УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №5.2

по предмету «Основы алгоритмизации и программирования»

Вариант 18

Выполнил:

Егоров А.С.

Гр. 351005

Проверил:

Данилова Г. В.

Минск 2024

**Задание:**

Деревья. Вывести номера вершин, для которых высота левого

поддерева не равна высоте правого поддерева. Дерево

визуализировать.

**Код программы Delphi:**

**MainUnit.pas**

Unit MainUnit5\_2;

Interface

Uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Buttons, System.ImageList,

Vcl.ImgList, Vcl.Grids, Vcl.StdCtrls, Vcl.Menus,

InstructionUnit5\_2, AboutTheDeveloperUnit5\_2, BackendUnit5\_2,

ExitUnit5\_2, System.Generics.Collections, OutputUnit5\_2;

Type

// форма

TuVCLMain = Class(TForm)

LbTaskInfo: TLabel;

LbSizeInfo: TLabel;

ESize: TEdit;

BitBtAcceptSize: TButton;

StrGrList: TStringGrid;

ImageList1: TImageList;

OpenDialog1: TOpenDialog;

SaveDialog1: TSaveDialog;

MainMenu1: TMainMenu;

BtFile: TMenuItem;

BtOpenFile: TMenuItem;

BtSaveFile: TMenuItem;

BtInstruction: TMenuItem;

BtAboutTheDeveloper: TMenuItem;

LbElementsInfo: TLabel;

BitBtFindRoots: TBitBtn;

BitBtShowTree: TBitBtn;

LValuesOfTree: TLabel;

Procedure ESizeKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure ESizeKeyPress(Sender: TObject; Var Key: Char);

Procedure BtInstructionClick(Sender: TObject);

Procedure BtAboutTheDeveloperClick(Sender: TObject);

Procedure ESizeChange(Sender: TObject);

Procedure StrGrListKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure BitBtAcceptSizeClick(Sender: TObject);

Procedure StrGrListKeyPress(Sender: TObject; Var Key: Char);

Procedure BitBtFindRootsClick(Sender: TObject);

Procedure BtOpenFileClick(Sender: TObject);

Procedure BtSaveFileClick(Sender: TObject);

Procedure StrGrListKeyUp(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure BitBtFindRootsKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Function FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Procedure FormCreate(Sender: TObject);

Procedure FormDestroy(Sender: TObject);

Procedure BitBtShowTreeClick(Sender: TObject);

Private

Size: Integer;

IsFileSaved: Boolean;

IsArrayFilled: Boolean;

IsBitBtFindPressed: Boolean;

WasChanges: Boolean;

BufferHandler: TBufferHandler;

ListOfValues: TList<Integer>;

Public

{ Public declarations }

End;

Var

UVCLMain: TuVCLMain;

Implementation

{$R \*.dfm}

Procedure TuVCLMain.BtAboutTheDeveloperClick(Sender: TObject);

Begin

Application.CreateForm(TuVCLAboutTheDeveloper, UVCLAboutTheDeveloper);

UVCLAboutTheDeveloper.Show;

uVCLAboutTheDeveloper.Destroy();

End;

Procedure TuVCLMain.BtInstructionClick(Sender: TObject);

Begin

Application.CreateForm(TuVCLInstruction, UVCLInstruction);

UVCLInstruction.ShowModal;

UVCLInstruction.Destroy;

End;

Procedure TuVCLMain.BtOpenFileClick(Sender: TObject);

Var

FileReader: TFileReader;

I: Integer;

Size: Integer;

List: TList<Integer>;

A: Word;

B: TShiftState;

Begin

List := TList<Integer>.Create;

If OpenDialog1.Execute() Then

Begin

FileReader := TFileReader.Create();

FileReader.FileName := OpenDialog1.FileName;

FileReader.CheckFile();

// TODO статусы

If FileReader.Status = FsGood Then

Begin

Size := FileReader.ReadSize();

If (FileReader.Status = FsGood) Then

Begin

ESize.Text := IntToStr(Size);

BitBtAcceptSize.Click;

List := FileReader.ReadList(Size);

If (FileReader.Status = FsGood) Then

Begin

For I := 1 To Size Do

Begin

StrGrList.Cells[I, 0] := IntToStr(I);

StrGrList.Cells[I, 1] := IntToStr(List[I - 1]);

End;

End;

StrGrListKeyUp(Sender, A, B);

End

Else

MessageBox(UVCLMain.Handle, ListOfMessages[FileReader.Status],

'Ой-йой', MB\_ICONERROR);

End

Else

MessageBox(UVCLMain.Handle, ListOfMessages[FileReader.Status],

'Ой-йой', MB\_ICONERROR);

End;

End;

Procedure TuVCLMain.BtSaveFileClick(Sender: TObject);

Var

FileWriter: TFileWriter;

Begin

If SaveDialog1.Execute() Then

Begin

FileWriter := TFileWriter.Create();

FileWriter.FileName := SaveDialog1.FileName;

FileWriter.CheckFile();

If FileWriter.Status = FsGood Then

Begin

FileWriter.WriteList(ListOfValues);

If FileWriter.Status <> FsGood Then

Begin

MessageBox(UVCLMain.Handle, ListOfMessages[FileWriter.Status],

'Ой-йой', MB\_ICONERROR);

End

Else

IsFileSaved := True;

End

Else

MessageBox(UVCLMain.Handle, ListOfMessages[FileWriter.Status],

'Ой-йой', MB\_ICONERROR);

FileWriter.Destroy();

End;

End;

Procedure TuVCLMain.BitBtAcceptSizeClick(Sender: TObject);

Const

MAX\_SIZE: Integer = 1000;

MIN\_SIZE: Integer = 0;

Var

I: Integer;

Begin

// creating Grid

Size := StrToInt(ESize.Text);

If WasChanges Then

Begin

If (MIN\_SIZE < Size) And (Size < MAX\_SIZE) Then

Begin

// list1

StrGrList.RowCount := 2;

StrGrList.ColCount := Size + 1;

StrGrList.FixedCols := 1;

StrGrList.FixedRows := 1;

StrGrList.Cells[0, 0] := '№';

StrGrList.Cells[0, 1] := 'Элемент';

For I := 1 To Size Do

Begin

StrGrList.Cells[I, 0] := IntToStr(I);

StrGrList.Cells[I, 1] := '';

End;

StrGrList.Enabled := True;

StrGrList.Visible := True;

BitBtAcceptSize.Enabled := False;

End

Else

MessageBox(UVCLMain.Handle, 'Размер не соответствует границам!

Проверьте данные.', 'Ой-йой', MB\_ICONERROR);

End;

WasChanges := False;

End;

Procedure TuVCLMain.BitBtFindRootsClick(Sender: TObject);

Var

I: Integer;

Begin

If Tree <> Nil Then

Tree.Destroy();

Tree := TTree.Create();

For I := 1 To Size Do

Begin

Tree.Add(StrToInt(StrGrList.Cells[I, 1]));

End;

ListOfValues := Tree.FindValues();

If ListOfValues.Count = 0 Then

LValuesOfTree.Caption := 'Таких вершин не существует в данном дереве!'

Else

LValuesOfTree.Caption := '';

For I := 1 To ListOfValues.Count Do

Begin

LValuesOfTree.Caption := LValuesOfTree.Caption + IntToStr(ListOfValues[I - 1]) + ' ';

End;

LValuesOfTree.Visible := True;

BitbtShowTree.Enabled := True;

BitbtShowTree.Visible := True;

BtSaveFile.Enabled := True;

IsFileSaved := False;

IsBitBtFindPressed := True;

End;

Procedure TuVCLMain.BitBtShowTreeClick(Sender: TObject);

Var

I: Integer;

Begin

If Not IsBitBtFindPressed Then

Begin

If Tree <> Nil Then

Tree.Destroy();

Tree := TTree.Create();

For I := 1 To Size Do

Begin

Tree.Add(StrToInt(StrGrList.Cells[I, 1]));

End;

End;

Application.CreateForm(TuVCLOutputTree, UVCLOutputTree);

UVCLOutputTree.ShowModal;

UVCLOutputTree.Destroy;

End;

Procedure TuVCLMain.FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Begin

If Not IsBitBtFindPressed Or IsFileSaved Then

Begin

Application.CreateForm(TuVCLExit, UVCLExit);

UVCLExit.ShowModal;

CanClose := UVCLExit.GetStatus();

UVCLExit.Destroy();

End

Else If IsBitBtFindPressed Then

Begin

Repeat

ExitCode := MessageBox(UVCLMain.Handle, 'Сохранить данные в файл

перед выходом?', 'Подверждение',

MB\_ICONQUESTION + MB\_YESNOCANCEL);

If ExitCode = ID\_YES Then

Begin

BtSaveFileClick(UVCLMain);

CanClose := True;

End

Else If ExitCode = ID\_NO Then

CanClose := True

Else

CanClose := False;

Until IsFileSaved Or (ExitCode = ID\_NO) Or (ExitCode = ID\_CANCEL);

End;

End;

Procedure TuVCLMain.FormCreate(Sender: TObject);

Begin

BufferHandler := TBufferHandler.Create();

End;

Procedure TuVCLMain.FormDestroy(Sender: TObject);

Begin

BufferHandler.Destroy();

End;

Function TuVCLMain.FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

End;

End.

**BackendUnit.pas**

Unit BackendUnit5\_2;

Interface

Uses System.SysUtils, System.Generics.Collections;

Type

PtrRoot = ^TRoot;

TRoot = Record

FValue: Integer;

FRight: PtrRoot;

FLeft: PtrRoot;

FPrev: PtrRoot;

Procedure Init(Value: Integer);

End;

TTree = Class

Private

FPtrHeader: PtrRoot;

Function FindValues(Var Tree: PtrRoot; Var List: TList<Integer>)

: Integer; Overload;

Public

Procedure Add(Value: Integer);

Function GetRight(Root : PtrRoot): PtrRoot;

Function GetLeft(Root : PtrRoot) : PtrRoot;

Property Head : PtrRoot read FPtrHeader;

Function FindValues(): TList<Integer>; Overload;

End;

TTypes = (TpInteger, TpUInteger, TpReal, TpString);

TBufferHandler = Class

Private

FEditText: String;

Function CountSymbol(Const Symbol: Char): Integer;

Public

Function CheckInput(Const InputType: TTypes): Boolean;

Function CheckRange(Const MIN, MAX; Const InputType: TTypes): Boolean;

Procedure DeleteLeadingZeros(Const InputType: TTypes);

Property EditText: String Read FEditText Write FEditText;

End;

TFileStatus = (FsGood, FsNotFound, FsNotTxt, FsNotReadable, FsNotWritable,

FsEmpty, FsWrongData, FsUnexpected);

TFileReader = Class

Private

FFileName: String;

FInFile: TextFile;

FFileStatus: TFileStatus;

Function IsFileTxt(): Boolean;

Function IsFileReadable(): Boolean;

Function IsEmpty(): Boolean;

Procedure SetFileName(Const FFileName: String);

Public

Property Status: TFileStatus Read FFileStatus;

Property FileName: String Read FFileName Write SetFileName;

Procedure CheckFile();

Function ReadSize(): Integer;

Function ReadList(Const Size: Integer): TList<Integer>;

End;

TFileWriter = Class

Private

FFileName: String;

FOutFile: TextFile;

FFileStatus: TFileStatus;

Function IsFileTxt(): Boolean;

Function IsFileWritable(): Boolean;

Procedure SetFileName(Const FFileName: String);

Public

Constructor Create();

Property Status: TFileStatus Read FFileStatus;

Property FileName: String Read FFileName Write SetFileName;

Procedure CheckFile();

Procedure WriteList(Var List: TList<Integer>);

End;

TListOfMessages = Array [TFileStatus] Of PWideChar;

Const

ListOfMessages: TListOfMessages = ('Информация записана!',

'Файл не найден! Повторите ещё раз.',

'Файл не текстовый! Повторите ещё раз.',

'Файл не доступен для чтения! Повторите ещё раз.',

'Файл не доступен для записи! Повторите ещё раз.',

'Файл пустой! Повторите ещё раз.',

'Не верные данные в файле! Повторите ещё раз.',

'Упс... Что-то пошло не так. Потворите ещё раз.');

Var

Tree: TTree;

Implementation

{ TTree.TRoot }

Procedure TRoot.Init(Value: Integer);

Var

PtrTemp: PtrRoot;

Begin

Self.FValue := Value;

Self.FRight := Nil;

Self.FLeft := Nil;

Self.FPrev := Nil;

End;

{ TTree }

Procedure TTree.Add(Value: Integer);

Var

CurRoot: PtrRoot;

IsWriten: Boolean;

Begin

If FPtrHeader = Nil Then

Begin

New(FPtrHeader);

FPtrHeader.Init(Value);

End

Else

Begin

CurRoot := FPtrHeader;

IsWriten := False;

While Not IsWriten Do

Begin

If CurRoot.FValue > Value Then

Begin

If CurRoot.FLeft = Nil Then

Begin

New(CurRoot.FLeft);

CurRoot.FLeft.Init(Value);

CurRoot.FLeft.FPrev := CurRoot;

IsWriten := True;

End

Else

CurRoot := CurRoot.FLeft;

End

Else If CurRoot.FValue < Value Then

Begin

If CurRoot.FRight = Nil Then

Begin

New(CurRoot.FRight);

CurRoot.FRight.Init(Value);

CurRoot.FRight.FPrev := CurRoot;

IsWriten := True;

End

Else

CurRoot := CurRoot.FRight;

End

Else

IsWriten := True;

End;

End;

End;

Function TTree.FindValues: TList<Integer>;

Var

List: TList<Integer>;

Begin

List := TList<Integer>.Create();

FindValues(FPtrHeader, List);

FindValues := List;

End;

function TTree.GetLeft(Root : PtrRoot): PtrRoot;

begin

GetLeft := Root.FLeft;

end;

Function TTree.GetRight(Root : PtrRoot): PtrRoot;

Begin

GetRight := Root.FRight;

End;

Function TTree.FindValues(Var Tree: PtrRoot; Var List: TList<Integer>): Integer;

Var

Left: Integer;

Right: Integer;

Begin

If Tree = Nil Then

Exit(0);

Left := FindValues(Tree.FLeft, List) + 1;

Right := FindValues(Tree.FRight, List) + 1;

If Left <> Right Then

List.Add(Tree.FValue);

If Left > Right Then

FindValues := Left

Else

FindValues := Right;

End;

End.

**Код программы СSharp:**

**Tree.cs**

namespace Lab5\_2;  
  
/// <summary>  
/// Реализация бинарного дерева  
/// </summary>  
/// <typeparam name="T">int, double, extended, char</typeparam>  
public class Tree<T> where T : IComparable<T>  
{  
 private class Root   
 {  
 public T value { get; set; }  
 public Root? right { get; set; }  
 public Root? left { get; set; }  
 public Root? perent { get; set; }  
  
 public Root(T value)  
 {  
 this.value = value;  
 right = null;  
 left = null;  
 perent = null;  
 }  
 }  
  
 private Root tree;  
  
 public Tree()  
 {  
 tree = null;  
 }  
   
 public void Add(T value)  
 {  
 if (tree == null)  
 tree = new Root(value);  
 else  
 {  
 Root cur = tree;  
 bool isWriten = false;  
 while (!isWriten)  
 {  
 if (cur.value.CompareTo(value) > 0)  
 {  
 if (cur.left == null)  
 {  
 cur.left = new Root(value);  
 cur.left.perent = cur;  
 isWriten = true;  
 }  
 else  
 {  
 cur = cur.left;  
 }  
 }  
 else if (cur.value.CompareTo(value) < 0)  
 {  
 if (cur.right == null)  
 {  
 cur.right = new Root(value);  
 cur.right.perent = cur;  
 isWriten = true;  
 }  
 else  
 {  
 cur = cur.right;  
 }  
 }  
 else  
 {  
 isWriten = true;  
 }  
 }  
 }  
 }  
   
 /// <summary>  
 /// функция находит максимальную высоту дерева и список,  
 /// в котором записаны значения веток, у которых  
 /// высота левого поддерева не равно правому   
 /// </summary>  
 /// <param name="temp">текущая ветка</param>  
 /// <param name="list">список</param>  
 /// <returns>максимальную высоту дерева</returns>  
 private int FindValues(Root temp, List<T> list)  
 {  
 if (temp == null)  
 return 0;  
 int left = FindValues(temp.left!, list) + 1;  
 int right = FindValues(temp.right!, list) + 1;  
 if (left != right)  
 list.Add(temp.value);  
 return (left > right) ? left : right;  
 }  
   
 public List<T> FindValues()  
 {  
 List<T> list = new List<T>();  
 FindValues(tree, list);  
 return list;  
 }  
}

**Program.cs**

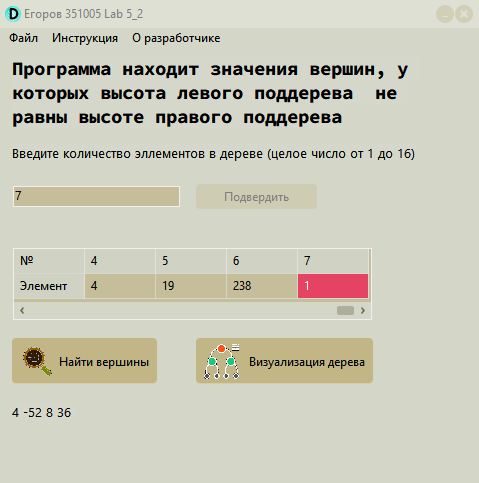
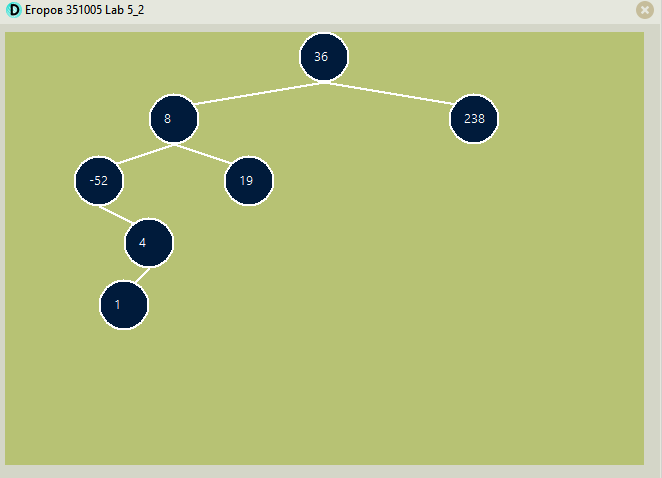
using Lab5\_2;  
  
Tree<int> tree = new Tree<int>();  
MainMenu mainMenu = new MainMenu();  
int choose = 0;  
  
mainMenu.ShowProgramInfo();  
  
// input block  
do  
{  
 mainMenu.InputShowMenu();  
 choose = mainMenu.InputChoice();  
 // initialing reader  
 switch ((WorkingTypes)choose)  
 {  
 case WorkingTypes.WtConsole:  
 {  
 ConsoleReader consoleReader = new ConsoleReader();  
 tree = consoleReader.InputElementsOfTree(consoleReader.InputSizeOfTree());  
 }  
 break;  
 case WorkingTypes.WtFile:  
 {  
 FileReader fileReader = new FileReader();  
 int size = 0;  
 do  
 {  
 fileReader.FilePath = mainMenu.InputFilePath();  
 // reading size and elements and checking status every operation  
 if (fileReader.FileStatus == FileStatus.FsGood)  
 {  
 size = fileReader.InputSizeOfTree();  
 }  
 if (fileReader.FileStatus == FileStatus.FsGood)  
 {  
 tree = fileReader.InputElementsOfTree(size);  
 }  
 } while (fileReader.FileStatus != FileStatus.FsGood);  
 }  
 break;  
 default:  
 {  
 mainMenu.ShowWrongKeyMessage();  
 choose = 0;  
 }  
 break;  
 }  
} while (choose < (int)WorkingTypes.WtConsole || (int)WorkingTypes.WtFile < choose);  
  
// main block  
List<int> listOfValues = tree.FindValues();  
  
// output block  
do  
{  
 mainMenu.OutputShowMenu();  
 choose = mainMenu.InputChoice();  
 // initialing reader  
 switch ((WorkingTypes)choose)  
 {  
 case WorkingTypes.WtConsole:  
 {  
 ConsoleWriter consoleWriter = new ConsoleWriter();  
 consoleWriter.Output(listOfValues);  
 }  
 break;  
 case WorkingTypes.WtFile:  
 {  
 FileWriter fileWriter = new FileWriter();  
 do  
 {  
 fileWriter.FilePath = mainMenu.InputFilePath();  
 if (fileWriter.FileStatus == FileStatus.FsGood)  
 fileWriter.Output(listOfValues);  
 mainMenu.ShowFileStatusMessage(fileWriter.FileStatus);  
 } while (fileWriter.FileStatus != FileStatus.FsGood);  
 }  
 break;  
 default:  
 {  
 mainMenu.ShowWrongKeyMessage();  
 choose = 0;  
 }  
 break;  
 }  
} while (choose < 0 || (int)WorkingTypes.WtFile < choose);

**MainMenu.cs**  
  
public class MainMenu  
{  
  
 public static readonly String[] ErrorMessagesArr =  
 {  
 "Выбран не существующий пункт! Повторите еще раз.",  
 "Не верный тип данных! Повторите ещё раз.",  
 "Число не может быть меньше 1! Повторите ещё раз.",  
 };  
  
 private static readonly String[] FileStatusMessages =  
 {  
 "Операция прошла успешно!",  
 "Файл не найден! Повторите ещё раз.",  
 "Файл не текстовый! Повторите ещё раз.",  
 "Файл закрыт для чтения! Повторите ещё раз.",  
 "Файл закрыт для записи! Повторите ещё раз.",  
 "Файл пустой. Повторите ещё раз.",  
 "Информация в файле неправильная! Повторите ещё раз.",  
 "Количество элементов не совпадает с указанным размером массива! Потворите еще раз.",  
 "Упс... Что-то пошло не так!"  
 };  
  
 public void ShowProgramInfo()  
 {  
 Console.WriteLine("Программа находит значения вершин, у которых высота левого

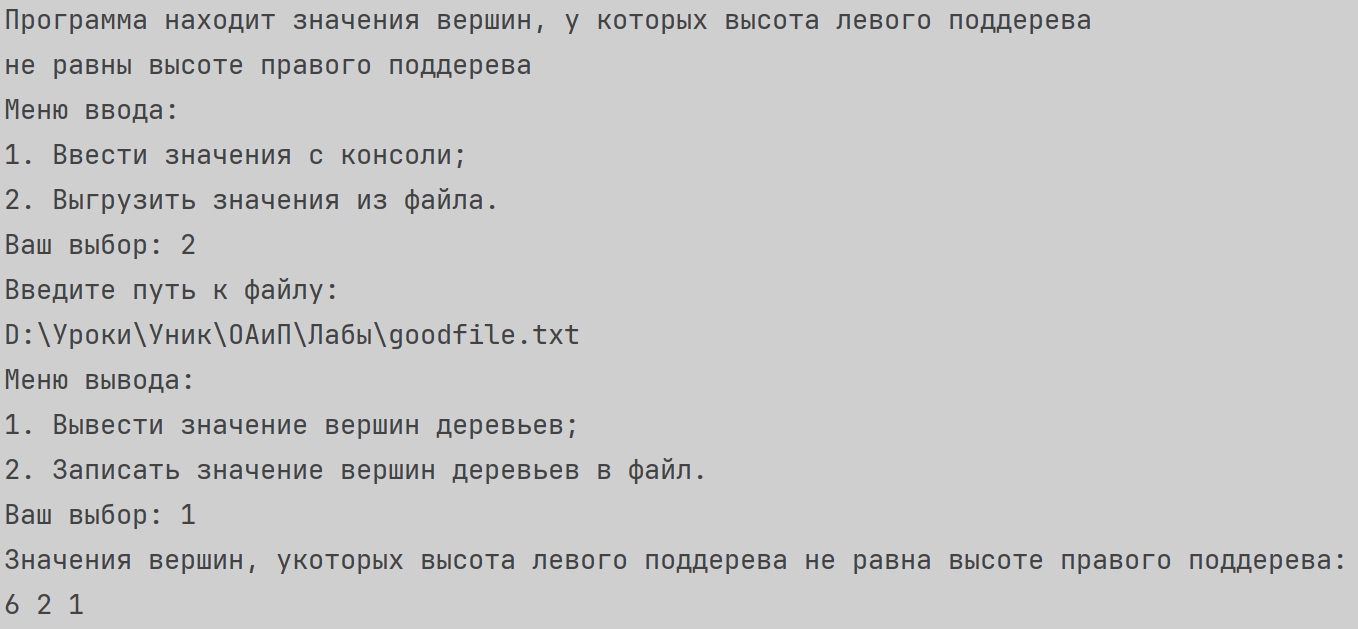
поддерева\nне равны высоте правого поддерева");  
 }  
  
 public int InputChoice()  
 {  
 int choose = 0;  
 bool isBad = true;  
 do  
 {  
 Console.Write("Ваш выбор: ");  
 if (int.TryParse(Console.ReadLine(), out choose))  
 isBad = false;  
 else  
 Console.WriteLine(ErrorMessagesArr[(int)ErrorMessages.EmWrongType]);  
 } while (isBad);  
  
 return choose;  
  
 }  
  
 public void ShowWrongKeyMessage()  
 {  
 Console.WriteLine(ErrorMessagesArr[(int)ErrorMessages.EmWrongKey]);  
 }  
  
 public string? InputFilePath()  
 {  
 Console.WriteLine("Введите путь к файлу:");  
 return Console.ReadLine();  
 }  
  
 public void ShowFileStatusMessage(FileStatus fileStatus)  
 {  
 Console.WriteLine(FileStatusMessages[(int)fileStatus]);  
 }  
  
 public void InputShowMenu()  
 {  
 Console.WriteLine("Меню ввода:\n" +  
 "1. Ввести значения с консоли;\n" +  
 "2. Выгрузить значения из файла.");  
 }  
  
 public void OutputShowMenu()  
 {  
 Console.WriteLine("Меню вывода:\n" +  
 "1. Вывести значение вершин деревьев;\n" +  
 "2. Записать значение вершин деревьев в файл.");  
 }  
}

**Скриншоты:**

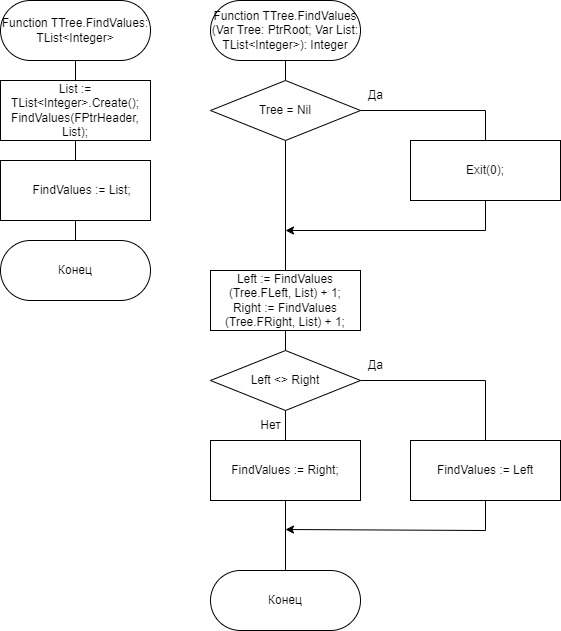
**Delphi:**



**СSharp:**



**Блок-схема:**



**UnitTests на CSharp**

**Код Lab5\_2.Tests.cs**

using System.Collections.Generic;  
using JetBrains.Annotations;  
using Microsoft.VisualStudio.TestTools.UnitTesting;  
using Lab5\_2;  
  
[TestClass]  
[TestSubject(typeof(FileReader))]  
public class FileReaderTests  
{  
 private static FileReader fileReader = new FileReader();   
  
 [TestMethod]  
 public void CheckFileStatus\_givenOnlyWriteFile\_FsNotReadable()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\onlyWriteFile.txt";  
 fileReader.InputSizeOfTree();  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsNotReadable);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenMp3File\_FsNotTxt()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\music.mp3";  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsNotTxt);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenUnexistenFile\_FsNotFound()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\NotExist.txt";  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsNotFound);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongSizeFile\_FsWrongDataType()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\WrongSizeFile.txt";  
 fileReader.InputSizeOfTree();  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsWrongDataType);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongCountFile\_FsWrongCount()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\WrongCountFile.txt";  
 fileReader.InputElementsOfTree(fileReader.InputSizeOfTree());  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsWrongCount);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongElementFile\_FsWrongDataType()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\WrongTypeFile.txt";  
 fileReader.InputElementsOfTree(fileReader.InputSizeOfTree());  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsWrongDataType);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenEmptyFile\_FsEmpty()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\EmptyFile.txt";  
 fileReader.InputSizeOfTree();  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsEmpty);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongElementFile\_FsGood()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\GoodFile.txt";  
 fileReader.InputElementsOfTree(fileReader.InputSizeOfTree());  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsGood);  
 }  
}

[TestClass]  
[TestSubject(typeof(FileWriter))]  
public class FileWriterTexts  
{  
 private static FileWriter fileWriter = new FileWriter();  
   
 [TestMethod]  
 public void CheckFileStatus\_givenOnlyReadFile\_FsNotWritable()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\onlyReadFile.txt";  
 List<int> testList = new List<int>(){1,2,3,4};  
 fileWriter.Output(testList);  
 Assert.AreEqual( fileWriter.FileStatus,FileStatus.FsNotWritable);  
 }   
   
 [TestMethod]  
 public void CheckFileStatus\_givenMp3File\_FsNotTxt()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\music.mp3";  
 Assert.AreEqual(fileWriter.FileStatus,FileStatus.FsNotTxt);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenUnexistenFile\_FsNotFound()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\NotExist.txt";  
 Assert.AreEqual(fileWriter.FileStatus,FileStatus.FsNotFound);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWriteFile\_FsGood()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\OnlyWriteFile.txt";  
 Assert.AreEqual(fileWriter.FileStatus,FileStatus.FsGood);  
 }  
}

[TestClass]  
[TestSubject(typeof(Tree<int>))]  
public class FindValuesTest

{

[TestMethod]  
 public void FindValues\_givenUIntTree\_Equal()  
 {  
 Tree<uint> testTree = new Tree<uint>();  
 testTree.Add(10);  
 testTree.Add(20);  
 testTree.Add(1);  
 testTree.Add(3);  
 testTree.Add(2);  
 testTree.Add(42);  
 List<uint> answer = new List<uint>() { 3,1,20,10 };  
 CollectionAssert.AreEquivalent(testTree.FindValues(),answer);  
 }  
   
 [TestMethod]  
 public void FindValues\_givenIntTree\_Equal()  
 {  
 Tree<int> testTree = new Tree<int>();  
 testTree.Add(10);  
 testTree.Add(20);  
 testTree.Add(-5);  
 testTree.Add(3);  
 testTree.Add(42);  
 List<int> answer = new List<int>() { -5, 20 };  
 CollectionAssert.AreEquivalent(testTree.FindValues(),answer);  
 }  
   
 [TestMethod]  
 public void FindValues\_givenDoubleTree\_Equal()  
 {  
 Tree<double> testTree = new Tree<double>();  
 testTree.Add(10.5);  
 testTree.Add(10.2);  
 testTree.Add(10.7);  
 testTree.Add(-3.234);  
 testTree.Add(42.24);  
 List<double> answer = new List<double>() { 10.2, 10.7 };  
 CollectionAssert.AreEquivalent(testTree.FindValues(),answer);  
 }  
   
 [TestMethod]  
 public void FindValues\_givenCharTree\_Equal()  
 {  
 Tree<char> testTree = new Tree<char>();  
 testTree.Add('g');  
 testTree.Add('d');  
 testTree.Add('i');  
 testTree.Add('z');  
 testTree.Add('h');  
 testTree.Add('a');  
 List<char> answer = new List<char>() { 'd' };  
 CollectionAssert.AreEquivalent(testTree.FindValues(),answer);  
 }  
}

**Скриншоты:**

