**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#»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6305 |  | Новиков Б.М. |
| Преподаватель |  | Пешехонов К.А. |

Санкт-Петербург

2020

**Цель**: ознакомиться с базовыми средствами языка C#.

**Задание**:

1. Реализовать связный список (без использования стандартных коллекций/LINQ, кроме IEnumerable) со следующими операциями: создание, удаление, добавление произвольных элементов, реверс списка.
2. Реализовать бинарное дерево (без использования стандартных деревьев) со следующими операциями: поиск элемента, удаление элемента.
3. Реализовать сортировку вставками (без использования метода OrderBy()).

**Связный список**

- базовая динамическая структура данных в информатике, состоящая из узлов, каждый из которых содержит как собственно данные, так и одну или две ссылки («связки») на следующий и/или предыдущий узел списка.

**Реализация.** Листинг программы см. Приложение.

**Результаты работы программы.** Пошагово выполняется исполнение операций над связным списком и вывод элементов списка на экран. Первой исполняется добавление в конец списка значений 1, 2, 3. Далее демонстрируется удаление элемента со значением 2. После в начало списка добавляется элемент со значением 0. Последней исполняется операция реверс.

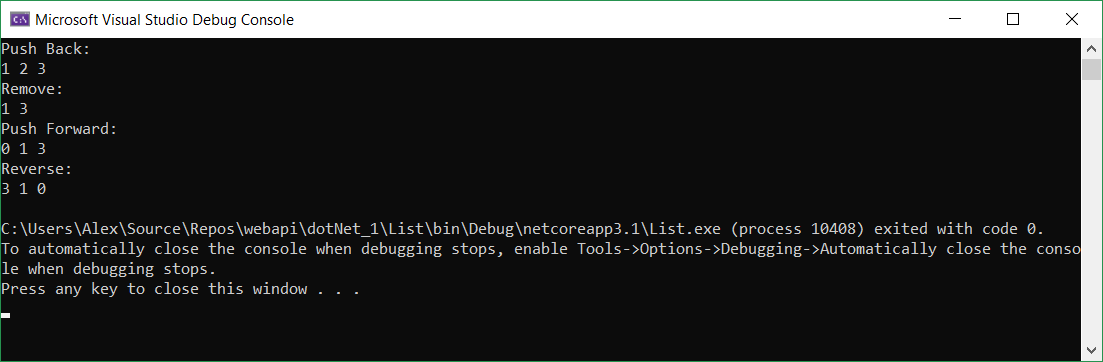


Рисунок 1. Результат работы программы «Связный список»

**Бинарное дерево поиска**

- структура данных для работы с упорядоченными множествами.

- обладает следующим свойством: если *x* — узел бинарного дерева с ключом *k*, то все узлы в левом поддереве должны иметь ключи, меньшие *k*, а в правом поддереве большие *k*.

**Реализация**. Листинг программы см. Приложение.

**Результаты работы программы.** Пошагово выполняется исполнение операций над бинарным деревом поиска. Первой исполняется добавление в дерево значений 1, 3, -1, 4, 7. Далее происходит прямой обход дерева с выводом значений на экран. Потом демонстрируется удаление элемента со значением 3. Затем проводится поиск значения 3 и 7 в дереве с выводом результата поиска на экран. В конце повторяем обход с выводом содержимого дерева на экран.

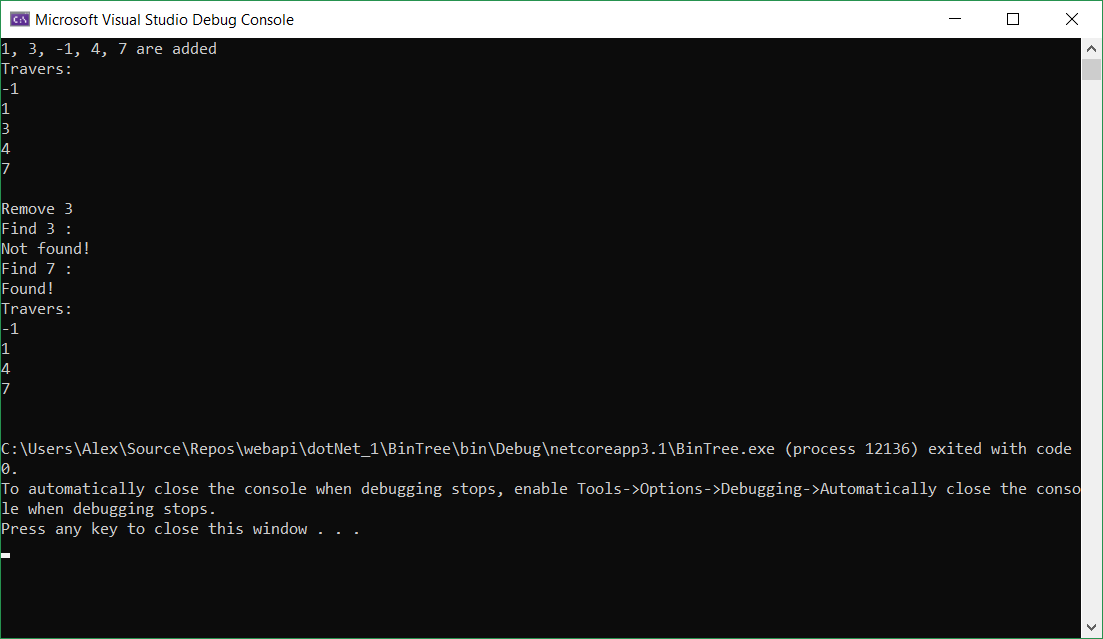


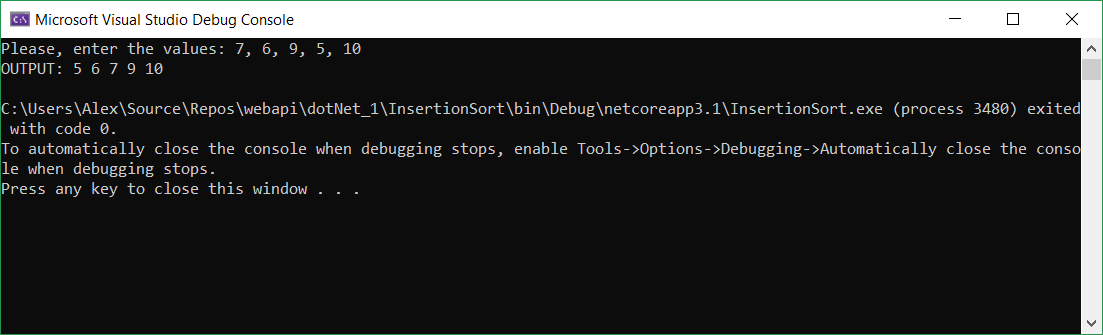
Рисунок 2. Результат работы программы «Двоичное дерево поиска»

**Сортировка вставками**

- алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

**Реализация**. Листинг программы см. Приложение.

**Результаты работы программы.** Пользователь вводит с клавиатуры целые числа, программа формирует из них массив, сортирует его и выводит на экран.



Рисуок 3. Результат работы программы «Сортировка вставками»

**Приложение**

1. Листинг программы «Связный список»:

using System;

using System.Collections;

using System.Collections.Generic;

namespace List

{

public class Node<T>

{

public T Value { get; }

public Node<T> Next { get; set; }

public Node(T value) { Value = value; }

}

public class List<T> : IEnumerable<T>

{

private Node<T> \_head;

private Node<T> \_tail;

private int \_count;

public bool IsEmpty => \_count == 0;

public int Count => \_count;

public void Push\_Back(T value)

{

if (value == null)

throw new ArgumentNullException(nameof(value));

var node = new Node<T>(value);

if (\_head == null)

\_head = node;

else

\_tail.Next = node;

\_tail = node;

\_count++;

}

public void Push\_Forward(T value)

{

var node = new Node<T>(value);

node.Next = \_head;

\_head = node;

if (IsEmpty)

\_tail = \_head;

\_count++;

}

public bool Remove(T value)

{

Node<T> cur = \_head;

Node<T> prev = null;

while (cur != null)

{

if (value != null && cur.Value.Equals(value))

{

if (prev != null)

{

prev.Next = cur.Next;

if (cur.Next == null)

\_tail = prev;

}

else

{

\_head = \_head.Next;

if (\_head == null)

\_tail = null;

}

\_count--;

return true;

}

prev = cur;

cur = cur.Next;

}

return false;

}

public void Reverse()

{

Node<T> cur = \_head;

Node<T> prev = null;

Node<T> next;

while (cur != null)

{

next = cur.Next;

if (prev != null)

{

cur.Next = prev;

}

else

{

cur.Next = \_tail.Next;

\_tail = cur;

}

prev = cur;

cur = next;

}

\_head = prev;

}

public void Clear()

{

\_head = null;

\_tail = null;

\_count = 0;

}

public IEnumerator<T> GetEnumerator()

{

var cur = \_head;

while (cur != null)

{

yield return cur.Value;

cur = cur.Next;

}

}

IEnumerator IEnumerable.GetEnumerator()

{

return (this as IEnumerable).GetEnumerator();

}

}

public static class Program

{

public static void Main(string[] args)

{

var list = new List<int>();

Console.WriteLine("Push Back:");

list.Push\_Back(1);

list.Push\_Back(2);

list.Push\_Back(3);

foreach (var item in list)

Console.Write(item + " ");

Console.WriteLine();

Console.WriteLine("Remove:");

list.Remove(2);

foreach (var item in list)

Console.Write(item + " ");

Console.WriteLine();

Console.WriteLine("Push Forward:");

list.Push\_Forward(0);

foreach (var item in list)

Console.Write(item + " ");

Console.WriteLine();

Console.WriteLine("Reverse:");

list.Reverse();

foreach (var item in list)

Console.Write(item + " ");

Console.WriteLine();

list.Clear();

}

}

}

1. Листинг программы «Бинарное дерево поиска»

using System;

namespace BinaryTree

{

public class Node

{

public int Value;

public Node Left;

public Node Right;

}

public class BinaryTree

{

private Node \_root;

public BinaryTree()

{

\_root = null;

}

public void InsertNode(int key)

{

if (\_root != null)

InsertNode(key, \_root);

else

\_root = new Node

{

Value = key,

Left = null,

Right = null

};

}

private void InsertNode(int key, Node leaf)

{

if (leaf == null)

throw new ArgumentNullException(nameof(leaf));

while (true)

{

if (key < leaf.Value)

{

if (leaf.Left != null)

{

leaf = leaf.Left;

continue;

}

leaf.Left = new Node

{

Value = key,

Left = null,

Right = null

};

}

else if (key >= leaf.Value)

{

if (leaf.Right != null)

{

leaf = leaf.Right;

continue;

}

leaf.Right = new Node

{

Value = key,

Right = null,

Left = null

};

}

break;

}

}

public Node SearchNode(int key)

{

return SearchNode(key, \_root);

}

private static Node SearchNode(int key, Node leaf)

{

while (true)

{

if (leaf != null)

{

if (key == leaf.Value)

return leaf;

leaf = key < leaf.Value ? leaf.Left : leaf.Right;

}

else

{

return null;

}

}

}

public void RemoveNode(int key)

{

RemoveNode(\_root, SearchNode(key, \_root));

}

private static Node RemoveNode(Node root, Node removableNode)

{

if (root == null)

return null;

if (removableNode.Value < root.Value)

root.Left = RemoveNode(root.Left, removableNode);

if (removableNode.Value > root.Value)

root.Right = RemoveNode(root.Right, removableNode);

if (removableNode.Value != root.Value)

return root;

switch (root.Left)

{

case null when root.Right == null:

{

return null;

}

case null:

{

root = root.Right;

break;

}

default:

{

if (root.Right == null)

{

root = root.Left;

}

else

{

var minimalNode = GetMinNode(root.Right);

root.Value = minimalNode.Value;

root.Right = RemoveNode(root.Right, minimalNode);

}

break;

}

}

return root;

}

private static Node GetMinNode(Node currentNode)

{

while (currentNode?.Left != null)

currentNode = currentNode.Left;

return currentNode;

}

public void InOrderTravers()

{

InOrderTravers(\_root);

Console.WriteLine("");

}

private static void InOrderTravers(Node leaf)

{

while (true)

{

if (leaf != null)

{

InOrderTravers(leaf.Left);

Console.WriteLine("{0}", leaf.Value);

leaf = leaf.Right;

continue;

}

break;

}

}

}

public class Program

{

public static void Main(string[] args)

{

var tree = new BinaryTree();

Console.WriteLine("1, 3, -1, 4, 7 are added");

tree.InsertNode(Convert.ToInt32(1));

tree.InsertNode(Convert.ToInt32(3));

tree.InsertNode(Convert.ToInt32(-1));

tree.InsertNode(Convert.ToInt32(4));

tree.InsertNode(Convert.ToInt32(7));

Console.WriteLine("Travers: ");

tree.InOrderTravers();

Console.WriteLine("Remove 3");

tree.RemoveNode(Convert.ToInt32(3));

Console.WriteLine("Find 3 :");

var temp = tree.SearchNode(Convert.ToInt32(3));

Console.WriteLine(temp != null ? "Found!" : "Not found!");

Console.WriteLine("Find 7 :");

temp = tree.SearchNode(Convert.ToInt32(7));

Console.WriteLine(temp != null ? "Found!" : "Not found!");

Console.WriteLine("Travers: ");

tree.InOrderTravers();

}

}

}

1. Листинг программы «Сортировка вставками»

using System;

namespace InsertionSort

{

public static class Program

{

private static void Swap(ref int A, ref int B)

{

var temp = A;

A = B;

B = temp;

}

private static int[] InsertionSort(int[] array)

{

for (var index = 1; index < array.Length; index++)

{

var key = array[index];

var indexOfSorted = index;

while (indexOfSorted > 0 && array[indexOfSorted - 1] > key)

{

Swap(ref array[indexOfSorted - 1], ref array[indexOfSorted]);

indexOfSorted--;

}

array[indexOfSorted] = key;

}

return array;

}

public static void Main(string[] args)

{

Console.Write("Please, enter the values: ");

var values = Console.ReadLine()?.Split(new[] { " ", "," },

StringSplitOptions.RemoveEmptyEntries);

var array = new int[values.Length];

for (var index = 0; index < values.Length; index++)

array[index] = Convert.ToInt32(values[index]);

Console.WriteLine("OUTPUT: {0}", string.Join(" ", InsertionSort(array)));

}

}

}