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 № 1

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

Необходимо реализовать заданную структуру данных, продемонстрировать характерные особенности, реализовать возможность добавления и удаления элементов, визуализировать дерево. В программе должны быть предусмотрены три варианта заполнения: пользователем с клавиатуры, из файла и случайными числами.

# Исходные данные

В качестве исходных данных программа использует, вводимое пользователем числа, которые будут представляться в программе как целочисленная переменные, являющимися значениями в определённых узлах.

# Особые ситуации

Необходимо рассмотреть следующие особые ситуации.

* Если пользователь ввёл число, которое уже присутствует в дереве, то оно не может быть в него добавлено.

# Математические методы и алгоритмы решения задач

Дерево обладает следующими свойствами:

* оба поддерева – левое и правое – являются двоичными деревьями;
* у всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X;
* у всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа самого узла Х.

Для целей реализации двоичное дерево поиска можно определить так:

* Двоичное дерево состоит из узлов (вершин) — записей вида (data, left, right), где data — некоторые данные, привязанные к узлу, left и right — ссылки на узлы, являющиеся детьми данного узла — левый и правый сыновья соответственно. Для оптимизации алгоритмов конкретные реализации предполагают также определения поля parent в каждом узле (кроме корневого) — ссылки на родительский элемент.
* Данные (data) обладают ключом (key), на котором определена операция сравнения «меньше». В конкретных реализациях это может быть пара (key, value) — (ключ и значение), или ссылка на такую пару, или простое определение операции сравнения на необходимой структуре данных или ссылке на неё.
* Для любого узла X выполняются свойства дерева поиска: key[left[X]] < key[X] ≤ key[right[X]], то есть ключи данных родительского узла больше ключей данных левого сына и нестрого меньше ключей данных правого.

# Форматы представления данных

Программа использует следующие переменные:

Таблица 1 – Переменные, используемые в программе

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| choice | userChoice | Выбор способа ввода |
| nesting | static int | Вложенность рекурсии |
| maxNesting | static int | Максимальная вложенность рекурсии |
| data | int | Данные (ключ) узла |
| parent | Node | Узел сверху |
| left | Node | Узел слева |
| right | Node | Узел справа |
| path | string | Путь к файлу |
| tempFile | FileInfo | Информация о файле |
| newFile | StreamWriter | Временный файл |
| array | List<int> | Все числа дерева |
| file | StreamWriter | Файл для сохранения/чтения данных |
| tempArray | List<string> | Массив для проверки правильности ланных |
| number | int | Число для ввода |
| binaryTree | Node | Бинарное дерево |
| rnd | Random | Генерация случайных чисел |
| countOfNumbers | int | Количество случайных чисел |
| choice | Interaction | Выбор взаимодействия с деревом |

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 |
| FirstNode | Const int | 2147483647 | Значение первого узла в пустом дереве |

Таблица 3 – Пользовательские типы

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| Node | Узел бинарного дерева |
| UserChoice | Способы ввода данных |
| Interaction | Способы взаимодействия с деревом |

В файле каждое число хранится в отдельной строчке, где первое число соответственно корень дерева.

# Структура программы

Программа разделена на следующие модули:

1. BinaryTree.cs – реализация бинарного дерева;
2. File.cs – ввод/вывод из файла;
3. Input.cs – генерация данных дерева;
4. Program.cs – функция main;
5. UseOfBinaryTree.cs – взаимодействие с деревом.

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 |
| DepthOfTree | Получить глубину дерева |
| EmptyTree | Проверка на пустоту дерева |
| ClearTree | Очистка дерева |
| Продолжение таблицы 4 – Функции, составляющие программу | |
| 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

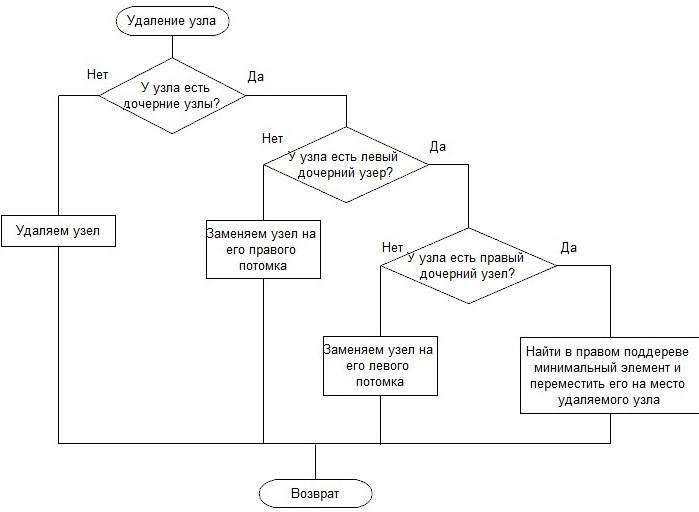


Рисунок 1 – Блок-схема алгоритма решения

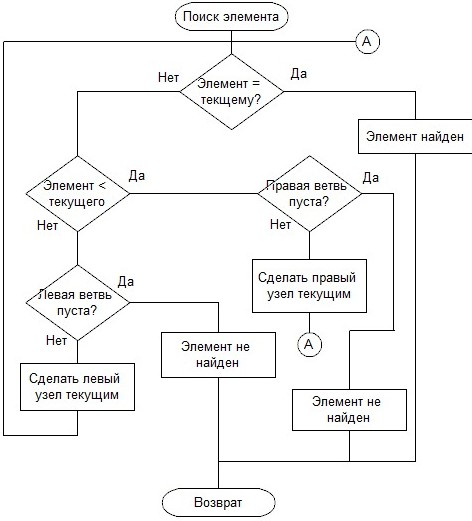


Рисунок 2 – Блок-схема решения алгоритма

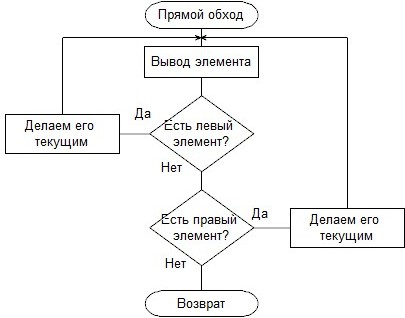


Рисунок 3 – Блок-схема алгоритма решения

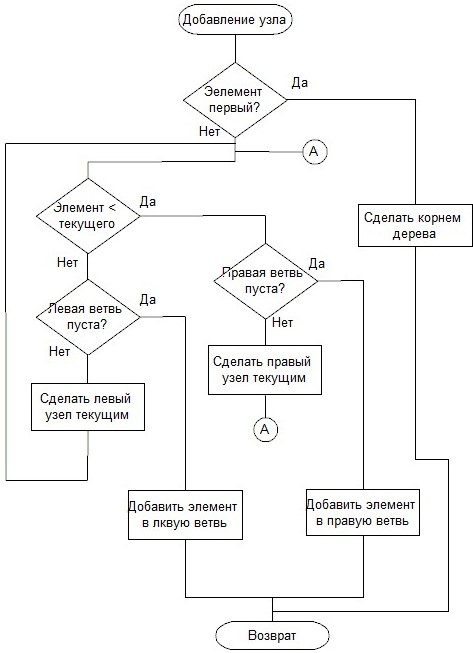


Рисунок 4 – Блок-схема алгоритма решения

# 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.
2. Реализован процесс добавление и удаления элемента, также вывод бинарного дерева в консоль.
3. Удаление корневого элемента выполняется неправильно, исправлено удаление корневого элемента при всех возможных сценариях.
4. Вывод дерева происходит с искажениями, исправлена логика вывода бинарного дерева на экран.

# 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

{

public const int FirstNode = int.MaxValue;

static public int nesting = 0;

static public int maxNesting = 0;

private int data;

private Node parent;

private Node left;

private Node right;

public Node()

{

parent = null;

left = null;

right = null;

data = FirstNode;

}

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;

}

public void Add(int data)

{

if (GetParent() == null && GetLeft() == null && GetRight() == null &&

GetData() == FirstNode)

{

SetData(data);

}

if (GetData() != FirstNode)

{

if (data < GetData())

{

if (GetLeft() != null)

{

GetLeft().Add(data);

}

if (GetLeft() == null)

{

SetLeft(new Node());

GetLeft().SetData(data);

GetLeft().SetParent(this);

}

}

if (data > GetData())

{

if (GetRight() != null)

{

GetRight().Add(data);

}

if (GetRight() == null)

{

SetRight(new Node());

GetRight().SetData(data);

GetRight().SetParent(this);

}

}

}

}

public void DeleteElement(int data)

{

if (GetData() != data)

{

if (GetLeft() != null)

{

GetLeft().DeleteElement(data);

}

if (GetRight() != null)

{

GetRight().DeleteElement(data);

}

}

if (GetData() == data)

{

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

{

if (GetParent() != null)

{

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

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

{

GetParent().SetLeft(null);

}

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

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

{

GetParent().SetRight(null);

}

}

if (GetParent() == null)

{

SetData(0);

}

}

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

{

if (GetParent() != null)

{

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

{

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

{

GetParent().SetLeft(GetLeft());

GetLeft().SetParent(GetParent());

}

}

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

{

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

{

GetParent().SetRight(GetLeft());

GetLeft().SetParent(GetParent());

}

}

}

if (GetParent() == null)

{

SetData(GetLeft().GetData());

SetLeft(GetLeft().GetLeft());

SetRight(GetLeft().GetRight());

}

}

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

{

if (GetParent() != null)

{

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

{

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

{

GetParent().SetLeft(GetRight());

GetRight().SetParent(GetParent());

}

}

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

{

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

{

GetParent().SetRight(GetRight());

GetRight().SetParent(GetParent());

}

}

}

if (GetParent() == null)

{

SetData(GetRight().GetData());

SetLeft(GetRight().GetLeft());

SetRight(GetRight().GetRight());

}

}

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

{

if (GetParent() != null)

{

var lowestElement = FindNext();

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 (GetParent() == null)

{

var arrayList = PreOrder();

arrayList.Remove(GetData());

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

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

ClearTree();

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

{

Add(arrayList[i]);

}

}

}

}

}

public List<int> PreOrder()

{

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

PreOrderInner(arrayList);

return arrayList;

}

public List<int> InOrder()

{

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

InOrderInner(arrayList);

return arrayList;

}

public List<int> PostOrder()

{

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

PostOrderInner(arrayList);

return arrayList;

}

private void PreOrderInner(List<int> listArray)

{

listArray.Add(GetData());

if (GetLeft() != null)

{

GetLeft().PreOrderInner(listArray);

}

if (GetRight() != null)

{

GetRight().PreOrderInner(listArray);

}

}

private void InOrderInner(List<int> listArray)

{

if (GetLeft() != null)

{

GetLeft().PreOrderInner(listArray);

}

listArray.Add(GetData());

if (GetRight() != null)

{

GetRight().PreOrderInner(listArray);

}

}

private void PostOrderInner(List<int> listArray)

{

if (GetLeft() != null)

{

GetLeft().PreOrderInner(listArray);

}

if (GetRight() != null)

{

GetRight().PreOrderInner(listArray);

}

listArray.Add(GetData());

}

public Node FindMinimum()

{

if (GetLeft() == null)

{

return this;

}

return GetLeft().FindMinimum();

}

public Node FindNext()

{

return GetRight().FindMinimum();

}

public void ShowBinaryTree()

{

maxNesting = 0;

DepthOfTree();

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

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

{

treeInString[i] = "";

}

ShowBinaryTreeInner(treeInString);

}

private void ShowBinaryTreeInner(string[] treeInString)

{

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] += GetData();

if (GetLeft() != null)

{

nesting++;

GetLeft().ShowBinaryTreeInner(treeInString);

nesting--;

}

if (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 (GetRight() != null)

{

nesting++;

GetRight().ShowBinaryTreeInner(treeInString);

nesting--;

}

if (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()

{

if (maxNesting < nesting)

{

maxNesting = nesting;

}

if (GetLeft() != null)

{

nesting++;

GetLeft().DepthOfTree();

nesting--;

}

if (GetRight() != null)

{

nesting++;

GetRight().DepthOfTree();

nesting--;

}

}

public bool EmptyTree()

{

if (GetLeft() == null && GetRight() == null && GetData() == FirstNode)

{

return true;

}

else

{

return false;

}

}

public void ClearTree()

{

SetLeft(null);

SetRight(null);

SetData(FirstNode);

}

}

}

[BinaryTree.cs]

[File.cs]

using System;

using System.Collections.Generic;

using System.IO;

using System.Text;

namespace buildingTree

{

class File

{

static public void SaveInFile(Node 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 Node GetDataFromFile()

{

Node binaryTree = new Node();

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]

[Input.cs]

using System;

using System.Collections.Generic;

using System.Text;

namespace buildingTree

{

class Input

{

const int LowerLimit = 5;

const int UpperLimit = 12;

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;

}

static public Node RandomInput()

{

Node binaryTree = new Node();

Random rnd = new Random();

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;

}

}

}

[Input.cs]

[Program.cs]

using System;

using System.Collections.Generic;

using System.IO;

namespace buildingTree

{

class Program

{

enum UserChoice

{

consoleInput = 1,

fileInput,

randomInput,

end

}

static void Main()

{

Node binaryTree = new Node();

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.");

UserChoice choice;

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");

choice = (UserChoice)Input.GetInt();

if (choice == UserChoice.consoleInput)

{

}

if (choice == UserChoice.fileInput)

{

binaryTree = File.GetDataFromFile();

}

if (choice == UserChoice.randomInput)

{

binaryTree = Input.RandomInput();

}

if (choice == UserChoice.end)

{

break;

}

if (choice > UserChoice.end || choice < UserChoice.consoleInput)

{

Console.WriteLine("We dont have these option, please try again");

continue;

}

UseOfBinaryTree.Interact(binaryTree);

} while (true);

}

}

}

[Program.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(Node 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