УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №5.2

по предмету

Основы алгоритмизации и программирования

Вариант 3

Выполнил:

Бетеня К.С.

Проверила:

Данилова Г.В.

Группа 351005

Минск 2024

Задание:

Деревья. Найти путь максимальной длины в дереве и отразить дерево зеркально относительно этого пути.

Код программы на **Delphi**:

**Unit MainFormUnit;**

Interface

Uses

Winapi.Windows,

Winapi.Messages,

System.SysUtils,

System.Variants,

System.Classes,

Vcl.Graphics,

Vcl.Controls,

Vcl.Forms,

Vcl.Dialogs,

Vcl.Menus,

System.ImageList,

Vcl.ImgList,

Vcl.StdCtrls,

Vcl.Mask,

Vcl.ExtCtrls,

Vcl.Buttons,

Clipbrd;

Type

TLabeledEdit = Class(Vcl.ExtCtrls.TLabeledEdit)

Public

Procedure WMPaste(Var Msg: TMessage); Message WM\_PASTE;

End;

TMainForm = Class(TForm)

TaskLabel: TLabel;

MainMenu: TMainMenu;

MMImgList: TImageList;

FileScroll: TMenuItem;

Open: TMenuItem;

Save: TMenuItem;

Line: TMenuItem;

Exit: TMenuItem;

Instraction: TMenuItem;

AboutEditor: TMenuItem;

CreateTreeSpButton: TSpeedButton;

WatchTreeSpButton: TSpeedButton;

NewTrickLEdit: TLabeledEdit;

NewTrickLabel: TLabel;

BranchCostLEdit: TLabeledEdit;

BranchCostLabel: TLabel;

EndSpButton: TSpeedButton;

MBImgList: TImageList;

AddSpButton: TSpeedButton;

OpenDialog: TOpenDialog;

SaveDialog: TSaveDialog;

Procedure FormCreate(Sender: TObject);

Procedure CreateTreeSpButtonClick(Sender: TObject);

Procedure NewTrickLEditContextPopup(Sender: TObject; MousePos: TPoint; Var

Handled: Boolean);

Procedure NewTrickLEditChange(Sender: TObject);

Procedure AddSpButtonClick(Sender: TObject);

Procedure EndSpButtonClick(Sender: TObject);

Procedure BranchCostLEditContextPopup(Sender: TObject; MousePos: TPoint;

Var Handled: Boolean);

Procedure BranchCostLEditChange(Sender: TObject);

Procedure BranchCostLEditKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

Procedure NewTrickLEditKeyDown(Sender: TObject; Var Key: Word; Shift:

TShiftState);

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

Procedure FormKeyDown(Sender: TObject; Var Key: Word; Shift: TShiftState);

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

Procedure InstractionClick(Sender: TObject);

Procedure AboutEditorClick(Sender: TObject);

Procedure SaveClick(Sender: TObject);

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

Procedure ExitClick(Sender: TObject);

Procedure WatchTreeSpButtonClick(Sender: TObject);

Procedure OpenClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

MainForm: TMainForm;

IfDataSavedInFile: Boolean = False;

Const

ZERO\_KEY: Char = '0';

NULL\_POINT: Char = #0;

DELETE\_KEY: Char = #127;

BACK\_SPACE: Char = #08;

MINUS\_KEY: Char = '-';

MIN\_INT: Integer = -1\_000\_000;

MAX\_INT: Integer = +1\_000\_000;

Implementation

{$R \*.dfm}

Uses

FrontendUnit,

BackendUnit,

DrawUnit,

TreeUnit;

Procedure TMainForm.AddSpButtonClick(Sender: TObject);

Const

MAX\_KNOTS: Integer = 30;

Var

Cost, Child: Integer;

Begin

Cost := 0;

If (GetExistPointsCount() <> 0) Then

Cost := StrToInt(BranchCostLEdit.Text);

Child := StrToInt(NewTrickLEdit.Text);

If (GetExistPoints(Child) <> -1) Then

Application.Messagebox('Такой узел уже существует!', 'Ошибка',

MB\_ICONERROR + MB\_DEFBUTTON2)

Else

If (GetExistPointsCount = MAX\_KNOTS) Then

Application.Messagebox('Вы достигли максимального количества узлов!',

'Ошибка', MB\_ICONERROR + MB\_DEFBUTTON2)

Else

Begin

InsertNewBranch(Child, Cost);

BranchCostLEdit.Visible := True;

BranchCostLabel.Visible := True;

NewTrickLEdit.Text := '';

BranchCostLEdit.Text := '';

EndSpButton.Enabled := True;

End;

End;

Procedure TMainForm.EndSpButtonClick(Sender: TObject);

Begin

SearchLongestWay();

ToMirrorTree();

BranchCostLEdit.Visible := False;

BranchCostLabel.Visible := False;

NewTrickLEdit.Visible := False;

NewTrickLabel.Visible := False;

AddSpButton.Visible := False;

Open.Enabled := False;

Save.Enabled := True;

EndSpButton.Visible := False;

WatchTreeSpButton.Enabled := True;

End;

Procedure TMainForm.SaveClick(Sender: TObject);

Var

IsCorrect: Boolean;

Begin

Repeat

If SaveDialog.Execute Then

Begin

IsCorrect := IsWriteable(SaveDialog.FileName);

InputInFile(IsCorrect, SaveDialog.FileName);

If Not IsCorrect Then

MessageBox(0, 'Невозможна запись в файл!', 'Ошибка',

MB\_ICONERROR);

End

Else

IsCorrect := True;

Until IsCorrect;

End;

End.

**Unit FrontendUnit;**

Interface

Uses

Winapi.Windows,

Winapi.Messages,

System.SysUtils,

System.Variants,

System.Classes,

Vcl.Graphics,

Vcl.Controls,

Vcl.Forms,

Vcl.Dialogs,

Vcl.Menus,

System.ImageList,

Vcl.ImgList,

Vcl.StdCtrls,

Vcl.Mask,

Vcl.ExtCtrls,

Vcl.Buttons;

Procedure CreateModalForm(CaptionText, LabelText: String; ModalWidth, ModalHeight:

Integer);

Function IsCorrectClipboard(ClipbrdText: String; NumLabEd:

Vcl.ExtCtrls.TLabeledEdit): Boolean;

Function CheckInput(Text: String): Boolean;

Function CheckInputKey(LEdit: Vcl.ExtCtrls.TLabeledEdit; Key: Char): Char;

Function IsCorrectDelete(Key: Char; CurentText: String; SelStart: Integer): Char;

Function IsCorrectSelDelete(Key: Char; CurentText, SelText: String; SelStart:

Integer): Char;

Implementation

Uses

MainFormUnit;

Procedure CreateModalForm(CaptionText, LabelText: String; ModalWidth, ModalHeight: Integer);

Const

LEFT\_MARGIN: Integer = 10;

TOP\_MARGIN: Integer = 5;

Var

ModalForm: TForm;

ModalLabel: TLAbel;

Begin

ModalForm := TForm.Create(Nil);

Try

ModalForm.Caption := CaptionText;

ModalForm.Width := ModalWidth;

ModalForm.Height := ModalHeight;

ModalForm.Position := PoScreenCenter;

ModalForm.BorderStyle := BsSingle;

ModalForm.BorderIcons := [BiSystemMenu];

ModalForm.FormStyle := FsStayOnTop;

ModalForm.Icon := MainForm.Icon;

ModalLabel := TLabel.Create(ModalForm);

ModalLabel.Parent := ModalForm;

ModalLabel.Caption := LabelText;

ModalLabel.Left := LEFT\_MARGIN;

ModalLabel.Top := TOP\_MARGIN;

ModalForm.ShowModal;

Finally

ModalForm.Free;

End;

End;

End.

**Unit BackendUnit;**

Interface

Uses

Winapi.Windows,

Winapi.Messages,

System.SysUtils,

System.Variants,

System.Classes,

Vcl.Graphics,

Vcl.Controls,

Vcl.Forms,

Vcl.Dialogs,

Vcl.Menus,

System.ImageList,

Vcl.ImgList,

Vcl.StdCtrls,

Vcl.Mask,

Vcl.ExtCtrls,

Vcl.Buttons,

Clipbrd;

Function IsWriteable(FilePath: String): Boolean;

Procedure InputInFile(Var IsCorrect: Boolean; FilePath: String);

Function IsReadable(FilePath: String): Boolean;

Procedure ReadFromFile(Var IsCorrect: Boolean; FilePath: String);

Implementation

Uses

MainFormUnit,

FrontendUnit,

TreeUnit;

Procedure InputInFile(Var IsCorrect: Boolean; FilePath: String);

Var

MyFile: TextFile;

Begin

If IsCorrect Then

Begin

AssignFile(MyFile, FilePath, CP\_UTF8);

Try

ReWrite(MyFile);

Try

Write(MyFile, 'Ваше дерево:'#13#10);

Write(MyFile, PrintConsoleTree());

Finally

Close(MyFile);

End;

IfDataSavedInFile := True;

Except

IsCorrect := False;

End;

End;

End;

Procedure ReadingProcess(Var IsCorrect: Boolean; Var MyFile: TextFile);

Var

Child, Cost: Integer;

Begin

Try

CreateTree();

Cost := 0;

Read(MyFile, Child);

InsertNewBranch(Child, Cost);

While Not(EOF(MyFile)) Do

Begin

Read(MyFile, Child);

Read(MyFile, Cost);

InsertNewBranch(Child, Cost);

End;

IsCorrect := True;

Except

IsCorrect := False;

End;

IsCorrect := IsCorrect And SeekEOF(MyFile);

End;

Procedure ReadFromFile(Var IsCorrect: Boolean; FilePath: String);

Var

MyFile: TextFile;

Begin

If IsCorrect Then

Begin

AssignFile(MyFile, FilePath);

Try

Reset(MyFile);

Try

ReadingProcess(IsCorrect, MyFile);

MainForm.EndSpButton.Click;

Finally

Close(MyFile);

End;

Except

IsCorrect := False;

End;

End;

End;

End.

**Unit TreeUnit;**

Interface

Uses

Winapi.Windows,

Winapi.Messages,

System.SysUtils,

System.Variants,

System.Classes,

Vcl.Graphics,

Vcl.Controls,

Vcl.Forms,

Vcl.Dialogs,

Vcl.ExtCtrls,

System.Generics.Collections;

Type

Node = ^TBinTree;

TBinTree = Record

Data: Integer;

Parent: Node;

LCost: Integer;

RCost: Integer;

Left: Node;

Right: Node;

End;

Procedure CreateTree();

Function GetExistPointsCount(): Integer;

Function GetExistPoints(Value: Integer): Integer;

Procedure InsertNewBranch(Child, Cost: Integer);

Procedure SearchLongestWay();

Procedure ToMirrorTree();

Procedure DrawTree(PaintBox: TPaintBox);

Function PrintConsoleTree(): String;

Procedure FreeTree();

Var

BinaryTree: Node;

Head: Node;

ExistPoints: TList<Integer>;

LongWayCost: Integer;

LongestPoint: Node;

Implementation

Procedure CreateTree();

Const

MIN\_INT\_VALUE: Integer = -4\_000\_000;

Begin

ExistPoints := TList<Integer>.Create;

LongWayCost := MIN\_INT\_VALUE;

LongestPoint := Nil;

End;

Function GetExistPointsCount(): Integer;

Begin

GetExistPointsCount := ExistPoints.Count;

End;

Function GetExistPoints(Value: Integer): Integer;

Begin

GetExistPoints := ExistPoints.IndexOf(Value);

End;

Function InsertProcess(Root, Parent: Node; Child, Cost: Integer): Node;

Begin

If (Root = Nil) Then

Begin

New(Root);

If (ExistPoints.Count = 0) Then

Head := Root

Else

Root.Parent := Parent;

ExistPoints.Add(Child);

Root.Data := Child;

Root.LCost := 0;

Root.RCost := 0;

Root.Left := Nil;

Root.Right := Nil;

End;

If (ExistPoints.IndexOf(Child) = -1) And (Root.Data > Child) Then

Begin

If (Root.Left = Nil) Then

Root.LCost := Cost;

Root.Left := InsertProcess(Root.Left, Root, Child, Cost);

End

Else

If (ExistPoints.IndexOf(Child) = -1) Then

Begin

If (Root.Right = Nil) Then

Root.RCost := Cost;

Root.Right := InsertProcess(Root.Right, Root, Child, Cost);

End;

InsertProcess := Root;

End;

Procedure InsertNewBranch(Child, Cost: Integer);

Begin

BinaryTree := InsertProcess(BinaryTree, BinaryTree, Child, Cost);

End;

Procedure LongestWay(Root: Node; Cost: Integer);

Begin

If Root <> Nil Then

Begin

LongestWay(Root.Left, Cost + Root.LCost);

If Cost > LongWayCost Then

Begin

LongWayCost := Cost;

LongestPoint := Root;

End;

LongestWay(Root.Right, Cost + Root.RCost);

End;

End;

Procedure SearchLongestWay();

Begin

LongestWay(BinaryTree, 0);

BinaryTree := Head;

End;

Procedure ToMirrorTree();

Var

Temp, LHead: Node;

Begin

LHead := LongestPoint;

While LongestPoint <> Nil Do

Begin

Temp := LongestPoint.Right;

LongestPoint.Right := LongestPoint.Left;

LongestPoint.Left := Temp;

LongestPoint := LongestPoint.Parent;

End;

LongestPoint := LHead;

End;

Procedure ConsoleTree(Node: Node; Var OutputString: String; Prefix: String;

IsLeft: Boolean);

Var

TreeEl: String;

Begin

If (Node = Nil) Then

Exit;

If IsLeft Then

TreeEl := '├── '

Else

TreeEl := '└── ';

OutputString := OutputString + Prefix + TreeEl + IntToStr(Node.Data) + #13#10;

If IsLeft Then

TreeEl := '│ '

Else

TreeEl := ' ';

ConsoleTree(Node.Left, OutputString, Prefix + TreeEl, True);

If IsLeft Then

TreeEl := '│ '

Else

TreeEl := ' ';

ConsoleTree(Node.Right, OutputString, Prefix + TreeEl, False);

End;

Function PrintConsoleTree(): String;

Var

ResultStr: String;

Begin

ResultStr := '';

BinaryTree := Head;

ConsoleTree(BinaryTree, ResultStr, '', False);

BinaryTree := Head;

PrintConsoleTree := ResultStr;

End;

Function CheckCurentElipsColor(Root: Node): TColor;

Const

CustomColor: TColor = $00FFC8B0;

Var

LHead: Node;

ResColor: TColor;

Begin

LHead := LongestPoint;

ResColor := ClWhite;

While (LongestPoint <> Nil) Do

Begin

If (LongestPoint.Data = Root.Data) Then

ResColor := CustomColor;

LongestPoint := LongestPoint.Parent;

End;

LongestPoint := LHead;

CheckCurentElipsColor := ResColor;

LHead := Nil;

End;

Procedure PrintDrawTree(Root: Node; PBox: TPaintBox; X, Y, XOffset: Integer);

Begin

If Root = Nil Then

Exit;

PBox.Canvas.Font.Size := 7;

PBox.Canvas.Brush.Color := CheckCurentElipsColor(Root);

If Root.Left <> Nil Then

Begin

PBox.Canvas.MoveTo(X, Y);

PBox.Canvas.LineTo(X - XOffset, Y + 100);

PrintDrawTree(Root.Left, PBox, X - XOffset, Y + 100, XOffset Div 2);

End;

PBox.Canvas.Brush.Color := CheckCurentElipsColor(Root);

If Root.Right <> Nil Then

Begin

PBox.Canvas.MoveTo(X, Y);

PBox.Canvas.LineTo(X + XOffset, Y + 100);

PrintDrawTree(Root.Right, PBox, X + XOffset, Y + 100, XOffset Div 2);

End;

PBox.Canvas.Ellipse(X - 25, Y - 25, X + 25, Y + 25);

PBox.Canvas.TextOut(X - 17, Y - 7, IntToStr(Root.Data));

End;

Procedure ClearTreeMemory(Root: Node);

Begin

If Root = Nil Then

Exit;

ClearTreeMemory(Root.Left);

ClearTreeMemory(Root.Right);

Dispose(Root);

End;

Procedure FreeTree();

Begin

BinaryTree := Head;

ClearTreeMemory(BinaryTree);

Head := Nil;

LongestPoint := Nil;

End;

Procedure DrawTree(PaintBox: TPaintBox);

Begin

PrintDrawTree(Head, PaintBox, PaintBox.Width Div 2, 30, PaintBox.Width Div 4);

BinaryTree := Head;

End;

End.

Код программы на **C#**:

namespace Proj5\_2 {

    public class Node {

        public readonly int Data;

        public readonly Node? Parent;

        public int LCost;

        public int RCost;

        public Node? Left;

        public Node? Right;

        public Node(int data, Node? parent) {

            Data = data;

            Parent = parent;

            Left = null;

            LCost = 0;

            RCost = 0;

            Right = null;

        }

    }

    internal enum WRITE\_MENU {

        EXIT = 1,

        CONTINUE

    }

    internal enum IOChoose {

        FILE = 1,

        CONSOLE

    }

    public static class Program {

        private const int FILE\_VALUE = (int)IOChoose.FILE;

        private const int CONSOLE\_VALUE = (int)IOChoose.CONSOLE;

        private const int EXIT = (int)WRITE\_MENU.EXIT;

        private const int CONTINUE = (int)WRITE\_MENU.CONTINUE;

        private static Node? root;

        private static Node? longestPoint = root;

        private const int MAX\_INT = +1\_000\_000;

        private const int MIN\_INT = -1\_000\_000;

        private const int MAX\_KNOTS = 50;

        private const int SPACE\_LIMITS = 4;

        private const int MIN\_FILE\_WAY\_SIZE = 4;

        private static int longWayCost;

        private static readonly HashSet<int> existPoints = new HashSet<int>();

        private static void conditionOutput(){

            Console.WriteLine($"""

                Binary search trees.

                    1. Initially, you enter, either through a file or through the

console, the nodes of

                       your tree, as well as the weight of the branch at this

node.

                    2. The first node cannot have weight as a branch, because

there is no other value.

                    3. Both price and weight are limited in the range from

{MIN\_INT} to {MAX\_INT}.

                    4. When entering/outputting through a file, only the extension

can be used.txt!

                """);

        }

        private static bool isProcessOfFileOutputCorrect(string filePath, string

resultStr) {

            try {

                using (StreamWriter writerOutput = new StreamWriter(filePath))

                    writerOutput.WriteLine(resultStr);

                Console.WriteLine("Data successfully written to file.");

                return true;

            }

            catch {

                Console.Error.WriteLine("Error in writing. Try again.");

                return false;

            }

        }

        private static Node insertProcess(Node? root, Node? parent, int child, int

cost) {

            if (root == null) {

                existPoints.Add(child);

                root = new Node(child, parent);

                return root;

            }

            if (root.Data > child) {

                if (root.Left?.Data == null) root.LCost = cost;

                root.Left = insertProcess(root.Left, root, child, cost);

            }

            else {

                if (root.Right?.Data == null) root.RCost = cost;

                root.Right = insertProcess(root.Right, root, child, cost);

            }

            return root;

        }

        private static void insertNewBranch(int child, int cost, int exitCode =

CONTINUE) {

            if (exitCode != EXIT) root = insertProcess(root, root, child, cost);

        }

        private static void outputTextAboutIoSelection(string ioTextInfo) {

            string outputString = $"""

            Select how you will {ioTextInfo} data:

                  {IOChoose.FILE}: {FILE\_VALUE}    {IOChoose.CONSOLE}:

{CONSOLE\_VALUE}

            Your option:

            """;

            Console.Write(outputString);

        }

        /// <summary>

        /// Here you can write a file for what purposes you are using

(input|output)

        /// </summary>

        /// <param name="ioTextInfo"></param>

        /// <returns></returns>

        private static IOChoose chooseIoWay(string ioTextInfo) {

            outputTextAboutIoSelection(ioTextInfo);

            IOChoose result = 0;

            int chosenPath = 0;

            bool isCorrect;

            do {

                isCorrect = true;

                try {

                    chosenPath = Convert.ToInt32(Console.ReadLine());

                }

                catch {

                    isCorrect = false;

                }

                switch (chosenPath) {

                    case FILE\_VALUE: result = IOChoose.FILE; break;

                    case CONSOLE\_VALUE: result = IOChoose.CONSOLE; break;

                    default: isCorrect = false; break;

                }

                if (!isCorrect) Console.Error.Write($"You should write one natural

number({FILE\_VALUE}|

{CONSOLE\_VALUE}): ");

                else Console.WriteLine();

            } while (!isCorrect);

            return result;

        }

        private static bool pathCondition(string filePath) {

            if (filePath.Length < MIN\_FILE\_WAY\_SIZE) {

                Console.Error.Write("The path is too short. Try again: ");

                return false;

            }

            string buffer = filePath.Substring(filePath.Length –

MIN\_FILE\_WAY\_SIZE);

            if (buffer.Equals(".txt")) return true;

            Console.Error.Write("Write .txt file. Try again: ");

            return false;

        }

        private static string inputFilePath() {

            string filePath = Console.ReadLine() ?? string.Empty;

            while (!pathCondition(filePath)) filePath = Console.ReadLine() ??

string.Empty;

            return filePath;

        }

        private static bool isCanOpenFile(string filePath) {

            FileInfo fileInfo = new FileInfo(filePath);

            return fileInfo.Exists;

        }

        private static bool isWriteable(string filePath) {

            try {

                using StreamWriter writer = new StreamWriter(filePath);

                writer.WriteLine(string.Empty);

                writer.Close();

                return true;

            }

            catch {

                return false;

            }

        }

        private static bool isReadable(string filePath) {

            try {

                using StreamReader reader = new StreamReader(filePath);

                reader.Read();

                reader.Close();

                return true;

            }

            catch {

                return false;

            }

        }

        private static bool accessModifierControl(string accessModifier, string

filePath) {

            bool resultModifier = true;

            switch (accessModifier) {

                case "input": resultModifier = isReadable(filePath); break;

                case "output": resultModifier = isWriteable(filePath); break;

            }

            return resultModifier;

        }

        /// <summary>

        /// Write "input" if you want to get the file path for input.

        /// Write "output" if you want to get the path to the output file.

        /// </summary>

        /// <param name="accessModifier"></param>

        /// <returns></returns>

        private static string inputPathToTheFile(string accessModifier) {

            string filePath;

            bool isCorrect;

            do {

                filePath = inputFilePath();

                isCorrect = accessModifierControl(accessModifier, filePath) &&

isCanOpenFile(filePath);

                if (!isCorrect) Console.Error.Write("Can't open a file. Try write

another way: ");

            } while (!isCorrect);

            return filePath;

        }

        private static int inputNumberFromFile(StreamReader inputReader, ref bool

isCorrectInput, int minNum, int

maxNum) {

            int num = 0, minCount = 1, spaceCounter = 0, character;

            bool endOfNum = false;

            while (isCorrectInput && !(endOfNum) && (character =

inputReader.Read()) != -1) {

                int bufChar = character;

                isCorrectInput = isCorrectInput && !((character != ' ') &&

!(character is > '/' and < ':') &&

(character != '\n') && (character != '\r') &&

(character != '-'));

                if (character == ' ') ++spaceCounter;

                else spaceCounter = 0;

                isCorrectInput = spaceCounter != SPACE\_LIMITS;

                if (character is > '/' and < ':')

                    num = num \* 10 + character - 48;

                if (character == '-') minCount = -1;

                isCorrectInput = isCorrectInput && !((character == '-') &&

(minCount != -1));

                endOfNum = (character == ' ' || character == '\n') && (bufChar >

'/' || bufChar < ':');

                isCorrectInput = isCorrectInput && !((num == 0) && character is >

'/' and < ':');

                isCorrectInput = isCorrectInput && !(num > maxNum);

            }

            isCorrectInput = isCorrectInput && (endOfNum ||

inputReader.EndOfStream);

            isCorrectInput = isCorrectInput && !(num > maxNum || num < minNum);

            if (isCorrectInput) num = minCount \* num;

            return num;

        }

        private static int[,] inputBranchesFromFile(StreamReader inputReader, ref

bool isCorrectInput)

        {

            existPoints.Clear();

            int[,] parameters = new int[MAX\_KNOTS,2];

            parameters[0, 0] = inputNumberFromFile(inputReader, ref

isCorrectInput, MIN\_INT,

MAX\_INT);

            existPoints.Add(parameters[0, 0]);

            parameters[0, 1] = 0;

            int counter = 1;

            while (!inputReader.EndOfStream && isCorrectInput) {

                parameters[counter, 0] = inputNumberFromFile(inputReader, ref

isCorrectInput,

MIN\_INT, MAX\_INT);

                if (isCorrectInput) isCorrectInput =

!existPoints.Contains(parameters[counter, 0]);

                existPoints.Add(parameters[counter, 0]);

                isCorrectInput = isCorrectInput && !inputReader.EndOfStream;

                parameters[counter, 1] = inputNumberFromFile(inputReader, ref

isCorrectInput,

MIN\_INT, MAX\_INT);

                counter++;

            }

            int[,] resultArr = new int[counter, 2];

            for (int j = 0; j < counter && isCorrectInput; ++j) {

                resultArr[j, 0] = parameters[j, 0];

                resultArr[j, 1] = parameters[j, 1];

            }

            existPoints.Clear();

            return resultArr;

        }

        private static bool isProcessOfFileInputCorrect(string filePath) {

            bool isCorrectInput = true;

            using StreamReader inputReader = new StreamReader(filePath);

            int[,] parameters = inputBranchesFromFile(inputReader, ref

isCorrectInput);

            for (int i = 0; i < parameters.Length / 2 && isCorrectInput; ++i)

                insertNewBranch(parameters[i,0], i == 0 ? 0 : parameters[i,1]);

            isCorrectInput = isCorrectInput && inputReader.EndOfStream;

            if (!isCorrectInput) Console.Error.WriteLine("Error in reading. Try

again.");

            inputReader.Close();

            return isCorrectInput;

        }

        private static void inputFormFile() {

            Console.Clear();

            string filePath;

            do {

                Console.Write("Write way to your file (\*.txt): ");

                filePath = inputPathToTheFile("input");

            } while (!isProcessOfFileInputCorrect(filePath));

        }

        private static void inputBranch(out int child, out int cost, ref int

exitCode) {

            child = 0;

            cost = 0;

            bool isCorrect;

            do {

                string[] parameters = { };

                try {

                    string input = Console.ReadLine() ?? string.Empty;

                    parameters = input.Split(' ');

                    child = int.Parse(parameters[0]);

                    if (existPoints.Count != 0) cost = int.Parse(parameters[1]);

                    isCorrect = true;

                }

                catch {

                    isCorrect = false;

                }

                if (existPoints.Count != 0 && parameters is ["exit"]) { exitCode =

EXIT;

return; }

                if (existPoints.Count != 0) isCorrect = isCorrect &&

!(parameters.Length > 2);

                else isCorrect = isCorrect && !(parameters.Length > 1);

                isCorrect = isCorrect && !existPoints.Contains(child);

                isCorrect = isCorrect && child is <= MAX\_INT and >= MIN\_INT;

                isCorrect = isCorrect && cost is <= MAX\_INT and >= MIN\_INT;

                if (!isCorrect) Console.Write("Bad input! Try again: ");

            } while (!isCorrect);

        }

        private static void inputFromConsole() {

            int exitCode = CONTINUE;

            Console.Clear();

            do {

                switch (existPoints.Count) {

                    case MAX\_KNOTS: Console.WriteLine("You write maximum count of

knots!"); return;

                    case 0: Console.WriteLine("It is your first knot, write only

kid."); break;

                    case 1: Console.WriteLine("\nWrite 'exit' for stop

writing.\nWrite kid and after write

cost of new branch."); break;

                }

                Console.Write("Input: ");

                inputBranch(out int child, out int cost, ref exitCode);

                insertNewBranch(child, cost, exitCode);

            } while (exitCode == CONTINUE);

        }

        private static void inputTree() {

            IOChoose path = chooseIoWay("input");

            switch (path)

            {

                case IOChoose.FILE: inputFormFile(); break;

                case IOChoose.CONSOLE: inputFromConsole(); break;

            }

            Console.Clear();

        }

        private static void searchLongestWay(Node? root, int cost) {

            if (root == null) return;

            searchLongestWay(root.Left, cost + root.LCost);

            if (cost > longWayCost) {

                longWayCost = cost;

                longestPoint = root;

            }

            searchLongestWay(root.Right, cost + root.RCost);

        }

        private static void toMirrorTree() {

            while (longestPoint?.Parent != null)

longestPoint = longestPoint.Parent;

            Node? temp = longestPoint?.Right;

            if (temp == null) return;

            longestPoint!.Right = longestPoint.Left;

            longestPoint.Left = temp;

        }

        private static void treatmentTree() {

            searchLongestWay(root, 0);

            toMirrorTree();

        }

        private static void printTree(Node? node, ref string outputString, string

prefix = "", bool isLeft = false) {

            if (node == null) return;

            outputString += prefix + (isLeft ? "├── " : "└── ") + node.Data +

'\n';

            printTree(node.Left, ref outputString, prefix + (isLeft ? "│   " :

"    "), true);

            printTree(node.Right, ref outputString, prefix + (isLeft ? "│   " :

"    "));

        }

        private static void outputFromConsole(string resultStr) {

            Console.WriteLine(resultStr);

        }

        static void outputFormFile(string resultStr) {

            Console.Clear();

            string filePath;

            do {

                Console.Write("Write way to your file (\*.txt): ");

                filePath = inputPathToTheFile("output");

            } while (!isProcessOfFileOutputCorrect(filePath, resultStr));

        }

        private static void outputTree() {

            string resultStr = "Your result tree: \n";

            printTree(root, ref resultStr);

            IOChoose path = chooseIoWay("output");

            switch (path) {

                case IOChoose.FILE: outputFormFile(resultStr); break;

                default: outputFromConsole(resultStr); break;

            }

        }

        public static void Main() {

            existPoints.Clear();

            conditionOutput();

            inputTree();

            treatmentTree();

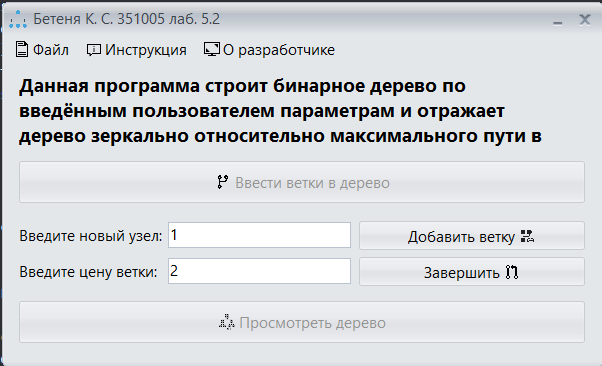
            outputTree();

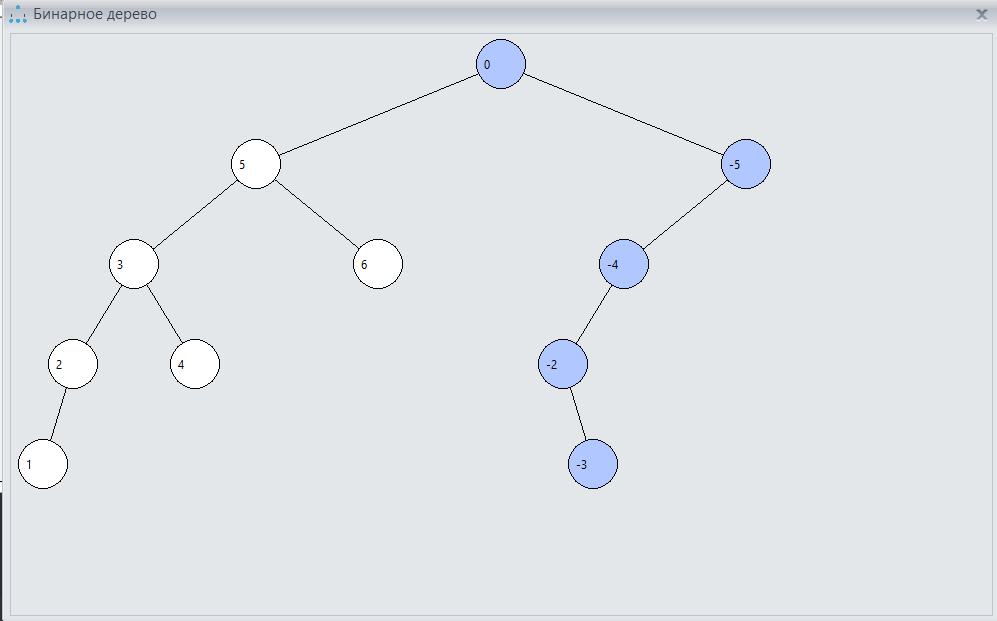
        }

    }

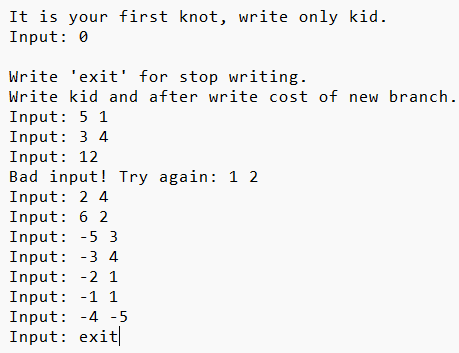
}

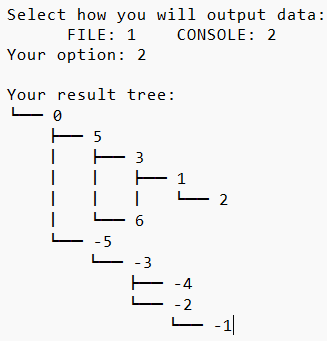
Результат в **Delphi**:





Результат в **C#**:





Блок-схема:

