УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №5.2

по предмету «Основы алгоритмизации и программирования»

Вариант 20

Выполнил:

Захвей И.В.

Гр. 351005

Проверил:

Данилова Г. В.

Минск 2024

**Задание:**

Построить дерево двоичного поиска места вставки заданного числа из данного массива. Дерево визуализировать.

**Код программы Delphi:**

**MainUnit.pas**

Var

MainForm: TMainForm;

Tree: TTree;

ChoosedNum: Integer;

Node: TNodePointer;

Implementation

{$R \*.dfm}

Procedure ClearGrid(Grid: TStringGrid);

Var

J, I: Integer;

Begin

Grid.Visible := False;

For I := 0 To Grid.ColCount - 1 Do

For J := 0 To Grid.RowCount - 1 Do

Grid.Cells[I, J] := '';

Grid.ColCount := 1;

End;

Procedure TMainForm.AddButtonClick(Sender: TObject);

Var

NewNodeData, MaxCountOfNode, TreeHigh: Integer;

IsAdded: Boolean;

Begin

NewNodeData := StrToInt(NewNodeEdit.Text);

NewNodeEdit.Text := '';

If Add(Tree^.Next, NewNodeData) Then

Begin

AddElemToGrid(StringGrid1, NewNodeData);

ChoosedNum := NewNodeData;

End

Else

ErrMessage();

If Tree^.Next <> Nil Then

FreeButton.Enabled := True;

TreeHigh := GetHigh(Tree^.Next);

MaxCountOfNode := Pow(2, TreeHigh - 1);

MainPaintBox.Width := NODE\_SIZE \* MaxCountOfNode;

MainPaintBox.Height := LAYER\_SIZE \* TreeHigh;

MainPaintBox.Canvas.MoveTo(MainPaintBox.Width Div 2, NODE\_SIZE Div 2);

PrintTree(Tree^.Next, MainPaintBox.Canvas, 0, MainPaintBox.Width, 0,

ChoosedNum);

End;

Procedure TMainForm.AddElemToGrid(Grid: TStringGrid; Elem: Integer);

Begin

With Grid Do

Begin

If Visible Then

Begin

ColCount := ColCount + 1;

End

Else

Visible := True;

Cells[ColCount - 1, 0] := IntToStr(ColCount) + '.';

Cells[ColCount - 1, 1] := IntToStr(Elem);

End;

End;

Procedure TMainForm.DeveloperButtonMenuClick(Sender: TObject);

Var

DeveloperForm: TDeveloperForm;

Begin

DeveloperForm := TDeveloperForm.Create(Self);

DeveloperForm.ShowModal;

DeveloperForm.Free;

End;

Procedure TMainForm.ErrMessage;

Begin

Application.MessageBox('Такой элемент уже существует', 'Ошибочка',

MB\_OK + MB\_ICONASTERISK + MB\_DEFBUTTON2)

End;

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

Begin

Case Application.MessageBox('Вы точно хотите выйти?', 'Выход',

MB\_YESNO + MB\_ICONQUESTION + MB\_DEFBUTTON2) Of

IDYES:

CanClose := True;

IDNO:

CanClose := False;

End;

End;

Procedure TMainForm.FormCreate(Sender: TObject);

Begin

Tree := CreateTree();

NewNodeEdit.Left := B\_PADDING;

NewNodeEdit.Top := B\_PADDING;

AddButton.Top := B\_PADDING;

FreeButton.Top := B\_PADDING;

StringGrid1.Top := B\_PADDING + FreeButton.Height + 3;

StringGrid1.Left := B\_PADDING;

End;

Procedure TMainForm.FormResize(Sender: TObject);

Begin

ScrollBox.Left := B\_PADDING;

ScrollBox.Width := Self.ClientWidth - B\_PADDING - ScrollBox.Left;

ScrollBox.Height := Self.ClientHeight - B\_PADDING - ScrollBox.Top;

StringGrid1.Width := ScrollBox.Width;

TaskLabel.Left := B\_PADDING;

End;

Procedure TMainForm.FreeButtonClick(Sender: TObject);

Begin

DeleteTree(Tree^.Next);

FreeButton.Enabled := False;

ClearGrid(StringGrid1);

MainPaintBox.OnPaint(Sender);

End;

Procedure TMainForm.MainPaintBoxPaint(Sender: TObject);

Begin

With MainPaintBox.Canvas Do

Begin

Pen.Width := 3;

Brush.Color := ClMenu;

FillRect(MainPaintBox.ClientRect);

End;

MainPaintBox.Canvas.MoveTo(MainPaintBox.Width Div 2, NODE\_SIZE Div 2);

PrintTree(Tree^.Next, MainPaintBox.Canvas, 0, MainPaintBox.Width, 0,

ChoosedNum);

End;

Procedure TMainForm.ManualButtonMenuClick(Sender: TObject);

Var

ManualForm: TManualForm;

Begin

ManualForm := TManualForm.Create(Self);

ManualForm.ShowModal;

ManualForm.Free;

End;

Procedure TMainForm.NewNodeEditChange(Sender: TObject);

Begin

With Sender As TEdit Do

If (Text <> '') And (Text <> '-') Then

AddButton.Enabled := True

Else

AddButton.Enabled := False;

End;

Procedure TMainForm.NewNodeEditKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

If ((SsShift In Shift) Or (SsCtrl In Shift)) And

Not((Key = VK\_RIGHT) Or (Key = VK\_LEFT)) Then

Key := 0;

If Key = VK\_DELETE Then

Key := 0;

If (Key = VK\_RETURN) And AddButton.Enabled Then

AddButton.Click();

End;

Procedure TMainForm.NewNodeEditKeyPress(Sender: TObject; Var Key: Char);

Begin

With Sender As TEdit Do

TotalKeyPress(Key, SelStart, SelLength, MIN\_NODE, MAX\_NODE, Text);

End;

Procedure TMainForm.StringGrid1Click(Sender: TObject);

Begin

With Sender As TStringGrid Do

ChoosedNum := StrToInt(Cells[Col, Row]);

MainPaintBox.OnPaint(Sender);

End;

End.

**ManualUnit.pas**

Const

TextInf = '1. Элементы дерева должны быть от -99999 до 99999 ' + #13#10 +

'2. Элементы не могут повторяться' + #13#10 +

'3. Красным цветом подсвечивется элемент выбранный элемент в массиве' +

#13#10 + '4. Чтобы выбрать другой элемент, просто кликните по нему в таблице';

Var

ManualForm: TManualForm;

Implementation

{$R \*.dfm}

Procedure TManualForm.FormCreate(Sender: TObject);

Begin

ManualLabel.Caption := TextInf;

End;

End.

**BackUnit.pas**

Implementation

Procedure PrintTree(Node: TNodePointer; Canvas: TCanvas;

LBorder, RBorder, Layer, Num: Integer);

Var

Center, TextX, TextY, CurLayer: Integer;

Text: String;

Begin

If Node <> Nil Then

With Canvas Do

Begin

Center := LBorder + (RBorder - LBorder) Div 2;

CurLayer := Layer \* LAYER\_SIZE;

// draw line from previous Node

LineTo(Center, CurLayer + NODE\_SIZE Div 2);

// set some settings

if Node^.Data = Num then

Canvas.Brush.Color := ClRed

Else

Canvas.Brush.Color := ClWhite;

Canvas.Font.Color := ClBlack;

Canvas.Font.Size := FONT\_SIZE;

// draw Node

Ellipse(Center - NODE\_SIZE Div 2, CurLayer,

Center + NODE\_SIZE Div 2, CurLayer + NODE\_SIZE);

// draw Node data

Text := IntToStr(Node^.Data);

TextX := Center - TextWidth(Text) Div 2;

TextY := CurLayer + (NODE\_SIZE - TextHeight(Text)) Div 2;

TextOut(TextX, TextY, Text);

MoveTo(Center, CurLayer + NODE\_SIZE);

If Node^.Right <> Nil Then

PrintTree(Node^.Right, Canvas, Center, RBorder, Layer + 1, Num);

MoveTo(Center, CurLayer + NODE\_SIZE);

If Node^.Left <> Nil Then

PrintTree(Node^.Left, Canvas, LBorder, Center, Layer + 1, Num);

End;

End;

Function Pow(Base, Degree: Integer): Integer;

Var

Number: Integer;

I: Integer;

Begin

Number := 1;

If Degree > 0 Then

Begin

Number := 1;

For I := 1 To Degree Do

Number := Number \* Base;

End;

Pow := Number;

End;

Function InsertKey(Index: Integer; SubStr: Char; SelLen: Integer;

Text: String): String;

Var

ResultText: String;

Begin

ResultText := Text;

If (SubStr = BACKSPACE) And (SelLen = 0) Then

Delete(ResultText, Index, 1)

Else

Begin

Delete(ResultText, Index + 1, SelLen);

If Substr <> BACKSPACE Then

ResultText.Insert(Index, String(SubStr));

End;

InsertKey := ResultText;

End;

Function CountOfSymbolInt(Num: Integer): Integer;

Var

NumLen: Integer;

Begin

NumLen := 0;

If Num < 0 Then

Inc(NumLen);

Repeat

Inc(NumLen);

Num := Num Div 10;

Until (Num = 0);

CountOfSymbolInt := NumLen;

End;

Procedure TotalKeyPress(Var Key: Char; SelStart, SelLength: Integer;

Const MIN, MAX: Integer; Text: String);

Var

ResultNum, RBorder, NumLen: Integer;

Buffer, Output: String;

Begin

Output := InsertKey(SelStart, Key, SelLength, Text);

If (Length(Output) <> 0) And (Output <> '-') Then

Begin

Try

ResultNum := StrToInt(Output);

Except

Key := VOID;

End;

If Key <> VOID Then

Begin

NumLen := CountOfSymbolInt(ResultNum);

If NumLen <> Length(Output) Then

Key := VOID;

If (ResultNum > MAX) Or (ResultNum < MIN) Then

Key := VOID;

End;

End

Else If (Output = '-') And (MIN >= 0) Then

Key := VOID;

End;

End.

**TreeUnit.pass**

Interface

Type

TNodePointer = ^TNode;

TNode = Record

Data: Integer;

Left, Right: TNodePointer;

End;

TTree = ^TTreeHead;

TTreeHead = Record

Next: TNodePointer;

End;

Function CreateTree(): TTree;

Function Add(Var Node: TNodePointer; Elem: Integer): Boolean;

Function GetHigh(Node: TNodePointer): Integer;

Procedure DeleteTree(Var Node: TNodePointer);

Implementation

Procedure DeleteTree(Var Node: TNodePointer);

Begin

if Node^.Left <> Nil then

DeleteTree(Node^.Left);

if Node^.Right <> Nil then

DeleteTree(Node^.Right);

Node^.Data := 0;

Node := Nil;

Dispose(Node);

End;

Function GetHigh(Node: TNodePointer): Integer;

Var

LCount, RCount, High: Integer;

Begin

High := 0;

if Node <> Nil then

Begin

Inc(High);

LCount := High + GetHigh(Node^.Left);

RCount := High + GetHigh(Node^.Right);

if LCount > RCount then

High := LCount

Else

High := RCount;

End;

GetHigh := High;

End;

Function Add(Var Node: TNodePointer; Elem: Integer): Boolean;

Var

IsAdded: Boolean;

Begin

IsAdded := False;

if Node = Nil then

Begin

New(Node);

Node^.Data := Elem;

Node^.Left := Nil;

Node^.Right := Nil;

IsAdded := True;

End

Else if Elem > Node^.Data then

IsAdded := Add(Node^.Right, Elem)

Else if Elem < Node^.Data then

IsAdded := Add(Node^.Left, Elem);

Add := IsAdded;

End;

Function CreateTree(): TTree;

Var

Tree: TTree;

Begin

New(Tree);

Tree^.Next := Nil;

CreateTree := Tree;

End;

End.

**БИБЛИОТЕЧНЫЙ МОДУЛЬ**

Library AwesomeLibrary;

{$R \*.res}

Type

TNodePointer = ^TNode;

TNode = Record

Data: Integer;

Left, Right: TNodePointer;

End;

TTree = ^TTreeHead;

TTreeHead = Record

Next: TNodePointer;

End;

Var

Tree: TTree;

Procedure DeleteTreePlease(Var Node: TNodePointer);

Begin

If Node^.Left <> Nil Then

DeleteTreePlease(Node^.Left);

If Node^.Right <> Nil Then

DeleteTreePlease(Node^.Right);

Node^.Data := 0;

Node := Nil;

Dispose(Node);

End;

Procedure DeleteTree();

Begin

DeleteTreePlease(Tree^.Next);

End;

Function GetHighTree(Node: TNodePointer): Integer;

Var

LCount, RCount, High: Integer;

Begin

High := 0;

If Node <> Nil Then

Begin

Inc(High);

LCount := High + GetHighTree(Node^.Left);

RCount := High + GetHighTree(Node^.Right);

If LCount > RCount Then

High := LCount

Else

High := RCount;

End;

GetHighTree := High;

End;

Function GetHigh(): Integer;

Begin

GetHigh := GetHighTree(Tree^.Next);

End;

Function AddToTree(Var Node: TNodePointer; Elem: Integer): Boolean;

Var

IsAdded: Boolean;

Begin

IsAdded := False;

If Node = Nil Then

Begin

New(Node);

Node^.Data := Elem;

Node^.Left := Nil;

Node^.Right := Nil;

IsAdded := True;

End

Else If Elem > Node^.Data Then

IsAdded := AddToTree(Node^.Right, Elem)

Else If Elem < Node^.Data Then

IsAdded := AddToTree(Node^.Left, Elem);

AddToTree := IsAdded;

End;

Function Add(Elem: Integer): Boolean;

Begin

Add := AddToTree(Tree^.Next, Elem);

End;

Procedure CreateTree();

Begin

New(Tree);

Tree^.Next := Nil;

End;

Exports Add, DeleteTree, CreateTree, GetHigh;

Begin

End.

**Код программы Java:**

import java.util.Scanner;

class HeadPt {

Node root;

HeadPt (Node root) {

this.root = root;

}

}

class Node {

int data;

Node leftPt;

Node rightPt;

Node (int data) {

this.data = data;

this.leftPt = null;

this.rightPt = null;

}

}

public class Main {

static HeadPt head;

enum ErrCode {

SUCCESS,

INCORRECT\_DATA,

TREE\_NOT\_EXIST,

SUCH\_ELEMENT\_ALREADY\_EXIST

}

static final String[] ERRORS = {"Удача",

"Данные некорректные или число слишком большое (должно быть от %d до %d)\n",

"Сначала стоит создать дерево)",

"Такой элемент уже существует"};

enum Choice {

createTree("Создать новое дерево"),

addElem("Добавить элемент"),

print("Визуализировать дерево"),

close("Закрыть");

private final String inf;

Choice (String infLine) {

this.inf = infLine;

}

private String getInf(){return this.ordinal() + ") " + this.inf;}

}

static final int MIN\_NODE = -99999,

MAX\_NODE = 99999;

static final char ROOT\_CHAR = '+',

LEFT\_CHAR = 'L',

RIGHT\_CHAR = 'R';

static final String INFORMATION\_TEXT = """

Инструкция:

1) Элементы дерева должны быть от -99999 до 99999

2) Элементы не могут повторяться

""",

ATTENTION\_TEXT = """

Внимание! Старое дерево удалится, вы уверены?

1) Да

2) Нет

""";

static HeadPt createTree(int elem) {

Node firstNode = new Node(elem);

return new HeadPt(firstNode);

}

static boolean addElem(Node head, int elem) {

boolean isAdded = true;

if (elem > head.data) {

if (head.rightPt != null)

isAdded = addElem(head.rightPt, elem);

else

head.rightPt = new Node(elem);

} else if (elem < head.data) {

if (head.leftPt != null)

isAdded = addElem(head.leftPt, elem);

else

head.leftPt = new Node(elem);

} else {

isAdded = false;

}

return isAdded;

}

static void printTree(Node node, int layer, char side) {

if (node.rightPt != null)

printTree(node.rightPt, layer + 1, RIGHT\_CHAR);

for (int i = 0; i < layer; i++)

System.out.print(" ");

System.out.println("(" + side + ")" + node.data);

if (node.leftPt != null)

printTree(node.leftPt, layer + 1, LEFT\_CHAR);

}

static void printMenu() {

Choice[] choices = Choice.values();

for (Choice choice : choices) {

System.out.println(choice.getInf());

}

}

static void printInf(Scanner input) {

System.out.println(INFORMATION\_TEXT);

System.out.println("нажмите enter чтобы продолжить");

input.nextLine();

}

static ErrCode enterOneNum(int[] numberArr, Scanner input, final int MIN, final int MAX) {

int number = 0;

ErrCode err = ErrCode.SUCCESS;

try {

number = Integer.parseInt(input.nextLine());

} catch (NumberFormatException e) {

err = ErrCode.INCORRECT\_DATA;

}

if ((err == ErrCode.SUCCESS) && (number < MIN || number > MAX))

err = ErrCode.INCORRECT\_DATA;

numberArr[0] = err == ErrCode.SUCCESS ? number : 0;

return err;

}

static int getNumConsole(Scanner input, final int MIN, final int MAX) {

ErrCode err;

int[] numberArr = {0};

do {

err = enterOneNum(numberArr, input, MIN, MAX);

if (err != ErrCode.SUCCESS) {

System.err.printf(ERRORS[err.ordinal()], MIN, MAX);

System.out.println("Введите снова");

}

} while (err != ErrCode.SUCCESS);

return numberArr[0];

}

static Choice getChoice(Scanner input) {

int choice;

int maxChoice = Choice.values().length - 1;

choice = getNumConsole(input, 0, maxChoice);

return Choice.values()[choice];

}

static boolean doFunction(Scanner input) {

Choice choice = getChoice(input);

boolean isClose = false;

switch (choice) {

case createTree -> {

System.out.println(ATTENTION\_TEXT);

int localChoice = getNumConsole(input, 1, 2);

if (localChoice == 1) {

System.out.print("Введите корень дерева: ");

int root = getNumConsole(input, MIN\_NODE, MAX\_NODE);

head = createTree(root);

}

}

case addElem -> {

if (head != null) {

System.out.println("Введите новый элемент: ");

int newElem = getNumConsole(input, MIN\_NODE, MAX\_NODE);

if (!addElem(head.root, newElem)) {

System.err.println(ERRORS[ErrCode.SUCH\_ELEMENT\_ALREADY\_EXIST.ordinal()]);

}

}

else

System.err.println(ERRORS[ErrCode.TREE\_NOT\_EXIST.ordinal()]);

}

case print -> {

if (head != null)

printTree(head.root, 0, ROOT\_CHAR);

else

System.err.println(ERRORS[ErrCode.TREE\_NOT\_EXIST.ordinal()]);

System.out.println();

}

case close -> isClose = true;

}

return isClose;

}

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

printInf(input);

boolean isClose;

do {

printMenu();

isClose = doFunction(input);

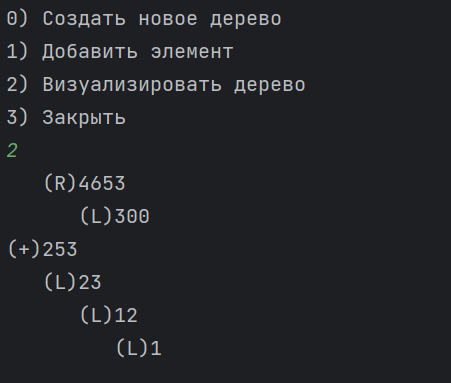
} while (!isClose);

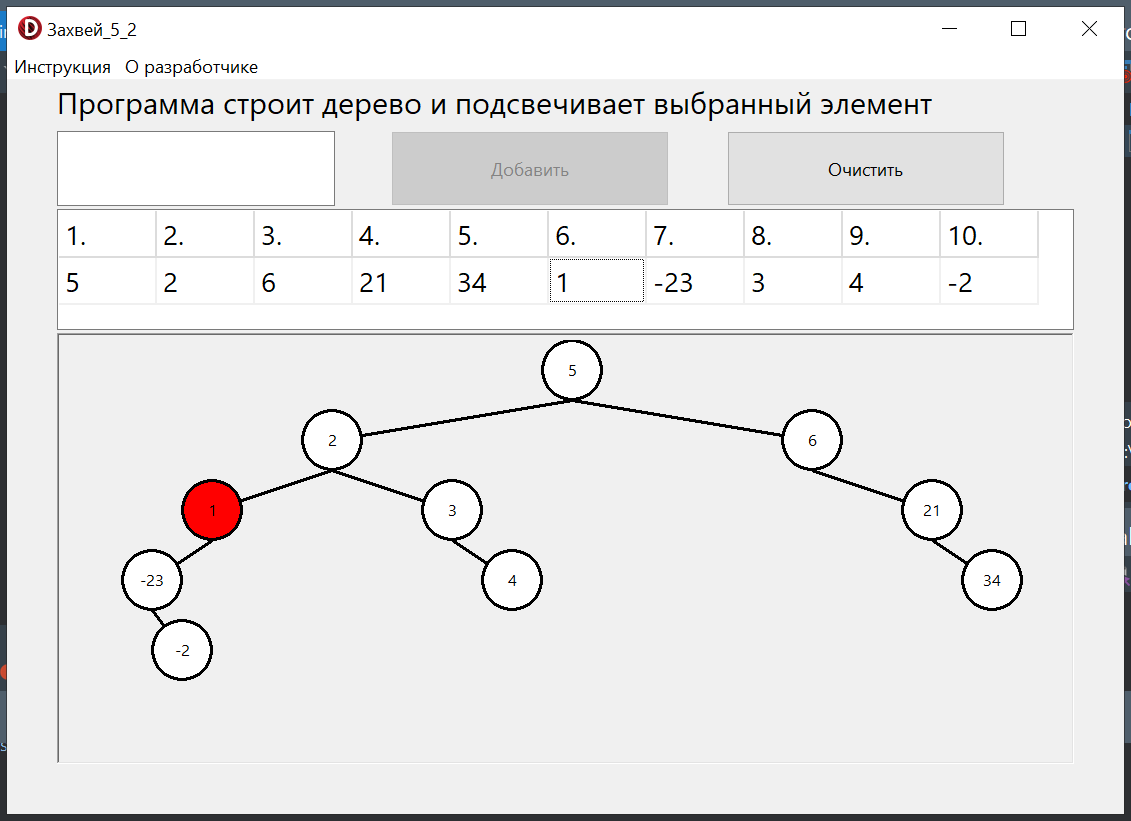
input.close();

}

}

**Скриншоты:**

**Delphi:** **Java:**

****

**Блок-схема:**

