МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра вычислительной техники

**КУРСОВОЙ ПРОЕКТ**

**ПО ДИСЦИПЛИНЕ**

**«ПРОГРАММИРОВАНИЕ»**

Тема: Упрощенное 2-3 дерево

Студент: Крохалев А.Г.

Группа: ДТ-660

Преподаватель: Гейнц О.М.

Представлено к защите:

Новосибирск 2017 г.

**Задача:**

Разработать класс, реализующий абстрактный тип данных (АТД) – 2-3 дерево.

Содержание проекта

[1. Структурное описание разработки 3](#_Toc501961160)

[1.1. Структура 2-3 дерева 3](#_Toc501961161)

[1.2. Способ построения 2-3 дерева 4](#_Toc501961162)

[2 Функциональное описание 7](#_Toc501961163)

[2.1. Псевдокод реализации 2-3 дерева 8](#_Toc501961164)

[2.2. Полный код реализации программы. 9](#_Toc501961165)

# Структурное описание разработки

В данной работе реализована структура – абстрактный тип данных – 2-3 дерево.

Сбалансированные деревья являются важными структурами данных позволяющие хранить упорядоченные данные и упрощают поиск этих данных. В данной работе описан абстрактный тип данных - 2-3 деревья - представляющий собой частный случай сбалансированных B-деревьев. В работе рассмотрена структура такого дерева, а также приводится алгоритм и псевдокод вставки и обхода 2-3 дерева.

## Структура 2-3 дерева

2-3 деревья – структура данных, которая представляет собой сбалансированное дерево, такое, что каждое поддерево имеет одинаковую высоту. Узлы таких деревьев хранят один или два ключа, и два или три дочерних объекта: узел с одним ключом имеет два дочерних объекта, узел с двумя ключами - три. Узел, который не имеет дочерних объектов, называется листом, может хранить один или два ключа. Каждый ключ соответствует полю данных.

Любое непустое 2-3 дерево имеет узел, который является основанием дерева и находится на самом верхнем уровне – корень. Корень может быть представлен либо узлом с дочерними объектами, либо листом. Корневой узел не имеет предков. Схематически узел представлен на рисунке 1.

К1 – ключ 1| К2 – ключ 2

L – указатель на левое поддерево

M - указатель на центральное поддерево

R - указатель на правое поддерево

Рисунок 1

Каждый узел является одним из 3-х, описанных ниже:

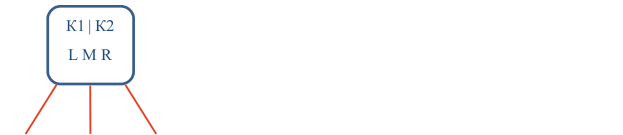
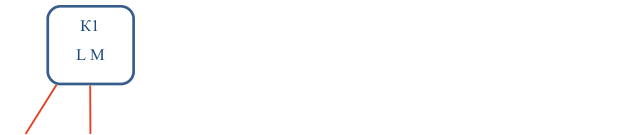
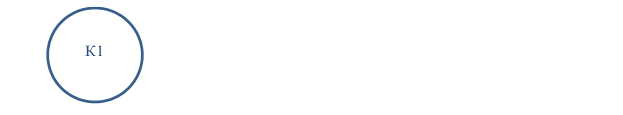
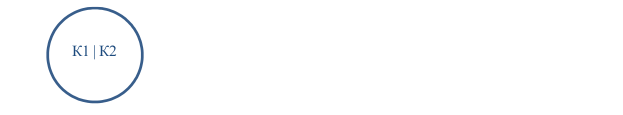
1. Узел, хранящий два поля данных и три дочерних объекта – 
2. Узел, хранящий одно поле данных и два дочерних объекта – 
3. Лист (или терминальный узел), с двумя и одним полем данных, соответственно – 

Схема 2-3 дерева приведена на рисунке 2.

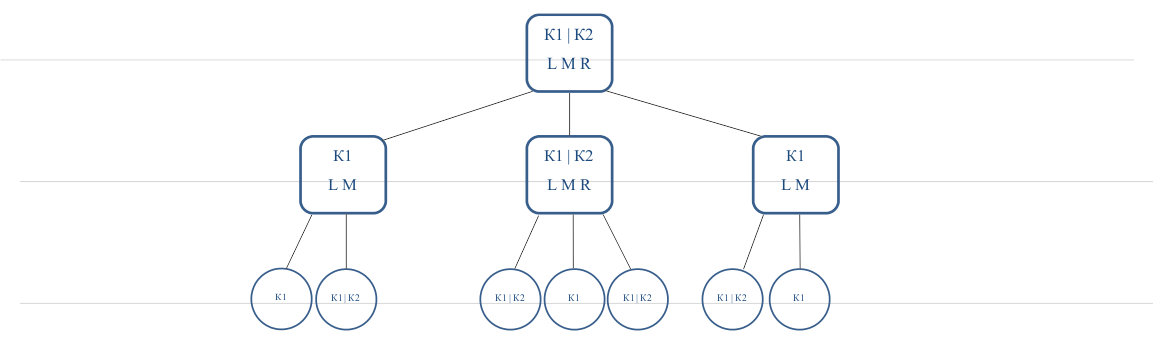


Рисунок 2

В 2-3 дереве данные упорядочены следующим образом:

1. К1 < К2;
2. данные, хранящиеся в левом поддереве всегда меньше К1;
3. данные, хранящиеся в центральном поддереве больше К1, но меньше К2;
4. данные, хранящиеся в правом поддереве всегда больше К2.

## Способ построения 2-3 дерева

**Дерево, представленное одним узлом.**

Если дерево представлено одним узлом, который не имеет дочерних объектов, то данные вставляются в узел «слева направо», т.е. если узел не содержит данных – вставляется элемент в поле с ключом К1, если поле с ключом занято – элемент вставляется в поле с ключом К2. Если оба поля в корне заняты, при этом корень не имеет дочерних объектов, такой узел «делится» на 2 дочерних узла. Образуя дерево из 3- узлов: один – корневой, и два листа. Наименьший из 3 – элементов занимает поле в левом поддереве, элемент с максимальным значением занимает поле в правом поддереве, элемент со средним значением заполняет поле в корневом узле. Отметим, что заполнение полей в каждом узле происходит аналогично описанному выше способу: сначала заполняется поле с ключом К1. Опишем процесс на примере.

Пусть имеется дерево, представленное одним узлом, следовательно, данный узел является одновременно и корнем, и листом. Так же, пусть узел хранит два поля К1 = 1, К2 = 3. Мы хотим вставить в данное дерево элемент, значение которого равно 2.

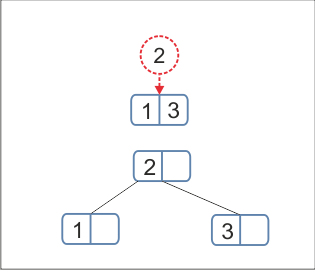


Рисунок 3

Мы проверяем, является ли узел листом и сколько полей в нем занято. Сравниваем элементы, хранящиеся в узле и элемент, который необходимо вставить в дерево.

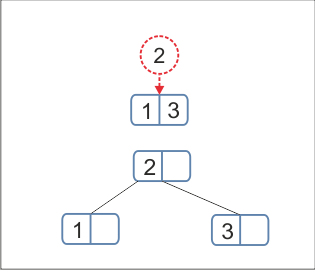


Рисунок 4

Убедившись, что узел является листом и хранит два поля, проверяем, является ли узел корнем. Получив значение – истина, производим деление узла, как показано на рисунке 4. Создаем 2 дочерних узла, помещаем в левый узел меньшее значение, в правый узел – большее значение и в корневой узел помещаем среднее значение.

**Дерево, имеющее больше одного узла**

Если дерево непустое, вставка элемента в дерево осуществляется по следующему алгоритму: сначала ищется подходящий терминальный узел (лист) и вставляется элемент в него по принципу «слева направо», если оба поля заняты, то лист разбивается на 2 листа а среднее значение «выталкивается» в направлении корня. Аналогично для каждого узла. Данный процесс проиллюстрирован на рисунках 5 – 9. Предположим необходимо вставить поочередно значения 18, 19 и 20.

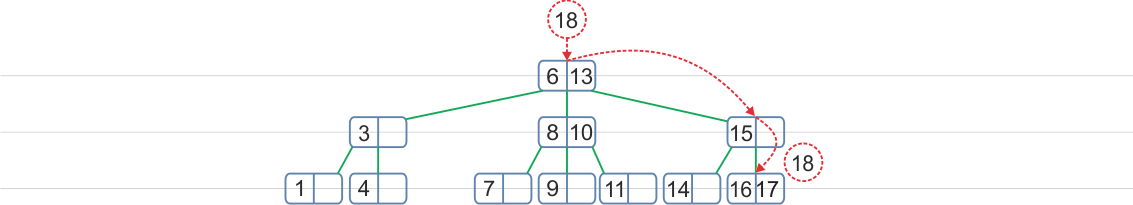


Рисунок 5

Вставляем элемент 18. Ищем терминальный узел. В корневом узле оба поля меньше 18 - идем в правое поддерево. В следующем внутреннем узле хранится одно поле и его значение меньше 18 – идем в среднее поддерево. Достигнув листа, так как оба поля заняты, сравниваем эти поля со значением 18.

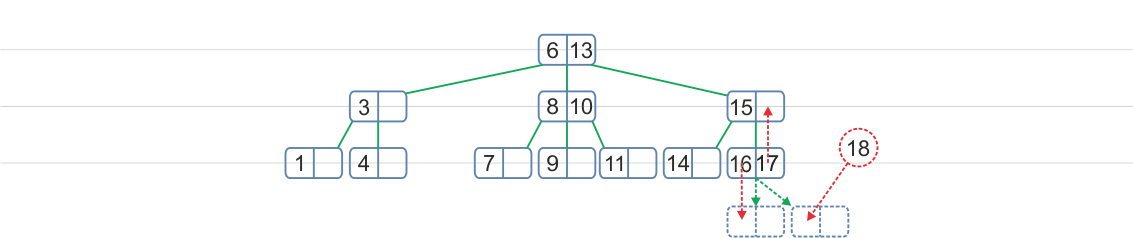


Рисунок 6

Создаем два узла (обозначены пунктирными линиями контура), передаем в них минимальное (16) и максимальное (18) значения, передаем все родительские связи из листа во вновь созданные узлы, которые так же являются листами. Лист с минимальным значением становится средним поддеревом родительского узла, а лист с максимальным значением становится правым поддеревом родительского узла. Среднее значение (17) передается в родительский узел и занимает в нем свободное поле. Полученное дерево изображено на рисунке 7.

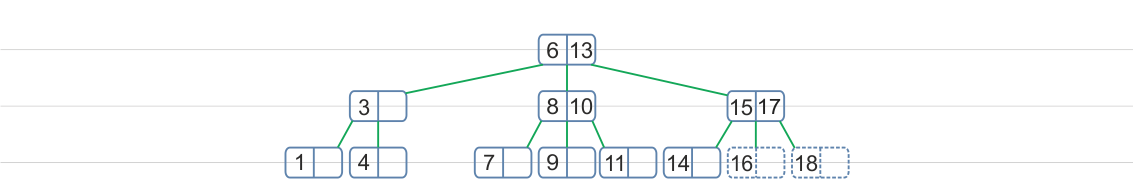


Рисунок 7

Элемент со значением 19 встанет рядом с элементом 18. При вставке элемента со значением 20 процесс усложнится, и дерево вырастет в высоту. До листа процесс аналогичен, описанному выше.

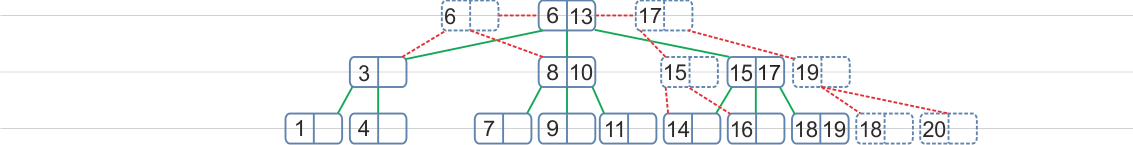


Рисунок 8

При делении листа (18|19) поле 19 передается в родительский узел, который в свою очередь также будет делиться. При делении этого узла дочерние связи данного родительского узла передадутся в новые узлы. Данные связи изображены на рисунке 8 красными пунктирными линиями. В итоге мы придем к узлу, являющемуся корнем данного дерева и этот узел также будет делиться. Окончательно получим дерево, изображенное на рисунке 9. Красными линиями обозначены новые связи, образовавшиеся в результате вставки элемента 20.

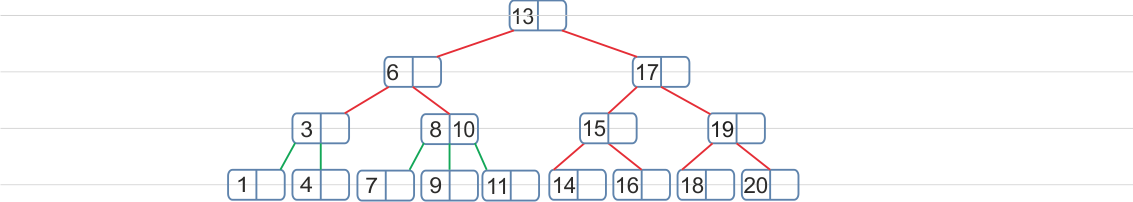


Рисунок 9

***Резюмируем основные свойства 2-3 дерева*.**

1. Любое непустое дерево имеет корень;
2. Каждый узел такого дерева имеет минимум один ключ;
3. Данные в каждом узле упорядочены (если оба поля заняты);
4. Каждое левое поддерево меньше К1 в данной вершине, каждое правое поддерево больше К2 в данной вершине;
5. Дерево растет вверх от корня, но не вниз от листьев.
6. Заполнение узлов данных начинается от листьев последовательно вверх.

# Функциональное описание

В данной программе создано два класса (class Tree и class TreeNode).

Класс TreeNode описывает узел дерева, который содержит два ключа, переменную, хранящую количество ключей в узле, указатели на левое, центральное и правое поддерево.

Класс Tree описывает реализацию 2-3 дерева, содержит вершину дерева, методы вставки и обхода дерева.

Определение класса TreeNode:

class TreeNode

{

friend class Tree;

private:

int sizeNode;//количество элементов в узле

int smallItem, largeItem;//ключи:меньшее и большее соответственно

TreeNode\* leftChildPtr; // указатель на левое поддерево

TreeNode\* midChildPtr; // указатель на центральное поддерево

TreeNode\* rightChildPtr; // указатель на правое поддерево

TreeNode\* parentPtr; // указатель на родителя

TreeNode\*tempLink;//вспомогательный указатель

public:

TreeNode();//конструктор по умолчанию без параметров

TreeNode(const int& anItem); //конструктор с одним ссылочным параметром

TreeNode(const int& smllItm, const int& lrgItm, TreeNode\* leftPtr, TreeNode\* midPtr, TreeNode\* rightPtr, TreeNode\* prntPtr); //конструктор с ссылочными параметрами и параметрами указателями

bool isLeaf() const; //проверка является ли узел листом

bool isTwoNode() const;

bool isThreeNode() const; //проверка узла на количество дочерних объектов

};

Определение класса Tree:

class Tree

{

private:

TreeNode \*rootPtr; // корень дерева

//TreeNode \*tempNode;

void inorder(TreeNode\*);//обход дерева

void insert(TreeNode\*, const int val);//вставка эелемента

void split(TreeNode\*, TreeNode\*, const int );//

void splitLeaf(TreeNode \*, const int );//деление листа

void splitNodeOne(TreeNode\*, TreeNode\*, const int);//вставка в узел с одним ключом

void splitNodeTwo(TreeNode\*, TreeNode\*, const int);//деление узла с двумя ключами

public:

Tree();//конструктор класса

void inordercall();//вызов функции обхода дерева из main()

void insertCall(const int );//вызов функции вставки элемента дерева из main()

}; // конец класса Tree

## Псевдокод реализации 2-3 дерева

Функция обхода 2-3 дерева

inorder(корневой узел)//Обходит непустое 2-3 дерево в порядке следования поисковых ключей.

{

if (корень дерева - лист) {

*Посещаем элемент(ы)*

}

else if (корень содержит один элемент)

{

inorder(левое поддерево корня)

*Посещаем элемент*

inorder(правое поддерево корня)

}

else // Корень содержит два ключа

{

inorder(левое поддерево корня)

*Посещаем первый элемент*

inorder(среднее поддерево корня)

*Посещаем второй элемент*

inorder(правое поддерево корня)

}

}

Функция вставки элемента в 2-3 дерево.

insert(корневой узел, элемент для вставки)

{

if (узел является листом) {

if (узел имеет один ключ) {

if (значение ключа меньше вставляемого элемента) { *Присваиваем значение большему ключу*

*Увеличиваем количество ключей в узле*

}

else if (значение вставляемого элемента меньше ключа в узле)

{

*Ключ становится большим*

*Значение элемента становится меньшим*

*Увеличиваем количество ключей в узле*

*}*

else {

*Совпадение ключа и элемента, выводим «дубль»*

}

}

else//два ключа

{

if (меньший ключ больше значения вводимого элемента) {

*Сохраняем меньший ключ во временную переменную*

*Меньшему ключу присваиваем значение вводимого элемента*

*Делим узел на два узла*

}

else if (значение элемента лежит между меньшим и большим ключами)

{

*Делим узел на два узла*

}

else if (значение элемента больше большего ключа) {

*Сохраняем больший ключ во временную переменную*

*Большему ключу присваиваем значение вводимого элемента*

*Делим узел на два узла*

}

else

{

*Совпадение ключа и элемента, выводим «дубль»*

}

}

}

else {//если узел не лист

if (узел имеет один ключ) {

if (значение вставляемого элемента меньше ключа в узле) {

*идем в левое поддерево*

}

else if (значение больше ключа)

{

*идем в среднее поддерево*

}

else

{

С*овпадение ключа и элемента, выводим «дубль»*

}

}

else//если два ключа

{

if (если значение меньше меньшего ключа) {//

*идем в левое поддерево*

}

else if (значение между ключами)

{

***идем в среднее поддерево***

}

else if (значение больше большего ключа)

{

*идем в правое поддерево*

}

else

{

*Совпадение ключа и элемента, выводим «дубль»*

}

}

}

}

## Полный код реализации программы.

Реализация класса TreeNode:

#pragma once

using namespace std;

class TreeNode

{

friend class Tree;

private:

int sizeNode;//количество элементов в узле

int smallItem, largeItem;//ключи: меньшее и большее соответственно

TreeNode\* leftChildPtr; // указатель на левое поддерево

TreeNode\* midChildPtr; // указатель на центральное поддерево

TreeNode\* rightChildPtr; // указатель на правое поддерево

TreeNode\* parentPtr; // указатель на родителя

TreeNode\*tempLink;//вспомогательный указатель

public:

TreeNode();//конструктор по умолчанию

TreeNode(const int& anItem);//конструктор с одним ссылочным параметром

TreeNode(const int& smllItm, const int& lrgItm, TreeNode \* leftPtr, TreeNode \* midPtr, TreeNode \* rightPtr, TreeNode \*prntPtr);//конструктор, параметры ссылки и указатели

bool isLeaf() const;//проверка листа true

bool isTwoNode() const;

bool isThreeNode() const;//проверка узла на количество дочерних объектов

};

inline TreeNode::TreeNode()

: smallItem(0), largeItem(0), sizeNode(0), leftChildPtr(NULL), midChildPtr(NULL), rightChildPtr(NULL), parentPtr(NULL)

{

}

inline TreeNode::TreeNode(const int & anItem)

: smallItem(anItem), largeItem(0), sizeNode(1), leftChildPtr(NULL), midChildPtr(NULL), rightChildPtr(NULL), parentPtr(NULL)

{

}

inline TreeNode::TreeNode(const int& smllItm, const int& lrgItm, TreeNode \* leftPtr, TreeNode \* midPtr, TreeNode \* rightPtr, TreeNode \*prntPtr)

: smallItem(smllItm), largeItem(lrgItm), sizeNode(1), leftChildPtr(leftPtr), midChildPtr(midPtr), rightChildPtr(rightPtr), parentPtr(prntPtr)

{

}

bool TreeNode::isLeaf() const

{

if (leftChildPtr == NULL && midChildPtr == NULL && rightChildPtr == NULL) {

return true;

}

else {

return false;

}

}

bool TreeNode::isTwoNode() const

{

cout << "isTwoNode " << endl;

if (leftChildPtr != NULL || midChildPtr != NULL || rightChildPtr != NULL) {

return true;

}

else

{

return false;

}

}

bool TreeNode::isThreeNode() const

{

cout << "isThreeNode " << endl;

if (leftChildPtr != NULL && midChildPtr != NULL && rightChildPtr != NULL) {

return true;

}

else

{

return false;

}

}

Реализация класса Tree:

class Tree

{

private:

TreeNode \*rootPtr; // корень дерева

void inorder(TreeNode\*);//обход дерева

void insert(TreeNode\*, const int val);//вставка эелемента

void split(TreeNode\*, TreeNode\*, const int );//

void splitLeaf(TreeNode \*, const int );//деление листа

void splitNodeOne(TreeNode\*, TreeNode\*, const int);//вставка в узел с одним ключом

void splitNodeTwo(TreeNode\*, TreeNode\*, const int);//деление узла с двумя ключами

int n;

long T0 = clock();// Начальное значение времени в миллисекундах

public:

Tree();//конструктор класса

void inordercall();//вызов функции обхода дерева из main()

void insertCall(const int );//вызов функции вставки элемента дерева из main()

}; // конец класса Tree

Tree::Tree()

{

rootPtr = new TreeNode(-3); //

cout << " Конструктор из дерева " << this << endl;

n = 0;

} // конец конструктора Tree

inline void Tree::insertCall(const int p\_val)

{

insert(rootPtr, p\_val);

}

inline void Tree::inordercall()

{

inorder(rootPtr);

}

void Tree::inorder(TreeNode \*p\_ptrRoot)

{

// Обходит непустое 2-3 дерево в

// порядке следования поисковых ключей.

if (p\_ptrRoot->isLeaf()) {//Посещаем элемент(ы)

cout << p\_ptrRoot->smallItem << "~|~" << p\_ptrRoot->largeItem << endl;

}

else if (p\_ptrRoot->sizeNode < 2)//если один ключ - идем в левое и среднее поддерево

//и печатаем ключ в текущем узле

{

inorder(p\_ptrRoot->leftChildPtr);//левое поддерево корня

if(p\_ptrRoot->parentPtr==NULL)cout << "Это корень ";

cout << p\_ptrRoot->smallItem << endl;//Посещаем первый элемент

inorder(p\_ptrRoot->midChildPtr);//среднее поддерево корня

}

else // Корень root содержит два ключа

{

if (p\_ptrRoot->parentPtr == NULL)cout << "Это корень ";

inorder(p\_ptrRoot->leftChildPtr);//левое поддерево корня

cout << p\_ptrRoot->smallItem << endl;

inorder(p\_ptrRoot->midChildPtr);//среднее поддерево корня

cout << p\_ptrRoot->largeItem << endl;

inorder(p\_ptrRoot->rightChildPtr);//правое поддерево корня

} // end if

}

inline void Tree::insert(TreeNode \*p\_ptrRoot, const int p\_val)

{

n++;

if (n % 10 == 0) printf("%d\t%ld\n", n, clock() - T0);

if (p\_ptrRoot->isLeaf()) {//если узел - лист

if (p\_ptrRoot->sizeNode < 2) {//если лист содержит 1 ключ

if (p\_ptrRoot->smallItem < p\_val) {//сравнивает ключи: если введенное

//значение больше ключа

p\_ptrRoot->largeItem = p\_val;

p\_ptrRoot->sizeNode = 2;

}

else if (p\_ptrRoot->smallItem > p\_val)//сравнивает ключи: если

//введенное значение меньше ключа

{

p\_ptrRoot->largeItem = p\_ptrRoot->smallItem;

p\_ptrRoot->smallItem = p\_val;

p\_ptrRoot->sizeNode = 2;

}

else {

cout << "Дубль" << endl;

}

}

else//два ключа

{

if (p\_val < p\_ptrRoot->smallItem) {//если значение меньше меньшего

//ключа

int temp = p\_ptrRoot->smallItem;

p\_ptrRoot->smallItem = p\_val;//сортируем

split(p\_ptrRoot, NULL, temp);//передаем среднее значение в сплит

}

else if (p\_val > p\_ptrRoot->smallItem && p\_val < p\_ptrRoot->largeItem)//если значение между меньшим и большим

{

split(p\_ptrRoot, NULL, p\_val);//передаем в сплит значение

}

else if (p\_val > p\_ptrRoot->largeItem) {//значение больше большего

//ключа

int temp = p\_ptrRoot->largeItem;

p\_ptrRoot->largeItem = p\_val;//сортируем

split(p\_ptrRoot, NULL, temp);//передаем в сплит среднее значение

}

else

{

cout << "Дубль" << endl;

}

}

}

else {//если узел не лист

if (p\_ptrRoot->sizeNode < 2) {//если один ключ сравниваем значение с ключом

if (p\_val < p\_ptrRoot->smallItem) {//если значение меньше ключа

insert(p\_ptrRoot->leftChildPtr, p\_val);//идем в левое поддерево

}

else if (p\_val > p\_ptrRoot->smallItem)//если значение больше ключа

{

insert(p\_ptrRoot->midChildPtr, p\_val);//идем в среднее поддерево

}

else

{

cout << "Дубль" << endl;

}

}

else//если два ключа

{

if (p\_val < p\_ptrRoot->smallItem) {// если значение меньше меньшего

//ключа

insert(p\_ptrRoot->leftChildPtr, p\_val);//идем в левое поддерево

}

else if (p\_val > p\_ptrRoot->smallItem && p\_val < p\_ptrRoot->largeItem)//если значение между ключами

{

insert(p\_ptrRoot->midChildPtr, p\_val);//идем в среднее поддерево

}

else if (p\_val > p\_ptrRoot->largeItem)// если значение больше большего

//ключа

{

insert(p\_ptrRoot->rightChildPtr, p\_val);// идем в правое

//поддерево

}

else

{

cout << "Дубль" << endl;

}

}

}

}

inline void Tree::split(TreeNode \*p\_ptrRoot, TreeNode \*p\_tempNode, const int p\_val) {

if (p\_ptrRoot->parentPtr == NULL)

{

if (p\_ptrRoot->isLeaf()) {//узел - корень-лист и имеет два ключа

splitLeaf(p\_ptrRoot, p\_val);

}

else if (p\_ptrRoot->sizeNode == 2)

splitNodeTwo(p\_ptrRoot, p\_tempNode, p\_val);

}

else

{

splitNodeOne(p\_ptrRoot, p\_tempNode, p\_val);

}

}

else if (p\_ptrRoot->isLeaf())// если узел - лист

{

splitLeaf(p\_ptrRoot, p\_val);

split(p\_ptrRoot->parentPtr, p\_ptrRoot, p\_ptrRoot->smallItem);

}

else// если узел не является ни листом ни корнем

{

if (p\_ptrRoot->sizeNode == 2) //если в узле 2 ключа

{

splitNodeTwo(p\_ptrRoot, p\_tempNode, p\_val);

split(p\_ptrRoot->parentPtr, p\_ptrRoot, p\_ptrRoot->smallItem);

}

else

{//если в узле 1 ключ

splitNodeOne(p\_ptrRoot, p\_tempNode, p\_val);

}

}

}

void Tree::splitLeaf(TreeNode \*p\_ptrRoot, const int p\_val) {

p\_ptrRoot->sizeNode = 1;

TreeNode \*leftNode = new TreeNode(p\_ptrRoot->smallItem, 0, NULL, NULL, NULL, p\_ptrRoot);

TreeNode \*rightNode = new TreeNode(p\_ptrRoot->largeItem, 0, NULL, NULL, NULL, p\_ptrRoot);

p\_ptrRoot->smallItem = p\_val;

p\_ptrRoot->largeItem = 0;

p\_ptrRoot->leftChildPtr = leftNode;

p\_ptrRoot->midChildPtr = rightNode;

p\_ptrRoot->rightChildPtr = NULL;

}

inline void Tree::splitNodeOne(TreeNode \*p\_ptrRoot, TreeNode \*p\_tempNode, const int p\_val)

{

if (p\_val < p\_ptrRoot->smallItem) {//если вводимое значение меньше, чем ключ в узле(прилетело слева)

p\_ptrRoot->sizeNode = 2;

p\_ptrRoot->largeItem = p\_ptrRoot->smallItem;

p\_ptrRoot->smallItem = p\_val;

p\_ptrRoot->rightChildPtr = p\_ptrRoot->midChildPtr;

p\_ptrRoot->leftChildPtr = p\_tempNode->leftChildPtr;

p\_ptrRoot->midChildPtr = p\_tempNode->midChildPtr;

/\*старые родительские связи нужно изменить-------\*/

p\_ptrRoot->leftChildPtr->parentPtr = p\_ptrRoot;

p\_ptrRoot->midChildPtr->parentPtr = p\_ptrRoot;

/\*-----------------------------------------------\*/

delete p\_tempNode;

}

else//если вводимое значение больше, чем ключ в узле(прилетело справа)

{

p\_ptrRoot->sizeNode = 2;

p\_ptrRoot->largeItem = p\_val;/////////////p\_tempNode->largeItem

p\_ptrRoot->midChildPtr = p\_tempNode->leftChildPtr;

p\_ptrRoot->rightChildPtr = p\_tempNode->midChildPtr;

/\*старые родительские связи нужно изменить-------\*/

p\_ptrRoot->rightChildPtr->parentPtr = p\_ptrRoot;

p\_ptrRoot->midChildPtr->parentPtr = p\_ptrRoot;

/\*-----------------------------------------------\*/

delete p\_tempNode;

}

}

inline void Tree::splitNodeTwo(TreeNode \*p\_ptrRoot, TreeNode \*p\_tempNode, const int p\_val)

{

if (p\_val < p\_ptrRoot->smallItem)//прилетело слева

{

p\_ptrRoot->sizeNode = 1;

TreeNode \*rightNode = new TreeNode(p\_ptrRoot->largeItem, 0, p\_ptrRoot->midChildPtr, p\_ptrRoot->rightChildPtr, NULL, p\_ptrRoot);

p\_ptrRoot->largeItem = 0;

p\_ptrRoot->midChildPtr = rightNode;

p\_ptrRoot->rightChildPtr = NULL;

/\*старые родительские связи нужно изменить-------\*/

rightNode->leftChildPtr->parentPtr = rightNode;

rightNode->midChildPtr->parentPtr = rightNode;

/\*-----------------------------------------------\*/

}

else if (p\_val > p\_ptrRoot->smallItem && p\_val < p\_ptrRoot->largeItem)//прилетело из центра

{

p\_ptrRoot->sizeNode = 1;

TreeNode \*rightNode = new TreeNode(p\_ptrRoot->largeItem, 0, p\_tempNode->midChildPtr, p\_ptrRoot->rightChildPtr, NULL, p\_ptrRoot);

p\_ptrRoot->largeItem = 0;

p\_tempNode->smallItem = p\_ptrRoot->smallItem;

p\_ptrRoot->smallItem = p\_val;

p\_tempNode->midChildPtr = p\_tempNode->leftChildPtr;

p\_tempNode->leftChildPtr = p\_ptrRoot->leftChildPtr;

p\_ptrRoot->leftChildPtr = p\_tempNode

p\_ptrRoot->midChildPtr = rightNode;

p\_ptrRoot->rightChildPtr = NULL;

/\*старые родительские связи нужно изменить-------\*/

rightNode->midChildPtr->parentPtr = rightNode;

rightNode->leftChildPtr->parentPtr = rightNode;

p\_tempNode->leftChildPtr->parentPtr = p\_tempNode;

p\_tempNode->midChildPtr->parentPtr = p\_tempNode;

/\*-----------------------------------------------\*/

}

else//прилетело справа

{

p\_ptrRoot->sizeNode = 1;

TreeNode \*leftNode = new TreeNode(p\_ptrRoot->smallItem, 0, p\_ptrRoot->leftChildPtr, p\_ptrRoot->midChildPtr, NULL, p\_ptrRoot);

p\_ptrRoot->smallItem = p\_ptrRoot->largeItem;

p\_ptrRoot->largeItem = 0;

p\_ptrRoot->leftChildPtr = leftNode;

p\_ptrRoot->midChildPtr = p\_tempNode;

p\_ptrRoot->rightChildPtr = NULL;

/\*старые родительские связи нужно изменить-------\*/

leftNode->leftChildPtr->parentPtr = leftNode;

leftNode->midChildPtr->parentPtr = leftNode;

/\*-----------------------------------------------\*/

}

}

Точка входа программы для проверки работоспособности:

#include <iostream>

#include "TreeNode.h"

#include "Tree.h"

#include <fstream>

#include <ctime>

using namespace std;

void main() {

setlocale(LC\_ALL, "ru-ru");

int buff = 0;

srand(time(NULL));

int i = 0;

Tree myTree;

cout << "Введите целое число" << endl;

ifstream fin("cppstudio.txt"); // открыли файл для чтения

while (fin.get() != EOF) {

fin >> buff;

//cout << buff << endl;

myTree.insertCall(buff);

}

fin.close(); // закрываем файл

cout << "Итоговое 2-3 дерево" << endl;

myTree.inordercall();

system("pause");

}