

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

**КУРСОВАЯ РАБОТА**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: «Идеально сбалансированное бинарное**  
**дерево поиска»**

Студент гр. 9382

\_\_\_\_\_

Иерусалимов Н.

Преподаватель

\_\_\_\_\_

Фирсов М. А.

Санкт-Петербург

2020

## ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Иерусалимов Н.

Группа 9382

Тема работы: Обработка чисел

Исходные данные:

"Демонстрация" - визуализация структур данных, алгоритмов, действий.

Демонстрация должна быть подробной и понятной (в том числе сопровождаться пояснениями), чтобы программу можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с нею действий.

Содержание пояснительной записки:

«Содержание», «Введение», «Структура программы», «Тестирование»,  
«Выводы»

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 15.12.2019

Дата защиты реферата: 16.12.2019

Студент		Иерусалимов Н.
Преподаватель		Фирсов М. А.

## **АННОТАЦИЯ**

В данной курсовой работе была реализована программа, построения идеально сбалансированного БДП

Основной код программы приведён в приложении А.

## **SUMMARY**

In this course work, a program was implemented to build a perfectly balanced BDP  
The main program code is shown in Appendix A.

## СОДЕРЖАНИЕ

Введение	5
Задание	6
Описание структур данных	7
Описание алгоритма	8
Описание интерфейса пользователя	9
Тестирование	19
Вывод	21
Приложение А	22

## ВВЕДЕНИЕ

### Цель работы.

Ознакомиться с такой структурой данных, как идеально сбалансированное бинарное дерево поиска, и научиться применять БДП на практике.

### Основные теоретические положения.

Бинарное дерево назовем идеально сбалансированным, если для каждой его вершины количество вершин в левом и правом поддереве различаются не более чем на 1. (Рис.1)

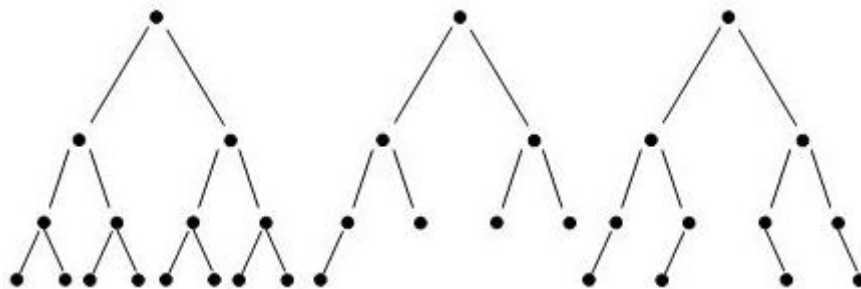


Рис.1 Примеры идеально сбалансированных бинарных деревьев

- Длина внутреннего пути в идеально сбалансированном дереве, содержащем  $n$  вершин, не превосходит величины:

$$(n + 1)[\log_2 n] - 2 * 2^{[\log_2 n]} - 2$$

- Доказательство.

Ясно, что только одна вершина (а именно корень) может находиться на нулевом расстоянии от корня; не более двух вершин могут находиться на расстоянии 1 от корня; не более четырех вершин могут находиться от корня на расстоянии, равном 2 и т.д. Мы видим, что длина внутреннего пути всегда не больше суммы первых  $n$  членов ряда (Рис.2)

- Теперь легко понять, что сумма первых  $n$  членов равна:

$$\sum_{k=1}^n [\log_2 k] = (n+1)[\log_2 n] - \frac{2^{[\log_2 n]+1} - 2}{2-1}$$

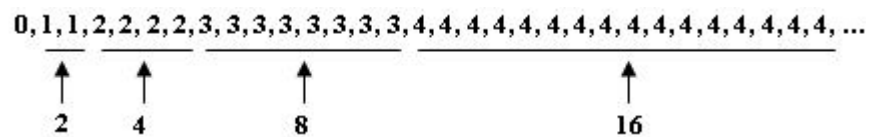


Рис.2. Длина внутреннего пути

откуда и следует утверждение теоремы.

Алгоритм построения идеально сбалансированного дерева при известном числе вершин  $n$  лучше всего формулируется с помощью рекурсии. При этом необходимо лишь учесть, что для достижения минимальной высоты при заданном числе вершин, нужно располагать максимально возможное число вершин на всех уровнях, кроме самого нижнего. Это можно сделать очень просто, если распределять все поступающие в дерево вершины поровну слева и справа от каждой вершины.

- взять одну вершину в качестве корня.
- построить левое поддереву с  $n_l = n \text{ DIV } 2$  вершинами тем же способом.
- построить правое поддереву с  $n_r = n - n_l - 1$  вершинами тем же способом.

В процессе добавления или удаления узлов в дереве возможно возникновение ситуации, когда balance factor некоторых узлов оказывается равными 2 или -2, т.е. возникает *расбалансировка* поддерева. Для выправления ситуации

применяются хорошо нам известные повороты вокруг тех или иных узлов дерева.

Вставка нового ключа в дерево выполняется, по большому счету, так же, как это делается в простых деревьях. 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.

## **Задание.**

### **Вариант 21.**

Идеально сбалансированные БДП – вставка и исключение.

Демонстрация. действие: 1+2а

1) По заданному файлу  $F$  (типа *file of Elem*), построить БДП определённого типа;

2) Для построенной структуры данных проверить, входит ли в неё элемент  $e$  типа *Elem*, и если входит, то в скольких экземплярах. Добавить элемент  $e$  в структуру данных. Предусмотреть возможность повторного выполнения с другим элементом.

Демонстрация - визуализация структур данных, алгоритмов, действий. Демонстрация должна быть подробной и понятной (в том числе сопровождаться пояснениями), чтобы программу можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с ней действий.

### **Пояснение задания.**

На вход программе подаются целочисленные данные из файла или консоли. Нужно построить сбалансированное БДП реализовать поиск, вставку, удаление по этому дереву.



## Описание структур данных

Для создания БДП был создан класс TREE – в котором описывается структура дерева и методы для взаимодействия с ней:

1) Private: - переменные класса

- a) Int key – переменная куда будут записываться данные узла.
- b) TREE\* duk – указатель на корень дерева.
- c) TREE\* Left – указатель на левого ребенка.
- d) TREE\* Right – указатель на правого ребенка.
- e) Int height – высота до корня.

2) Public: - методы класса

- a) TREE() – конструктор класса обнуляет корень.
- b) TREE(int k) – конструктор который создает корень и обнуляет левого и правого ребенка.
  - i) Int k – число которое вставить в корень
- c) ~TREE() – Деструктор проходит по всем узлам ЛПК обходом и очищает память листьев - дерева.
- d) TREE\* rotateLeft(TREE \* node,int n) – делает левый поворот.
  - i) TREE \* node – указатель на корень дерева для поворота
  - ii) int n – количество элементов, для вывода доп данных
- e) TREE\* rotateRight(TREE \* node,int n) – делает правый поворот
  - i) TREE \* node – указатель на корень дерева для поворота
  - ii) int n – количество элементов, для вывода доп данных
- f) TREE\*\*GetDuk() – Возвращает оригинал указателя на корень узла.

- g) `int Height(TREE** p)` – проверяет нулевой ли `p` если да возвращает 0 если нет возвращает высоту данного корня.
- i) `TREE** p` – ссылка на дерево.
- h) `TREE* balance(TREE* p, int n)` – Балансирует дерево, происходит логика с поворотами и нужны ли они вообще.
- i) `TREE* p` – указатель на дерево
- ii) `int n` – число элементов, для вывод доп данных.
- i) `int balanceFactor(TREE** p)` – отнимает значение высоты правого поддерева от левого и возвращает значение.
- i) `TREE** p` – ссылка на дерево.
- j) `void fixHeight(TREE** p)` – исправляет высоту переданного дерева.
- i) `TREE* p` – указатель на дерево
- k) `TREE* findmin(TREE* p)` – находит лист, и возвращает его.
- i) `TREE* p` – указатель на дерево
- l) `TREE* removemin(TREE* p, int n)` – проверяет всех левых детей на наличие правого ребенка.
- i) `TREE* p` – указатель на дерево
- ii) `int n` – число элементов, для вывод доп данных.
- m) `TREE* Insert(int n, TREE** node, int newKey, int prevKey)` – вставляем переданный элемент в дерево. Возвращает указатель на новое дерево.
- i) `TREE** node` – ссылка на дерево
- ii) `int n` – число элементов, для вывод доп данных.
- iii) `int newKey` – новый элемент
- iv) `int prevKey` – предыдущий ключ, доп.данные

- n) `TREE* popElem(int n, TREE* p, int k);`
- i) `TREE* p` – указатель на дерево
  - ii) `int n` – число элементов, для вывод доп данных
  - iii) `int k` – удаляемый элемент.
- o) `Void Search(int num, int* count)` – Производит поиск числа проходя по дереву с помощью ЛПК обхода.
- i) `Int num` – искомое число.
  - ii) `Int* count` – указатель на счетчик.
- p) `Void Tree(int, TREE**, char side, int depth, int parent)` – записывает в дерево элементы из консоли.
- i) `Int` – количество вершин.
  - ii) `TREE**` - оригинальный указатель на корень дерева.
  - iii) `Char side` – для вывода промежуточных данных, дает понять куда идет элемент.
  - iv) `Int depth` - для вывода промежуточных данных, глубина рекурсии.
  - v) `int parent` - для вывода промежуточных данных, родитель нынешнего элемента.
- vi) `Void TreeFromFile(int, TREE**, ifstream in, char side, int depth, int parent)` – записывает в дерево элементы из файла.
- i) `ifstream in` – переменная для потокового считывания из файла.
  - ii) Все остальные аргументы те же что и у метода “e”
- q) `Void print1(TREE** root, short x, Short y, short a, char c)`- выводит дерево в красивом виде.
- i) `TREE** root` - оригинальный указатель на корень дерева.
  - ii) `short x` – первый отступ по x для позиционирования курсора в консоли

- iii) Short `y` – первый отступ по `y`, для позиционирования курсора в консоли
  - iv) short `a` – количество уровней в дереве.
  - v) char `c` – определяет в какой стороне печатать символ лево право корень.
- r) Void `Vyvod(TREE** ,int)` – более простая реализация вывода дерева для написания черновика дерева.
- i) `TREE**` - оригинальный указатель на корень дерева.
  - ii) `Int` – количество вершин.

## Описание алгоритма.

На вход программе подаются целочисленные данные из файла или консоли. Из файла считывает пока не закончатся символы, причем первый символ должен обозначать количество элементов в файле. Из консоли считывается пока не закончится количество элементов которые вы ввели сначала.

Построение идеального бинарного дерева реализуется с помощью метода TREE\*\* Tree(...). Поиск же реализуется с помощью метода Void Search().

1. Tree – рекурсивный метод который принимает в виде входных данных количество вершин и оригинальный указатель на корень дерева. С помощью количества вершин определяется количество узлов в левом поддереве и в правом. Потом записывается веденый элемент во временную переменную после чего вызывается этот же метод для левого поддерева с вычисленным количеством вершин для левого поддерева. Алгоритм выполняется пока количество вершин не будет равняться нулю. После чего вызывается метод уже для правого поддерева, с таким же алгоритмом.
2. Search – Рекурсивный метод который обходит дерево ЛПК обходом. Доходя до листьев, смотрит, равняется ли элемент искомому, если да то инкрементирует счетчик.
3. Insert - опускается в самый левый нижний корень и вставляет новый элемент, после чего происходит балансировка всего дерева.
4. popElem - происходит путем поиска нужного элемента, как находит, записывает его указатели во временные переменные и удаляется этот узел после чего находится самый нижний его элемент в правом или левом поддереве, который и станет заменой удаленному. Происходит балансировка при каждом выходе из рекурсивного метода.

## **Описание интерфейса пользователя**

Приведем примеры на тесте. И покажем получившуюся визуализацию.

Возьмем: 10

12345678910

```

C:\Users\niki\CLionProjects\Algos5\cmake-build-debug\Algos5.exe
0 - Exit. 1 - Create Tree. 2 - Print tree. 3 - Search. 4 - Insert. 5 - Remove elem.
1
1 - Console. 2 - File2
Enter the input file name:
input.txt

/\Add Root = 1 ;

/\Add Left children = 2;
/\His parrent is = 1;
/\Became a tree node...
  1
  /
  2

/\Add Left children = 3;
/\His parrent is = 2;
/\Became a tree node...
    2
    /
    3

/\Add Left children = 4;
/\His parrent is = 3;
/\reached the leaf = 4
      3
      /
      4
     / \
    null null

/\Add Right children = 5 ;
/\His parrent is = 2;
/\Became a tree node...
    2
     \
     5

/\Add Left children = 6;
/\His parrent is = 5;
/\reached the leaf = 6
      5
      /
      6
     / \
    null null

```

Рис.1.1 Создание дерева

```
C:\Users\niki\CLionProjects\Algos5\cmake-build-debug\Algos5.exe

/\Add Right children = 7 ;
/\His parrent is = 1;
/\Became a tree node...
    1
     \
      7

-----

/>\Add Left children = 8;
/>\His parrent is = 7;
/>\Became a tree node...
    7
   /
  8

-----

/>\Add Left children = 9;
/>\His parrent is = 8;
/>\reached the leaf = 9
    8
   /
  9
 / \
null null

-----

/\Add Right children = 10 ;
/\His parrent is = 7;
/\reached the leaf = 10
    7
     \
      10
     / \
    null null

-----

A draft of the received data:

                        10
                       7
                      8
                     9
                    1
                   5
                  6
                 2
                3
               4

Для продолжения нажмите любую клавишу . . .
0 - Exit. 1 - Create Tree. 2 - Print tree. 3 - Search. 4 - Insert. 5 - Remove elem.
```

Рис.1.2 Создание дерева



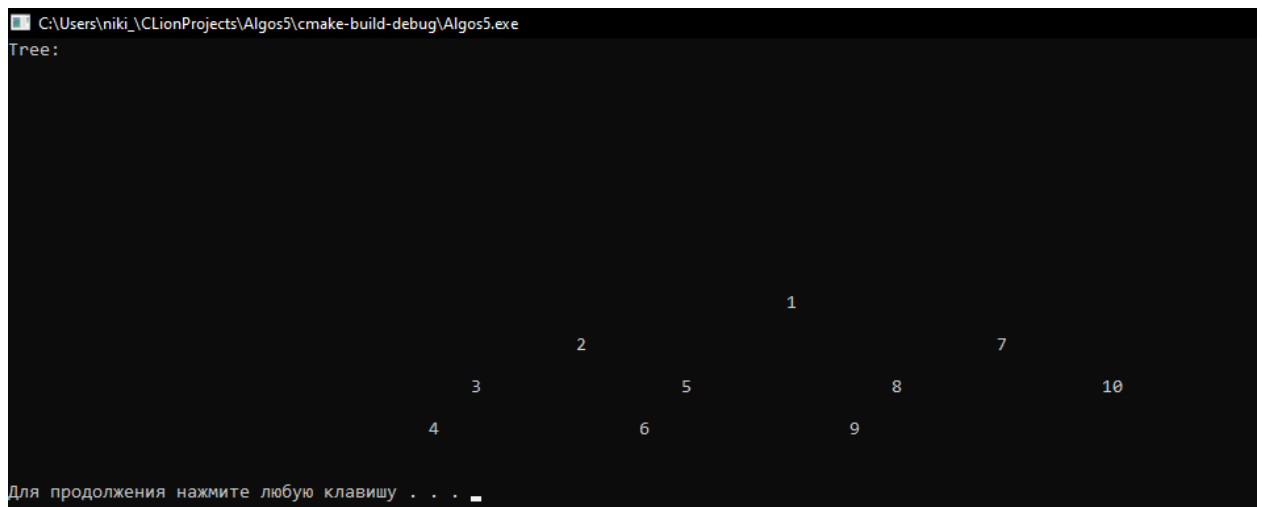


Рис.2 Отрисовка дерева

```
0 - Exit. 1 - Create Tree. 2 - Print tree. 3 - Search. 4 - Insert. 5 - Remove elem.
3
Enter the item you are looking for - ...
5
    Start!O(n)
4 != 5
3 != 5
6 != 5
    Found!
-----
5 == 5
Count = 1.
Height is - 2.
-----
2 != 5
9 != 5
8 != 5
10 != 5
7 != 5
1 != 5
Elemet : 5 amount 1
Для продолжения нажмите любую клавишу . . .
```

Рис.3 Поиск элемента

```

C:\Users\niki\CLionProjects\Algos5\cmake-build-debug\Algos5.exe
0 - Exit. 1 - Create Tree. 2 - Print tree. 3 - Search. 4 - Insert. 5 - Remove elem.
4
Input an insert number: 1
Reached the leaf: 4
Make new leaf or left child: 1
      4
     / \
    null null
     ||
     1

      Next - rebalance Tree.
#####
              4
             1

      Starting balancing !!!

Correcting the height to the root ...
3) Already balanced!
#####
      Next - rebalance Tree.
#####
              3
             4
            1

      Starting balancing !!!

Correcting the height to the root ...
Check if the tree is balanced.
There may be 3 cases:
1) Difference between the heights of the right and left subtree = 2
2) Difference between the heights of the right and left subtree = -2
3) Balancing is not needed.

2) 0-2 = -2

The difference showed the left subtree was larger than the right. (=-2)
Now let's see what kind of rotation we should do
1) Right - if right minus left subtree > 0
2) Left - if the left minus right subtree < 0 .

R - L = 0-1 = -1

Right Rotate!
New Tree:
              3
             4
            1

      Next - rebalance Tree.
#####
              5
             6
            3
           4
          1
            2

      Starting balancing !!!

Correcting the height to the root ...
3) Already balanced!
#####

```

Рис.4.1 Вставка в дерево

```
C:\Users\niki_\CLionProjects\Algos5\cmake-build-debug\Algos5.exe
Next - rebalance Tree.
#####
              10
             7
            8
           9
          1
         5
        2
       4
      3
     1

Starting balancing !!!

Correcting the height to the root ...
3) Already balanced!
#####
Для продолжения нажмите любую клавишу . . .
```

Рис.4.2 Вставка в дерево

```

C:\Users\niki\CLionProjects\Algos5\cmake-build-debug\Algos5.exe
0 - Exit. 1 - Create Tree. 2 - Print tree. 3 - Search. 4 - Insert. 5 - Remove elem.
5
Enter the number you want to remove - ...
1

Found!!!

Remove node...
Found the lowest left leaf and make him new node;

Next - balance this tree
#####
3

Starting balancing !!!

Correcting the height to the root ...
3) Already balanced!
#####

Next - balance this tree
#####
3
4

Starting balancing !!!

Correcting the height to the root ...
3) Already balanced!
#####

Next - balance this tree
#####
6

Starting balancing !!!

Correcting the height to the root ...
3) Already balanced!
#####

Next - balance this tree
#####
6
2
3
4

Starting balancing !!!

Correcting the height to the root ...
3) Already balanced!
#####

Next - balance this tree
#####
9

Starting balancing !!!

Correcting the height to the root ...
3) Already balanced!

```

Рис.5.1 Удаление элемента

```
C:\Users\niki\CLionProjects\Algos5\cmake-build-debug\Algos5.exe

Next - balance this tree
#####
10

Starting balancing !!!

Correcting the height to the root ...
3) Already balanced!
#####

Next - balance this tree
#####
10
7
8
9

Starting balancing !!!

Correcting the height to the root ...
3) Already balanced!
#####

Found!!!

Remove node...
Found the lowest left leaf and make him new node;

Starting balancing !!!

Correcting the height to the root ...
3) Already balanced!
#####

Starting balancing !!!

Correcting the height to the root ...
3) Already balanced!
#####

Next - balance this tree
#####
10
7
8
9
6
2
3
4

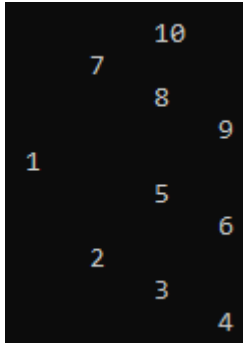
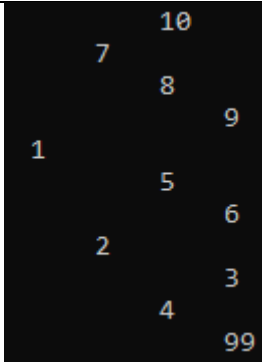
Starting balancing !!!

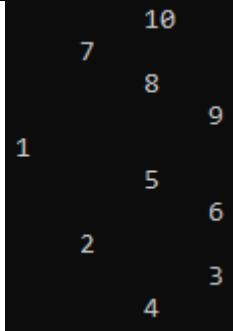
Correcting the height to the root ...
3) Already balanced!
#####
0 - Exit. 1 - Create Tree. 2 - Print tree. 3 - Search. 4 - Insert. 5 - Remove elem.
```

Рис.5.2 Удаление элемента

## Тестирование:

Табл.№1

№	Входные данные	Выходные данные	Комментарии
1.	10 1 2 3 4 5 6 7 8 9 10		Черновик дерева.
Действия с деревом			
2.	2	Рис.2	Отрисовка дерева
3.	3 5	<p>Enter the item you are looking for - ...</p> <p>5</p> <p>Start!O(n)</p> <p>4 != 5</p> <p>3 != 5</p> <p>6 != 5</p> <p>Found!</p> <hr/> <p>5 == 5</p> <p>Count = 1.</p> <p>Height is - 2.</p> <hr/> <p>2 != 5</p> <p>9 != 5</p> <p>8 != 5</p> <p>10 != 5</p> <p>7 != 5</p> <p>1 != 5</p> <p>Elemet : 5 amount 1</p>	Поиск
4.	4 99		Вставка

5.	5 99		Удаление элемента
----	---------	---	-------------------

### **Вывод.**

В процессе выполнения лабораторной работы были продуманы, созданы и реализованы на практике алгоритмы и методы работы со идеальным БДП. Также была реализована визуализация структур данных, алгоритмов, действий. Был получен опыт в подробной, понятной, с пояснениями демонстрации хода алгоритма.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

**Название файла: main.cpp**

```
#include "Avl.h"
int main ()
{
    TREE A;
    A.Start();
}
```

**Название файла: Avl.h**

```
#ifndef ALGOS5_AVL_H
#define ALGOS5_AVL_H

#include<iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fstream>
#include <conio.h>
#include <Windows.h>
using namespace std;

class TREE
{
private:

    int Key;
    unsigned int height;

    TREE *Left;
    TREE *Right;
public:
    TREE *duk; //Корень дерева.
    TREE(int k) {
        duk = nullptr;
        Left = nullptr;
        Right = nullptr;
        height = 1;
        Key = k;
    }
    TREE() {
        duk = nullptr;
        Left = nullptr;
        Right = nullptr;
        height = 1;
    }
}
```



```

~TREE(){
    delete this;
}
TREE* rotateLeft(TREE * node,int n);
TREE* rotateRight(TREE * node,int n);

TREE **GetDuk() { return &duk; }
int Height(TREE** p){
    return (*p)?(*p)->height:0;
}
TREE* balance(TREE* p, int n);

int balanceFactor(TREE** p){
    // cout<< Height(&((*p)->Right)) <<"-"<< Height(&((*p)->Left))
    <<" = "<< int(Height(&((*p)->Right)) - Height(&((*p)->Left)))<<"\n";
    return Height(&((*p)->Right))-Height(&((*p)->Left));
}

void fixHeight(TREE** p){
    int hl = Height(&((*p)->Left));
    int hr = Height(&((*p)->Right));
    (*p)->height = (hl>hr?hl:hr)+1;
}
TREE* findmin(TREE* p) // поиск узла с минимальным ключом в дереве
p
{
    return p->Left?findmin(p->Left):p;
}

TREE* removemin(TREE* p,int n) // удаление узла с минимальным
ключом из дерева p
{
    if( p->Left==nullptr )
        return p->Right;
    p->Left = removemin(p->Left,n);
    return balance(p,n);
}

void Search(int num, int* count);
TREE* Insert(int n,TREE** node, int newKey,int prevKey);
TREE* popElem(int n,TREE* p, int k);

void Tree (int, TREE **,char side,int depth, int parrent);
void TreeFromFile (int n,TREE **p,ifstream& in,char side,int
depth,int parrent);

void print1(TREE ** root, short x, short y, short a, char c);
void printDepth(int depth);

```

```

        void Vyvod (TREE **, int);
        void Start();
};

#endif //ALGOS5_AVL_H

```

### Название файла: Avl.cpp

```
#include "Avl.h"
```

```

TREE* TREE::balance(TREE* p,int n) // балансировка узла p
{
    cout<<"\n\tStarting balancing
!!!\n_____ \n";
    cout<<"Correcting the height to the root ...\n";
    fixHeight(&p);
    if( balanceFactor(&p)==2 )
    {
        cout<<"Check if the tree is balanced.\nThere may be 3
cases:\n1) Difference between the heights of the right and left
subtree = 2\n2) Difference between the heights of the right and left
subtree = -2\n3) Balancing is not needed.\n\n";

        cout<<"1) "<<Height(&p->Right) <<"-"<< Height(&p->Left) <<" =
"<< int(Height(&p->Right)) - Height(&p->Left)<<"\n\n";
        cout<<"The difference showed the right subtree was larger
than the left. (= 2)\nNow let's see what kind of rotation we should
do\n1) Right - if right minus left subtree > 0 \\\n2) Left - if the
left minus right subtree < 0 .\\n\\n";
        cout<<" R - L = "<< Height(&((*p).Right->Right) <<"-"<<
Height(&((*p).Right->Left)) <<" = "<< int(Height(&((*p).Right->Right))
- Height(&((*p).Right->Left)))<<"\n";
        if( balanceFactor(&((*p).Right)) < 0 )
            p->Right = rotateRight(p->Right,n);
        return rotateLeft(p,n);
    }
    if( balanceFactor(&p)==-2 )
    {
        cout<<"Check if the tree is balanced.\nThere may be 3
cases:\n1) Difference between the heights of the right and left
subtree = 2\n2) Difference between the heights of the right and left
subtree = -2\n3) Balancing is not needed.\n\n";

        cout<<"2) "<<Height(&p->Right) <<"-"<< Height(&p->Left) <<" =
"<< int(Height(&p->Right)) - Height(&p->Left)<<"\n\n";

```

```

        cout<<"The difference showed the left subtree was larger than
the right. (=-2)\nNow let's see what kind of rotation we should do\n1)
Right - if right minus left subtree > 0 \n2) Left - if the left minus
right subtree < 0 .\n\n";
        cout<<" R - L = " <<Height(&((*p).Left)->Right) <<"-"<<
Height(&((*p).Left->Left)) <<" = "<< int(Height(&((*p).Left->Right)) -
Height(&((*p).Left->Left)))<<"\n\n";
        if( balanceFactor(&((*p).Left)) > 0 )
            p->Left = rotateLeft(p->Left,n);
        return rotateRight(&(*p),n);
    }
    cout<<"3) Already
balanced!\n#####
\n";
    return p; // балансировка не нужна
}

```

```

TREE* TREE::rotateRight(TREE * node,int n){
    cout<<"Right Rotate!\n";
    TREE* b = node->Left;
    node->Left = b->Right;
    b->Right = node;
    fixHeight(&node);
    fixHeight(&b);
    cout<<"New Tree: \n";
    Vyvod(&b,n);
    return b;
}

```

```

TREE* TREE::rotateLeft(TREE * node, int n){
    cout<<"Left Rotate!\n";
    TREE* b = node->Right;
    node->Right = b->Left;
    b->Left = node;
    fixHeight(&node);
    fixHeight(&b);
    return b;
}

```

```

TREE* TREE::popElem(int n ,TREE* p, int k){
    if( !p ) return 0;
    if( p->Left != nullptr)
        p->Left = popElem(n,p->Left,k);
    if( p->Right != nullptr)
        p->Right = popElem(n,p->Right,k);
}

```

```

        if(p->Key == k)// k == p->key
        {
            cout<<"\n\n\tFound!!!\n_____ \nRemove
node...\nFound the lowest left leaf and make him new node;\n";
            TREE* q = p->Left;
            TREE* r = p->Right;
            p = nullptr;

            if( !r ) return q;
            TREE* min = findmin(r);

            min->Right = removemin(r,n);
            min->Left = q;
            //cout<<"We balance the new result under the
tree.\n_____ \n";
            cout<<"\n\n\n\t\tNext - balance this
tree\n#####\n";
            Vyvod(&min,n);
            return balance(min,n);
        }cout<<"\n\t\tNext - balance this
tree\n#####\n";
        Vyvod(&p,n);
        return balance(p,n);
    }

TREE* TREE::Insert(int n,TREE** node, int newKey,int prevKey){

    if((*node) == nullptr){
        cout<<"Reached the leaf: "<< prevKey<<"\n";
        cout<<"Make new leaf or left child: "<<
newKey<<"\n\t"<<prevKey<<'\n';
        cout<<"          / \\\n";
        cout<<"    "<<"null    null\n";
        cout<<"    "<<"||\n    "<<newKey<<'\n';
        TREE* b = new TREE(newKey);

        return b;
    }else{
        prevKey = (*node)->Key;
        (*node)->Left = Insert(n,&((*node)->Left),newKey,prevKey);
    }
    cout<<"\t\tNext - rebalance
Tree.\n#####\n";

```

```

        Vyvod(&(*node),n);
        return balance(*node,n);
    }

void TREE::printDepth(int depth){
    for(int i =0; i < depth; ++i){
        cout << "/\\\\";
    }
}

void TREE::Search(int num,int* count) {
    if(this->duk != nullptr){
        duk->Search(num,count);
    }else{
        TREE *l =this->Left, *r = this->Right;
        if(l != nullptr){
            l->Search(num, count);
        }
        if(r != nullptr){
            r->Search(num, count);
        }

        if(Key == num) {

cout<<"\n\tFound!"<<"\n_____\\n";
        cout<<Key<<" == "<<num<<'\n';

        *count += 1;
        cout<<"Count = "<< *count<<".\nHeight is - "<<this-
>height<<".\n_____\\n";
        }else{
            cout<<Key<<" != "<<num<<'\n';
        }

    }
}

void TREE::TreeFromFile (int n,TREE **p,ifstream& in,char side,int
depth, int parrent){
    TREE *now;
    int x,nl,nr;
    ++depth;

```

```

now = *p;
if (n==0) *p = NULL;
else
{
    nl = n/2;
    nr = n - nl - 1;
    in>>x;
    now = new TREE;
    (*now).Key = x;
    (*now).height = Height(&now);
    //(*now).height = balanceFactor(&now);
    cout<<"\n_____ \n";
    printDepth(depth);

    if(side == 'l'){
        cout<<"Add Left children = "<< x<<";\n";
        printDepth(depth);
        cout<<"His parrent is = "<< parrent<<";\n";
        if(nl+nr==0){
            printDepth(depth);
            cout<<"reached the leaf = "<<x<<' \n';
            cout<<"\t "<<parrent<<"\n\t /\n\t"<<x<<"\n";
            cout<<"      /  \\\n";
            cout<<"      "<<"null    null\n";
        }else {
            printDepth(depth);
            cout<<"Became a tree node...\n";
            cout<<"\t "<<parrent<<"\n\t /\n\t"<<x<<"\n";
        }
    }else if(side == 'r'){
        cout<<"Add Right children = "<< x<<" ;\n";
        printDepth(depth);
        cout<<"His parrent is = "<< parrent<<";\n";
        if(nl+nr==0){

            printDepth(depth);
            cout<<"reached the leaf = "<<x<<' \n';
            cout<<"\t "<<parrent<<"\n\t  \\\n\t  "<<x<<"\n";
            cout<<"\t /  \\\n";
            cout<<"      "<<"null    null\n";
        }else {
            printDepth(depth);
            cout << "Became a tree node...\n";
        }
    }
}

```

```

        cout<<"\t "<<parent<<"\n\t  \\\n\t  "<<x<<"\n";

    }
    }else if(side == 'k'){
        cout<<"Add Root = "<< x<<" ;\n";
    }
    cout<<"\n_____ \n";
    TreeFromFile (nl,&((*now).Left),in,'l',depth, x);
    --depth;
    TreeFromFile (nr,&((*now).Right), in, 'r',depth,x);
    fixHeight(&now);
    *p = now;

}
--depth;
}

```

```

void TREE::Tree (int n,TREE **p,char side,int depth,int parent){
// Построение идеально сбалансированного
//          дерева с n вершинами.
// *p - указатель на корень дерева.

```

```

    TREE *now;
    int x,nl,nr;
    ++depth;
    now = *p;
    if (n==0) *p = NULL;
    else
    {
        nl = n/2;
        nr = n - nl - 1;
        if(!(cin>>x)){

            cout<<"Error input should be digit!\n";
            cin.clear();
            exit(0);
        }
        else {
            now = new TREE;
            (*now).Key = x;
            (*now).height = Height(&now);
            printDepth(depth);
            if(side == 'l'){
                cout<<"Add Left children = "<< x<<" ;\n";
            }
        }
    }
}

```

```

        printDepth(depth);
        cout<<"His parrent is = "<< parrent<<";\n";
        if(nl+nr==0){
            printDepth(depth);
            cout<<"reached the leaf = "<<x<<' \n';
            cout<<"\t "<<parrent<<"\n\t /\n\t"<<x<<"\n";
            cout<<"      / \\\n";
            cout<<"      "<<"null    null\n";
        }else {
            printDepth(depth);
            cout<<"Became a tree node...\n";
            cout<<"\t "<<parrent<<"\n\t /\n\t"<<x<<"\n";

        }
    }else if(side == 'r'){
        cout<<"Add Right children = "<< x<<" ;\n";
        printDepth(depth);
        cout<<"His parrent is = "<< parrent<<";\n";
        if(nl+nr==0){

            printDepth(depth);
            cout<<"reached the leaf = "<<x<<' \n';
            cout<<"\t "<<parrent<<"\n\t \\\n\t "<<x<<"\n";
            cout<<"\t / \\\n";
            cout<<"      "<<"null    null\n";
        }else {
            printDepth(depth);
            cout << "Became a tree node...\n";
            cout<<"\t "<<parrent<<"\n\t \\\n\t "<<x<<"\n";

        }
    }else if(side == 'k'){
        cout<<"Add Root = "<< x<<" ;\n";
    }
    cout<<"\n_____ \n";
    Tree(nl, &(*now).Left, 'l', depth, x);
    Tree(nr, &(*now).Right, 'r', depth, x);
    fixHeight(&now);
    *p = now;
}
}
--depth;
}

```



```

void TREE::Vyvod (TREE **w,int l)
// Изображение бинарного дерева, заданного
// указателем *w на экране дисплея.
{
    if (*w!=NULL)
    {
        Vyvod (&(**w).Right,l+1);
        for (int i=1; i<=l; i++) cout<<"    ";
        cout<<(**w).Key<<endl;
        Vyvod (&(**w).Left,l+1);
    }
}

void GoToXY (short x, short y)
{
    HANDLE StdOut = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD coord = {x, y};
    SetConsoleCursorPosition(StdOut, coord);
}

void TREE::print1(TREE ** root, short x, short y, short a, char c)
{
    if ((*root) != nullptr)
    {
        if (a>0 && c!='k')
        {
            if (c=='l')
                x-=10*a;
            else
                x+=10*a;
        }
        else
            if (c!='k')
            {
                if (c=='l')
                    x-=4;
                else
                    x+=4;
            }

        GoToXY (x,y+=2);

        a--;

        cout<<(**root).Key;
        print1(&(**root)->Left), x, y, a, 'l');
        print1(&(**root)->Right), x, y, a, 'r');
    }
}

```

```

    }
}

void TREE::Start(){
    TREE A;
    int n=0, count=0, depth = 0;
    int chose, search, fileOrConsole;
    int *arr;

    while(TRUE){
        cout << "0 - Exit. 1 - Create Tree. 2 - Print tree. 3 -
Search. 4 - Insert. 5 - Remove elem.\n";
        fflush (stdin);
        cin >> chose;
        switch(chose){
            case 0:
                exit (0);
                break;
            case 1:
                cout<<"1 - Console. 2 - File";
                if(!(cin>>fileOrConsole)) {
                    cout<<"Error you should input only digit\n";
                    cin.clear();
                }else {
                    if (fileOrConsole == 2) {
                        string input_filename;
                        const string output_filename = "output.txt";
                        ifstream in;
                        ofstream out;
                        out.open(output_filename);
                        out << "";
                        out.close();
                        cout << "Enter the input file name: \n\n";
                        cin >> input_filename;
                        if(input_filename == "1"){
                            in.open("input.txt");
                        }else{
                            in.open(input_filename);
                        }

                        if (in.is_open()) {
                            in >> n;
                            A.TreeFromFile(n, A.GetDuk(), in, 'k',
depth, 0);

```

```

        in.close();
        cout << "A draft of the received data:\n";
        A.Vyvod(A.GetDuk(), n);
    }

    } else {
        cout << "Enter the number of vertices -...\n";
        if((cin >> n)){
            if (n == 0) {
                cout << "Empty array!\n";
            } else {
                cout << "Enter keys...\n";
                A.Tree(n, A.GetDuk(), 'k', depth, 0);
                cout << "A draft of the received
data:\n";

                A.Vyvod(A.GetDuk(), n);

            }
        }else{
            cout<<"Error you should input only
digit\n";

            cin.clear();
        }
        cout<<"The number of characters in the tree -
"<<n<<".\n Its height is - "<<A.Height(A.GetDuk())<<"\n";
    }
    system("pause");
}
break;
case 2:
    system ("cls");
    if(n==0){
        cout<<"None tree\n";
    }else {
        cout << "Tree: \n";
        A.print1(A.GetDuk(), 74, 10, 3, 'k');
    }
    cout<<"\n\n\n\n\n";

    system ("pause");
    break;
case 3:
    cout << "Enter the item you are looking for - ...\n";
    if(!(cin >> search)){

```

```

        cout<<"Error input should be digit!\n";
        cin.clear();
    }else {
        cout<<"\tStart!0(n)\n";
        A.Search(search, &count);
        cout << "Elemet : " << search << " amount " <<
count << '\n';
        count = 0;
    }
    system ("pause");
    break;
case 4:
    int a;
    cout<<"Input an insert number: ";
    if(!(cin>>a)){
        cout<<"\nError input should be digit!\n";
        cin.clear();
    }else {
        A.duk = A.Insert(n, A.GetDuk(), a,99999);
        ++n;
    }
    system ("pause");
    break;
case 5:
    if(n<=1){
        cout<<"Empty tree!\n";
    }else{
        int a;
        cout << "Enter the number you want to remove -
... \n";

        if(!(cin>>a)){
            cout<<"\nError input should be digit!\n";
            cin.clear();
        }else {
            A.duk = A.popElem(n,*A.GetDuk(),a);
            --n;
        }
    }

    break;
default:
    cout<<"Wrong enter!\n";
    break;
}

```

```
        //system ("cls");  
    }  
}
```