МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛАРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Отчёт по лабораторной работе №1

на тему «Создание и обход бинарных деревьев»

по дисциплине «Алгоритмы и структуры данных»

Кафедра программного обеспечения информационных систем и технологий

Выполнил

студент группы 10701121

Проверил

М. Ю. Иванов

И.Д. Горновская

Минск 2023

**Цель работы:** Получение практических навыков объектно-ориентированного программирования задач с использованием бинарных деревьев.

# Задание по лабораторной работе

1. Создать приложение WindowsForms для работы с бинарным дерево поиска, содержащим не менее 4 уровней, заполняя элементы дерева целыми числами.

2. Вывести на экран дерево (все узлы дерева).

3. Реализовать в дереве методы в соответствии с вариантом

4. Создать программу для выполнения операций с деревом в соответствии с вариантом и отобразить выполнение этой программы на экране.

Вариант 10:

* Вставка нового узла с подсчётом количества операций сравнения
* Симметричный обход дерева с выводом значений ключей
* Удаление узла без потомков
* Определить количество потомков узлов на втором уровне

# Текст программы

Текст программы приведён ниже:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

namespace Lab01

{

public class TreeNode

{

public int value;

public TreeNode left;

public TreeNode right;

public TreeNode(int value)

{

this.value = value;

}

}

public class Tree

{

public TreeNode root;

static bool isLeft = false;

public static int compares = 0;

// Конструкторы

public Tree()

{

}

public Tree(int value)

{

this.root = new TreeNode(value);

}

public Tree(TreeNode root)

{

this.root = root;

}

//Прямой обход

private void PrefixIteration(TreeNode node, List<int> valuesList)

{

if (node != null)

{

valuesList.Add(node.value);

PrefixIteration(node.left, valuesList);

PrefixIteration(node.right, valuesList);

}

}

public List<int> Prefix()

{

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

PrefixIteration(this.root, list);

return list;

}

// Симметричный обход

private void InfixIteration(TreeNode node, List<int> valuesList)

{

if (node != null)

{

InfixIteration(node.left, valuesList);

valuesList.Add(node.value);

InfixIteration(node.right, valuesList);

}

}

public List<int> Infix()

{

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

InfixIteration(this.root, list);

return list;

}

// Обратный обход

private void PostfixIteration(TreeNode node, List<int> valuesList)

{

if (node != null)

{

InfixIteration(node.left, valuesList);

InfixIteration(node.right, valuesList);

valuesList.Add(node.value);

}

}

public List<int> Postfix()

{

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

PostfixIteration(this.root, list);

return list;

}

// Вставка

public void Insert(int value)

{

TreeNode treeNode = new TreeNode(value);

this.Insert(treeNode);

}

public void Insert(TreeNode treeNode)

{

compares = 0;

InsertIteration(ref this.root, ref treeNode);

}

private void InsertIteration(ref TreeNode parent, ref TreeNode child)

{

if (parent == null)

{

parent = child;

compares++;

}

else

{

if (child.value <= parent.value)

{

compares++;

InsertIteration(ref parent.left, ref child);

}

else

{

InsertIteration(ref parent.right, ref child);

}

}

}

// Удаление листа

public void Delete(int value)

{

TreeNode node = this.root;

isLeft = true;

if (node.value == value && node.left == null && node.right == null)

{

this.root = null;

}

else

{

DeleteIteration(ref node, ref node, value);

}

}

public void DeleteIteration(ref TreeNode parent, ref TreeNode child, int value)

{

if (child != null)

{

if (child.value == value)

{

if (child.left == null && child.right == null)

{

if (isLeft)

{

parent.left = null;

}

else

{

parent.right = null;

}

}

else

{

isLeft = true;

DeleteIteration(ref child, ref child.left, value);

}

}

else

{

if (child.value > value)

{

isLeft = true;

DeleteIteration(ref child, ref child.left, value);

}

else

{

isLeft = false;

DeleteIteration(ref child, ref child.right, value);

}

}

}

}

// Подсчёт числа потомков

public int CountChildsAfterSecondLevel()

{

int counter = 0;

if (root.left != null)

{

if (root.left.left != null)

{

counter += Count(root.left.left) - 1;

}

if (root.left.right != null)

{

counter += Count(root.left.right) - 1;

}

}

if (root.right != null)

{

if (root.right.left != null)

{

counter += Count(root.right.left) - 1;

}

if (root.right.right != null)

{

counter += Count(root.right.right) - 1;

}

}

return counter;

}

public static int Count(TreeNode node)

{

int counter = 1;

CountIteration(node, ref counter);

return counter;

}

private static void CountIteration(TreeNode node, ref int counter)

{

if (node != null)

{

if (node.left != null)

{

counter++;

CountIteration(node.left, ref counter);

}

if (node.right != null)

{

counter++;

CountIteration(node.right, ref counter);

}

}

}

// Прочее

public static Tree FromQueue(Queue<int> initialQueue)

{

Tree tree = new Tree();

while (initialQueue.Count > 0)

{

//tree.Insert(initialQueue.Dequeue());

TreeNode node = new TreeNode(initialQueue.Dequeue());

tree.Insert(node);

}

return tree;

}

public static void Draw(ref Graphics graphics, TreeNode tree, int x, int y, int dx = 50, int dy = 20)

{

if (tree != null)

{

Random random = new Random();

if (tree.left != null)

{

graphics.DrawLine(new Pen(Color.CornflowerBlue, 1.6f), new Point(x + 9, y + 9), new Point(x - dx + 9, y + dy + 9));

}

if (tree.right != null)

{

graphics.DrawLine(new Pen(Color.CornflowerBlue, 1.6f), new Point(x + 9, y + 9), new Point(x + dx + 9, y + dy + 9));

}

graphics.FillEllipse(new SolidBrush(Color.White), new Rectangle(x - 2, y - 1, 19, 19));

graphics.DrawEllipse(new Pen(Color.CornflowerBlue, 1.6f), new Rectangle(x - 2, y - 1, 19, 19));

string txt = tree.value.ToString();

if (txt.Length == 1)

{

txt = "0" + txt;

}

graphics.DrawString(txt, new Font("Arial", 8), new SolidBrush(Color.Black), x, y+1);

double cx = 0.8;

double cy = 1.1;

Draw(ref graphics, tree.left, x - dx, y + dy, (int)(dx \* cx), (int)(dy \* cy));

Draw(ref graphics, tree.right, x + dx, y + dy, (int)(dx \* cx), (int)(dy \* cy));

}

}

}

}

# 3 Результаты выполнения программы

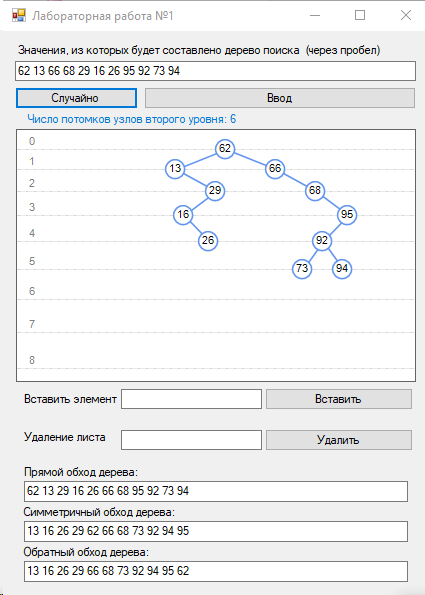


Рисунок 4.1 – Результаты выполнения программы.

# Оценка алгоритма

Сложность всех представленных алгоритмов (вставка, обход, поиск, удаление) стремится к логарифмической O(log n), если дерево сбалансировано.

# Выводы

Изучены бинарные деревья, бинарные деревья поиска. Рассмотрены алгоритмы их обхода, а также вставки, удаления и поиска элементов.