МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет прикладной математики, информатики и механики

Кафедра программного обеспечения

и администрирования информационных систем

**Сбалансированные бинарные деревья поиска (АВЛ-деревья).**

Курсовая работа

Направление 09.03.03 – прикладная информатика (в юриспруденции)



Зав. кафедрой д. ф.-м. н., проф. Артёмов М. А.

Обучающийся 2 к. 10 гр. Фамилия И. О.

Руководитель

Воронеж 2022

# Аннотация

Данная работа позволяет выявить особенности работы с сбалансированными бинарными деревьями на примере языка *C#*.

В работе рассмотрены возможности, предоставляемые данным языком для работы со сбалансированным бинарным деревом, и представлена реализация программы, обеспечивающей выполнение балансировки бинарного дерева.

Тема данной работы имеет важное учебное значение и позволяет изучить основные принципы и идеи сбалансированного бинарного дерева на примере выбранного языка *C#*.

# Содержание

[Введение 4](#_Введение)

[1. Постановка задачи 5](#_1._Постановка_задачи)

[2. Анализ задачи 6](#_2._Анализ_задачи)

[2.1. Общий анализ задачи 6](#_2.1._Общий_анализ)

[2.2. Структура программы 6](#_2.2._Структура_программы)

[2.2.1. Структура основного модуля 6](#_2.2.1._Структура_основного)

[2.2.2. Структура модуля для работы с пользователем 7](#_2.2.2._Структура_модуля)

[2.2.3. Структура связывающего модуля 7](#_2.2.3._Структура_связывающего)

[3. Средства реализации 8](#_3._Средства_реализации)

[4. Требования к программному и аппаратному обеспечению 9](#_4._Требования_к)

[5. Интерфейс пользователя 10](#_5._Интерфейс_пользователя)

[6. Реализация 12](#_6._Реализация)

[6.1. Структура данных 12](#_6.1._Структура_данных)

[6.2. Взаимодействие модулей 12](#_6.2._Взаимодействие_модулей)

[6.3. Структура модулей 13](#_6.3._Структура_модулей)

[6.4. Описание модулей 13](#_6.4._Описание_модулей)

[6.4.1. Структура связывающего модуля 13](#_6.4.1._Структура_связывающего)

[6.4.2. Структура основного модуля 14](#_6.4.2._Структура_основного)

[6.4.3. Структура модуля для работы с пользователем 15](#_6.4.3._Структура_модуля)

[7. План тестирования 17](#_7._План_тестирования)

[7.1. Тест «Ввод некорректной строки» 17](file:///C:\Users\SPuDI\Desktop\Ввод#_7.1._Тест_)

[7.2. Тест «Удаление значения, которого нету в АВЛ-дереве» 17](file:///C:\Users\SPuDI\Desktop\Удаление#_7.2._Тест_)

[7.3. Тест «Удаление значения, из пустого АВЛ-дерева» 17](file:///C:\Users\SPuDI\Desktop\Удаление#_7.3._Тест_)

[Заключение 18](#_Заключение)

[Список литературы 19](#_Список_литературы)

[Приложение 1. Листинг файла MainForm.cs 20](#_Приложение_1._Листинг)

[Приложение 2. Листинг файла TreeAVL.cs 23](#_Приложение_2._Листинг)

[Приложение 3. Листинг файла Node.cs 25](#_Приложение_3._Листинг)

# Введение

Курсовая работа посвящена изучению сбалансированных бинарных деревьев на языке *C#*.

Сбалансированное бинарное дерево поиска или АВЛ-дерево – это дерево, у которого для каждого узла высота двух его поддеревьев отличается не более чем на 1 [2].

АВЛ – аббревиатура, образованная первыми буквами создателей (советских учёных) Адельсон-Вельского Георгия Максимовича и Ландиса Евгения Михайловича [1].

Цель работы – изучить возможности, предоставляемые языком для работы с АВЛ-деревом, и реализовать программу, которая при добавлении или удалении значения из АВЛ-дерева будет выполнять балансировку бинарного дерева.

Для достижения этой цели необходимо решить следующие задачи:

* Выбор средств реализации.
* Разработка структуры программы.
* Реализация программы, демонстрирующей балансировку бинарного дерева.
* Тестирование работоспособности программы.

Тема работы имеет важное учебное значение. Она позволяет освоить работу с АВЛ-деревом и использовать полученные знания при решении практических задач.

# 1. Постановка задачи

Реализовать программу, которая иллюстрирует добавление и удаление значения из АВЛ-дерева.

В программе должны быть реализованы следующие функции:

* Добавление в АВЛ-дерево.
* Удаление из АВЛ-дерева.
* Вывод на форму.

# 2. Анализ задачи

## 2.1. Общий анализ задачи

Для решения поставленной задачи необходимо реализовать программу, которая будет осуществлять работу с АВЛ-деревом.

Для этого необходимо разработать методы, позволяющие выполнять добавление и удаление значения из АВЛ-дерева.

Необходимо обеспечить корректную работу предоставляемых функций.

## 2.2. Структура программы

Для удобства внутри проекта будут выделены три модуля: основной, в котором реализована работа с АВЛ-деревом; модуль для работы с пользователем; связывающий модуль, который связывает основной модуль и модуль для работы с пользователем. Модули содержат методы, выполняющие определенные группы действий.

Каждый модуль содержит класс, реализующий методы, которые выполняют определенные группы действий.

### 2.2.1. Структура основного модуля

Данный модуль состоит из одного класса, в него входят следующие методы, которые осуществляют работу с АВЛ-деревом:

* Метод, который добавляет элемент в АВЛ-дерево и выполняет балансировку дерева.
* Метод, осуществляющий добавление элементов дерева в компонент класса *TreeView*.
* Метод, который очищает дерево.
* Метод, который удаляет элемент из АВЛ-дерева и выполняет балансировку дерева.

### 2.2.2. Структура модуля для работы с пользователем

Данный модуль содержит класс, являющийся наследником встроенного класса *Form*. Объект данного класса является формой, на которой расположены визуальные компоненты. Они предоставляют пользователю следующие возможности:

* Ввод значения.
* Добавление значения в АВЛ-дерево.
* Заполнение АВЛ-дерева случайными значениями.
* Удаление значения из АВЛ-дерева.
* Очистка компонентов и АВЛ-дерева.
* Вывод АВЛ-дерева на форму.

### 2.2.3. Структура связывающего модуля

Данный модуль состоит из одного класса, который связывает модуль для работы с пользователем и основной модуль. В нём отсутствуют вспомогательные переменные и методы, которые использовались в основном модуле.

# 3. Средства реализации

В качестве языка программирования был выбран язык *C#*. Возможности данного языка предоставляют все необходимые инструменты для реализации поставленных в данной работе целей и задач. При реализации проекта использовалась среда разработки *Visual Studio 2022*.

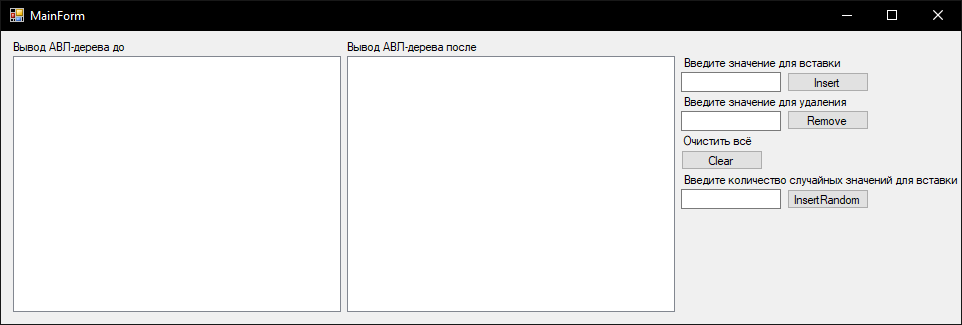
# 4. Требования к программному и аппаратному обеспечению

Приложение предназначено для использования на компьютерах, с установленной операционной системой *Windows XP*, *Windows Vista*, *Windows 7*, *Windows 8*, *Windows 10*.

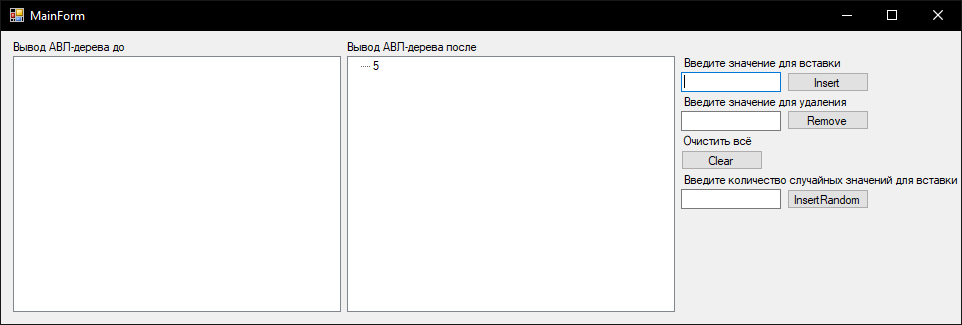
Минимальные системные требования приложения обусловлены требованиями операционной системы.

# 5. Интерфейс пользователя

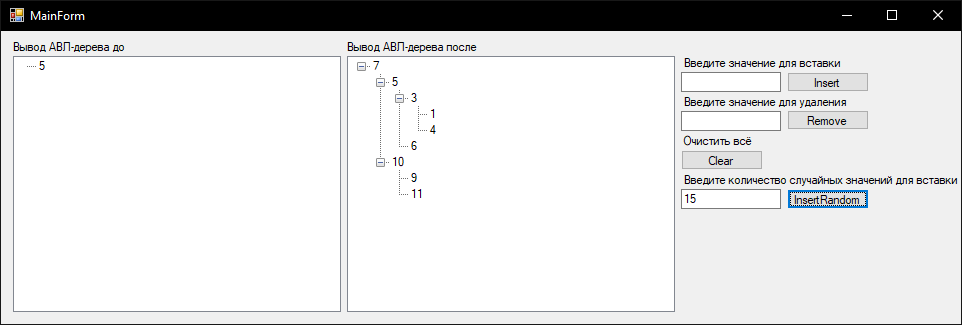
В начале работы пользователю доступна форма, представленная на рис. 5.1.

  
Рис. 5.1. Форма при запуске программы

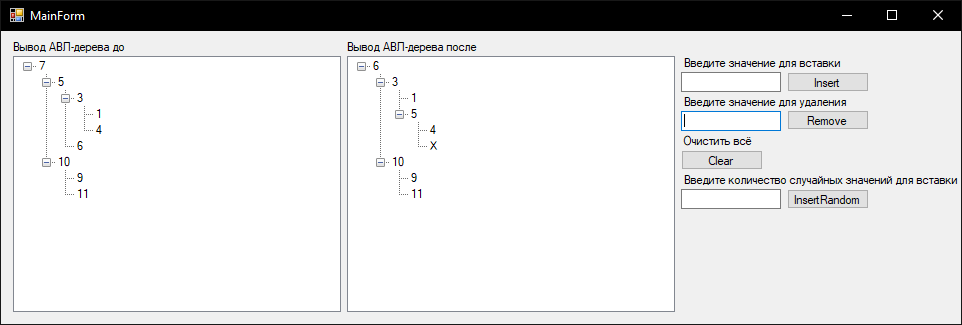
Курсор будет находиться в текстовом поле *textBoxInsert*, в котором пользователю нужно ввести число, которое он хочет добавить в АВЛ-дерево, а затем нажать кнопку *Insert*. АВЛ-дерево до и после добавления числа выведется на компоненты *treeViewWas* и *treeViewBecome* класса *TreeView*. Результат представлен на рис. 5.2.

  
Рис. 5.2. Добавление числа в АВЛ-дерево

При выборе поля *textBoxInsertRandom*, пользователь может написать количество случайных значений, которые добавятся в АВЛ-дерево после нажатия кнопки *InsertRandom*. АВЛ-дерево до и после добавления случайных чисел выведется на компоненты *treeViewWas* и *treeViewBecome* класса *TreeView*. Пример представлен на рис. 5.3.

  
Рис. 5.3. Добавление случайных значений в АВЛ-дерево

Если пользователю нужно удалить значение из АВЛ-дерева, тогда ему нужно выбрать поле *textBoxRemove* и ввести число, а затем нажать кнопку *Remove*. АВЛ-дерево до и после удаления числа семь выведется на компоненты *treeViewWas* и *treeViewBecome* класса *TreeView*. Результат представлен на рис. 5.4.

  
Рис. 5.4. Удаление числа из АВЛ-дерева

Кнопка *Clear* очищает все компоненты, АВЛ-дерево и возвращает форму к первоначальному виду. Результат нажатия представлен на рис. 5.1.

# 6. Реализация

## 6.1. Структура данных

Структура основного модуля:

private int \_value;//Значение элемента дерева

private int \_count;//Количество одинаковых значений

private enum TreeBalance

{

LeftSubtreeGreater = -1,

Balanced = 0,

RightSubtreeGreater = 1

}

private TreeBalance \_balance;//Разность высоты левого

//и правого поддерева

private Node \_left, \_right;//Левое и правое поддерево

Структура связывающего модуля:

private Node \_root;//Корень дерева

private bool \_changedTree;//Изменилось ли дерево

//после удаления

## 6.2. Взаимодействие модулей

Проект состоит из трёх модулей и главной программы, в каждом из модулей расположены подпрограммы, отвечающие за выполнение действий определенного характера. Схема взаимодействия функциональных блоков показана на рис. 6.1.

Рис. 6.1. Схема взаимодействия модулей программы

## 6.3. Структура модулей

1. Модуль для работы с пользователем, содержит форму, из которой происходит вызов необходимых элементов модуля.

2. Модуль, который связывает модуль для работы с пользователем и основной модуль, вызывает методы основного модуля без лишних переменных.

3. Модуль для работы с АВЛ-деревом реализует методы: добавления, удаления элемента, очистки дерева и последующий вывод АВЛ-дерева на форму.

## 6.4. Описание модулей

### 6.4.1. Структура связывающего модуля

В модуле описан класс *TreeAVL*, содержащий следующие элементы:

* public TreeAVL()

Конструктор инициализирует корень дерева в *null*.

* public void Insert(int value)

Процедура *Insert* добавляет значение *value* в АВЛ-дерево.

* public bool Remove(int value)

Функция *Remove* удаляет значение *value* из АВЛ-дерева и возвращает *true*, если такое значение имеется в АВЛ-дереве.

* public bool IsEmpty()

Функция *IsEmpty* возвращает *true*, если корень не равен *null*.

* public void Clear()

Процедура *Clear* очищает АВЛ-дерево и корню присваивает *null*.

* public void Show(TreeView treeView)

Процедура *Show* выводит АВЛ-дерево в компонент класса *TreeView*.

### 6.4.2. Структура основного модуля

В модуле описан класс *Node*, содержащий следующие элементы:

* public Node()

Конструктор инициализирует элемент АВЛ-дерева.

* public bool IsEmpty()

Функция *IsEmpty* возвращает *true*, если корень не равен *null*.

* public Node Insert(int value, out bool isNeedToBalance)

Функция *Insert* добавляет значение *value* в АВЛ-дерево, в зависимости от параметра *isNeedToBalance* балансирует поддерево и возвращает отбалансированное поддерево.

* public Node Remove(int value, out bool isNeedToBalance, out bool isValueFound)

Функция *Remove* удаляет значение *value* из АВЛ-дерева, в зависимости от параметра *isNeedToBalance*, балансирует поддерево, если значение есть в АВЛ-дереве, то параметру *isValueFound* присвоится *true,* и возвращает от балансированное поддерево.

* private Node BalanceLeft(ref bool isNeedToBalance)

Функция *BalanceLeft* возвращает левое от балансированное поддерево и в зависимости от результата работы функции изменяет параметр *isNeedToBalance.*

* private Node BalanceRight(ref bool isNeedToBalance)

Функция *BalanceRight* возвращает правое от балансированное поддерево и в зависимости от результата работы функции изменяет параметр *isNeedToBalance.*

* private Node RemoveNode (Node parent, out bool isNeedToBalance)

Функция *RemoveNode* возвращает левое от балансированное поддерево и заменяет значение родителя *parent* наибольшим значением из потомков и в зависимости от результата работы функции изменяет параметр *isNeedToBalance.*

* public void Show(TreeView treeView)

Процедура *Show* выводит корень в компонент класса *TreeView.*

* private void ShowChilds(TreeNode treeNode)

Процедура *ShowChilds* выводит поддеревья корня *treeNode* в компонент класса *TreeNode.*

### 6.4.3. Структура модуля для работы с пользователем

В модуле описан класс *MainForm*, содержащий следующие элементы:

* public MainForm()

Конструктор инициализирует инициализирует АВЛ-дерево.

* private void buttonInsert\_Click(object sender, EventArgs e)

Процедура *buttonInsert\_Click* добавляет значение, введенное в компоненте *textBoxInsert* класса *TextBox*, в АВЛ-дерево и выводит результат в компоненты *treeViewWas* и *treeViewBecome* класса *TreeView*.

* private void buttonRemove\_Click(object sender, EventArgs e)

Процедура *buttonRemove\_Click* удаляет значение, введенное в компоненте *textBoxRemove* класса *TextBox*, из АВЛ-дерева и выводит результат в компонент *treeViewWas* и *treeViewBecome* класса *TreeView*.

* private void buttonClear\_Click(object sender, EventArgs e)

Процедура *buttonClear\_Click* очищает все компоненты и АВЛ-дерево.

* private void buttonInsertRandom\_Click(object sender, EventArgs e)

Процедура *buttonInsertRandom\_Click* добавляет случайные значения в количестве, введенном в компоненте *textBoxInsertRandom* типа *TextBox*, в АВЛ-дерево и выводит результат в компоненты *treeViewWas* и *treeViewBecome* класса *TreeView*.

* private void textBoxInsert\_Click(object sender, EventArgs e)

Процедура *textBoxInsert\_Click* очищает компоненты *textBoxRemove* и *textBoxInsertRandom* класса *TextBox*.

* private void textBoxInsertRandom\_Click(object sender, EventArgs e)

Процедура *textBoxInsertRandom\_Click* очищает компоненты *textBoxRemove* и *textBoxInsert* класса *TextBox*

* private void textBoxRemove\_Click(object sender, EventArgs e)

Процедура *textBoxRemove\_Click* очищает компоненты *textBoxInsert* и *textBoxInsertRandom* класса *TextBox.*

# 7. План тестирования

Для проверки корректности работы данной программы необходимо провести определённый набор тестов, каждый из которых будет проверять работоспособность определенной операции, производимой с пользовательским интерфейсом программы.

Тестирование данного приложения заключается в проверке работоспособности программы при различных данных, которые может ввести пользователь.

## 7.1. Тест «Ввод некорректной строки»

Цель теста: проверить корректность работы программы в случае ввода пустой строки, или строки из пробелов, или не числовых символов, или дробных чисел в компоненты класса *TextBox*.

Описание теста: вводится пустая строка, или строка из пробелов, или не числовые символы, или дробные числа.

Результат: программа работает корректно. Пользователю выводится сообщение «Некорректное значение».

## 7.2. Тест «Удаление значения, которое отсутствует в АВЛ-дереве»

Цель теста: проверить корректность работы программы в случае ввода значения, которого нет в АВЛ-дереве в компонент *textBoxRemove* класса *TextBox*.

Описание теста: вводится значение, которое отсутствует в АВЛ-дереве.

Результат: программа работает корректно. Пользователю выводится сообщение «Такое значение отсутствует в АВЛ-дереве».

## 7.3. Тест «Удаление значения из пустого АВЛ-дерева»

Цель теста: проверить корректность работы программы в случае удаления значения из пустого АВЛ-дерева.

Описание теста: попытка удаления значение из пустого АВЛ-дерева.

Результат: программа работает корректно. Пользователю выводится сообщение «Пустое АВЛ-дерево».

# Заключение

В ходе выполнения работы были изучены возможности, предоставляемые языком *C#* для работы со сбалансированным бинарным деревом, реализована и протестирована программа, которая выполняет балансировку бинарного дерева при добавлении или удалении значения из АВЛ-дерева.

Работа позволила изучить основные принципы и идеи сбалансированного бинарного дерева на языке *C#*.

В ходе выполнения работы были выполнены следующие задачи:

* Выбраны средства реализации.
* Разработана структура программы.
* Реализована программа, демонстрирующая балансировку бинарного дерева.
* Протестирована работоспособность программы.

# Список литературы

1. АВЛ-деревья. – URL: <https://habr.com/ru/post/150732/> (дата обращения: 06.05.2023).

2. Вирт Н. Алгоритмы и структуры данных / Н. Вирт ; пер. с англ. Д. Б. Подшиваловой. – 2-е изд., испр. – Санкт-Петербург : Нев. диалект, 2001. – 352 с.

# Приложение 1. Листинг файла MainFrom.cs

using System;

using System.Windows.Forms;

namespace AVL

{

public partial class MainForm : Form

{

TreeAVL treeAVL;

public MainForm()

{

InitializeComponent();

treeAVL = new TreeAVL();

}

private void buttonInsert\_Click(object sender, EventArgs e)

{

if (!int.TryParse(textBoxInsert.Text, out int value))

{

textBoxInsert.Clear();

MessageBox.Show("Некорректное значение");

return;

}

treeAVL.Show(treeViewWas);

treeAVL.Insert(value);

treeAVL.Show(treeViewBecome);

textBoxInsert.Clear();

textBoxInsert.Focus();

}

private void buttonRemove\_Click(object sender, EventArgs e)

{

if (treeAVL.IsEmpty())

{

MessageBox.Show("Пустое АВЛ-дерево");

textBoxRemove.Clear();

textBoxRemove.Focus();

return;

}

if (!int.TryParse(textBoxRemove.Text, out int value))

{

textBoxRemove.Clear();

MessageBox.Show("Некорректное значение");

return;

}

treeAVL.Show(treeViewWas);

if (!treeAVL.Remove(value))

MessageBox.Show("Такое значение отсутствует в АВЛ-дереве");

treeAVL.Show(treeViewBecome);

textBoxRemove.Clear();

textBoxRemove.Focus();

}

private void buttonClear\_Click(object sender, EventArgs e)

{

treeAVL.Clear();

treeViewWas.Nodes.Clear();

treeViewBecome.Nodes.Clear();

textBoxInsert.Clear();

textBoxRemove.Clear();

textBoxInsert.Focus();

}

private void buttonInsertRandom\_Click(object sender, EventArgs e)

{

if (!int.TryParse(textBoxInsertRandom.Text, out int value))

{

textBoxRemove.Clear();

MessageBox.Show("Некорректное значение");

return;

}

int maxCount;

Random random = new Random();

treeAVL.Show(treeViewWas);

if (value > 10000)

maxCount = 10000;

else

maxCount = value;

for (int i = 0; i < maxCount; i++)

treeAVL.Insert(random.Next(value));

treeAVL.Show(treeViewBecome);

}

private void textBoxInsert\_Click(object sender, EventArgs e)

{

textBoxRemove.Clear();

textBoxInsertRandom.Clear();

}

private void textBoxRemove\_Click(object sender, EventArgs e)

{

textBoxInsert.Clear();

textBoxInsertRandom.Clear();

}

private void textBoxInsertRandom\_Click(object sender, EventArgs e)

{

textBoxInsert.Clear();

textBoxRemove.Clear();

}

}

}

# Приложение 2. Листинг файла TreeAVL.cs

using System.Windows.Forms;

namespace AVL

{

public class TreeAVL

{

private Node \_root;

private bool \_changedTree;

public TreeAVL()

{

\_changedTree = true;

}

public void Insert(int value)

{

if (IsEmpty())

{

\_root = new Node();

}

\_root = \_root.Insert(value, out \_);

}

public bool Remove(int value)

{

\_root = \_root.Remove(value, out \_, out bool isValueFound);

return isValueFound;

}

public void Clear()

{

\_root = null;

}

public void Show(TreeView treeView)

{

if (!\_changedTree)

{

\_changedTree = true;

}

treeView.Nodes.Clear();

if (!IsEmpty())

{

\_root.Show(treeView);

treeView.ExpandAll();

}

}

public bool IsEmpty()

{

return \_root == null;

}

}

}

# Приложение 3. Листинг файла Node.cs

using System.Windows.Forms;

namespace AVL

{

public class Node

{

private int \_value;

private int \_count;

private enum TreeBalance

{

LeftSubtreeGreater = -1,

Balanced = 0,

RightSubtreeGreater = 1

}

private TreeBalance \_balance;

private Node \_left, \_right;

public Node() { }

public Node Insert(int value, out bool isNeedToBalance)

{

if (IsEmpty())

{

isNeedToBalance= true;

\_value = value;

\_count = 1;

\_left = new Node();

\_right = new Node();

return this;

}

if (\_value > value)

{

\_left = \_left.Insert(value, out isNeedToBalance);

if (!isNeedToBalance)

return this;

switch (\_balance)

{

case TreeBalance.RightSubtreeGreater:

\_balance = TreeBalance.Balanced;

isNeedToBalance = false;

return this;

case TreeBalance.Balanced:

\_balance = TreeBalance.LeftSubtreeGreater;

return this;

case TreeBalance.LeftSubtreeGreater:

Node tempLeft = \_left;

if (tempLeft.\_balance == TreeBalance.LeftSubtreeGreater)

{

\_left = tempLeft.\_right;

tempLeft.\_right = this;

\_balance = TreeBalance.Balanced;

isNeedToBalance = false;

tempLeft.\_balance = TreeBalance.Balanced;

return tempLeft;

}

Node tempLeftRight = tempLeft.\_right;

tempLeft.\_right = tempLeftRight.\_left;

tempLeftRight.\_left = tempLeft;

\_left = tempLeftRight.\_right;

tempLeftRight.\_right = this;

if (tempLeftRight.\_balance == TreeBalance.RightSubtreeGreater)

tempLeft.\_balance = TreeBalance.LeftSubtreeGreater;

else

tempLeft.\_balance = TreeBalance.Balanced;

\_balance = TreeBalance.Balanced;

isNeedToBalance = false;

return tempLeftRight;

default: return null;

}

}

if (\_value < value)

{

\_right = \_right.Insert(value, out isNeedToBalance);

if (!isNeedToBalance)

return this;

switch (\_balance)

{

case TreeBalance.LeftSubtreeGreater:

\_balance = TreeBalance.Balanced;

isNeedToBalance = false;

return this;

case TreeBalance.Balanced:

\_balance = TreeBalance.RightSubtreeGreater;

return this;

case TreeBalance.RightSubtreeGreater:

Node tempRight = \_right;

if (tempRight.\_balance == TreeBalance.RightSubtreeGreater)

{

\_right = tempRight.\_left;

tempRight.\_left = this;

\_balance = TreeBalance.Balanced;

isNeedToBalance = false;

tempRight.\_balance = TreeBalance.Balanced;

return tempRight;

}

Node tempRightLeft = tempRight.\_left;

tempRight.\_left = tempRightLeft.\_right;

tempRightLeft.\_right = tempRight;

\_right = tempRightLeft.\_left;

tempRightLeft.\_left = this;

if (tempRightLeft.\_balance == TreeBalance.LeftSubtreeGreater)

tempRight.\_balance = TreeBalance.RightSubtreeGreater;

else

tempRight.\_balance = TreeBalance.Balanced;

\_balance = TreeBalance.Balanced;

isNeedToBalance = false;

return tempRightLeft;

default: return null;

}

}

\_count++;

isNeedToBalance = false;

return this;

}

public Node Remove(int value, out bool isNeedToBalance, out bool isValueFound)

{

if (IsEmpty())

{

isValueFound = false;

isNeedToBalance = false;

return this;

}

if (\_value > value)

{

\_left = \_left.Remove(value, out isNeedToBalance, out isValueFound);

if (isNeedToBalance)

return BalanceLeft(ref isNeedToBalance);

return this;

}

if (\_value < value)

{

\_right = \_right.Remove(value, out isNeedToBalance, out isValueFound);

if (isNeedToBalance)

return BalanceRight(ref isNeedToBalance);

return this;

}

isValueFound = true;

if (\_right.IsEmpty())

{

isNeedToBalance = true;

return \_left;

}

if (\_left.IsEmpty())

{

isNeedToBalance = true;

return \_right;

}

\_left = \_left.RemoveNode(this, out isNeedToBalance);

if (isNeedToBalance)

return BalanceLeft(ref isNeedToBalance);

return this;

}

public void Show(TreeView treeView)

{

if (IsEmpty())

return;

treeView.Nodes.Add(\_value.ToString());

ShowChilds(treeView.Nodes[0]);

}

private void ShowChilds(TreeNode treeNode)

{

if (!\_left.IsEmpty() || !\_right.IsEmpty())

{

if (!\_left.IsEmpty())

{

treeNode.Nodes.Add(\_left.\_value.ToString());

\_left.ShowChilds(treeNode.Nodes[0]);

}

else treeNode.Nodes.Add("X");

if (!\_right.IsEmpty())

{

treeNode.Nodes.Add(\_right.\_value.ToString());

\_right.ShowChilds(treeNode.Nodes[1]);

}

else treeNode.Nodes.Add("X");

}

}

private Node BalanceLeft(ref bool isNeedToBalance)

{

switch (\_balance)

{

case TreeBalance.LeftSubtreeGreater:

\_balance = TreeBalance.Balanced;

return this;

case TreeBalance.Balanced:

\_balance = TreeBalance.RightSubtreeGreater;

isNeedToBalance = false;

return this;

case TreeBalance.RightSubtreeGreater:

Node tempRight = \_right;

if (tempRight.\_balance == TreeBalance.Balanced ||

tempRight.\_balance == TreeBalance.RightSubtreeGreater)

{

\_right = tempRight.\_left;

tempRight.\_left = this;

if (tempRight.\_balance == TreeBalance.Balanced)

{

\_balance = TreeBalance.RightSubtreeGreater;

tempRight.\_balance = TreeBalance.LeftSubtreeGreater;

isNeedToBalance = false;

return tempRight;

}

\_balance = TreeBalance.Balanced;

tempRight.\_balance = TreeBalance.Balanced;

return tempRight;

}

Node tempRightLeft = tempRight.\_left;

tempRight.\_left = tempRightLeft.\_right;

tempRightLeft.\_right = tempRight;

\_right = tempRightLeft.\_left;

tempRightLeft.\_left = this;

if (tempRightLeft.\_balance == TreeBalance.RightSubtreeGreater)

\_balance = TreeBalance.LeftSubtreeGreater;

else

\_balance = TreeBalance.Balanced;

if (tempRightLeft.\_balance == TreeBalance.LeftSubtreeGreater)

tempRight.\_balance = TreeBalance.RightSubtreeGreater;

else

\_balance = TreeBalance.Balanced;

tempRightLeft.\_balance = TreeBalance.Balanced;

return tempRightLeft;

default: return null;

}

}

private Node BalanceRight(ref bool isNeedToBalance)

{

switch (\_balance)

{

case TreeBalance.RightSubtreeGreater:

\_balance = TreeBalance.Balanced;

return this;

case TreeBalance.Balanced:

\_balance = TreeBalance.LeftSubtreeGreater;

isNeedToBalance = false;

return this;

case TreeBalance.LeftSubtreeGreater:

Node tempLeft = \_left;

if (tempLeft.\_balance == TreeBalance.LeftSubtreeGreater ||

tempLeft.\_balance == TreeBalance.Balanced)

{

\_left = tempLeft.\_right;

tempLeft.\_right = this;

if (tempLeft.\_balance == TreeBalance.Balanced)

{

\_balance = TreeBalance.LeftSubtreeGreater;

tempLeft.\_balance = TreeBalance.RightSubtreeGreater;

isNeedToBalance = false;

return tempLeft;

}

\_balance = TreeBalance.Balanced;

tempLeft.\_balance = TreeBalance.Balanced;

return tempLeft;

}

Node tempLeftRight = tempLeft.\_right;

tempLeft.\_right = tempLeftRight.\_left;

tempLeftRight.\_left = tempLeft;

\_left = tempLeftRight.\_right;

tempLeftRight.\_right = this;

if (tempLeftRight.\_balance == TreeBalance.LeftSubtreeGreater)

\_balance = TreeBalance.RightSubtreeGreater;

else

\_balance = TreeBalance.Balanced;

if (tempLeftRight.\_balance == TreeBalance.RightSubtreeGreater)

tempLeft.\_balance = TreeBalance.LeftSubtreeGreater;

else

\_balance = TreeBalance.Balanced;

tempLeftRight.\_balance = TreeBalance.Balanced;

return tempLeftRight;

default: return null;

}

}

private Node RemoveNode(Node parentNode, out bool isNeedToBalance)

{

if (\_right.IsEmpty())

{

parentNode.\_value = \_value;

parentNode.\_count = \_count;

isNeedToBalance = true;

return \_left;

}

\_right = \_right.RemoveNode(parentNode, out isNeedToBalance);

if (isNeedToBalance)

return BalanceRight(ref isNeedToBalance);

return this;

}

private bool IsEmpty() { return \_count == 0; }

}

}