Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Рязанский государственный радиотехнический университет

имени В.Ф.Уткина»

Кафедра «Электронные вычислительные машины»

Отчет по лабораторной работе №8

на тему

«Деревья»

по дисциплине  
**«Визуальное программирование»**

Выполнили:

Студенты группы №140

Бригада 5

Сафаров Д. А.

Тимохин Е. С.

Проверили:  
ст. преп. Хизриева Н.И.

ст. преп. Бастрычкин А.С.

**Цель работы**

Реализовать работу с бинарными деревьями на языке C#

**Задание**

Реализовать обобщенное бинарное дерево поиска.

Использовать рекурсивные алгоритмы для реализации основных операций над деревом.

1) Создать класс Node<T> для хранения информации об узле дерева.

2) Создать класс Tree<T>. Класс должен реализовать интерфейс IEnumerable<T>.

3) Реализовать в классе дерева основные операции:

a. Find (T element)

b. Insert (T element)

c. Remove (T element)

d. GetEnumerator() – для выполнения прямого обхода

4) Продемонстрировать работу с деревьями на примерах Tree<int> и Tree<Item> (из ПР 7).

**Практическая часть**

Код программы представлен в приложении А. Результат её работы представлен на рисунке 1.

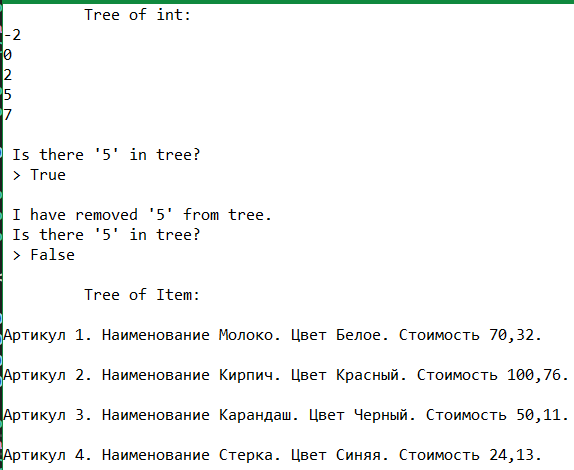


Рисунок 1 – Результаты работы программы

Приложение А. Текст программы

**Node.cs**

namespace TreesLogic

{

/// Класс - узел бинарного дерева

public class Node<T> : IComparable

where T : IComparable

{

public Node<T>? left;

public Node<T>? right;

public T data;

public Node(T data, Node<T>? left = null, Node<T>? right = null)

{

this.data = data;

this.left = left;

this.right = right;

}

public int CompareTo(object? obj)

{

return data.CompareTo(obj);

}

}

}

**Tree.cs**

using System.Collections;

namespace TreesLogic

{

/// Класс - бинарное дерево

public class Tree<T> : IEnumerable

where T : IComparable

{

// Корень дерева

private Node<T>? \_root = null;

// Поиск элемента в дереве

public bool Find(T element)

{

//if (\_root == null) throw new Exception("Tree is empty!");

return FindRecursive(\_root, element);

}

// Вставка элемента в дерево

public void Insert(T element) => \_root = InsertRecursive(\_root, element);

// Удаление элемента из дерева

public void Remove(T element)

{

if (\_root == null) throw new Exception("Tree is empty!");

RemoveRecursive(\_root, element);

}

// Реализация интерфейса

public IEnumerator<T> GetEnumerator() => Traversal(\_root).GetEnumerator();

// Реализация интерфейса

IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();

// Рекурсивный прямой обход дерева (НА САМОМ ДЕЛЕ СИММЕТРИЧНЫЙ)

private IEnumerable<T> Traversal(Node<T>? node)

{

if (node == null) yield break;

foreach (var item in Traversal(node.left))

yield return item;

yield return node.data;

foreach (var item in Traversal(node.right))

yield return item;

}

// Рекурсивный поиск элемента

private bool FindRecursive(Node<T>? node, T element)

{

if (node == null)

return false;

// Поиск в левом поддереве

if (element.CompareTo(node.data) < 0)

return FindRecursive(node.left, element);

// Поиск в правом поддереве

else if (element.CompareTo(node.data) > 0)

return FindRecursive(node.right, element);

// Узел найден

else return true;

}

// Рекурсивная вставка элемента

private Node<T>? InsertRecursive(Node<T>? node, T element)

{

if (node == null)

return new Node<T>(element);

if (element.CompareTo(node.data) < 0)

node.left = InsertRecursive(node.left, element);

else if (element.CompareTo(node.data) > 0)

node.right = InsertRecursive(node.right, element);

return node;

}

// Рекурсивное удаление элемента

private Node<T>? RemoveRecursive(Node<T>? node, T element)

{

if (node == null)

return null;

if (element.CompareTo(node.data) < 0)

node.left = RemoveRecursive(node.left, element);

else if (element.CompareTo(node.data) > 0)

node.right = RemoveRecursive(node.right, element);

else // Найден узел для удаления

{

if (node.left == null)

return node.right;

else if (node.right == null)

return node.left;

// Если у узла есть два потомка

Node<T> temp = node.right;

while (temp.left != null)

temp = temp.left;

node.data = temp.data;

node.right = RemoveRecursive(node.right, temp.data);

}

return node;

}

}

}

**Program.cs**

using TreesLogic;

using ItemLogic;

Tree<int> TreeOfInt = new();

TreeOfInt.Insert(5);

TreeOfInt.Insert(-2);

TreeOfInt.Insert(0);

TreeOfInt.Insert(7);

TreeOfInt.Insert(2);

Console.WriteLine("\t Tree of int:");

foreach (var item in TreeOfInt)

Console.WriteLine(item);

Console.WriteLine("\n Is there '5' in tree?");

Console.WriteLine(" > " + TreeOfInt.Find(5));

TreeOfInt.Remove(5);

Console.WriteLine("\n I have removed '5' from tree.");

Console.WriteLine(" Is there '5' in tree?");

Console.WriteLine(" > " + TreeOfInt.Find(5));

Tree<Item> TreeOfItem = new();

TreeOfItem.Insert(new Item(2, "Кирпич", "Красный", 100.76));

TreeOfItem.Insert(new Item(1, "Молоко", "Белое", 70.32));

TreeOfItem.Insert(new Item(4, "Стерка", "Синяя", 24.13));

TreeOfItem.Insert(new Item(3, "Карандаш", "Черный", 50.11));

Console.WriteLine("\n\t Tree of Item:\n");

foreach (var item in TreeOfItem)

Console.WriteLine(item);