xМИНОБРНАУКИ РОСИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Восточно-Сибирский государственный университет технологий и управления»

(ФГБОУ ВО ВСГУТУ)

Факультет компьютерных наук и технологий

Кафедра программной инженерии и искусственного интеллекта

**ОТЧЕТ**по лабораторной работе №4  
на тему «Реализация бинарного дерева»

по дисциплине «Структуры и алгоритмы обработки данных»

Выполнил: студент гр. Б763-2 Угнивенко Д.А

Проверил: преподаватель Николаева Т.В.

Улан-Удэ

2024

## Словесная постановка задачи:

Выполнить программную реализацию бинарного дерева. Реализовать добавление и удаление узлов.

## Алгоритмы решения задачи:

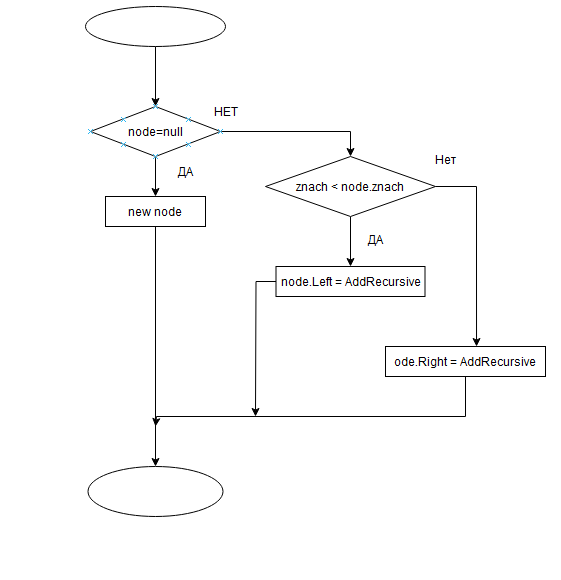
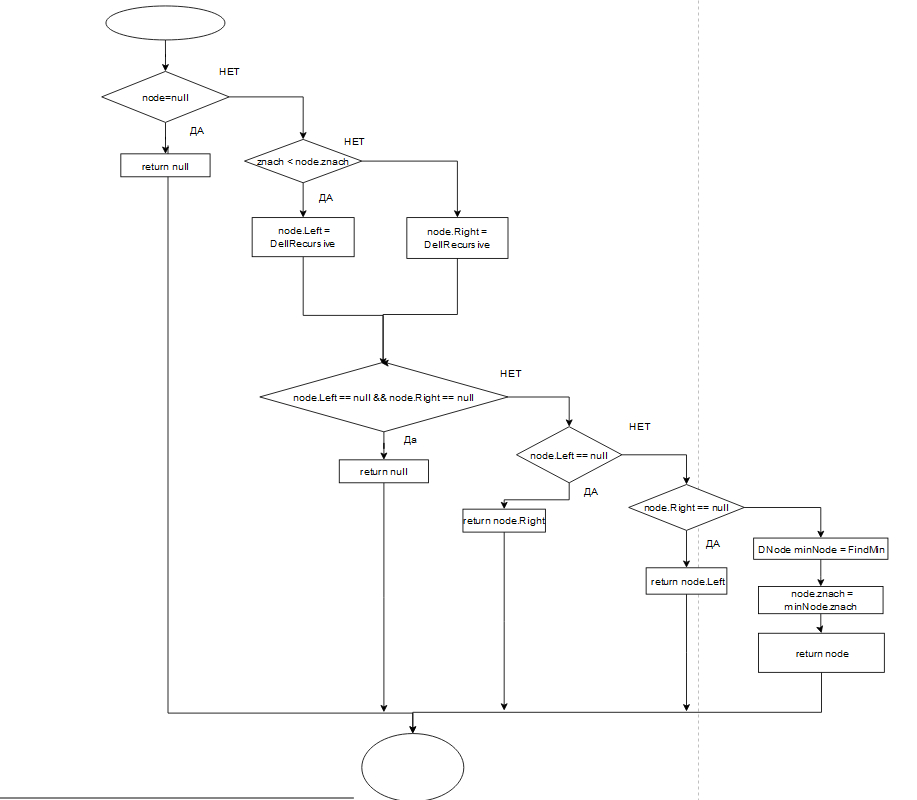


Рисунок 1 - Блок-схема добавления элемента в дерево



## Рисунок 2 - Блок-схема удаления элемента из дерева

## Листинг программы:

Программа реализует бинарное дерево были добавлены элементы 33, 19, 7, 21, 36, 38, 260, 100, 63, 180, 286, 271, 55, 77.

.

Класс

public class DNode

{

public int znach { get; set; }

public DNode Left { get; set; }

public DNode Right { get; set; }

public DNode(int znach)

{

this.znach = znach;

Left = null;

Right = null;

}

}

public class BinaryDrev

{

public DNode Root { get; private set; }

public void Add(int znach)

{

Root = AddRecursive(Root, znach);

}

private DNode AddRecursive(DNode node, int znach)

{

if (node == null) // просто добовляем

{

return new DNode(znach);

}

if (znach < node.znach) // добовляем в лево

{

node.Left = AddRecursive(node.Left, znach);

}

else if (znach >node.znach)// в право

{

node.Right = AddRecursive(node.Right, znach);

}

return node;

}

public void Dell(int znach)

{

Root = DellRecursive(Root, znach);

}

private DNode DellRecursive(DNode node, int znach)

{

if (node == null) return null;

if (znach < node.znach)

{

node.Left = DellRecursive(node.Left, znach);

}

else if (znach > node.znach)

{

node.Right = DellRecursive(node.Right, znach);

}

else

{

// У узла нет потомков

if (node.Left == null && node.Right == null)

{

return null;

}

// У узла только один потомок

else if (node.Left == null) //Узел имеет только правого потомка. Правый потомок становится новым узлом.

{

return node.Right;

}

else if (node.Right == null) //Узел имеет только левого потомка. Левый потомок становится новым узлом.

{

return node.Left;

}

// У узла два потомка

else //Узел имеет двух потомков. Находится минимальный узел в правом поддереве, его значение копируется в текущий узел, а затем минимальный узел удаляется рекурсивно.

{

// Найдите минимальный узел в правом поддереве

DNode minNode = FindMin(node.Right);

node.znach = minNode.znach; // Замена значения

node.Right = DellRecursive(node.Right, minNode.znach); // Удаление минимального узла

}

}

return node;

}

private DNode FindMin(DNode node)

{

while (node.Left != null)

{

node = node.Left;

}

return node;

}

}

Main Window.xmall

<Window x:Class="BinaryTreeApp.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="Дерево" Height="600" Width="800">

<Grid RenderTransformOrigin="0.505,0.161">

<TextBox x:Name="ValueInput" Height="30" VerticalAlignment="Top" Margin="310,10,315,0" TextChanged="ValueInput\_TextChanged"/>

<Button Content="Добавить" Height="30" VerticalAlignment="Top" Margin="113,10,573,0" Click="AddButton\_Click"/>

<Button Content="Удалить" Height="30" VerticalAlignment="Top" Margin="578,10,98,0" Click="RemoveButton\_Click"/>

<Canvas x:Name="TreeCanvas" Background="White" VerticalAlignment="Top" Height="677" Margin="0,107,0,0"/>

</Grid>

</Window>

Main Window.xmall.cs

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media;

using System.Windows.Shapes;

namespace BinaryTreeApp

{

public partial class MainWindow : Window

{

private BinaryDrev tree;

public MainWindow()

{

InitializeComponent();

tree = new BinaryDrev();

}

// данные добавить

private void AddButton\_Click(object sender, RoutedEventArgs e)

{

if (int.TryParse(ValueInput.Text, out int value))

{

tree.Add(value);

ValueInput.Clear();

DrawTree();

}

else

{

MessageBox.Show("Введите корректное значение.");

}

}

//данные удаления

private void RemoveButton\_Click(object sender, RoutedEventArgs e)

{

if (int.TryParse(ValueInput.Text, out int value))

{

tree.Dell(value);

ValueInput.Clear();

DrawTree();

}

else

{

MessageBox.Show("Введите корректное значение.");

}

}

private void DrawTree()

{

TreeCanvas.Children.Clear();

DrawNode(tree.Root, 300, 20, 100);

}

private void DrawNode(DNode node, double x, double y, double offset)

{

if (node == null) return;

// Рисуем узел

Ellipse ellipse = new Ellipse { Width = 30, Height = 30, Fill = Brushes.LightBlue };

Canvas.SetLeft(ellipse, x - 15);

Canvas.SetTop(ellipse, y - 15);

TreeCanvas.Children.Add(ellipse);

// Рисуем текст

TextBlock textBlock = new TextBlock { Text = node.znach.ToString(), Foreground = Brushes.Black };

Canvas.SetLeft(textBlock, x - 10);

Canvas.SetTop(textBlock, y - 10);

TreeCanvas.Children.Add(textBlock);

// Рисуем связи

if (node.Left != null)

{

Line line = new Line

{

X1 = x,

Y1 = y,

X2 = x - offset,

Y2 = y + 50,

Stroke = Brushes.Black,

StrokeThickness = 2

};

TreeCanvas.Children.Add(line);

DrawNode(node.Left, x - offset, y + 50, offset / 2); // Рекурсивно рисуем левое поддерево

}

if (node.Right != null)

{

Line line = new Line

{

X1 = x,

Y1 = y,

X2 = x + offset,

Y2 = y + 50,

Stroke = Brushes.Black,

StrokeThickness = 2

};

TreeCanvas.Children.Add(line);

DrawNode(node.Right, x + offset, y + 50, offset / 2); // Рекурсивно рисуем правое поддерево

}

}

private void ValueInput\_TextChanged(object sender, TextChangedEventArgs e)

{

}

}

}

**тест :**

// Класс узла для двоичного дерева

public class DNode

{

public int znach { get; set; } // Значение узла

public DNode Left { get; set; } // Ссылка на левого потомка

public DNode Right { get; set; } // Ссылка на правого потомка

// Конструктор, который задаёт значение узла

public DNode(int znach)

{

this.znach = znach; // Сохраняем переданное значение

Left = null; // Левый потомок пока отсутствует

Right = null; // Правый потомок пока отсутствует

}

}

// Класс двоичного дерева

public class BinaryDrev

{

public DNode Root { get; private set; } // Корневой узел дерева

// Метод для добавления нового значения в дерево

public void Add(int znach)

{

Root = AddRecursive(Root, znach); // Вызываем рекурсивный метод добавления

}

private DNode AddRecursive(DNode node, int znach)

{

if (node == null) // Если текущего узла нет, создаём новый узел

{

return new DNode(znach);

}

if (znach < node.znach) // Если значение меньше текущего узла, идём влево

{

node.Left = AddRecursive(node.Left, znach);

}

else if (znach > node.znach) // Если значение больше текущего узла, идём вправо

{

node.Right = AddRecursive(node.Right, znach);

}

return node; // Возвращаем текущий узел

}

// Метод для удаления узла по значению

public void Dell(int znach)

{

Root = DellRecursive(Root, znach); // Вызываем рекурсивный метод удаления

}

private DNode DellRecursive(DNode node, int znach)

{

if (node == null) return null; // Если узла нет, возвращаем null

if (znach < node.znach) // Если значение меньше текущего узла, ищем в левом поддереве

{

node.Left = DellRecursive(node.Left, znach);

}

else if (znach > node.znach) // Если значение больше текущего узла, ищем в правом поддереве

{

node.Right = DellRecursive(node.Right, znach);

}

else // Нашли узел, который нужно удалить

{

if (node.Left == null && node.Right == null) // Если у узла нет потомков

{

return null; // Просто удаляем узел

}

else if (node.Left == null) // Если у узла есть только правый потомок

{

return node.Right; // Возвращаем правого потомка

}

else if (node.Right == null) // Если у узла есть только левый потомок

{

return node.Left; // Возвращаем левого потомка

}

else // Если у узла два потомка

{

// Находим минимальный узел в правом поддереве

DNode minNode = FindMin(node.Right);

node.znach = minNode.znach; // Копируем значение минимального узла

node.Right = DellRecursive(node.Right, minNode.znach); // Удаляем минимальный узел

}

}

return node; // Возвращаем текущий узел

}

private DNode FindMin(DNode node)

{

while (node.Left != null) // Ищем самый левый узел

{

node = node.Left;

}

return node; // Возвращаем минимальный узел

}

}

// XAML разметка для пользовательского интерфейса

<Window x:Class="BinaryTreeApp.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="Дерево" Height="600" Width="800">

<Grid RenderTransformOrigin="0.505,0.161">

<!-- Поле ввода значения -->

<TextBox x:Name="ValueInput" Height="30" VerticalAlignment="Top" Margin="310,10,315,0" TextChanged="ValueInput\_TextChanged"/>

<!-- Кнопка для добавления узла -->

<Button Content="Добавить" Height="30" VerticalAlignment="Top" Margin="113,10,573,0" Click="AddButton\_Click"/>

<!-- Кнопка для удаления узла -->

<Button Content="Удалить" Height="30" VerticalAlignment="Top" Margin="578,10,98,0" Click="RemoveButton\_Click"/>

<!-- Область для рисования дерева -->

<Canvas x:Name="TreeCanvas" Background="White" VerticalAlignment="Top" Height="677" Margin="0,107,0,0"/>

</Grid>

</Window>

// Логика интерфейса на C#

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media;

using System.Windows.Shapes;

namespace BinaryTreeApp

{

public partial class MainWindow : Window

{

private BinaryDrev tree; // Экземпляр дерева

public MainWindow()

{

InitializeComponent();

tree = new BinaryDrev(); // Инициализируем дерево

}

private void AddButton\_Click(object sender, RoutedEventArgs e)

{

if (int.TryParse(ValueInput.Text, out int value)) // Проверяем, что введено число

{

tree.Add(value); // Добавляем значение в дерево

ValueInput.Clear(); // Очищаем поле ввода

DrawTree(); // Перерисовываем дерево

}

else

{

MessageBox.Show("Введите корректное значение."); // Выводим ошибку, если ввод некорректен

}

}

private void RemoveButton\_Click(object sender, RoutedEventArgs e)

{

if (int.TryParse(ValueInput.Text, out int value)) // Проверяем, что введено число

{

tree.Dell(value); // Удаляем значение из дерева

ValueInput.Clear(); // Очищаем поле ввода

DrawTree(); // Перерисовываем дерево

}

else

{

MessageBox.Show("Введите корректное значение."); // Выводим ошибку, если ввод некорректен

}

}

private void DrawTree()

{

TreeCanvas.Children.Clear(); // Очищаем холст

DrawNode(tree.Root, 300, 20, 100); // Рисуем дерево с корня

}

private void DrawNode(DNode node, double x, double y, double offset)

{

if (node == null) return; // Если узла нет, ничего не рисуем

// Рисуем узел (окружность)

Ellipse ellipse = new Ellipse { Width = 30, Height = 30, Fill = Brushes.LightBlue };

Canvas.SetLeft(ellipse, x - 15);

Canvas.SetTop(ellipse, y - 15);

TreeCanvas.Children.Add(ellipse);

// Добавляем текст (значение узла)

TextBlock textBlock = new TextBlock { Text = node.znach.ToString(), Foreground = Brushes.Black };

Canvas.SetLeft(textBlock, x - 10);

Canvas.SetTop(textBlock, y - 10);

TreeCanvas.Children.Add(textBlock);

// Если есть левый потомок, рисуем линию и рекурсивно вызываем метод для левого узла

if (node.Left != null)

{

Line line = new Line

{

X1 = x,

Y1 = y,

X2 = x - offset,

Y2 = y + 50,

Stroke = Brushes.Black,

StrokeThickness = 2

};

TreeCanvas.Children.Add(line);

DrawNode(node.Left, x - offset, y + 50, offset / 2);

}

// Если есть правый потомок, рисуем линию и рекурсивно вызываем метод для правого узла

if (node.Right != null)

{

Line line = new Line

{

X1 = x,

Y1 = y,

X2 = x + offset,

Y2 = y + 50,

Stroke = Brushes.Black,

StrokeThickness = 2

};

TreeCanvas.Children.Add(line);

DrawNode(node.Right, x + offset, y + 50, offset / 2);

}

}

private void ValueInput\_TextChanged(object sender, TextChangedEventArgs e)

{

// Обработчик изменения текста в поле ввода (пока ничего не делает)

}

}

}

## }Тестирование:

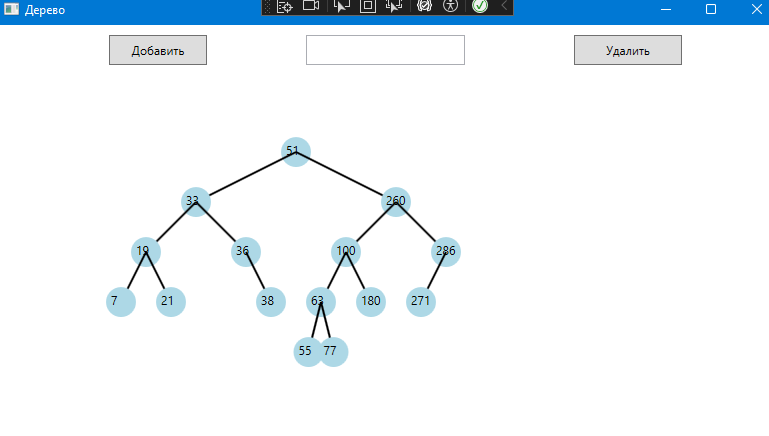


Рисунок 3 – Дерево

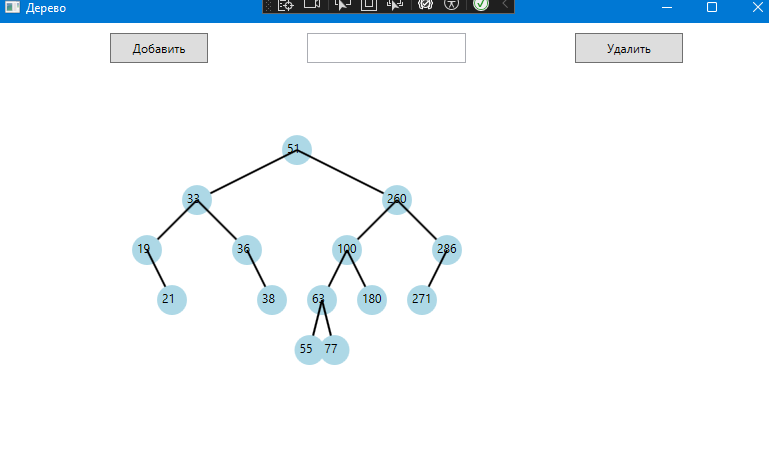


Рисунок 4 - Удаление элемента «7» из дерева

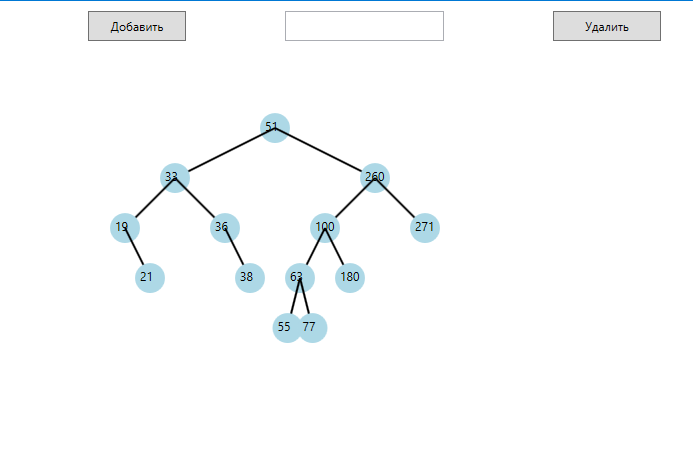


Рисунок 5 - Удаление элемента «286» из дерева

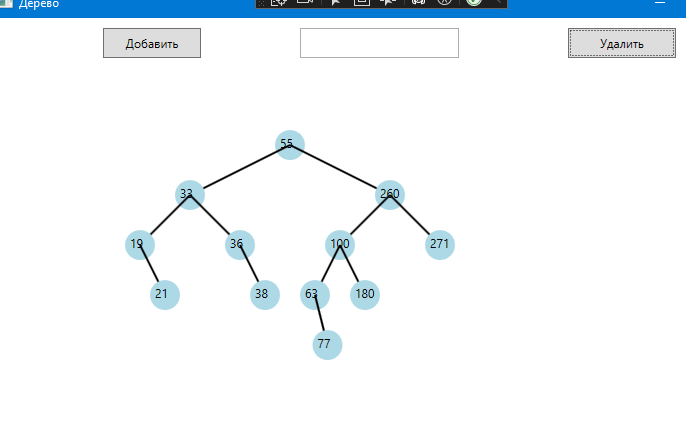


Рисунок6 - Удаление элемента «51» из дерева

Из рисунков, представленных в графе тестирование видно, что добавление происходит с углублением в дерево по правилу вправо если больше, влево если меньше. Для удаления же существуют 3 ситуации: в случае отсутствия дочерних узлов, текущий узел удаляется. В случае существования одного дочернего узла, после удаления текущего, дочерний подтягивается на место удаленного. В третьем случае находится минимальный элемент правого поддерева, который ставится на место удаляемого, в случае если этот элемент имеет дочерний, то он смещается по второму случаю. Что полостью соответствует их теоретической основе.

Ответы на вопросы.

1. Что такое дерево, двоичное дерево, поддерево?

* Дерево: Это абстрактная структура данных, представляющая иерархическую связь между элементами (узлами). Каждый узел может иметь несколько “детей” (потомков), но только одного “родителя” (предка).
  + Корень: Верхний узел дерева, не имеющий предка.
  + Лист: Узел, не имеющий потомков.
  + Уровень: Расстояние от корня до узла (количество ребер на пути).
  + Высота: Максимальный уровень узлов дерева.
* Двоичное дерево: Это дерево, в котором каждый узел может иметь максимум двух потомков: левого и правого.

Поддерево: Это часть дерева, которое само является деревом. Например, левое и правое поддеревья - это поддеревья, связанные с левым и правым потомками корневого узла.

2. Рекурсивные определения:

Дерево - это конечный набор узлов, где:

* Есть один узел, называемый корнем.
* Остальные узлы (если они есть) разделены на непустые, непересекающиеся поддеревья, каждое из которых также является деревом.

Упорядоченное дерево: Упорядоченное дерево - это дерево, в котором потомки каждого узла упорядочены. В двоичном дереве это означает, что у каждого узла есть *левый* и *правый* потомки, и порядок их назначения имеет значение.

3.В этой модифицированной процедуре Add:

* Сначала проверяется, пустое ли дерево. Если да, то новый узел становится корнем.
* В противном случае запускается рекурсивная функция \_AddRecursive.
* В \_AddRecursive перед добавлением нового узла проверяется, не существует ли уже узел с таким же значением. Если такой узел уже есть, то новое значение не добавляется.

4. Сбалансированные деревья

* Идеально сбалансированное дерево: Это двоичное дерево, в котором высота левого и правого поддеревьев каждого узла отличается не более чем на 1.
* Дерево, сбалансированное по АВЛ (AVL-дерево): Это самобалансирующееся бинарное дерево поиска, где для каждого узла разница высот левого и правого поддеревьев не превышает 1. AVL-деревья гарантируют логарифмическую сложность операций поиска, вставки и удаления, в отличие от несбалансированных деревьев, которые могут иметь линейную сложность.