**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной Техники**

**ОТЧЕТ**

**по лабораторной работе №1**

**по дисциплине «Программирование в среде DOTNet»**

**ТЕМА: Реализация базовых алгоритмов средствами языка C#**

|  |  |  |  |
| --- | --- | --- | --- |
| Студенты гр. 6307 | |  | Лазарев С. О. |
| Преподаватель | |  | Пешехонов К. А. |
|  |
|  |  |  |  |

Санкт-Петербург

2020

**Цели работы:**

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()

**CustomBinaryTree.cs**

|  |
| --- |
| using System; |
|  |  |
|  | namespace CustomBinaryTree |
|  | { |
|  | public enum Side |
|  | { |
|  | Left, |
|  | Right |
|  | } |
|  | public class BinaryTreeNode |
|  | { |
|  | public BinaryTreeNode(int data) |
|  | { |
|  | Data = data; |
|  | } |
|  |  |
|  | public int Data { get; set; } |
|  |  |
|  | public BinaryTreeNode LeftNode { get; set; } |
|  |  |
|  | public BinaryTreeNode RightNode { get; set; } |
|  |  |
|  | public BinaryTreeNode ParentNode { get; set; } |
|  |  |
|  | public Side? NodeSide => |
|  | ParentNode == null |
|  | ? (Side?)null |
|  | : ParentNode.LeftNode == this |
|  | ? Side.Left |
|  | : Side.Right; |
|  |  |
|  | public override string ToString() => Data.ToString(); |
|  | } |
|  | public class BinaryTree<T> where T : IComparable |
|  | { |
|  | public BinaryTreeNode RootNode { get; set; } |
|  |  |
|  | public BinaryTreeNode Add(BinaryTreeNode node, BinaryTreeNode 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); |
|  | } |
|  |  |
|  | public BinaryTreeNode Add(int data) |
|  | { |
|  | return Add(new BinaryTreeNode(data)); |
|  | } |
|  |  |
|  | public BinaryTreeNode FindNode(T data, BinaryTreeNode 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 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 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 if (node.RightNode == null) |
|  | { |
|  | if (currentNodeSide == Side.Left) |
|  | { |
|  | node.ParentNode.LeftNode = node.LeftNode; |
|  | } |
|  | else |
|  | { |
|  | node.ParentNode.RightNode = node.LeftNode; |
|  | } |
|  |  |
|  | node.LeftNode.ParentNode = node.ParentNode; |
|  | } |
|  | 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 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); |
|  | } |
|  | } |
|  | } |
|  | } |

## ****CustomLinkedList.cs****

|  |
| --- |
| using System; |
|  | using System.Collections; |
|  | using System.Collections.Generic; |
|  |  |
|  | namespace CustomLinkedList |
|  | { |
|  | public class Node<T> |
|  | { |
|  | public Node(T data) |
|  | { |
|  | Data = data; |
|  | } |
|  | public T Data { get; set; } |
|  | public Node<T> Next { get; set; } |
|  | } |
|  |  |
|  | public class LinkedList<T> : IEnumerable<T> |
|  | { |
|  | Node<T> head; |
|  | Node<T> tail; |
|  | int count; |
|  |  |
|  | public void Add(T data) |
|  | { |
|  | Node<T> node = new Node<T>(data); |
|  |  |
|  | if (head == null) |
|  | head = node; |
|  | else |
|  | tail.Next = node; |
|  | tail = node; |
|  |  |
|  | count++; |
|  | } |
|  |  |
|  | public void reverse() |
|  | { |
|  | Node<T> p = head, n = null; |
|  | while (p != null) |
|  | { |
|  | Node<T> tmp = p.Next; |
|  | p.Next = n; |
|  | n = p; |
|  | p = tmp; |
|  | } |
|  | head = n; |
|  | } |
|  |  |
|  | public bool Remove(T data) |
|  | { |
|  | Node<T> current = head; |
|  | Node<T> previous = null; |
|  |  |
|  | while (current != null) |
|  | { |
|  | if (current.Data.Equals(data)) |
|  | { |
|  |  |
|  | if (previous != null) |
|  | { |
|  |  |
|  | previous.Next = current.Next; |
|  |  |
|  | if (current.Next == null) |
|  | tail = previous; |
|  | } |
|  | else |
|  | { |
|  |  |
|  | head = head.Next; |
|  |  |
|  | if (head == null) |
|  | tail = null; |
|  | } |
|  | count--; |
|  | return true; |
|  | } |
|  |  |
|  | previous = current; |
|  | current = current.Next; |
|  | } |
|  | return false; |
|  | } |
|  |  |
|  | IEnumerator IEnumerable.GetEnumerator() |
|  | { |
|  | return ((IEnumerable)this).GetEnumerator(); |
|  | } |
|  |  |
|  | IEnumerator<T> IEnumerable<T>.GetEnumerator() |
|  | { |
|  | Node<T> current = head; |
|  | while (current != null) |
|  | { |
|  | yield return current.Data; |
|  | current = current.Next; |
|  | } |
|  | } |
|  | } |
|  | } |

## ****InsertionSort.cs****

|  |
| --- |
| namespace CustomInsertionSort |
|  | { |
|  | public class InsertionSort |
|  | { |
|  | public static int[] insertionSort(int[] array) |
|  | { |
|  | for (int i = 1; i < array.Length; i++) |
|  | { |
|  | int cur = array[i]; |
|  | int j = i; |
|  | while (j > 0 && cur < array[j - 1]) |
|  | { |
|  | array[j] = array[j - 1]; |
|  | j--; |
|  | } |
|  | array[j] = cur; |
|  | } |
|  | return array; |
|  | } |
|  | } |
|  | } |

## ****Program.cs****

|  |
| --- |
| using System; |
|  | using CustomBinaryTree; |
|  | using sort = CustomInsertionSort.InsertionSort; |
|  | using CustomLinkedList; |
|  |  |
|  | namespace dotnet |
|  | { |
|  | class Program |
|  | { |
|  | static void Main(string[] args) |
|  | { |
|  |  |
|  | /// Бинарное дерево /// |
|  |  |
|  | 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.PrintTree(); |
|  |  |
|  | Console.WriteLine(new string('-', 40)); |
|  | binaryTree.Remove(3); |
|  | binaryTree.PrintTree(); |
|  |  |
|  | Console.WriteLine(new string('-', 40)); |
|  | binaryTree.Remove(8); |
|  | binaryTree.PrintTree(); |
|  |  |
|  | /// Сортировка вставками /// |
|  |  |
|  | int[] array = { 6, 3, 1, 9, 5, 7, 2 }; |
|  |  |
|  | var newArray = sort.insertionSort(array); // 1, 2, 3, 5, 6, 7, 9 |
|  |  |
|  | /// Связный список /// |
|  |  |
|  | LinkedList<string> linkedList = new LinkedList<string>(); |
|  |  |
|  | linkedList.Add("Tom"); |
|  | linkedList.Add("Alice"); |
|  | linkedList.Add("Bob"); |
|  | linkedList.Add("Sam"); |
|  |  |
|  | // Tom, Alice, Bob, Sam |
|  |  |
|  | linkedList.Remove("Alice"); |
|  |  |
|  | // Tom, Bob, Sam |
|  |  |
|  | linkedList.reverse(); |
|  |  |
|  | // Sam, Bob, Tom |
|  | } |
|  | } |
|  | } |

**Вывод:**в ходе выполнения данной лабораторной работы мы реализовали 3 алгоритма.