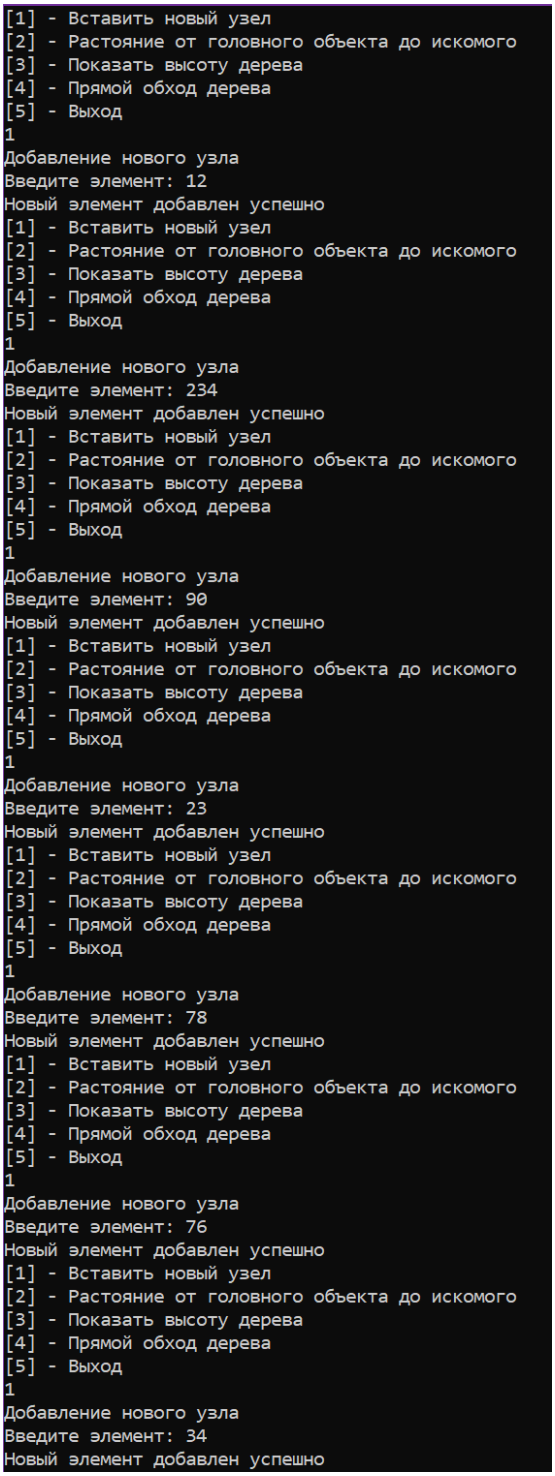

Рис.12 Скриншот добавления элементов в дерево

```
[1] - Вставить новый узел
[2] - Растояние от головного объекта до искомого
[3] - Показать высоту дерева
[4] - Прямой обход дерева
[5] - Выход
3
Дерево имеет высоту: 4
```

Рис.13 Скриншот вывода высоты дерева

```
[1] - Вставить новый узел
[2] - Растояние от головного объекта до искомого
[3] - Показать высоту дерева
[4] - Прямой обход дерева
[5] - Выход
2
Введите искомый элемент: 90
Растояние: 1
```

Рис.14 Скриншот вывода расстояния от головного объекта

```
[1] - Вставить новый узел
[2] - Растояние от головного объекта до искомого
[3] - Показать высоту дерева
[4] - Прямой обход дерева
[5] - Выход
4
Вывод дерева:
78      23      12      76      34      90      234
```

Рис.15 Скриншот вывода дерева

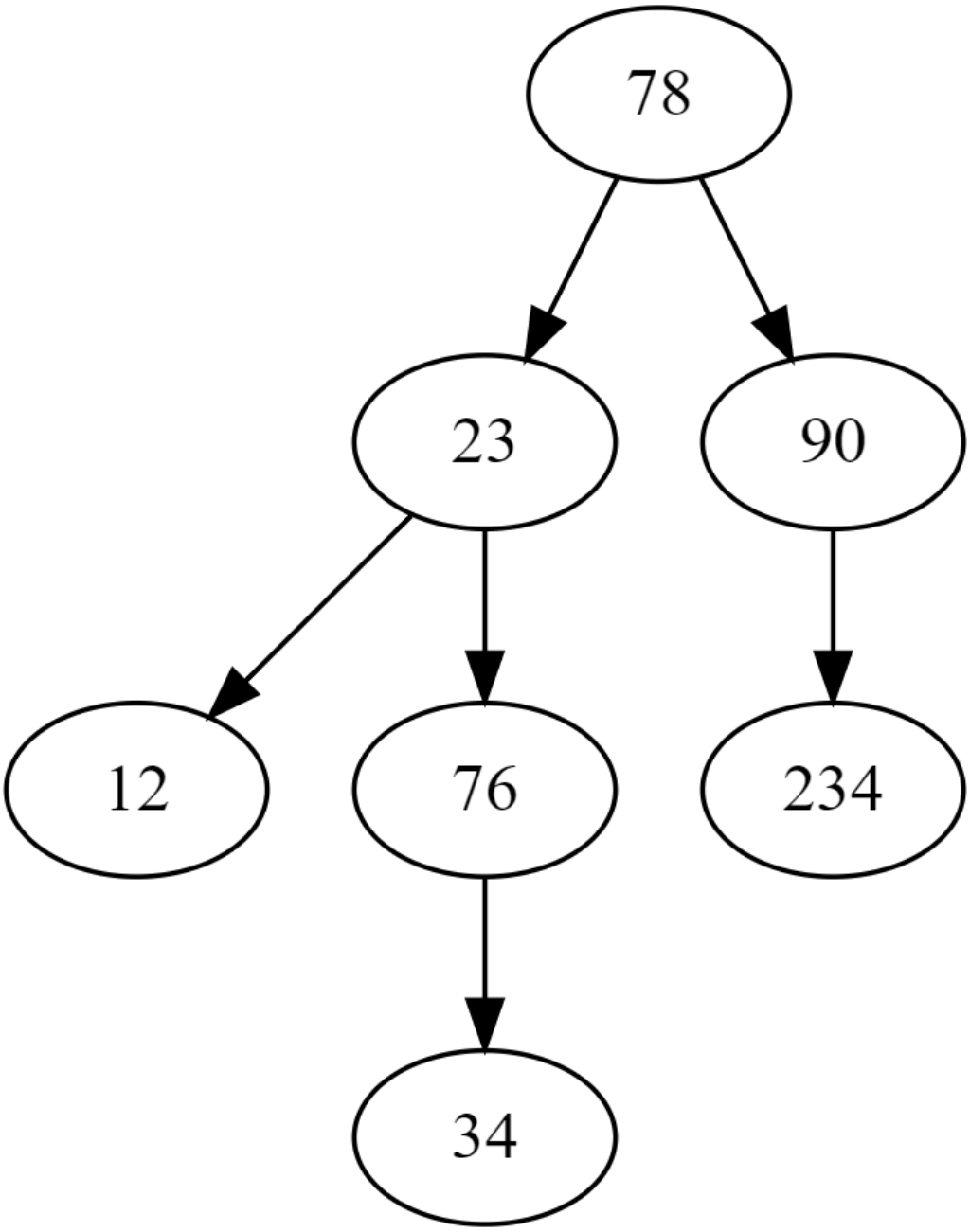

Рис.15 Скриншот визуального вывода дерева после балансировки

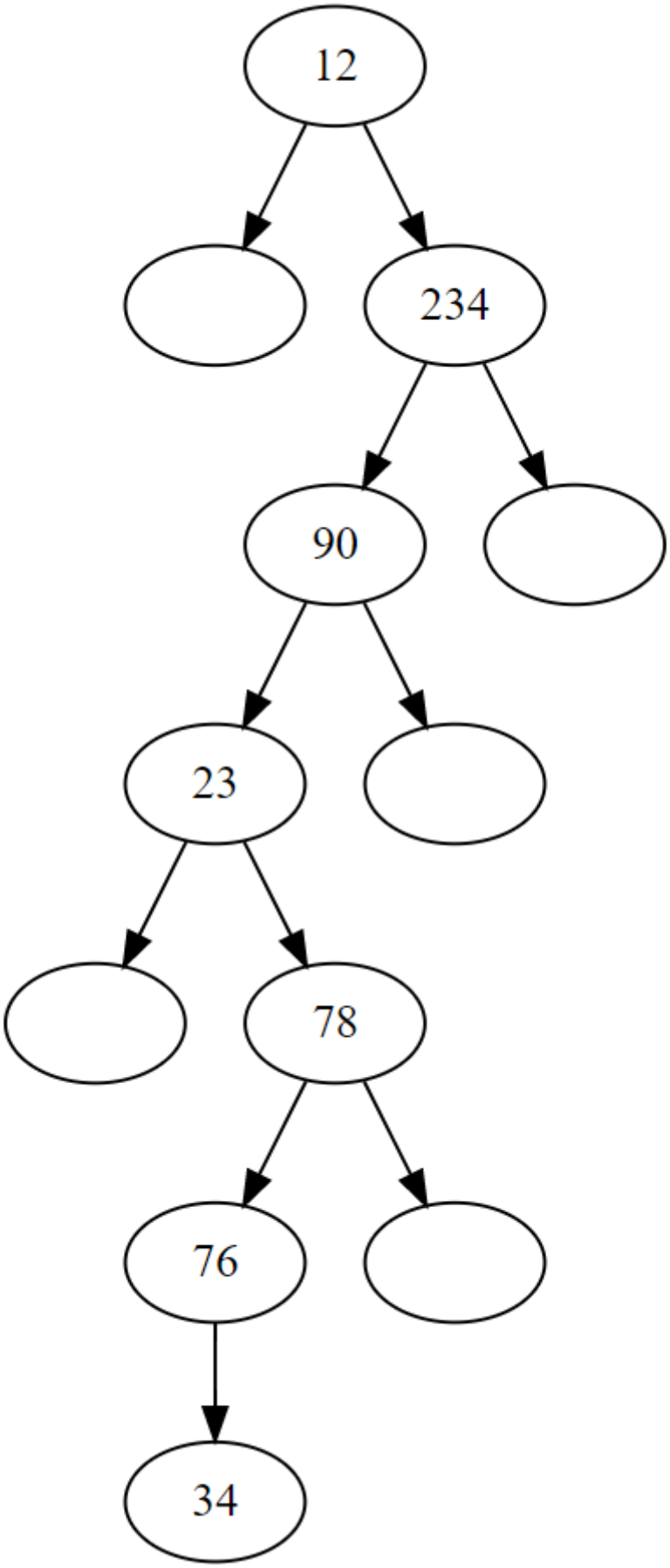

Рис.15 Скриншот визуального вывода дерева до балансировки

Выводы

1. В ходе работы была создана программа для работы с AVL-деревьями.
2. Также были реализованы функции добавления, поиска элементов, а также вычисления высоты.
3. Были изучены преимущества и недостатки хранения данных в AVL - деревьях:
4. Преимущества: поиск в сбалансированном AVL – дереве имеет временную сложность log(n), что является довольно эфективным по сравнению с аналогами.
5. Недостатки: не смотря на довольно низкую временную сложность поиска главным недостатком AVL-дерева является необходимость постоянной автобалансировки, что требует некоторых затрат по времени. Также поиск в AVL – дереве сильно уступает ХЭШ – таблице по скорости.
6. Таким образом, была изучена работа AVL - деревьев

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 30.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 30.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 30.11.2020).
5. Описание AVL - деревьев. интернет-ресурс: https://ru.wikipedia.org/wiki/АВЛ-дерево (Дата обращения – 30.11.2020).
6. Построение графов по DOT-нотации. интернет-ресурс: https://dreampuf.github.io/GraphvizOnline (Дата обращения – 30.11.2020).