МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

НЕЙРОННЫЕ СЕТИ

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

студента 5 курса 531 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Алексеева Александра Александровича

Преподаватель
ассистент _____ И. И. Слеповичев
подпись, дата

Саратов 2023

1. Создание ориентированного графа

На входе: текстовый файл с описанием графа в виде списка дуг:

$$(a_1, b_1, n_1), (a_2, b_2, n_2), \dots, (a_k, b_k, n_k)$$

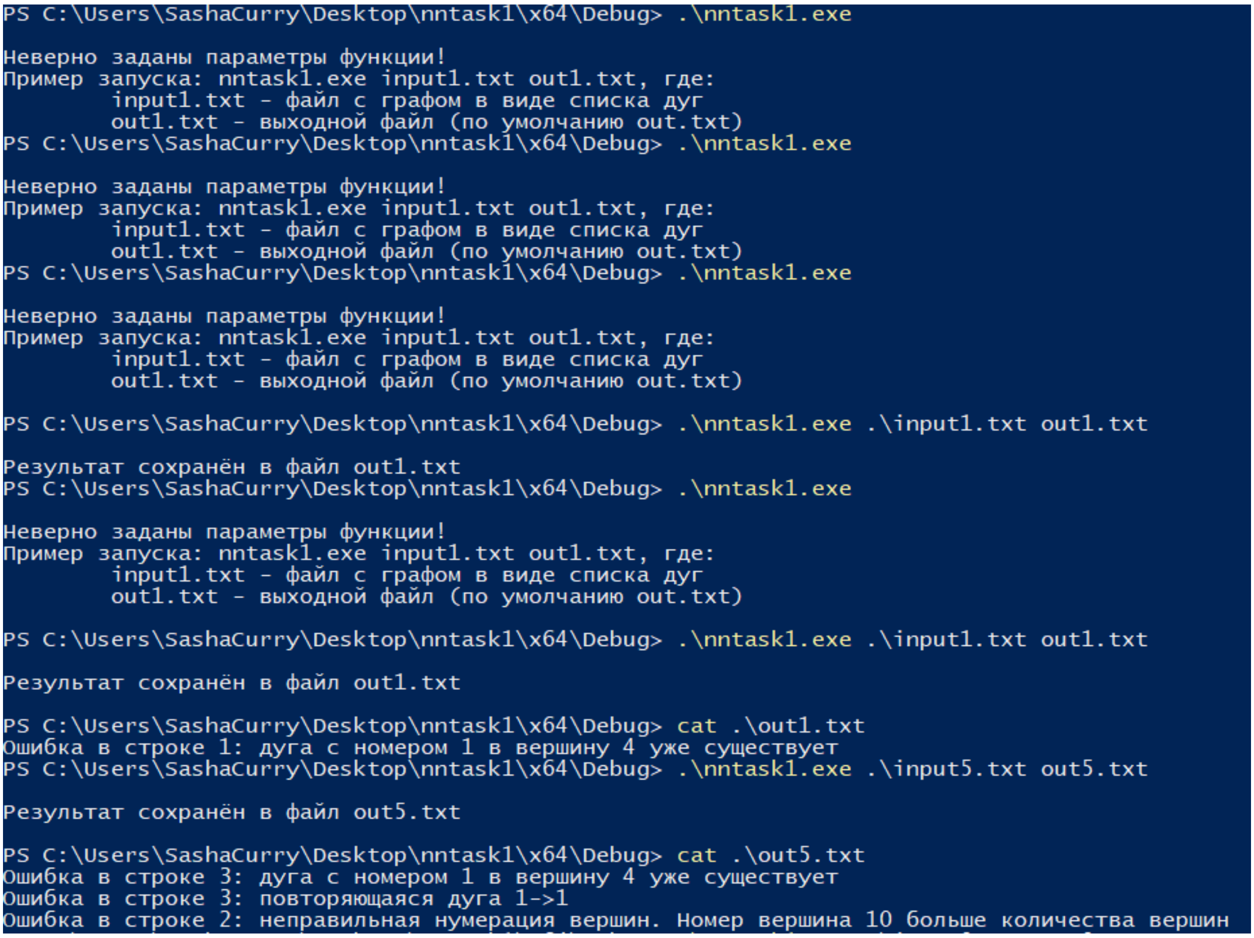
где a_i – начальная вершина дуги i, b_i – конечная вершина дуги i, n_i – порядковый номер дуги в списке всех заходящих в вершину b_i дуг.

На выходе:

а) Ориентированный граф с именованным вершинами и линейно упорядоченными дугами (в соответствии с порядком из текстового файла).

б) Сообщение об ошибке, если ошибка присутствует.

Пример работы программы



```
PS C:\Users\SashaCurry\Desktop\nntask1\x64\Debug> .\nntask1.exe

Неверно заданы параметры функции!
Пример запуска: nntask1.exe input1.txt out1.txt, где:
        input1.txt - файл с графом в виде списка дуг
        out1.txt - выходной файл (по умолчанию out.txt)
PS C:\Users\SashaCurry\Desktop\nntask1\x64\Debug> .\nntask1.exe

Неверно заданы параметры функции!
Пример запуска: nntask1.exe input1.txt out1.txt, где:
        input1.txt - файл с графом в виде списка дуг
        out1.txt - выходной файл (по умолчанию out.txt)
PS C:\Users\SashaCurry\Desktop\nntask1\x64\Debug> .\nntask1.exe

Неверно заданы параметры функции!
Пример запуска: nntask1.exe input1.txt out1.txt, где:
        input1.txt - файл с графом в виде списка дуг
        out1.txt - выходной файл (по умолчанию out.txt)

PS C:\Users\SashaCurry\Desktop\nntask1\x64\Debug> .\nntask1.exe .\input1.txt out1.txt

Результат сохранён в файл out1.txt
PS C:\Users\SashaCurry\Desktop\nntask1\x64\Debug> .\nntask1.exe

Неверно заданы параметры функции!
Пример запуска: nntask1.exe input1.txt out1.txt, где:
        input1.txt - файл с графом в виде списка дуг
        out1.txt - выходной файл (по умолчанию out.txt)

PS C:\Users\SashaCurry\Desktop\nntask1\x64\Debug> .\nntask1.exe .\input1.txt out1.txt

Результат сохранён в файл out1.txt

PS C:\Users\SashaCurry\Desktop\nntask1\x64\Debug> cat .\out1.txt
Ошибка в строке 1: дуга с номером 1 в вершину 4 уже существует
PS C:\Users\SashaCurry\Desktop\nntask1\x64\Debug> .\nntask1.exe .\input5.txt out5.txt

Результат сохранён в файл out5.txt

PS C:\Users\SashaCurry\Desktop\nntask1\x64\Debug> cat .\out5.txt
Ошибка в строке 3: дуга с номером 1 в вершину 4 уже существует
Ошибка в строке 3: повторяющаяся дуга 1->1
Ошибка в строке 2: неправильная нумерация вершин. Номер вершина 10 больше количества вершин
```

```
PS C:\Users\SashaCurry\Desktop\nntask1\x64\Debug> .\nntask1.exe .\input9.txt out9.txt

Результат сохранён в файл out9.txt

PS C:\Users\SashaCurry\Desktop\nntask1\x64\Debug> cat .\out9.txt
<graph>
        <vertex>v1</vertex>
        <vertex>v2</vertex>
        <vertex>v3</vertex>
        <vertex>v4</vertex>
        <vertex>v5</vertex>
        <vertex>v6</vertex>
        <vertex>v7</vertex>
        <vertex>v8</vertex>
        <vertex>v9</vertex>
        <vertex>v10</vertex>
        <vertex>v11</vertex>
        <arc>
                <from>v1</from>
                <to>v2</to>
                <order>1</order>
        </arc>
        <arc>
                <from>v2</from>
                <to>v3</to>
                <order>1</order>
        </arc>
        <arc>
                <from>v3</from>
                <to>v4</to>
                <order>1</order>
        </arc>
        <arc>
                <from>v4</from>
                <to>v5</to>
                <order>1</order>
        </arc>
        <arc>
                <from>v5</from>
                <to>v6</to>
                <order>1</order>
        </arc>
        <arc>
                <from>v6</from>
```

2. Создание функции по графу

На входе: ориентированный граф с именованными вершинами как описано в задании 1.

На выходе: линейное представление функции, реализуемой графом в префиксной скобочной записи:

$$A_1(B_1(C_1(\ldots), \ldots, C_m(\ldots)), \ldots, B_n(\ldots)).$$

Способ проверки результата:

а) выгрузка в текстовый файл результата преобразования графа в имя функции;

б) сообщение о наличии циклов в графе, если они присутствуют.

Пример работы программы

```
PS C:\Users\SashaCurry\Desktop\nntask2\x64\Debug> .\nntask2.exe

Неверно заданы параметры функции!
Пример запуска: NN_3.exe input1.txt out1.txt, где:
        input1.txt - ориентированный граф с именованными вершинами
        out1.txt - выходной файл (по умолчанию out.txt)
PS C:\Users\SashaCurry\Desktop\nntask2\x64\Debug> .\nntask2.exe .\input1.txt out1.txt

Данные успешно сохранены в out1.txt

PS C:\Users\SashaCurry\Desktop\nntask2\x64\Debug> cat .\out1.txt
7(5(4(2(1),3(1))),6(4(2(1),3(1))))
PS C:\Users\SashaCurry\Desktop\nntask2\x64\Debug> .\nntask2.exe .\input4.txt out4.txt

Ошибка в строке 4: вершины 0 быть не может
Ошибка в строке 4: дуга с номером 1 в вершину 4 уже существует
Ошибка в строке 1: в графе существует цикл
Ошибка в строке 2: в графе существует цикл
PS C:\Users\SashaCurry\Desktop\nntask2\x64\Debug> .\nntask2.exe .\input9.txt out9.txt

Данные успешно сохранены в out9.txt

PS C:\Users\SashaCurry\Desktop\nntask2\x64\Debug> cat .\out9.txt
7(5(4(2(1),3(1))),6(4(2(1),3(1))))
PS C:\Users\SashaCurry\Desktop\nntask2\x64\Debug>
```

3. Вычисление значения функции на графе

На входе:

а) Текстовый файл с описанием графа в виде списка дуг (из задания 1);

б) Текстовый файл соответствий арифметических операций именами вершин:

a_1: операция$_1$

a_2: операция$_2$

…

a_n: операция$_n$

где a_i – имя i-й вершина, операция$_i$ – символ операции, соответствующий вершине a_i.

Допустимы следующие символы операций:

+ – сумма значений,

***** – произведение значений,

exp – экспонирование входного значения,

число – любая числовая константа.

На выходе: значение функции, построенной по графу а) и файлу б).

Пример работы программы

```
PS C:\Users\SashaCurry\Desktop\nntask3\x64\Debug> .\nntask3.exe

Неверно заданы параметры функции!
Пример запуска: NN_3.exe input1.txt input1cmd.txt out1.txt, где:
        input1.txt - файл с графом
        input1cmd.txt - файл с командами
        out1.txt - выходной файл (по умолчанию out.txt)

PS C:\Users\SashaCurry\Desktop\nntask3\x64\Debug> .\nntask3.exe .\input1.txt .\input1cmd.txt out1.txt

Данные успешно записаны в файл out1.txt

PS C:\Users\SashaCurry\Desktop\nntask3\x64\Debug> cat .\out1.txt
Функция: 6(4(3(1,2)),5(3(1,2)))
Значение функции: 53,93
PS C:\Users\SashaCurry\Desktop\nntask3\x64\Debug> .\nntask3.exe .\input2.txt .\input2cmd.txt out2.txt

Данные успешно записаны в файл out2.txt

PS C:\Users\SashaCurry\Desktop\nntask3\x64\Debug> cat .\out2.txt
Функция: 7(5(3(1,2),4),6)
Значение функции: 4500000,00
PS C:\Users\SashaCurry\Desktop\nntask3\x64\Debug> .\nntask3.exe .\input4.txt .\input4cmd.txt out4.txt

Данные успешно записаны в файл out4.txt

PS C:\Users\SashaCurry\Desktop\nntask3\x64\Debug> cat .\out4.txt
Функция: 5(4(3(2(1))))
Значение функции: 490,19
PS C:\Users\SashaCurry\Desktop\nntask3\x64\Debug>
```

4. Построение многослойной нейронной сети

На входе:

а) Текстовый файл с набором матриц весов межнейронных связей:

M1 : [M1[1,1], M1[1,2],..., M1[1,n]], ..., [M1[m,1], M1[m,2],...,M1[m,n]]
M2 : [M2[1,1], M2[1,2],..., M2[1,n]], ..., [M2[m,1], M2[m,2],...,M2[m,n]]
...
Mp : [Mp[1,1], Mp[1,2],..., Mp[1,n]], ..., [Mp[m,1], Mp[m,2],...,Mp[m,n]]

б) Текстовый файл с входным вектором в формате:

$$x_1, x_2, \ldots, x_n.$$

На выходе:

а) Сериализованная многослойная нейронная сеть (в формате XML или JSON) с полносвязной межслойной структурой.

Файл с выходным вектором – результатом вычислений НС в формате:

$$y_1, y_2, \ldots, y_n.$$

б) Сообщение об ошибке, если в формате входного вектора или файла описания НС допущена ошибка.

Пример работы программы

```
PS C:\Users\SashaCurry\Desktop\nntask4\x64\Debug> .\nntask4.exe

Неверно заданы параметры функции!
Пример запуска: NN_4.exe matrix1.txt vector1.txt outNetwork1.txt outValue1.txt, где:
        matrix1.txt - файл с матрицей
        vector1.txt - файл с вектором
        outNetwork1.txt - многослойная нейронная сеть в формате XML (по умолчанию outNetwork.txt)
        outValue1.txt - результат вычислений нейронной сети (по умолчанию outValue.txt)

PS C:\Users\SashaCurry\Desktop\nntask4\x64\Debug> .\nntask4.exe .\matrix1.txt .\vector1.txt .\outNetwork1.txt .\outValue
1.txt

Многослойная нейронная сеть успешно записана в файл .\outNetwork1.txt
Результат вычислений НС успешно записан в файл .\outValue1.txt
PS C:\Users\SashaCurry\Desktop\nntask4\x64\Debug> cat .\outValue1.txt
0.983198 0.985357 0.987025
PS C:\Users\SashaCurry\Desktop\nntask4\x64\Debug> cat .\outNetwork1.txt
<network>
        <layer>
                <connections>
                        <weight>1</weight>
                        <weight>2</weight>
                        <weight>3</weight>
                </connections>
                <connections>
                        <weight>4</weight>
                        <weight>5</weight>
                        <weight>6</weight>
                </connections>
                <connections>
                        <weight>7</weight>
                        <weight>8</weight>
                        <weight>9</weight>
                </connections>
        </layer>
        <layer>
                <connections>
                        <weight>10</weight>
                        <weight>11</weight>
                        <weight>12</weight>
                </connections>
                <connections>
                        <weight>13</weight>
```

```
PS C:\Users\SashaCurry\Desktop\nntask4\x64\Debug> .\nntask4.exe .\matrix2.txt .\vector2.txt .\outNetwork2.txt .\outValue
2.txt

Многослойная нейронная сеть успешно записана в файл .\outNetwork2.txt
Результат вычислений НС успешно записан в файл .\outValue2.txt
PS C:\Users\SashaCurry\Desktop\nntask4\x64\Debug> cat .\outValue2.txt
0.985317
PS C:\Users\SashaCurry\Desktop\nntask4\x64\Debug> cat .\outNetwork2.txt
<network>
        <layer>
                <connections>
                        <weight>25</weight>
                        <weight>75</weight>
                </connections>
                <connections>
                        <weight>1</weight>
                        <weight>33</weight>
                </connections>
        </layer>
        <layer>
                <connections>
                        <weight>12</weight>
                        <weight>56</weight>
                </connections>
        </layer>
</network>
```

```
PS C:\Users\SashaCurry\Desktop\nntask4\x64\Debug> .\nntask4.exe .\matrix4.txt .\vector4.txt .\outNetwork4.txt outValue4.
txt

Многослойная нейронная сеть успешно записана в файл .\outNetwork4.txt
Результат вычислений НС успешно записан в файл outValue4.txt
PS C:\Users\SashaCurry\Desktop\nntask4\x64\Debug> cat .\outValue4.txt
0.979465 0.988944 0.993107 0.99742
PS C:\Users\SashaCurry\Desktop\nntask4\x64\Debug> cat .\outNetwork4.txt
<network>
        <layer>
                <connections>
                        <weight>2</weight>
                        <weight>75</weight>
                        <weight>23</weight>
                        <weight>42</weight>
                </connections>
                <connections>
                        <weight>12</weight>
                        <weight>23</weight>
                        <weight>32</weight>
                        <weight>33</weight>
                </connections>
                <connections>
                        <weight>122</weight>
                        <weight>56</weight>
                        <weight>12</weight>
                        <weight>84</weight>
                </connections>
                <connections>
                        <weight>12</weight>
                        <weight>31</weight>
                        <weight>2</weight>
                        <weight>55</weight>
                </connections>
        </layer>
        <layer>
                <connections>
                        <weight>2</weight>
                        <weight>10</weight>
                        <weight>13</weight>
                        <weight>4</weight>
                </connections>
                <connections>
```

5. Реализация метода обратного распространения ошибки для многослойной НС

На входе:

а) Текстовый файл с описанием НС (формат см. в задании 4).

б) Текстовый файл с обучающей выборкой:

$$[x_{11}, x_{12}, \ldots, x_{1n}] \rightarrow [y_{11}, y_{12}, \ldots, y_{1m}]$$

$$\ldots$$

$$[x_{k1}, x_{k2}, \ldots, x_{kn}] \rightarrow [y_{k1}, y_{k2}, \ldots, y_{km}]$$

Формат описания входного вектора x и выходного вектора y соответствует формату из задания 4.

в) Число итераций обучения (в строке параметров).

На выходе:

Текстовый файл с историей N итераций обучения методом обратного распространения ошибки:

$$1: \text{Ошибка}_1$$

$$2: \text{Ошибка}_2$$

$$\ldots$$

$$N: \text{Ошибка}_N$$

Пример работы программы

```
PS C:\Users\SashaCurry\Desktop\nntask5\x64\Debug> .\nntask5.exe

Неверно заданы параметры функции!
Пример запуска: NN_4.exe matrix1.json training1.txt 20 outRes1.txt, где:
        matrix1.txt - файл с описание нейронной сети
        training.txt - файл с обучающей выборкой
        20 - число итераций обучения
        outRes1.txt - файл с историей 20 итераций обучения (по умолчанию outValue.txt)

PS C:\Users\SashaCurry\Desktop\nntask5\x64\Debug> .\nntask5.exe .\matrix1.json .\training1.json 30 outValue1.txt

Результат успешно записан в файл outValue1.txt


PS C:\Users\SashaCurry\Desktop\nntask5\x64\Debug> cat .\outValue1.txt
Итерация 1: [0.0, 0.04601906620676518, 0.0902596596880356, 0.13279836302196324, 0.17370788746129157, 0.21305730011355678, 0.2509122359
062895, 0.287335095482364, 0.322385230075034, 0.4309041283336664]
Итерация 2: [1.139301881463638E-5, 0.023139583658110988, 0.04453778027217453, 0.06434892087355124, 0.08270219631042547, 0.099714572413
86049, 0.11549212134943922, 0.1301311888373507, 0.1437194198031962, 0.2250038987242403]
Итерация 3: [1.1720946353361596E-5, 0.01149744310687397 8, 0.021734764714270858, 0.030872360640948858, 0.03903845437888449, 0.046344032
91351679, 0.052885493555268615, 0.058746833868962754, 0.06400147206255062, 0.11599094791970145]
Итерация 4: [6.968182315059297E-6, 0.004834650678569379, 0.008984029079194197, 0.012558732722306397, 0.015644235067440747, 0.018311464
109986336, 0.020619610966132677, 0.022618332620635943, 0.024349492743838987, 0.05057410730795944]
Итерация 5: [1.1849039960458 45E-6, 6.95764348047973E-4, 0.0012719892265711642, 0.0017516305804972632, 0.0021518977400495215, 0.0024865
16424456771, 0.002766523940811527, 0.003000862740025325, 0.003196828341625257, 0.007544060626435457]
Итерация 6: [-3.6149830151432134E-6, -0.0020345796675078484, -0.0036623525108527653, -0.004927250325780218, -0.006030914706811199, -0.
006886897903314106, -0.007579582012029399, -0.008139473142562553, -0.008590722670430866, -0.022861430792118956]
Итерация 7: [-6.443990745808007E-6, -0.003919718542250888, -0.006952459667401737, -0.009315712369225 78, -0.0111636030811709 4, -0.01261
046188649871, -0.013742284234128464, -0.014624561269235262, -0.015307727929154549, -0.04563886038906138]
Итерация 8: [-6.791619858113513E-6, -0.005268149398762694, -0.009214229893914514, -0.012193111160588819, -0.014449791120376702, -0.0161
60066544671184, -0.017452230155529 64, -0.018421316030245 07, -0.019138661305576853, -0.06356430352755536]
Итерация 9: [-4.366073732276837E-6, -0.006259978011516268, -0.010804293290387125, -0.014130000453987535, -0.01657188427925056, -0.0183
62730659598214, -0.019667667564975993, -0.020605890779776315, -0.02126476924563139, -0.07828660565091337]
Итерация 10: [1.0262256086895102E-6, -0.007005976315530074, -0.011940074100996633, -0.015443064458058209, -0.01793662629626783, -0.019
7053564691027, -0.02094581003884543, -0.021796467302815294, -0.022356597517453603, -0.0907696960914384]
Итерация 11: [-4.884932675336205E-5, 0.045960060719337816, 0.09020224163938556, 0.13275040373 54823, 0.17367463256097834, 0.21304218584
896525, 0.2509174364024936, 0.2873618968693736 3, 0.3224342904958679 7, 0.4310092804922415]
Итерация 12: [-1.3615793558247577E-5, 0.023119238757473175, 0.04452748058877112, 0.0643515537938257, 0.0827191258354708 2, 0.0997462310
1651073, 0.11553837724081313, 0.1301915766750656 7, 0.143793286666 1864, 0.22517428752859794]
Итерация 13: [-9.168290088909342E-7, 0.011492576049519483, 0.021739679599028516, 0.030887654866447272, 0.03906396928637336, 0.04637922
3674058966, 0.05292963818370862, 0.058799150637620326, 0.06406118409180155, 0.11618719446637998]
Итерация 14: [1.5672521305325624E-6, 0.004836605836553432, 0.008993510958381676, 0.012575333982948 94, 0.01566730771147 1592, 0.01834028
9946888074, 0.020653485335144314, 0.022656602673157233, 0.024391572019873728, 0.05072600799236879]
Итерация 15: [3.9756461387634963E-7, 7.007987350230199E-4, 0.0012820280862291648, 0.001765919636880668, 0.002169772429390907 7, 0.00250
```

```
PS C:\Users\SashaCurry\Desktop\nntask5\x64\Debug> .\nntask5.exe .\matrix2.json .\training2.json 20 outValue2.txt

Результат успешно записан в файл outValue2.txt


PS C:\Users\SashaCurry\Desktop\nntask5\x64\Debug> cat .\outValue2.txt
Итерация 1: [0.0, 0.0]
Итерация 2: [0.0, 0.0]
Итерация 3: [0.0, 0.0]
Итерация 4: [0.0, 0.0]
Итерация 5: [0.0, 0.0]
Итерация 6: [0.0, 0.0]
Итерация 7: [0.0, 0.0]
Итерация 8: [0.0, 0.0]
Итерация 9: [0.0, 0.0]
Итерация 10: [0.0, 0.0]
Итерация 11: [0.0, 0.0]
Итерация 12: [0.0, 0.0]
Итерация 13: [0.0, 0.0]
Итерация 14: [0.0, 0.0]
Итерация 15: [0.0, 0.0]
Итерация 16: [0.0, 0.0]
Итерация 17: [0.0, 0.0]
Итерация 18: [0.0, 0.0]
Итерация 19: [0.0, 0.0]
Итерация 20: [0.0, 0.0]
```

```
PS C:\Users\SashaCurry\Desktop\nntask5\x64\Debug> .\nntask5.exe

Неверно заданы параметры функции!
Пример запуска: NN_4.exe matrix1.json training1.txt 20 outRes1.txt, где:
        matrix1.txt - файл с описание нейронной сети
        training.txt - файл с обучающей выборкой
        20 - число итераций обучения
        outRes1.txt - файл с историей 20 итераций обучения (по умолчанию outValue.txt)

PS C:\Users\SashaCurry\Desktop\nntask5\x64\Debug> .\nntask5.exe .\matrix4.json .\training4.json 10 outValue4.txt

Файл .\matrix4.json не является файлом JSON-формата1
```

ПРИЛОЖЕНИЕ А

Листинг кода для задания 1

```
#include "iostream"
#include "cmath"
#include "vector"
#include "string"
#include "set"
#include "fstream"

using namespace std;

class Edge {
public:
	int from = -1;
	short rawFrom = 0;
	int to = -1;
	short rawTo = 0;
	short order = -1;
	Edge() {
	}
	Edge(int from, short rawFrom, int to = -1, short rawTo = 0, short
order = -1) {
		this->from = from;
		this->rawFrom = rawFrom;
		this->to = to;
		this->rawTo = rawTo;
		this->order = order;
	}
	~Edge() {//It's destructor
	}
};



class Graph {
private:
	bool checkStrDigit(string str, short raw) {
		if (str.empty()) {
			this->errors.push_back(string{ "Ошибка в строке " +
to_string(raw) + ": некорретные данные (введены символы вместо цифр)" });
			return false;
		}
		for (int i = 0; i < str.length(); i++)
			if (!isdigit(str[i])) {
				this->errors.push_back(string{ "Ошибка в строке
" + to_string(raw) + ": некорретные данные (введены символы вместо цифр)" });
				return false;
			}
		return true;
	}


	void checkSameEdge(int from, int to, short raw) {
		for (short i = 0; i < edges.size(); i++)
			if (edges[i].from == from && edges[i].to == to)
				this->errors.push_back(string{ "Ошибка в строке
" + to_string(raw) + ": повторяющаяся дуга " +

		to_string(from) + "->" + to_string(to) });
	}
```

```
        void checkSameNumEdge(int to, int order, short raw) {
                for (short i = 0; i < edges.size(); i++)
                        if (edges[i].to == to && edges[i].order == order)
                                this->errors.push_back(string{ "Ошибка в строке
" + to_string(raw) + ": дуга с номером " + to_string(order) +

           " в вершину " + to_string(to) + " уже существует" });
        }


        void checkNumVertex() {
                int vertexSize = vertexes.size();
                for (short i = 0; i < edges.size(); i++) {
                        if (edges[i].from > vertexSize)
                                this->errors.push_back(string{ "Ошибка в строке
" + to_string(edges[i].rawFrom) + ": неправильная нумерация вершин." +

           " Номер вершина " + to_string(edges[i].from) + " больше количества
вершин" });
                        if (edges[i].to > vertexSize)
                                this->errors.push_back(string{ "Ошибка в строке
" + to_string(edges[i].rawTo) + ": неправильная нумерация вершин." +

           " Номер вершина " + to_string(edges[i].to) + " больше количества
вершин" });
                }
        }


        void addNumEdges(int to, short order, int raw) {
                bool emptyVertex = true;
                for (short i = 0; i < numEdges.size(); i++)
                        if (numEdges[i].first == to) {
                                this-
>numEdges[i].second.insert(make_pair(order, raw));
                                emptyVertex = false;
                                break;
                        }
                if (emptyVertex) {
                        this->numEdges.push_back(make_pair(to, set <pair <int,
int>> {make_pair(order, raw)}));
                }
        }


        void checkNumEdges() {
                for (int i = 0; i < numEdges.size(); i++) {
                        int numPrev = 0;
                        for (auto j = numEdges[i].second.begin(); j !=
numEdges[i].second.end(); j++) {
                                pair <int, int> num = *j;
                                if (num.first - numPrev == 1 || num.first -
numPrev == 0)
                                        numPrev = num.first;
                                else {
                                        this->errors.push_back(string{ "Ошибка
в строке " + to_string(num.second) + ": неправильно заданы номера дуг" });
                                        numPrev = num.first;
                                }
                        }
                }
        }
```

```
        void getData() {
                ifstream fin(inFile);
                if (!fin.is_open())
                        throw string{ "Файл " + inFile + " не найден! \n" };

                string str = "";
                string partEdge = "";
                short raw = 1;
                for (char el, elPrev = ' '; fin.get(el); elPrev = el) {
                        if (el == '\n')
                                raw++;
                        else if (el == ' ' && elPrev == ',')
                                continue;
                        else if (el == '(' && partEdge == "") {
                                partEdge = "from";
                                str = "";
                        }
                        else if (el == ')' && !edges.empty() &&
edges.back().order != -1)
                                this->errors.push_back(string{ "Ошибка в строке
" + to_string(raw) + ": неправильно задана компонента. Формат: (a, b, n)" });
                        else if (partEdge != "" && (el == ',' || el == ')')) {
                                if (!checkStrDigit(str, raw))
                                        str = "-1";

                                if (partEdge == "from") {
                                        if (str == "0")
                                                this->errors.push_back(string{
"Ошибка в строке " + to_string(raw) + ": вершины 0 быть не может" });

                                        this->vertexes.insert(stoi(str));
                                        this->edges.push_back(Edge(stoi(str),
raw));
                                        partEdge = "to";
                                        str = "";
                                }
                                else if (partEdge == "to") {
                                        if (str == "0")
                                                this->errors.push_back(string{
"Ошибка в строке " + to_string(raw) + ": вершины 0 быть не может" });
                                        checkSameEdge(edges.back().from,
stoi(str), raw);

                                        this->vertexes.insert(stoi(str));
                                        this->edges.back().to = stoi(str);
                                        this->edges.back().rawTo = raw;
                                        partEdge = "order";
                                        str = "";
                                }
                                else if (partEdge == "order") {
                                        checkSameNumEdge(edges.back().to,
stoi(str), raw);

                                        this->edges.back().order = stoi(str);
                                        partEdge = "";
                                        str = "";

                                        addNumEdges(edges.back().to,
edges.back().order, raw);
                                }
                        }
                        else
                                str += el;
                }
```

```
                checkNumVertex();
                checkNumEdges();
                fin.close();
        }
public:
        string inFile;
        string outFile = "out.txt";
        set <int> vertexes;
        vector <Edge> edges;
        vector <string> errors;
        vector <pair <int, set <pair <int, int>>>> numEdges;

        Graph(string inFile, string outFile) {
                this->inFile = inFile;
                this->outFile = outFile;
                getData();
                printInFile();
        }
        ~Graph() {//It's destructor
        }


        void printData() {
                if (edges.empty())
                        cout << "\nФайл не содержить граф!";
                else {
                        cout << "\nВершины: ";
                        for (auto i = vertexes.begin(); i != vertexes.end();
i++)
                                cout << *i << " ";
                        cout << "\nДуги: ";
                        for (short i = 0; i < edges.size(); i++)
                                cout << "(" << edges[i].from << ", " <<
edges[i].to << ", " << edges[i].order << ") ";
                }
        }


        void printInFile() {
                ofstream fout(outFile);
                if (!fout.is_open())
                        throw string{ "Файл " + outFile + " не найден! \n" };

                if (!errors.empty())
                        for (short i = 0; i < errors.size(); i++)
                                fout << errors[i] << endl;
                else {
                        fout << "<graph>";
                        for (auto i = vertexes.begin(); i != vertexes.end();
i++)
                                fout << "\n\t<vertex>v" << *i << "</vertex>";
                        for (short i = 0; i < edges.size(); i++) {
                                fout << "\n\t<arc>";
                                fout << "\n\t\t<from>v" << edges[i].from <<
"</from>";
                                fout << "\n\t\t<to>v" << edges[i].to <<
"</to>";
                                fout << "\n\t\t<order>" << edges[i].order <<
"</order>";
                                fout << "\n\t</arc>";
                        }
                        fout << "\n</graph>";
                }
```

```
                fout.close();
                cout << "\nРезультат сохранён в файл " << outFile << endl;
        }
};



int main(int argc, char* argv[]) {
        setlocale(LC_ALL, "ru");

        if (argc != 2 && argc != 3) {
                cout << "\nНеверно заданы параметры функции!";
                cout << "\nПример запуска: nntask1.exe input1.txt out1.txt,
где:";
                cout << "\n\tinput1.txt - файл с графом в виде списка дуг
\n\tout1.txt - выходной файл (по умолчанию out.txt) \n\n";
                return 0;
        }

        string inFile = argv[1];
        string outFile = "out.txt";
        if (argc == 3)
                outFile = argv[2];

        try {
                Graph G(inFile, outFile);
        }
        catch (string& error) {
                cout << error;
        }

        cout << endl;
        return 0;
}
```

ПРИЛОЖЕНИЕ Б

Листинг кода для задания 2

```
#include "iostream"
#include "cmath"
#include "vector"
#include "string"
#include "set"
#include "fstream"
#include "algorithm"

using namespace std;

class Edge {
public:
	int from = -1;
	short rawFrom = 0;
	int to = -1;
	short rawTo = 0;
	short order = -1;

	Edge(int from, short rawFrom, int to = -1, short rawTo = 0, short
order = -1) {
		this->from = from;
		this->rawFrom = rawFrom;
		this->to = to;
		this->rawTo = rawTo;
		this->order = order;
	}
	~Edge() {//It's destructor
	}

	bool operator == (Edge& edge) const {
		return (this->from == edge.from && this->to == edge.to);
	}
	bool operator < (Edge& edge) const {
		if (this->from == edge.from)
			return this->to < edge.to;
		return this->from < edge.from;
	}
	bool operator > (Edge& edge) const {
		if (this->from == edge.from)
			return this->to > edge.to;
		return this->from > edge.from;
	}

	Edge reverse() {
		swap(from, to);
		return *this;
	}
};



class Graph {
protected:
	string inFile;
	string outFile;
	set <int> vertexes;
	vector <Edge> edges;
	set <string> errors;
	vector <pair <int, set <pair <int, int>>>> numEdges;
```

```
        bool checkStrDigit(string str, short raw) {
                if (str.empty()) {
                        this->errors.insert(string{ "Ошибка в строке " +
to_string(raw) + ": некорретные данные (введены символы вместо цифр)" });
                        return false;
                }
                for (int i = 0; i < str.length(); i++)
                        if (!isdigit(str[i])) {
                                this->errors.insert(string{ "Ошибка в строке "
+ to_string(raw) + ": некорретные данные (введены символы вместо цифр)" });
                                return false;
                        }
                return true;
        }


        void checkSameEdge(int from, int to, short raw) {
                for (short i = 0; i < edges.size(); i++)
                        if (edges[i].from == from && edges[i].to == to)
                                this->errors.insert(string{ "Ошибка в строке "
+ to_string(raw) + ": повторяющаяся дуга " +

           to_string(from) + "->" + to_string(to) });
        }


        void checkSameNumEdge(int to, int order, short raw) {
                for (short i = 0; i < edges.size(); i++)
                        if (edges[i].to == to && edges[i].order == order)
                                this->errors.insert(string{ "Ошибка в строке "
+ to_string(raw) + ": дуга с номером " + to_string(order) +

           " в вершину " + to_string(to) + " уже существует" });
        }


        void checkNumVertex() {
                int vertexSize = vertexes.size();
                for (short i = 0; i < edges.size(); i++) {
                        if (edges[i].from > vertexSize)
                                this->errors.insert(string{ "Ошибка в строке "
+ to_string(edges[i].rawFrom) + ": неправильная нумерация вершин." +

           " Номер вершина " + to_string(edges[i].from) + " больше количества
вершин" });
                        if (edges[i].to > vertexSize)
                                this->errors.insert(string{ "Ошибка в строке "
+ to_string(edges[i].rawTo) + ": неправильная нумерация вершин." +

           " Номер вершина " + to_string(edges[i].to) + " больше количества
вершин" });
                }
        }


        void addNumEdges(int to, short order, int raw) {
                bool emptyVertex = true;
                for (short i = 0; i < numEdges.size(); i++)
                        if (numEdges[i].first == to) {
                                this-
>numEdges[i].second.insert(make_pair(order, raw));
                                emptyVertex = false;
                                break;
                        }
```

```
                if (emptyVertex) {
                        this->numEdges.push_back(make_pair(to, set <pair <int,
int>> {make_pair(order, raw)}));
                }
        }


        void checkNumEdges() {
                for (int i = 0; i < numEdges.size(); i++) {
                        int numPrev = 0;
                        for (auto j = numEdges[i].second.begin(); j !=
numEdges[i].second.end(); j++) {
                                pair <int, int> num = *j;
                                if (num.first - numPrev == 1 || num.first -
numPrev == 0)
                                        numPrev = num.first;
                                else {
                                        this->errors.insert(string{ "Ошибка в
строке " + to_string(num.second) + ": неправильно заданы номера дуг" });
                                        numPrev = num.first;
                                }
                        }
                }
        }


        void getData() {
                ifstream fin(inFile);
                if (!fin.is_open())
                        throw string{ "Файл " + inFile + " не найден! \n" };

                string str = "";
                string partEdge = "";
                short raw = 1;
                for (char el, elPrev = ' '; fin.get(el); elPrev = el) {
                        if (el == '\n')
                                raw++;
                        else if (el == ' ' && elPrev == ',')
                                continue;
                        else if (el == '(' && partEdge == "") {
                                partEdge = "from";
                                str = "";
                        }
                        else if (el == ')' && !edges.empty() &&
edges.back().order != -1)
                                this->errors.insert(string{ "Ошибка в строке "
+ to_string(raw) + ": неправильно задана компонента. Формат: (a, b, n)" });
                        else if (partEdge != "" && (el == ',' || el == ')')) {
                                if (!checkStrDigit(str, raw))
                                        str = "-1";

                                if (partEdge == "from") {
                                        if (str == "0")
                                                this->errors.insert(string{
"Ошибка в строке " + to_string(raw) + ": вершины 0 быть не может" });

                                        this->vertexes.insert(stoi(str));
                                        this->edges.push_back(Edge(stoi(str),
raw));

                                        partEdge = "to";
                                        str = "";
                                }
                                else if (partEdge == "to") {
                                        if (str == "0")
```

```
                                                this->errors.insert(string{
"Ошибка в строке " + to_string(raw) + ": вершины 0 быть не может" });
                                        checkSameEdge(edges.back().from,
stoi(str), raw);

                                        this->vertexes.insert(stoi(str));
                                        this->edges.back().to = stoi(str);
                                        this->edges.back().rawTo = raw;
                                        partEdge = "order";
                                        str = "";
                                }
                                else if (partEdge == "order") {
                                        checkSameNumEdge(edges.back().to,
stoi(str), raw);

                                        this->edges.back().order = stoi(str);
                                        partEdge = "";
                                        str = "";

                                        addNumEdges(edges.back().to,
edges.back().order, raw);
                                }
                        }
                        else
                                str += el;
                }

                checkNumVertex();
                checkNumEdges();
                fin.close();
        }
public:
        Graph(string inFile, string outFile) {
                this->inFile = inFile;
                this->outFile = outFile;
                getData();

                if (edges.empty())
                        throw (string{ "\nФайл не содержит граф!" });
                else if (!errors.empty()) {
                        for (auto i = errors.begin(); i != errors.end(); i++)
                                cout << endl << *i;
                        errors.clear();
                }
        }
        ~Graph() {//It's destructor
        }


        void printGraph() {
                if (!errors.empty())
                        return;
                cout << "\nВершины: ";
                for (auto i = vertexes.begin(); i != vertexes.end(); i++)
                        cout << *i << " ";
                cout << "\nДуги: ";
                for (short i = 0; i < edges.size(); i++)
                        cout << "(" << edges[i].from << ", " << edges[i].to <<
", " << edges[i].order << ") ";
        }


        void printGraphInFile() {
                if (!errors.empty())
```

```
                return;

        ofstream fout(outFile);
        if (!fout.is_open())
                throw string{ "Файл " + outFile + " не найден! \n" };

        fout << "<graph>";
        for (auto i = vertexes.begin(); i != vertexes.end(); i++)
                fout << "\n\t<vertex>v" << *i << "</vertex>";
        for (short i = 0; i < edges.size(); i++) {
                fout << "\n\t<arc>";
                fout << "\n\t\t<from>v" << edges[i].from << "</from>";
                fout << "\n\t\t<to>v" << edges[i].to << "</to>";
                fout << "\n\t\t<order>" << edges[i].order <<
"</order>";
                fout << "\n\t</arc>";
        }
        fout << "\n</graph>";
        cout << "\nРезультат сохранён в файл " << outFile;
        fout.close();
    }
};



class GraphFunction : Graph {
protected:
    string graphFunction = "";

    void checkCycles(int curV, int* usedV) {
        usedV[curV - 1] = 1;
        for (short i = 0; i < edges.size(); i++)
            if (edges[i].from == curV) {
                if (usedV[edges[i].to - 1] == 1)
                    errors.insert(string{ "Ошибка в строке
" + to_string(edges[i].rawTo) + ": в графе существует цикл" });
                else {
                    checkCycles(edges[i].to, usedV);
                    usedV[edges[i].to - 1] = 0;
                }
            }
    }


    vector <Edge> sortEdges(vector <Edge> edges) {//edges - копия
переменной this->edges
        for (short i = 0; i < edges.size(); i++)
            edges[i].reverse();

        vector <Edge> res;
        while (!edges.empty()) {
            pair <Edge, short> minEdge = make_pair(edges[0], 0);
            for (short i = 1; i < edges.size(); i++)
                if (edges[i] < minEdge.first)
                    minEdge = make_pair(edges[i], i);
            res.push_back(minEdge.first);
            edges.erase(edges.begin() + minEdge.second);
        }

        reverse(res.begin(), res.end());
        for (short i = 0; i < res.size() - 1; i++)
            if (res[i].from == res[i + 1].from && res[i].to > res[i
+ 1].to)
                swap(res[i], res[i + 1]);
```

```
                return res;
        }


        int findToAndDeleteEdge(vector <Edge>& edges, int vFrom) {
                for (short i = 0; i < edges.size(); i++)
                        if (edges[i].from == vFrom) {
                                int res = edges[i].to;
                                edges.erase(edges.begin() + i);
                                return res;
                        }
                return 0;
        }

        string createGraphFunction(vector <Edge> edges, int v, int vPrev = 0,
string acc = "") {
                int vNext = findToAndDeleteEdge(edges, v);
                if (vNext == 0) {
                        if (findToAndDeleteEdge(edges, vPrev) == 0)
                                return acc += to_string(v) + ")";
                        else
                                return acc += to_string(v) + ",";
                }

                acc += to_string(v) + "(";
                while (vNext != 0) {
                        acc = createGraphFunction(edges, vNext, v, acc);
                        vNext = findToAndDeleteEdge(edges, v);
                }

                if (vNext == 0) {
                        if (findToAndDeleteEdge(edges, vPrev) == 0)
                                acc += ")";
                        else
                                acc += ",";
                }
                return acc;
        }
public:
        GraphFunction(string inFile, string outFile = "out.txt") :
Graph(inFile, outFile) {
                int* usedV = new int[*(--vertexes.end())]{};
                auto i = vertexes.find(0) != vertexes.end() ?
++vertexes.begin() : vertexes.begin();
                for (; i != vertexes.end(); i++)
                        checkCycles(*i, usedV);
                delete[] usedV;

                if (!errors.empty()) {
                        if (!errors.empty())
                                for (auto i = errors.begin(); i !=
errors.end(); i++)
                                        cout << endl << *i;
                        return;
                }

                vector <Edge> sortedEdges = sortEdges(this->edges);
                graphFunction = createGraphFunction(sortedEdges,
sortedEdges[0].from);
                graphFunction.erase(--graphFunction.end());
        }
        ~GraphFunction() {//It's destructor
        }
```

```
        string getGraphFunction() {
                return graphFunction;
        }


        void printGraphFunctionInFile(string outFile = "") {
                if (!errors.empty())
                        return;

                if (outFile == "")
                        outFile = this->outFile;
                ofstream fout(outFile);
                if (!fout.is_open())
                        throw string{ "Недостаточно прав для создания файла " +
outFile + "!\n" };

                fout << graphFunction;
                fout.close();
                cout << "\nДанные успешно сохранены в " << outFile << endl;
        }
};



int main(int argc, char* argv[]) {
        setlocale(LC_ALL, "ru");

        if (argc != 2 && argc != 3) {
                cout << "\nНеверно заданы параметры функции!";
                cout << "\nПример запуска: NN_3.exe input1.txt out1.txt,
где:";
                cout << "\n\tinput1.txt - ориентированный граф с именованными
вершинами \n\tout1.txt - выходной файл (по умолчанию out.txt) \n";
                return 0;
        }

        string inFile = argv[1];
        string outFile = "out.txt";
        if (argc == 3)
                outFile = argv[2];

        try {
                GraphFunction gFun(inFile);
                gFun.printGraphFunctionInFile(outFile);
        }
        catch (string& error) {
                cout << error;
        }
        cout << endl;
        return 0;
}
```

ПРИЛОЖЕНИЕ В

Листинг кода для задания 3

```
#include "iostream"
#include "cmath"
#include "vector"
#include "string"
#include "set"
#include "map"
#include "fstream"
#include "algorithm"

using namespace std;
const double EXP = 2.71;

class File {
public:
	static string read(string inFile) {
		ifstream fin(inFile);
		if (!fin.is_open())
			throw string{ "Файл " + inFile + " не найден! \n" };

		string str = "";
		for (char el; fin.get(el);)
			str += el;

		fin.close();
		return str;
	}

	static void write(string str, string outFile) {
		ofstream fout(outFile);
		if (!fout.is_open())
			throw string{ "Недостаточно прав для создания файла " +
outFile + "!\n" };

		fout << str;
		fout.close();
	}

	static void append(string str, string outFile) {
		ofstream fout(outFile, ios::app);
		if (!fout.is_open())
			throw string{ "Файл " + outFile + " не найден! \n" };

		fout << endl << str;
		fout.close();
	}
};



class Edge {
public:
	int from = -1;
	short rawFrom = 0;
	int to = -1;
	short rawTo = 0;
	short order = -1;

	Edge(int from, short rawFrom, int to = -1, short rawTo = 0, short
order = -1) {
		this->from = from;
```

```
                this->rawFrom = rawFrom;
                this->to = to;
                this->rawTo = rawTo;
                this->order = order;
        }
        ~Edge() {//It's destructor
        }

        bool operator == (Edge& edge) const {
                return (this->from == edge.from && this->to == edge.to);
        }
        bool operator < (Edge& edge) const {
                if (this->from == edge.from)
                        return this->to < edge.to;
                return this->from < edge.from;
        }
        bool operator > (Edge& edge) const {
                if (this->from == edge.from)
                        return this->to > edge.to;
                return this->from > edge.from;
        }

        Edge reverse() {
                swap(from, to);
                return *this;
        }
};



class Graph {
protected:
        string inFile;
        string outFile;
        set <int> vertexes;
        vector <Edge> edges;
        set <string> errors;
        vector <pair <int, set <pair <int, int>>>> numEdges;

        bool checkStrDigit(string str, short raw) {
                if (str.empty()) {
                        this->errors.insert(string{ "Ошибка в строке " +
to_string(raw) + ": некорретные данные (введены символы вместо цифр)" });
                        return false;
                }
                for (int i = 0; i < str.length(); i++)
                        if (!isdigit(str[i])) {
                                this->errors.insert(string{ "Ошибка в строке "
+ to_string(raw) + ": некорретные данные (введены символы вместо цифр)" });
                                return false;
                        }
                return true;
        }


        void checkSameEdge(int from, int to, short raw) {
                for (short i = 0; i < edges.size(); i++)
                        if (edges[i].from == from && edges[i].to == to)
                                this->errors.insert(string{ "Ошибка в строке "
+ to_string(raw) + ": повторяющаяся дуга " +

           to_string(from) + "->" + to_string(to) });
        }
```

```
        void checkSameNumEdge(int to, int order, short raw) {
                for (short i = 0; i < edges.size(); i++)
                        if (edges[i].to == to && edges[i].order == order)
                                this->errors.insert(string{ "Ошибка в строке "
+ to_string(raw) + ": дуга с номером " + to_string(order) +

           " в вершину " + to_string(to) + " уже существует" });
        }


        void checkNumVertex() {
                int vertexSize = vertexes.size();
                for (short i = 0; i < edges.size(); i++) {
                        if (edges[i].from > vertexSize)
                                this->errors.insert(string{ "Ошибка в строке "
+ to_string(edges[i].rawFrom) + ": неправильная нумерация вершин." +

           " Номер вершина " + to_string(edges[i].from) + " больше количества
вершин" });
                        if (edges[i].to > vertexSize)
                                this->errors.insert(string{ "Ошибка в строке "
+ to_string(edges[i].rawTo) + ": неправильная нумерация вершин." +

           " Номер вершина " + to_string(edges[i].to) + " больше количества
вершин" });
                }
        }


        void addNumEdges(int to, short order, int raw) {
                bool emptyVertex = true;
                for (short i = 0; i < numEdges.size(); i++)
                        if (numEdges[i].first == to) {
                                this-
>numEdges[i].second.insert(make_pair(order, raw));
                                emptyVertex = false;
                                break;
                        }
                if (emptyVertex) {
                        this->numEdges.push_back(make_pair(to, set <pair <int,
int>> {make_pair(order, raw)}));
                }
        }


        void checkNumEdges() {
                for (int i = 0; i < numEdges.size(); i++) {
                        int numPrev = 0;
                        for (auto j = numEdges[i].second.begin(); j !=
numEdges[i].second.end(); j++) {
                                pair <int, int> num = *j;
                                if (num.first - numPrev == 1 || num.first -
numPrev == 0)
                                        numPrev = num.first;
                                else {
                                        this->errors.insert(string{ "Ошибка в
строке " + to_string(num.second) + ": неправильно заданы номера дуг" });
                                        numPrev = num.first;
                                }
                        }
                }
        }
```

```
        void getData() {
                ifstream fin(inFile);
                if (!fin.is_open())
                        throw string{ "Файл " + inFile + " не найден! \n" };

                string str = "";
                string partEdge = "";
                short raw = 1;
                for (char el, elPrev = ' '; fin.get(el); elPrev = el) {
                        if (el == '\n')
                                raw++;
                        else if (el == ' ' && elPrev == ',')
                                continue;
                        else if (el == '(' && partEdge == "") {
                                partEdge = "from";
                                str = "";
                        }
                        else if (el == ')' && !edges.empty() &&
edges.back().order != -1)
                                this->errors.insert(string{ "Ошибка в строке "
+ to_string(raw) + ": неправильно задана компонента. Формат: (a, b, n)" });
                        else if (partEdge != "" && (el == ',' || el == ')')) {
                                if (!checkStrDigit(str, raw))
                                        str = "-1";

                                if (partEdge == "from") {
                                        if (str == "0")
                                                this->errors.insert(string{
"Ошибка в строке " + to_string(raw) + ": вершины 0 быть не может" });

                                        this->vertexes.insert(stoi(str));
                                        this->edges.push_back(Edge(stoi(str),
raw));
                                        partEdge = "to";
                                        str = "";
                                }
                                else if (partEdge == "to") {
                                        if (str == "0")
                                                this->errors.insert(string{
"Ошибка в строке " + to_string(raw) + ": вершины 0 быть не может" });
                                        checkSameEdge(edges.back().from,
stoi(str), raw);

                                        this->vertexes.insert(stoi(str));
                                        this->edges.back().to = stoi(str);
                                        this->edges.back().rawTo = raw;
                                        partEdge = "order";
                                        str = "";
                                }
                                else if (partEdge == "order") {
                                        checkSameNumEdge(edges.back().to,
stoi(str), raw);

                                        this->edges.back().order = stoi(str);
                                        partEdge = "";
                                        str = "";

                                        addNumEdges(edges.back().to,
edges.back().order, raw);
                                }
                        }
                        else
                                str += el;
```

```
                }

                checkNumVertex();
                checkNumEdges();
                fin.close();
        }
public:
        Graph(string inFile, string outFile) {
                this->inFile = inFile;
                this->outFile = outFile;
                getData();

                if (edges.empty())
                        throw (string{ "\nФайл не содержит граф!" });
                else if (!errors.empty()) {
                        for (auto i = errors.begin(); i != errors.end(); i++)
                                cout << endl << *i;
                        errors.clear();
                }
        }
        ~Graph() {//It's destructor
        }


        void printGraph() {
                if (!errors.empty())
                        return;
                cout << "\nВершины: ";
                for (auto i = vertexes.begin(); i != vertexes.end(); i++)
                        cout << *i << " ";
                cout << "\nДуги: ";
                for (short i = 0; i < edges.size(); i++)
                        cout << "(" << edges[i].from << ", " << edges[i].to <<
", " << edges[i].order << ") ";
        }


        void printGraphInFile() {
                if (!errors.empty())
                        return;

                ofstream fout(outFile);
                if (!fout.is_open())
                        throw string{ "Файл " + outFile + " не найден! \n" };

                fout << "<graph>";
                for (auto i = vertexes.begin(); i != vertexes.end(); i++)
                        fout << "\n\t<vertex>v" << *i << "</vertex>";
                for (short i = 0; i < edges.size(); i++) {
                        fout << "\n\t<arc>";
                        fout << "\n\t\t<from>v" << edges[i].from << "</from>";
                        fout << "\n\t\t<to>v" << edges[i].to << "</to>";
                        fout << "\n\t\t<order>" << edges[i].order <<
"</order>";
                        fout << "\n\t</arc>";
                }
                fout << "\n</graph>";
                cout << "\nРезультат сохранён в файл " << outFile;
                fout.close();
        }
};
```

```
class GraphFunction : protected Graph {
protected:
	string graphFunction = "";

	void checkCycles(int curV, int* usedV) {
		usedV[curV - 1] = 1;
		for (short i = 0; i < edges.size(); i++)
			if (edges[i].from == curV) {
				if (usedV[edges[i].to - 1] == 1)
					errors.insert(string{ "Ошибка в строке
" + to_string(edges[i].rawTo) + ": в графе существует цикл" });
				else {
					checkCycles(edges[i].to, usedV);
					usedV[edges[i].to - 1] = 0;
				}
			}
	}


	vector <Edge> sortEdges(vector <Edge> edges) {//edges - копия
переменной this->edges
		for (short i = 0; i < edges.size(); i++)
			edges[i].reverse();

		vector <Edge> res;
		while (!edges.empty()) {
			pair <Edge, short> minEdge = make_pair(edges[0], 0);
			for (short i = 1; i < edges.size(); i++)
				if (edges[i] < minEdge.first)
					minEdge = make_pair(edges[i], i);
			res.push_back(minEdge.first);
			edges.erase(edges.begin() + minEdge.second);
		}

		reverse(res.begin(), res.end());
		for (short i = 0; i < res.size() - 1; i++)
			if (res[i].from == res[i + 1].from && res[i].to > res[i
+ 1].to)
				swap(res[i], res[i + 1]);

		return res;
	}


	int findToAndDeleteEdge(vector <Edge>& edges, int vFrom) {
		for (short i = 0; i < edges.size(); i++)
			if (edges[i].from == vFrom) {
				int res = edges[i].to;
				edges.erase(edges.begin() + i);
				return res;
			}
		return 0;
	}

	string createGraphFunction(vector <Edge> edges, int v, int vPrev = 0,
string acc = "") {
		int vNext = findToAndDeleteEdge(edges, v);
		if (vNext == 0) {
			if (findToAndDeleteEdge(edges, vPrev) == 0)
				return acc += to_string(v) + ")";
			else
				return acc += to_string(v) + ",";
		}
```

```
                acc += to_string(v) + "(";
                while (vNext != 0) {
                        acc = createGraphFunction(edges, vNext, v, acc);
                        vNext = findToAndDeleteEdge(edges, v);
                }

                if (vNext == 0) {
                        if (findToAndDeleteEdge(edges, vPrev) == 0)
                                acc += ")";
                        else
                                acc += ",";
                }
                return acc;
        }
public:
        GraphFunction(string inFile, string outFile = "out.txt") :
Graph(inFile, outFile) {
                int* usedV = new int[*(--vertexes.end())]{};
                auto i = vertexes.find(0) != vertexes.end() ?
++vertexes.begin() : vertexes.begin();
                for (; i != vertexes.end(); i++)
                        checkCycles(*i, usedV);
                delete[] usedV;

                if (!errors.empty()) {
                        for (auto i = errors.begin(); i != errors.end(); i++)
                                cout << endl << *i;
                        return;
                }

                vector <Edge> sortedEdges = sortEdges(this->edges);
                this->graphFunction = createGraphFunction(sortedEdges,
sortedEdges[0].from);
                this->graphFunction.erase(--graphFunction.end());
        }
        ~GraphFunction() {//It's destructor
        }


        string getGraphFunction() {
                return graphFunction;
        }


        void printGraphFunctionInFile(string outFile = "") {
                if (!errors.empty())
                        return;

                if (outFile == "")
                        outFile = this->outFile;
                ofstream fout(outFile);
                if (!fout.is_open())
                        throw string{ "Файл " + outFile + " не найден! \n" };

                fout << graphFunction;
                fout.close();
                cout << "\nДанные успешно сохранены в " << outFile;
        }
};



class ValueGraphFunction : protected GraphFunction {
protected:
```

```
        string cmdFile;
        map <int, string> cmds;
        double valueGF;

        void getCommands() {
                unsigned short iStart = 0, twoPoints = 0;
                string str = File::read(cmdFile) + "\n";
                replace(str.begin(), str.end(), '.', ',');
                for (unsigned short i = 0; i < str.length(); i++) {
                        if (str[i] == ':')
                                twoPoints = i;
                        else if (str[i] == '\n') {
                                this-
>cmds.insert(make_pair(stoi(str.substr(iStart, iStart - twoPoints)),
str.substr(twoPoints + 2, i - twoPoints - 2)));
                                iStart = i + 1;
                        }
                }
        }


        double getValue(vector <Edge> edges, int v, int vPrev = 0, double res
= 0) {
                int vNext = findToAndDeleteEdge(edges, v);
                if (vNext == 0) {
                        if (cmds[vPrev] == "+")
                                return res += stoi(cmds[v]);
                        else if (cmds[vPrev] == "*")
                                return res == 0 ? stoi(cmds[v]) : res *
stoi(cmds[v]);
                        else if (cmds[vPrev] == "exp") {
                                return pow(EXP, stod(cmds[v]));
                        }
                }

                double resHelp = 0;
                while (vNext != 0) {
                        resHelp = getValue(edges, vNext, v, resHelp);
                        vNext = findToAndDeleteEdge(edges, v);
                }

                if (cmds[vPrev] == "+")
                        return res += resHelp;
                else if (cmds[vPrev] == "*")
                        return res == 0 ? resHelp : res * resHelp;
                else if (cmds[vPrev] == "exp")
                        return pow(EXP, resHelp);

                return resHelp;
        }
public:
        ValueGraphFunction(string inFile, string cmdFile, string outFile) :
GraphFunction(inFile, outFile) {
                this->cmdFile = cmdFile;
                getCommands();
                vector <Edge> sortedEdges = sortEdges(this->edges);
                this->valueGF = getValue(sortedEdges, sortedEdges[0].from);
        }
        ~ValueGraphFunction() {
        }


        void printValueGFinFile() {
                string str = to_string(this->valueGF);
```

```
                for (unsigned short i = str.length() - 1; str[i - 2] != ',';
i--)
                        str.pop_back();

                File::write("Функция: " + graphFunction, outFile);
                File::append("Значение функции: " + str, outFile);
                cout << "\nДанные успешно записаны в файл " + outFile;
        }
};



int main(int argc, char* argv[]) {
        setlocale(LC_ALL, "ru");

        if (argc != 3 && argc != 4) {
                cout << "\nНеверно заданы параметры функции!";
                cout << "\nПример запуска: NN_3.exe input1.txt input1cmd.txt
out1.txt, где:";
                cout << "\n\tinput1.txt - файл с графом \n\tinput1cmd.txt -
файл с командами \n\tout1.txt - выходной файл (по умолчанию out.txt) \n";
                return 0;
        }

        string inFile = argv[1];
        string cmdFile = argv[2];
        string outFile = "out.txt";
        if (argc == 4)
                outFile = argv[3];

        try {
                ValueGraphFunction valueFun(inFile, cmdFile, outFile);
                valueFun.printValueGFinFile();
        }
        catch (string& error) {
                cout << error;
        }
        cout << endl;
        return 0;
}
```

ПРИЛОЖЕНИЕ Г

Листинг кода для задания 4

```
#include "iostream"
#include "vector"
#include "string"
#include "fstream"
#include "set"

using namespace std;

class File
{
public:
	static string read(string inFile) {
		ifstream fin(inFile);
		if (!fin.is_open())
			throw string{ "Файл " + inFile + " не найден! \n" };

		string str = "";
		for (char el; fin.get(el);)
			str += el;
		if (str.back() != '\n')
			str += '\n';

		fin.close();
		return str;
	}

	static void write(string str, string outFile) {
		ofstream fout(outFile);
		if (!fout.is_open())
			throw string{ "Недостаточно прав для создания файла " +
outFile + "!\n" };

		fout << str;
		if (str.back() != '\n')
			fout << "\n";
		fout.close();
	}

	static void append(string str, string outFile) {
		ofstream fout(outFile, ios::app);
		if (!fout.is_open())
			throw string{ "Файл " + outFile + " не найден! \n" };

		fout << endl << str;
		if (str.back() != '\n')
			fout << '\n';
		fout.close();
	}
};



class NeurealNetwork
{
private:
	bool checkStrDigit(string str, short raw, string file)
	{
		if (str.empty()) {
```

```
                        errors.insert(string{ "Ошибка в файле " + file + ", " +
"строка " + to_string(raw) + ": некорретные данные(введены символы вместо
цифр)" });
                        return false;
                }
                for (int i = 0; i < str.length(); i++)
                        if (!isdigit(str[i])) {
                                errors.insert(string{ "Ошибка в файле " + file
+ ", " + "строка " + to_string(raw) + ": некорретные данные(введены символы
вместо цифр)" });
                                return false;
                        }
                return true;
        }
public:
        vector <vector <vector <int>>> matrix;
        vector <double> vec;
        set <string> errors;
        string matrixFile, vectorFile;
        vector <double> valueNN;

        NeurealNetwork(string matrixFile, string vectorFile)
        {
                this->matrixFile = matrixFile;
                this->vectorFile = vectorFile;
                getData(matrixFile, vectorFile);
                checkErrors();
                getValueNN();
        }


        void getData(string matrixFile, string vectorFile)
        {
                matrix.clear(), vec.clear();

                string str = File::read(matrixFile);
                vector <vector <int>> raw;
                string substr = "";
                unsigned short numStr = 1;
                for (unsigned short i = 0; i < str.size(); i++)
                {
                        switch (str[i])
                        {
                        case '[':
                                substr = "";
                                raw.push_back(vector <int> {});
                                break;
                        case ']':
                                if (!checkStrDigit(substr, numStr, matrixFile))
                                        continue;
                                raw.back().push_back(stoi(substr));
                                break;
                        case ' ':
                                if (str[i - 1] == ']' || !checkStrDigit(substr,
numStr, matrixFile))
                                        continue;
                                raw.back().push_back(stoi(substr));
                                substr = "";
                                break;
                        case '\n':
                                matrix.push_back(raw);
                                raw.clear();
                                numStr++;
                                break;
```

```
                        default:
                                substr += str[i];
                        }
                }

                str = File::read(vectorFile);
                substr = "";
                for (unsigned short i = 0; i < str.size(); i++)
                {
                        if (str[i] == ' ' || str[i] == '\n')
                        {
                                if (!checkStrDigit(substr, numStr, vectorFile))
                                        continue;
                                vec.push_back(stoi(substr));
                                substr = "";
                        }
                        else
                                substr += str[i];
                }
        }


        void checkErrors()
        {
                unsigned short vecSize = vec.size();
                for (unsigned short i = 0; i < matrix[0].size(); i++)
                        if (matrix[0][i].size() != vecSize)
                                errors.insert(string{ "Ошибка в файле " +
matrixFile + ", " + "строка 1: разное количество элементов нейрона и вектора"
});

                for (unsigned short i = 1; i < matrix.size(); i++)
                {
                        for (unsigned short j = 0; j < matrix[i].size(); j++)
                        {
                                if (matrix[i][j].size() != matrix[i -
1].size())
                                {
                                        errors.insert(string{ "Ошибка в файле"
+ matrixFile + ", строка " + to_string(i + 1) +

": несоответствие размеров нейрона и предыдущего слоя" });
                                }
                        }
                }
        }


        vector <double> getValueNN() {
                if (!errors.empty())
                {
                        for (auto i = errors.begin(); i != errors.end(); i++)
                                cout << endl << *i;
                        return vector <double> {-1};
                }

                vector <double> res = this->vec;
                vector <double> vec;
                for (unsigned short i = 0; i < matrix.size(); i++)
                {
                        vec = res;
                        res.clear();
                        for (unsigned short j = 0; j < matrix[i].size(); j++)
                        {
```

```
                                double val = 0;
                                for (unsigned short k = 0; k <
matrix[i][j].size(); k++)
                                        val += matrix[i][j][k] * vec[k];
                                res.push_back(val / (1 + abs(val)));
                        }
                }
                this->valueNN = res;
                return res;
        }


        void printResultInFile(string outNetFile = "outNetwork.txt", string
outValFile = "outValue.txt")
        {
                ofstream fout(outNetFile);
                if (!fout.is_open())
                        throw string{ "Недостаточно прав для создания файла " +
outNetFile + "!\n" };

                fout << "<network>";
                for (unsigned short i = 0; i < matrix.size(); i++)
                {
                        fout << "\n\t<layer>";
                        for (unsigned short j = 0; j < matrix[i].size(); j++)
                        {
                                fout << "\n\t\t<connections>";
                                for (unsigned short k = 0; k <
matrix[i][j].size(); k++)
                                        fout << "\n\t\t\t<weight>" <<
matrix[i][j][k] << "</weight>";
                                fout << "\n\t\t</connections>";
                        }
                        fout << "\n\t</layer>";
                }
                fout << "\n</network>";
                cout << "\nМногослойная нейронная сеть успешно записана в файл
" << outNetFile;
                fout.close();

                fout.open(outValFile);
                if (!fout.is_open())
                        throw string{ "Недостаточно прав для создания файла " +
outValFile + "!\n" };

                for (unsigned short i = 0; i < valueNN.size(); i++)
                        fout << valueNN[i] << " ";
                cout << "\nРезультат вычислений НС успешно записан в файл " <<
outValFile;
                fout.close();
        }
};

int main(int argc, char* argv[])
{
        setlocale(LC_ALL, "ru");

        if (argc != 3 && argc != 4 && argc != 5) {
                cout << "\nНеверно заданы параметры функции!";
                cout << "\nПример запуска: NN_4.exe matrix1.txt vector1.txt
outNetwork1.txt outValue1.txt, где:";
                cout << "\n\tmatrix1.txt - файл с матрицей \n\tvector1.txt -
файл с вектором";
```

```
                cout << "\n\toutNetwork1.txt - многослойная нейронная сеть в
формате XML (по умолчанию outNetwork.txt)";
                cout << "\n\toutValue1.txt - результат вычислений нейронной
сети (по умолчанию outValue.txt) \n\n";
                return 0;
        }

        string matrixFile = argv[1];
        string vectorFile = argv[2];
        string outNetworkFile = "outNetwork.txt";
        string outValueFile = "outValue.txt";
        if (argc >= 4)
                outNetworkFile = argv[3];
        if (argc == 5)
                outValueFile = argv[4];

        try
        {
                NeurealNetwork nn(matrixFile, vectorFile);
                nn.printResultInFile(outNetworkFile, outValueFile);
        }
        catch (string& errors)
        {
                cout << endl << errors;
        }

        cout << endl;
        return 0;
}
```

ПРИЛОЖЕНИЕ Д

Листинг кода для задания 5

```
#include "iostream"
#include "vector"
#include "string"
#include "fstream"
#include "set"
#include <nlohmann/json.hpp>

using namespace std;
using json = nlohmann::json;
using namespace nlohmann;

class NeuronValue
{
public:
	double before, after;

	NeuronValue(double before, double after)
	{
		this->before = before;
		this->after = after;
	}
};



class TrainingData
{
public:
	vector <double> input, output;

	TrainingData(vector <double> input, vector <double> output)
	{
		this->input = input;
		this->output = output;
	}
};



class NeurealNetwork
{
private:
	vector <vector <double>> mult(vector <vector <double>> a, vector
<vector <double>> b)
	{
		unsigned short rawsA = a.size();
		unsigned short colsA = a[0].size();
		unsigned short colsB = b[0].size();
		vector <vector <double>> res(rawsA, vector <double>(colsB,
0));

		for (unsigned short i = 0; i < rawsA; i++)
			for (unsigned short j = 0; j < colsB; j++)
				for (unsigned short k = 0; k < colsA; k++)
					res[i][j] += a[i][k] * b[k][j];
		return res;
	}


	vector <vector <double>> transpose(vector <vector <double>> matrix)
```

```
        {
                unsigned short raws = matrix.size();
                unsigned short cols = matrix[0].size();
                vector <vector <double>> res(cols, vector <double>(raws, 0));

                for (unsigned short i = 0; i < raws; i++)
                        for (unsigned short j = 0; j < cols; j++)
                                res[j][i] = matrix[i][j];
                return res;
        }
public:
        vector <int> numNeuron;
        vector <vector <vector <double>>> neuronet;
        set <string> errors;
        vector <string> outputData;

        NeurealNetwork(string matrixFile) {
                ifstream fin(matrixFile);
                if (!fin.is_open())
                        throw string{ "Файл " + matrixFile + " не найден! \n"
};
                json jsonMas;

                try
                {
                        fin >> jsonMas;
                }
                catch (const json::parse_error& e)
                {
                        throw string{ "Файл " + matrixFile + " не является
файлом JSON-формата!" };
                }

                for (unsigned short i = 0; i < jsonMas.size(); i++)
                {
                        json layerJSON = jsonMas[i];
                        vector <vector <double>> layer;

                        for (unsigned short j = 0; j < layerJSON.size(); j++)
                        {
                                json neuronJSON = layerJSON[j];
                                vector <double> neuron;

                                for (unsigned short k = 0; k <
neuronJSON.size(); k++)
                                        neuron.push_back(neuronJSON[k]);

                                if (numNeuron.empty())
                                        numNeuron.push_back(neuron.size());
                                else if (numNeuron.back() != neuron.size())
                                {
                                        errors.insert(string{ "Ошибка в файле"
+ matrixFile + ", строка " + to_string(i + 1) +

": несоответствие размеров нейрона и предыдущего слоя" });
                                }

                                layer.push_back(neuron);
                        }

                        numNeuron.push_back(layer.size());
                        neuronet.push_back(transpose(layer));
                }
        }
```

```
        void training(string trainingFile, unsigned short n)
        {
                double k = 0.01;
                ifstream fin(trainingFile);
                if (!fin.is_open())
                        throw string{ "Файл " + trainingFile + " не найден! \n"
};

                string str;
                vector <TrainingData> trData;
                unsigned short raw = 1;
                while (getline(fin, str))
                {
                        json jsonObject;
                        try
                        {
                                jsonObject = json::parse(str);
                        }
                        catch (json::parse_error error)
                        {
                                throw string{ "Файл " + trainingFile + " не
является файлом JSON-формата!" };
                        }

                        auto inputMas = jsonObject["i"];
                        vector <double> input = inputMas.get<vector
<double>>();
                        auto outputMas = jsonObject["o"];
                        vector <double> output = outputMas.get<vector
<double>>();

                        if (input.size() != numNeuron[0] || output.size() !=
numNeuron.back())
                        {
                                errors.insert(string{ "Ошибка в строке " +
to_string(raw) + ": неверное количестве входных\\выходных данных!" });
                                raw++;
                                continue;
                        }

                        trData.push_back(TrainingData(input, output));
                }
                fin.close();

                if (!errors.empty())
                        return;
                else if (trData.empty())
                {
                        errors.insert(string{ "Не найдено данных для
тренировки" });
                        return;
                }

                int numberTraining = 0;
                for (unsigned short i = 1; i <= n; i++)
                {
                        vector <vector <NeuronValue>> neuronsValues;
                        TrainingData curTrData = trData[numberTraining];
                        vector <double> input = curTrData.input;
                        vector <NeuronValue> neuronsValuesInLayer;

                        for (double d : input)
```

```
                                neuronsValuesInLayer.push_back(NeuronValue(d,
d));
                        neuronsValues.push_back(neuronsValuesInLayer);

                        vector <vector <double>> res;
                        res.push_back(input);

                        for (auto m : this->neuronet)
                        {
                                neuronsValuesInLayer.clear();
                                res = mult(res, m);

                                for (unsigned short j = 0; j < res[0].size();
j++)
                                {
                                        NeuronValue curVal(res[0][j], res[0][j]
/ (1 + abs(res[0][j])));

                                        res[0][j] = curVal.after;
                                        neuronsValuesInLayer.push_back(curVal);
                                }

                                neuronsValues.push_back(neuronsValuesInLayer);
                        }

                        vector <vector < vector <double>>> deltaNeuronet;
                        vector <double> errors;

                        for (unsigned short m = 0, j = neuronsValues.size() -
1; m < neuronsValues[j].size(); m++)
                        {
                                double tk = curTrData.output[m];
                                double yk = neuronsValues[j][m].after;
                                double help = 1.0 / ((1 +
abs(neuronsValues[j][m].before)) * (1 + abs(neuronsValues[j][m].before)));
                                errors.push_back((tk - yk) * help);
                        }

                        vector <vector <double>> deltaLayer;
                        vector <vector <double>> layer = neuronet[i - 1];
                        for (unsigned short m = 0; m < layer[0].size(); m++)
                        {
                                vector <double> deltaNeuron;
                                for (unsigned short l = 0; l <
deltaNeuron.size(); l++)
                                        deltaNeuron.push_back(k * errors[m] *
neuronsValues[i - 1][l].after);
                                deltaLayer.push_back(deltaNeuron);
                        }
                        deltaNeuronet.push_back(transpose(deltaLayer));

                        for (unsigned short j = neuronsValues.size() - 2; j >=
1; j--)
                        {
                                layer = neuronet[j];
                                vector <double> errorsIn;

                                for (unsigned short m = 0; m < layer.size();
m++)
                                {
                                        double errorInj = 0;
                                        for (unsigned l = 0; l <
layer[m].size(); l++)
                                                errorInj += errors[l] *
layer[m][l];
```

```
                                        errorsIn.push_back(errorInj);
                                }

                                errors.clear();
                                for (unsigned short m = 0; m <
neuronsValues[i].size(); m++)
                                {
                                        double help = 1.0 / ((1 +
abs(neuronsValues[j][m].before)) * (1 + abs(neuronsValues[j][m].before)));
                                        errors.push_back(errorsIn[m] * help);
                                }

                                layer = neuronet[i - 1];
                                deltaLayer.clear();

                                for (unsigned short m = 0; m < layer.size();
m++)
                                {
                                        vector <double> deltaNeuron;
                                        for (unsigned short l = 0; l <
layer[m].size(); l++)
                                                deltaNeuron.push_back(k *
errors[l] * neuronsValues[i - 1][m].after);
                                        deltaLayer.push_back(deltaNeuron);
                                }

                                deltaNeuronet.push_back(deltaLayer);
                        }

                        for (unsigned short m = 0; m < neuronet.size(); m++)
                        {
                                layer = neuronet[m];
                                deltaLayer = deltaNeuronet[neuronet.size() - m
- 1];

                                for (unsigned short l = 0; l < layer.size();
l++)
                                {
                                        vector <double> neuron = layer[l];
                                        vector <double> deltaNeuron =
deltaLayer[l];

                                        for (unsigned r = 0; r < neuron.size();
r++)
                                                neuron[r] += deltaNeuron[r];
                                }
                        }

                        string str = "Итерация " + to_string(raw) + ": [";
                        raw++;
                        for (unsigned short i = 0; i < errors.size(); i++)
                                str += to_string(errors[i]) + ", ";
                        str.pop_back();
                        str += "]";
                        outputData.push_back(str);

                        numberTraining = (numberTraining + 1) % trData.size();
                }
        }


        void printResultInFile(string outFile, int n)
        {
                ofstream fout(outFile);
```

```
                if (!fout.is_open())
                        throw string{ "Недостаточно прав для создания файла " +
outFile + "!\n" };

                for (unsigned short i = 0; i < outputData.size(); i++)
                        fout << outputData[i] << endl;

                fout.close();
        }
};



int main(int argc, char* argv[])
{
        setlocale(LC_ALL, "ru");

        if (argc != 3 && argc != 4 && argc != 5) {
                cout << "\nНеверно заданы параметры функции!";
                cout << "\nПример запуска: NN_4.exe matrix1.json training1.txt
20 outRes1.txt, где:";
                cout << "\n\tmatrix1.txt - файл с описание нейронной сети
\n\ttraining.txt - файл с обучающей выборкой";
                cout << "\n\t20 - число итераций обучения";
                cout << "\n\toutRes1.txt - файл с историей 20 итераций
обучения (по умолчанию outValue.txt) \n";
                return 0;
        }

        string matrixFile = argv[1];
        string trainingFile = argv[2];
        int n = stoi(argv[3]);
        string outFile = "outValue.txt";
        if (argc == 5)
                outFile = argv[4];

        try
        {
                NeurealNetwork nn(matrixFile);
                nn.training(trainingFile, n);
                nn.printResultInFile(outFile, n);
        }
        catch (string& error)
        {
                cout << endl << error;
        }

        cout << endl;
        return 0;
}
```