Изображение выглядит как цепь

Автоматически созданное описание

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| УГНС | | 09.00.00 | Информатика и вычислительная техника | | |
| Направление подготовки | | 09.03.01 | Информатика и вычислительная техника | | |
| Направленность (профиль) | |  | Автоматизированные системы обработки информации и управления | | |
| Форма обучения | |  | очная | | |
|  | |  |  | | |
| Факультет | |  | Информационных технологий и управления | | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Разработка программных систем | | |
| Курс | II | | | Группа | 403 |

Отчёт по лабораторной работе № 2

Вариант № 9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 403 |  |  |  | Шишко Даниил Юрьевич |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Федин Алексей Константинович |

**Оглавление**

[Постановка задачи 3](#_Toc95648822)

[Исходные данные 3](#_Toc95648823)

[Особые ситуации 3](#_Toc95648824)

[Математические методы и алгоритмы решения задач 3](#_Toc95648825)

[Форматы представления данных 4](#_Toc95648826)

[Структура программы 5](#_Toc95648827)

[Блок-схемы алгоритмов программы 6](#_Toc95648828)

[Описание хода выполнения работы 8](#_Toc95648829)

[Результат работы программы 9](#_Toc95648830)

[Исходный текст программы 11](#_Toc95648831)

[Документирование и комментирование исходного текста 24](#_Toc95648832)

# Постановка задачи

Создать интерфейс ICipher, который определяет методы поддержки шифрования строк. В интерфейсе объявляются два метода Encode() и Decode(), которые используются для шифрования и дешифрования строк, соответственно. Реализовать 2 класса реализующих данный интерфейс. Один из алгоритмов в соответствие с вариантом, второй по выбору.

# Исходные данные

В качестве исходных данных программа использует, вводимое пользователем строку, и ключ, которые шифруются и дешифруются.

# Особые ситуации

Необходимо рассмотреть следующие особые ситуации.

* Пустая строка с ключом или сообщением.
* Определитель матрицы ключа равен нулю.
* НОД определителя матрицы ключа и длинны алфавита равны определителю ма

# Математические методы и алгоритмы решения задач

Шифр Хилла является полиграммным шифром, который может использовать большие блоки с помощью линейной алгебры. Каждой букве алфавита сопоставляется число по модулю 26. Для латинского алфавита часто используется простейшая схема: A = 0, B = 1, …, Z = 25, но это не является существенным свойством шифра. Блок из n букв рассматривается как n-мерный вектор и умножается по модулю 26 на матрицу размера n × n. Если в качестве основания модуля используется число больше чем 26, то можно использовать другую числовую схему для сопоставления буквам чисел и добавить пробелы и знаки пунктуации. Элементы матрицы являются ключом. Матрица должна быть обратима, чтобы была возможность расшифрования.

Для n = 3 система может быть описана так:

Или C =KP (mod 26),

где P и C – векторы-столбцы высоты 3, представляющие открытый и зашифрованный текста соответственно, K – матрица 3x3, представляющая ключ шифрования. Операции выполняются по модулю 26.

Для того, чтобы расшифровать сообщение, требуется получить обратную матрицу ключа . Существуют стандартные методы вычисления обратных матриц (см. способы нахождения обратной матрицы), но не все матрицы имеют обратную (см. обратная матрица). Матрица будет иметь обратную в том и только в том случае, когда её детерминант не равен нулю и не имеет общих делителей с основанием модуля. Если детерминант матрицы равен нулю или имеет общие делители с основанием модуля, то такая матрица не может использоваться в шифре Хилла, и должна быть выбрана другая матрица (в противном случае шифротекст будет невозможно расшифровать). Тем не менее, матрицы, которые удовлетворяют вышеприведенным условиям, существуют в изобилии.

В общем случае, алгоритм шифрования может быть выражен в следующем виде:

Шифрование: C = E (K, P) = KP (mod 26).

Расшифрование: P = D (K, C) = C (mod 26) = KP (mod 26) = P.

Алгоритм Цезаря:

Если сопоставить каждому символу алфавита его порядковый номер (нумеруя с 0), то шифрование и дешифрование можно выразить формулами модульной арифметики:

где x – символ открытого текста, y = символ шифрованного текста, n – мощность алфавита, а k – ключ.

# Форматы представления данных

Программа использует следующие переменные:

Таблица 1 – Переменные, используемые в программе

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| fullAlfabet | string | Алфавит |
| letterQty | int | Длинна алфавита |
| retVal | stirng | Возвращаемое значение |
| plainMessage | string | Сообщение для шифровки |
| key | int | Ключ для шифровки |
| Продолжение таблицы 1 – Переменные, используемые в программе | | |
| encryptedMessage | string | Зашифрованное сообщение |
| message | string | Передаваемое сообщение |
| path | string | Путь к файлу |
| stringFromFile | string | Строка из файла |
| someInput | IInputData | Создатель для типа ввода |
| size | int | Размер |
| isSquare | bool | Квадрат числа |
| matrixOfKey | int[, ] | Матрица ключа |
| asciCode | int | Число в АСКИ2 |
| lenghtOfMessage | int | Длинна сообщения |
| result | int[, ] | Результат вычислений |
| allIntMessage | int[, ] | Матричное сообщение |
| decription | int[, ] | Матрица |
| badData | bool | Неподходящие данные |
| messageFromConsole | string | Строка записанная из консоли |
| randomString | string | Строка сгенерированная случайно |
| letter | int | Символ алфавита |
| number | int | Число |
| text | sting | Строка из консоли |
| choice | int | Выбор пользователя |
| algorithm | ICipher | Алгоритм шифрования |

Программа использует следующие константы:

Таблица 2 – Константы, используемые в программе

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Тип** | **Значение** | **Описание** |
| LowerLimit | const int | 5 | Минимальное число возможных значений |
| UpperLimit | const int | 12 | Максимальное число возможных значений |
| MaxInt | const int | 2147483647 | Значение первого узла в пустом дереве |
| LENGHTOFALPHABET | const int | 37 | Длинна алфавита |

Таблица 3 – Пользовательские типы

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| UserChoice | Способы ввода данных |
| Interaction | Способы взаимодействия с деревом |

В файле каждое сообщения хранятся в одной строке.

# Структура программы

Программа разделена на следующие модули:

1. CaesarCipher.cs – реализация шифра Цезаря;
2. File.cs – ввод/вывод из файла;
3. Input.cs – генерация данных дерева;
4. Program.cs – функция main;
5. HillCipher.cs – реализация шифра Хилла;
6. GetInput.cs – создаёт объекты классов создателей;
7. IInputData.cs – интерфейс для создателей ввода;
8. WorkWithMatrix.cs – операции с матрицой;

Таблица 4 – Функции, составляющие программу

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| GetData | Получение значения узла |
| SetData | Установка значения узла |
| GetParent | Получение предка узла |
| GetLeft | Получение левого потомка узла |
| GetRight | Получение правого потомка узла |
| SetParent | Установка предка |
| SetRight | Установка правого потомка |
| SetLeft | Установка левого потомка |
| Add | Добавление элемента в дерево |
| DeleteElement | Удаление элемента из дерева |
| PreOrder | Создание массива для обратного обхода |
| InOrder | Создание массива для центрированного обхода |
| PostOrder | Создание массива для обратного обхода |
| PreOrderInner | Вывод элементов дерева прямым обходом |
| InOrderInner | Вывод дерева центрированным обходом |
| PostOrderInner | Вывод элементов дерева обратным обходом |
| FindMinimum | Найти минимальный элемент дерева |
| FindNext | Найти следующий по счёту элемент дерева |
| ShowBinaryTree | Создание массива для вывода дерева в консоль |
| ShowBinaryTreeInner | Вывод дерева в консоль |
| Продолжение таблицы 4 – Функции, составляющие программу | |
| DepthOfTree | Получить глубину дерева |
| EmptyTree | Проверка на пустоту дерева |
| ClearTree | Очистка дерева |
| SaveInFile | Сохранение данных в файл |
| GetDataFromFile | Получение данных из файла |
| GetInt | Получение целого числа |
| RandomInput | Ввод массива случайными значениями |
| Interact | Взаимодействие с деревом |

# Блок-схемы алгоритмов программы

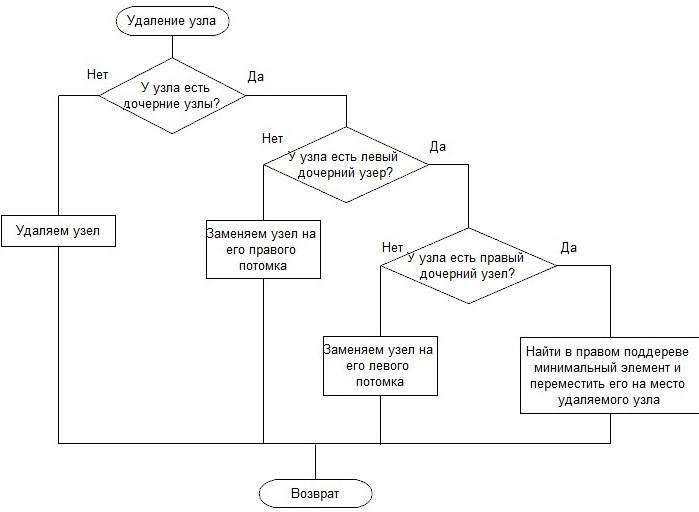


Рисунок 1 – Блок-схема алгоритма решения

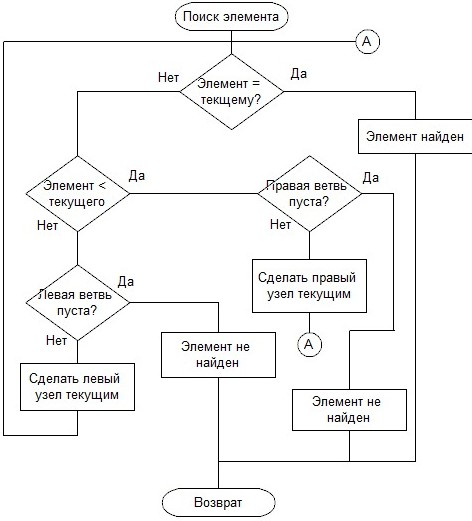


Рисунок 2 – Блок-схема решения алгоритма

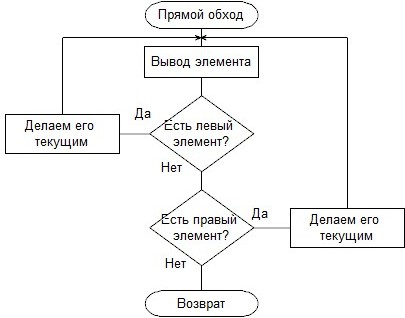


Рисунок 3 – Блок-схема алгоритма решения

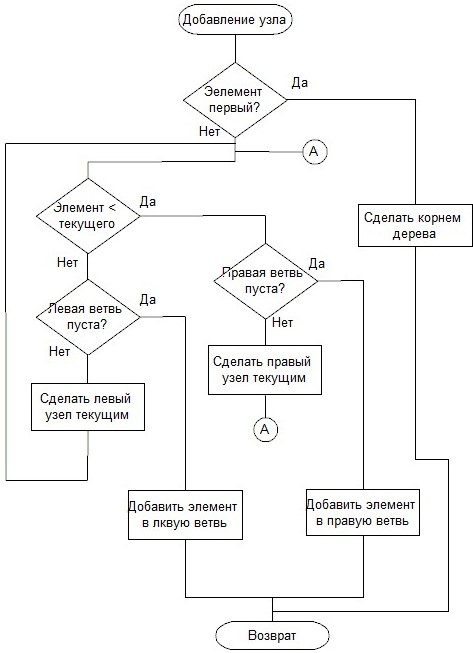


Рисунок 4 – Блок-схема алгоритма решения

# Описание хода выполнения работы

1. В ходе работы было создано решение (Solution) в интегрированной среде разработки Microsoft Visual Studio 2015. В нём был создан проект.

# Результат работы программы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 – Результат работы программы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6 – Результат работы программы

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 7 – Результат работы программы

# Исходный текст программы

[ Начало программы ---]

[BinaryTree.cs]

using System;

using System.Collections.Generic;

using System.Text;

namespace BuildingTree

{

public class Node

{

private int data;

private Node parent;

private Node left;

private Node right;

public Node()

{

parent = null;

left = null;

right = null;

data = 0;

}

public int GetData()

{

return data;

}

public void SetData(int newData)

{

data = newData;

}

public Node GetParent()

{

return parent;

}

public Node GetLeft()

{

return left;

}

public Node GetRight()

{

return right;

}

public void SetParent(Node newParent)

{

parent = newParent;

}

public void SetRight(Node newRight)

{

right = newRight;

}

public void SetLeft(Node newLeft)

{

left = newLeft;

}

}

}

[BinaryTree.cs]

[File.cs]

using System;

using System.Collections.Generic;

using System.IO;

using System.Text;

namespace BuildingTree

{

class File

{

static public void SaveInFile(Tree binaryTree)

{

string path = "";

do

{

Console.WriteLine("Please enter path to file");

path = Console.ReadLine();

FileInfo tempFile = new FileInfo(path);

if (!tempFile.Exists)

{

try

{

StreamWriter newFile = new StreamWriter(path);

newFile.Close();

}

catch

{

Console.WriteLine("Bad name for file, please try again");

continue;

}

}

if (tempFile.Exists)

{

if (tempFile.IsReadOnly)

{

Console.WriteLine("Something wrong with file. please try again");

continue;

}

else

{

Console.WriteLine("Do you want to rewrite file?" + Environment.NewLine”

+ "1 – + Yes");

if (Input.GetInt() != 1)

{

continue;

}

}

}

tempFile.Delete();

break;

}

while (true);

List<int> array = binaryTree.PreOrder();

StreamWriter file = new StreamWriter(path);

for (int i = 0; i < array.Count; i++)

{

file.WriteLine(array[i].ToString());

}

file.Close();

Console.WriteLine("File saved");

}

static public Tree GetDataFromFile()

{

Tree binaryTree = new Tree();

string path;

do

{

List<string> tempArray = new List<string>();

Console.WriteLine("Please enter path to file");

path = Console.ReadLine();

FileInfo tempFile = new FileInfo(path);

if (!tempFile.Exists)

{

Console.WriteLine("File is not existing");

continue;

}

StreamReader tempOpenedFile = new StreamReader(path);

while (!tempOpenedFile.EndOfStream)

{

tempArray.Add(tempOpenedFile.ReadLine());

}

tempOpenedFile.Close();

bool badData = false;

for (int i = 0; i < tempArray.Count; i++)

{

if (!int.TryParse(tempArray[i], out int number))

{

Console.WriteLine("Bad data");

badData = true;

}

}

if (badData)

{

continue;

}

break;

}

while (true);

StreamReader file = new StreamReader(path, false);

List<int> array = new List<int>();

while (!file.EndOfStream)

{

array.Add(Convert.ToInt32(file.ReadLine()));

}

file.Close();

for (int i = 0; i < array.Count; i++)

{

binaryTree.Add(array[i]);

}

Console.WriteLine("Binary tree:" + Environment.NewLine);

binaryTree.ShowBinaryTree();

return binaryTree;

}

}

}

[File.cs]

[GetInput.cs]

using System;

using System.Collections.Generic;

using System.Text;

namespace BuildingTree

{

static class GetInput

{

public enum UserChoice

{

ConsoleInput = 1,

FileInput,

RandomInput,

End

}

static public IInputData GetSomeInput(int choice)

{

IInputData someInput = null;

if((UserChoice)choice == UserChoice.RandomInput)

{

someInput = new RandomInput();

}

if((UserChoice)choice == UserChoice.FileInput)

{

someInput = new FileInput();

}

if((UserChoice)choice == UserChoice.ConsoleInput)

{

someInput = null;

}

return someInput;

}

}

}

[GetInput.cs]

[IInputData.cs]

using BuildingTree;

using System;

using System.Collections.Generic;

using System.IO;

using System.Text;

namespace BuildingTree

{

public interface IInputData

{

Tree input();

}

public class RandomInput : IInputData

{

const int LowerLimit = 5;

const int UpperLimit = 12;

static Random rnd = new Random();

public Tree input()

{

Tree binaryTree = new Tree();

int countOfNUmbers = rnd.Next(LowerLimit, UpperLimit);

int[] array = new int[countOfNUmbers];

for (int i = 0; i < countOfNUmbers; i++)

{

array[i] = i + 1;

}

int n = countOfNUmbers;

while (n > 1)

{

n--;

int f = rnd.Next(n + 1);

int value = array[f];

array[f] = array[n];

array[n] = value;

}

for (int i = 0; i < countOfNUmbers; i++)

{

binaryTree.Add(array[i]);

}

binaryTree.ShowBinaryTree();

return binaryTree;

}

}

public class FileInput : IInputData

{

public Tree input()

{

Tree binaryTree = new Tree();

string path;

do

{

List<string> tempArray = new List<string>();

Console.WriteLine("Please enter path to file");

path = Console.ReadLine();

FileInfo tempFile = new FileInfo(path);

if (!tempFile.Exists)

{

Console.WriteLine("File is not existing");

continue;

}

StreamReader tempOpenedFile = new StreamReader(path);

while (!tempOpenedFile.EndOfStream)

{

tempArray.Add(tempOpenedFile.ReadLine());

}

tempOpenedFile.Close();

bool badData = false;

for (int i = 0; i < tempArray.Count; i++)

{

if (!int.TryParse(tempArray[i], out int number))

{

Console.WriteLine("Bad data");

badData = true;

}

}

if (badData)

{

continue;

}

break;

}

while (true);

StreamReader file = new StreamReader(path, false);

List<int> array = new List<int>();

while (!file.EndOfStream)

{

array.Add(Convert.ToInt32(file.ReadLine()));

}

file.Close();

for (int i = 0; i < array.Count; i++)

{

binaryTree.Add(array[i]);

}

Console.WriteLine("Binary tree:" + Environment.NewLine);

binaryTree.ShowBinaryTree();

return binaryTree;

}

}

}

[IInputData.cs]

[Input.cs]

using System;

using System.Collections.Generic;

using System.Text;

namespace BuildingTree

{

class Input

{

static public int GetInt()

{

int number = 0;

while (true)

{

Console.Write("Please enter integer: ");

string text = Console.ReadLine();

if (int.TryParse(text, out number))

{

break;

}

Console.WriteLine("This is not a integer, try again");

}

return number;

}

}

}

[Input.cs]

[Program.cs]

using System;

using System.Collections.Generic;

using System.IO;

namespace BuildingTree

{

class Program

{

static void Main()

{

Tree binaryTree = new Tree();

Console.WriteLine("Created by: Shishko Daniil Yrevich" + Environment.NewLine +

"Option: 9" + Environment.NewLine + "Aim: Realize binary search tree, show”+

“features, realize function to" +

" create, delete nodes and visualize tree. Also need manual, file and random”+

“enter." + Environment.NewLine +

"Features: key in every node must be greater than keys in left brench and less”+

“than keys in right bench" + Environment.NewLine+ "Welcome.");

do

{

Console.WriteLine(Environment.NewLine + Environment.NewLine +"1 - Console input"

+ Environment.NewLine

+ "2 - File input" + Environment.NewLine + "3 - Random input" +

Environment.NewLine + "4 - End program");

var choice = Input.GetInt();

if((GetInput.UserChoice)choice < GetInput.UserChoice.ConsoleInput ||

(GetInput.UserChoice)choice > GetInput.UserChoice.End)

{

Console.WriteLine("We don't have these choice, please try again");

continue;

}

if((GetInput.UserChoice)choice == GetInput.UserChoice.End)

{

break;

}

IInputData someInput = GetInput.GetSomeInput(choice);

if (someInput != null)

{

binaryTree = someInput.input();

}

UseOfBinaryTree.Interact(binaryTree);

} while (true);

}

}

}

[Program.cs]

[Tree.cs]

using System;

using System.Collections.Generic;

using System.Text;

namespace BuildingTree

{

public class Tree

{

private Node firstNode;

static public int nesting = 0;

static public int maxNesting = 0;

public Tree()

{

firstNode = null;

}

public void Add(int data)

{

AddInner(data, firstNode);

}

private void AddInner(int data, Node currentNode)

{

if (currentNode == null)

{

this.firstNode = new Node();

this.firstNode.SetData(data);

}

if (currentNode != null)

{

if (data < currentNode.GetData())

{

if (currentNode.GetLeft() != null)

{

AddInner(data, currentNode.GetLeft());

}

if (currentNode.GetLeft() == null)

{

currentNode.SetLeft(new Node());

currentNode.GetLeft().SetData(data);

currentNode.GetLeft().SetParent(currentNode);

}

}

if (data > currentNode.GetData())

{

if (currentNode.GetRight() != null)

{

AddInner(data, currentNode.GetRight());

}

if (currentNode.GetRight() == null)

{

currentNode.SetRight(new Node());

currentNode.GetRight().SetData(data);

currentNode.GetRight().SetParent(currentNode);

}

}

}

}

public void DeleteElement(int data)

{

DeleteElementInner(data, firstNode);

}

private void DeleteElementInner(int data, Node currentNode)

{

if (currentNode.GetData() != data)

{

if (currentNode.GetLeft() != null)

{

DeleteElementInner(data, currentNode.GetLeft());

}

if (currentNode.GetRight() != null)

{

DeleteElementInner(data, currentNode.GetRight());

}

}

if (currentNode.GetData() == data)

{

if (currentNode.GetRight() == null && currentNode.GetLeft() == null)

{

if (firstNode != currentNode)

{

if (currentNode.GetParent().GetLeft() != null &&

currentNode.GetParent().GetLeft().GetData() == data)

{

currentNode.GetParent().SetLeft(null);

}

if (currentNode.GetParent().GetRight() != null &&

currentNode.GetParent().GetRight().GetData() == data)

{

currentNode.GetParent().SetRight(null);

}

}

if (firstNode == currentNode)

{

firstNode.SetData(0);

}

}

if (currentNode.GetRight() == null && currentNode.GetLeft() != null)

{

if (firstNode != currentNode)

{

if (currentNode.GetParent().GetLeft() != null)

{

if (currentNode.GetParent().GetLeft().GetData() == data)

{

currentNode.GetParent().SetLeft(currentNode.GetLeft());

currentNode.GetLeft().SetParent(currentNode.GetParent());

}

}

if (currentNode.GetParent().GetRight() != null)

{

if (currentNode.GetParent().GetRight().GetData() == data)

{

currentNode.GetParent().SetRight(currentNode.GetLeft());

currentNode.GetLeft().SetParent(currentNode.GetParent());

}

}

}

if (firstNode == currentNode)

{

currentNode.SetData(currentNode.GetLeft().GetData());

currentNode.SetRight(currentNode.GetLeft().GetRight());

currentNode.SetLeft(currentNode.GetLeft().GetLeft());

}

}

if (currentNode.GetRight() != null && currentNode.GetLeft() == null)

{

if (firstNode != currentNode)

{

if (currentNode.GetParent().GetLeft() != null)

{

if (currentNode.GetParent().GetLeft().GetData() == data)

{

currentNode.GetParent().SetLeft(currentNode.GetRight());

currentNode.GetRight().SetParent(currentNode.GetParent());

}

}

if (currentNode.GetParent().GetRight() != null)

{

if (currentNode.GetParent().GetRight().GetData() == data)

{

currentNode.GetParent().SetRight(currentNode.GetRight());

currentNode.GetRight().SetParent(currentNode.GetParent());

}

}

}

if (firstNode == currentNode)

{

currentNode.SetData(currentNode.GetRight().GetData());

currentNode.SetLeft(currentNode.GetRight().GetLeft());

currentNode.SetRight(currentNode.GetRight().GetRight());

}

}

if (currentNode.GetRight() != null && currentNode.GetLeft() != null)

{

if (currentNode != firstNode)

{

var lowestElement = FindNext(currentNode);

currentNode.SetData(lowestElement.GetData());

if (lowestElement.GetRight() == null)

{

if (lowestElement.GetParent().GetLeft() != null &&

lowestElement.GetParent().GetLeft().GetData() ==

lowestElement.GetData())

{

lowestElement.GetParent().SetLeft(null);

}

if (lowestElement.GetParent().GetRight() != null &&

lowestElement.GetParent().GetRight().GetData() ==

lowestElement.GetData())

{

lowestElement.GetParent().SetRight(null);

}

}

if (lowestElement.GetRight() != null)

{

if (lowestElement.GetParent().GetLeft().GetData() ==

lowestElement.GetData())

{

lowestElement.GetParent().SetLeft(lowestElement.GetRight());

lowestElement.GetRight().SetParent(lowestElement.GetParent());

}

if (lowestElement.GetParent().GetRight().GetData() ==

lowestElement.GetData())

{

lowestElement.GetParent().SetRight(lowestElement.GetRight());

lowestElement.GetRight().SetParent(lowestElement.GetParent());

}

}

}

if (firstNode == currentNode)

{

var arrayList = PreOrder();

arrayList.Remove(currentNode.GetData());

arrayList.Remove(currentNode.GetRight().GetData());

arrayList.Insert(0, currentNode.GetRight().GetData());

ClearTree();

for (int i = 0; i < arrayList.Count; i++)

{

Add(arrayList[i]);

}

}

}

}

if(firstNode.GetRight() == null && firstNode.GetLeft() == null &&

firstNode.GetData() == 0)

{

ClearTree();

}

}

public List<int> PreOrder()

{

List<int> arrayList = new List<int>();

PreOrderInner(arrayList, firstNode);

return arrayList;

}

public List<int> InOrder()

{

List<int> arrayList = new List<int>();

InOrderInner(arrayList, firstNode);

return arrayList;

}

public List<int> PostOrder()

{

List<int> arrayList = new List<int>();

PostOrderInner(arrayList, firstNode);

return arrayList;

}

private void PreOrderInner(List<int> listArray, Node currentNode)

{

listArray.Add(currentNode.GetData());

if (currentNode.GetLeft() != null)

{

PreOrderInner(listArray, currentNode.GetLeft());

}

if (currentNode.GetRight() != null)

{

PreOrderInner(listArray, currentNode.GetRight());

}

}

private void InOrderInner(List<int> listArray, Node currentNode)

{

if (currentNode.GetLeft() != null)

{

InOrderInner(listArray, currentNode.GetLeft());

}

listArray.Add(currentNode.GetData());

if (currentNode.GetRight() != null)

{

InOrderInner(listArray, currentNode.GetRight());

}

}

private void PostOrderInner(List<int> listArray, Node currentNode)

{

if (currentNode.GetLeft() != null)

{

PostOrderInner(listArray, currentNode.GetLeft());

}

if (currentNode.GetRight() != null)

{

PostOrderInner(listArray, currentNode.GetRight());

}

listArray.Add(currentNode.GetData());

}

public Node FindMinimum()

{

return FindMinimumInner(firstNode);

}

private Node FindMinimumInner(Node currentNode)

{

if (currentNode.GetLeft() == null)

{

return currentNode;

}

return FindMinimumInner(currentNode.GetLeft());

}

public Node FindNext(Node currentNode)

{

return FindNextInner(currentNode);

}

private Node FindNextInner(Node currentNode)

{

return FindMinimumInner(currentNode.GetRight());

}

public void ShowBinaryTree()

{

maxNesting = 0;

DepthOfTree();

string[] treeInString = new string[maxNesting + 1];

for (int i = 0; i < maxNesting + 1; i++)

{

treeInString[i] = "";

}

ShowBinaryTreeInner(treeInString, firstNode);

}

private void ShowBinaryTreeInner(string[] treeInString, Node currentNode)

{

int countOfNumbersOnLastLayer = Convert.ToInt32(Math.Pow(2.0,

Convert.ToDouble(maxNesting)));

int countOfNumbersOnLayer = Convert.ToInt32(Math.Pow(2.0,

Convert.ToDouble(nesting)));

var count = countOfNumbersOnLastLayer / countOfNumbersOnLayer;

if (treeInString[nesting] == "")

{

for (int i = 0; i < count - 1; i++)

{

treeInString[nesting] += " ";

}

}

else

{

for (int i = 0; i < (count \* 2) - 1; i++)

{

treeInString[nesting] += " ";

}

}

treeInString[nesting] += currentNode.GetData();

if (currentNode.GetLeft() != null)

{

nesting++;

ShowBinaryTreeInner(treeInString, currentNode.GetLeft());

nesting--;

}

if (currentNode.GetLeft() == null && nesting != maxNesting)

{

for (int i = nesting + 1; i < maxNesting + 1; i++)

{

int tempCountOfNumbersOnLayer = Convert.ToInt32(Math.Pow(2.0,

Convert.ToDouble(i)));

int newCount = countOfNumbersOnLastLayer / tempCountOfNumbersOnLayer;

for (int j = 0; j < Convert.ToInt32(Math.Pow(2.0, Convert.ToDouble(i –

(nesting + 1)))); j++)

{

if (treeInString[i] == "")

{

for (int k = 0; k < newCount - 1; k++)

{

treeInString[i] += " ";

}

treeInString[i] += ".";

}

else

{

for (int k = 0; k < (newCount \* 2) - 1; k++)

{

treeInString[i] += " ";

}

treeInString[i] += ".";

}

}

}

}

if (currentNode.GetRight() != null)

{

nesting++;

ShowBinaryTreeInner(treeInString, currentNode.GetRight());

nesting--;

}

if (currentNode.GetRight() == null && nesting != maxNesting)

{

for (int i = nesting + 1; i < maxNesting + 1; i++)

{

int tempCountOfNumbersOnLayer = Convert.ToInt32(Math.Pow(2.0,

Convert.ToDouble(i)));

int newCount = countOfNumbersOnLastLayer / tempCountOfNumbersOnLayer;

for (int j = 0; j < Convert.ToInt32(Math.Pow(2.0, Convert.ToDouble(i –

(nesting + 1)))); j++)

{

if (treeInString[i] == "")

{

for (int k = 0; k < newCount - 1; k++)

{

treeInString[i] += " ";

}

treeInString[i] += ".";

}

else

{

for (int k = 0; k < (newCount \* 2) - 1; k++)

{

treeInString[i] += " ";

}

treeInString[i] += ".";

}

}

}

}

if (nesting == 0)

{

for (int i = 0; i < maxNesting + 1; i++)

{

Console.WriteLine(treeInString[i]);

}

}

}

public void DepthOfTree()

{

DepthOfTreeInner(firstNode);

}

private void DepthOfTreeInner(Node currentNode)

{

if (maxNesting < nesting)

{

maxNesting = nesting;

}

if (currentNode.GetLeft() != null)

{

nesting++;

DepthOfTreeInner(currentNode.GetLeft());

nesting--;

}

if (currentNode.GetRight() != null)

{

nesting++;

DepthOfTreeInner(currentNode.GetRight());

nesting--;

}

}

public bool EmptyTree()

{

if (firstNode == null)

{

return true;

}

else

{

return false;

}

}

public void ClearTree()

{

firstNode = null;

}

}

}

[Tree.cs]

[UseOfBinaryTree.cs]

using System;

using System.Collections.Generic;

using System.Text;

namespace BuildingTree

{

class UseOfBinaryTree

{

enum Interaction

{

AddElement = 1,

ShowTree,

DeleteElement,

ClearTree,

SaveData,

GoBack

}

public static void Interact(Tree binaryTree)

{

Interaction choice;

do

{

Console.WriteLine(Environment.NewLine + "1 - Add element" + Environment.NewLine +

"2 - Show tree" +

Environment.NewLine + "3 - Delete element" + Environment.NewLine + "4 – Clear”

+ ”tree "+ Environment.NewLine

+"5 - Save data" + Environment.NewLine + "6 - Return");

choice = (Interaction)Input.GetInt();

if (choice == Interaction.AddElement)

{

int data = Input.GetInt();

binaryTree.Add(data);

Console.WriteLine(" ");

}

if (choice == Interaction.ShowTree)

{

if (!binaryTree.EmptyTree())

{

Console.WriteLine("Binary tree:" + Environment.NewLine);

binaryTree.ShowBinaryTree();

}

else

{

Console.WriteLine("Tree is empty");

}

}

if (choice == Interaction.DeleteElement)

{

if (!binaryTree.EmptyTree())

{

int data = Input.GetInt();

binaryTree.DeleteElement(data);

}

else

{

Console.WriteLine("Tree is empty");

}

}

if(choice == Interaction.ClearTree)

{

if (!binaryTree.EmptyTree())

{

binaryTree.ClearTree();

}

if (binaryTree.EmptyTree())

{

Console.WriteLine("Tree is empty");

}

}

if (choice == Interaction.SaveData)

{

File.SaveInFile(binaryTree);

}

if (choice > Interaction.GoBack || choice < Interaction.AddElement)

{

Console.WriteLine("We have not these choice, plese try again");

}

} while (choice != Interaction.GoBack);

}

}

}

[UseOfBinaryTree.cs]

[ --- Конец программы]

# Документирование и комментирование исходного текста