Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САРАТОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

**НЕЙРОННЫЕ СЕТИ**

**ОТЧЕТ ПО ПРАКТИЧЕСКОМУ КУРСУ**

студента 5 курса 531 группы

факультета компьютерных наук и информационных технологий

*Кузнецова Егора Дмитриевича*

фамилия, имя, отчество

Научный руководитель

Ст. преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ И.И. Слеповичев

подпись, дата

Саратов 2024

**ПРАКТИЧЕСКАЯ РАБОТА**

**Задание 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), ..., (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:**

1. Ориентированный граф с именованными вершинами и линейно упорядоченными дугами (в соответствии с порядком из текстового файла).
2. Сообщение об ошибке в формате файла, если ошибка присутствует.

**Способ проверки результата:**

1. Сериализованная структура графа в формате XML или JSON.

**Пример:**

<graph>

    <vertex>v1</vertex>

    <vertex>v2</vertex>

    <vertex>v3</vertex>

    <arc>

        <from>v1</from>

        <to>v3</to>

        <order>1</order>

    </arc>

    <arc>

        <from>v2</from>

        <to>v3</to>

        <order>2</order>

    </arc>

</graph>

1. Сообщение об ошибке с указанием номера строки с ошибкой во входном файле.

**Параметры запуска программы.**

.\nntask1.exe -input1 .\input.txt

**Пример 1.**

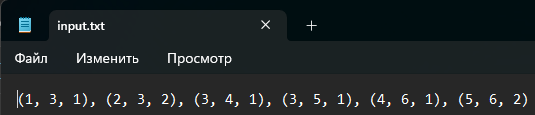
****

Рис. 1 – Входные данные программы.

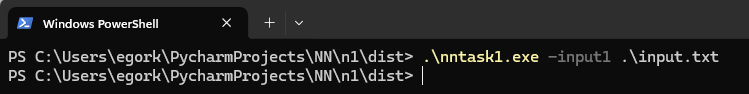


Рис. 2 – Запуск программы.

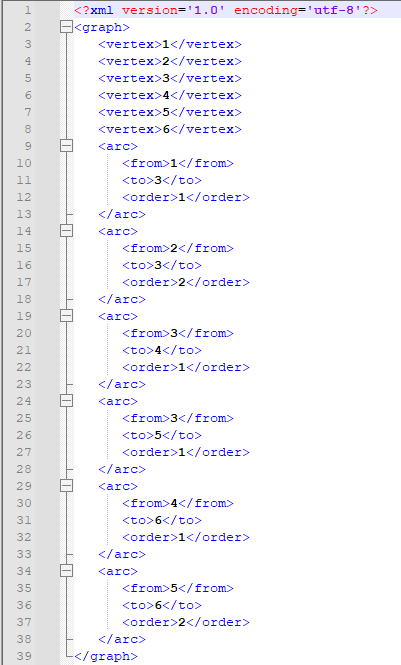


Рис. 3 – Результат работы программы.

**Пример 2.**

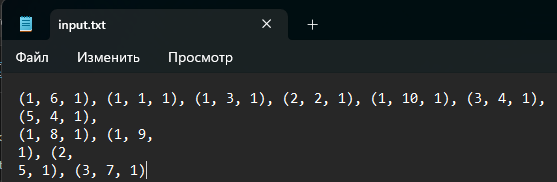
****

Рис. 4 – Входные данные программы.

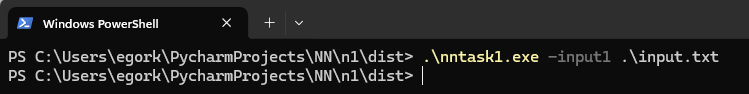


Рис. 5 – Запуск программы.

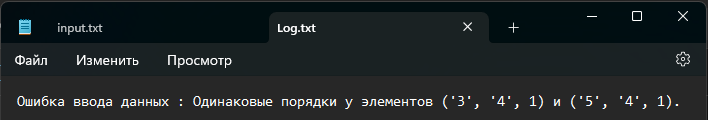


Рис. 6 – Результат работы программы.

**Листинг кода.**

import argparse  
import os  
import xml.etree.ElementTree as ET  
  
  
def log(error):  
 log\_file = open("Log.txt", "w", encoding="utf-8")  
 log\_file.write(error)  
  
  
def sort1(edges):  
 return sorted(edges, key=lambda x: (x[0]))  
  
  
def sort2(edges):  
 return sorted(edges, key=lambda x: (x[1]))  
  
  
def change\_order(edges):  
 result = []  
  
 for i in range(len(edges)):  
 elem = edges[i]  
 if elem not in result:  
 result.append(elem)  
 for j in range(i + 1, len(edges)):  
 sec = edges[j]  
 if elem[1] == sec[1]:  
 if elem[2] > sec[2]:  
 if sec not in result:  
 result.insert(result.index(elem), sec)  
 if elem[2] < sec[2]:  
 if sec not in result:  
 result.insert(result.index(elem) + 1, sec)  
  
 return result  
  
  
def check\_graph(input\_file):  
 if not os.path.isfile(input\_file):  
 log(f"Файл {input\_file} не существует")  
 exit()  
  
 edges = []  
 line = ""  
 try:  
 with open(input\_file, 'r', encoding='utf-8') as file:  
 for st in file:  
 line += st.replace("\n", " ")  
 except Exception:  
 log(f"Ошибка считывания данных из файла")  
 exit()  
  
 edges += (line.replace(" ", "").strip("\n").split("),"))  
  
 edge\_number = 1  
  
 for i in range(len(edges)):  
 try:  
 a, b, n = edges[i].strip(",()").split(",")  
 edges[i] = (a, b, int(n))  
 except Exception:  
 log(f"Ошибка ввода данных дуги {edge\_number}: Некорректный формат записи.")  
 exit()  
  
 edge\_number += 1  
  
 unique = set()  
 for x1, x2, \_ in edges:  
 if (x1, x2) in unique:  
 log(f"Ошибка ввода данных. Дуга из вершины {x1} в {x2} уже есть")  
 exit()  
 unique.add((x1, x2))  
  
 for i in range(len(edges)):  
 current = edges[i]  
 for j in range(i + 1, len(edges)):  
 elem = edges[j]  
 if current[1] == elem[1] and current[2] == elem[2]:  
 log(f"Ошибка ввода данных : Одинаковые порядки у элементов {current} и {elem}.")  
 exit()  
  
 # edges = change\_order(edges)  
  
 return edges  
  
  
def graph\_to\_xml(edges, path):  
 vertices = []  
  
 for (x1, x2, \_) in edges:  
 if int(x1) not in vertices:  
 vertices.append(int(x1))  
 if int(x2) not in vertices:  
 vertices.append(int(x2))  
  
 root = ET.Element('graph')  
 vertices.sort()  
 for vertex in vertices:  
 vertex\_elem = ET.SubElement(root, 'vertex')  
 vertex\_elem.text = str(vertex)  
  
 for vertex in edges:  
 arc\_elem = ET.SubElement(root, 'arc')  
 ET.SubElement(arc\_elem, 'from').text = vertex[0]  
 ET.SubElement(arc\_elem, 'to').text = vertex[1]  
 ET.SubElement(arc\_elem, 'order').text = str(vertex[2])  
  
 root = ET.fromstring(ET.tostring(root, encoding='utf-8'))  
 tree = ET.ElementTree(root)  
  
 ET.indent(tree, ' ')  
  
 tree.write(path, encoding="utf-8", xml\_declaration=True)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 parser = argparse.ArgumentParser(  
 prog="nntask1.exe",  
 description="Creating a directed graph")  
 parser.add\_argument("-input1", nargs=1, default=["input.txt"])  
 parser.add\_argument("-input2", nargs=1, default=["input2.txt"])  
 parser.add\_argument("-output1", nargs=1, default=["output.xml"])  
 parser.add\_argument("-output2", nargs=1, default=["output2.xml"])  
  
 args = parser.parse\_args()  
  
 first\_input = args.input1[0]  
 second\_input = args.input2[0]  
 first\_output = args.output1[0]  
 second\_output = args.output2[0]  
  
 graph = check\_graph(first\_input)  
  
 graph\_to\_xml(graph, first\_output)

**Задание 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:

A1(B1(C1(...),..., Cm(...)),..., Bn(...))

**Способ проверки результата:**

1. выгрузка в текстовый файл результата преобразования графа в имя функции.
2. сообщение о наличии циклов в графе, если они присутствуют.

**Параметры запуска программы.**

.\nntask2.exe -input1 .\input.xml

**Пример 1.**

****

Рис. 7 – Входные данные программы.

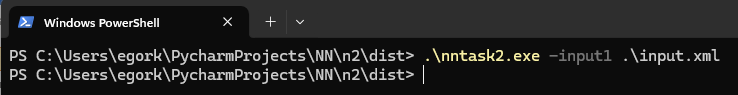
****

Рис. 8 – Запуск программы.

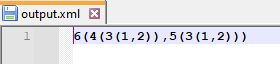


Рис. 9 – Результат работы программы.

**Пример 2.**

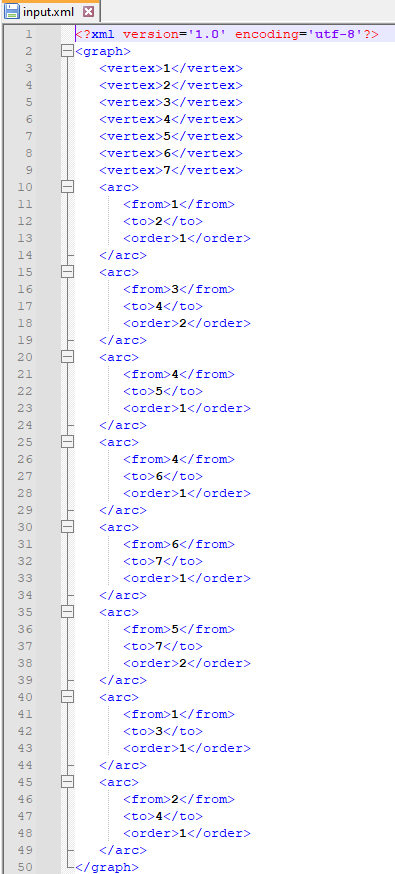
****

Рис. 10 – Входные данные программы.

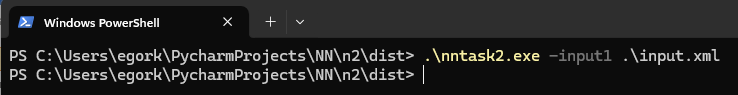
****

Рис. 11 – Запуск программы.



Рис. 12 – Результат работы программы.

**Листинг кода.**

import argparse  
import os  
import xml.etree.ElementTree as ET  
  
  
def log(error):  
 log\_file = open("Log.txt", "w", encoding="utf-8")  
 log\_file.write(error)  
  
  
def dfs(graph, start, visited=None):  
 if visited is None:  
 visited = set()  
 visited.add(start)  
 print(start)  
 for nextElem in graph[start] - visited:  
 dfs(graph, nextElem, visited)  
 return visited  
  
  
def check\_cycle(graph):  
 adj\_list = {}  
 for v1, v2, \_ in graph:  
 if v1 not in adj\_list:  
 adj\_list[v1] = []  
 if v2 not in adj\_list:  
 adj\_list[v2] = []  
 adj\_list[v1].append(v2)  
  
 for v in adj\_list:  
 visited = set()  
 stack = [v]  
 while stack:  
 current = stack.pop()  
 if current in visited:  
 return True  
 visited.add(current)  
 for neighbor in adj\_list[current]:  
 if neighbor not in visited:  
 stack.append(neighbor)  
  
 return False  
  
  
def get\_data(input\_file):  
 if not os.path.isfile(input\_file):  
 log(f"Файл {input\_file} не существует")  
 exit()  
  
 tree = ET.parse(input\_file)  
 root = tree.getroot()  
  
 edges = []  
  
 try:  
 for arc in root.findall('arc'):  
 from\_vertex = arc.find('from').text  
 to\_vertex = arc.find('to').text  
 order = int(arc.find('order').text)  
  
 edges.append((int(from\_vertex), int(to\_vertex), order))  
 except Exception:  
 log(f"Ошибка считывания данных из файла")  
 exit()  
  
 unique = set()  
 for x1, x2, \_ in edges:  
 if (x1, x2) in unique:  
 log(f"Ошибка ввода данных. Дуга из вершины {x1} в {x2} уже есть")  
 exit()  
 unique.add((x1, x2))  
  
 for i in range(len(edges)):  
 current = edges[i]  
 for j in range(i + 1, len(edges)):  
 elem = edges[j]  
 if current[1] == elem[1] and current[2] == elem[2]:  
 log(f"Ошибка ввода данных : Одинаковые порядки у элементов {current} и {elem}.")  
 exit()  
  
 return edges  
  
  
def find\_sink(data):  
 for key, value in data.items():  
 if not value['out']:  
 return key  
  
  
def build\_function(node, adj\_list):  
 result = []  
  
 for \_, elem in adj\_list.get(node)['in'].items():  
 if elem == node:  
 continue  
 else:  
 result.append(  
 f"{elem}({build\_function(elem, adj\_list)})" if adj\_list.get(elem)['in'] else f"{elem}")  
 return ",".join(result) if result else ""  
  
  
def get\_fun\_by\_graph(path):  
 graph = get\_data(path)  
  
 if check\_cycle(graph):  
 log("Ошибка. В графе есть цикл.")  
 else:  
 adj\_list = {}  
 for a, b, n in graph:  
 if a not in adj\_list:  
 adj\_list[a] = {'out': [], 'in': {}}  
 if b not in adj\_list:  
 adj\_list[b] = {'out': [], 'in': {}}  
  
 adj\_list[a]['out'].append((b, n))  
 adj\_list[b]['in'][n] = a  
  
 sink = find\_sink(adj\_list)  
  
 return f"{sink}({build\_function(sink, adj\_list)})"  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 parser = argparse.ArgumentParser(  
 prog="nntask2.exe",  
 description="Creating a directed graph")  
 parser.add\_argument("-input1", nargs=1, default=["input.xml"])  
 parser.add\_argument("-input2", nargs=1, default=["input2.xml"])  
 parser.add\_argument("-output1", nargs=1, default=["output.xml"])  
 parser.add\_argument("-output2", nargs=1, default=["output2.xml"])  
  
 args = parser.parse\_args()  
  
 first\_input = args.input1[0]  
 second\_input = args.input2[0]  
 first\_output = args.output1[0]  
 second\_output = args.output2[0]  
  
 result\_str = get\_fun\_by\_graph(first\_input)  
  
 with open(first\_output, "w") as file:  
 file.write(result\_str)

**Задание 3. Вычисление значения функции на графе.**

**На входе:**

1. Текстовый файл с описанием графа в виде списка дуг (смотри задание 1).
2. Текстовый файл соответствий арифметических операций именам вершин:

{

a\_1 : операция\_1

a\_2 : операция\_2

...

a\_n : операция\_n

}

где a\_i – имя i-й вершины, операция\_i – символ операции, соответствующий вершине a\_i.

Допустимы следующие символы операций:

+ – cумма значений,

\* – произведение значений,

*exp* – экспонирование входного значения,

*число* – любая числовая константа.

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

**Способ проверки результата:** результат вычисления, выведенный в файл.

**Параметры запуска программы.**

.\nntask3.exe -input1 .\input1.txt -input2 .\input2.txt

**Пример 1.**

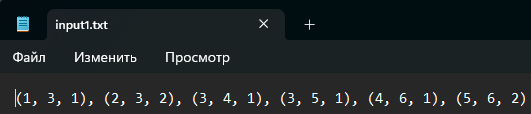
****

Рис. 13 – Входные данные программы (файл input1.txt).

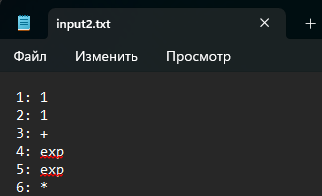
****

Рис. 14 – Входные данные программы (файл input2.txt).

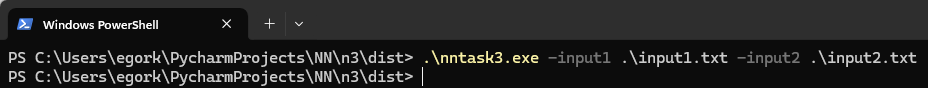
****

Рис. 15 – Запуск программы.

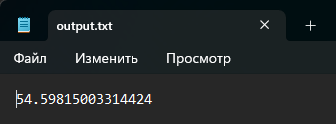


Рис. 16 – Результат работы программы.

**Пример 2.**

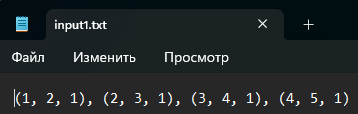
****

Рис. 17 – Входные данные программы (файл input1.txt).

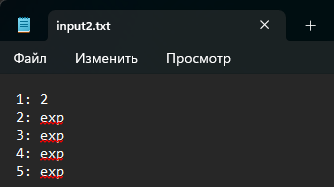
****

Рис. 18 – Входные данные программы (файл input2.txt).

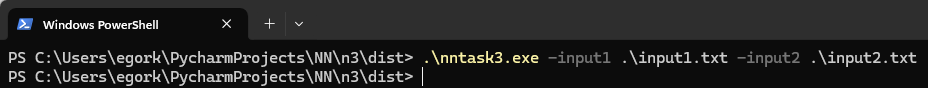
****

Рис. 19 – Запуск программы.

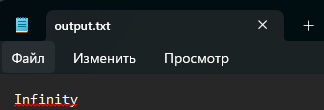


Рис. 20 – Результат работы программы.

**Листинг кода.**

import argparse  
import math  
import os  
  
  
def log(error):  
 log\_file = open("Log.txt", "w", encoding="utf-8")  
 log\_file.write(error)  
  
  
adj = {}  
  
  
def check\_graph(input\_file):  
 if not os.path.isfile(input\_file):  
 log(f"Файл {input\_file} не существует")  
 exit()  
  
 edges = []  
 line = ""  
 try:  
 with open(input\_file, 'r', encoding='utf-8') as file:  
 for st in file:  
 line += st.replace("\n", " ")  
 except Exception:  
 log(f"Ошибка считывания данных из файла")  
 exit()  
  
 edges += (line.replace(" ", "").strip("\n").split("),"))  
  
 edge\_number = 1  
  
 for i in range(len(edges)):  
 try:  
 a, b, n = edges[i].strip(",()").split(",")  
 edges[i] = (int(a), int(b), int(n))  
 except Exception:  
 log(f"Ошибка ввода данных дуги {edge\_number}: Некорректный формат записи.")  
 exit()  
  
 edge\_number += 1  
  
 unique = set()  
 for x1, x2, \_ in edges:  
 if (x1, x2) in unique:  
 log(f"Ошибка ввода данных. Дуга из вершины {x1} в {x2} уже есть")  
 exit()  
 unique.add((x1, x2))  
  
 for i in range(len(edges)):  
 current = edges[i]  
 for j in range(i + 1, len(edges)):  
 elem = edges[j]  
 if current[1] == elem[1] and current[2] == elem[2]:  
 log(f"Ошибка ввода данных : Одинаковые порядки у элементов {current} и {elem}.")  
 exit()  
  
 adj\_list = {}  
 for a, b, n in edges:  
 if a not in adj\_list:  
 adj\_list[a] = {'out': [], 'in': {}}  
 if b not in adj\_list:  
 adj\_list[b] = {'out': [], 'in': {}}  
  
 adj\_list[a]['out'].append((b, n))  
 adj\_list[b]['in'][n] = a  
  
 global adj  
 adj = adj\_list  
  
 return adj\_list  
  
  
def check\_operations(input\_file):  
 if not os.path.isfile(input\_file):  
 log(f"Файл {input\_file} не существует")  
 exit()  
  
 oper = {}  
 try:  
 with open(input\_file, 'r') as f:  
 for line in f:  
 key, value = line.strip().split(':')  
 oper[int(key.strip('a\_'))] = value.strip()  
 except Exception:  
 log(f"Ошибка считывания данных из файла")  
 exit()  
  
 return oper  
  
  
def find\_sink(data):  
 for key, value in data.items():  
 if not value['out']:  
 return key  
  
  
def calculate(node, operation, output):  
 global result  
 try:  
 if adj.get(node)['in']:  
 if operation[node] == '+':  
 result = 0  
 for \_, elem in adj.get(node)['in'].items():  
 result += calculate(elem, operation, output)  
 elif operation[node] == '\*':  
 result = 1  
 for \_, elem in adj.get(node)['in'].items():  
 result \*= calculate(elem, operation, output)  
 elif operation[node] == 'exp':  
 for \_, elem in adj.get(node)['in'].items():  
 result = math.exp(calculate(elem, operation, output))  
 return result  
 else:  
 return int(operation[node])  
 except OverflowError:  
 with open(output, "w") as file:  
 file.write("Infinity")  
 exit()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 parser = argparse.ArgumentParser(  
 prog="nntask3.exe",  
 description="Creating a directed graph")  
 parser.add\_argument("-input1", nargs=1, default=["input1.txt"])  
 parser.add\_argument("-input2", nargs=1, default=["input2.txt"])  
 parser.add\_argument("-output1", nargs=1, default=["output1.txt"])  
 parser.add\_argument("-output2", nargs=1, default=["output2.txt"])  
  
 args = parser.parse\_args()  
  
 first\_input = args.input1[0]  
 second\_input = args.input2[0]  
 first\_output = args.output1[0]  
 second\_output = args.output2[0]  
  
 graph = check\_graph(first\_input)  
  
 operations = check\_operations(second\_input)  
  
 sink = find\_sink(graph)  
  
 res = calculate(sink, operations, first\_output)  
  
 with open(first\_output, "w") as file:  
 file.write(str(res))

**Задание 4. Построение многослойной нейронной сети.**

**На входе:**

1. Текстовый файл с набором матриц весов межнейронных связей:

М1:[M1[1,1],M1[1,2],...,M1[1,n]],...,[M1[m,1],M1[m,2],...,M1[m,n]]

М2:[M2[1,1],M2[1,2],...,M2[1,n]],...,[M2[m,1],M2[m,2],...,M2[m,n]]

...

Мp:[Mp[1,1], Mp[1,2],..., Mp[1,n]], ..., [Mp[m,1], Mp[m,2],...,Mp[m,n]]

1. Текстовый файл с входным вектором в формате:

x1, x2, …, xn.

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

**Способ проверки результата:** результат вычисления, выведенный в файл.

**Параметры запуска программы.**

.\nntask4.exe -input1 .\input1.txt -input2 .\input2.txt

**Пример 1.**

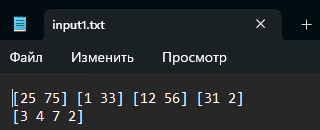
****

Рис. 21 – Входные данные программы (файл input1.txt).

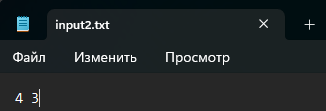
****

Рис. 22 – Входные данные программы (файл input2.txt).

****

Рис. 23 – Запуск программы.

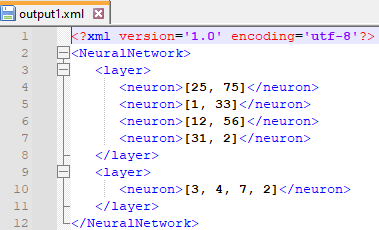


Рис. 24 – Результат работы программы (файл output1.xml).

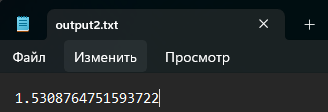


Рис. 25 – Результат работы программы (файл output2.txt).

**Пример 2.**

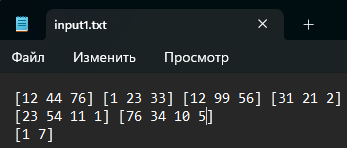
****

Рис. 26 – Входные данные программы (файл input1.txt).

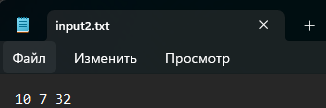
****

Рис. 27 – Входные данные программы (файл input2.txt).

****

Рис. 28 – Запуск программы.

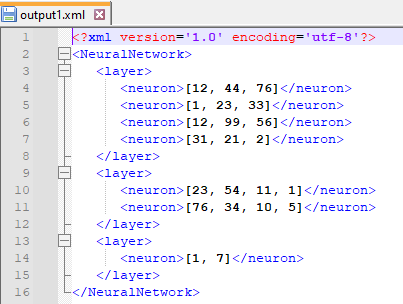


Рис. 29 – Результат работы программы (файл output1.xml).

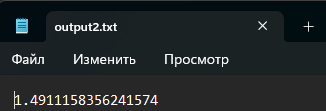


Рис. 30 – Результат работы программы (файл output2.txt).

**Листинг кода.**

import argparse  
import math  
import os  
import xml.etree.ElementTree as ET  
  
  
def log(error):  
 log\_file = open("Log.txt", "w", encoding="utf-8")  
 log\_file.write(error)  
  
  
def load\_matrix(matrix\_file):  
 if not os.path.isfile(matrix\_file):  
 log(f"Файл {matrix\_file} не существует")  
 exit()  
  
 matrix = []  
 try:  
 with open(matrix\_file, 'r') as f:  
 for line in f:  
 lines = []  
 elements = line.strip().split('] [')  
 for element in elements:  
 lines.append([int(x) for x in element.strip("[]").split()])  
 matrix.append(lines)  
 except Exception:  
 log(f"Ошибка считывания данных из файла")  
 exit()  
 return matrix  
  
  
def load\_vector(vector\_file):  
 if not os.path.isfile(vector\_file):  
 log(f"Файл {vector\_file} не существует")  
 exit()  
  
 try:  
 with open(vector\_file, 'r') as f:  
 for line in f:  
 return [int(x) for x in line.split()]  
 except Exception:  
 log(f"Ошибка считывания данных из файла")  
 exit()  
  
def sigmoid(x):  
 return 1 / (1 + math.exp(-x))  
  
def calculate(matrix, vector):  
 new\_matrix = []  
  
 for layer in matrix:  
 tmp = []  
 for neuron in layer:  
 if len(neuron) == len(vector):  
 weighted\_sum = sum(w \* x for w, x in zip(neuron, vector))  
 output = sigmoid(weighted\_sum)  
 tmp.append(output)  
 else:  
 log("Ошибка! Несоответствие размерностей.")  
 exit()  
 new\_matrix.append(tmp)  
 vector = tmp  
  
 return new\_matrix[-1]  
  
  
  
def matrix\_to\_xml(matrix, path):  
 root = ET.Element('NeuralNetwork')  
  
 for layer in matrix:  
 layer\_elem = ET.SubElement(root, 'layer')  
 for neuron in layer:  
 ET.SubElement(layer\_elem, 'neuron').text = str(neuron)  
  
 root = ET.fromstring(ET.tostring(root, encoding='utf-8'))  
 tree = ET.ElementTree(root)  
  
 ET.indent(tree, ' ')  
  
 tree.write(path, encoding="utf-8", xml\_declaration=True)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 parser = argparse.ArgumentParser(  
 prog="nntask4.exe",  
 description="Creating a directed graph")  
 parser.add\_argument("-input1", nargs=1, default=["input1.txt"])  
 parser.add\_argument("-input2", nargs=1, default=["input2.txt"])  
 parser.add\_argument("-output1", nargs=1, default=["output1.xml"])  
 parser.add\_argument("-output2", nargs=1, default=["output2.txt"])  
  
 args = parser.parse\_args()  
  
 first\_input = args.input1[0]  
 second\_input = args.input2[0]  
 first\_output = args.output1[0]  
 second\_output = args.output2[0]  
  
 matrix = load\_matrix(first\_input)  
  
 vector = load\_vector(second\_input)  
  
 result = calculate(matrix, vector)  
  
 matrix\_to\_xml(matrix, first\_output)  
  
 with open(second\_output, "w") as file:  
 file.write(" ".join(str(x) for x in result))

**Задание 5. Реализация метода обратного распространения ошибки для многослойной НС.**

**На входе:**

1. Текстовый файл с описанием НС (формат см. в задании 4).
2. Текстовый файл с обучающей выборкой:

[x11, x12, ..., x1n] -> [y11, y12, ..., y1m]

...

[xk1, xk2, ..., xkn] -> [yk1, yk2, ..., ykm]

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

1. Число итераций обучения (в строке параметров).

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

**Способ проверки результата:** результат вычисления, выведенный в файл.

**Параметры запуска программы.**

.\nntask5.exe -input1 .\input1.txt -input2 .\input2.txt -c 10000

**Пример 1.**

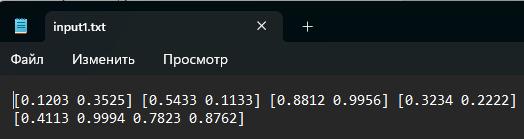
****

Рис. 31 – Входные данные программы (файл input1.txt).

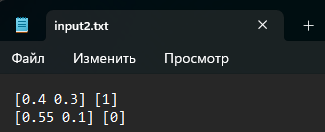
****

Рис. 32 – Входные данные программы (файл input2.txt).

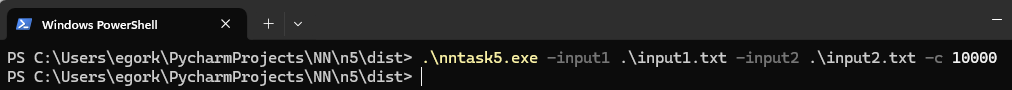
****

Рис. 33 – Запуск программы.



Рис. 34 – Результат работы программы.

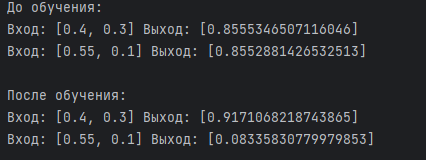
****

Рис. 35 – Результат работы программы.

**Пример 2.**

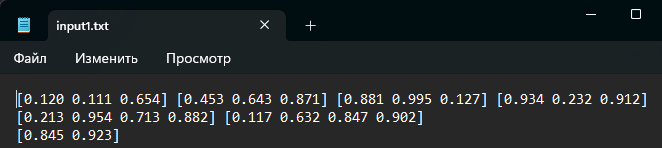
****

Рис. 36 – Входные данные программы (файл input1.txt).

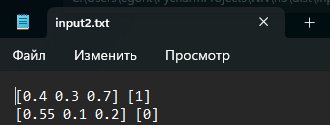
****

Рис. 37 – Входные данные программы (файл input2.txt).

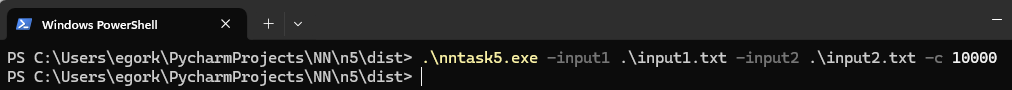
****

Рис. 38 – Запуск программы.

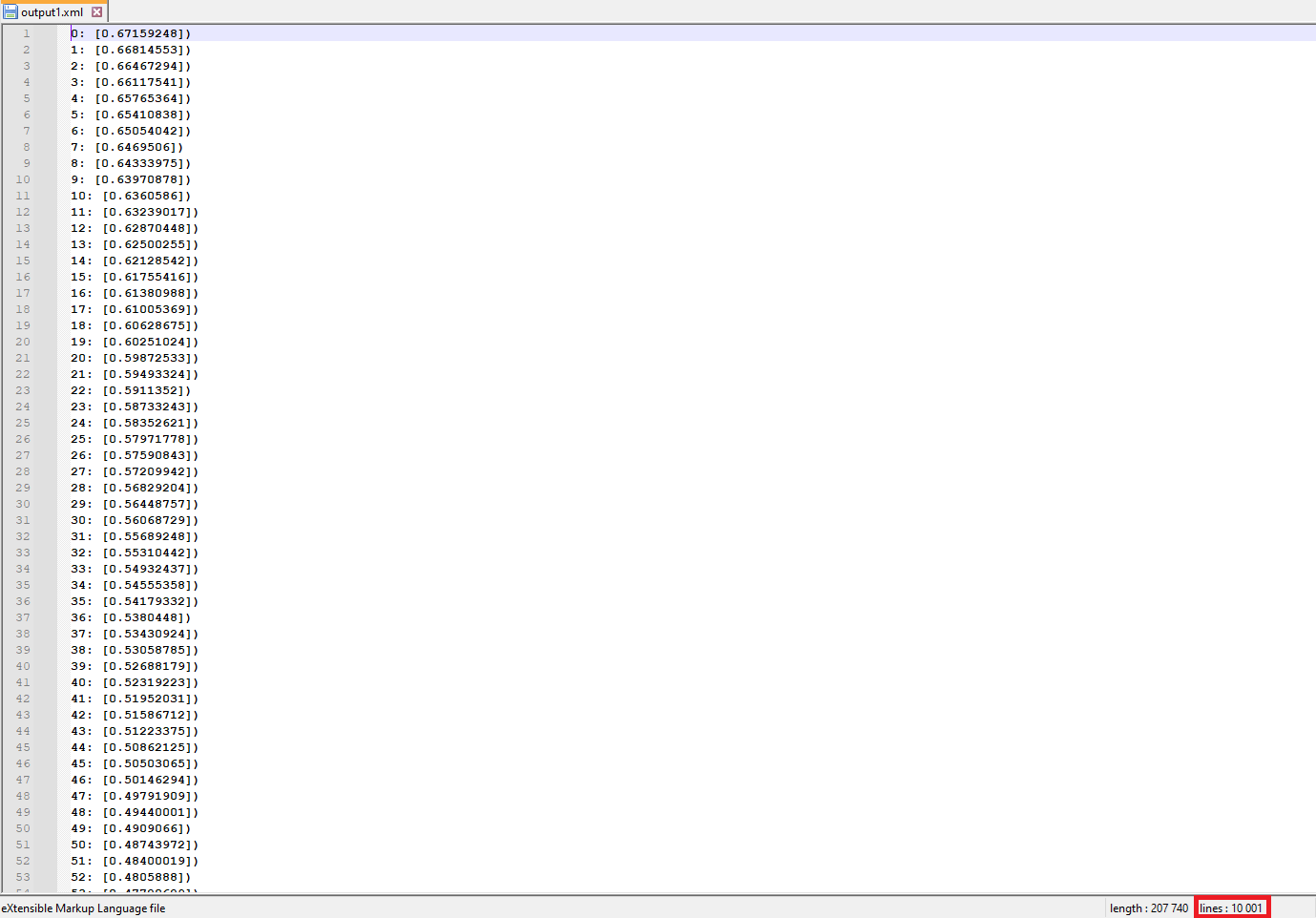


Рис. 39 – Результат работы программы.

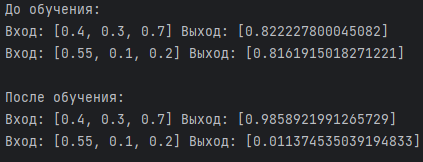
****

Рис. 40 – Результат работы программы.

**Листинг кода.**

import argparse  
import math  
import os  
import numpy as np  
  
  
def log(error):  
 log\_file = open("Log.txt", "w", encoding="utf-8")  
 log\_file.write(error)  
  
  
def load\_matrix(matrix\_file):  
 if not os.path.isfile(matrix\_file):  
 log(f"Файл {matrix\_file} не существует")  
 exit()  
  
 matrix = []  
 try:  
 with open(matrix\_file, 'r') as f:  
 for line in f:  
 lines = []  
 elements = line.strip().split('] [')  
 for element in elements:  
 lines.append([float(x) for x in element.strip("[]").split()])  
 matrix.append(lines)  
 except Exception:  
 log(f"Ошибка считывания данных из файла")  
 exit()  
 return matrix  
  
  
def load\_vector(vector\_file):  
 if not os.path.isfile(vector\_file):  
 log(f"Файл {vector\_file} не существует")  
 exit()  
  
 try:  
 xArray = []  
 yArray = []  
 with open(vector\_file, 'r') as f:  
 for line in f:  
 res = line.strip("[]\n").split("] [")  
 xArray.append([float(x) for x in res[0].split()])  
 yArray.append([float(y) for y in res[1].split()])  
 return xArray, yArray  
 except Exception:  
 log(f"Ошибка считывания данных из файла")  
 exit()  
  
  
def sigmoid(x):  
 return 1 / (1 + math.exp(-x))  
  
  
def sigmoid\_derivative(x):  
 return x \* (1 - x)  
  
  
def mean\_squared\_error(act, pred):  
 pred = np.asarray(pred)  
 act = np.asarray(act)  
 diff = pred - act  
 differences\_squared = diff \*\* 2  
 mean\_diff = sum(0.5 \* differences\_squared)  
  
 return mean\_diff  
  
class Neuron:  
 def \_\_init\_\_(self, weights):  
 self.weights = weights  
 self.activation\_function = sigmoid  
 self.activation\_derivative = sigmoid\_derivative  
 self.output = None  
  
 def activate(self, inputs):  
 z = 0  
 for i in range(len(inputs)):  
 z += self.weights[i] \* inputs[i]  
 self.output = self.activation\_function(z)  
 return self.output  
  
  
class Layer:  
 def \_\_init\_\_(self, neurons):  
 self.neurons = neurons  
 self.output = None  
 self.error = None  
 self.delta = None  
  
 def forward(self, inputs):  
 self.output = [neuron.activate(inputs) for neuron in self.neurons]  
 return self.output  
  
  
class Network:  
 def \_\_init\_\_(self, layers):  
 self.layers = layers  
  
 def forward(self, X):  
 for layer in self.layers:  
 X = layer.forward(X)  
 return X  
  
 def backward(self, X, y, learning\_rate):  
 output = self.forward(X)  
  
 error = [y[0] - output[0]]  
 self.layers[-1].error = error  
 self.layers[-1].delta = [error[0] \* self.layers[-1].neurons[0].activation\_derivative(output[0])]  
  
 for i in reversed(range(len(self.layers) - 1)):  
 layer = self.layers[i]  
 next\_layer = self.layers[i + 1]  
 layer.error = [0] \* len(layer.neurons)  
 for j, neuron in enumerate(next\_layer.neurons):  
 for k, weight in enumerate(neuron.weights):  
 layer.error[k] += next\_layer.delta[j] \* weight  
  
 layer.delta = [  
 layer.error[j] \* neuron.activation\_derivative(neuron.output)  
 for j, neuron in enumerate(layer.neurons)  
 ]  
  
 for i in range(len(self.layers)):  
 layer = self.layers[i]  
 inputs = X if i == 0 else self.layers[i - 1].output  
 for j, neuron in enumerate(layer.neurons):  
 for k in range(len(neuron.weights)):  
 neuron.weights[k] += learning\_rate \* layer.delta[j] \* inputs[k]  
  
 return error  
  
 def learning(self, X, y, learning\_rate, epochs):  
 errors = []  
 for epoch in range(epochs):  
 cur\_error = []  
 for xi, yi in zip(X, y):  
 cur\_error.append(self.backward(xi, yi, learning\_rate))  
 errors.append(mean\_squared\_error(cur\_error, y))  
  
 return errors  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 parser = argparse.ArgumentParser(  
 prog="nntask5.exe",  
 description="Creating a directed graph")  
 parser.add\_argument("-input1", nargs=1, default=["input1.txt"])  
 parser.add\_argument("-input2", nargs=1, default=["input2.txt"])  
 parser.add\_argument("-output1", nargs=1, default=["output1.xml"])  
 parser.add\_argument("-c", nargs=1, default=-1)  
  
 args = parser.parse\_args()  
  
 first\_input = args.input1[0]  
 second\_input = args.input2[0]  
 first\_output = args.output1[0]  
 epochs = int(args.c[0])  
  
 matrix = load\_matrix(first\_input)  
  
 network\_layers = []  
  
 for layers in matrix:  
 layer = []  
 for neuron in layers:  
 layer.append(Neuron(neuron))  
 network\_layers.append(Layer(layer))  
  
 network = Network(network\_layers)  
  
 xArray, yArray = load\_vector(second\_input)  
  
 print("До обучения:")  
 for xi in xArray:  
 output = network.forward(xi)  
 print("Вход:", xi, "Выход:", output)  
  
 print()  
  
 result = network.learning(xArray, yArray, 0.1, epochs)  
  
 print("После обучения:")  
 for xi in xArray:  
 output = network.forward(xi)  
 print("Вход:", xi, "Выход:", output)  
  
 with open(first\_output, "w") as file:  
 for i, elem in enumerate(result):  
 file.write(f"{i}: {elem})\n")