МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

ОТЧЕТ

По лабораторной работе №4

«Бинарное дерево поиска»

По дисциплине «Л и ОА в ИЗ»

Выполнили: ст. гр. 22ВВ4
Жуков Илья
Чумаев Сабит

Приняли: Юрова О.В.
Акифьев И.В.

2023

Цель работы:

Написать код программы, выполнив следующие задания:

1.Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве.
2.Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.
3.* Изменить функцию добавления элементов для исключения добавления одинаковых символов.
4.* Оценить сложность процедуры поиска по значению в бинарном дереве.

Ход работы:

<u>Описание кода программы:</u>

```
using System;
using System.Collections.Generic;
using System.IO.Ports;
using System.Xml.Linq;

namespace Laba4
{
    //это объявление класса Node, который представляет узел в бинарном дереве. Он содержит публичные поля value, left и right,
    //которые представляют значение узла и ссылки на левого и правого потомков.
    class Node
    {
        public int value;
        public Node left;
        public Node right;

        //это конструктор класса Node, который принимает значение узла и создаёт новый узел с заданным значением
        public Node(int data)
        {
            value = data;
            left = null;
            right = null;
        }
    }

    //это объявление класса Binarytree, который представляет бинарное дерево.
    class Binarytree
    {
```

```
        //Он содержит приватное поле root, которое представляет корень дерева, и набор методов для работы с деревом.
        private Node root;

        // это конструктор класса Binarytree, который создаёт новый экземпляр бинарного дерева. В конструкторе инициализируется поле root значением null.
        public Binarytree()
        {
            root = null;
        }

        //это метод класса Binarytree, который добавляет новый узел с заданным значением в бинарное дерево
        public void Insert(int value)
        {
            //Если дерево пустое, создаётся корневой узел
            if (root == null)
            {
                root = new Node(value);
            }
            else
            {
            //Если значение уже присутствует в дереве, то выдает ошибку
                if (ContainsValue(root, value))
                {
               throw new ArgumentException("Element already exists in the tree");
                }
            //Иначе, вызывается вспомогательный метод InsertRecursive, который рекурсивно проходит по дереву и добавляет новый узел на правильное место.
                else
                {
                    InsertRecursive(root, value);
                }
            }
        }

      //это вспомогательный метод класса Binarytree, который рекурсивно проверяет, содержит ли дерево узел с заданным значением.
        private bool ContainsValue(Node currentNode, int value)
        {
         //Если достигнут конец дерева (текущий узел равен null), метод возвращает false
            if (currentNode == null)
            {
                return false;
            }
```

```
         //Если текущий узел равен заданному значению, метод возвращает
true.
            if (currentNode.value == value)
            {
                return true;
            }
         //Если заданное значение меньше значения текущего узла, метод
вызывает себя рекурсивно для левого потомка
            if (value < currentNode.value)
            {
                return ContainsValue(currentNode.left, value);
            }
            //Иначе метод вызывает себя рекурсивно для правого потомка
            else
            {
                return ContainsValue(currentNode.right, value);
            }
        }

      //это вспомогательный метод класса Binarytree, который рекурсивно
добавляет новый узел с заданным значением в бинарное дерево.
        private void InsertRecursive(Node currentNode, int value)
        {
         //Если значение меньше значения текущего узла, метод проверяет,
есть ли левый потомок у текущего узла
            if (value < currentNode.value)
            {
                //Если нет, создаётся новый узел с заданным значением
и присваивается полю left текущего узла
                if (currentNode.left == null)
                {
                    currentNode.left = new Node(value);
                }
            //Если левый потомок уже существует, метод вызывает себя
рекурсивно для левого потомка.
                else if (currentNode.left.value != value)
                {
                   InsertRecursive(currentNode.left, value);
                }
                else
                {
               //Иначе, метод выбрасывает исключение ArgumentException
с сообщением "Элемент уже существует в дереве".
                throw new ArgumentException("Элемент уже существует
в дереве");
                }
            }
            //Аналогичные действия выполняются, если значение больше
значения текущего узла и для правого потомка.
            else if (value > currentNode.value)
```

```
            {
                if (currentNode.right == null)
                {
                    currentNode.right = new Node(value);
                }
                else if (currentNode.right.value != value)
                {
                    InsertRecursive(currentNode.right, value);
                }
                else
                {
               throw new ArgumentException("Элемент уже существует
в дереве");
                }
            }
            //Иначе, метод выбрасывает исключение ArgumentException с
сообщением "Элемент уже существует в дереве".
            else
            {
                throw new ArgumentException("Элемент уже существует в
дереве");
            }
        }

      //это метод класса Binarytree, который возвращает количество узлов
с заданным значением в бинарном дереве.
      //Метод вызывает вспомогательный метод CountOccurrencesRecursive,
который рекурсивно проходит по дереву и
      //увеличивает счётчик, если текущий узел имеет заданное значение.
        public int CountOccurrences(int value)
        {
            return CountOccurrencesRecursive(root, value);
        }

      //это вспомогательный метод класса Binarytree, который рекурсивно
подсчитывает количество узлов с заданным значением в бинарном дереве
      private int CountOccurrencesRecursive(Node currentNode, int value)
        {
            if (currentNode == null)
            {
                return 0;
            }

            int count = 0;
            //Если текущий узел равен заданному значению, счётчик
увеличивается на 1.
            if (currentNode.value == value)
            {
                count = 1;
            }
```

```
         //Затем метод вызывает себя рекурсивно для левого и правого потомков текущего узла и суммирует результаты.
         count += CountOccurrencesRecursive(currentNode.left, value);
         count += CountOccurrencesRecursive(currentNode.right, value);

            return count;
        }
      //это метод класса Binarytree, который выводит на экран значения узлов в бинарном дереве.
        //Метод вызывает вспомогательный метод PrintRecursive, который рекурсивно проходит по дереву и выводит значения узлов.
        public void Print()
        {
            PrintRecursive(root, "", NodePosition.Center);
        }
      //Это вспомогательный метод класса Binarytree, который рекурсивно выводит значения узлов в бинарном дереве.
        private void PrintRecursive(Node currentNode, string indent, NodePosition position)
        {
            //Если текущий узел не равен null, метод выводит значение текущего узла, вызывает себя рекурсивно для левого потомка,
            //а затем вызывает себя рекурсивно для правого потомка.
            if (currentNode != null)
            {
                Console.Write(indent);
                if (position == NodePosition.Left)
                {
                    Console.Write("├─");
                    indent += "│  ";
                }
                else if (position == NodePosition.Right)
                {
                    Console.Write("└─");
                    indent += "   ";
                }
                Console.WriteLine(currentNode.value);

                // Рекурсивный вызов для элементов в левой ветке
                PrintRecursive(currentNode.left,                indent, NodePosition.Left);

                // Рекурсивный вызов для элементов в правой ветке
                PrintRecursive(currentNode.right,                indent, NodePosition.Right);
            }
        }

        private enum NodePosition
```

```
        {
            Center,
            Left,
            Right
        }

        //Метод "Search" используется для поиска значения в двоичном
дереве
        public bool Search(int value)
        {
         //Он вызывает метод "SearchRecursive" для выполнения поиска,
начиная с корневого узла.
            Node result = SearchRecursive(root, value);
            return result != null;
        }

        //Метод "SearchRecursive" - это частный вспомогательный метод,
который рекурсивно выполняет поиск значения в двоичном дереве.
        //Он сравнивает значение с текущим значением узла и рекурсивно
выполняет поиск в левом или правом поддереве до тех пор,
        //пока не найдет соответствующее значение или не достигнет
нулевого узла.
        private Node SearchRecursive(Node currentNode, int value)
        {
            if (currentNode == null || currentNode.value == value)
            {
                return currentNode;
            }

            if (value < currentNode.value)
            {
                return SearchRecursive(currentNode.left, value);
            }

            return SearchRecursive(currentNode.right, value);
        }
      //Метод, который удаляет узел с заданным значением из бинарного
дерева. Метод вызывает вспомогательный метод DeleteRecursive,
      //который рекурсивно проходит по дереву и удаляет узел с заданным
значением.
        public void Delete(int value)
        {
            root = DeleteRecursive(root, value);
        }

        //Метод, который рекурсивно удаляет узел с заданным значением
из бинарного дерева и возвращает изменённый корень дерева.
        private Node DeleteRecursive(Node currentNode, int value)
        {
            //Если текущий узел равен null, метод возвращает null.
```

```
            if (currentNode == null)
            {
                return null;
            }
        //Если заданное значение меньше значения текущего узла, метод
вызывает себя рекурсивно для левого потомка
            if (value < currentNode.value)
            {
                currentNode.left  =  DeleteRecursive(currentNode.left,
value);
            }
        //Если заданное значение больше значения текущего узла, метод
вызывает себя рекурсивно для правого потомка
            else if (value > currentNode.value)
            {
                currentNode.right = DeleteRecursive(currentNode.right,
value);
            }
            //Если заданное значение равно значению текущего узла,
выполняется удаление узла
            else
            {
                //Узел без потомков возвращается null
            if (currentNode.left == null && currentNode.right == null)
                {
                    return null;
                }

            //Узел с одним потомком: возвращается ссылка на потомка.
                if (currentNode.left == null)
                {
                    return currentNode.right;
                }
                else if (currentNode.right == null)
                {
                    return currentNode.left;
                }

                //Узел с двумя потомками
            //Находим минимальный узел в правом поддереве, заменяется
значение текущего узла на найденное минимальное значение,
                //а затем вызывается рекурсивное удаление для правого
поддерева, чтобы удалить повторяющееся значение.
                int minValue = FindMinValue(currentNode.right);
                currentNode.value = minValue;
                currentNode.right = DeleteRecursive(currentNode.right,
minValue);
            }

            return currentNode;
```

```
        }

        private int FindMinValue(Node currentNode)
        {
            //Создается переменная "minValue" и присваивается значение свойства объекта "currentNode" с именем "value".
            int minValue = currentNode.value;
         //Запускается цикл while, который продолжается, пока у объекта "currentNode" есть левый потомок (left != null).
            while (currentNode.left != null)
            {
                //Внутри цикла значение свойства объекта "currentNode.left.value" присваивается переменной "minValue".
                minValue = currentNode.left.value;
             //Затем объект "currentNode" обновляется, присваивая ему значение объекта "currentNode.left".
                currentNode = currentNode.left;
            }
            return minValue;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
         //это создание нового экземпляра класса Binarytree, который представляет бинарное дерево.
            Binarytree tree = new Binarytree();

            Console.Write("Введите значения для дерева разделенные пробелом: ");
            string input = Console.ReadLine();
            string[] values = input.Split(' ');

            foreach (string value in values)
            {
                int num = int.Parse(value);
                tree.Insert(num);
            }

            string menuOption;
            int deleteValue;
            int insertValue;
            int searchValue;
            int count;
            int searchValueCheck;
            bool result;

            do
```

```
            {
                Console.WriteLine("Выберите опцию:");
                Console.WriteLine("1.Просмотр элементов дерева");
                Console.WriteLine("2.Удаление элемента из дерева");
                Console.WriteLine("3.Добавление элемента в дерево");
             Console.WriteLine("4.Подсчитать число вхождений заданного
элемента в дерево");
                Console.WriteLine("5.Поиск элемента в дереве");
                Console.WriteLine("6.Выход");

                menuOption = Console.ReadLine();

                switch (menuOption)
                {
                    case "1":
                        Console.WriteLine("Элементы дерева: ");
                        tree.Print();
                        Console.WriteLine();
                        break;
                   case "2":
                        Console.Write("Введите значение для удаления:
");
                        deleteValue = int.Parse(Console.ReadLine());
                        tree.Delete(deleteValue);
                  Console.WriteLine($"Удаление элемента {deleteValue}
завершено");
                        break;
                   case "3":
                  Console.Write("Введите значение для добавления:
");
                        insertValue = int.Parse(Console.ReadLine());
                        tree.Insert(insertValue);
                        Console.WriteLine();
                        break;
                   case "4":
                  Console.Write("Введите число для поиска в дереве:
");
                        searchValue = int.Parse(Console.ReadLine());
                        count = tree.CountOccurrences(searchValue);
                        Console.WriteLine($"Значение найдено в дереве
{count} раз(а)");
                        break;
                   case "5":
                        Console.Write("Введите число для проверки: ");
                  searchValueCheck = int.Parse(Console.ReadLine());
                        result = tree.Search(searchValueCheck);

                        if (result)
                        {
                     Console.WriteLine("Значение найдено в дереве");
```

```
                        break;
                    }

                    else
                    {
                     Console.WriteLine("Значение не найдено в дереве");
                    }
                    break;
                case "6":
                    Console.WriteLine("Программа завершена");
                    break;
                default:
                    Console.WriteLine("Неверная опция, попробуйте
ещё раз");
                    break;
            }
            Console.WriteLine();
        } while (menuOption != "6");
    }
}
```

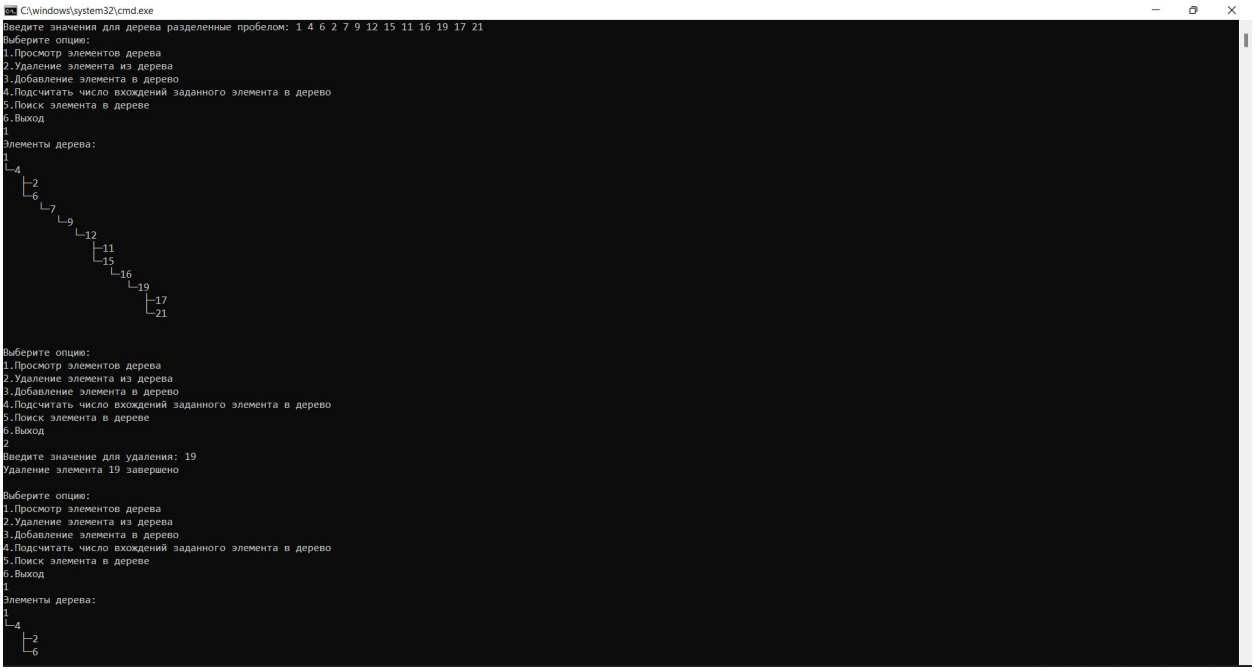
Оценка сложности процедуры поиска по значению в бинарном дереве:

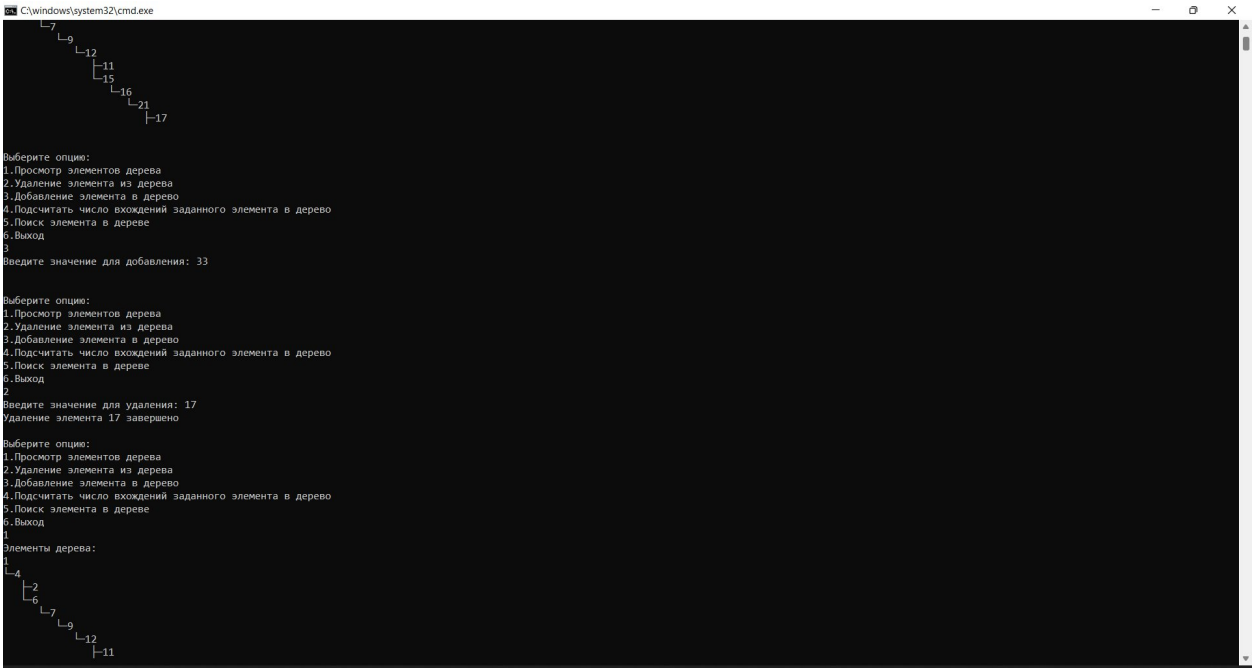
Временная сложность процедуры поиска в двоичном дереве зависит от формы дерева. В худшем случае, когда двоичное дерево искажено и напоминает связанный список, временная сложность процедуры поиска будет равна O(n), где n - количество узлов в дереве. Это связано с тем, что в этом случае нам, возможно, придется посетить каждый узел в дереве, прежде чем найти целевое значение.

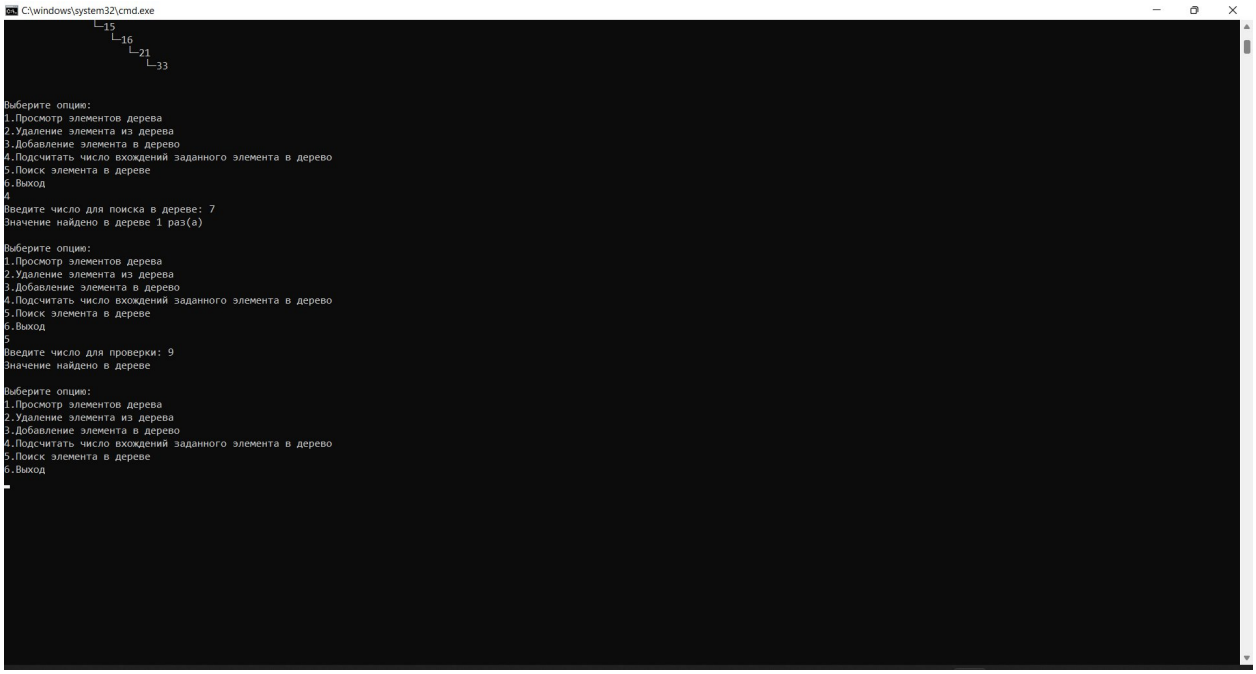
Однако в среднем и наилучшем сценарии, когда двоичное дерево сбалансировано, временная сложность процедуры поиска будет равна O(log n), где n - количество узлов в дереве. Это связано с тем, что на каждом шаге поиска мы можем отбросить половину оставшихся узлов в дереве.

Таким образом, общая сложность процедуры поиска в двоичном дереве может быть описана как O(log n) в среднем и наилучшем случаях и O(n) в наихудшем сценарии.

Результат работы программы:







Вывод: в данной лабораторной работе мы научились работать с «бинарным деревом». Выполнили следующие задания: реализацию алгоритма поиска вводимого с клавиатуры значения в уже созданном дереве, реализацию функции подсчёта числа вхождений заданного элемента в дерево, изменение функции добавления элементов для исключения добавления одинаковых символов. А также оценили сложность процедуры поиска по значению в бинарном дереве.