Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Организация работы бинарного сбалансированного дерева для хранения и обработки данных
Тема

Преподаватель _____ Р. Ю. Царев
подпись, дата | инициалы, фамилия

Студент КИ19-17/1б 031939175 _____ А. Д. Непомнящий
номер группы, зачетной книжки | подпись, дата | инициалы, фамилия

Красноярск 2021

1 Цель работы

Изучение такой структуры данных, как бинарное сбалансированное дерево.

2 Задачи

Написать программу, реализующую бинарное сбалансированное дерево для хранения данных в соответствии с вариантом.

Предъявлены следующие требования к выполнению работы.

1. Строгое соответствие программы и результатов ее работы с полученным заданием.

2. Самостоятельные тестирование и отладка программы.

3. Предоставление демонстрационного примера и исходного текста программы для защиты.

4. Предоставление отчета по практическому заданию, содержащего описание реализованного алгоритма, программы, результатов работы программы (отчет необходимо загрузить на сайт курса).

3 Описание реализованной структуры данных

Реализовано АВЛ-дерево. Реализованы методы для добавления, удаления, поиска вершин, обхода дерева тремя методами.

4 Описание программы

Для решения задачи была написана программа на языке C#. Было создано шесть классов.

Классы BinarySearchTree и BTreeNode совместно реализуют бинарное дерево и все операции над ним. Бинарное дерево может хранить любые данные, которые могут быть сравнены между собой (имплементируют интерфейс IComparable). Для смены поля, по которому осуществляется сортировка, метод сравнения CompareTo() должен быть переопределен необходимым образом, а дерево собрано заново.

Класс TreeDrawer необходим для вывода в консоль бинарного дерева в легко воспринимаемом виде.

Класс Laundry в соответствии с вариантом 14 хранит информацию о заказах в химчистке. Данные хранятся в виде экземпляров класса LaundryOrder с помощью бинарного дерева поиска.

Класс Program необходим для демонстрации примера работы.

Листинг 1 – Код в файле BTreeNode.cs

```
using System;

namespace Lab4
{
    public class BTreeNode <T> where T : IComparable
    {
        public int Height;
        public T Data { get; set; }
        internal BTreeNode <T> Left, Right;

        public BTreeNode(T data)
        {
            Data = data;
            Height = 1;
        }
    }
}
```

Листинг 2 – Код в файле BinarySearchTree.cs

```
using System;

namespace Lab4
{
    public class BinarySearchTree<T> where T : IComparable
    {
        private BTreeNode<T> _root;

        private static int Height(BTreeNode<T> node)
        {
            return node?.Height ?? 0;
        }

        private static BTreeNode<T> RightRotate(BTreeNode<T> y)
        {
            var x = y.Left;
            var t2 = x.Right;
```

Продолжение листинга 2

```
            x.Right = y;
            y.Left = t2;

            y.Height = Math.Max(Height(y.Left), Height(y.Right)) + 1;
            x.Height = Math.Max(Height(x.Left), Height(x.Right)) + 1;

            return x;
        }

        private static BTreeNode<T> LeftRotate(BTreeNode<T> x)
        {
            var y = x.Right;
            var t2 = y.Left;

            y.Left = x;
            x.Right = t2;

            x.Height = Math.Max(Height(x.Left), Height(x.Right)) + 1;
            y.Height = Math.Max(Height(y.Left), Height(y.Right)) + 1;

            return y;
        }

        private static int GetBalance(BTreeNode<T> node)
        {
            return node != null ? Height(node.Left) - Height(node.Right) : 0;
        }

        public void Insert(T data)
        {
            _root = Insert(_root, data);
        }

        private static BTreeNode<T> Insert(BTreeNode<T> bTreeNode, T data)
        {
```

Продолжение листинга 2

```
if (bTreeNode == null)
    return new BTreeNode<T>(data);

switch (data.CompareTo(bTreeNode.Data))
{
    case < 0:
        bTreeNode.Left = Insert(bTreeNode.Left, data);
        break;
    case > 0:
        bTreeNode.Right = Insert(bTreeNode.Right, data);
        break;
    default:
        return bTreeNode;
}

bTreeNode.Height = 1 + Math.Max(Height(bTreeNode.Left),
    Height(bTreeNode.Right));

var balance = GetBalance(bTreeNode);

switch (balance)
{
    case > 1 when data.CompareTo(bTreeNode.Left.Data) < 0:
        return RightRotate(bTreeNode);
    case < -1 when data.CompareTo(bTreeNode.Right.Data) > 0:
        return LeftRotate(bTreeNode);
    case > 1 when data.CompareTo(bTreeNode.Left.Data) > 0:
        bTreeNode.Left = LeftRotate(bTreeNode.Left);
        return RightRotate(bTreeNode);
    case < -1 when data.CompareTo(bTreeNode.Right.Data) < 0:
        bTreeNode.Right = RightRotate(bTreeNode.Right);
        return LeftRotate(bTreeNode);
    default:
        return bTreeNode;
}
```

Продолжение листинга 2

```
        }

        private static BTreeNode<T> MinValueNode(BTreeNode<T> bTreeNode)
        {
            var current = bTreeNode;

            while (current.Left != null)
                current = current.Left;

            return current;
        }

        private void Remove(T data)
        {
            _root = DeleteNode(_root, data);
        }

        public bool Remove(object value)
        {
            if (!FindValue(_root, value, out var data)) return false;
            Remove(data);
            return true;
        }

        public bool FindValue(object value, out T found)
        {
            if (FindValue(_root, value, out found)) return true;
            found = default;
            return false;
        }

        private bool FindValue(BTreeNode<T> node, object value, out T found)
        {
            if (node == null)
            {
```

Продолжение листинга 2

```
            found = default;
            return false;
        }

        switch (node.Data.CompareTo(value))
        {
            case < 0: return FindValue(node.Right, value, out found);
            case > 0: return FindValue(node.Left, value, out found);
            default:
                found = node.Data;
                return true;
        }
    }

    private static BTreeNode<T> DeleteNode(BTreeNode<T> root, T data)
    {
        if (root == null)
            return null;

        switch (data.CompareTo(root.Data))
        {
            case < 0:
                root.Left = DeleteNode(root.Left, data);
                break;
            case > 0:
                root.Right = DeleteNode(root.Right, data);
                break;
            default:
            {
                if (root.Left == null || root.Right == null)
                {
                    var temp = root.Left ?? root.Right;

                    root = temp;
                }
```

Продолжение листинга 2

```
            else
            {

                var temp = MinValueNode(root.Right);

                root.Data = temp.Data;

                root.Right = DeleteNode(root.Right, temp.Data);
            }

            break;
        }
    }

    if (root == null)
        return null;

    root.Height = Math.Max(Height(root.Left),
        Height(root.Right)) + 1;

    var balance = GetBalance(root);

    switch (balance)
    {
        case > 1 when GetBalance(root.Left) >= 0:
            return RightRotate(root);
        case > 1 when GetBalance(root.Left) < 0:
            root.Left = LeftRotate(root.Left);
            return RightRotate(root);
        case < -1 when GetBalance(root.Right) <= 0:
            return LeftRotate(root);
        case < -1 when GetBalance(root.Right) > 0:
            root.Right = RightRotate(root.Right);
            return LeftRotate(root);
        default:
```

Продолжение листинга 2

```
                return root;
            }
        }

        public enum TraverseOrder
        {
            Infix,
            Prefix,
            Postfix
        }

        public void TraverseTree(Action<T> action, TraverseOrder order =
TraverseOrder.Prefix)
        {
            switch (order)
            {
                case TraverseOrder.Infix:
                    InfixTraverse(_root, action);
                    break;
                case TraverseOrder.Prefix:
                    PrefixTraverse(_root, action);
                    break;
                case TraverseOrder.Postfix:
                    PostfixTraverse(_root, action);
                    break;
                default:
                    throw new ArgumentOutOfRangeException(nameof(order), order,
null);
            }
        }

        private static void InfixTraverse(BTreeNode<T> node, Action<T> action)
        {
            if (node == null) return;
            if (node.Left != null) InfixTraverse(node.Left, action);
            action(node.Data);
```

Окончание листинга 2

```
            if (node.Right != null) InfixTraverse(node.Right, action);
        }

        private static void PrefixTraverse(BTreeNode<T> node, Action<T> action)
        {
            if (node == null) return;
            action(node.Data);
            if (node.Left != null) PrefixTraverse(node.Left, action);
            if (node.Right != null) PrefixTraverse(node.Right, action);
        }

        private static void PostfixTraverse(BTreeNode<T> node, Action<T> action)
        {
            if (node == null) return;
            if (node.Left != null) PostfixTraverse(node.Left, action);
            if (node.Right != null) PostfixTraverse(node.Right, action);
            action(node.Data);
        }

        public void PrintTree()
        {
            TreeDrawer<T>.PrintNode(_root, "");
        }
    }
}
```

Листинг 3 – Код в файле TreeDrawer.cs

```
using System;

namespace Lab4
{
    public static class TreeDrawer<T> where T : IComparable
    {
        private const string Cross = " ├─";
        private const string Corner = " └──";
        private const string Vertical = " │ ";
```

Окончание листинга 3

```
        private const string Space = "    ";

        public static void PrintNode(BTreeNode<T> bTreeNode, string indent)
        {
            if (bTreeNode == null)
            {
                return;
            }
            Console.WriteLine(bTreeNode.Data);
            if (bTreeNode.Left != null) PrintChildNode(bTreeNode.Left, indent,
bTreeNode.Right == null);
            if (bTreeNode.Right != null) PrintChildNode(bTreeNode.Right, indent,
true);

            static void PrintChildNode(BTreeNode<T> node, string indent, bool
isLast)
            {
                Console.Write(indent);

                if (isLast)
                {
                    Console.Write(Corner);
                    indent += Space;
                }
                else
                {
                    Console.Write(Cross);
                    indent += Vertical;
                }

                PrintNode(node, indent);
            }
        }
    }
}
```

Листинг 4 – Код в файле LaundryOrder.cs

```
using System;

namespace Lab4
{
```

Окончание листинга 4

```
    public class LaundryOrder : IComparable
    {
        private int OrderId { get; }
        private string Name { get; }
        private string ClotheType { get; }



        public LaundryOrder(int id, string name, string clotheType)
        {
            OrderId = id;
            Name = name;
            ClotheType = clotheType;
        }
        public int CompareTo(object obj)
        {
            switch (obj)
            {
                case LaundryOrder laundryOrder:
                    return string.Compare(Name, laundryOrder.Name,
StringComparison.Ordinal);
                case string name:
                    return string.Compare(Name, name, StringComparison.Ordinal);
                default:
                    throw new ArgumentException("Object is not a LaundryOrder");
            }
        }

        public override string ToString()
        {
            return $"Order {OrderId} with clothe {ClotheType} by {Name}";
        }
    }
}
```

Листинг 5 – Код в файле Laundry.cs

```
using System;

namespace Lab4
{
    public class Laundry
```

Окончание листинга 5

```
    {
        private int _nextId = 1;

        private BinarySearchTree<LaundryOrder> Orders { get; }

        public void AddOrder(string name, string clotheType)
        {
            Orders.Insert(new LaundryOrder(_nextId++, name, clotheType));
        }

        public bool DeleteOrderByName(string name)
        {
            return Orders.Remove(name);
        }

        public string OrderToStringByName(string name)
        {
            return Orders.FindValue(name, out var found) ? found.ToString() :
null;
        }

        public void PrintAllOrders()
        {
            Orders.TraverseTree(x => Console.WriteLine(x.ToString()));
        }

        public void PrintAsTree()
        {
            Orders.PrintTree();
        }

        public Laundry()
        {
            Orders = new BinarySearchTree<LaundryOrder>();
        }
    }
}
```

Листинг 6 – Код в файле Program.cs

```
using System;
```

Продолжение листинга 6

```
namespace Lab4
{
    class Program
    {
        static void Main(string[] args)
        {
            BinarySearchTree<int> binarySearchTree = new();

            binarySearchTree.Insert(10);
            binarySearchTree.Insert(5);
            binarySearchTree.Insert(0);
            binarySearchTree.Insert(6);
            binarySearchTree.Insert(18);
            binarySearchTree.Insert(3);
            binarySearchTree.Insert(4);
            binarySearchTree.Insert(9);
            binarySearchTree.Insert(11);
            binarySearchTree.Insert(-1);
            binarySearchTree.Insert(1);
            binarySearchTree.Insert(2);

            binarySearchTree.PrintTree();

            Console.WriteLine("Infix traverse order:");
            binarySearchTree.TraverseTree(x => Console.Write($"{x} "),
BinarySearchTree<int>.TraverseOrder.Infix);
            Console.WriteLine("\nPostfix traverse order:");
            binarySearchTree.TraverseTree(x => Console.Write($"{x} "),
BinarySearchTree<int>.TraverseOrder.Postfix);
            Console.WriteLine("\nPrefix traverse order:");
            binarySearchTree.TraverseTree(x => Console.Write($"{x} "));
            Console.WriteLine("\nRemoving elements (true/false = found value in
list");

            Console.WriteLine($"Removing {5} {binarySearchTree.Remove(5)}");
            Console.WriteLine($"Removing {14} {binarySearchTree.Remove(14)}");
            Console.WriteLine($"Removing {1} {binarySearchTree.Remove(1)}");
            Console.WriteLine($"Removing {2} {binarySearchTree.Remove(2)}");
            Console.WriteLine($"Removing {0} {binarySearchTree.Remove(0)}");
            Console.WriteLine($"Removing {-1} {binarySearchTree.Remove(-1)}");
            Console.WriteLine($"Removing {-10} {binarySearchTree.Remove(-10)}");
```

Окончание листинга 6

```
            Console.WriteLine($"Removing {9} {binarySearchTree.Remove(9)}");
            Console.WriteLine("After removing");

            binarySearchTree.PrintTree();

            var laundry = new Laundry();
            laundry.AddOrder("Vasiliy", "Suit");
            laundry.AddOrder("Ivan", "Black shirt");
            laundry.AddOrder("Rodion", "Jeans");
            laundry.AddOrder("Jhon", "Leather bag");
            laundry.AddOrder("Pyotr", "White shirt");
            laundry.AddOrder("Bojack", "Coat");
            laundry.AddOrder("Sergey", "Sneakers");
            Console.WriteLine("List as tree");
            laundry.PrintAsTree();

            Console.WriteLine($"Found by name Bojack
{laundry.OrderToStringByName("Bojack")}");
            Console.WriteLine("Delete Vasiliy");
            laundry.DeleteOrderByName("Vasiliy");
            Console.WriteLine("List as tree");
            laundry.PrintAsTree();
            Console.WriteLine("List as raw list");
            laundry.PrintAllOrders();
        }
    }
}
```

5 Результаты работы программы

На рисунках 1 и 2 приведены скриншоты с результатами работы программы для бинарных деревья объектов int и объектов LaundryOrder соответственно.

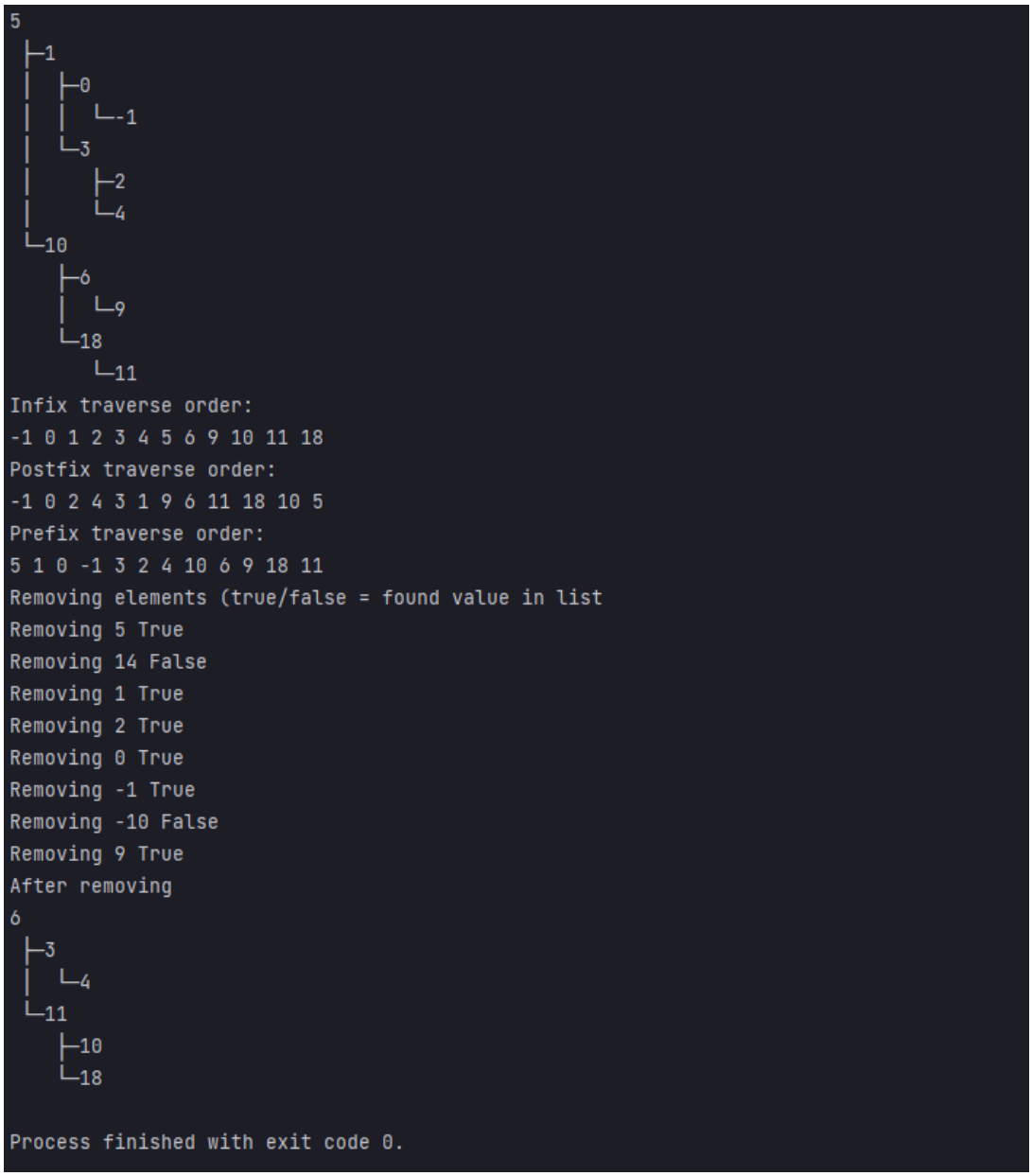

Рисунок 1 – Результат работы программы с деревом экземпляров int

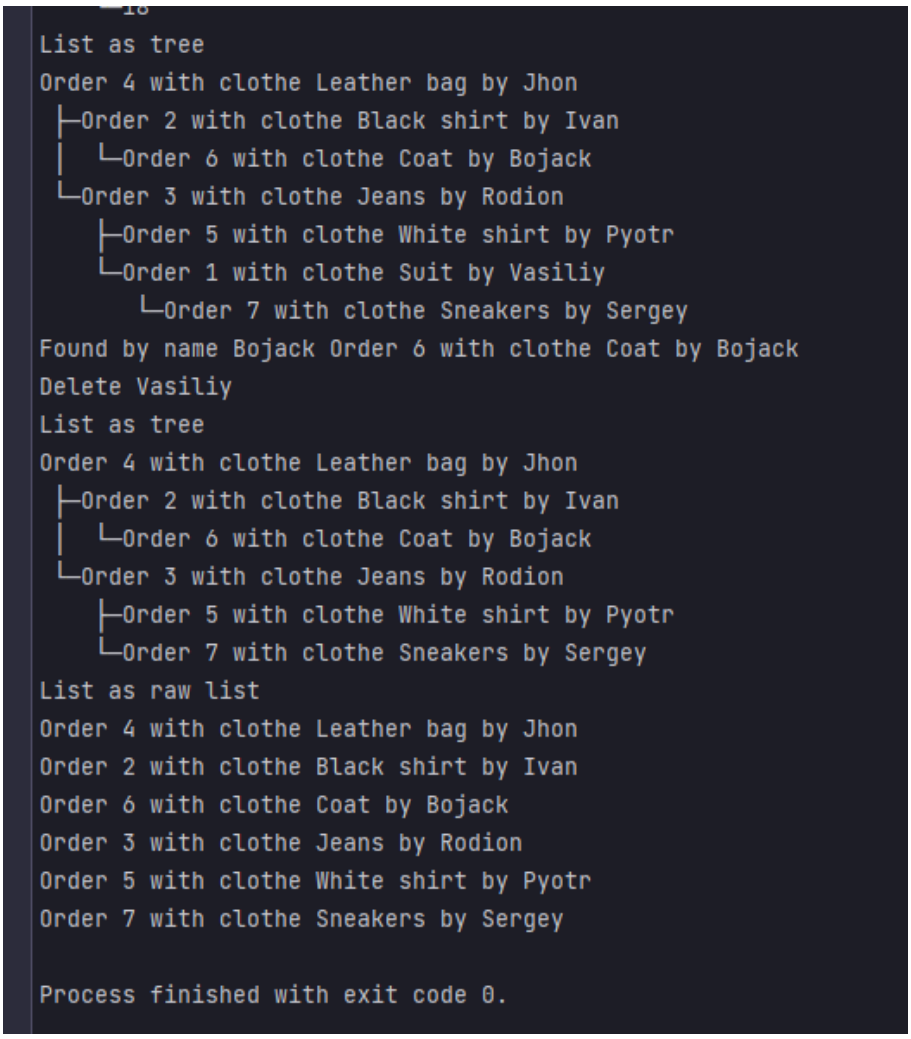

Рисунок 2 – Результат работы программы с деревом экземпляров LaundryOrder