УО ‘‘Белорусский государственный университет информатики и радиоэлектроники’’

Кафедра ПОИТ

Отчет по лабораторной работе №6.2

По предмету

Основы Алгоритмизации и Программирования

Вариант 25

Выполнил

Тимофиевич А.Д.

Проверила

Данилова Г.В.

Группа:

851001

**Минск 2019**

**Задание**

Вывести номера вершин, для которых высота левого поддерева не равна высоте правого поддерева. Визуализировать.

**Код программы**

**(Delphi)**

**unit** Unit62;

**interface**

**uses**

System.SysUtils, System.Types, System.UITypes, System.Classes,

System.Variants,

FMX.Types, FMX.Controls, FMX.Forms, FMX.Graphics, FMX.Dialogs,

FMX.Objects,

FMX.Controls.Presentation, FMX.StdCtrls, System.ImageList, FMX.ImgList,

FMX.Menus;

**type**

PTree = ^Tree;

Tree = **record**

Num, LeftCount, RightCount: Byte;

Up: PTree;

Left: PTree;

Right: PTree;

**end**;

TLab\_6\_2 = class(TForm)

AddPanel: TPanel;

AddRight: TButton;

AddLeft: TButton;

MainMenu1: TMainMenu;

Help: TMenuItem;

Exit: TMenuItem;

AboutTheDeveloper: TMenuItem;

AboutTheProgram: TMenuItem;

Find: TButton;

PointLabel: TLabel;

Result: TLabel;

HowToUse: TMenuItem;

**procedure** FormShow(Sender: TObject);

**procedure** AddRightClick(Sender: TObject);

**procedure** CirCreate;

**procedure** AddLeftClick(Sender: TObject);

**procedure** FormKeyUp(Sender: TObject; **var** Key: Word; **var** KeyChar: Char;

Shift: TShiftState);

**procedure** FormCloseQuery(Sender: TObject; **var** CanClose: Boolean);

**procedure** ExitClick(Sender: TObject);

**procedure** AboutTheDeveloperClick(Sender: TObject);

**procedure** AboutTheProgramClick(Sender: TObject);

**procedure** DrawL;

**procedure** FindClick(Sender: TObject);

**function** LeftRightCenter(P: PTree): Byte;

**procedure** HowToUseClick(Sender: TObject);

**private**

ArrC: array [1..63] of TCircle;

Root, PTemp: PTree;

Count, Step, Buffer: Integer;

Element, k: Byte;

IsRight: Boolean;

Values: array [1..63] of Byte;

**public**

{ Public declarations }

**end**;

**var**

Lab\_6\_2: TLab\_6\_2;

**const**

Wid = 20;

Down = 40;

**implementation**

{$R \*.fmx}

**procedure** TLab\_6\_2.AboutTheDeveloperClick(Sender: TObject);

**begin**

MessageDlg('This program developed by Temofievich Alexey,

gr.851001.',TMsgDlgType.mtInformation, mbOKCancel, 0);

**end**;

**procedure** TLab\_6\_2.AboutTheProgramClick(Sender: TObject);

**begin**

MessageDlg('This program finds vertices, whose lengths of the left and

right subtrees are different.',TMsgDlgType.mtInformation,

mbOKCancel, 0);

**end**;

**procedure** TLab\_6\_2.AddLeftClick(Sender: TObject);

**begin**

AddLeft.Enabled := True;

AddRight.Enabled := True;

New(PTemp^.Left);

PTemp^.Left^.Up := PTemp;

PTemp := PTemp^.Left;

IsRight := False;

CirCreate;

DrawL;

**end**;

**procedure** TLab\_6\_2.DrawL;

**var**

Brush: TStrokeBrush;

**begin**

Brush := TStrokeBrush.Create(TBrushKind.Solid, TAlphaColorRec.Black);

Brush.Thickness := 1;

with **Canvas** do

**begin**

**BeginScene**;

**if** **not** (IsRight) **then**

DrawLine(PointF(Step + Count \* 2 + wid/2, Buffer - 2\*down +

wid),PointF(Step+ wid/2, Buffer - Down),1, Brush)

**else**

DrawLine(PointF(step - Count \* 2 + wid/2, Buffer - 2\*down +

wid),PointF(Step+ wid/2, Buffer - Down),1, Brush);

**EndScene**;

**end**;

**end**;

**procedure** TLab\_6\_2.AddRightClick(Sender: TObject);

**begin**

AddLeft.Enabled := True;

AddRight.Enabled := True;

New(PTemp^.Right);

PTemp^.Right^.Up := PTemp;

PTemp := PTemp^.Right;

IsRight := True;

CirCreate;

DrawL;

**end**;

**procedure** TLab\_6\_2.CirCreate;

**var**

TempC: TCircle;

TempL: TLabel;

**begin**

PTemp^.Num := Element;

PTemp^.Left := nil;

PTemp^.Right := nil;

TempC := TCircle.Create(Lab\_6\_2);

TempC.Parent := Lab\_6\_2;

TempC.Width := Wid;

TempC.Height := Wid;

**if** IsRight **then**

TempC.Position.X := Count + Step

**else**

TempC.Position.X := Step - Count;

Step := Trunc(TempC.Position.X);

Count := Count div 2;

TempC.Position.Y := Buffer;

TempC.Fill.Color := TAlphaColorRec.Red;

TempL := TLabel.Create(TempC);

TempL.Parent := TempC;

TempL.Width := wid;

TempL.Height := wid;

TempL.Position.X := 3;

TempL.Text := IntToStr(Element);

ArrC[Element] := TempC;

Buffer := Buffer + Down;

**if** (Buffer > (5 \* Down)) **then**

**begin**

AddRight.Enabled := False;

AddLeft.Enabled := False;

**end**;

PointLabel.Text := 'You are at the vertex ' + IntToStr(Element);

Inc(Element);

**end**;

**procedure** TLab\_6\_2.ExitClick(Sender: TObject);

**begin**

Close;

**end**;

**function** TLab\_6\_2.LeftRightCenter(P: PTree): Byte;

**begin**

**if** P.Left <> nil **then**

P.LeftCount := 1 + LeftRightCenter(P.Left)

**else**

P.LeftCount := 0;

**if** P.Right <> nil **then**

P.RightCount := 1 + LeftRightCenter(P.Right)

**else**

P.RightCount := 0;

LeftRightCenter := P.LeftCount + P.RightCount;

**if** P.LeftCount <> P.RightCount **then**

**begin**

Values[k] := P.Num;

inc(k);

**end**;

**end**;

**procedure** TLab\_6\_2.FindClick(Sender: TObject);

**var**

i: Byte;

**begin**

PTemp := Root;

LeftRightCenter(PTemp);

Result.Text := 'Vertex numbers for which the height of the right subtree

is not equal to the height of the left subtree is:';

**for** i := 1 **to** k - 1 **do**

**begin**

Result.Text := Result.Text + ' ' + IntToStr(Values[i]);

**end**;

k := 1;

Buffer := Down;

Step := Trunc(ArrC[1].Position.X);

Count := (Lab\_6\_2.Width - wid) div 4;

PointLabel.Text := 'You are at the vertex 1';

**end**;

**procedure** TLab\_6\_2.FormCloseQuery(Sender: TObject; var CanClose: Boolean);

**var**

ButtonSelected: Byte;

**begin**

ButtonSelected := MessageDlg('Are you sure you want to

exit?',TMsgDlgType.mtConfirmation, mbYesNo, 0);

**if** ButtonSelected <> mrYes **then**

CanClose := False;

**end**;

**procedure** TLab\_6\_2.UpClick;

**var**

i: Byte;

**begin**

**if** PTemp^.Up <> nil **then**

**begin**

PTemp := PTemp^.Up;

i := PTemp^.Num;

Count := Count \* 2;

Buffer := Buffer - Down;

Step := Trunc(ArrC[i].Position.X);

PointLabel.Text := 'You are at the vertex ' + IntToStr(i);

**end**;

**end**;

**procedure** TLab\_6\_2.LeftClick;

**var**

i: Byte;

**begin**

**if** PTemp^.Left <> nil **then**

**begin**

PTemp := PTemp^.Left;

i := PTemp^.Num;

Count := Count div 2;

Buffer := Buffer + Down;

Step := Trunc(ArrC[i].Position.X);

PointLabel.Text := 'You are at the vertex ' + IntToStr(i);

**end**;

**end**;

**procedure** TLab\_6\_2.RightClick;

**var**

i: Byte;

**begin**

**if** PTemp^.Right <> nil **then**

**begin**

PTemp := PTemp^.Right;

i := PTemp^.Num;

Count := Count div 2;

Buffer := Buffer + Down;

Step := Trunc(ArrC[i].Position.X);

PointLabel.Text := 'You are at the vertex ' + IntToStr(i);

**end**;

**end**;

**procedure** TLab\_6\_2.FormKeyUp(Sender: TObject; **var** Key: Word; **var** KeyChar:

Char; Shift: TShiftState);

**begin**

AddRight.Enabled := False;

AddLeft.Enabled := False;

**case** Key **of**

vkUp:

UpClick;

vkLeft:

LeftClick;

vkRight:

RightClick;

**end**;

**if** PTemp^.Left = nil **then**

**begin**

IsRight := False;

AddLeft.Enabled := True;

**end**;

**if** PTemp^.Right = nil **then**

**begin**

IsRight := True;

AddRight.Enabled := True;

**end**;

**if** (Buffer > (5 \* Down)) **then**

**begin**

AddRight.Enabled := False;

AddLeft.Enabled := False;

**end**;

**end**;

**procedure** TLab\_6\_2.FormShow(Sender: TObject);

**begin**

k := 1;

Buffer := 0;

Step := 0;

Count := (Lab\_6\_2.Width - wid) div 2;

Element := 1;

New(Root);

Root^.Up := nil;

PTemp := Root;

IsRight := True;

CirCreate;

**end**;

**procedure** TLab\_6\_2.HowToUseClick(Sender: TObject);

**begin**

MessageDlg('Use the keys ''Up'', ''Left'', ''Right'' to move around the

tree.',TMsgDlgType.mtInformation, mbOKCancel, 0);

**end**;

**end**.

**unit MyUnit;**

**interface**

**uses**

FMX.Forms,FMX.StdCtrls, FMX.Objects, System.UITypes,System.SysUtils, FMX.Graphics, System.Types;

**type**

PTree = ^Tree;

Tree = record

Num, LeftCount, RightCount: Byte;

Up: PTree;

Left: PTree;

Right: PTree;

end;

ArrV = array [1..63] of Byte;

**var**

ArrC: array [1..63] of TCircle;

Root, PTemp: PTree;

Count, Step, Buffer: Integer;

Temp: Single;

Element, k: Byte;

IsRight: Boolean;

Values: ArrV;

**const**

Wid = 20;

down = 40;

**procedure** AddL;

**procedure** Start(Width: Single);

**procedure** CirDraw(Form: TForm);

**procedure** AddR;

**procedure** DrawL(Form: TForm);

**function** Find: ArrV;

**function** Up: Byte;

**function** Left: Byte;

**function** Right: Byte;

**function** CheckLeft: Boolean;

**function** CheckRight: Boolean;

**function** CheckDown: Boolean;

**implementation**

**procedure** AddR;

**begin**

New(PTemp^.Right);

PTemp^.Right^.Up := PTemp;

PTemp := PTemp^.Right;

IsRight := True;

**end**;

**function** CheckLeft: Boolean;

**begin**

Result := False;

**if** PTemp^.Left = nil **then**

Result := True;

**end**;

**function** CheckRight: Boolean;

**begin**

Result := False;

**if** PTemp^.Right = nil **then**

Result := True;

**end**;

**function** CheckDown: Boolean;

**begin**

Result := True;

**if** (Buffer > (5 \* Down)) **then**

Result := False;

**end**;

**procedure** DrawL(Form: TForm);

**var**

Brush: TStrokeBrush;

**begin**

Brush := TStrokeBrush.Create(TBrushKind.Solid, TAlphaColorRec.Black);

Brush.Thickness := 1;

**with** Form, Canvas **do**

**begin**

**BeginScene**;

**if** **not** (IsRight) **then**

DrawLine(PointF(step + Count \* 2 + wid/2, Buffer - 2\*down +

wid),PointF(Step + wid/2, Buffer - Down),1, Brush)

**else**

DrawLine(PointF(step - Count \* 2 + wid/2, Buffer - 2\*down +

wid),PointF(Step + wid/2, Buffer - Down),1, Brush);

**EndScene**;

**end**;

**end**;

**function** Up: Byte;

**var**

i: Byte;

**begin**

i := PTemp^.Num;

if PTemp^.Up <> nil **then**

**begin**

PTemp := PTemp^.Up;

i := PTemp^.Num;

Count := Count \* 2;

Buffer := Buffer - Down;

Step := Trunc(ArrC[i].Position.X);

**end**;

Result := i;

**end**;

**function** Left: Byte;

**var**

i: Byte;

**begin**

i := PTemp^.Num;

**if** PTemp^.Left <> nil **then**

**begin**

PTemp := PTemp^.Left;

i := PTemp^.Num;

Count := Count div 2;

Buffer := Buffer + Down;

Step := Trunc(ArrC[i].Position.X);

**end**;

Result := i;

**end**;

**function** Right: Byte;

**var**

i: Byte;

**begin**

i := PTemp^.Num;

**if** PTemp^.Right <> nil **then**

**begin**

PTemp := PTemp^.Right;

i := PTemp^.Num;

Count := Count div 2;

Buffer := Buffer + Down;

Step := Trunc(ArrC[i].Position.X);

**end**;

Result := i;

**end**;

**procedure** AddL;

**begin**

New(PTemp^.Left);

PTemp^.Left^.Up := PTemp;

PTemp := PTemp^.Left;

IsRight := False;

**end**;

**procedure** Start(Width: Single);

**begin**

k := 1;

Buffer := 0;

Step := 0;

Temp := Width;

Count := Trunc((Width - wid) / 2);

Element := 1;

New(Root);

Root^.Up := nil;

PTemp := Root;

IsRight := True;

**end**;

**procedure** CirDraw(Form: TForm);

**var**

TempC: TCircle;

TempL: TLabel;

**begin**

PTemp^.Num := Element;

PTemp^.Left := nil;

PTemp^.Right := nil;

TempC := TCircle.Create(Form);

TempC.Parent := Form;

TempC.Width := Wid;

TempC.Height := Wid;

**if** IsRight **then**

TempC.Position.X := Count + Step

**else**

TempC.Position.X := Step - Count;

Step := Trunc(TempC.Position.X);

Count := Count div 2;

TempC.Position.Y := Buffer;

TempC.Fill.Color := TAlphaColorRec.Red;

TempL := TLabel.Create(TempC);

TempL.Parent := TempC;

TempL.Width := wid;

TempL.Height := wid;

TempL.Position.X := 3;

TempL.Text := IntToStr(Element);

ArrC[Element] := TempC;

Buffer := Buffer + Down;

Inc(Element);

**end**;

**function** LeftRightCenter(P: PTree): Byte;

**begin**

**if** P.Left <> nil **then**

P.LeftCount := 1 + LeftRightCenter(P.Left)

**else**

P.LeftCount := 0;

**if** P.Right <> nil **then**

P.RightCount := 1 + LeftRightCenter(P.Right)

**else**

P.RightCount := 0;

LeftRightCenter := P.LeftCount + P.RightCount;

**if** P.LeftCount <> P.RightCount **then**

**begin**

Values[k] := P.Num;

inc(k);

**end**;

**end**;

**function** Find: ArrV;

**begin**

LeftRightCenter(Root);

Result := Values;

k := 1;

**end**;

**end**.

**library TreeLib;**

**uses**

System.SysUtils,

System.Classes;

**type**

PTree = ^Tree;

Tree = record

Num, LeftCount, RightCount: Byte;

Up: PTree;

Left: PTree;

Right: PTree;

**end**;

ArrV = array [1..63] of Byte;

{$R \*.res}

**procedure** \_AddR(**var** PTemp: PTree; **var** Element: Byte); **stdcall**;

**begin**

PTemp^.Right^.Up := PTemp;

PTemp := PTemp^.Right;

PTemp^.Num := Element;

PTemp^.Left := nil;

PTemp^.Right := nil;

**end**;

**function** \_CheckLeft(**var** PTemp: PTree): Boolean; **stdcall**;

**begin**

Result := False;

**if** PTemp^.Left = nil **then**

Result := True;

**end**;

**function** \_CheckRight(**var** PTemp: PTree): Boolean; **stdcall**;

**begin**

Result := False;

**if** PTemp^.Right = nil **then**

Result := True;

**end**;

**procedure** \_AddL(**var** PTemp: PTree; **var** Element: Byte); **stdcall**;

**begin**

PTemp^.Left^.Up := PTemp;

PTemp := PTemp^.Left;

PTemp^.Num := Element;

PTemp^.Left := nil;

PTemp^.Right := nil;

**end**;

**procedure** \_Start(**var** PTemp, Root: PTree; **var** Element: Byte); **stdcall**;

**begin**

Root^.Up := nil;

PTemp := Root;

Element := 1;

PTemp^.Num := Element;

PTemp^.Left := nil;

PTemp^.Right := nil;

**end**;

**function** \_LeftRightCenter(**var** P: PTree; **var** Values: ArrV; **var** k: Byte): Byte;

**stdcall**;

**begin**

**if** P.Left <> nil **then**

P.LeftCount := 1 + \_LeftRightCenter(P.Left, Values, k)

**else**

P.LeftCount := 0;

**if** P.Right <> nil **then**

P.RightCount := 1 + \_LeftRightCenter(P.Right, Values, k)

**else**

P.RightCount := 0;

\_LeftRightCenter := P.LeftCount + P.RightCount;

**if** P.LeftCount <> P.RightCount **then**

**begin**

Values[k] := P.Num;

inc(k);

**end**;

**end**;

**exports**

\_LeftRightCenter,

\_Start,

\_AddL,

\_CheckRight,

\_CheckLeft,

\_AddR;

**begin**

**end**.

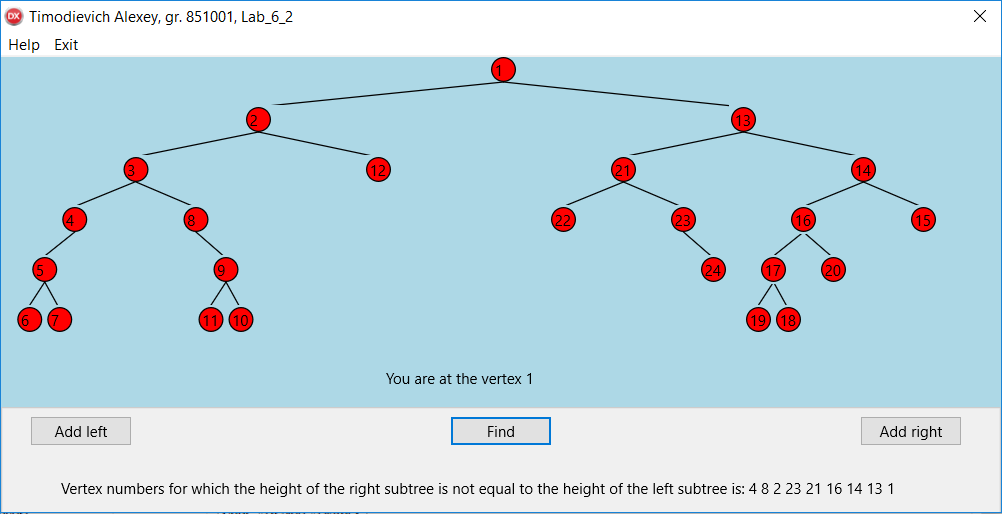
**Блок-схема**







**Скриншоты**

****