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

*Дусалиева Тахира Ахатовича*

фамилия, имя, отчество

Научный руководитель

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

подпись, дата

Саратов 2015

**СОДЕРЖАНИЕ**

[Задание 1. Создание ориентированного графа 3](#_Toc184282203)

[Задание 2. Создание функции по графу 8](#_Toc184282204)

[Задание 3. Вычисление значения функции на графе 10](#_Toc184282205)

[Задание 4. Построение многослойной нейронной сети 13](#_Toc184282206)

[Задание 5. Реализация метода обратного распространения ошибки для многослойной НС 16](#_Toc184282207)

[ПРИЛОЖЕНИЕ А 20](#_Toc184282208)

[ПРИЛОЖЕНИЕ Б 26](#_Toc184282209)

[ПРИЛОЖЕНИЕ В 33](#_Toc184282210)

[ПРИЛОЖЕНИЕ Г 41](#_Toc184282211)

[ПРИЛОЖЕНИЕ Д 45](#_Toc184282212)

# Задание 1. Создание ориентированного графа

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

(, , ), (, , ), ..., (, , ),

где – начальная вершина дуги , - конечная вершина дуги , - порядковый номер дуги в списке всех заходящих в вершину 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.

**Описание работы программы**

Шаг 1. Открывается файл с текстовым описанием графа, после чего выбираются все данные (дуги), что соответствуют следующему формату при этом игнорируются все пробелы, все текстовые символы между дугами. Достигается это с помощью регулярных выражений.

Шаг 2. Полученные данные форматируются, создаются экземпляры класса Vertex для и , где описывается количество дуг, которые в них заходят и исходят. Создаётся экземпляр класса Arc для вершин и и , при этом увеличивая соответствующие счётчики исходящих и заходящих дуг в вершинах. Если не соответствует количеству заходящих дуг в вершине , то соотвествующая запись заносится в файл с ошибками, а последующая сериализация графа прерывается.

Шаг 3. Создаётся экземпляр класса Graph с сортированным списком вершин Vertex и списком дуг Arc, с тем же порядком, что указан в файле. По этому графу проводится сериализация в xml файл.

**Примеры исполнения**

|  |
| --- |
| Рисунок 1 – Ориентированный граф 1 |

Вид этого графа в сериализованном виде:

<?xml version="1.0" encoding="utf-8"?>

<graph>

    <vertex>v1</vertex>

    <vertex>v2</vertex>

    <vertex>v3</vertex>

    <arc>

        <from>v1</from>

        <to>v2</to>

        <order>2</order>

    </arc>

    <arc>

        <from>v3</from>

        <to>v2</to>

        <order>1</order>

    </arc>

    <arc>

        <from>v2</from>

        <to>v3</to>

        <order>1</order>

    </arc>

    <arc>

        <from>v3</from>

        <to>v1</to>

        <order>1</order>

    </arc>

</graph>

|  |
| --- |
| Рисунок 2 – Ориентированный граф 2 |

Вид этого графа в сериализованном виде:

<?xml version="1.0" encoding="utf-8"?>

<graph>

    <vertex>v1</vertex>

    <vertex>v2</vertex>

    <vertex>v3</vertex>

    <vertex>v4</vertex>

    <vertex>v5</vertex>

    <vertex>v6</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>v1</from>

        <to>v3</to>

        <order>2</order>

    </arc>

    <arc>

        <from>v3</from>

        <to>v4</to>

        <order>1</order>

    </arc>

    <arc>

        <from>v1</from>

        <to>v4</to>

        <order>2</order>

    </arc>

    <arc>

        <from>v4</from>

        <to>v5</to>

        <order>1</order>

    </arc>

    <arc>

        <from>v4</from>

        <to>v6</to>

        <order>1</order>

    </arc>

    <arc>

        <from>v5</from>

        <to>v6</to>

        <order>2</order>

    </arc>

    <arc>

        <from>v2</from>

        <to>v5</to>

        <order>2</order>

    </arc>

</graph>

Вид графов в программе::

|  |
| --- |
| Рисунок 3 – Входные файлы задания 1 |
| Рисунок 4 – Исполнение программы задания 1 |
| Рисунок 5 – Результат работы программы задания 1 |

Исходный код программы указан в приложении А.

# 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:

Способ проверки результата:

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

**Описание работы программы**

Шаг 1. Распаковывается файл с графом в формате xml и создается экземпляр объекта Graph, состоящий из подклассов Arc и Vertex описывающие дуги и вершины соответственно.

Шаг 2. Строится префиксная запись по следующим шагам:

Шаг 2.1. Graph проверяется на присутствие циклов с помощью метода поиска в глубину (DFS). Если циклы присутствуют, то в файл с ошибками добавляется соответствующая запись, а процесс построения префиксной записи прерывается.

Шаг 2.2. Идёт поиск корневой вершины, то есть той вершины, у которой нет заходящих в неё дуг. Если такой вершины нет, то в файл с ошибками добавляется соответствующая запись, а процесс построения префиксной записи прерывается.

Шаг 2.3. Строится префиксная запись.

Шаг 3. Префиксная запись записывается в файл.

**Примеры исполнения**

Для ориентированного графа 1 из рисунка 1 префиксной записи не существует, поскольку присутствуют цикл.

Для ориентированного графа 2 из рисунка 2 префиксная запись будет выглядеть так.

Вид из программы:

|  |
| --- |
| Рисунок 6 – Входные данные задания 2    Рисунок 7 – Исполнение программы задания 2    Рисунок 8 – Результат работы программы задания 2 |

Исходный код программы указан в приложении Б.

# 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:

{

...

},

где – имя -й вершины, – символ операции, соответствующий вершине .

Допустимы следующие символы операций:

– cумма значений,

– произведение значений,

– экспонирование входного значения,

– любая числовая константа.

На выходе: значение функции, построенной по графу a) и файлу b).

Способ проверки результата: результат вычисления, выведенный в файл.

**Описание работы программы**

Шаг 1. Распаковывается файл с графом в формате xml и создается экземпляр объекта Graph, состоящий из подклассов Arc и Vertex описывающие дуги и вершины соответственно.

Шаг 2. Распаковывается файл с операциями и вершинами, которым эти операции присвоены. В классе Graph соответствующим Vertex присваиваются операции.

Шаг 3. Строится префиксная запись по следующим шагам:

Шаг 3.1. Graph проверяется на присутствие циклов с помощью метода поиска в глубину (DFS). Если циклы присутствуют, то в файл с ошибками добавляется соответствующая запись, а процесс построения префиксной записи прерывается.

Шаг 3.2. Идёт поиск корневой вершины, то есть той вершины, у которой нет заходящих в неё дуг. Если такой вершины нет, то в файл с ошибками добавляется соответствующая запись, а процесс построения префиксной записи прерывается.

Шаг 3.3. Строится префиксная запись, где вместо вершин присутствуют их операции.

Шаг 4. Вычисляется значение префиксной записи с операциями и записывается в файл.

**Пример исполнения**

|  |
| --- |
| Рисунок 9 – Ориентированный граф 2  с определёнными на ними операциями |

Для графа из рисунка 9 вычисление значения функции на этом графе будет выглядеть следующим образом:

Вид из программы:

|  |
| --- |
| Рисунок 10 – Входные данные из задания 3    Рисунок 11 – Исполнение программы из задания 3    Рисунок 12 – Результат работы программы из задания 3 |

Исходный код программы указан в приложении В.

# Задание 4. Построение многослойной нейронной сети

На входе:

а) Текстовый файл с набором матриц весов межнейронных связей:

б) Текстовый файл с входным вектором в формате:

На выходе:

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

Файл с выходным вектором – результатом вычислений НС в формате:

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

**Описание работы программы**

Шаг 1. Открывается файл с несериализированной НС, после чего данные разделяются на слои. Создаётся экземпляр класса Network с именнованными слоями , , если слои в соответствии с порядком в текстовом файле и имеют связности друг другом, то соответствующая запись записывается в файл с ошибками и дальнейший процесс вычисления выходного вектора прерывается. Сериализованная НС записывается в файл.

Шаг 2. Открывается файл с входным вектором .

Шаг 3. Последовательно вычисляются выходные векторы . Входной вектор , где это выходной вектор к которому была применена процедура .

Шаг 4. Последний выходной вектор записывается в файл.

**Пример исполнения**

Возьмём НС вида:

и входной вектор:

Тогда вычисление выходного вектора будет выглядеть следующим образом:

Выходной вектор равен:

Вид в программе:

|  |
| --- |
| Рисунок 13 – Входные данные из задания 4    Рисунок 14 – Исполнение программы из задания 4    Рисунок 15 – Результат работы программы из задания 4 |

Исходный код программы указан в приложении Г.

# Задание 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:

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

в) Текстовый файл с параметрами обучения (колличество итераций, скорость обучения, погрешность)

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

1 : Ошибка1

2 : Ошибка2

...

N : ОшибкаN

**Описание работы программы**

Шаг 1. Распаковываются файлы с сериализованной НС, обучающей выборкой и параметрами.

Шаг 2. На вход НС подаётся обучающий пример (один из входных векторов обучающей выборки). НС вычисляет для каждого слоя выходной вектор и производную , а также дельты (разницы) для каждого слоя по формуле , где – транспонированная матрица , причём для последнего -го слоя формула другая , где - ожидаемый вектор.

Шаг 3. Сравнивается полученный выход с ожидаемым (из обучающей выборки) и вычисляется ошибка по формуле , а также

Шаг 4. Распространить ошибку вверх по сети последовательно умножая ошибку на дельты слоев.

Шаг 5. Изменить веса в слоях, для уменьшения ошибок по следующей формуле , где – коэффициент характеризующий скорость обучения.

Шаг 6. Положить , где – итерация тренировки. Если закончить тренировку и записать список всех ошибок в файл, иначе перейти на шаг 1.

**Пример исполнения**

Допустим необходимо, чтобы НС могла определять число, загоревшееся на экране из таблицы 1.

Таблица 1 – Экран с пронумерованными пикселями

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 |

Например, для числа 8 экран должен гореть как показано в таблице 2.

Таблица 2 – Экран с горящим числом 8

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 |

То есть входной вектор для числа 8 будет и выходной вектор должен быть для него .

Вид в программе:

|  |
| --- |
| Рисунок 16 – Входные данные из задания 5    Рисунок 17 – Исполнение программы из задания 5    Рисунок 18 – Результат работы программы из задания 5 |

Результат окончательный оказался весьма близким к истинным.

X: [0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0.], Y: [0.], out: [0.00987635]

X: [0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1.], Y: [0.1], out: [0.10086522]

X: [0. 1. 1. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 1. 1. 1.], Y: [0.2], out: [0.20167746]

X: [0. 1. 1. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0.], Y: [0.3], out: [0.30199688]

X: [0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 0. 1. 1. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 1.], Y: [0.4], out: [0.40344592]

X: [1. 1. 1. 1. 1. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0.], Y: [0.5], out: [0.48448955]

X: [0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0.], Y: [0.6], out: [0.62387609]

X: [1. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0.], Y: [0.7], out: [0.68975164]

X: [0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0.], Y: [0.8], out: [0.79577815]

X: [0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 0.], Y: [0.9], out: [0.89587261]

it: 25000, error: 5.3374824708575825e-05

Исходный код программы указан в приложении Д.

# ПРИЛОЖЕНИЕ А

Исходный код программы nntask1.exe.

Листинг 1 – Скрипт main.py

import sys

import argparse

from serialization import \*

import os

path\_to\_dir = os.path.dirname(sys.executable)

# path\_to\_dir, \_ = os.path.split(os.path.abspath(\_\_file\_\_))

def format\_file\_path(file):

    head, tail = os.path.split(file)

    if head:

        return file

    else:

        return os.path.join(path\_to\_dir, tail)

def get\_graph(input\_file, output\_file):

    formated\_input\_file = format\_file\_path(input\_file)

    formated\_output\_file = format\_file\_path(output\_file)

    formated\_errors\_file = format\_file\_path(f'errors.txt')

    try:

        serialization(input\_file=formated\_input\_file, output\_file=formated\_output\_file)

        return 1

    except Exception as e:

        with open(formated\_errors\_file, 'a') as f:

            f.write(f'{formated\_input\_file} : {formated\_output\_file}\n{e}\n')

        return -1

def parse\_value(value : str):

    parts = value.split('=')

    if len(parts) != 2:

        raise argparse.ArgumentTypeError('Неверный формат введённых данных')

    return parts[0], parts[1]

if \_\_name\_\_ == '\_\_main\_\_':

    parser = argparse.ArgumentParser()

    parser.add\_argument('params', nargs='+', type=parse\_value)

    args = parser.parse\_args()

    params = dict(args.params)

    input = params.get('input')

    output = params.get('output')

    error\_file = format\_file\_path('errors.txt')

    r = get\_graph(input, output)

    dct = {-1 : 'Попытка завершилась с ошибками',

           1 :'Попытка завершилась успешно'

           }

    print(dct[r])

Листинг 2 – Скрипт serialization.py

import xml.etree.ElementTree as ET

import xml.dom.minidom as minidom

import re

import os.path

from utils import \*

\_\_all\_\_ = ['serialization']

def get\_data(file):

    try:

        with open(file, 'r') as f:

            all\_data = f.read()

    except:

        raise FileNotFoundError(f'FileNotFoundError: Файл {file} не найден')

    match = re.findall(r"\(\s\*?\w+\d+\s\*?,\s\*?\w+\d+\s\*?,\s\*?\d+\s\*?\)", re.sub(r'\s\*', '', all\_data))

    if not match:

        raise ValueError(f'ValueError: В файле {file} не указаны или неверно заданы данные')

    graph = Graph()

    sorted\_match = sorted(match, key = lambda x: int(x.split(',')[2].strip(')')))

    arcs = []

    for raw\_data in sorted\_match:

        str\_vertex1, str\_vertex2 = re.findall(r"\b\w+\d+", raw\_data)

        vertex1 = graph.get\_vertex(str\_vertex1)

        if vertex1 is None:

            vertex1 = Vertex(str\_vertex1)

            graph.add\_vertex(vertex1)

        vertex2 = graph.get\_vertex(str\_vertex2)

        if vertex2 is None:

            vertex2 = Vertex(str\_vertex2)

            graph.add\_vertex(vertex2)

        arcs.append(Arc(vertex1, vertex2, int(re.search(r'\b\d+', raw\_data)[0])))

    for raw\_data in match:

        str\_vertex1, str\_vertex2 = re.findall(r"\b\w+\d+", raw\_data)

        vertex1 = graph.get\_vertex(str\_vertex1)

        vertex2 = graph.get\_vertex(str\_vertex2)

        for arc in arcs:

            if arc.arc\_from == vertex1 and arc.arc\_to == vertex2 and arc.order == int(re.search(r'\b\d+', raw\_data)[0]):

                graph.add\_arc(arc)

                break

    return graph

def pretty\_print(xml\_elem):

    rough\_string = ET.tostring(xml\_elem, 'utf-8')

    reparsed = minidom.parseString(rough\_string)

    return reparsed.toprettyxml(indent='    ', encoding='utf-8')

class FileExtensionError(Exception):

    pass

def serialization(input\_file, output\_file):

    if os.path.exists(output\_file):

        raise FileExistsError(f'FileExistsError: Файл {output\_file} уже существует')

    \_, extension = os.path.splitext(output\_file)

    if extension != '.xml':

        raise FileExtensionError(f'FileExtensionError: Выходной файл должен быть формата XML-документа')

    graph = get\_data(input\_file)

    # vertexes, arcs = get\_data(input\_file)

    data\_xml = ET.Element('graph')

    for v in sorted(graph.vertexes, key=lambda x: x.name):

        vertex\_xml = ET.SubElement(data\_xml, 'vertex')

        vertex\_xml.text = v.name

    for arc in graph.arcs:

        arc\_xml = ET.SubElement(data\_xml, 'arc')

        arc\_xml\_from = ET.SubElement(arc\_xml, 'from')

        arc\_xml\_from.text = arc.arc\_from.name

        arc\_xml\_to = ET.SubElement(arc\_xml, 'to')

        arc\_xml\_to.text = arc.arc\_to.name

        arc\_xml\_order = ET.SubElement(arc\_xml, 'order')

        arc\_xml\_order.text = str(arc.order)

    str\_xml = pretty\_print(data\_xml)

    with open(output\_file, 'wb') as f:

        f.write(str\_xml)

Листинг 3 – Скрипт utils.py

class Vertex:

    def \_\_init\_\_(self, name : str):

        self.name = name

        self.arc\_to = 0

        self.arc\_from = 0

    def \_\_str\_\_(self):

        return f'Vertex({self.name})'

    def \_\_repr\_\_(self):

        return f'Vertex({self.name})'

class OrderError(Exception):

    pass

class Arc:

    def \_\_init\_\_(self, vertex1 : Vertex, vertex2 : Vertex, order : int):

        self.arc\_from = vertex1

        self.arc\_to = vertex2

        self.order = order

        self.arc\_from.arc\_from += 1

        self.arc\_to.arc\_to += 1

        if self.order != self.arc\_to.arc\_to:

            raise OrderError(f'OrderError: Порядок дуги ({self.arc\_from.name}, {self.arc\_to.name}, {self.order}) указан неверно')

    def \_\_str\_\_(self):

        return f'Arc(from: {self.arc\_from}, to: {self.arc\_to})'

    def \_\_repr\_\_(self):

        return f'Arc(from: {self.arc\_from}, to: {self.arc\_to})'

    def \_get\_vertexes(self):

        return self.arc\_from, self.arc\_to

class QuadMatrixError(Exception):

    pass

class QuadMatrix:

    def \_\_init\_\_(self, args : list):

        if len(args) != len(args[0]):

            raise QuadMatrixError('QuadMatrixError: На вход попал список из которого невозмоно создать квадратную матрицу')

        self.args = args

    def \_\_len\_\_(self):

        return len(self.args)

    def \_\_getitem\_\_(self, index : int):

        return self.args[index]

    def \_\_setitem\_\_(self, index : int, value):

        self.args[index] = value

    def \_\_str\_\_(self):

        return '\n'.join(map(lambda x: ' '.join(map(lambda y: str(y), x)), self.args))

    def \_\_repr\_\_(self):

        return f'Matrix({self.args})'

    def transpose(self):

        self.args = [list(row) for row in zip(\*self.args)]

        return self

class GraphHasNotVertex(Exception):

    pass

class Graph:

    def \_\_init\_\_(self, vertexes : list = None, arcs : list = None):

        if vertexes is not None:

            self.vertexes = set(self.from\_list(vertexes, Vertex))

        else:

            self.vertexes = set()

        if arcs is not None:

            self.arcs = self.from\_list(arcs, Arc)

        else:

            self.arcs = list()

    def from\_list(self, li : list, obj):

        if all(map(lambda x: isinstance(x, obj), li)):

            return li

    def has\_vertex(self, other : str):

        if other in map(lambda x: x.name, self.vertexes):

            return True

        return False

    def get\_vertex(self, other : str):

        if self.has\_vertex(other):

            for vertex in self.vertexes:

                if vertex.name == other:

                    return vertex

    def add\_vertex(self, other):

        if isinstance(other, Vertex):

            self.vertexes.add(other)

        else:

            raise ValueError('ValueError: На вход попал объект отличный от Vertex')

    def pop\_vertex(self, other):

        if isinstance(other, Vertex):

            self.vertexes.discard(other)

            remove\_arc\_indexes = []

            for index, arc in enumerate(self.arcs):

                vertex1, vertex2 = arc.\_get\_vertexes()

                if vertex1 == other or vertex2 == other:

                    remove\_arc\_indexes.append(index)

            map(lambda x: self.arcs.pop(x), remove\_arc\_indexes)

        else:

            raise ValueError('ValueError: На вход попал объект отличный от Vertex')

    def add\_arc(self, other):

        if isinstance(other, Arc):

            vertex1, vertex2 = other.\_get\_vertexes()

            if vertex1 not in self.vertexes and vertex2 not in self.vertexes:

                raise GraphHasNotVertex(f'GraphHasNotVertex: Вершины {vertex1} или {vertex2} нет в Graph')

            self.arcs.append(other)

        else:

            raise ValueError('ValueError: На вход попал не объект Arc')

    def pop\_arc(self, index : int = -1):

        self.arcs.pop(index)

    def \_\_str\_\_(self):

        dct = {'vertexes' : sorted(self.vertexes, key=lambda x: x.name),

               'arcs' : self.arcs}

        return f'Graph({dct})'

    def \_\_repr\_\_(self):

        dct = {'vertexes' : sorted(self.vertexes, key=lambda x: x.name),

               'arcs' : self.arcs}

        return f'Graph({dct})'

    def get\_matrix\_adjency(self):

        n = len(self.vertexes)

        self.adj = QuadMatrix([[0 for \_ in range(n)] for \_ in range(n)])

        dct = dict(map(lambda x: (x[1], x[0]), enumerate(sorted(self.vertexes, key=lambda x: x.name))))

        for arc in self.arcs:

            self.adj[dct[arc.arc\_from]][dct[arc.arc\_to]] += 1

        return self

# ПРИЛОЖЕНИЕ Б

Исходный код программы nntask2.exe.

Листинг 1 – Скрипт main.py

import sys

import re

import os

from cycles import prefix\_function

import argparse

path\_to\_dir = os.path.dirname(sys.executable)

# path\_to\_dir, \_ = os.path.split(os.path.abspath(\_\_file\_\_))

def format\_file\_path(file):

    head, tail = os.path.split(file)

    if head:

        return file

    else:

        return os.path.join(path\_to\_dir, tail)

def parse\_value(value : str):

    parts = value.split('=')

    if len(parts) != 2:

        raise argparse.ArgumentTypeError('Неверный формат введённых данных')

    return parts[0], parts[1]

def get\_prefix\_notation(input\_file, output\_file):

    formated\_input\_file = format\_file\_path(input\_file)

    formated\_output\_file = format\_file\_path(output\_file)

    formated\_errors\_file = format\_file\_path('errors\_.txt')

    try:

        prefix\_function(formated\_input\_file, formated\_output\_file)

        return 1

    except Exception as e:

        with open(formated\_errors\_file, 'a') as f:

            f.write(f'{formated\_input\_file} : {formated\_output\_file}\n{e}\n')

        return -1

if \_\_name\_\_ == '\_\_main\_\_':

    parser = argparse.ArgumentParser()

    parser.add\_argument('params', nargs='+', type=parse\_value)

    args = parser.parse\_args()

    params = dict(args.params)

    input = params.get('input')

    output = params.get('output')

    error\_file = format\_file\_path('errors.txt')

    r = get\_prefix\_notation(input, output)

    dct = {-1 : 'Попытка завершилась с ошибками',

           1 :'Попытка завершилась успешно'

           }

    print(dct[r])

Листинг 2 – Скрипт cycles.py

import xml.etree.ElementTree as ET

from utils import Graph, Arc, Vertex

import os.path

class FileExtensionError(Exception):

    pass

def unpack(input\_file):

    \_, extension = os.path.splitext(input\_file)

    if extension != '.xml':

        raise FileExtensionError(f'FileExtensionError: Входящий файл должен быть формата XML-документа')

    try:

        with open(input\_file, 'r') as f:

            xml\_data = f.read()

    except:

        raise FileNotFoundError(f"FileNotFoundError: Файл {input\_file} не был найден")

    tree = ET.fromstring(xml\_data)

    graph = Graph()

    for vertex in tree.findall('vertex'):

        graph.add\_vertex(Vertex(vertex.text))

    for arc in sorted(tree.findall('arc'), key = lambda x: int(x.findall('order')[0].text)):

        from\_vertex = graph.get\_vertex(arc.findall('from')[0].text)

        to\_vertex = graph.get\_vertex(arc.findall('to')[0].text)

        order = int(arc.findall('order')[0].text)

        graph.add\_arc(Arc(from\_vertex, to\_vertex, order))

    return graph

class HasCyclesError(Exception):

    pass

class RootVertexNotFoundError(Exception):

    pass

def prefix\_function(input\_file, output\_file):

    graph = unpack(input\_file)

    has\_cycles = graph.cycles()

    if has\_cycles:

        raise HasCyclesError(f'HasCyclesError: В графе присутствуют циклы')

    root\_vertex = None

    for vertex in graph.vertexes:

        if vertex.arc\_to == 0:

            root\_vertex = vertex

    if root\_vertex is None:

        raise RootVertexNotFoundError(f'RootVertexNotFoundError: В графе отсутствуют вершины без входящих в них дуг')

    prefix\_notation = build\_prefix\_notation(graph, root\_vertex)

    if os.path.exists(output\_file):

        raise FileExistsError(f"FileExistsError: Файл {output\_file} уже существует")

    with open(output\_file, 'w') as f:

        f.write(strfprefix(prefix\_notation))

def build\_prefix\_notation(graph : Graph, root\_vertex : Vertex):

    result = {root\_vertex : {}}

    for arc in graph.arcs:

        if arc.arc\_from == root\_vertex:

            result[root\_vertex].update(build\_prefix\_notation(graph, arc.arc\_to))

    return result

def strfprefix(prefix\_notation : dict):

    result = []

    for key, value in prefix\_notation.items():

        result.append(key.name)

        if value == {}:

            continue

        result[-1] += f'({strfprefix(value)})'

    return ', '.join(result)

Листинг 3 – Скрипт utils.py

class Vertex:

    def \_\_init\_\_(self, name : str):

        self.name = name

        self.arc\_to = 0

        self.arc\_from = 0

    def \_\_str\_\_(self):

        return f'Vertex({self.name})'

    def \_\_repr\_\_(self):

        return f'Vertex({self.name})'

class OrderError(Exception):

    pass

class Arc:

    def \_\_init\_\_(self, vertex1 : Vertex, vertex2 : Vertex, order : int):

        self.arc\_from = vertex1

        self.arc\_to = vertex2

        self.order = order

        self.arc\_from.arc\_from += 1

        self.arc\_to.arc\_to += 1

        if self.order != self.arc\_to.arc\_to:

            raise OrderError(f'OrderError: Порядок дуги ({self.arc\_from.name}, {self.arc\_to.name}, {self.order}) указан неверно')

    def \_\_str\_\_(self):

        return f'Arc(from: {self.arc\_from}, to: {self.arc\_to})'

    def \_\_repr\_\_(self):

        return f'Arc(from: {self.arc\_from}, to: {self.arc\_to})'

    def \_get\_vertexes(self):

        return self.arc\_from, self.arc\_to

    def \_\_getitem\_\_(self, value : Vertex):

        if value == self.arc\_from:

            return self.arc\_to

class QuadMatrixError(Exception):

    pass

class QuadMatrix:

    def \_\_init\_\_(self, args : list):

        if len(args) != len(args[0]):

            raise QuadMatrixError('QuadMatrixError: На вход попал список из которого невозмоно создать квадратную матрицу')

        self.args = args

    def \_\_len\_\_(self):

        return len(self.args)

    def \_\_getitem\_\_(self, index : int):

        return self.args[index]

    def \_\_setitem\_\_(self, index : int, value):

        self.args[index] = value

    def \_\_str\_\_(self):

        return '\n'.join(map(lambda x: ' '.join(map(lambda y: str(y), x)), self.args))

    def \_\_repr\_\_(self):

        return f'Matrix({self.args})'

    def transpose(self):

        self.args = [list(row) for row in zip(\*self.args)]

        return self

class GraphHasNotVertex(Exception):

    pass

class Graph:

    def \_\_init\_\_(self, vertexes : list = None, arcs : list = None):

        if vertexes is not None:

            self.vertexes = set(self.from\_list(vertexes, Vertex))

        else:

            self.vertexes = set()

        if arcs is not None:

            self.arcs = self.from\_list(arcs, Arc)

        else:

            self.arcs = list()

    def from\_list(self, li : list, obj):

        if all(map(lambda x: isinstance(x, obj), li)):

            return li

    def has\_vertex(self, other : str):

        if other in map(lambda x: x.name, self.vertexes):

            return True

        return False

    def get\_vertex(self, other : str):

        if self.has\_vertex(other):

            for vertex in self.vertexes:

                if vertex.name == other:

                    return vertex

    def add\_vertex(self, other):

        if isinstance(other, Vertex):

            self.vertexes.add(other)

        else:

            raise ValueError('ValueError: На вход попал объект отличный от Vertex')

    def pop\_vertex(self, other):

        if isinstance(other, Vertex):

            self.vertexes.discard(other)

            remove\_arc\_indexes = []

            for index, arc in enumerate(self.arcs):

                vertex1, vertex2 = arc.\_get\_vertexes()

                if vertex1 == other or vertex2 == other:

                    remove\_arc\_indexes.append(index)

            map(lambda x: self.arcs.pop(x), remove\_arc\_indexes)

        else:

            raise ValueError('ValueError: На вход попал объект отличный от Vertex')

    def add\_arc(self, other):

        if isinstance(other, Arc):

            vertex1, vertex2 = other.\_get\_vertexes()

            if vertex1 not in self.vertexes and vertex2 not in self.vertexes:

                raise GraphHasNotVertex(f'GraphHasNotVertex: Вершины {vertex1} или {vertex2} нет в Graph')

            self.arcs.append(other)

        else:

            raise ValueError('ValueError: На вход попал не объект Arc')

    def pop\_arc(self, index : int = -1):

        self.arcs.pop(index)

    def \_\_str\_\_(self):

        dct = {'vertexes' : sorted(self.vertexes, key=lambda x: x.name),

               'arcs' : self.arcs}

        return f'Graph({dct})'

    def \_\_repr\_\_(self):

        dct = {'vertexes' : sorted(self.vertexes, key=lambda x: x.name),

               'arcs' : self.arcs}

        return f'Graph({dct})'

    def get\_matrix\_adjency(self):

        n = len(self.vertexes)

        self.adj = QuadMatrix([[0 for \_ in range(n)] for \_ in range(n)])

        dct = dict(map(lambda x: (x[1], x[0]), enumerate(sorted(self.vertexes, key=lambda x: x.name))))

        for arc in self.arcs:

            self.adj[dct[arc.arc\_from]][dct[arc.arc\_to]] += 1

        return self

    def cycles(self):

        def dfs(vertex : Vertex, visited : set, stack : set):

            visited.add(vertex)

            stack.add(vertex)

            for arc in self.arcs:

                if arc.arc\_from == vertex and arc.arc\_to not in visited:

                    if dfs(arc.arc\_to, visited, stack):

                        return True

                elif arc.arc\_from == vertex and arc.arc\_to in stack:

                    return True

            stack.remove(vertex)

            return False

        visited = set()

        for vertex in self.vertexes:

            if vertex not in visited:

                if dfs(vertex, visited, set()):

                    return True

        return False

# ПРИЛОЖЕНИЕ В

Исходный код программы nntask3.exe.

Листинг 1 – Скрипт main.py

import sys

import os

from calculate import prefix\_function

import argparse

path\_to\_dir = os.path.dirname(sys.executable)

# path\_to\_dir, \_ = os.path.split(os.path.abspath(\_\_file\_\_))

def format\_file\_path(file):

    head, tail = os.path.split(file)

    if head:

        return file

    else:

        return os.path.join(path\_to\_dir, tail)

def get\_prefix\_notation(input\_file, input\_oper\_file, output\_file):

    formated\_input\_file = format\_file\_path(input\_file)

    formated\_input\_oper\_file = format\_file\_path(input\_oper\_file)

    formated\_output\_file = format\_file\_path(output\_file)

    formated\_errors\_file = format\_file\_path('errors\_.txt')

    try:

        prefix\_function(formated\_input\_file, formated\_input\_oper\_file, formated\_output\_file)

        return 1

    except Exception as e:

        with open(formated\_errors\_file, 'a') as f:

            f.write(f'{formated\_input\_file} : {formated\_output\_file}\n{e}\n')

        return 0

def parse\_value(value : str):

    parts = value.split('=')

    if len(parts) != 2:

        raise argparse.ArgumentTypeError('Неверный формат введённых данных')

    return parts[0], parts[1]

if \_\_name\_\_ == '\_\_main\_\_':

    parser = argparse.ArgumentParser()

    parser.add\_argument('params', nargs='+', type=parse\_value)

    args = parser.parse\_args()

    params = dict(args.params)

    input1 = params.get('input1')

    input2 = params.get('input2')

    output = params.get('output')

    error\_file = format\_file\_path('errors.txt')

    r = get\_prefix\_notation(input1, input2, output)

    dct = {-1 : 'Попытка завершилась с ошибками',

           1 :'Попытка завершилась успешно'

           }

    print(dct[r])

Листинг 2 – Скрипт calculate.py

import xml.etree.ElementTree as ET

from utils import Graph, Arc, Vertex

import os.path

import re

from math import exp

class FileExtensionError(Exception):

    pass

\_NUMBER\_MATCH = re.compile(

    r"-?\b\d+\.?\d\*|\.\d+\b"

)

\_INT\_MATCH = re.compile(

    r"-?\b\d+"

)

def unpack(input\_file):

    \_, extension = os.path.splitext(input\_file)

    if extension != '.xml':

        raise FileExtensionError(f'FileExtensionError: Входящий файл должен быть формата XML-документа')

    try:

        with open(input\_file, 'r') as f:

            xml\_data = f.read()

    except:

        raise FileNotFoundError(f"FileNotFoundError: Файл {input\_file} не был найден")

    tree = ET.fromstring(xml\_data)

    graph = Graph()

    for vertex in tree.findall('vertex'):

        graph.add\_vertex(Vertex(vertex.text))

    for arc in sorted(tree.findall('arc'), key = lambda x: int(x.findall('order')[0].text)):

        from\_vertex = graph.get\_vertex(arc.findall('from')[0].text)

        to\_vertex = graph.get\_vertex(arc.findall('to')[0].text)

        order = int(arc.findall('order')[0].text)

        graph.add\_arc(Arc(from\_vertex, to\_vertex, order))

    return graph

def summarize(\*args):

    res = 0

    for ch in args:

        res += ch

    return res

def multiplie(\*args):

    res = 1

    for ch in args:

        res \*= ch

    return res

def my\_exp(\*args):

    res = []

    for ch in args:

        res.append(exp(ch))

    return res

def get\_number(s):

    if \_NUMBER\_MATCH.match(s):

        if \_INT\_MATCH.match(s):

            return int(s)

        else:

            return float(s)

def get\_operations(input\_file, graph : Graph):

    try:

        with open(input\_file, 'r') as f:

            operations\_data = f.read()

    except:

        raise FileNotFoundError(f"FileNotFoundError: Файл {input\_file} не был найден")

    match = map(lambda x: x, re.findall(r'(\b\w+\d+)\s\*:\s\*([+\*]|exp|(-?\b\d+\.?\d\*|\.\d+\b))', operations\_data))

    if not match:

        raise ValueError(f'ValueError: В файле {input\_file} не указаны или неверно заданы данные')

    operations\_dct = {

        '+' : summarize,

        '\*' : multiplie,

        'exp' : my\_exp,

        'NUMBER' : get\_number

    }

    for raw\_data in match:

        vertex = raw\_data[0]

        v = graph.get\_vertex(vertex)

        if v is None:

            raise ValueError(f'ValueError: Вершины {vertex} не существует в графе')

        operation = raw\_data[1]

        if operation in operations\_dct.keys():

            v.init\_operation((operation, operations\_dct[operation]))

        else:

            v.init\_value(operations\_dct['NUMBER'](operation))

    return graph

class HasCyclesError(Exception):

    pass

class RootVertexNotFoundError(Exception):

    pass

def prefix\_function(input\_graph\_file, input\_oper\_file, output\_file):

    graph = unpack(input\_graph\_file)

    has\_cycles = graph.cycles()

    if has\_cycles:

        raise HasCyclesError(f'HasCyclesError: В графе присутствуют циклы')

    root\_vertex = None

    for vertex in graph.vertexes:

        if vertex.arc\_to == 0:

            root\_vertex = vertex

    if root\_vertex is None:

        raise RootVertexNotFoundError(f'RootVertexNotFoundError: В графе отсутствуют вершины без входящих в них дуг')

    graph = get\_operations(input\_oper\_file, graph)

    prefix\_notation = build\_prefix\_notation(graph, root\_vertex)

    str\_oper\_prefix\_notation = strfprefix(prefix\_notation)

    result = calculate\_prefix(prefix\_notation, root\_vertex)

    if os.path.exists(output\_file):

        raise FileExistsError(f"FileExistsError: Файл {output\_file} уже существует")

    with open(output\_file, 'w') as f:

        f.write(str\_oper\_prefix\_notation + ' = ' + str(result))

def calculate\_prefix(prefix\_notation, root\_vertex : Vertex):

    result = None

    if root\_vertex.value is not None:

        result = root\_vertex.value

    else:

        tmp = []

        for key in prefix\_notation[root\_vertex].keys():

            x = calculate\_prefix(prefix\_notation[root\_vertex], key)

            if isinstance(x, list):

                tmp.extend(x)

            else:

                tmp.append(x)

        result = root\_vertex.operation[1](\*tmp)

    return result

def build\_prefix\_notation(graph : Graph, root\_vertex : Vertex):

    result = {root\_vertex : {}}

    for arc in graph.arcs:

        if arc.arc\_from == root\_vertex:

            result[root\_vertex].update(build\_prefix\_notation(graph, arc.arc\_to))

    return result

def strfprefix(prefix\_notation : dict):

    result = []

    for key, value in prefix\_notation.items():

        if key.operation is not None:

            result.append(key.operation[0])

        else:

            result.append(str(key.value))

        if value == {}:

            continue

        result[-1] += f'({strfprefix(value)})'

    return ', '.join(result)

Листинг 3 – Скрипт utils.py

class Vertex:

    def \_\_init\_\_(self, name : str):

        self.name = name

        self.arc\_to = 0

        self.arc\_from = 0

        self.operation = None

        self.value = None

    def \_\_str\_\_(self):

        return f'Vertex({self.name})'

    def \_\_repr\_\_(self):

        return f'Vertex({self.name})'

    def init\_operation(self, operation):

        self.operation = operation

    def init\_value(self, value):

        self.value = value

class OrderError(Exception):

    pass

class Arc:

    def \_\_init\_\_(self, vertex1 : Vertex, vertex2 : Vertex, order : int):

        self.arc\_from = vertex1

        self.arc\_to = vertex2

        self.order = order

        self.arc\_from.arc\_from += 1

        self.arc\_to.arc\_to += 1

        if self.order != self.arc\_to.arc\_to:

            raise OrderError(f'OrderError: Порядок дуги ({self.arc\_from.name}, {self.arc\_to.name}, {self.order}) указан неверно')

    def \_\_str\_\_(self):

        return f'Arc(from: {self.arc\_from}, to: {self.arc\_to})'

    def \_\_repr\_\_(self):

        return f'Arc(from: {self.arc\_from}, to: {self.arc\_to})'

    def \_get\_vertexes(self):

        return self.arc\_from, self.arc\_to

    def \_\_getitem\_\_(self, value : Vertex):

        if value == self.arc\_from:

            return self.arc\_to

class QuadMatrixError(Exception):

    pass

class QuadMatrix:

    def \_\_init\_\_(self, args : list):

        if len(args) != len(args[0]):

            raise QuadMatrixError('QuadMatrixError: На вход попал список из которого невозмоно создать квадратную матрицу')

        self.args = args

    def \_\_len\_\_(self):

        return len(self.args)

    def \_\_getitem\_\_(self, index : int):

        return self.args[index]

    def \_\_setitem\_\_(self, index : int, value):

        self.args[index] = value

    def \_\_str\_\_(self):

        return '\n'.join(map(lambda x: ' '.join(map(lambda y: str(y), x)), self.args))

    def \_\_repr\_\_(self):

        return f'Matrix({self.args})'

    def transpose(self):

        self.args = [list(row) for row in zip(\*self.args)]

        return self

class GraphHasNotVertex(Exception):

    pass

class Graph:

    def \_\_init\_\_(self, vertexes : list = None, arcs : list = None):

        if vertexes is not None:

            self.vertexes = set(self.from\_list(vertexes, Vertex))

        else:

            self.vertexes = set()

        if arcs is not None:

            self.arcs = self.from\_list(arcs, Arc)

        else:

            self.arcs = list()

    def from\_list(self, li : list, obj):

        if all(map(lambda x: isinstance(x, obj), li)):

            return li

    def has\_vertex(self, other : str):

        if other in map(lambda x: x.name, self.vertexes):

            return True

        return False

    def get\_vertex(self, other : str):

        if self.has\_vertex(other):

            for vertex in self.vertexes:

                if vertex.name == other:

                    return vertex

    def add\_vertex(self, other):

        if isinstance(other, Vertex):

            self.vertexes.add(other)

        else:

            raise ValueError('ValueError: На вход попал объект отличный от Vertex')

    def pop\_vertex(self, other):

        if isinstance(other, Vertex):

            self.vertexes.discard(other)

            remove\_arc\_indexes = []

            for index, arc in enumerate(self.arcs):

                vertex1, vertex2 = arc.\_get\_vertexes()

                if vertex1 == other or vertex2 == other:

                    remove\_arc\_indexes.append(index)

            map(lambda x: self.arcs.pop(x), remove\_arc\_indexes)

        else:

            raise ValueError('ValueError: На вход попал объект отличный от Vertex')

    def add\_arc(self, other):

        if isinstance(other, Arc):

            vertex1, vertex2 = other.\_get\_vertexes()

            if vertex1 not in self.vertexes and vertex2 not in self.vertexes:

                raise GraphHasNotVertex(f'GraphHasNotVertex: Вершины {vertex1} или {vertex2} нет в Graph')

            self.arcs.append(other)

        else:

            raise ValueError('ValueError: На вход попал не объект Arc')

    def pop\_arc(self, index : int = -1):

        self.arcs.pop(index)

    def \_\_str\_\_(self):

        dct = {'vertexes' : sorted(self.vertexes, key=lambda x: x.name),

               'arcs' : self.arcs}

        return f'Graph({dct})'

    def \_\_repr\_\_(self):

        dct = {'vertexes' : sorted(self.vertexes, key=lambda x: x.name),

               'arcs' : self.arcs}

        return f'Graph({dct})'

    def get\_matrix\_adjency(self):

        n = len(self.vertexes)

        self.adj = QuadMatrix([[0 for \_ in range(n)] for \_ in range(n)])

        dct = dict(map(lambda x: (x[1], x[0]), enumerate(sorted(self.vertexes, key=lambda x: x.name))))

        for arc in self.arcs:

            self.adj[dct[arc.arc\_from]][dct[arc.arc\_to]] += 1

        return self

    def cycles(self):

        def dfs(vertex : Vertex, visited : set, stack : set):

            visited.add(vertex)

            stack.add(vertex)

            for arc in self.arcs:

                if arc.arc\_from == vertex and arc.arc\_to not in visited:

                    if dfs(arc.arc\_to, visited, stack):

                        return True

                elif arc.arc\_from == vertex and arc.arc\_to in stack:

                    return True

            stack.remove(vertex)

            return False

        visited = set()

        for vertex in self.vertexes:

            if vertex not in visited:

                if dfs(vertex, visited, set()):

                    return True

        return False

# ПРИЛОЖЕНИЕ Г

Исходный код программы nntask4.exe.

Листинг 1 – Скрипт main.py

import sys

import os.path

from calculateN import \*

from serializationN import \*

import argparse

from datetime import datetime

path\_to\_dir = os.path.dirname(sys.executable)

# path\_to\_dir, \_ = os.path.split(os.path.abspath(\_\_file\_\_))

def format\_file\_path(file):

    head, tail = os.path.split(file)

    if head:

        return file

    else:

        return os.path.join(path\_to\_dir, tail)

def parse\_value(value : str):

    parts = value.split('=')

    if len(parts) != 2:

        raise argparse.ArgumentTypeError('Неверный формат введённых данных')

    return parts[0], parts[1]

if \_\_name\_\_ == '\_\_main\_\_':

    parser = argparse.ArgumentParser()

    parser.add\_argument('params', nargs='+', type=parse\_value)

    args = parser.parse\_args()

    params = dict(args.params)

    input1 = params.get('input1')

    input2 = params.get('input2')

    output = params.get('output')

    error\_file = format\_file\_path('errors.txt')

    try:

        Name\_network = serialization\_network(format\_file\_path(input1))

        Name\_entry = serialization\_entry(format\_file\_path(input2))

        Network(Name\_network, Name\_entry, format\_file\_path(output))

    except Exception as e:

        print(e)

        with open(error\_file, 'a') as f:

            f.write(f'{datetime.now()} : {e}\n')

Листинг 2 – Скрипт serializationN.py

import json

import re

import os.path

def serialization\_network(input\_file):

    try:

        with open(input\_file, 'r') as f:

            data = f.read()

    except:

        raise FileNotFoundError(f'Файл {input\_file} не найден')

    match = re.findall(r'\[(\[.\*\])\]', data)

    if match[0] is None:

        raise ValueError(f'Файл {input\_file} не содержит данных')

    k = 1

    Network = dict()

    for raw\_data in match:

        W = re.findall(r'\[[^\]]\*\]', raw\_data)

        new\_W = list()

        for layer in W:

            numbers = re.findall(r'(-?\b\d+\.?\d\*|\.\d+\b)', layer)

            new\_W.append(list(map(lambda x: float(x), numbers)))

        k\_len = len(str(k))

        layer\_num = str(k)

        if k\_len < len(match) // 10:

            layer\_num = '0' \* (len(match) // 10 - k\_len) + str(k)

        Network[f'W{layer\_num}'] = new\_W

        k += 1

    filename, \_ = os.path.splitext(input\_file)

    output\_file = filename + '.json'

    i = 1

    while os.path.exists(output\_file):

        output\_file = filename + f' ({i})' + '.json'

        i += 1

    with open(output\_file, 'w') as f:

        json.dump(Network, f)

    return output\_file

def serialization\_entry(input\_file):

    try:

        with open(input\_file, 'r') as f:

            data = f.read()

    except:

        raise FileNotFoundError(f'Файл {input\_file} не найден')

    match = re.findall(r'(-?\b\d+\.?\d\*|\.\d+\b)', data)

    if match[0] is None:

        raise ValueError(f'Файл {input\_file} не содержит данных')

    Entry = {'X' : list(map(lambda x: float(x), match))}

    filename, \_ = os.path.splitext(input\_file)

    output\_file = filename + '.json'

    i = 1

    while os.path.exists(output\_file):

        output\_file = filename + f' ({i})' + '.json'

        i += 1

    with open(output\_file, 'w') as f:

        json.dump(Entry, f)

    return output\_file

Листинг 3 – Скрипт calculateN.py

import numpy as np

import json

from math import exp

def sigmoid(x):

    return 1 / (1 + exp(-x))

class Layer:

    def \_\_init\_\_(self, name, W):

        self.name = name

        self.W = np.array(W)

        shape = self.W.shape

        self.n = shape[0]

        self.m = shape[1]

        self.X = np.zeros(self.m)

        self.Y = np.zeros(self.n)

    def set\_x(self, X):

        self.X = np.array(X)

        assert(self.X.shape[0] != self.n)

    def calculate(self):

        print(f'{self.name}:\n{self.W}')

        print(('\t' \* (self.m // 2)) + '\*')

        print('X:' + str(self.X))

        Y = np.dot(self.W, self.X)

        print(('\t' \* (self.m // 2)) + '||')

        print('Y:' + str(Y))

        print('Применяется процедура сигмоида на Y')

        for i in range(self.n):

            Y[i] = sigmoid(Y[i])

        print('Y:' + str(Y))

        self.Y = Y

        return self.Y

class LayerShapeError(Exception):

    pass

class Network:

    def \_\_init\_\_(self, input\_layers, input\_entry, output):

        weights = load(input\_layers)

        self.layers = []

        sorted\_layers = sorted(weights.keys())

        for w in sorted\_layers:

            self.layers.append(Layer(w, weights[w]))

        for w1, w2 in zip(self.layers[:-1], self.layers[1:]):

            if w1.n != w2.m:

                raise LayerShapeError(f'Слой {w1.name} не имеет связности со слоем {w2.name}')

        X = load(input\_entry)

        self.layers[0].X = np.array(X['X'])

        dump(output, {'Y' : list(self.calculate())})

    def \_\_len\_\_(self):

        return len(self.layers)

    def calculate(self):

        k = len(self)

        for i in range(k):

            print('-----------------------------------------------')

            l = self.layers[i]

            new\_x = l.calculate()

            if i < k - 1:

                self.layers[i + 1].X = new\_x

            print('-----------------------------------------------')

        return self.layers[-1].Y

def load(file):

    with open(file, 'r') as f:

        return json.load(f)

def dump(file, data):

    with open(file, 'w') as f:

        json.dump(data, f)

    print(data)

# ПРИЛОЖЕНИЕ Д

Исходный код программы nntask5.exe.

Листинг 1 – Скрипт main.py

import sys

import os.path

from calculateN import \*

import argparse

from datetime import datetime

path\_to\_dir = os.path.dirname(sys.executable)

# path\_to\_dir, \_ = os.path.split(os.path.abspath(\_\_file\_\_))

def format\_file\_path(file):

    head, tail = os.path.split(file)

    if head:

        return file

    else:

        return os.path.join(path\_to\_dir, tail)

def parse\_value(value : str):

    parts = value.split('=')

    if len(parts) != 2:

        raise argparse.ArgumentTypeError('Неверный формат введённых данных')

    return parts[0], parts[1]

if \_\_name\_\_ == '\_\_main\_\_':

    parser = argparse.ArgumentParser()

    parser.add\_argument('params', nargs='+', type=parse\_value)

    args = parser.parse\_args()

    params = dict(args.params)

    input1 = params.get('input1')

    input2 = params.get('input2')

    input3 = params.get('input3')

    output = params.get('output')

    error\_file = format\_file\_path('errors.txt')

    try:

        network = Network(format\_file\_path(input1), format\_file\_path(input2), format\_file\_path(input3), format\_file\_path(output))

        result = network.train()

        with open(format\_file\_path(output), 'w') as f:

            f.write(result)

    except Exception as e:

        print(e)

        with open(error\_file, 'a') as f:

            f.write(f'{datetime.now()} : {e}\n')

Листинг 2 – Скрипт calculateN.py

import numpy as np

import json

from math import exp

def sigmoid(x):

    return 1 / (1 + exp(-x))

class Layer:

    def \_\_init\_\_(self, name, W):

        self.name = name

        self.W = np.array(W)

        shape = self.W.shape

        self.n = shape[0]

        self.m = shape[1]

        self.X = np.zeros(self.m)

        self.Y = np.zeros(self.n)

        self.diff = np.zeros(self.n)

    def calculate(self):

        for i in range(self.n):

            y = 0

            for j in range(self.m):

                y += self.W[i][j] \* self.X[j]

            self.Y[i] = sigmoid(y)

            self.diff[i] = self.Y[i] \* (1 - self.Y[i])

        return self.Y

class LayerShapeError(Exception):

    pass

class Network:

    def \_\_init\_\_(self, input\_Network, input\_Train, input\_Params, output):

        weights = load(input\_Network)

        self.layers = []

        sorted\_layers = sorted(weights.keys())

        for w in sorted\_layers:

            self.layers.append(Layer(w, weights[w]))

        self.deltas = [[]] \* len(self)

        XY = load(input\_Train)

        self.X = np.array(XY['X'])

        self.Y = np.array(XY['Y'])

        train\_params = load(input\_Params)

        self.iters = int(train\_params['iters'])

        self.alpha = float(train\_params['alpha'])

        self.eps = float(train\_params['eps'])

    def \_\_len\_\_(self):

        return len(self.layers)

    def forward(self, input):

        k = len(self)

        for i in range(k):

            l = self.layers[i]

            if i == 0:

                l.X = input

            new\_x = l.calculate()

            if i < k - 1:

                self.layers[i + 1].X = new\_x

        return self.layers[-1].Y

    def backward(self, output):

        error = 0

        last\_layer = self.layers[-1]

        out\_len = len(output)

        self.deltas[-1] = np.zeros(out\_len)

        for i in range(out\_len):

            e = last\_layer.Y[i] - output[i]

            self.deltas[-1][i] = e \* last\_layer.diff[i]

            error += e \* e / 2

        k = len(self)

        for i in range(k - 1, 0, -1):

            layer\_i = self.layers[i]

            self.deltas[i - 1] = np.zeros(layer\_i.m)

            for a in range(layer\_i.m):

                for b in range(layer\_i.n):

                    self.deltas[i - 1][a] += layer\_i.W[b][a] \* self.deltas[i][b]

                self.deltas[i - 1][a] \*= self.layers[i - 1].diff[a]

        return error

    def update(self):

        for i in range(len(self)):

            layer = self.layers[i]

            for a in range(layer.n):

                for b in range(layer.m):

                    layer.W[a][b] -= self.alpha \* self.deltas[i][a] \* layer.X[b]

    def train(self):

        it = 0

        error = 1

        result = ''

        while it < self.iters and error > self.eps:

            it += 1

            errors = []

            for i in range(len(self.X)):

                out = self.forward(self.X[i])

                errors.append(self.backward(self.Y[i]))

                self.update()

                print(f'X: {self.X[i]}, Y: {self.Y[i]}, out: {out}')

                error = np.mean(np.array(errors))

            # print(f'it: {it}, error: {error}')

            result += f'{it}: {error}\n'

            print('---------------------')

        # for i in range(len(self.X)):

        #     print(f'X: {self.X[i]}, Y: {self.Y[i]}, out: {self.forward(self.X[i])}')

        return result

def load(file):

    with open(file, 'r') as f:

        return json.load(f)

def dump(file, data):

    with open(file, 'w') as f:

        json.dump(data, f)

    print(data)