МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ВТ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Программирование в среде .NET»
Тема: РЕАЛИЗАЦИЯ БАЗОВЫХ АЛГОРИТМОВ СРЕДСТВАМИ ЯЗЫКА C#

Студент гр. 6306 _____ Жукаускайте А.В.

Преподаватель _____ Пешехонов К.А.

Санкт-Петербург
2020

Цель работы

Реализовать несколько базовых алгоритмов в среде .NET с помощью языка C#.

Задание

1) Реализовать связный список: создание, удаление, добавление произвольных элементов, реверс списка - без использования стандартных коллекций/LINQ (только IEnumerable);

2) Реализовать бинарное дерево: заполнение, поиск, удаление элемента - без использования стандартных деревьев;

3) Реализовать сортировку вставками - без .OrderBy().

Код программы

Класс LinkedList (здесь же реализована сортировка):

```
public class LinkedList<T>: IEnumerable<T> where T : IComparable, new()
{
    private Node<T> _begin;
    private Node<T> _end;
    private int _count;

    private bool IsEmpty => _count == 0;

    public LinkedList(int size)
    {
        if (size <= 0) return;

        for (int i = 0; i < size; i++)
            Add(new T());
    }

    public void Add(T data)
    {
        Node<T> node = new Node<T>(data);
        if (_begin == null)
            _begin = node;
        else
            _end.next = node;
        _end = node;
        _count++;
    }

    public bool Remove(T data)
    {
        if (IsEmpty) return false;
        Node<T> current = _begin;
        Node<T> previous = null;
        bool isFound = false;
```

```
    for (int i = 0; i < _count; i++)
    {
        if (current.data.Equals(data))
        {
            isFound = true;
            break;
        }
        previous = current;
        current = current.next;
    }

    if (!isFound) return false;

    if (previous == null)
    {
        if (current.next == null)
        {
            _begin = null;
            _end = null;
        }
        else
            _begin = current.next;
    }
    else
    {
        if (current.next == null)
        {
            previous.next = null;
            _end = previous;
        }
        else
            previous.next = current.next;
    }
    _count--;
    return true;
}

public void Output()
{
    if(IsEmpty) Console.Write("Empty");
    Node<T> node = _begin;
    while(node != null)
    {
        Console.Write("{0} ", node.data);
        node = node.next;
    }
    Console.Write('\n');
}

public void Reverse()
{
    if (_count < 2) return;
    Node<T> previousNode, currentNode = _begin;
    _begin = _end;
    _end = currentNode;
    previousNode = currentNode;
    currentNode = currentNode.next;
    if (currentNode.next == null)
    {
```

```
            currentNode.next = previousNode;
            previousNode.next = null;
        }
        else
        {
            previousNode.next = null;
            while (currentNode != null)
            {
                Node<T> nextNode = currentNode.next;
                currentNode.next = previousNode;
                previousNode = currentNode;
                currentNode = nextNode;
            }
        }
    }

    public void Sort()
    {
        Node<T> newBegin = null;
        Node<T> newEnd = _begin;
        while (_begin != null)
        {
            Node<T> node = _begin;
            if (_begin.CompareTo(newEnd) > 0)
                newEnd = _begin;
            _begin = _begin.next;
            if (newBegin == null || node.CompareTo(newBegin) < 0)
            {
                node.next = newBegin;
                newBegin = node;
            }
            else
            {
                Node<T> current= newBegin;
                while (current.next != null && node.CompareTo(current.next) >= 0)
                {
                    current = current.next;
                }
                node.next = current.next;
                current.next = node;
            }
        }
        _begin = newBegin;
        _end = newEnd;
    }

    public IEnumerator<T> GetEnumerator()
    {
        Node<T> current = _begin;

        while (current != null)
        {
            yield return current.data;
            current = current.next;
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
```

```
        return ((IEnumerable<T>)this).GetEnumerator();
    }
}
```

Класс Tree:

```
public class Tree<T> where T : IComparable
{
    public TreeNode<T> RootNode { get; set; }

    public void Add(TreeNode<T> node, TreeNode<T> currentNode)
    {
        if (RootNode == null)
        {
            node.ParentNode = null;
            RootNode = node; return;
        }
        node.ParentNode = currentNode;
        int result = node.Data.CompareTo(currentNode.Data);
        if (result < 0)
        {
            if(currentNode.LeftNode != null)
                Add(node, currentNode.LeftNode);
            else
                currentNode.LeftNode = node;
        }
        else
        {
            if (currentNode.RightNode != null)
                Add(node, currentNode.RightNode);
            else
                currentNode.RightNode = node;
        }
    }

    public void Add(T data)
    {
        Add(new TreeNode<T>(data), RootNode);
    }

    public TreeNode<T> FindNode(T data, TreeNode<T> startWithNode = null)
    {
        startWithNode ??= RootNode;
        int result = data.CompareTo(startWithNode.Data);
        if (result == 0)
            return startWithNode;
        if (result < 0)
            return startWithNode.LeftNode == null ? null : FindNode(data, start-
WithNode.LeftNode);
        return startWithNode.RightNode == null ? null : FindNode(data, startWith-
Node.RightNode);
    }

    public void Remove(TreeNode<T> node)
    {
       if (node == null) return;
       var currentNodeSide = node.NodeSide;
       if (node.LeftNode == null && node.RightNode == null)
```

```
        {
            if (currentNodeSide == Side.Left)
                node.ParentNode.LeftNode = null;
            else
                node.ParentNode.RightNode = null;
        }
        else //если нет левого, то правый ставим на место удаляемого
             if (node.LeftNode == null)
             {
                 if (currentNodeSide == Side.Left)
                     node.ParentNode.LeftNode = node.RightNode;
                 else
                     node.ParentNode.RightNode = node.RightNode;
                 node.RightNode.ParentNode = node.ParentNode;
             }
             else //если нет правого, то левый ставим на место удаляемого
                 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);
    }
```

```
    private void PrintTree(TreeNode<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);
            PrintTree(startNode.LeftNode, indent, Side.Left);
            PrintTree(startNode.RightNode, indent, Side.Right);
        }
    }

    public void PrintTree()
    {
        PrintTree(RootNode);
    }

}
```

Вывод

В процессе выполнения лабораторной работы были реализованы несколько стандартных структур данных средствами языка C# (связный список, бинарное дерево). Были получены навыки работы в среде .NET на языке C#.