БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Лабораторная работа № 8

по дисциплине «Структуры и алгоритмы обработки данных»

Вариант № 4

Выполнил студент: Ефименко Павел Викторович,

группа 981063

Минск 2020

**Условие:**

Ввести 10-15 целых чисел и построить из них с помощью указателей бинарное дерево поиска. Обойти его прямым, симметричным и обратным способами. Реализовать процедуры поиска, вставки и удаления элементов бинарного дерева поиска.

**Листинг:**

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Ввести 10-15 целых чисел и построить из них с помощью указателей бинарное дерево поиска. Обойти его прямым, симметричным и обратным способами. Реализовать процедуры поиска, вставки и удаления элементов бинарного дерева поиска.");

var binaryTree = new BinaryTree<int>();

binaryTree.Add(8);

binaryTree.Add(3);

binaryTree.Add(10);

binaryTree.Add(1);

binaryTree.Add(6);

binaryTree.Add(4);

binaryTree.Add(7);

binaryTree.Add(14);

binaryTree.Add(16);

binaryTree.Add(5);

Console.WriteLine("Наше дерево");

binaryTree.PrintTree();

Console.WriteLine(new string('-', 40));

Console.WriteLine("Добавление 9");

binaryTree.Add(9);

binaryTree.PrintTree();

Console.WriteLine(new string('-', 40));

Console.WriteLine("Удаление 7");

binaryTree.Remove(7);

binaryTree.PrintTree();

Console.WriteLine(new string('-', 40));

Console.WriteLine("Поиск значения 14");

if (binaryTree.FindNode(14) != null)

{

Console.WriteLine("Значение есть");

}

else Console.WriteLine("Значения нет");

Console.ReadLine();

}

}

public enum Side

{

Left,

Right

}

public class BinaryTreeNode<T>

where T : IComparable

{

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="data">Данные</param>

public BinaryTreeNode(T data)

{

Data = data;

}

/// <summary>

/// Данные которые хранятся в узле

/// </summary>

public T Data { get; set; }

/// <summary>

/// Левая ветка

/// </summary>

public BinaryTreeNode<T> LeftNode { get; set; }

/// <summary>

/// Правая ветка

/// </summary>

public BinaryTreeNode<T> RightNode { get; set; }

/// <summary>

/// Родитель

/// </summary>

public BinaryTreeNode<T> ParentNode { get; set; }

/// <summary>

/// Расположение узла относительно его родителя

/// </summary>

public Side? NodeSide =>

ParentNode == null

? (Side?)null

: ParentNode.LeftNode == this

? Side.Left

: Side.Right;

/// <summary>

/// Преобразование экземпляра класса в строку

/// </summary>

/// <returns>Данные узла дерева</returns>

public override string ToString() => Data.ToString();

}

public class BinaryTree<T> where T : IComparable

{

public BinaryTreeNode<T> RootNode { get; set; }

public BinaryTreeNode<T> Add(T data)

{

return Add(new BinaryTreeNode<T>(data));

}

public BinaryTreeNode<T> FindNode(T data, BinaryTreeNode<T> startWithNode = null)

{

startWithNode = startWithNode ?? RootNode;

int result;

return (result = data.CompareTo(startWithNode.Data)) == 0

? startWithNode

: result < 0

? startWithNode.LeftNode == null

? null

: FindNode(data, startWithNode.LeftNode)

: startWithNode.RightNode == null

? null

: FindNode(data, startWithNode.RightNode);

}

public void Remove(BinaryTreeNode<T> node)

{

if (node == null)

{

return;

}

var currentNodeSide = node.NodeSide;

//если у узла нет подузлов, можно его удалить

if (node.LeftNode == null && node.RightNode == null)

{

if (currentNodeSide == Side.Left)

{

node.ParentNode.LeftNode = null;

}

else

{

node.ParentNode.RightNode = null;

}

}

//если нет левого, то правый ставим на место удаляемого

else if (node.LeftNode == null)

{

if (currentNodeSide == Side.Left)

{

node.ParentNode.LeftNode = node.RightNode;

}

else

{

node.ParentNode.RightNode = node.RightNode;

}

node.RightNode.ParentNode = node.ParentNode;

}

//если нет правого, то левый ставим на место удаляемого

else if (node.RightNode == null)

{

if (currentNodeSide == Side.Left)

{

node.ParentNode.LeftNode = node.LeftNode;

}

else

{

node.ParentNode.RightNode = node.LeftNode;

}

node.LeftNode.ParentNode = node.ParentNode;

}

//если оба дочерних присутствуют,

//то правый становится на место удаляемого,

//а левый вставляется в правый

else

{

switch (currentNodeSide)

{

case Side.Left:

node.ParentNode.LeftNode = node.RightNode;

node.RightNode.ParentNode = node.ParentNode;

Add(node.LeftNode, node.RightNode);

break;

case Side.Right:

node.ParentNode.RightNode = node.RightNode;

node.RightNode.ParentNode = node.ParentNode;

Add(node.LeftNode, node.RightNode);

break;

default:

var bufLeft = node.LeftNode;

var bufRightLeft = node.RightNode.LeftNode;

var bufRightRight = node.RightNode.RightNode;

node.Data = node.RightNode.Data;

node.RightNode = bufRightRight;

node.LeftNode = bufRightLeft;

Add(bufLeft, node);

break;

}

}

}

public void Remove(T data)

{

var foundNode = FindNode(data);

Remove(foundNode);

}

public void PrintTree()

{

PrintTree(RootNode);

}

private void PrintTree(BinaryTreeNode<T> startNode, string indent = "", Side? side = null)

{

if (startNode != null)

{

var nodeSide = side == null ? "+" : side == Side.Left ? "L" : "R";

Console.WriteLine($"{indent} [{nodeSide}]- {startNode.Data}");

indent += new string(' ', 3);

//рекурсивный вызов для левой и правой веток

PrintTree(startNode.LeftNode, indent, Side.Left);

PrintTree(startNode.RightNode, indent, Side.Right);

}

}

public BinaryTreeNode<T> Add(BinaryTreeNode<T> node, BinaryTreeNode<T> currentNode = null)

{

if (RootNode == null)

{

node.ParentNode = null;

return RootNode = node;

}

currentNode = currentNode ?? RootNode;

node.ParentNode = currentNode;

int result;

return (result = node.Data.CompareTo(currentNode.Data)) == 0

? currentNode

: result < 0

? currentNode.LeftNode == null

? (currentNode.LeftNode = node)

: Add(node, currentNode.LeftNode)

: currentNode.RightNode == null

? (currentNode.RightNode = node)

: Add(node, currentNode.RightNode);

}

}

**Выполнение:**

