**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра ВТ**

**ОТЧЕТ**

**по лабораторной работе №1**

**по дисциплине «Основы разработки корпоративных систем на платформе .NET»**

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

Студент гр. 6306 Гордиенко М. Е.

Преподаватель Пешехонов К. А.

Санкт-Петербург

2020

**Цель работы**

Реализация базовых алгоритмов с помощью языка C#.

**Задачи**

1. Реализация двусвязного списка
2. Реализация бинарного дерева
3. Реализация алгоритма сортировки

**Описание работы и примеры кода**

**Двусвязный список**

public class DoublyLinkedList<T> : IEnumerable<T> **{** private T Data { get; set; }  
 private DoublyLinkedList<T> Previous { get; set; }  
 private DoublyLinkedList<T> Next { get; set; }  
  
 private DoublyLinkedList<T> \_head;  
 private DoublyLinkedList<T> \_tail;  
  
 public DoublyLinkedList(T data) {  
 Data = data;  
 }  
  
  
 public void Add(T data) {  
 var node = new DoublyLinkedList<T>(data);  
  
 if (\_head == null)  
 \_head = node;  
 else {  
 \_tail.Next = node;  
 node.Previous = \_tail;  
 }  
 \_tail = node;  
 Count++;  
 }  
   
 public void AddFirst(T data) {  
 var node = new DoublyLinkedList<T>(data);  
 var temp = \_head;  
 node.Next = temp;  
 \_head = node;  
 if (Count == 0)  
 \_tail = \_head;  
 else  
 temp.Previous = node;  
 Count++;  
 }  
  
 public bool Remove(T data) {  
 var current = \_head;  
   
 while (current != null) {  
 if (current.Data.Equals(data))  
 {  
 break;  
 }  
 current = current.Next;  
 }  
  
 if (current == null) return false;  
  
 if(current.Next!=null) {  
 current.Next.Previous = current.Previous;  
 }  
 else {  
 \_tail = current.Previous;  
 }  
   
 if(current.Previous!=null) {  
 current.Previous.Next = current.Next;  
 }  
 else {  
 \_head = current.Next;  
 }  
 Count--;  
 return true;  
 }  
  
 private int Count { get; set; }  
 public bool IsEmpty => Count == 0;  
  
 public void Clear() {  
 \_head = null;  
 \_tail = null;  
 Count = 0;  
 }  
  
 public bool Contains(T data) {  
 var current = \_head;  
 while (current != null) {  
 if (current.Data.Equals(data))  
 return true;  
 current = current.Next;  
 }  
 return false;  
 }  
   
 IEnumerator IEnumerable.GetEnumerator() {  
 return ((IEnumerable)this).GetEnumerator();  
 }  
  
 IEnumerator<T> IEnumerable<T>.GetEnumerator() {  
 var current = \_head;  
 while (current != null) {  
 yield return current.Data;  
 current = current.Next;  
 }  
 }  
  
 public IEnumerable<T> BackEnumerator() {  
 var current = \_tail;  
 while (current != null) {  
 yield return current.Data;  
 current = current.Previous;  
 }  
 }  
}

**Бинарное дерево**

public class BinaryTree<T> where T : IComparable<T> **{** private BinaryTree<T> \_parent, \_left, \_right;  
 private T \_val;  
  
 public BinaryTree(T val, BinaryTree<T> parent) {  
 \_val = val;  
 \_parent = parent;  
 }  
  
 public void Add(T val) {  
 if (val.CompareTo(\_val) < 0) {  
 if (\_left == null) {  
 \_left = new BinaryTree<T>(val, this);  
 }  
 else {  
 \_left?.Add(val);  
 }  
 }  
 else {  
 if (\_right == null) {  
 \_right = new BinaryTree<T>(val, this);  
 }  
 else {  
 \_right?.Add(val);  
 }  
 }  
 }  
  
 private static BinaryTree<T> \_search(BinaryTree<T> tree, T val) {  
 if (tree == null) return null;  
 return val.CompareTo(tree.\_val) switch {  
 1 => \_search(tree.\_right, val),  
 -1 => \_search(tree.\_left, val),  
 0 => tree,  
 \_ => null  
 };  
 }  
  
 private BinaryTree<T> Search(T val) {  
 return \_search(this, val);  
 }  
  
 public bool Remove(T val) {  
 var tree = Search(val);  
 if (tree == null) {  
 return false;  
 }  
  
 BinaryTree<T> curTree;  
   
 if (tree == this) {  
 if (tree.\_right != null) {  
 curTree = tree.\_right;  
 }  
 else curTree = tree.\_left;  
  
 while (curTree.\_left != null) {  
 curTree = curTree.\_left;  
 }  
  
 var temp = curTree.\_val;  
 Remove(temp);  
 tree.\_val = temp;  
  
 return true;  
 }  
   
 if (tree.\_left == null && tree.\_right == null && tree.\_parent != null) {  
 if (tree == tree.\_parent.\_left)  
 tree.\_parent.\_left = null;  
 else {  
 tree.\_parent.\_right = null;  
 }  
  
 return true;  
 }  
   
 if (tree.\_left != null && tree.\_right == null) {  
 tree.\_left.\_parent = tree.\_parent;  
 if (tree.\_parent != null && tree == tree.\_parent.\_left) {  
 tree.\_parent.\_left = tree.\_left;  
 }  
 else if (tree.\_parent != null && tree == tree.\_parent.\_right) {  
 tree.\_parent.\_right = tree.\_left;  
 }  
  
 return true;  
 }  
   
 if (tree.\_left == null && tree.\_right != null) {  
 tree.\_right.\_parent = tree.\_parent;  
 if (tree.\_parent != null && tree == tree.\_parent.\_left) {  
 tree.\_parent.\_left = tree.\_right;  
 }  
 else if (tree.\_parent != null && tree == tree.\_parent.\_right) {  
 tree.\_parent.\_right = tree.\_right;  
 }  
  
 return true;  
 }  
  
 if (tree.\_right == null || tree.\_left == null) return false;  
 curTree = tree.\_right;  
  
 while (curTree.\_left != null) {  
 curTree = curTree.\_left;  
 }  
   
 if (curTree.\_parent == tree) {  
 curTree.\_left = tree.\_left;  
 tree.\_left.\_parent = curTree;  
 curTree.\_parent = tree.\_parent;  
 if (tree.\_parent != null && tree == tree.\_parent.\_left) {  
 tree.\_parent.\_left = curTree;  
 }  
 else if (tree.\_parent != null && tree == tree.\_parent.\_right) {  
 tree.\_parent.\_right = curTree;  
 }  
  
 return true;  
 }  
  
 if (curTree.\_right != null) {  
 curTree.\_right.\_parent = curTree.\_parent;  
 }  
  
 curTree.\_parent.\_left = curTree.\_right;  
 curTree.\_right = tree.\_right;  
 curTree.\_left = tree.\_left;  
 tree.\_left.\_parent = curTree;  
 tree.\_right.\_parent = curTree;  
 curTree.\_parent = tree.\_parent;  
 if (tree.\_parent != null && tree == tree.\_parent.\_left) {  
 tree.\_parent.\_left = curTree;  
 }  
 else if (tree.\_parent != null && tree == tree.\_parent.\_right) {  
 tree.\_parent.\_right = curTree;  
 }  
  
 return true;  
  
 }  
  
 public override string ToString() {  
 return \_val.ToString();  
 }  
}

**Алгоритм пузырьковой сортировки**

static int[] BubbleSort(int[] mas) **{** for (var i = 0;  
 i < mas.Length;  
 i++) {  
 for (var j = i + 1; j < mas.Length; j++) {  
 if (mas[i] <= mas[j]) continue;  
 var temp = mas[i];  
 mas[i] = mas[j];  
 mas[j] = temp;  
 }  
 }  
  
 return mas;  
**}**

**Вывод**

В ходе выполнения работы были получены навыки работы с языком C# посредством реализации базовых структур данных и алгоритмов.