Минобрнауки России

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт

(технический университет)»

­­­­

**Дисциплина «Разработка ПС»**

Отчёт по лабораторной работе №1.

Алгоритмы и структуры данных

Реализовать AVL дерево.

Преподаватели: Корниенко Иван Григорьевич

Федин Алексей Константинович

Исполнил студент 405 группы: Вишняков Виталий

Санкт-Петербург

2022

Постановка задачи

Необходимо реализовать AVL дерево, 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.

АВЛ-дерево — сбалансированное по высоте [двоичное дерево поиска](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0): для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

Исходные данные

В качестве исходных данных программа использует: пользовательский ввод чисел через консоль, из файла и случайными числами.

Особые ситуации

Необходимо рассмотреть следующие особые ситуации:

1. Некорректный ввод элементов дерева (символы, буквы, лишних пробелов)
2. Отсутствие ожидаемых программой файлов на чтение или содержание некорректных данных внутри существующих файлов.
3. Запись работы программы в уже существующий файл или создание недопустимого файла, а также проверка атрибутов существующего файла (Атрибут «Только для чтения»).

Математические методы и алгоритмы решения задач



Форматы представления данных

Формат внешних файлов, из которых производится ввод данных – строго текстовый формат txt. Внутри читаемого txt файла для корректной работы должны присутствовать цифры и числа разделённые знаком пробела.

Формат файлов для сохранения результатов работы программы - только текстовый формат txt.

Структура программы

Программа разбита на 4 класса, также отдельно подключен проект для тестирования.

Основная последовательность работы программы – ожидания решения пользователя. Программа ожидает пользовательские команды. Программа выводит дерево по команде, сохраняет его в файл. Может добавлять и убирать элементы.

Таблица 1 – Основные переменные программы

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| tree | AVL | Дерево |
| currentComand | string | Содержит текущую команду |
| isWorking | bool | Работа программы |

Таблица 2 – Классы, используемые в программе

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Описание** | **Методы** | **Описание** |
| AVL | Осуществляет работу с деревом | DisplayTreeInFile  Add  Balance\_tree  Delete  DisplayTree | Сохраняет дерево в файл  Добавляет элемент  Балансирует дерево  Удаляет элемент  Выводит в консоль дерево |
| Node | Узел дерева | Конструктор |  |
| AuxFunc | Вспомогательные функции | SayHello  ShowComand  GetInt  GetRandNums  GetDataFromFile | Выводит приветствие  Выводит список команд  Получить целое число  Получить случайные числа  Получить числа из файла |

Блок-схемы алгоритмов программы



Рисунок 1 - Блок-схема алгоритма добавления

Описание хода выполнения лабораторной работы

В ходе лабораторной работы было создано решение (Solution) интегрированной среде разработки Microsoft Visual Studio C# 2019. В нём был создан проект.

Программа была разбита на 4 условные части: модуль основной программы, модуль с классом вспомогательных функций, модуль с классом узла и модуль с классом дерева.

После написания основного цикла работы программы, были созданы функции пользовательского ввода, чтения данных из файла и случайного заполнения дерева. После был создан алгоритм печати полученных данных.

В ходе работы над проектом были учтены и обработаны ошибки ввода некорректных данных, некорректное чтение файлов, а также обработаны возникающие исключения.

Также в код программы были включены модульные тесты, предназначенные для тестирования программы. Данные для тестирования заранее подготовлены.

Результаты работы программы

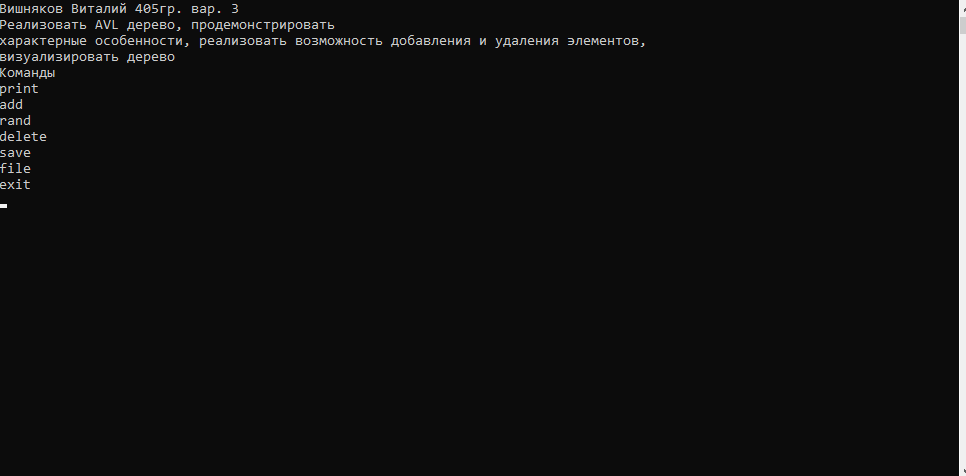


Рисунок 2 – Пример работы программы при старте

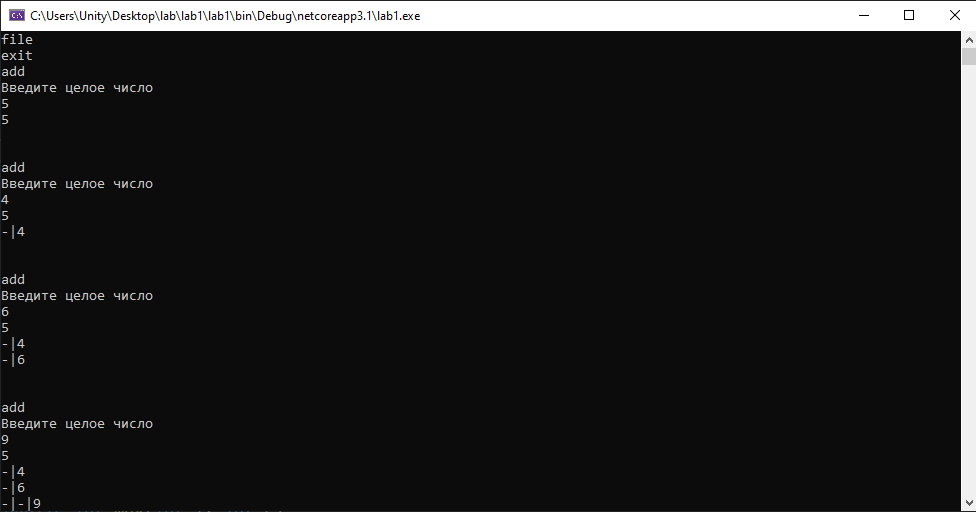


Рисунок 3 – Пример работы программы при работе команды add

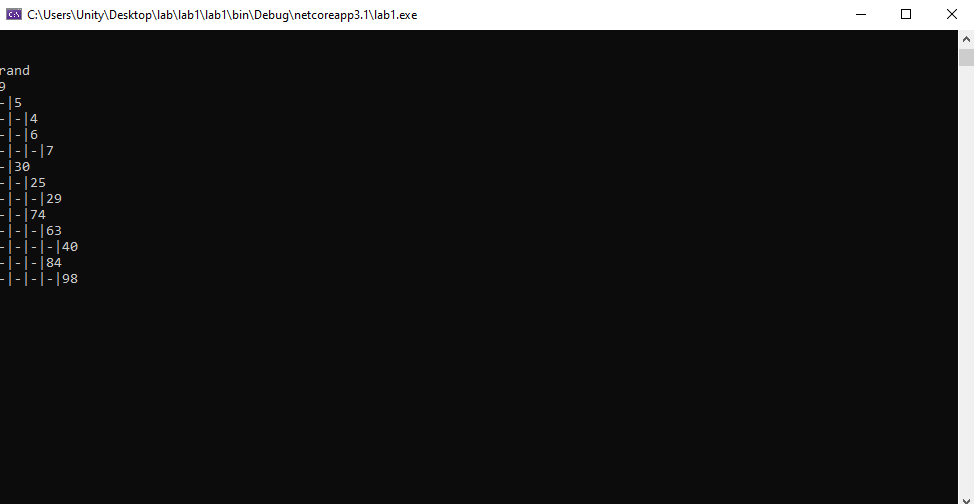


Рисунок 4 – Пример работы программы при работе команды rand, заполняет рандомными числами

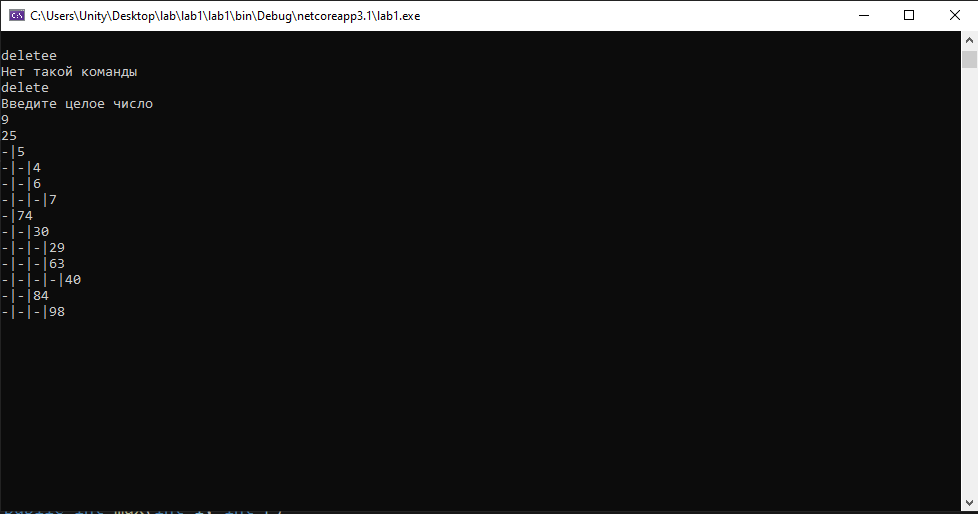


Рисунок 5 – Пример работы программы при работе команды delete

Исходный текст программы

[Начало программы ---]

[Начало Program.cs ---]

using System;

using System.Collections.Generic;

using System.Linq;

namespace lab1

{

class Program

{

static void Main(string[] args)

{

AuxFunc.SayHello();

AuxFunc.ShowComand();

bool isWorking = true;

AVL tree = new AVL();

string currentComand = "";

while (isWorking)

{

currentComand = Console.ReadLine().Trim();

if (currentComand == "exit")

{

isWorking = false;

}

else if (currentComand == "add") {

tree.Add(AuxFunc.GetInt());

tree.DisplayTree();

}

else if (currentComand == "delete") {

tree.Delete(AuxFunc.GetInt());

tree.DisplayTree();

}

else if (currentComand == "print") {

tree.DisplayTree();

}

else if (currentComand == "rand")

{

int[] tmp = AuxFunc.GetRandNums();

for (int i = 0; i < tmp.Length; i++)

{

tree.Add(tmp[i]);

}

tree.DisplayTree();

}

else if (currentComand == "save")

{

tree.DisplayTreeInFile();

}

else if (currentComand == "file")

{

int[] tmp = AuxFunc.GetDataFromFile();

if (tmp!= null) {

for (int i = 0; i < tmp.Length; i++)

{

tree.Add(tmp[i]);

}

tree.DisplayTree();

}

}

else

{

Console.WriteLine("Нет такой команды");

}

}

}

}

}

[Конец Program.cs --- ]

[Начало AuxFunc.cs ---]

using System;

using System.Collections.Generic;

using System.Text;

using System.IO;

using System.Linq;

namespace lab1

{

class AuxFunc

{

public static void SayHello()

{

Console.WriteLine("Вишняков Виталий 405гр. вар. 3");

Console.WriteLine(@"Реализовать AVL дерево, продемонстрировать

характерные особенности, реализовать возможность добавления и удаления элементов,

визуализировать дерево");

}

public static void ShowComand()

{

Console.WriteLine("Команды");

Console.WriteLine("print");

Console.WriteLine("add");

Console.WriteLine("rand");

Console.WriteLine("delete");

Console.WriteLine("save");

Console.WriteLine("file");

Console.WriteLine("exit");

}

public static int GetInt()

{

int tmp = 0;

Console.WriteLine("Введите целое число");

while (!int.TryParse(Console.ReadLine(), out tmp))

{

Console.WriteLine("Ошибка ввода! Введите целое число");

}

return tmp;

}

public static int[] GetRandNums()

{

Random rnd = new Random();

int len = rnd.Next(5, 15);

int[] arr = new int[len];

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

{

arr[i]= rnd.Next(1,100);

}

return arr;

}

public static int[] GetDataFromFile()

{

Console.WriteLine("Введите имя файла");

string path = Console.ReadLine();

if (File.Exists(path))

{

try

{

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

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

arr = File.ReadAllText(path).Split(' ').ToList();

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

{

int number;

bool success = int.TryParse(arr[i], out number);

if (success)

{

arrInt.Add(number);

}

}

return arrInt.ToArray();

}

catch (Exception)

{

Console.WriteLine("Ошибка в данных");

return null;

}

}

else

{

Console.WriteLine("Файл не найден");

return null;

}

}

}

}

[Конец AuxFunc.cs --- ]

[Начало AVL.cs ---]

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace lab1

{

public class AVL : InterfaceAVL

{

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

public Node root;

public void Add(int data)

{

Node newItem = new Node(data);

if (root == null)

{

root = newItem;

}

else

{

root = RecursiveInsert(root, newItem);

}

}

private Node RecursiveInsert(Node current, Node newItem)

{

if (current == null)

{

current = newItem;

return current;

}

else if (newItem.data < current.data)

{

current.left = RecursiveInsert(current.left, newItem);

current = BalanceTree(current);

}

else if (newItem.data > current.data)

{

current.right = RecursiveInsert(current.right, newItem);

current = BalanceTree(current);

}

return current;

}

private Node BalanceTree(Node current)

{

int b\_factor = BalanceRate(current);

if (b\_factor > 1)

{

if (BalanceRate(current.left) > 0)

{

current = SmallRight(current);

}

else

{

current = BigRight(current);

}

}

else if (b\_factor < -1)

{

if (BalanceRate(current.right) > 0)

{

current = BigLeft(current);

}

else

{

current = SmallLeft(current);

}

}

return current;

}

public void Delete(int target)

{

root = Delete(root, target);

}

public Node Delete(Node current, int target)

{

Node parent;

if (current == null)

{ return null; }

else

{

//left subtree

if (target < current.data)

{

current.left = Delete(current.left, target);

if (BalanceRate(current) == -2)

{

if (BalanceRate(current.right) <= 0)

{

current = SmallLeft(current);

}

else

{

current = BigLeft(current);

}

}

}

//right subtree

else if (target > current.data)

{

current.right = Delete(current.right, target);

if (BalanceRate(current) == 2)

{

if (BalanceRate(current.left) >= 0)

{

current = SmallRight(current);

}

else

{

current = BigRight(current);

}

}

}

//if target is found

else

{

if (current.right != null)

{

parent = current.right;

while (parent.left != null)

{

parent = parent.left;

}

current.data = parent.data;

current.right = Delete(current.right, parent.data);

if (BalanceRate(current) == 2)//rebalancing

{

if (BalanceRate(current.left) >= 0)

{

current = SmallRight(current);

}

else { current = BigRight(current); }

}

}

else

{ //if current.left != null

return current.left;

}

}

}

return current;

}

public void DisplayTree()

{

if (root == null)

{

Console.WriteLine("Tree is empty");

return;

}

arrForPrint = new List<string>();

for (int i = 0; i < getHeight(root); i++)

{

arrForPrint.Add("");

}

InOrderDisplayTree(root, 0);

Console.WriteLine();

for (int i = 0; i < getHeight(root) ; i++)

{

Console.Write(string.Concat(Enumerable.Repeat(" ", (4 - i) \* 2)));

Console.WriteLine(arrForPrint[i]);

}

Console.WriteLine();

}

public void DisplayTreeInFile()

{

Console.WriteLine("Введите имя файла");

string path=Console.ReadLine();

if (root == null)

{

File.AppendAllText(path,"Tree is empty");

return;

}

arrForPrint = new List<string>();

for (int i = 0; i < getHeight(root); i++)

{

arrForPrint.Add("");

}

InOrderDisplayTree(root, 0);

File.AppendAllText(path, Environment.NewLine);

for (int i = 0; i < getHeight(root); i++)

{

File.AppendAllText(path, string.Concat(Enumerable.Repeat(" ", (4 - i) \* 2)));

File.AppendAllText(path, arrForPrint[i] + Environment.NewLine);

}

File.AppendAllText(path, Environment.NewLine);

Console.WriteLine("Сделано");

}

private void InOrderDisplayTree(Node current, int currentHeight)

{

if (current != null)

{

arrForPrint[currentHeight] += current.data.ToString() + " ";

InOrderDisplayTree(current.left, currentHeight + 1);

InOrderDisplayTree(current.right, currentHeight + 1);

}

}

public int max(int l, int r)

{

return l > r ? l : r;

}

public int getHeight(Node current)

{

int height = 0;

if (current != null)

{

int l = getHeight(current.left);

int r = getHeight(current.right);

int m = max(l, r);

height = m + 1;

}

return height;

}

private int BalanceRate(Node current)

{

int l = getHeight(current.left);

int r = getHeight(current.right);

int b\_factor = l - r;

return b\_factor;

}

private Node SmallLeft(Node parent)

{

Node pivot = parent.right;

parent.right = pivot.left;

pivot.left = parent;

return pivot;

}

private Node SmallRight(Node parent)

{

Node pivot = parent.left;

parent.left = pivot.right;

pivot.right = parent;

return pivot;

}

private Node BigRight(Node parent)

{

Node pivot = parent.left;

parent.left = SmallLeft(pivot);

return SmallRight(parent);

}

private Node BigLeft(Node parent)

{

Node pivot = parent.right;

parent.right = SmallRight(pivot);

return SmallLeft(parent);

}

}

}

[Конец AVL.cs --- ]

[Начало Node.cs ---]

using System;

using System.Collections.Generic;

using System.Text;

namespace lab1

{

public class Node : InterfaceNode

{

public int data;

public Node left;

public Node right;

public Node(int data)

{

this.data = data;

}

}

}

[Конец Node.cs --- ]

[Начало UnitTest1.cs ---]

using Microsoft.VisualStudio.TestTools.UnitTesting;

using lab1;

namespace TestProject1

{

[TestClass]

public class UnitTest1

{

AVL tree = new AVL();

Node expectForAdd = new Node(5);

const int expectForHeight = 2;

const int expectForMax = 10;

const Node expectForDelete = null;

[TestMethod]

public void AddTest() {

tree.Add(5);

Assert.AreEqual(expectForAdd.data, tree.root.data);

}

[TestMethod]

public void GetHeightTest() {

tree.Add(1);

tree.Add(2);

var height = tree.getHeight(tree.root);

Assert.AreEqual(expectForHeight, height);

}

[TestMethod]

public void DeleteTest() {

tree.Delete(5);

tree.Delete(1);

tree.Delete(2);

Assert.AreEqual(expectForDelete, tree.root);

}

[TestMethod]

public void MaxTest() {

Assert.AreEqual(expectForMax, tree.max(10, 9));

}

}

}

[Конец UnitTest1.cs --- ]

[Конец программы --- ]