МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: АВЛ-деревья: демонстрация вставки и удаления элементов

Студент гр. 9381 _____ Аухадиев А.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Аухадиев А.А.

Группа 9381

Тема работы: АВЛ-деревья: демонстрация вставки и удаления элементов

Исходные данные:

Компилятор g++

Данные для программы вводятся пользователем в консоли

Содержание пояснительной записки:

"Содержание", "Введение", "Ход выполнения работы", "Заключение", "Список использованных источников"

Предполагаемый объем пояснительной записки:

Не менее 12 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 15.12.2020

Дата защиты реферата:

Студент _____ Аухадиев А.А.

Преподаватель _____ Фирсов М.А.

АННОТАЦИЯ

В курсовой работе представлена реализация интерфейса для АВЛ-дерева в консольном приложении на C++. Пользователь может добавлять и удалять элементы дерева. Программа производит все действия пошагово с пояснениями и выводом дерева после каждого шага на экран.

SUMMARY

The course work presents the implementation of the interface for the AVL tree in a console application. The user can add and remove tree items. The program performs all actions step by step with explanations and displaying a tree after each step on the screen.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель: реализация алгоритмов вставки и удаления элементов АВЛ-дерева и его представление в консоли.

Задачи: Изучение структуры АВЛ-деревьев и основных алгоритмов работы с ними.

ХОД ВЫПОЛНЕНИЯ РАБОТЫ

1. Структура данных и функции работы с ней

1.1. Структура данных

АВЛ-дерево - бинарное дерево поиска, для которого выполняются два требования:

1. Для каждой вершины все вершины из ее левого поддерева меньше по величине, чем ее значение, а все вершины из правого поддерева - больше.

2. Разность высот левого и правого поддерева текущей вершины должно находится в диапазоне от -1 до 1. При нарушении требования происходит перебалансировка вершин, то есть правые и левые повороты.

class Node - класс, описывающий основные свойства узла АВЛ-дерева.
Поля класса:

1. int key_ - значение ключа узла

2. unsigned char height_ - высота узла

3. Node* left_ - левый сын

4. Node* right_ - правый сын

5. int count_ - количество узлов с таким же значением ключа

Конструктор:

Node(int key): key_(key), height_(1), left_(nullptr), right_(nullptr), count_(1) {};

1.2. Функции для работы с АВЛ-деревьями

1. void printTree(Node* root, int n, int flagRight = 0) - функция вывода АВЛ-дерева на экран, "лежащим на левом боку". Принимает указатель на АВЛ-дерево, количество табуляций и флаг, указывающий место для вывода ребра между узлами.

2. int height(Node* N) - возвращает значение высоты дерева N.

3. int max(int a, int b) - вспомогательная функция, возвращающая большее из чисел a и b.

4. Node* rightRotate(Node* N) - функция правого поворота дерева N, возвращает дерево после правого поворота. (Описание алгоритма см. ниже)

5. Node* leftRotate(Node* N) - функция левого поворота дерева N, возвращает дерево после левого поворота. (Описание алгоритма см. ниже)

6. int getBalance(Node* N) - возвращает баланс узла N (баланс - разница между высотами левого и правого сыновей.

7. Node* insert(Node* node, int key) - функция вставки элемента с ключом key в дерево node, возвращает указатель на дерево с добавленным элементом. (Описание алгоритма см. ниже)

8. Node* minValueNode(Node* node) - возвращает узел с минимальным значением ключа (самый левый лист) дерева node.

9. Node* deleteNode(Node* root, int key) - функция удаления элемента из дерева root по ключу key, возвращает дерево с удалённым элементом. (Описание алгоритма см. ниже)

10. void clearTree(Node* root) - рекурсивное высвобождение выделенной памяти под узлы дерева root.

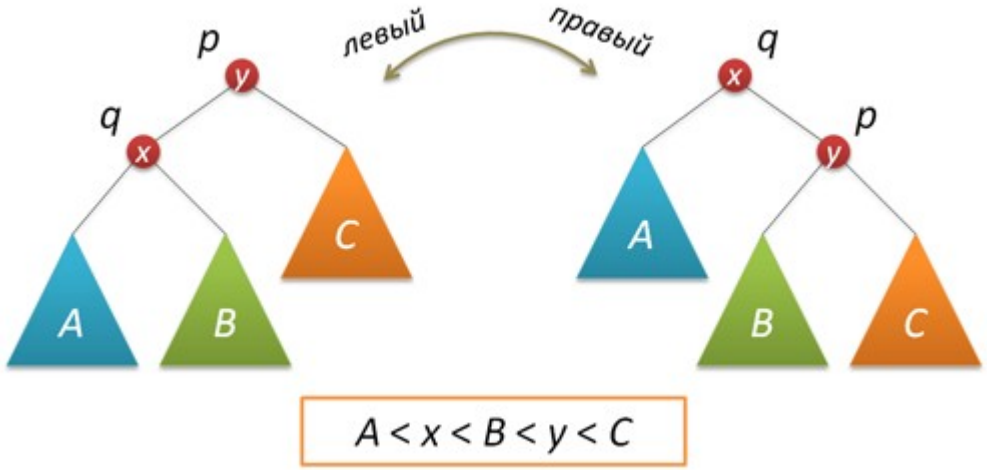
11. bool isDigit(std::string key) - проверка строки key, если она является числом, возвращает true, иначе - false.

2. Описание алгоритмов

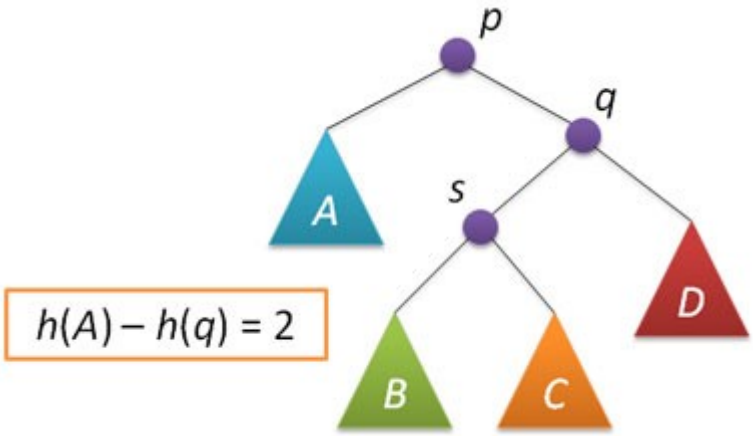
2.1. Алгоритм поворота дерева.

В процессе добавления или удаления узлов в АВЛ-дереве возможно возникновение ситуации, когда баланс некоторых узлов (разница между высотами левого и правого сыновей) оказывается равными 2 или -2, т.е. возникает расбалансировка поддерева. Для выправления ситуации применяются повороты вокруг тех или иных узлов дерева.

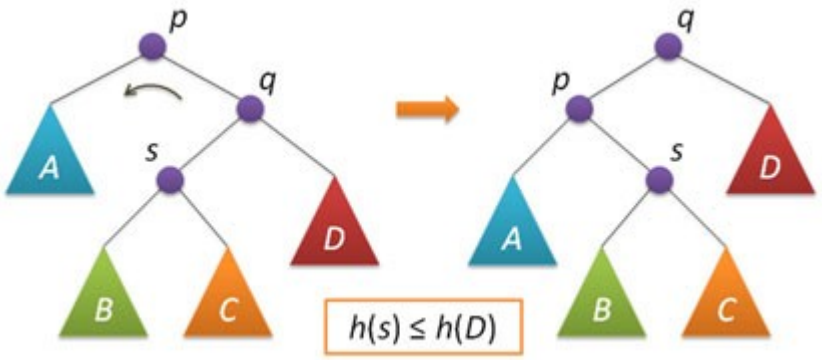
Простой поворот вправо (влево) производит следующую трансформацию дерева:



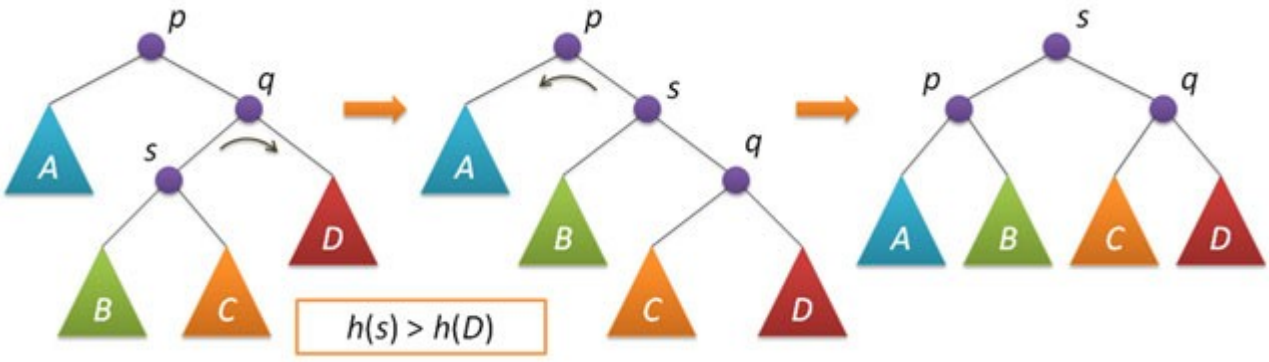
Существует ситуация дисбаланса, когда высота правого поддерева узла p на 2 больше высоты левого поддерева (обратный случай является симметричным и реализуется аналогично). Пусть q — правый дочерний узел узла p, а s — левый дочерний узел узла q.



Для исправления расбалансировки в узле p достаточно выполнить либо простой поворот влево вокруг p, либо так называемый большой поворот влево вокруг того же p. Простой поворот выполняется при условии, что высота левого поддерева узла q больше высоты его правого поддерева: h(s)≤h(D).



Большой поворот применяется при условии h(s)>h(D) и сводится в данном случае к двум простым — сначала правый поворот вокруг q и затем левый вокруг p.



Описанные алгоритмы поворотов и балансировки не содержат ни циклов, ни рекурсии, а значит выполняются за постоянное время, не зависящее от размера АВЛ-дерева.

В коде алгоритмы поворота вправо и влево осуществляют функции rightRotate() и leftRotate() (Описание функций см. выше).

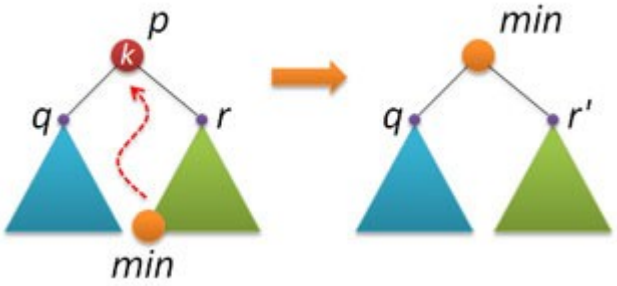
2.2. Алгоритм добавления элемента.

Вставка нового ключа в АВЛ-дерево выполняется так же, как это делается в простых деревьях поиска: осуществляется проход вниз по дереву, выбираются правое или левое направление движения в зависимости от результата сравнения ключа в текущем узле и вставляемого ключа. Единственное отличие заключается в том, что при возвращении из рекурсии (т.е. после того, как ключ вставлен либо в правое, либо в левое поддерево, и это дерево сбалансировано) выполняется балансировка текущего узла.

В коде алгоритм добавления элемента в дерево осуществляется с помощью функции insert() (Описание функции см. выше).

2.3. Алгоритм удаления элемента.

При удалении элемента происходит поиск узла p с заданным ключом k (если такого ключа нет, удаления не происходит). После чего в правом поддереве происходит поиск узла min с наименьшим ключом и удаляемый узел p заменяется на найденный узел min.

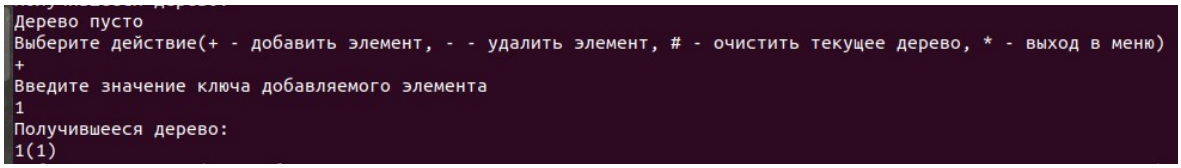


При реализации возникает несколько нюансов. Прежде всего, если найденный узел p не имеет правого поддерева, то по свойству АВЛ-дерева слева у этого узла может быть только один единственный дочерний узел (дерево высоты 1), либо узел p вообще лист. В обоих этих случаях узел p удаляется и возвращается в качестве результата указатель на левый дочерний узел узла p.

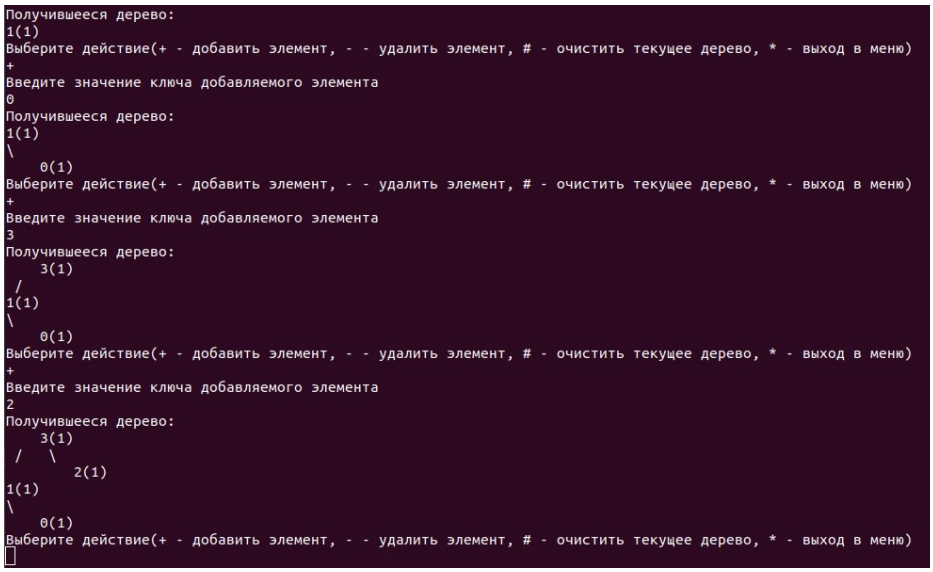
В коде алгоритм удаления узла выполняет функция deleteNode() с использованием функции minValueNode() для получения минимального узла в левом поддереве удаляемого узла. (Описание функций см. выше).

ТЕСТРИРОВАНИЕ

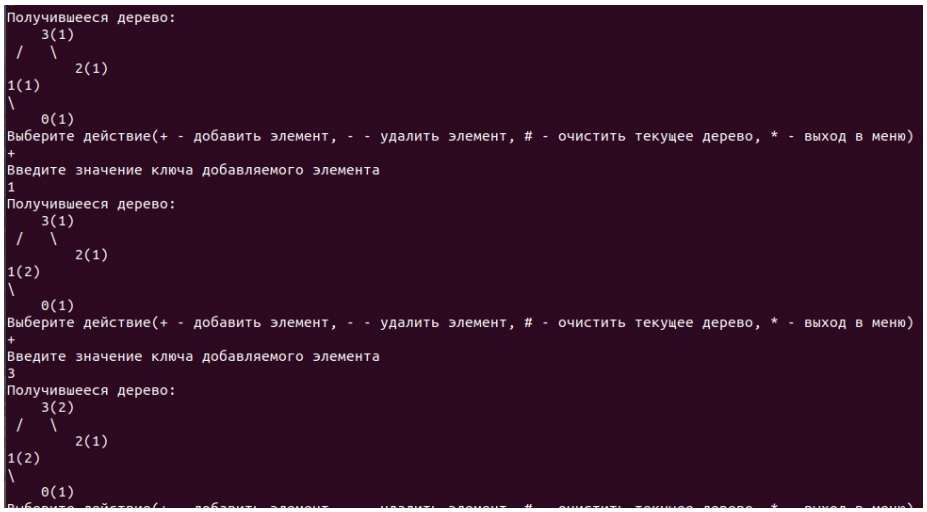
1. Создание дерева (добавление первого элемента).



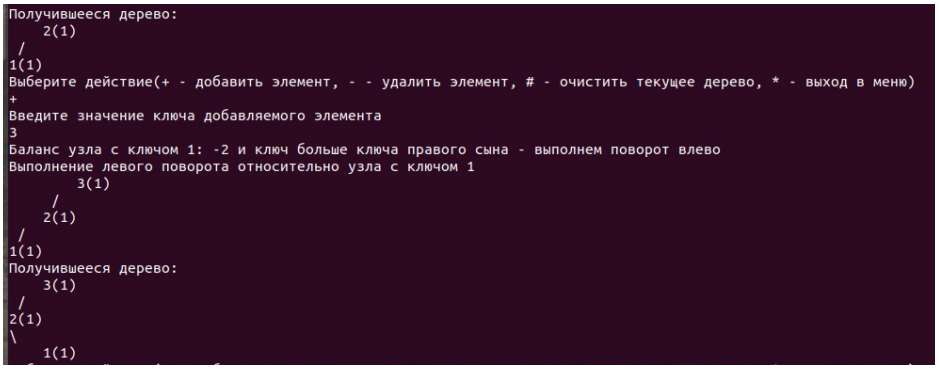
2. Добавление нескольких узлов.



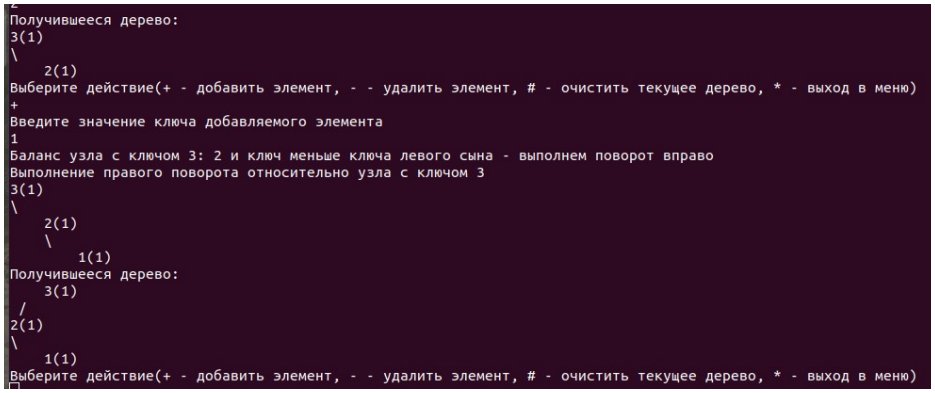
3. Добавление существующих узлов.



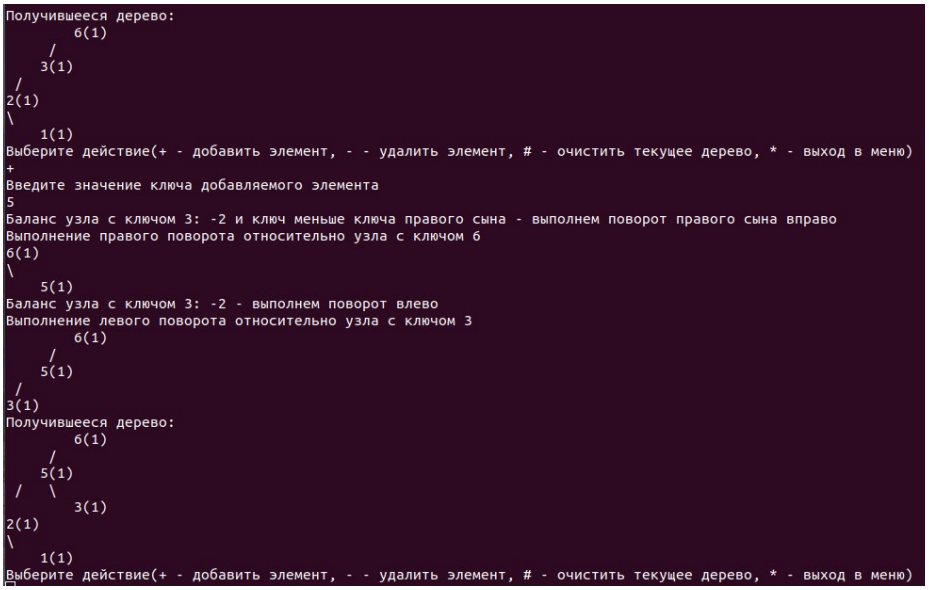
4. Добавление ключа с перебалансировкой (случай RR).



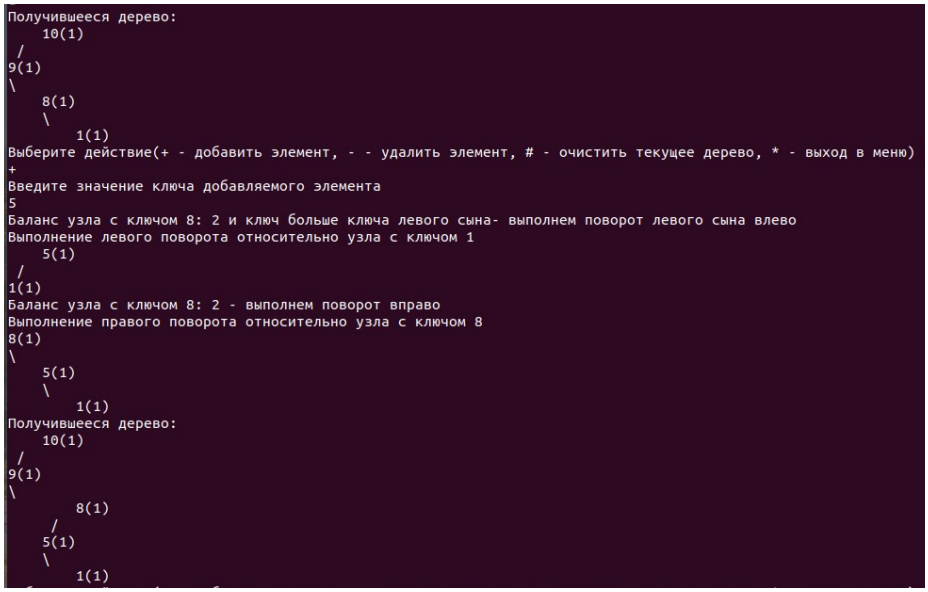
5. Добавление ключа с перебалансировкой (случай LL).



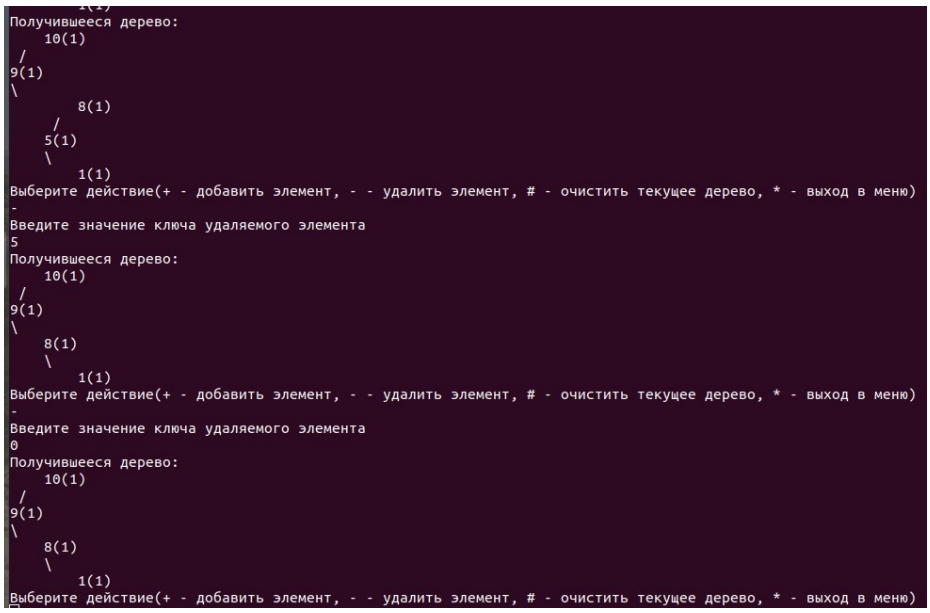
6. Добавление ключа с перебалансировкой (случай RL).



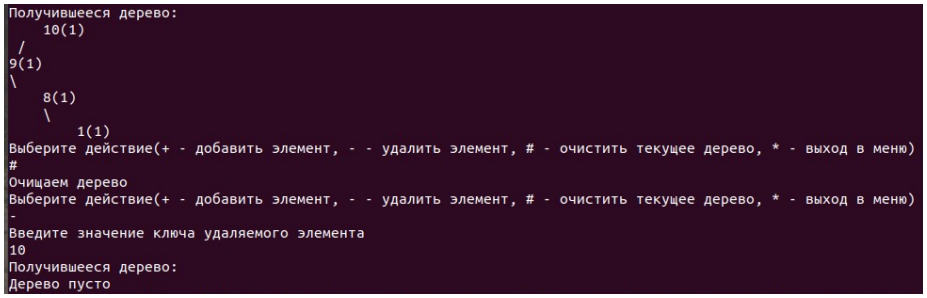
7. Добавление ключа с перебалансировкой (случай LR).



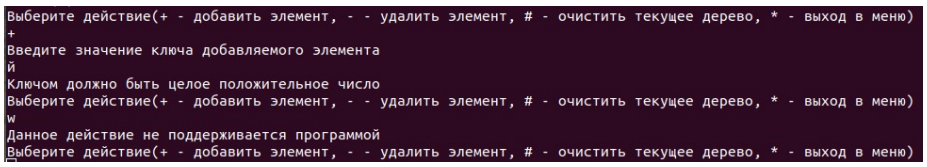
8. Удаление существующего и несуществующего элементов.



9. Удаление всего дерева и попытка удалить элемент пустого дерева.



10.Ввод непредусмотренных команд и букв вместо чисел.



ЗАКЛЮЧЕНИЕ

В ходе работы была изучена структура данных АВЛ-дерево, освоены алгоритмы работы с этой структурой данных, написана программа на языке программирования C++, способная добавлять и удалять элементы АВЛ-дерева и выводить результат в консоли.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Описание АВЛ-дерева и алгоритмов работы с ним // URL: https://habr.com/ru/post/150732/

2. Описание AVL-дерева с повторяющимися ключами и алгоритмов работы с ним // URL: http://espressocode.top/avl-with-duplicate-keys/

3. Описание AVL-дерева // URL: https://ru.wikipedia.org/wiki/%D0%90%D0%92%D0%9BD0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

ФАЙЛ: MAIN.CPP

```
#include <iostream>
#include <cctype>
#include <string>
#include <fstream>

class Node {
public:
   int key_;       //Значение в узле
   unsigned char height_; //высота узла
   Node* left_;         //левый сын
   Node* right_;        //правый сын
   int count_;             //Количество узлов с одинаковым ключом
   Node(int key): key_(key), height_(1), left_(nullptr), right_(nullptr), count_(1){};
};

void printTree(Node* root, int n, int flagRight = 0){   //Вывод дерева, "лежащего на левом боку"
   if(!root){
      std::cout << "Дерево пусто\n";
      return;
   }
   if(root->right_)
      printTree(root->right_, n+4, 1);
   for(auto i = 0; i<n; i++)
      std::cout << " ";
   std::cout << root->key_ << '(' << root->count_ << ")\n";
   if(flagRight || root->left_) {
      for (auto i = 0; i < n + 1; i++) {
         if (i == n - 3 && flagRight)
            std::cout << '/';
         if (i == n && root->left_)
            std::cout << "\\";
         std::cout << ' ';
      }
      std::cout << '\n';
```

```
    }
    if(root->left_)
        printTree(root->left_, n+4);
}

int height(Node* N){ //получение высоты дерева
    if (!N)
        return 0;

    return N->height_;
}

int max(int a, int b){ //Получение максимума из двух чисел
    return (a > b) ? a : b;
}

Node* rightRotate(Node* N) //Правый поворот
{
    std::cout << "Выполнение правого поворота относительно узла с ключом " << N->key_ << '\n';
    printTree(N, 0);
    Node* x = N->left_;
    Node* y = x->right_;

    // Выполнить вращение
    x->right_ = N;
    N->left_ = y;

    // Обновление высоты
    N->height_ = max(height(N->left_), height(N->right_)) + 1;
    x->height_ = max(height(x->left_), height(x->right_)) + 1;

    // Возвращаем новый корень
    return x;
}

Node* leftRotate(Node* N) //Левый поворот
{
    std::cout << "Выполнение левого поворота относительно узла с ключом " << N->key_ << "\n";
```

```
    printTree(N, 0);
    Node* x = N->right_;
    Node* y = x->left_;

    // Выполнить вращение
    x->left_ = N;
    N->right_ = y;

    // Обновление высоты
    N->height_ = max(height(N->left_), height(N->right_)) + 1;
    x->height_ = max(height(x->left_), height(x->right_)) + 1;

    return x;
}

int getBalance(Node* N) //Баланс узла
{
    if (!N)
        return 0;

    return height(N->left_) - height(N->right_);
}

Node* insert(Node* node, int key){

    if (!node)
        return (new Node(key));

    // Если ключ уже существует, увеличить счетчик и вернуть
    if (key == node->key_) {
        (node->count_)++;
        return node;
    }

    /* В противном случае вернемся вниз по дереву */
    if (key < node->key_)
        node->left_ = insert(node->left_, key);
    else
```

```
    node->right_ = insert(node->right_, key);

/* 2. Обновить высоту этого узла-предка */
node->height_ = max(height(node->left_), height(node->right_)) + 1;

/* 3. Получить коэффициент баланса этого узла-предка, чтобы проверить уравновешенность*/
int balance = getBalance(node);

// Если этот узел становится несбалансированным, то есть 4 случая
// Левый левый регистр
if (balance > 1 && key < node->left_->key_) {
    std::cout << "Баланс узла с ключом " << node->key_ << ": " << balance
              << " и ключ меньше ключа левого сына - выполнем поворот вправо\n";
    return rightRotate(node);
}

// правый правый случай
if (balance < -1 && key > node->right_->key_) {
    std::cout << "Баланс узла с ключом " << node->key_ << ": " << balance
              << " и ключ больше ключа правого сына - выполнем поворот влево\n";
    return leftRotate(node);
}

// левый правый регистр
if (balance > 1 && key > node->left_->key_) {
    std::cout << "Баланс узла с ключом " << node->key_ << ": " << balance <<
              " и ключ больше ключа левого сына- выполнем поворот левого сына влево\n";
    node->left_ = leftRotate(node->left_);
    std::cout << "Баланс узла с ключом " << node->key_ << ": " << balance <<
              " - выполнем поворот вправо\n";
    return rightRotate(node);
}

// Правый левый регистр
if (balance < -1 && key < node->right_->key_) {
    std::cout << "Баланс узла с ключом " << node->key_ << ": " << balance <<
              " и ключ меньше ключа правого сына - выполнем поворот правого сына вправо\n";
    node->right_ = rightRotate(node->right_);
```

```
        std::cout << "Баланс узла с ключом " << node->key_ << ": " << balance <<
                " - выполнем поворот влево\n";
        return leftRotate(node);
    }

    /* вернуть (неизмененный) указатель узла */
    return node;
}

// Если дано непустое двоичное дерево, вернуть узел с минимальным значением ключа, найденного в этом дереве.
Node* minValueNode(Node* node){

    Node* current = node;

    //поиск самого левого листа

    while (current->left_ != NULL)
        current = current->left_;

    return current;
}

Node* deleteNode(Node* root, int key){
    if (!root)
        return nullptr;

    // Если удаляемый ключ меньше ключа root,
    // тогда он лежит в левом поддереве
    if (key < root->key_)
        root->left_ = deleteNode(root->left_, key);

        // Если удаляемый ключ больше ключа root,
        // тогда оно лежит в правом поддереве
    else if (key > root->key_)
        root->right_ = deleteNode(root->right_, key);

        // если ключ совпадает с ключом root, удаляем
```

```
else {
    // Если ключ присутствует более одного раза, просто уменьшить счётчик
    if (root->count_ > 1) {
        (root->count_)--;
        return root;
    }

    // иначе удалим узел

    // узел только с одним дочерним элементом или без него
    if (!root->left_|| !root->right_) {
        Node* temp = root->left_ ? root->left_ : root->right_;

        // Нет дочернего случая
        if (!temp) {
            temp = root;
            root = nullptr;
        }

        else // Один дочерний случай
            *root = *temp; // Копируем содержимое непустого потомка

        delete temp;
    }

    else {
        // узел с двумя дочерними элементами: Получить преемника по порядку (наименьший
        // в правом поддереве)
        Node* temp = minValueNode(root->right_);

        // Копируем данные преемника inorder в этот узел и обновляем счетчик
        root->key_ = temp->key_;
        root->count_ = temp->count_;
        temp->count_ = 1;

        // Удалить наследник преемника
        root->right_ = deleteNode(root->right_, temp->key_);
```

```
        }

    }

    // Если у дерева был только один узел, возвращаем
    if (!root)
        return root;

    // ОБНОВЛЕНИЕ ВЫСОТЫ ТЕКУЩЕГО УЗЛА
    root->height_ = max(height(root->left_), height(root->right_)) + 1;

    //проверка сбалансированности
    int balance = getBalance(root);

    // Если этот узел становится несбалансированным, то есть 4 случая
    // Левый левый регистр
    if (balance > 1 && getBalance(root->left_) >= 0) {
        std::cout << "Баланс узла с ключом " << root->key_ << ": " << balance
                << " и баланс левого сына: " << getBalance(root->left_) << " >=0 => выполняем поворот
влево";
        return rightRotate(root);
    }

    // левый правый регистр
    if (balance > 1 && getBalance(root->left_) < 0) {
        std::cout << "Баланс узла с ключом " << root->key_ << ": " << balance
                << " и баланс левого сына: " << getBalance(root->left_)
                << " <0 => выполнем поворот левого сына влево";
        root->left_ = leftRotate(root->left_);
        std::cout << "Баланс узла с ключом " << root->key_ << ": " << balance <<
                " - выполнем поворот вправо\n";
        return rightRotate(root);
    }

    // правый правый случай
    if (balance < -1 && getBalance(root->right_) <= 0) {
        std::cout << "Баланс узла с ключом " << root->key_ << ": " << balance
                << " и баланс правого сына: " << getBalance(root->right_)
```

```
            << " <=0 => выполнем поворот влево";
        return leftRotate(root);
    }
    // Правый левый регистр
    if (balance < -1 && getBalance(root->right_) > 0) {
        std::cout << "Баланс узла с ключом " << root->key_ << ": " << balance
            << " и баланс правого сына: " << getBalance(root->right_)
            << " >0 => выполнем поворот правого сына вправо";
        root->right_ = rightRotate(root->right_);
        std::cout << "Баланс узла с ключом " << root->key_ << ": " << balance <<
            " - выполнем поворот влево\n";
        return leftRotate(root);
    }

    return root;
}

void clearTree(Node* root){
    if(!root)
        return;
    if(root->left_)
        clearTree(root->left_);
    if(root->right_)
        clearTree(root->right_);
    delete root;
}

bool isDigit(std::string key) {
    for (auto i = 0; i < key.length(); i++)
        if (!std::isdigit(key[i]))
            return false;
    return true;
}

int main()
{
    char chooseInput = 0;
    char chooseAction = 0;
```

```
bool fileOpenFlag = true;
std::string key;
Node* root = nullptr;
while(chooseInput != '2'){
    std::cout << "Выберите действие(0 - консоль, 1 - файл, 2 - выход из программы)\n";
    std::cin >> chooseInput;
    switch(chooseInput){
        case '0':
            while(chooseAction != '*') {
                std::cout << "Выберите действие(+ - добавить элемент, - - удалить элемент, "
                             "# - очистить текущее дерево, * - выход в меню)\n";
                std::cin >> chooseAction;
                switch(chooseAction){
                    case '+':
                        std::cout << "Введите значение ключа добавляемого элемента\n";
                        std::cin >> key;
                        if(!isDigit(key)){
                            std::cout << "Ключом должно быть целое положительное число\n";
                            continue;
                        }
                        root = insert(root, atoi(key.c_str()));
                        std::cout << "Получившееся дерево:\n";
                        printTree(root, 0);
                        break;
                    case '-':
                        std::cout << "Введите значение ключа удаляемого элемента\n";
                        std::cin >> key;
                        if(!isDigit(key)){
                            std::cout << "Ключом должно быть целое положительное число\n";
                            continue;
                        }
                        root = deleteNode(root, atoi(key.c_str()));
                        std::cout << "Получившееся дерево:\n";
                        printTree(root, 0);
                        break;
                    case '#':
                        std::cout<< "Очищаем дерево\n";
                        clearTree(root);
```

```
                root = nullptr;
                break;
            case '*':
                break;
            default:
                std::cout << "Данное действие не поддерживается программой\n";
                continue;
        }
    }
    break;
case '1':{
    std::string fileName;
    while(fileOpenFlag) {
        std::cout << "Введите название файла или q для выхода\n";
        std::cin >> fileName;
        std::fstream file(fileName);
        if (!file.is_open()) {
            std::cout << "Файл не может быть открыт\n";
            continue;
        }
        fileOpenFlag = false;
    }
    std::fstream file(fileName);
    while(chooseAction != '*') {
        std::cout << "Выберите действие(+ - добавить элемент, - - удалить элемент, "
                     "# - очистить дерево, * - выход в меню)\n";
        file >> chooseAction;
        std::cout << chooseAction << '\n';
        if (chooseAction != '-' && chooseAction != '+' && chooseAction != '*' && chooseAction != '#')
{
        }
        switch(chooseAction){
            case '+':
                std::cout << "Введите значение добавляемого ключа\n";
                file >> key;
                std::cout << key << '\n';
                if(!isDigit(key)){
                    std::cout << "Ключом должно быть целое положительное число\n";
```

```
                continue;
            }
            root = insert(root, atoi(key.c_str()));
            std::cout << "Получившееся дерево:\n";
            printTree(root, 0);
            break;
        case '-':
            std::cout << "Введите значение ключа удаляемого элемента\n";
            file >> key;
            if(!isDigit(key)){
                std::cout << "Ключом должно быть целое положительное число\n";
                continue;
            }
            root = deleteNode(root, atoi(key.c_str()));
            std::cout << "Получившееся дерево:\n";
            printTree(root, 0);
            break;
        case '#':
            std::cout<< "Очищаем дерево\n";
            clearTree(root);
            root = nullptr;
            break;
        case '*':
            break;
        default:
            std::cout << "Данное действие не поддерживается программой\n";
            continue;
      }
    }
  break;
  }
  case '2':
    std::cout << "Выход из программы\n";
    break;
  default:
    std::cout << "Программа не поддерживает такой формат ввода\n";
    break;
}
```

```
    }
    clearTree(root);
    return 0;
}
```