Министерство образования и науки Российской Федерации

Северо-Кавказский горно-металлургический институт (государственный технологический университет)

Кафедра информатики и вычислительной техники

**ОТЧЕТ**

**по лабораторной работе № 2**

**по дисциплине:**

**«лингвистическое и программное обеспечение САПР»**

**Выполнил ст.гр. ИВб-22-2**:

Степанян Константин Александрович

**Проверил**:

Преподаватель: Фомин Андрей Николаевич

Владикавказ 2025

ВАРИАНТ 1

Задание:

Для каждого из рассмотренных примеров реализуйте приложение на языке C#. Убедитесь в работоспособности примеров.

Код (Пример 1):

**using** System**;**

// Класс для узла дерева

class Node

**{**

public int Data**;**

public Node Left**,** Right**;**

public Node**(**int data**)**

**{**

Data **=** data**;**

Left **=** Right **=** null**;**

**}**

**}**

// Класс для бинарного дерева поиска

class BST

**{**

public Node Root**;**

// Вставка узла

public Node Insert**(**Node root**,** int data**)**

**{**

**if** **(**root **==** null**)**

**return** **new** Node**(**data**);**

**if** **(**data **<** root**.**Data**)**

root**.**Left **=** Insert**(**root**.**Left**,** data**);**

**else** **if** **(**data **>** root**.**Data**)**

root**.**Right **=** Insert**(**root**.**Right**,** data**);**

**return** root**;**

**}**

// Прямой обход (pre-order: корень, лево, право)

public void PreOrder**(**Node root**)**

**{**

**if** **(**root **==** null**)** **return;**

Console**.**Write**(**root**.**Data **+** " "**);**

PreOrder**(**root**.**Left**);**

PreOrder**(**root**.**Right**);**

**}**

// Симметричный обход (in-order: лево, корень, право)

public void InOrder**(**Node root**)**

**{**

**if** **(**root **==** null**)** **return;**

InOrder**(**root**.**Left**);**

Console**.**Write**(**root**.**Data **+** " "**);**

InOrder**(**root**.**Right**);**

**}**

// Обратный обход (post-order: лево, право, корень)

public void PostOrder**(**Node root**)**

**{**

**if** **(**root **==** null**)** **return;**

PostOrder**(**root**.**Left**);**

PostOrder**(**root**.**Right**);**

Console**.**Write**(**root**.**Data **+** " "**);**

**}**

// Высота дерева

public int Height**(**Node root**)**

**{**

**if** **(**root **==** null**)** **return** 0**;**

int leftHeight **=** Height**(**root**.**Left**);**

int rightHeight **=** Height**(**root**.**Right**);**

**return** Math**.**Max**(**leftHeight**,** rightHeight**)** **+** 1**;**

**}**

// Проверка движения от корня к узлу

public void PathToNode**(**Node root**,** int value**)**

**{**

Console**.**Write**(**$"Путь к {value}: "**);**

Node current **=** root**;**

**while** **(**current **!=** null **&&** current**.**Data **!=** value**)**

**{**

Console**.**Write**(**current**.**Data **+** " -> "**);**

**if** **(**value **<** current**.**Data**)**

current **=** current**.**Left**;**

**else**

current **=** current**.**Right**;**

**}**

**if** **(**current **!=** null**)**

Console**.**WriteLine**(**current**.**Data**);**

**else**

Console**.**WriteLine**(**"Узел не найден"**);**

**}**

**}**

class Program

**{**

static void Main**()**

**{**

BST tree **=** **new** BST**();**

int**[]** values **=** **{** 1**,** 2**,** 3**,** 4**,** 5**,** 6**,** 7**,** 8**,** 9 **};**

// Построение дерева

foreach **(**int value in values**)**

**{**

tree**.**Root **=** tree**.**Insert**(**tree**.**Root**,** value**);**

**}**

Console**.**WriteLine**(**"Прямой обход:"**);**

tree**.**PreOrder**(**tree**.**Root**);**

Console**.**WriteLine**();**

Console**.**WriteLine**(**"Симметричный обход:"**);**

tree**.**InOrder**(**tree**.**Root**);**

Console**.**WriteLine**();**

Console**.**WriteLine**(**"Обратный обход:"**);**

tree**.**PostOrder**(**tree**.**Root**);**

Console**.**WriteLine**();**

Console**.**WriteLine**(**$"Высота дерева: {tree.Height(tree.Root)}"**);**

// Проверка пути к узлу (например, к 8)

tree**.**PathToNode**(**tree**.**Root**,** 8**);**

// Добавляем паузу, чтобы консоль не закрывалась

Console**.**WriteLine**(**"\nНажмите Enter, чтобы закрыть..."**);**

Console**.**ReadLine**();**

**}**

**}**

Результат:



Код (Пример 2):

**using** System**;**

class BinaryTreeArray

**{**

static int**[]** A **=** **{** 0**,** 3**,** 2**,** 4**,** 1**,** 6**,** 5**,** 8**,** 7**,** 9 **};** // Массив A (индекс 0 не используется)

// Корень

static int Root**()**

**{**

**return** A**[**1**];**

**}**

// Левый потомок узла i

static int LeftChild**(**int i**)**

**{**

int left **=** 2 **\*** i**;**

**return** left **<** A**.**Length **?** A**[**left**]** **:** **-**1**;** // -1, если узла нет

**}**

// Правый потомок узла i

static int RightChild**(**int i**)**

**{**

int right **=** 2 **\*** i **+** 1**;**

**return** right **<** A**.**Length **?** A**[**right**]** **:** **-**1**;** // -1, если узла нет

**}**

// Родитель узла i

static int Parent**(**int i**)**

**{**

int parent **=** i **/** 2**;**

**return** parent **>** 0 **?** A**[**parent**]** **:** **-**1**;** // -1, если родителя нет

**}**

static void Main**()**

**{**

Console**.**WriteLine**(**$"Корень: {Root()}"**);**

// Пример для узла с индексом 2 (значение 2)

int i **=** 2**;**

Console**.**WriteLine**(**$"Узел {A[i]}:"**);**

Console**.**WriteLine**(**$"Левый потомок: {LeftChild(i)}"**);**

Console**.**WriteLine**(**$"Правый потомок: {RightChild(i)}"**);**

Console**.**WriteLine**(**$"Родитель: {Parent(i)}"**);**

// Пример для узла с индексом 3 (значение 4)

i **=** 3**;**

Console**.**WriteLine**(**$"\nУзел {A[i]}:"**);**

Console**.**WriteLine**(**$"Левый потомок: {LeftChild(i)}"**);**

Console**.**WriteLine**(**$"Правый потомок: {RightChild(i)}"**);**

Console**.**WriteLine**(**$"Родитель: {Parent(i)}"**);**

// Добавляем паузу, чтобы консоль не закрывалась

Console**.**WriteLine**(**"\nНажмите Enter, чтобы закрыть..."**);**

Console**.**ReadLine**();**

**}**

**}**

Результат:



Задание:

Разработать программу для работы с графами, имеющую следующий функционал:

Код:

**using** System**;**

**using** System**.**IO**;**

**using** System**.**Collections**.**Generic**;**

class Graph

**{**

private int**[,]** matrix**;** // Матрица смежности (вес ребра, 0 = нет ребра)

private int vertices**;** // Количество вершин

private int**[,]** coordinates**;** // Координаты вершин (x, y)

public Graph**(**int v**)**

**{**

vertices **=** v**;**

matrix **=** **new** int**[**v**,** v**];**

coordinates **=** **new** int**[**v**,** 2**];** // [i, 0] = x, [i, 1] = y

**}**

// Считывание графа из файла

public void LoadFromFile**(**string filename**)**

**{**

var lines **=** File**.**ReadAllLines**(**filename**);**

vertices **=** int**.**Parse**(**lines**[**0**]);** // Количество вершин

matrix **=** **new** int**[**vertices**,** vertices**];**

coordinates **=** **new** int**[**vertices**,** 2**];**

// Считываем координаты вершин

**for** **(**int i **=** 0**;** i **<** vertices**;** i**++)**

**{**

var coords **=** lines**[**i **+** 1**].**Split**();**

coordinates**[**i**,** 0**]** **=** int**.**Parse**(**coords**[**0**]);** // x

coordinates**[**i**,** 1**]** **=** int**.**Parse**(**coords**[**1**]);** // y

**}**

// Считываем матрицу смежности

**for** **(**int i **=** 0**;** i **<** vertices**;** i**++)**

**{**

var row **=** lines**[**i **+** vertices **+** 1**].**Split**();**

**for** **(**int j **=** 0**;** j **<** vertices**;** j**++)**

matrix**[**i**,** j**]** **=** int**.**Parse**(**row**[**j**]);**

**}**

**}**

// Сохранение графа в файл

public void SaveToFile**(**string filename**)**

**{**

**using** **(**StreamWriter writer **=** **new** StreamWriter**(**filename**))**

**{**

writer**.**WriteLine**(**vertices**);** // Количество вершин

**for** **(**int i **=** 0**;** i **<** vertices**;** i**++)**

writer**.**WriteLine**(**$"{coordinates[i, 0]} {coordinates[i, 1]}"**);** // Координаты вершин

**for** **(**int i **=** 0**;** i **<** vertices**;** i**++)**

**{**

**for** **(**int j **=** 0**;** j **<** vertices**;** j**++)**

writer**.**Write**(**matrix**[**i**,** j**]** **+** **(**j **<** vertices **-** 1 **?** " " **:** ""**));**

writer**.**WriteLine**();**

**}**

**}**

**}**

// Вывод графа (матрица смежности и координаты)

public void Display**()**

**{**

Console**.**WriteLine**(**"Матрица смежности:"**);**

**for** **(**int i **=** 0**;** i **<** vertices**;** i**++)**

**{**

**for** **(**int j **=** 0**;** j **<** vertices**;** j**++)**

Console**.**Write**(**matrix**[**i**,** j**]** **+** " "**);**

Console**.**WriteLine**();**

**}**

Console**.**WriteLine**(**"\nКоординаты вершин:"**);**

**for** **(**int i **=** 0**;** i **<** vertices**;** i**++)**

Console**.**WriteLine**(**$"Вершина {i}: ({coordinates[i, 0]}, {coordinates[i, 1]})"**);**

**}**

// Добавление вершины

public void AddVertex**(**int x**,** int y**)**

**{**

var newMatrix **=** **new** int**[**vertices **+** 1**,** vertices **+** 1**];**

**for** **(**int i **=** 0**;** i **<** vertices**;** i**++)**

**for** **(**int j **=** 0**;** j **<** vertices**;** j**++)**

newMatrix**[**i**,** j**]** **=** matrix**[**i**,** j**];**

var newCoords **=** **new** int**[**vertices **+** 1**,** 2**];**

**for** **(**int i **=** 0**;** i **<** vertices**;** i**++)**

**{**

newCoords**[**i**,** 0**]** **=** coordinates**[**i**,** 0**];**

newCoords**[**i**,** 1**]** **=** coordinates**[**i**,** 1**];**

**}**

newCoords**[**vertices**,** 0**]** **=** x**;**

newCoords**[**vertices**,** 1**]** **=** y**;**

vertices**++;**

matrix **=** newMatrix**;**

coordinates **=** newCoords**;**

**}**

// Добавление ребра

public void AddEdge**(**int u**,** int v**,** int weight**)**

**{**

matrix**[**u**,** v**]** **=** weight**;**

matrix**[**v**,** u**]** **=** weight**;** // Для неориентированного графа

**}**

// Удаление вершины

public void RemoveVertex**(**int v**)**

**{**

var newMatrix **=** **new** int**[**vertices **-** 1**,** vertices **-** 1**];**

var newCoords **=** **new** int**[**vertices **-** 1**,** 2**];**

int row **=** 0**,** col**;**

**for** **(**int i **=** 0**;** i **<** vertices**;** i**++)**

**{**

**if** **(**i **==** v**)** **continue;**

col **=** 0**;**

**for** **(**int j **=** 0**;** j **<** vertices**;** j**++)**

**{**

**if** **(**j **==** v**)** **continue;**

newMatrix**[**row**,** col**]** **=** matrix**[**i**,** j**];**

col**++;**

**}**

newCoords**[**row**,** 0**]** **=** coordinates**[**i**,** 0**];**

newCoords**[**row**,** 1**]** **=** coordinates**[**i**,** 1**];**

row**++;**

**}**

vertices**--;**

matrix **=** newMatrix**;**

coordinates **=** newCoords**;**

**}**

// Удаление ребра

public void RemoveEdge**(**int u**,** int v**)**

**{**

matrix**[**u**,** v**]** **=** 0**;**

matrix**[**v**,** u**]** **=** 0**;**

**}**

// Обход в ширину (BFS)

public void BFS**(**int start**)**

**{**

bool**[]** visited **=** **new** bool**[**vertices**];**

Queue**<**int**>** queue **=** **new** Queue**<**int**>();**

visited**[**start**]** **=** **true;**

queue**.**Enqueue**(**start**);**

Console**.**WriteLine**(**"Обход в ширину:"**);**

**while** **(**queue**.**Count **>** 0**)**

**{**

int v **=** queue**.**Dequeue**();**

Console**.**Write**(**v **+** " "**);**

**for** **(**int i **=** 0**;** i **<** vertices**;** i**++)**

**{**

**if** **(**matrix**[**v**,** i**]** **!=** 0 **&&** **!**visited**[**i**])**

**{**

visited**[**i**]** **=** **true;**

queue**.**Enqueue**(**i**);**

**}**

**}**

**}**

Console**.**WriteLine**();**

**}**

// Обход в глубину (DFS)

public void DFS**(**int start**)**

**{**

bool**[]** visited **=** **new** bool**[**vertices**];**

Console**.**WriteLine**(**"Обход в глубину:"**);**

DFSUtil**(**start**,** visited**);**

Console**.**WriteLine**();**

**}**

private void DFSUtil**(**int v**,** bool**[]** visited**)**

**{**

visited**[**v**]** **=** **true;**

Console**.**Write**(**v **+** " "**);**

**for** **(**int i **=** 0**;** i **<** vertices**;** i**++)**

**if** **(**matrix**[**v**,** i**]** **!=** 0 **&&** **!**visited**[**i**])**

DFSUtil**(**i**,** visited**);**

**}**

// Алгоритм Дейкстры

public void Dijkstra**(**int start**)**

**{**

int**[]** dist **=** **new** int**[**vertices**];**

bool**[]** visited **=** **new** bool**[**vertices**];**

**for** **(**int i **=** 0**;** i **<** vertices**;** i**++)**

dist**[**i**]** **=** int**.**MaxValue**;**

dist**[**start**]** **=** 0**;**

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

**{**

int u **=** MinDistance**(**dist**,** visited**);**

visited**[**u**]** **=** **true;**

**for** **(**int v **=** 0**;** v **<** vertices**;** v**++)**

**if** **(!**visited**[**v**]** **&&** matrix**[**u**,** v**]** **!=** 0 **&&** dist**[**u**]** **!=** int**.**MaxValue **&&** dist**[**u**]** **+** matrix**[**u**,** v**]** **<** dist**[**v**])**

dist**[**v**]** **=** dist**[**u**]** **+** matrix**[**u**,** v**];**

**}**

Console**.**WriteLine**(**"Кратчайшие пути (Дейкстра):"**);**

**for** **(**int i **=** 0**;** i **<** vertices**;** i**++)**

Console**.**WriteLine**(**$"До вершины {i}: {dist[i]}"**);**

**}**

private int MinDistance**(**int**[]** dist**,** bool**[]** visited**)**

**{**

int min **=** int**.**MaxValue**,** minIndex **=** **-**1**;**

**for** **(**int v **=** 0**;** v **<** vertices**;** v**++)**

**if** **(!**visited**[**v**]** **&&** dist**[**v**]** **<=** min**)**

**{**

min **=** dist**[**v**];**

minIndex **=** v**;**

**}**

**return** minIndex**;**

**}**

// Алгоритм Краскала

public void Kruskal**()**

**{**

int**[]** parent **=** **new** int**[**vertices**];**

**for** **(**int i **=** 0**;** i **<** vertices**;** i**++)**

parent**[**i**]** **=** i**;**

List**<**int**[]>** edges **=** **new** List**<**int**[]>();** // [u, v, w]

**for** **(**int i **=** 0**;** i **<** vertices**;** i**++)**

**for** **(**int j **=** i **+** 1**;** j **<** vertices**;** j**++)**

**if** **(**matrix**[**i**,** j**]** **!=** 0**)**

edges**.**Add**(new** int**[]** **{** i**,** j**,** matrix**[**i**,** j**]** **});**

edges**.**Sort**((**a**,** b**)** **=>** a**[**2**].**CompareTo**(**b**[**2**]));** // Сортировка по весу

Console**.**WriteLine**(**"Минимальное остовное дерево (Краскал):"**);**

foreach **(**var edge in edges**)**

**{**

int u **=** Find**(**parent**,** edge**[**0**]);**

int v **=** Find**(**parent**,** edge**[**1**]);**

**if** **(**u **!=** v**)**

**{**

Console**.**WriteLine**(**$"Ребро {edge[0]}-{edge[1]}: вес {edge[2]}"**);**

Union**(**parent**,** u**,** v**);**

**}**

**}**

**}**

private int Find**(**int**[]** parent**,** int i**)**

**{**

**if** **(**parent**[**i**]** **!=** i**)**

parent**[**i**]** **=** Find**(**parent**,** parent**[**i**]);**

**return** parent**[**i**];**

**}**

private void Union**(**int**[]** parent**,** int u**,** int v**)**

**{**

parent**[**Find**(**parent**,** u**)]** **=** Find**(**parent**,** v**);**

**}**

**}**

class Program

**{**

static void Main**()**

**{**

Graph graph **=** null**;**

**while** **(true)**

**{**

Console**.**WriteLine**(**"\n1. Загрузить граф из файла"**);**

Console**.**WriteLine**(**"2. Сохранить граф в файл"**);**

Console**.**WriteLine**(**"3. Показать граф"**);**

Console**.**WriteLine**(**"4. Добавить вершину"**);**

Console**.**WriteLine**(**"5. Добавить ребро"**);**

Console**.**WriteLine**(**"6. Удалить вершину"**);**

Console**.**WriteLine**(**"7. Удалить ребро"**);**

Console**.**WriteLine**(**"8. Обход в ширину (BFS)"**);**

Console**.**WriteLine**(**"9. Обход в глубину (DFS)"**);**

Console**.**WriteLine**(**"10. Алгоритм Дейкстры"**);**

Console**.**WriteLine**(**"11. Алгоритм Краскала"**);**

Console**.**WriteLine**(**"0. Выход"**);**

Console**.**Write**(**"Выберите действие: "**);**

int choice **=** int**.**Parse**(**Console**.**ReadLine**());**

**if** **(**choice **==** 0**)** **break;**

**switch** **(**choice**)**

**{**

**case** 1**:**

Console**.**Write**(**"Введите имя файла: "**);**

string filename **=** Console**.**ReadLine**();**

graph **=** **new** Graph**(**0**);**

graph**.**LoadFromFile**(**filename**);**

**break;**

**case** 2**:**

**if** **(**graph **!=** null**)**

**{**

Console**.**Write**(**"Введите имя файла для сохранения: "**);**

graph**.**SaveToFile**(**Console**.**ReadLine**());**

**}**

**else**

Console**.**WriteLine**(**"Граф не загружен!"**);**

**break;**

**case** 3**:**

**if** **(**graph **!=** null**)** graph**.**Display**();**

**else** Console**.**WriteLine**(**"Граф не загружен!"**);**

**break;**

**case** 4**:**

**if** **(**graph **!=** null**)**

**{**

Console**.**Write**(**"Введите координаты (x y): "**);**

var coords **=** Console**.**ReadLine**().**Split**();**

int x **=** int**.**Parse**(**coords**[**0**]),** y **=** int**.**Parse**(**coords**[**1**]);**

graph**.**AddVertex**(**x**,** y**);**

**}**

**else**

Console**.**WriteLine**(**"Граф не загружен!"**);**

**break;**

**case** 5**:**

**if** **(**graph **!=** null**)**

**{**

Console**.**Write**(**"Введите вершины и вес (u v w): "**);**

var edge **=** Console**.**ReadLine**().**Split**();**

int u **=** int**.**Parse**(**edge**[**0**]),** v **=** int**.**Parse**(**edge**[**1**]),** w **=** int**.**Parse**(**edge**[**2**]);**

graph**.**AddEdge**(**u**,** v**,** w**);**

**}**

**else**

Console**.**WriteLine**(**"Граф не загружен!"**);**

**break;**

**case** 6**:**

**if** **(**graph **!=** null**)**

**{**

Console**.**Write**(**"Введите вершину для удаления: "**);**

int v **=** int**.**Parse**(**Console**.**ReadLine**());**

graph**.**RemoveVertex**(**v**);**

**}**

**else**

Console**.**WriteLine**(**"Граф не загружен!"**);**

**break;**

**case** 7**:**

**if** **(**graph **!=** null**)**

**{**

Console**.**Write**(**"Введите ребро для удаления (u v): "**);**

var edge **=** Console**.**ReadLine**().**Split**();**

int u **=** int**.**Parse**(**edge**[**0**]),** v **=** int**.**Parse**(**edge**[**1**]);**

graph**.**RemoveEdge**(**u**,** v**);**

**}**

**else**

Console**.**WriteLine**(**"Граф не загружен!"**);**

**break;**

**case** 8**:**

**if** **(**graph **!=** null**)**

**{**

Console**.**Write**(**"Введите начальную вершину: "**);**

int start **=** int**.**Parse**(**Console**.**ReadLine**());**

graph**.**BFS**(**start**);**

**}**

**else**

Console**.**WriteLine**(**"Граф не загружен!"**);**

**break;**

**case** 9**:**

**if** **(**graph **!=** null**)**

**{**

Console**.**Write**(**"Введите начальную вершину: "**);**

int start **=** int**.**Parse**(**Console**.**ReadLine**());**

graph**.**DFS**(**start**);**

**}**

**else**

Console**.**WriteLine**(**"Граф не загружен!"**);**

**break;**

**case** 10**:**

**if** **(**graph **!=** null**)**

**{**

Console**.**Write**(**"Введите начальную вершину: "**);**

int start **=** int**.**Parse**(**Console**.**ReadLine**());**

graph**.**Dijkstra**(**start**);**

**}**

**else**

Console**.**WriteLine**(**"Граф не загружен!"**);**

**break;**

**case** 11**:**

**if** **(**graph **!=** null**)**

graph**.**Kruskal**();**

**else**

Console**.**WriteLine**(**"Граф не загружен!"**);**

**break;** **}** **}** **} }**

**Результат:**

****

****

****

****

****

****

****

****

****

****

****

****

****

****

****

**Задание 4:**

**«**Разработайте приложение на языке C# в соответствии со своим вариантом задания.»

**Код:**

**using** System**;**

**using** System**.**IO**;**

class Program

**{**

static void Main**(**string**[]** args**)**

**{**

string inputFile **=** "input.txt"**;**

string outputFile **=** "output.txt"**;**

// Проверка существования файла

**if** **(!**File**.**Exists**(**inputFile**))**

**{**

Console**.**WriteLine**(**"Файл 'input.txt' не найден!"**);**

Console**.**WriteLine**(**"Создайте файл 'input.txt' в папке с программой."**);**

Console**.**WriteLine**(**"Файл должен содержать матрицу смежности по шаблону:"**);**

Console**.**WriteLine**(**"Пример:"**);**

Console**.**WriteLine**(**"3"**);**

Console**.**WriteLine**(**"0 1 1"**);**

Console**.**WriteLine**(**"1 0 1"**);**

Console**.**WriteLine**(**"1 1 0"**);**

Console**.**WriteLine**();**

Console**.**WriteLine**(**"Нажмите любую клавишу для выхода..."**);**

Console**.**ReadKey**();**

**return;**

**}**

**try** **{**

// Считываем строки из файла input.txt

string**[]** lines **=** File**.**ReadAllLines**(**inputFile**);**

// Первая строка — количество вершин

int n **=** int**.**Parse**(**lines**[**0**]);**

// Создаём матрицу смежности

int**[,]** matrix **=** **new** int**[**n**,** n**];**

// Заполняем матрицу из файла

**for** **(**int i **=** 0**;** i **<** n**;** i**++)**

**{**

string**[]** row **=** lines**[**i **+** 1**].**Split**(**' '**);**

**for** **(**int j **=** 0**;** j **<** n**;** j**++)**

**{**

matrix**[**i**,** j**]** **=** int**.**Parse**(**row**[**j**]);**

**}**

**}**

// Проверка на симметричность (матрица неориентированного графа)

bool isUndirected **=** **true;**

**for** **(**int i **=** 0**;** i **<** n **&&** isUndirected**;** i**++)**

**{**

**for** **(**int j **=** 0**;** j **<** n**;** j**++)** **{**

**if** **(**matrix**[**i**,** j**]** **!=** matrix**[**j**,** i**]** **||** **(**i **==** j **&&** matrix**[**i**,** j**]** **!=** 0**))**

**{** isUndirected **=** **false;** **break;** **}** **}** **}**

// Записываем результат в output.txt

**using** **(**StreamWriter writer **=** **new** StreamWriter**(**outputFile**))**

**{**

writer**.**WriteLine**(**isUndirected **?** "YES" **:** "NO"**);**

**if** **(**isUndirected**)**

**{**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)**

**{**

int degree **=** 0**;**

**for** **(**int j **=** 0**;** j **<** n**;** j**++)**

**{**

degree **+=** matrix**[**i**,** j**];**

**}**

writer**.**Write**(**degree **+** " "**);**

**}**

**}**

**}**

Console**.**WriteLine**(**"Проверка завершена. Результат записан в 'output.txt'."**);**

**}**

**catch** **(**Exception ex**)**

**{**

// Выводим сообщение об ошибке, если данные в файле некорректны

Console**.**WriteLine**(**"Ошибка при обработке файла:"**);**

Console**.**WriteLine**(**ex**.**Message**);**

**}**

Console**.**WriteLine**();**

Console**.**WriteLine**(**"Нажмите любую клавишу для выхода..."**);**

Console**.**ReadKey**();**

**}**

**}**

**Результат:**

****