Государственный Университет Молдовы

Факультет Математики и Информатики

Департамент Информатики

Лабораторная работа №3

“Алгоритмы и структуры данных”

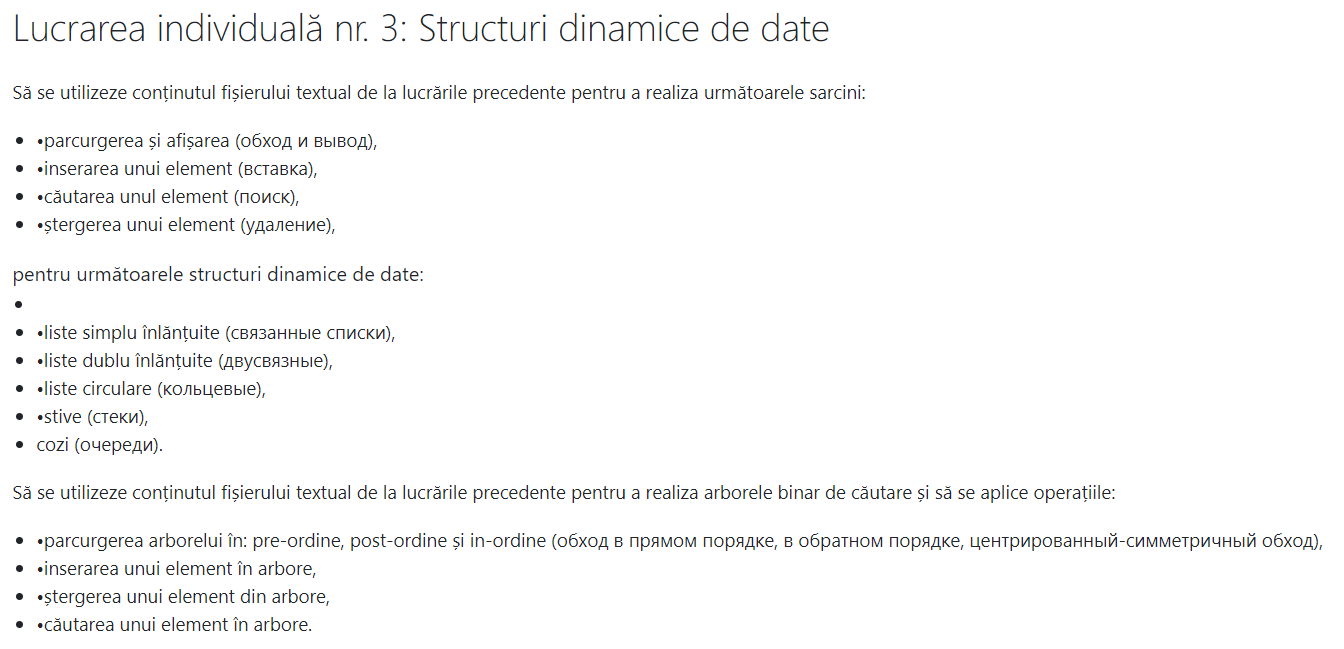
Тема:”Динамические структуры данных”

Проверил: Угнуряну Валерий

Выполнил: Чобану Артём

Группа: i1902

Кишинев 2021

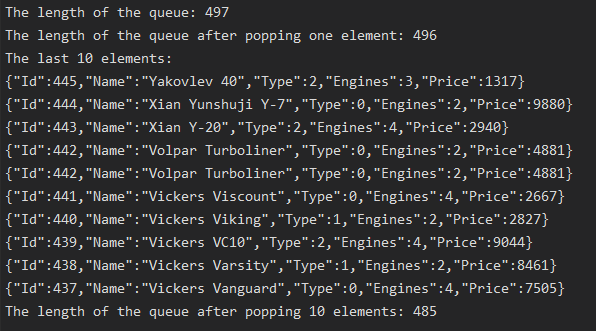
**Задание:**  


https://github.com/ArtiomCiobanu/Algorithms\_Labs

Метод Main:

private static void Main()  
{  
 StackTest();  
 QueueTest();  
 LinkedListTest();  
 DoublyLinkedListTest();  
 CircularListTest();  
 BinaryTreeTest();  
}

**Стек:**



private static void StackTest()  
{  
 var aircraftsStack = new Stack<Aircraft>();  
 foreach (var aircraft in Aircrafts)  
 {  
 aircraftsStack.Push(aircraft);  
 }  
  
 Console.WriteLine($"The length of the queue: {aircraftsStack.Length}");  
  
 var someAircraft = aircraftsStack.Pop();  
  
 Console.WriteLine($"The length of the queue after popping one element: {aircraftsStack.Length}");  
  
 var anotherAircraft = aircraftsStack.Pop();  
  
 Console.WriteLine("The last 10 elements:");  
  
 for (int i = 0; i < 10; i++)  
 {  
 Console.WriteLine(aircraftsStack.Pop());  
 }  
  
 Console.WriteLine($"The length of the queue after popping 10 elements: {aircraftsStack.Length}");  
}

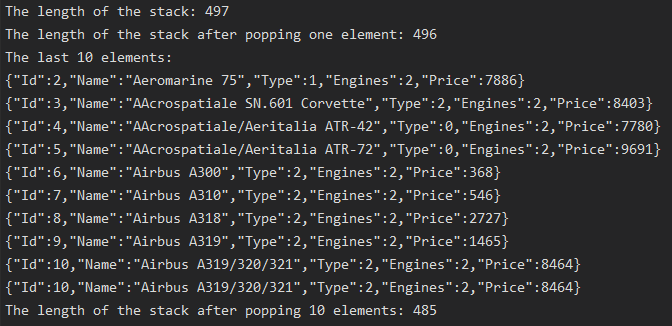
Класс Stack<T>:

public class Stack<T>  
{  
 private const int DefaultCapacity = 10;  
 private T[] \_array;  
 private int \_capacity;  
 public int Length { get; private set; }  
  
 public Stack()  
 {  
 \_array = new T[DefaultCapacity];  
 \_capacity = DefaultCapacity;  
 }

public void Push(T value)  
 {  
 if (Length < \_capacity)  
 {  
 \_array[Length] = value;  
 Length++;  
 }  
 else  
 {  
 var newArray = new T[Length + 1];  
 for (int i = 0; i < Length; i++)  
 {  
 newArray[i] = \_array[i];  
 }  
  
 \_array = newArray;  
  
 \_capacity = Length;  
 }  
 }  
 public T Pop()  
 {  
 if (Length > 0)  
 {  
 Length--;  
  
 return \_array[Length];  
 }

throw new IndexOutOfRangeException();  
 }  
 public void Clear()  
 {  
 Array.Clear(\_array, 0, Length);  
 }  
   
 public T this[int index] => \_array[index];  
}

**Очередь:**



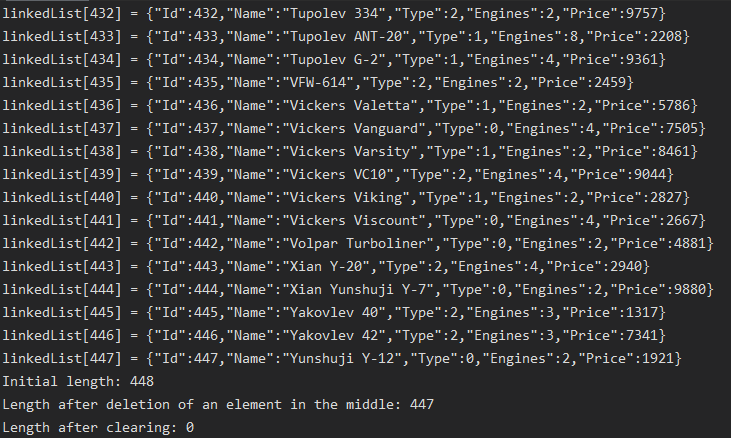
private static void QueueTest()  
{  
 var aircraftsQueue = new Queue<Aircraft>();  
 foreach (var aircraft in Aircrafts)  
 {  
 aircraftsQueue.Push(aircraft);  
 }  
  
 Console.WriteLine($"The length of the stack: {aircraftsQueue.Length}");  
  
 var someAircraft = aircraftsQueue.Pop();  
  
 Console.WriteLine($"The length of the stack after popping one element: {aircraftsQueue.Length}");  
  
 var anotherAircraft = aircraftsQueue.Pop();  
  
 Console.WriteLine("The last 10 elements:");  
  
 for (int i = 0; i < 10; i++)  
 {  
 Console.WriteLine(aircraftsQueue.Pop());  
 }  
  
 Console.WriteLine($"The length of the stack after popping 10 elements: {aircraftsQueue.Length}");  
}

Класс Queue<T>:

public class Queue<T>  
{  
 private const int DefaultCapacity = 10;  
 private T[] \_array;  
 private int \_capacity;  
 public int Length { get; private set; }  
 public Queue()  
 {  
 \_array = new T[DefaultCapacity];  
 \_capacity = DefaultCapacity;  
 }  
 public void Push(T value)  
 {  
 if (Length < \_capacity)  
 {  
 \_array[Length] = value;  
 Length++;  
 }  
 else  
 {  
 var newArray = new T[Length + 1];  
 for (int i = 0; i < Length; i++)  
 {  
 newArray[i] = \_array[i];  
 }  
  
 \_array = newArray;  
  
 \_capacity = Length;  
 }  
 }  
 public T Pop()  
 {  
 if (Length > 0)  
 {  
 Length--;  
  
 var result = \_array[0];  
  
 for (int i = 0; i < Length; i++)  
 {  
 \_array[i] = \_array[i + 1];  
 }  
  
 return result;  
 }  
 throw new IndexOutOfRangeException();  
 }

public void Clear()  
 {  
 Array.Clear(\_array, 0, Length);  
 }  
  
 public T this[int index] => \_array[index];  
}

**Связный список:**



private static void LinkedListTest()  
{  
 var linkedList = new LinkedList<Aircraft>();  
 foreach (var aircraft in Aircrafts)  
 {  
 linkedList.Add(aircraft);  
 }  
 for (int i = 0; i < linkedList.Length; i++)  
 {  
 linkedList[i] = linkedList[i];  
 Console.WriteLine($"linkedList[{i}] = {linkedList[i]}");  
 }  
 Console.WriteLine($"Initial length: {linkedList.Length}");  
 linkedList.RemoveAt(100);  
 Console.WriteLine($"Length after deletion of an element in the middle: {linkedList.Length}");  
 linkedList.Clear();  
 Console.WriteLine($"Length after clearing: {linkedList.Length}");  
}

Класс LinkedListNode<T>:

public class LinkedListNode<T>  
{  
 public T Value { get; set; }  
  
 public LinkedListNode<T> Next { get; set; }  
  
 public LinkedListNode(T value)  
 {  
 Value = value;  
 }  
}

Класс LinkedList<T>:

public class LinkedList<T>  
{  
 public int Length { get; private set; }  
  
 private LinkedListNode<T> Root { get; set; }

private LinkedListNode<T> Tail { get; set; }

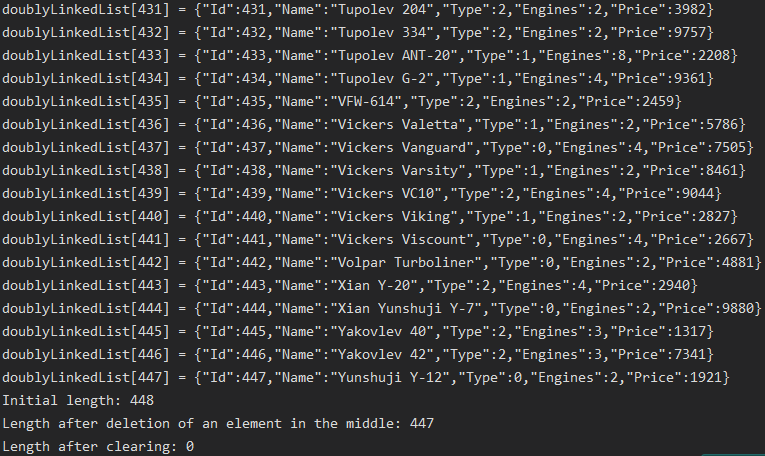
public LinkedList()  
 {  
 Length = 0;  
 }

public T this[int index]  
 {  
 get => FindNode(index).Value;  
 set  
 {  
 var foundNode = FindNode(index);  
 foundNode.Value = value;  
 }  
 }

private LinkedListNode<T> FindNode(int index)  
 {  
 if (index < 0 || index > Length)  
 {  
 throw new IndexOutOfRangeException();  
 }  
 var currentNode = Root;  
 for (int i = 1; i <= index; i++)  
 {  
 currentNode = currentNode.Next;  
 }  
  
 return currentNode;  
 }  
 public void Add(T value)  
 {  
 var newNode = new LinkedListNode<T>(value);  
 if (Length == 0)  
 {  
 Root = newNode;  
 Tail = newNode;  
 }  
 else  
 {  
 Tail.Next = newNode;  
 Tail = newNode;  
 }  
 Length++;  
 }

public void RemoveAt(int index)  
 {  
 var foundNode = FindNode(index);  
 if (index > 0)  
 {  
 var previous = FindNode(index - 1);  
  
 foundNode = foundNode.Next;  
 previous.Next = foundNode;  
 }  
 else  
 {  
 Root = foundNode;  
 }  
  
 Length--;  
 }  
  
 public void Clear()  
 {  
 Length = 0;  
 Root = null;  
 Tail = null;  
 }  
}

**Двусвязный список:**



private static void DoublyLinkedListTest()  
{  
 var doublyLinkedList = new DoublyLinkedList<Aircraft>();  
 foreach (var aircraft in Aircrafts)  
 {  
 doublyLinkedList.Add(aircraft);  
 }  
  
 for (int i = 0; i < doublyLinkedList.Length; i++)  
 {  
 doublyLinkedList[i] = doublyLinkedList[i];  
 Console.WriteLine($"doublyLinkedList[{i}] = {doublyLinkedList[i]}");  
 }  
  
 Console.WriteLine($"Initial length: {doublyLinkedList.Length}");  
 doublyLinkedList.RemoveAt(100);  
 Console.WriteLine($"Length after deletion of an element in the middle: {doublyLinkedList.Length}");  
  
 doublyLinkedList.Clear();  
 Console.WriteLine($"Length after clearing: {doublyLinkedList.Length}");  
}

Класс DoublyLinkedList<T>:

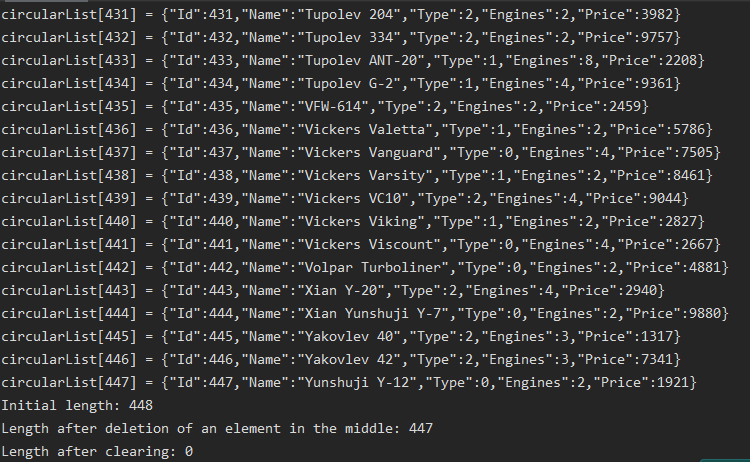
public class DoublyLinkedList<T>  
{  
 public int Length { get; private set; }  
  
 private DoublyLinkedListNode<T> Root { get; set; }  
 private DoublyLinkedListNode<T> Tail { get; set; }

public DoublyLinkedList()  
 {  
 Length = 0;  
 }  
  
 public T this[int index]  
 {  
 get => FindNode(index).Value;  
 set  
 {  
 var foundNode = FindNode(index);  
 foundNode.Value = value;  
 }  
 }  
  
 private DoublyLinkedListNode<T> FindNode(int index)  
 {  
 if (index < 0 || index > Length)  
 {  
 throw new IndexOutOfRangeException();  
 }  
  
 var currentNode = Root;  
 for (int i = 1; i <= index; i++)  
 {  
 currentNode = currentNode.Next;  
 }  
  
 return currentNode;  
 }  
  
 public void Add(T value)  
 {  
 var newNode = new DoublyLinkedListNode<T>(value);  
  
 if (Length == 0)  
 {  
 Root = newNode;  
 Tail = newNode;  
 }  
 else  
 {  
 Tail.Next = newNode;  
 newNode.Previous = Tail;  
 Tail = newNode;  
 }  
  
 Length++;  
 }  
  
 public void RemoveAt(int index)  
 {  
 var foundNode = FindNode(index);  
 if (index > 0)  
 {  
 var previous = FindNode(index - 1);  
  
 foundNode.Previous = previous;  
 foundNode = foundNode.Next;  
 previous.Next = foundNode;  
 }  
 else  
 {  
 foundNode.Next = Root.Next;  
 foundNode.Next.Previous = foundNode;  
 Root = foundNode;  
 }  
  
 Length--;  
 }  
  
 public void Clear()  
 {  
 Length = 0;  
 Root = null;  
 Tail = null;  
 }  
}

Класс DoublyLinkedListNode<T>:

public class DoublyLinkedListNode<T>  
{  
 public T Value { get; set; }  
  
 public DoublyLinkedListNode<T> Previous { get; set; }  
 public DoublyLinkedListNode<T> Next { get; set; }  
  
 public DoublyLinkedListNode(T value)  
 {  
 Value = value;  
 }  
}

**Кольцевой список:**



private static void CircularListTest()  
{  
 var circularList = new CircularList<Aircraft>();  
 foreach (var aircraft in Aircrafts)  
 {  
 circularList.Add(aircraft);  
 }  
  
 for (int i = 0; i < circularList.Length; i++)  
 {  
 circularList[i] = circularList[i];  
 Console.WriteLine($"circularList[{i}] = {circularList[i]}");  
 }  
  
 Console.WriteLine($"Initial length: {circularList.Length}");  
 circularList.RemoveAt(100);  
 Console.WriteLine($"Length after deletion of an element in the middle: {circularList.Length}");  
  
 circularList.Clear();  
 Console.WriteLine($"Length after clearing: {circularList.Length}");  
}

Класс CircularListNode<T>:

public class CircularListNode<T>  
{  
 public T Value { get; set; }  
 public CircularListNode<T> Previous { get; set; }  
 public CircularListNode<T> Next { get; set; }  
 public CircularListNode(T value)  
 {  
 Value = value;  
 }  
}

Класс CircularList<T>:

public class CircularList<T>  
{  
 public int Length { get; private set; }  
  
 private CircularListNode<T> Root { get; set; }

private CircularListNode<T> Tail { get; set; }  
  
 public CircularList()  
 {  
 Length = 0;  
 }  
  
 public T this[int index]  
 {  
 get => FindNode(index).Value;  
 set  
 {  
 var foundNode = FindNode(index);  
 foundNode.Value = value;  
 }  
 }  
  
 private CircularListNode<T> FindNode(int index)  
 {  
 if (index < 0 || index > Length)  
 {  
 throw new IndexOutOfRangeException();  
 }  
  
 var currentNode = Root;  
 for (int i = 1; i <= index; i++)  
 {  
 currentNode = currentNode.Next;  
 }  
  
 return currentNode;  
 }  
  
 public void Add(T value)  
 {  
 var newNode = new CircularListNode<T>(value);  
  
 if (Length == 0)  
 {  
 Root = newNode;  
 Tail = newNode;  
  
 Tail.Previous = Root;  
 Root.Next = Tail;  
 }  
 else  
 {  
 Tail.Next = newNode;  
 newNode.Previous = Tail;  
 Tail = newNode;  
 }  
  
 Tail.Next = Root;  
 Root.Previous = Tail;  
  
 Length++;  
 }  
  
 public void RemoveAt(int index)  
 {  
 var foundNode = FindNode(index);  
 if (index > 0)  
 {  
 var previous = FindNode(index - 1);  
  
 foundNode.Previous = previous;  
 foundNode = foundNode.Next;  
 previous.Next = foundNode;  
 }  
 else  
 {  
 foundNode.Next = Root.Next;  
 foundNode.Next.Previous = foundNode;  
 Root = foundNode;  
  
 Root.Previous = Tail;  
 Tail.Next = Root;  
 }  
  
 Length--;  
 }

public void Clear()  
 {  
 Length = 0;  
 Root = null;  
 Tail = null;  
 }  
}

**Бинарное дерево с имплементацией четырёх способов обхода и поиска:**

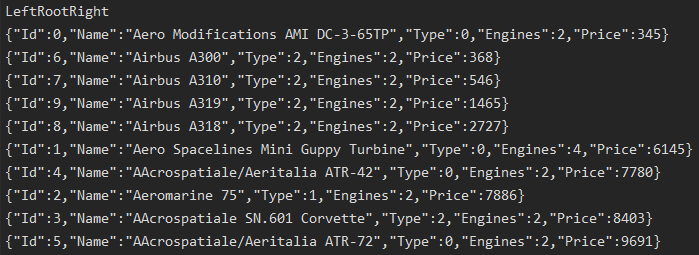
private static void BinaryTreeTest()  
{  
 var binaryTree = new BinaryTree.BinaryTreeWithAircrafts();  
  
 foreach (var aircraft in Aircrafts.Take(10))  
 {  
 binaryTree.Insert(aircraft);  
 }  
  
 binaryTree.LeftRootRight();  
 binaryTree.RightRootLeft();  
 binaryTree.RootLeftRight();  
 binaryTree.LeftRightRoot();  
   
 Console.WriteLine();  
 Console.WriteLine("Searching for the aircraft with the price 1465:");  
 Console.WriteLine(binaryTree.Find(1465));  
}

Поиск:



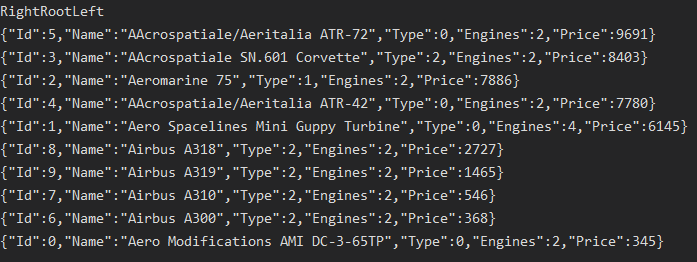
public Aircraft Find(int price)  
{  
 var currentNode = RootNode;  
  
 while (currentNode.Value.Price != price)  
 {  
 if (price > currentNode.Value.Price)  
 {  
 if (currentNode.Right != null)  
 {  
 currentNode = currentNode.Right;  
 }  
 }  
 else if (price < currentNode.Value.Price)  
 {  
 if (currentNode.Left != null)  
 {  
 currentNode = currentNode.Left;  
 }  
 }  
 }  
  
 return currentNode.Value;  
}

Левый-Корень-Правый



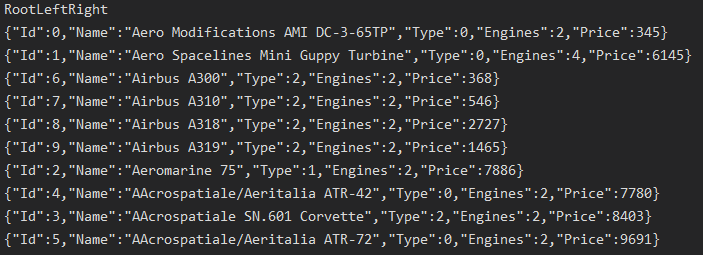
public void LeftRootRight()  
{  
 Console.WriteLine();  
 Console.WriteLine("LeftRootRight");  
  
 LeftRootRight(RootNode);  
}  
  
private static void LeftRootRight(BinaryTreeNode<Aircraft> node)  
{  
 if (node.Left != null)  
 {  
 LeftRootRight(node.Left);  
 }  
  
 Console.WriteLine($"{node.Value} ");  
  
 if (node.Right != null)  
 {  
 LeftRootRight(node.Right);  
 }  
}

Правый-Корень-Левый



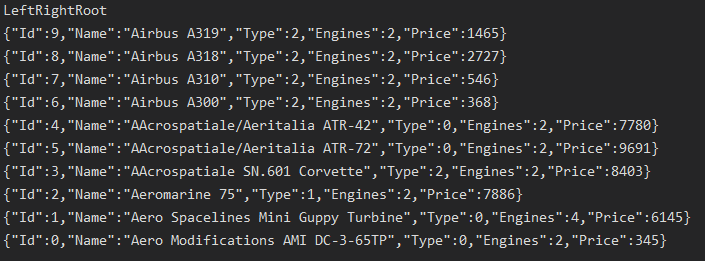
public void RightRootLeft()  
{  
 Console.WriteLine();  
 Console.WriteLine("RightRootLeft");  
  
 RightRootLeft(RootNode);  
}  
  
private static void RightRootLeft(BinaryTreeNode<Aircraft> node)  
{  
 if (node.Right != null)  
 {  
 RightRootLeft(node.Right);  
 }  
  
 Console.WriteLine($"{node.Value} ");  
  
 if (node.Left != null)  
 {  
 RightRootLeft(node.Left);  
 }  
}

Корень-Левый-Правый



public void RootLeftRight()  
{  
 Console.WriteLine();  
 Console.WriteLine("RootLeftRight");  
  
 RootLeftRight(RootNode);  
}  
  
private static void RootLeftRight(BinaryTreeNode<Aircraft> node)  
{  
 Console.WriteLine($"{node.Value} ");  
  
 if (node.Left != null)  
 {  
 RootLeftRight(node.Left);  
 }  
  
 if (node.Right != null)  
 {  
 RootLeftRight(node.Right);  
 }  
}

Корень-Правый-Левый



public void LeftRightRoot()  
{  
 Console.WriteLine();  
 Console.WriteLine("LeftRightRoot");  
  
 LeftRightRoot(RootNode);  
}  
  
private static void LeftRightRoot(BinaryTreeNode<Aircraft> node)  
{  
 if (node.Left != null)  
 {  
 LeftRightRoot(node.Left);  
 }  
  
 if (node.Right != null)  
 {  
 LeftRightRoot(node.Right);  
 }  
  
 Console.WriteLine($"{node.Value} ");  
}

Класс BinaryTreeNode<T>:

public class BinaryTreeNode<T>  
{  
 public T Value { get; set; }  
   
 public BinaryTreeNode<T> Left { get; set; }  
 public BinaryTreeNode<T> Right { get; set; }  
  
 public BinaryTreeNode(T value)  
 {  
 Value = value;  
 }  
}

Класс BinaryTree:

public class BinaryTreeWithAircrafts  
{  
 public BinaryTreeNode<Aircraft> RootNode { get; set; }  
  
 public void Insert(Aircraft value)  
 {  
 InsertNode(new BinaryTreeNode<Aircraft>(value));  
 }  
  
 public void InsertNode(BinaryTreeNode<Aircraft> node)  
 {  
 if (RootNode == null)  
 {  
 RootNode = node;  
 return;  
 }  
  
 var currentNode = RootNode;  
  
 while (true)  
 {  
 if (currentNode.Value == node.Value)  
 {  
 return;  
 }  
  
 if (node.Value.Price > currentNode.Value.Price)  
 {  
 if (currentNode.Right == null)  
 {  
 currentNode.Right = node;  
 return;  
 }  
  
 currentNode = currentNode.Right;  
 }  
 else if (node.Value.Price < currentNode.Value.Price)  
 {  
 if (currentNode.Left == null)  
 {  
 currentNode.Left = node;  
 return;  
 }  
  
 currentNode = currentNode.Left;  
 }  
 }  
 }  
  
 public void LeftRootRight()  
 {  
 Console.WriteLine();  
 Console.WriteLine("LeftRootRight");  
  
 LeftRootRight(RootNode);  
 }  
  
 private static void LeftRootRight(BinaryTreeNode<Aircraft> node)  
 {  
 if (node.Left != null)  
 {  
 LeftRootRight(node.Left);  
 }  
  
 Console.WriteLine($"{node.Value} ");  
  
 if (node.Right != null)  
 {  
 LeftRootRight(node.Right);  
 }  
 }  
  
 public void RightRootLeft()  
 {  
 Console.WriteLine();  
 Console.WriteLine("RightRootLeft");  
  
 RightRootLeft(RootNode);  
 }  
  
 private static void RightRootLeft(BinaryTreeNode<Aircraft> node)  
 {  
 if (node.Right != null)  
 {  
 RightRootLeft(node.Right);  
 }  
  
 Console.WriteLine($"{node.Value} ");  
  
 if (node.Left != null)  
 {  
 RightRootLeft(node.Left);  
 }  
 }  
  
 public void RootLeftRight()  
 {  
 Console.WriteLine();  
 Console.WriteLine("RootLeftRight");  
  
 RootLeftRight(RootNode);  
 }  
  
 private static void RootLeftRight(BinaryTreeNode<Aircraft> node)  
 {  
 Console.WriteLine($"{node.Value} ");  
  
 if (node.Left != null)  
 {  
 RootLeftRight(node.Left);  
 }  
  
 if (node.Right != null)  
 {  
 RootLeftRight(node.Right);  
 }  
 }  
  
  
 public void LeftRightRoot()  
 {  
 Console.WriteLine();  
 Console.WriteLine("LeftRightRoot");  
  
 LeftRightRoot(RootNode);  
 }  
  
 private static void LeftRightRoot(BinaryTreeNode<Aircraft> node)  
 {  
 if (node.Left != null)  
 {  
 LeftRightRoot(node.Left);  
 }  
  
 if (node.Right != null)  
 {  
 LeftRightRoot(node.Right);  
 }  
  
 Console.WriteLine($"{node.Value} ");  
 }  
   
 public Aircraft Find(int price)  
 {  
 var currentNode = RootNode;  
  
 while (currentNode.Value.Price != price)  
 {  
 if (price > currentNode.Value.Price)  
 {  
 if (currentNode.Right != null)  
 {  
 currentNode = currentNode.Right;  
 }  
 }  
 else if (price < currentNode.Value.Price)  
 {  
 if (currentNode.Left != null)  
 {  
 currentNode = currentNode.Left;  
 }  
 }  
 }  
  
 return currentNode.Value;  
 }  
}