Изображение выглядит как цепь

Автоматически созданное описание

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| УГНС | | | 09.00.00 | Информатика и вычислительная техника | | |
| Направление подготовки | | | 09.03.01 | Информатика и вычислительная техника | | |
| Направленность (профиль) | | |  | Автоматизированные системы обработки информации и управления | | |
| Форма обучения | | |  | очная | | |
|  | | |  |  | | |
| Факультет | | |  | Информационных технологий и управления | | |
| Кафедра | | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Разработка программных систем | | |
| Курс | I | | | Группа | 403 |

Отчёт по лабораторной работе № 1

Вариант № 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 403 |  |  |  | Старков Силантий Денисович |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Федин Алексей Константинович |

Оглавление

[1. Постановка задачи 3](#_Toc96990129)

[2. Исходный данные 3](#_Toc96990130)

[3. Особые ситуации 3](#_Toc96990131)

[4. Математические методы и алгоритмы решения задач 3](#_Toc96990132)

[5. Форматы представления данных 4](#_Toc96990133)

[6. Структура программы 4](#_Toc96990134)

[7. Блок-схемы алгоритмов программы 6](#_Toc96990135)

[8. Описание хода выполнения лабораторной работы 8](#_Toc96990136)

[9. Результаты работы программы 8](#_Toc96990137)

[10. Исходный текст программы 10](#_Toc96990138)

## Постановка задачи

Необходимо реализовать структуру данных «Двоичное дерево поиска», продемонстрировать ее характерные особенности, реализовать возможность добавления и удаления элементов, визуализировать дерево. В программе должны быть предусмотрены три варианта заполнения: пользователем с клавиатуры, из файла и случайными числами.

## Исходный данные

В качестве исходных данных программа использует вводимую пользователем последовательность чисел, в программе она парсится и каждое числовое значение, содержащееся в ней, становится частью объекта класса Node и представляет значение узла в дереве. В этом классе также хранится информация о ближайших большем и меньшем узле, а также информация о родителе узла, если таковые имеются.

## Особые ситуации

Необходимо рассмотреть следующие особые ситуации.

* Если пользователь ввёл значение, не соответствующее операбельному типу, вычислений не происходит, программа просит ввести корректное значение, либо выполняет преобразование к нужному типу.
* Если пользователь ввёл несколько одинаковых значений, в дерево попадает только первая итерация данного числа.
* Если при файловом вводе или при удалении или добавлении элемента возникает конфликт построения дерева, программа выводит сообщение об ошибке и переносит пользователя в главное меню.

## Математические методы и алгоритмы решения задач

Согласно постановке задачи для составления программы будет использовано двоичное дерево поиска.

У каждого узла будут члены, содержащие ссылки на правый элемент, левый элемент и элемент-родителя при их наличии.

Добавление, поиск и удаление узлов будут осуществляться с помощью рекурсии и проверки на соответствие числовых значений узлов.

:

## Форматы представления данных

Программа использует следующие переменные:

Таблица 1 – Переменные, используемые в программе

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Описание |
| Right | Node | Член-данное класса Node, хранящий правый узел от данного |
| Left | Node | Член-данное класса Node, хранящий левый узел от данного |
| Parent | Node | Член-данное класса Node, хранящий ссылку на родителя этого узла |
| Data | int | Член-данное класса Node, хранящий собственное значение |
| tree | BinaryTree | Объект, представляющий методы по добавлению, поиску и удалению элемента классу Node |
| root | Node | Объект – член класса BinaryTree, являющийся корнем всего дерева. |
| pictureBox | PictureBox | Контейнер, необходимый для визуализации дерева |
| NodeDraw | NodeForDraw | Объект, представляющий собой узел с собственным цветом, положением |
| nodeListLines | List  <List  <NodeForDraw  >> | Список линий, соединяющих узлы для отрисовки дерева |

Для хранения узлов дерева используется класс Node, в котором хранятся характеристики узла, а именно числовое значение узла, ссылки на узлы родителя, левого и правого ребенка.

Для реализации разных методов ввода используется абстрактный класс Input, от которого наследуются классы, в которых определены методы ввода с клавиатуры, из файла и случайными значениями.

## Структура программы

Программа разделена на следующие модули:

1. Program: начальный модуль с функцией Main. В нем происходит замуск главного окна приложения.
2. MainForm: в нем находятся функции, инициализирующие компоненты формы и осуществляющие связь компонентов формы с бизнес-логикой проиложения.
3. KeyboardInput, RandomInput, FileInput: модули, в которых определены методы для создания объектов класса в соответствии с выбором пользователя.
4. Tree: модуль, хранящий дерево и предоставляющий функции для работы с ним.
5. BinaryTreeTests: модуль, хранящий тесты основных расчётных функций.
6. MessageForm: модуль, инициализирующий компоненты окна с дополнительной информацией.
7. App.config: модуль, необходимый для сохранения пользовательских настроек.

Программа разделена на несколько функций:

Таблица 3 – Функции, составляющие программу

|  |  |
| --- | --- |
| Имя | Описание |
| Insert | Функция необходимая для корректного добавления новых узлов в дерево |
| Remove | Функция необходимая для корректного удаления узлов из дерева |
| Find | Функция для нахождения узла по числовому значению |
| Input | Перегруженный метод, осуществляющий генерацию массива чисел для добавления их в дерево. |
| PrintBinaryTree | Функция для корректного сохранения дерева файл |
| ShowGreeting | Функция, показывающая приветствие при запуске программы |
| SavelData | Функция, сохраняющая исходные данные |
| SavelResults | Функция, сохраняющая дерево в файл |
| PlantTree | Функция, осуществляющая отображение дерева в пользовательском интерфейсе |
| DrawLine | Функция, осуществляющая отрисовку линий между узлами в интерфейсе |
| MoveNode | Функция, позволяющая осуществлять передвижение узла пользователем |

## Блок-схемы алгоритмов программы

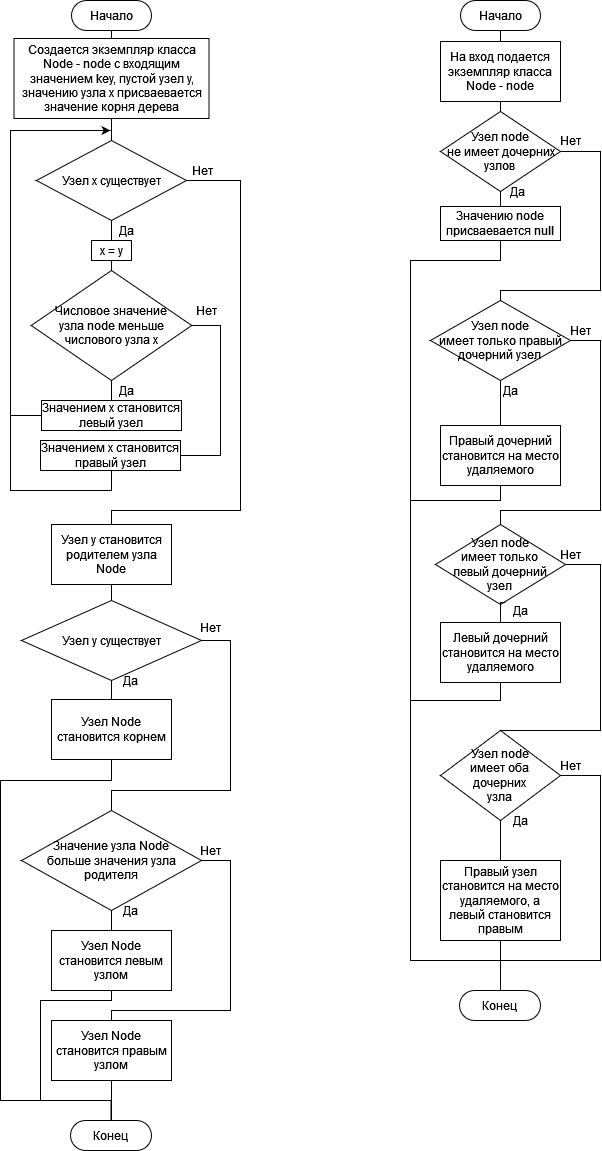


Рисунок 1 – Блок схема алгоритма добавления узла в дерево

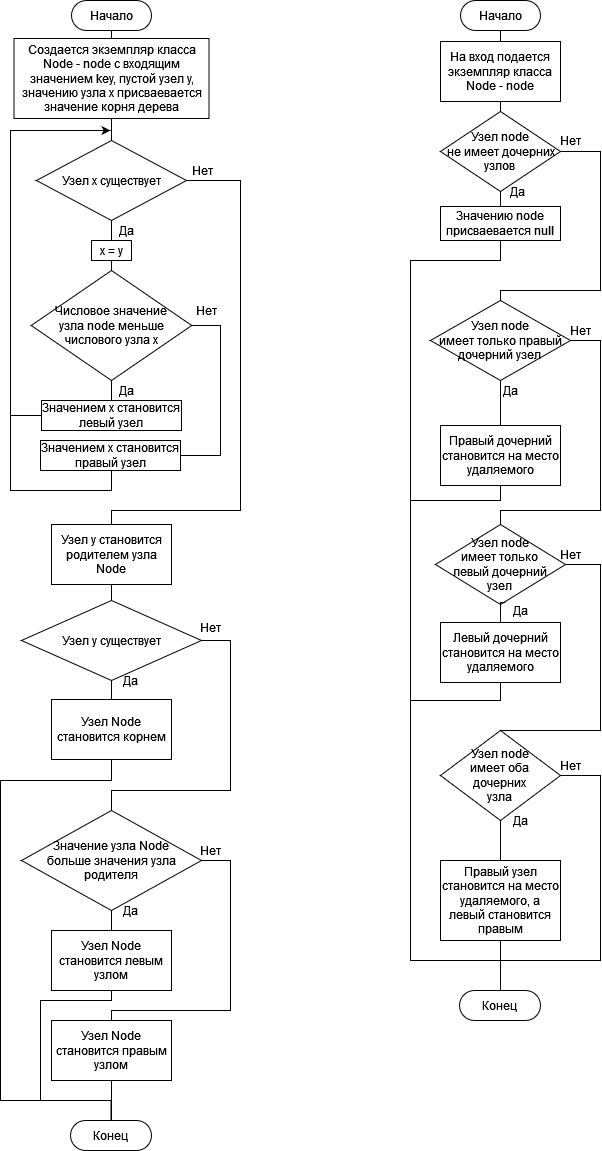


Рисунок 2 – Блок схема алгоритма удаления узла из дерева

## Описание хода выполнения лабораторной работы

1. В ходе лабораторной работы было создано решение (Solution) в интегрированной среде разработки Microsoft Visual Studio 2019. В нём был создан проект.
2. После написания приложения, было выяснено, что при движении объекта в PictureBox, происходит раздвоение этого объекта. Проблема была решена привязкой движения узла к движению курсора.
3. Была добавлена функция ClearPictureBox()для перерисовки дерева и отдельных его веток.

## Результаты работы программы

В результате работы программы, на экран выводится бинарное дерево чисел.

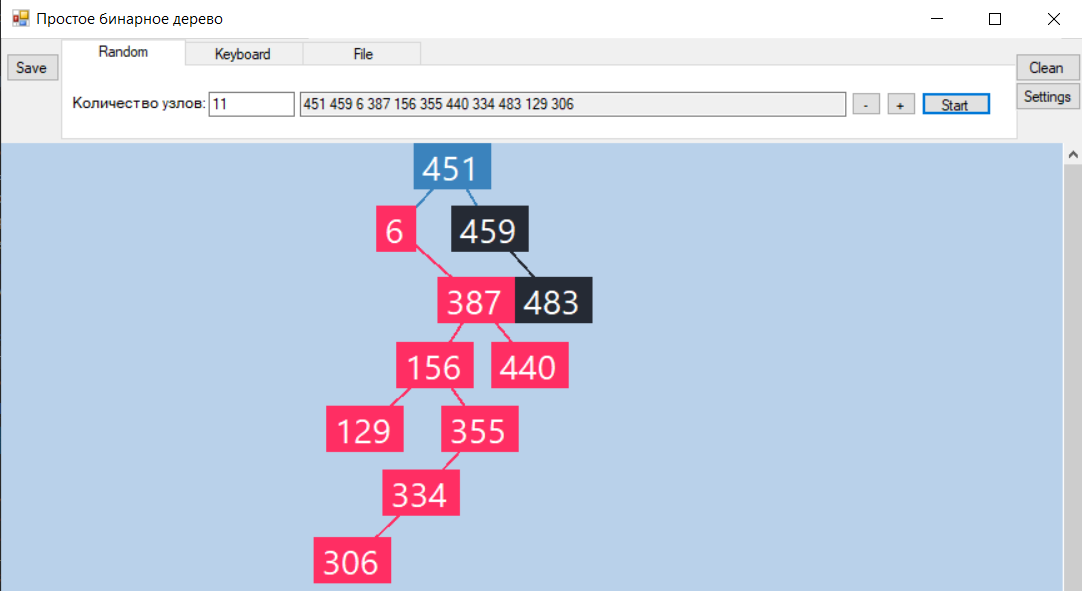


Рисунок 3 – Результат работы программы

Во всех текстовых полях осуществляется валидация входных значений, что исключает возможность добавления некорректных значений в дерево.

При добавлении в дерево уже существующего значения, появляется сообщение об ошибке.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 – Добавление элемента

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 – Сообщение об ошибке при добавлении уже существующего элемента

Аналогично при удалении несуществующего элемента.

При некорректности данных в открываемом файле появляется сообщение об ошибке.

**Изображение выглядит как текст

Автоматически созданное описание**

Рисунок 6 – Сообщение о некорректности вводимых данных

## Исходный текст программы

[ Начало программы ---]

//Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab1

{

static class Program

{

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainForm());

}

}

}

//Tree.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Windows.Forms;

using System.Drawing.Drawing2D;

namespace Lab1 {

public enum BinSide {

Left,

Right

}

public class Node {

public int? Data { get; set; }

public Node Parent { get; set; }

public Node Left { get; set; }

public Node Right { get; set; }

public Node(int? data = null) {

this.Data = data;

this.Parent = null;

this.Left = null;

this.Right = null;

}

}

public class BinaryTree {

public Node root;

public void Insert(int key) {

if (Find(key)==null) {

Node node = new Node(key);

Node y = null;

Node x = this.root;

while (x != null) {

y = x;

if (node.Data < x.Data) {

x = x.Left;

}

else {

x = x.Right;

}

}

node.Parent = y;

if (y == null) {

root = node;

}

else if (node.Data < y.Data) {

y.Left = node;

}

else {

y.Right = node;

}

}

}

private BinSide? MeForParent(Node node) {

if (node.Parent == null) return null;

if (node.Parent.Left == node) return BinSide.Left;

if (node.Parent.Right == node) return BinSide.Right;

return null;

}

public void RemoveByNode(Node node) {

if (node == null) return;

var me = MeForParent(node);

if (me == null) {

}

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

if (me == BinSide.Left) {

node.Parent.Left = null;

}

else {

node.Parent.Right = null;

}

return;

}

if (node.Left == null) {

if (me == BinSide.Left) {

node.Parent.Left = node.Right;

}

else {

node.Parent.Right = node.Right;

}

node.Right.Parent = node.Parent;

return;

}

if (node.Right == null) {

if (me == BinSide.Left) {

node.Parent.Left = node.Left;

}

else {

node.Parent.Right = node.Left;

}

node.Left.Parent = node.Parent;

return;

}

if (me == BinSide.Left) {

node.Parent.Left = node.Right;

}

if (me == BinSide.Right) {

node.Parent.Right = node.Right;

}

if (me == null) {

var bufLeft = node.Left;

var bufRightLeft = node.Right.Left;

var bufRightRight = node.Right.Right;

node.Data = node.Right.Data;

node.Right = bufRightRight;

node.Left = bufRightLeft;

InsertByNode(bufLeft, node, node);

}

else {

node.Right.Parent = node.Parent;

InsertByNode(node.Left, node.Right, node.Right);

}

}

public void Remove(int data) {

var removeNode = Find(data);

if (removeNode != null) {

RemoveByNode(removeNode);

}

}

private void InsertByNode(Node dataNode, Node node, Node parent) {

if (node.Data == null || node.Data == dataNode.Data) {

node.Data = dataNode.Data;

node.Left = dataNode.Left;

node.Right = dataNode.Right;

node.Parent = parent;

return;

}

if (node.Data > dataNode.Data) {

if (node.Left == null) node.Left = new Node();

InsertByNode(dataNode, node.Left, node);

}

else {

if (node.Right == null) node.Right = new Node();

InsertByNode(dataNode, node.Right, node);

}

}

public Node Find(int data) {

return this.Find(data, this.root);

}

public Node Find(int data, Node parent) {

if (parent != null) {

if (data == parent.Data) return parent;

if (data < parent.Data)

return Find(data, parent.Left);

else

return Find(data, parent.Right);

}

return null;

}

}

}

//RandomInput.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1.Properties {

public class RandomInput : Input {

public override int[] input(string inputStr) {

int size = int.Parse(inputStr);

List<int> arr = new List<int>();

Random rnd = new Random();

for (int i = 0; i < size; i++) {

int nextNodeValue = rnd.Next(0, 500);

while (arr.Contains(nextNodeValue))

nextNodeValue = rnd.Next(0, 500);

arr.Add(nextNodeValue);

}

return arr.ToArray();

}

}

}

//KeyboardInput.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1.Properties {

public class KeyboardInput : Input {

public override int[] input(string inputStr) {

StringBuilder oneNumInStr = new StringBuilder();

List<int> intArrayList = new List<int>();

for (int i = 0; i < inputStr.Length; i++) {

if (inputStr.ElementAt(i) <= '9' && inputStr.ElementAt(i) >= '0' || inputStr.ElementAt(i) == '-') {

oneNumInStr.Append(inputStr.ElementAt(i).ToString());

}

else if (oneNumInStr.Length > 0) {

string a = oneNumInStr.ToString();

int b = int.Parse(a);

intArrayList.Add(b);

oneNumInStr.Clear();

}

}

if (oneNumInStr.Length > 0) {

intArrayList.Add(int.Parse(oneNumInStr.ToString()));

oneNumInStr.Clear();

}

int[] clearArr = intArrayList.ToArray

return clearArr;

}

}

}

//FileInput.cs

using Lab1.Properties;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1.Properties {

public class FileInput : Input {

public override int[] input(string inputStr) {

StringBuilder oneNumInStr = new StringBuilder

List<int> intArrayList = new List<int>();

for (int i = 0; i < inputStr.Length; i++) {

if (inputStr.ElementAt(i) <= '9' && inputStr.ElementAt(i) >= '0' || inputStr.ElementAt(i) == '-') {

oneNumInStr.Append(inputStr.ElementAt(i).ToString());

}

else if (oneNumInStr.Length > 0) {

string a = oneNumInStr.ToString();

int b = int.Parse(a);

intArrayList.Add(b);

oneNumInStr.Clear();

}

}

if (oneNumInStr.Length > 0) {

intArrayList.Add(int.Parse(oneNumInStr.ToString()));

oneNumInStr.Clear();

}

int[] clearArr = intArrayList.ToArray();

return clearArr;

}

}

}

//Input.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1.Properties

{

public abstract class Input

{

public abstract int[] input(string inputStr);

}

}

//AddRemoveForm.sc

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab1 {

public partial class AddRemoveForm : Form {

public AddRemoveForm(int left, int top, string capure = "") {

InitializeComponent();

this.StartPosition = FormStartPosition.Manual;

this.Location = new Point(left - Width / 2, top - Height / 2);

this.Text = capure;

}

string RandomTextBoxLastCorrect = "";

private void textAddNumber\_TextChanged(object sender, EventArgs e) {

string sequenceTextNewValue = textAddNumber.Text;

for (int i = 0; i < sequenceTextNewValue.Length; i++) {

if ((sequenceTextNewValue.ElementAt(i) < '0' || sequenceTextNewValue.ElementAt(i) > '9' || (sequenceTextNewValue.Length > 1 && sequenceTextNewValue.ElementAt(0) == '0' && sequenceTextNewValue.ElementAt(1) == '0'))) {

textAddNumber.Text = RandomTextBoxLastCorrect;

return;

}

}

textAddNumber.Text = sequenceTextNewValue;

RandomTextBoxLastCorrect = sequenceTextNewValue;

return;

}

}

}

//MainForm.cs

using Lab1.Properties;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Configuration;

namespace Lab1 {

public enum Tabindex {

Random = 1,

Keyboard,

File

}

public partial class MainForm : Form {

public MainForm() {

InitializeComponent();

}

private void MainForm\_Load(object sender, EventArgs e) {

buttonStartRandom.Click += new EventHandler(this.PlantTree);

buttonStartKeyboard.Click += new EventHandler(this.PlantTree);

buttonStartFile.Click += new EventHandler(this.PlantTree);

buttonRemoveFile.Click += new EventHandler(ButtonRemove\_Click);

buttonRemoveKeyboard.Click += new EventHandler(ButtonRemove\_Click);

buttonRemoveRandom.Click += new EventHandler(ButtonRemove\_Click);

buttonRemoveRandom.Enabled = false;

buttonRemoveKeyboard.Enabled = false;

buttonRemoveFile.Enabled = false;

textRandomOutput.Enabled = false;

textRandomOutput.ReadOnly = true;

textFileOutput.Enabled = false;

textFileOutput.ReadOnly = true;

this.tabControl.SizeMode = TabSizeMode.Fixed;

buttonAddFile.Click += new EventHandler(this.ButtonAdd\_Click);

buttonAddKeyboard.Click += new EventHandler(this.ButtonAdd\_Click);

buttonAddRandom.Click += new EventHandler(this.ButtonAdd\_Click);

buttonStartRandom.Click += new EventHandler(this.ActivateElements);

buttonStartKeyboard.Click += new EventHandler(this.ActivateElements);

buttonStartFile.Click += new EventHandler(this.ActivateElements);

saveResultsToolStripMenuItem.Click += new EventHandler(SavelResults);

saveInnerDataToolStripMenuItem.Click += new EventHandler(SavelData);

openToolStripMenuItem.Click+= new EventHandler(ButtonOpenFile\_Click);

showGreetingsToolStripMenuItem.Click += new EventHandler(ShowGreetingChange);

aboutToolStripMenuItem.Click += new EventHandler(delegate (object s, EventArgs eve) { ShowGreetingForm(); });

}

private void ShowGreeting(object sender, EventArgs e) {

if (bool.Parse(ConfigurationManager.AppSettings["showGreeting"])) {

ShowGreetingForm();

showGreetingsToolStripMenuItem.Checked = true;

}

else {

showGreetingsToolStripMenuItem.Checked = false;

}

}

private void ShowGreetingForm() {

new lab1.MessageForm(

Left + Width / 2,

Top + Height / 2,

"Студент СПБГТИ(ТУ) \n" +

"Старков Силантий Денисович \n" +

"403 группа\nЛабораторная работа №1\n" +

"Вариант №1\nПостроение простого бинарного дерева").ShowDialog();

}

#region SAVE

private void SavelData(object sender, EventArgs e) {

if (tree != null && tree.root != null) {

SaveFileDialog saveFileDialog = new SaveFileDialog {

Filter = "Text files(\*.txt)|\*.txt|All files(\*.\*)|\*.\*"

};

if (saveFileDialog.ShowDialog() == DialogResult.Cancel)

return;

string filename = saveFileDialog.FileName;

string data;

if (tabControl.SelectedTab.TabIndex == (int)Tabindex.Random) {

data = textRandomOutput.Text;

}

else if (tabControl.SelectedTab.TabIndex == (int)Tabindex.Keyboard) {

data = textKeyboardInput.Text;

}

else {

data = textFileOutput.Text;

}

System.IO.File.WriteAllText(filename, data);

MessageBox.Show("Файл сохранен!");

}

}

private void SavelResults(object sender, EventArgs e) {

if (tree != null && tree.root != null) {

var saveResultString = lab1.Printer.PrintBinaryTree(tree.root);

SaveFileDialog saveFileDialog = new SaveFileDialog {

Filter = "Text files(\*.txt)|\*.txt|All files(\*.\*)|\*.\*"

};

if (saveFileDialog.ShowDialog() == DialogResult.Cancel)

return;

string treePrint = "";

for (int i = 0; i < saveResultString.Count; i++)

treePrint += saveResultString[i].ToString() + '\n';

System.IO.File.WriteAllText(saveFileDialog.FileName, treePrint);

MessageBox.Show("Файл сохранен");

for (int i = 0; i < saveResultString.Count; i++) {

Console.WriteLine(saveResultString[i]);

}

}

}

#endregion

#region SETTINGS

private void ShowGreetingChange(object sender, EventArgs e) {

var config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);

bool show = !bool.Parse(ConfigurationManager.AppSettings["showGreeting"]);

config.AppSettings.Settings["showGreeting"].Value = (show).ToString();

config.Save();

showGreetingsToolStripMenuItem.Checked = show;

ConfigurationManager.RefreshSection("appSettings");

}

#endregion

#region TREE

BinaryTree tree = null;

private void GetInputWay(ref Input inputWay, ref string sequenceTextForStart) {

switch (tabControl.SelectedTab.TabIndex) {

case (int)Tabindex.Random:

inputWay = new RandomInput();

sequenceTextForStart = textRandomInput.Text;

break;

case (int)Tabindex.Keyboard:

inputWay = new KeyboardInput();

sequenceTextForStart = textKeyboardInput.Text;

break;

case (int)Tabindex.File:

inputWay = new FileInput();

sequenceTextForStart = textFileOutput.Text;

break;

}

}

private void CreateTree(int[] arr) {

tree = new BinaryTree();

for (int i = 0; i < arr.Length; i++) {

tree.Insert(arr[i]);

}

}

private void PlantTree(object sender, EventArgs e) {

string sequenceTextForStart = "";

Input inputWay = null;

GetInputWay(ref inputWay, ref sequenceTextForStart);

if (sequenceTextForStart.Length > 0) {

int[] arr = inputWay.input(sequenceTextForStart);

CreateTree(arr);

if (tabControl.SelectedTab.TabIndex == (int)Tabindex.Random) {

textRandomOutput.Text = "";

textRandomOutput.Enabled = true;

for (int i = 0; i < arr.Length; i++) {

textRandomOutput.Text += arr[i] + " ";

}

}

else if (tabControl.SelectedTab.TabIndex == (int)Tabindex.Keyboard) {

textKeyboardInput.ReadOnly = true;

}

pictureBox.Size = new Size(scrollPanel.Width, scrollPanel.Height);

ClearPictureBox();

leftmost = Point.Empty;

Print(tree.root, pictureBox);

ClearPictureBox();

Print(tree.root, pictureBox, default, leftmost);

}

}

private void RepaintTree() {

ClearPictureBox();

pictureBox.Size = scrollPanel.MinimumSize;

Print(tree.root, pictureBox);

pictureBox.Update();

NodeForDraw.DrawLine(pictureBox);

}

Point leftmost = Point.Empty;

Point correction = new Point(40, 40);

public void Print(Node node, PictureBox pictureBox, Color color = default, Point rootLocation = new Point()) {

NodeForDraw.treeTab = tabControl.SelectedTab.TabIndex;

if (node.Data != null) {

if (node.Parent == null) {

Console.WriteLine("ROOT:" + node.Data);

NodeForDraw nodeDraw = new NodeForDraw(node.Data.ToString(), new Point(Math.Abs(rootLocation.X) + pictureBox.Width / 2, 0), Color.FromArgb(59, 131, 189));

NodeForDraw.drawList.Add(nodeDraw);

pictureBox.Controls.Add(nodeDraw);

nodeDraw.BringToFront();

}

else {

Point parentLocation = Point.Empty;

NodeForDraw parentNode = new NodeForDraw();

for (int i = 0; i < NodeForDraw.drawList.Count; i++) {

if (NodeForDraw.drawList[i].Text == node.Parent.Data.ToString()) {

parentNode = NodeForDraw.drawList[i];

break;

}

}

parentLocation = parentNode.Location;

NodeForDraw nodeDraw = new NodeForDraw();

if (node.Parent.Left == node) {

Console.WriteLine("Left for " + node.Parent.Data + " --> " + node.Data);

if (color == Color.FromArgb(255, 46, 99)) {

}

nodeDraw = new NodeForDraw(node.Data.ToString(), new Point(parentLocation.X - correction.X, parentLocation.Y + +20 + correction.Y), color);

if (nodeDraw.Location.X - nodeDraw.Width < 0) {

if (nodeDraw.Location.X - nodeDraw.Width < leftmost.X)

leftmost = new Point(Math.Abs(nodeDraw.Location.X - nodeDraw.Width), 0);

}

if ((nodeDraw.Location.Y + nodeDraw.Height) > pictureBox.Height) {

pictureBox.Height = nodeDraw.Location.Y + nodeDraw.Height \* 2;

}

NodeForDraw.drawList.Add(nodeDraw);

pictureBox.Controls.Add(nodeDraw);

nodeDraw.BringToFront();

}

if (node.Parent.Right == node) {

Console.WriteLine("Right for " + node.Parent.Data + " --> " + node.Data);

nodeDraw = new NodeForDraw(node.Data.ToString(), new Point(parentLocation.X + correction.X, parentLocation.Y + 20 + correction.Y), color);

if ((nodeDraw.Location.X + nodeDraw.Width) > pictureBox.Width) {

pictureBox.Width = nodeDraw.Location.X + nodeDraw.Width;

}

if ((nodeDraw.Location.Y + nodeDraw.Height) > pictureBox.Height) {

pictureBox.Height = nodeDraw.Location.Y + nodeDraw.Height \* 2;

}

NodeForDraw.drawList.Add(nodeDraw);

pictureBox.Controls.Add(nodeDraw);

nodeDraw.BringToFront();

}

correction.X += 40;

correction.Y += 40;

List<NodeForDraw> buffLine = new List<NodeForDraw> {

parentNode,

nodeDraw

};

NodeForDraw.nodeListForLines.Add(buffLine);

}

if (node.Left != null && node.Parent == null) {

color = Color.FromArgb(255, 46, 99);

Print(node.Left, pictureBox, color);

}

else if (node.Left != null) {

Print(node.Left, pictureBox, color);

}

if (node.Right != null && node.Parent == null) {

color = Color.FromArgb(37, 42, 52);

Print(node.Right, pictureBox, color);

}

else if (node.Right != null) {

Print(node.Right, pictureBox, color);

}

}

NodeForDraw.DrawLine(pictureBox);

correction = new Point(30, 30);

}

private void ClearPictureBox() {

pictureBox.Controls.Clear();

NodeForDraw.drawList.Clear();

NodeForDraw.nodeListForLines.Clear();

Graphics gr = pictureBox.CreateGraphics();

gr.Clear(pictureBox.BackColor);

pictureBox.Update();

}

class NodeForDraw : Label {

public static List<List<NodeForDraw>> nodeListForLines = new List<List<NodeForDraw>>();

public static List<NodeForDraw> drawList = new List<NodeForDraw>();

public static int? treeTab = null;

public NodeForDraw() { }

public NodeForDraw(string text, Point loc, Color color) {

AutoSize = true;

Location = loc;

BackColor = color;

Font = new Font("Segoe UI", 20F, FontStyle.Regular, GraphicsUnit.Point);

ForeColor = Color.White;

Text = text;

this.MouseDown += new MouseEventHandler((object sender, MouseEventArgs e) => { isMouseDown = true; cursorStartinLocation = Cursor.Position; });

this.MouseUp += new MouseEventHandler((object sender, MouseEventArgs e) => isMouseDown = false);

this.MouseMove += new MouseEventHandler(MoveNode);

}

Point cursorStartinLocation = new Point();

bool isMouseDown = false;

void MoveNode(object sender, MouseEventArgs e) {

NodeForDraw nodeToMove = (NodeForDraw)sender;

if (isMouseDown) {

Location = new Point(this.Location.X + (Cursor.Position.X - cursorStartinLocation.X), this.Location.Y + (Cursor.Position.Y - cursorStartinLocation.Y));

cursorStartinLocation = Cursor.Position;

//repaint lines

for (int i = 0; i < nodeListForLines.Count; i++) {

List<NodeForDraw> linesToMove = new List<NodeForDraw>();

if (nodeListForLines[i][0] == nodeToMove) {

nodeListForLines[i][0].Location = new Point(this.Location.X + (Cursor.Position.X - cursorStartinLocation.X), this.Location.Y + (Cursor.Position.Y - cursorStartinLocation.Y));

}

}

DrawLine((PictureBox)nodeToMove.Parent);

}

}

public static void DrawLine(PictureBox pictureBox) {

Graphics gr = pictureBox.CreateGraphics();

gr.Clear(pictureBox.BackColor);

pictureBox.Update();

if (nodeListForLines.Count != 0) {

for (int i = 0; i < nodeListForLines.Count; i++) {

gr.DrawLine(new Pen(nodeListForLines[i][0].BackColor, 2),

new Point(nodeListForLines[i][0].Location.X + nodeListForLines[i][0].Width / 2, nodeListForLines[i][0].Location.Y + nodeListForLines[i][0].Height / 2),

new Point(nodeListForLines[i][1].Location.X + nodeListForLines[i][1].Width / 2, nodeListForLines[i][1].Location.Y + nodeListForLines[i][1].Height / 2));

}

}

}

}

#endregion

#region WINFORMS

private void ScrollPanel\_SizeChanged(object sender, EventArgs e) {

pictureBox.MinimumSize = scrollPanel.Size;

NodeForDraw.DrawLine(pictureBox);

}

private void PictureBox\_Paint(object sender, PaintEventArgs e) {

NodeForDraw.DrawLine(pictureBox);

}

private void CleanBtn\_Click(object sender, EventArgs e) {

ClearPictureBox();

tree = new BinaryTree();

if (tabControl.SelectedTab.TabIndex == (int)Tabindex.Random) {

textRandomOutput.Text = "";

textRandomInput.Text = "";

buttonRemoveRandom.Enabled = false;

}

else if (tabControl.SelectedTab.TabIndex == (int)Tabindex.Keyboard) {

textKeyboardInput.Text = "";

buttonRemoveKeyboard.Enabled = false;

textKeyboardInput.Enabled = true;

textKeyboardInput.ReadOnly = false;

}

else {

textFileOutput.Text = "";

buttonRemoveFile.Enabled = false;

}

}

private void ActivateElements(object sender, EventArgs e) {

if (tabControl.SelectedTab.TabIndex == (int)Tabindex.Random) {

if (textRandomOutput.Text != "") {

buttonRemoveRandom.Enabled = true;

}

else {

buttonRemoveRandom.Enabled = false;

}

}

else if (tabControl.SelectedTab.TabIndex == (int)Tabindex.Keyboard) {

if (textKeyboardInput.Text != "") {

buttonRemoveKeyboard.Enabled = true;

}

else {

buttonRemoveKeyboard.Enabled = false;

}

}

else if (tabControl.SelectedTab.TabIndex == (int)Tabindex.File) {

if (textFileOutput.Text != "") {

buttonRemoveFile.Enabled = true;

}

else {

buttonRemoveFile.Enabled = false;

}

}

}

#endregion

#region ADD/REMOVE

AddRemoveForm addRemoveForm = null;

private void ButtonAdd\_Click(object sender, EventArgs e) {

addRemoveForm = new AddRemoveForm(this.Left + this.Width / 2, this.Top + this.Height / 2, "Adding Node");

addRemoveForm.buttonAdd.Click += new EventHandler(this.AddToTree);

addRemoveForm.ShowDialog();

}

private void AddToTree(object sender, EventArgs e) {

if (addRemoveForm.textAddNumber.Text.Length > 0) {

int newNode = int.Parse(addRemoveForm.textAddNumber.Text);

if (tree == null || tree.Find(newNode) == null) {

if (tabControl.SelectedTab.TabIndex == (int)Tabindex.Random) {

textRandomOutput.Text += addRemoveForm.textAddNumber.Text + " ";

if (textRandomInput.Text.Length > 0)

textRandomInput.Text = (int.Parse(textRandomInput.Text) + 1).ToString();

else

textRandomInput.Text = 1.ToString();

}

else if (tabControl.SelectedTab.TabIndex == (int)Tabindex.Keyboard) {

textKeyboardInput.Text += " " + addRemoveForm.textAddNumber.Text + " ";

}

else {

textFileOutput.Text += " " + addRemoveForm.textAddNumber.Text + " ";

}

if (tree == null || NodeForDraw.treeTab != tabControl.SelectedTab.TabIndex)

tree = new BinaryTree();

tree.Insert(newNode);

ClearPictureBox();

Print(tree.root, pictureBox);

}

else {

lab1.MessageForm mes = new lab1.MessageForm(Left + Width / 2, Top + Height / 2, "Элемент уже введен.");

mes.ShowDialog();

}

}

addRemoveForm.Close();

}

private void ButtonRemove\_Click(object sender, EventArgs e) {

addRemoveForm = new AddRemoveForm(Left + Width / 2, Top + Height / 2, "Removing Node");

addRemoveForm.buttonAdd.Click += new EventHandler(this.RemoveFromForm);

addRemoveForm.ShowDialog();

}

private void RemoveFromForm(object sender, EventArgs e) {

if (addRemoveForm.textAddNumber.Text.Length > 0) {

int newNode = int.Parse(addRemoveForm.textAddNumber.Text);

if (!(tree is null) && tree.Find(newNode) != null) {

if (tabControl.SelectedTab.TabIndex == (int)Tabindex.Random) {

if (textRandomOutput.Text.IndexOf(addRemoveForm.textAddNumber.Text + " ") != -1) {

textRandomOutput.Text = textRandomOutput.Text.Replace(addRemoveForm.textAddNumber.Text + " ", " ");

if (textRandomInput.Text.Length > 0)

textRandomInput.Text = (int.Parse(textRandomInput.Text) - 1).ToString();

RemoveFromTree(newNode, textRandomOutput.Text);

}

}

else if (tabControl.SelectedTab.TabIndex == (int)Tabindex.Keyboard) {

if (textKeyboardInput.Text.IndexOf(addRemoveForm.textAddNumber.Text) != -1) {

textKeyboardInput.Text = textKeyboardInput.Text.Replace(addRemoveForm.textAddNumber.Text, "");

RemoveFromTree(newNode, textKeyboardInput.Text);

}

}

else {

if (textFileOutput.Text.IndexOf(addRemoveForm.textAddNumber.Text) != -1) {

textFileOutput.Text = textFileOutput.Text.Replace(addRemoveForm.textAddNumber.Text, "");

RemoveFromTree(newNode, textFileOutput.Text);

}

}

}

else {

lab1.MessageForm mes = new lab1.MessageForm(Left + Width / 2, Top + Height / 2, "Элемент не найден.");

mes.ShowDialog();

}

}

addRemoveForm.Close();

}

private void RemoveFromTree(int data, string newTree) {

if (data != tree.root.Data) {

tree.Remove(data);

RepaintTree();

}

else {

ClearPictureBox();

CreateTree(new KeyboardInput().input(newTree));

Print(tree.root, pictureBox);

}

}

#endregion

#region INPUT

string KeyboardTextBoxLastCorrect;

string RandomTextBoxLastCorrect;

private void TextKeyboardInput\_TextChanged(object sender, EventArgs e) {

string sequenceTextNewValue = textKeyboardInput.Text;

if (sequenceTextNewValue.Length > 0) {

for (int i = 0; i < sequenceTextNewValue.Length; i++) {

if (sequenceTextNewValue.Length > 0 && (sequenceTextNewValue.ElementAt(i) < '0' || sequenceTextNewValue.ElementAt(i) > '9') && sequenceTextNewValue.ElementAt(i) != ' ') {

textKeyboardInput.Text = KeyboardTextBoxLastCorrect;

return;

}

}

while (sequenceTextNewValue.Contains(" "))

sequenceTextNewValue = sequenceTextNewValue.Replace(" ", " ");

if (sequenceTextNewValue.ElementAt(0) == ' ') {

sequenceTextNewValue = sequenceTextNewValue.Remove(0, 1);

}

textKeyboardInput.Text = sequenceTextNewValue;

KeyboardTextBoxLastCorrect = sequenceTextNewValue;

return;

}

}

private void ButtonOpenFile\_Click(object sender, EventArgs e) {

tabControl.SelectTab((int)Tabindex.File-1);

OpenFileDialog openFileDialog = new OpenFileDialog {

Filter = "Text files(\*.txt)|\*.txt|All files(\*.\*)|\*.\*"

};

if (openFileDialog.ShowDialog() == DialogResult.Cancel)

return;

string sequenceTextNewValue = System.IO.File.ReadAllText(openFileDialog.FileName);

for (int i = 0; i < sequenceTextNewValue.Length; i++) {

if (sequenceTextNewValue.Length > 0 && (sequenceTextNewValue.ElementAt(i) < '0' || sequenceTextNewValue.ElementAt(i) > '9') && sequenceTextNewValue.ElementAt(i) != ' ') {

textFileOutput.Text = "";

MessageBox.Show("Ошибка: Неверный формат записи файла");

return;

}

}

while (sequenceTextNewValue.Contains(" "))

sequenceTextNewValue = sequenceTextNewValue.Replace(" ", " ");

textFileOutput.Text = sequenceTextNewValue;

textFileOutput.Enabled = true;

PlantTree(sender, e);

return;

}

private void TextRandomInput\_TextChanged(object sender, EventArgs e) {

string sequenceTextNewValue = textRandomInput.Text;

for (int i = 0; i < sequenceTextNewValue.Length; i++) {

if ((sequenceTextNewValue.ElementAt(i) < '0' || sequenceTextNewValue.ElementAt(i) > '9')) {

textRandomInput.Text = RandomTextBoxLastCorrect;

return;

}

}

textRandomInput.Text = sequenceTextNewValue;

RandomTextBoxLastCorrect = sequenceTextNewValue;

return;

}

private void TextRandomOutput\_TextChanged(object sender, EventArgs e) {

if (textRandomOutput.Text.Length > 0)

buttonRemoveRandom.Enabled = true;

else

buttonRemoveRandom.Enabled = false;

}

#endregion

#region TREE MOVING

private bool Dragging;

private int xPos;

private int yPos;

private void PictureBox\_MouseMove(object sender, MouseEventArgs e) {

Control c = sender as Control;

if (Dragging && c != null) {

c.Top = e.Y + c.Top - yPos;

c.Left = e.X + c.Left - xPos;

}

}

private void PictureBox\_MouseDown(object sender, MouseEventArgs e) {

if (e.Button == MouseButtons.Left) {

Dragging = true;

xPos = e.X;

yPos = e.Y;

}

}

private void PictureBox\_MouseUp(object sender, MouseEventArgs e) {

Dragging = false;

}

#endregion

}

}

//MessageForm.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace lab1 {

public partial class MessageForm : Form {

public MessageForm(int left, int top, string message) {

InitializeComponent();

this.StartPosition = FormStartPosition.Manual;

this.textMessage.Text = message;

Point magin = textMessage.Location;

this.Size = new Size(magin.X\*2+textMessage.Width, magin.Y \* 3 + textMessage.Height);

this.MaximumSize = this.Size;

this.MinimumSize = this.Size;

this.FormBorderStyle = FormBorderStyle.FixedSingle;

this.Location = new Point(left - Width / 2, top - Height / 2);

}

private void MessageForm\_Load(object sender, EventArgs e) {

}

}

}

//Printer.sc

using Lab1;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab1 {

class Printer {

public static List<StringBuilder> PrintBinaryTree(Node node) {

List<StringBuilder> saveResultString = new List<StringBuilder> {

new StringBuilder()

};

Queue<Node> nodes = new Queue<Node>();

int depth = Depth(node);

int numberOfSpaces;

int numberOfBranchParts;

int elemPerLevel;

string levelPrint = "";

string leftBranch;

string rightBranch;

int n = FindMaxData(node);

nodes.Enqueue(node);

for (int level = 0; level < depth; level++) {

numberOfSpaces = (Convert.ToInt32(Math.Pow(2, (depth - 1 - level))) - 1) \* n;

numberOfBranchParts = ((numberOfSpaces / n - 1) / 2) \* n;

elemPerLevel = Convert.ToInt32(Math.Pow(2, level));

leftBranch = string.Concat(string.Concat(Enumerable.Repeat(" ", n - 1)), "┌", string.Concat(Enumerable.Repeat("─", numberOfBranchParts)));

rightBranch = string.Concat(string.Concat(Enumerable.Repeat("─", numberOfBranchParts)), "┐", string.Concat(Enumerable.Repeat(" ", n - 1)));

for (int elemPrint = 0; elemPrint < elemPerLevel; elemPrint++) {

node = nodes.Dequeue();

if (node != null) {

nodes.Enqueue(node.Left);

nodes.Enqueue(node.Right);

}

else {

nodes.Enqueue(null);

nodes.Enqueue(null);

}

if (node != null) {

if (node.Left != null) {

levelPrint = string.Concat(levelPrint, string.Concat(Enumerable.Repeat(" ", numberOfBranchParts)), leftBranch);

}

else {

levelPrint = string.Concat(levelPrint, string.Concat(Enumerable.Repeat(" ", numberOfSpaces)));

}

levelPrint = string.Concat(levelPrint, string.Concat(Enumerable.Repeat(" ", n - node.Data.ToString().Length)), node.Data);

if (node.Right != null) {

levelPrint = string.Concat(levelPrint, rightBranch, string.Concat(Enumerable.Repeat(" ", numberOfBranchParts)));

}

else {

levelPrint = string.Concat(levelPrint, string.Concat(Enumerable.Repeat(" ", numberOfSpaces)));

}

levelPrint = string.Concat(levelPrint, string.Concat(Enumerable.Repeat(" ", n)));

}

else {

levelPrint = string.Concat(levelPrint, string.Concat(Enumerable.Repeat(" ", numberOfSpaces)));

levelPrint = string.Concat(levelPrint, string.Concat(Enumerable.Repeat(" ", n)));

levelPrint = string.Concat(levelPrint, string.Concat(Enumerable.Repeat(" ", numberOfSpaces)));

levelPrint = string.Concat(levelPrint, string.Concat(Enumerable.Repeat(" ", n)));

}

}

saveResultString.Add(new StringBuilder(levelPrint));

levelPrint = "";

}

nodes.Clear();

bool clear = true;

for (int element = 0; element < saveResultString.Max(s => s.Length); ++element) {

for (int stringNumber = 0; stringNumber < saveResultString.Count; stringNumber++) {

if (saveResultString[stringNumber].Length > element && saveResultString[stringNumber][element] != ' ' && saveResultString[stringNumber][element] != '─') {

clear = false;

break;

}

}

if (clear) {

for (int stringNumber = 0; stringNumber < saveResultString.Count; stringNumber++) {

if (saveResultString[stringNumber].Length > element)

saveResultString[stringNumber] = saveResultString[stringNumber].Remove(element, 1);

}

element--;

}

clear = true;

}

return saveResultString;

}

public static int Depth(Node node) {

int depth = 0;

if (node != null) {

int leftD = Depth(node.Left);

int rightD = Depth(node.Right);

depth = Math.Max(leftD, rightD) + 1;

}

return depth;

}

public static int FindMaxData(Node node) {

while (node.Right != null) {

node = node.Right;

}

return node.Data.ToString().Length;

}

}

}

//App.config

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />

</startup>

<appSettings>

<add key ="showGreeting" value="true"/>

</appSettings>

</configuration>

//BinaryTree.cs

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Lab1;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1.Tests {

[TestClass()]

public class BinaryTreeTests {

private static BinaryTree tree;

[TestMethod()]

public void Test\_1\_Insert() {

tree = new BinaryTree {

root = new Node(55)

};

int newLeftNodeData = 45;

tree.Insert(newLeftNodeData);

Assert.AreEqual(tree.root.Left.Data, newLeftNodeData);

}

[TestMethod()]

public void Test\_2\_Remove() {

tree.Remove((int)tree.root.Left.Data);

Assert.AreEqual(null, tree.root.Left);

}

[TestMethod()]

public void Test\_3\_Find() {

Assert.AreEqual(null, tree.Find(45));

}

[TestMethod()]

public void Test\_4\_Duplicate() {

tree.Insert(2);

tree.Insert(2);

Assert.AreEqual(null, tree.root.Left.Left);

Assert.AreEqual(null, tree.root.Left.Right);

}

}

}

[--- Конец программы.]