Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

**Отчет по дисциплине**

**«Нейронные сети»**

студента 5 курса 531 группы

специальности 10.05.01 «Компьютерная безопасность»

факультета компьютерных наук и информационных технологий

Латанова Кирилла Вячеславовича

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Саратов 2022

**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 (рис. 1).

**На выходе:**

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

б) Сообщение об ошибке в формате файла, если ошибка присутствует.

**Способ проверки результата:**

а) Сериализованная структура графа в формате XML или JSON.

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

**Результат работы программы:**

Запускаем программу командой:

python main.py -i input.txt -o output.xml

Получаем результат (рис. 1 – 2):

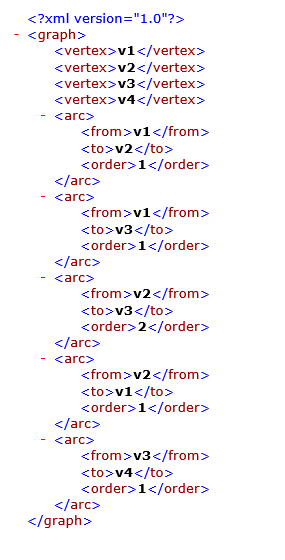


Рисунок 1 – Выходной файл output.xml

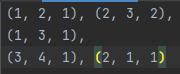


Рисунок 2 – Входной файл input.txt

**Листинг программы:**

import argparse  
from xml.dom import minidom  
import xml.etree.cElementTree as ET  
  
  
def read\_graph(input\_file):  
 with open(input\_file, 'r') as input\_graph:  
 all\_edges = {}  
 file = input\_graph.read().replace(' ', '').split('\n')  
 for j in range(len(file)):  
 if '-' in file[j]:  
 print(f'Ошибка в строке {j + 1}')  
 exit()  
 for i in file[j]:  
 if i.isalpha():  
 print(f'Ошибка в строке {j + 1}')  
 exit()  
 edges = ''.join(file).split('),(')  
 max\_vertex = int(edges[0][1]) if int(edges[0][1]) > int(edges[0][3]) else int(edges[0][3])  
 for i in range(1, len(edges)):  
 if max\_vertex < int(edges[i][0]):  
 max\_vertex = int(edges[i][0])  
 if max\_vertex < int(edges[i][2]):  
 max\_vertex = int(edges[i][2])  
 in\_number = [''] \* max\_vertex  
 for i in range(len(edges)):  
 if i == 0:  
 edges[i] = edges[i][1:]  
 if i == len(edges) - 1:  
 edges[i] = edges[i][:len(edges[i]) - 2] if edges[i][len(edges[i]) - 1] == '\n' else edges[i][:-1]  
 try:  
 edge = eval(edges[i])  
 if len(edge) != 3:  
 edge = str(edge).replace(' ', '')  
 for j in range(len(file)):  
 if edge in file[j]:  
 print(f'Ошибка в строке {j + 1}')  
 exit()  
 except:  
 edge = str(edge).replace(" ", '')  
 for j in range(len(file)):  
 if edge in file[j]:  
 print(f'Ошибка в строке {j + 1}')  
 exit()  
 if edge[0] not in all\_edges:  
 all\_edges[edge[0]] = []  
 if edge[1] not in all\_edges:  
 all\_edges[edge[1]] = []  
 in\_number[edge[1] - 1] = in\_number[edge[1] - 1] + " " + str(edge[2])  
 all\_edges[edge[0]].append([edge[2], edge[1]])  
 y = [x.split(' ') for x in in\_number]  
 for i in range(len(y)):  
 num = [int(y[i][j]) for j in range(1, len(y[i]))]  
 num.sort()  
 if len(num) == 1 and num[0] != 1:  
 print(f'Ошибка в строке {len(file)}')  
 exit()  
 for j in range(0, len(num) - 1):  
 if num[j] == num[j + 1]:  
 print(f'Ошибка в строке {len(file)}')  
 exit()  
 if num[j + 1] - num[j] != 1:  
 print(f'Ошибка в строке {len(file)}')  
 exit()  
 return all\_edges  
  
  
def main():  
 parser = argparse.ArgumentParser()  
 parser.add\_argument('-i', required=True, help='Имя входного файла')  
 parser.add\_argument('-o', required=True, help='Имя выходного файла')  
 args = parser.parse\_args()  
 all\_edges = read\_graph(args.i)  
 root = ET.Element('graph')  
 for x in range(len(all\_edges)):  
 ET.SubElement(root, 'vertex').text = 'v' + str(x + 1)  
 for x in range(1, len(all\_edges)):  
 for z in range(len(all\_edges[x])):  
 arc = ET.SubElement(root, 'arc')  
 ET.SubElement(arc, 'from').text = 'v' + str(x)  
 ET.SubElement(arc, 'to').text = 'v' + str(all\_edges[x][z][1])  
 ET.SubElement(arc, 'order').text = str(all\_edges[x][z][0])  
 dom = minidom.parseString(ET.tostring(root))  
 tree = dom.toprettyxml(indent='\t')  
 with open(args.o, 'w') as file:  
 file.write(tree)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**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:

a) выгрузка в текстовый файл результата преобразования графа в имя функции.

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

**Результат работы программы:**

Запускаем программу командой:

python main.py -i input.txt -o output.txt

Получаем результат (рис. 3 – 4):



Рисунок 3 – Входной файл input.txt



Рисунок 4 – Выходной файл output.txt

**Листинг программы:**

import argparse  
  
  
def read\_graph(input\_file):  
 with open(input\_file, 'r') as graph:  
 edges\_lst, lst = {}, {}  
 file = graph.read().replace(' ', '').split('\n')  
 for i in range(len(file)):  
 if '-' in file[i]:  
 print(f'Ошибка в строке {i + 1}')  
 exit()  
 for j in file[i]:  
 if j.isalpha():  
 print(f'Ошибка в строке {i + 1}')  
 exit()  
  
 edges = ''.join(file).split('),(')  
 max\_v = int(edges[0][1]) if int(edges[0][1]) > int(edges[0][3]) else int(edges[0][3])  
 for i in range(1, len(edges)):  
 if max\_v < int(edges[i][0]):  
 max\_v = int(edges[i][0])  
 elif max\_v < int(edges[i][2]):  
 max\_v = int(edges[i][2])  
  
 tmp = [''] \* max\_v  
 for i in range(len(edges)):  
 if i == 0:  
 edges[i] = edges[i][1:]  
 elif i == len(edges) - 1:  
 edges[i] = edges[i][:len(edges[i]) - 2] if edges[i][len(edges[i]) - 1] == '\n' else edges[i][:-1]  
  
 try:  
 edge = eval(edges[i])  
 if len(edge) != 3:  
 edge = str(edge).replace(' ', '')  
 for j in range(len(file)):  
 if edge in file[j]:  
 print(f'Ошибка в строке {j + 1}')  
 exit()  
 except:  
 edge = str(edge).replace(' ', '')  
 for j in range(len(file)):  
 if edge in file[j]:  
 print(f'Ошибка в строке {j + 1}')  
 exit()  
  
 if edge[0] not in edges\_lst:  
 edges\_lst[edge[0]] = []  
 lst[edge[0]] = [1, 0]  
 else:  
 lst[edge[0]][0] += 1  
 if edge[1] not in edges\_lst:  
 edges\_lst[edge[1]] = []  
 lst[edge[1]] = [0, 1]  
 else:  
 lst[edge[1]][1] += 1  
  
 tmp[edge[1] - 1] = tmp[edge[1] - 1] + ' ' + str(edge[2])  
 edges\_lst[edge[0]].append([edge[2], edge[1]])  
  
 y = [x.split(' ') for x in tmp]  
 for i in range(len(y)):  
 num = [int(y[i][j]) for j in range(1, len(y[i]))]  
 num.sort()  
 if len(num) == 1 and num[0] != 1:  
 print(f'Неправильная нумерация в строке {len(file)}')  
 exit()  
 for j in range(len(num) - 1):  
 if num[j] == num[j + 1] or num[j + 1] - num[j] != 1:  
 print(f'Неправильная нумерация в строке {len(file)}')  
 exit()  
  
 graph = {}  
 for j in range(1, len(edges\_lst) + 1):  
 graph[j] = []  
 for i in range(len(edges\_lst[j])):  
 graph[j].append([edges\_lst[j][i][0], edges\_lst[j][i][1]])  
  
 return graph, lst  
  
  
def dfs\_check(v, graph):  
 global ans  
 is\_any, is\_first = False, True  
 for vertex in graph[v]:  
 if not is\_first:  
 ans += ', '  
 is\_any = True  
 if is\_first:  
 ans += str(v) + '('  
 is\_first = False  
 dfs\_check(vertex[1], graph)  
 if is\_any:  
 ans += ')'  
 elif is\_first:  
 ans += str(v)  
 return ans  
  
  
def cycle(x, graph, used, tmp):  
 used[x] = True  
 dfs\_graph = {x: []}  
 for v in graph[x]:  
 if v[1] not in used:  
 dfs\_graph[x].append(cycle(v[1], graph, used, tmp))  
 else:  
 if v[1] not in tmp:  
 print(f'Ошибка - цикл между вершинами {x} и {v[1]}')  
 exit()  
 else:  
 dfs\_graph[x].append({v[1]: tmp[v[1]]})  
 tmp[x] = dfs\_graph[x]  
 return dfs\_graph  
  
  
def check\_cycle(graph, d):  
 v\_lst = [v for v in d if d[v][1] == 0]  
 for v in v\_lst:  
 cycle(v, graph, {}, {})  
  
  
def check\_cycle2(graph, d):  
 v\_lst = [v for v in d if d[v][1] == 0]  
 if not v\_lst:  
 print(f'Ошибка - цикл между вершинами 1 и {len(d)}')  
 exit()  
 global ans  
 ans = ''  
 for v in range(len(v\_lst)):  
 ans = dfs\_check(v\_lst[v], graph)  
 if v != len(v\_lst) - 1:  
 ans += ', '  
 return ans  
  
  
def reverse\_graph(graph):  
 ans = {}  
 for u in sorted(graph.keys()):  
 for v in graph[u]:  
 if u not in ans:  
 ans[u] = []  
 if v[1] not in ans:  
 ans[v[1]] = []  
 ans[v[1]].append([v[0], u])  
 return ans  
  
  
def main():  
 parser = argparse.ArgumentParser()  
 parser.add\_argument('-i', required=1, help='Имя входного файла')  
 parser.add\_argument('-o', required=1, help='Имя выходного файла')  
 args = parser.parse\_args()  
 edges\_lst, lst = read\_graph(args.i)  
 check\_cycle(edges\_lst, lst)  
  
 for i in range(1, len(lst) + 1):  
 lst[i][0], lst[i][1] = lst[i][1], lst[i][0]  
 rev\_graph = reverse\_graph(edges\_lst)  
 graph = [[]] \* (len(edges\_lst) + 1)  
 i = 0  
 for x in sorted(rev\_graph.keys()):  
 tmp = [[rev\_graph[x][i]] for i in range(len(rev\_graph[x]))]  
 graph[i + 1] = tmp  
 i += 1  
  
 ans = []  
 for i in graph:  
 tmp = [[i[j][0][0], i[j][0][1]] for j in range(len(i))]  
 tmp.sort(key=lambda x: x[0])  
 ans.append(tmp)  
  
 with open(args.o, 'w') as file:  
 file.write(check\_cycle2(ans, lst))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

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

**На входе:**

а) Текстовый файл с описанием графа в виде списка дуг (смотри задание 1).

б) 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.

**Результат работы программы:**

Запускаем программу командой:

python main.py -i input.txt -o output.txt -f operations.txt

Получаем результат (рис. 5 – 7):



Рисунок 5 – Входной файл input.txt

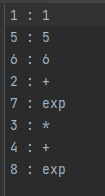


Рисунок 6 – Входной файл operations.txt

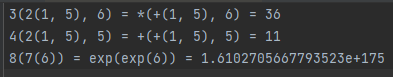


Рисунок 7 – Выходной файл output.txt

**Листинг программы:**

from argparse import ArgumentParser  
from math import exp  
  
  
def read\_graph(input\_file):  
 with open(input\_file, 'r') as graph:  
 edges\_lst, lst = {}, {}  
 file = graph.read().replace(' ', '').split('\n')  
 for i in range(len(file)):  
 if '-' in file[i]:  
 print(f'Ошибка в строке {i + 1}')  
 exit()  
 for j in file[i]:  
 if j.isalpha():  
 print(f'Ошибка в строке {i + 1}')  
 exit()  
  
 edges = ''.join(file).split('),(')  
 max\_v = int(edges[0][1]) if int(edges[0][1]) > int(edges[0][3]) else int(edges[0][3])  
 for i in range(1, len(edges)):  
 if max\_v < int(edges[i][0]):  
 max\_v = int(edges[i][0])  
 elif max\_v < int(edges[i][2]):  
 max\_v = int(edges[i][2])  
  
 tmp = [''] \* max\_v  
 for i in range(len(edges)):  
 if i == 0:  
 edges[i] = edges[i][1:]  
 elif i == len(edges) - 1:  
 edges[i] = edges[i][:len(edges[i]) - 2] if edges[i][len(edges[i]) - 1] == '\n' else edges[i][:-1]  
  
 try:  
 edge = eval(edges[i])  
 if len(edge) != 3:  
 edge = str(edge).replace(' ', '')  
 for j in range(len(file)):  
 if edge in file[j]:  
 print(f'Ошибка в строке {j + 1}')  
 exit()  
 except:  
 edge = str(edge).replace(' ', '')  
 for j in range(len(file)):  
 if edge in file[j]:  
 print(f'Ошибка в строке {j + 1}')  
 exit()  
  
 if edge[0] not in edges\_lst:  
 edges\_lst[edge[0]] = []  
 lst[edge[0]] = [1, 0]  
 else:  
 lst[edge[0]][0] += 1  
 if edge[1] not in edges\_lst:  
 edges\_lst[edge[1]] = []  
 lst[edge[1]] = [0, 1]  
 else:  
 lst[edge[1]][1] += 1  
  
 tmp[edge[1] - 1] = tmp[edge[1] - 1] + ' ' + str(edge[2])  
 edges\_lst[edge[0]].append([edge[2], edge[1]])  
  
 y = [x.split(' ') for x in tmp]  
 for i in range(len(y)):  
 num = [int(y[i][j]) for j in range(1, len(y[i]))]  
 num.sort()  
 if len(num) == 1 and num[0] != 1:  
 print(f'Неправильная нумерация в строке {len(file)}')  
 exit()  
 for j in range(len(num) - 1):  
 if num[j] == num[j + 1] or num[j + 1] - num[j] != 1:  
 print(f'Неправильная нумерация в строке {len(file)}')  
 exit()  
  
 graph = {}  
 for j in range(1, len(edges\_lst) + 1):  
 graph[j] = []  
 for i in range(len(edges\_lst[j])):  
 graph[j].append([edges\_lst[j][i][0], edges\_lst[j][i][1]])  
  
 return graph, lst  
  
  
def dfs\_check(x, graph):  
 is\_any, is\_first = False, True  
 global output, tmp  
  
 for v in graph[x]:  
 is\_any = True  
 if is\_first:  
 output += f'{x}('  
 tmp += f'{x}('  
 is\_first = False  
 else:  
 output += ', '  
 tmp += ', '  
 dfs\_check(v[1], graph)  
   
 if is\_any:  
 output += ')'  
 tmp += ')'  
 else:  
 if is\_first:  
 output += str(x)  
 tmp += str(x)  
   
 return output, tmp  
  
  
def cycle(x, graph, used, tmp):  
 used[x] = True  
 dfs\_graph = {x: []}  
   
 for v in graph[x]:  
 if v[1] not in used:  
 dfs\_graph[x].append(cycle(v[1], graph, used, tmp))  
 else:  
 if v[1] not in tmp:  
 print(f'Ошибка - цикл между вершинами {x} и {v[1]}')  
 exit()  
 else:  
 dfs\_graph[x].append({v[1]: tmp[v[1]]})  
 tmp[x] = dfs\_graph[x]  
   
 return dfs\_graph  
  
  
def check\_cycle(graph, d):  
 v\_lst = [vertex for vertex in d if d[vertex][1] == 0]  
 for v in v\_lst:  
 cycle(v, graph, {}, {})  
  
  
def check\_cycle2(graph, d):  
 v\_lst = [vertex for vertex in d if d[vertex][1] == 0]  
 if not v\_lst:  
 print(f'Ошибка - цикл между вершинами 1 и {len(d)}')  
 exit()  
  
 global output, fun, tmp  
 tmp, fun, output = '', [], '',  
  
 for v in range(len(v\_lst)):  
 output, tmp = dfs\_check(v\_lst[v], graph)  
 if v != len(v\_lst) - 1:  
 output += ', '  
 fun.append(tmp)  
 tmp = ''  
 fun.append(tmp)  
   
 return output, fun  
  
  
def reverse\_graph(graph):  
 ans = {}  
 for u in sorted(graph.keys()):  
 for v in graph[u]:  
 if u not in ans:  
 ans[u] = []  
 if v[1] not in ans:  
 ans[v[1]] = []  
 ans[v[1]].append([v[0], u])  
 return ans  
  
  
def read\_operations(v\_lst, file):  
 ops, i = {}, 0  
 op\_lst = ['+', '\*', 'exp']  
  
 with open(file, 'r') as input\_operations:  
 for line in input\_operations:  
 line = line[:(len(line) - 1)].replace(' ', '')  
 j = line.find(':')  
 if j == -1:  
 print(f'Ошибка операции, строка {i + 1}')  
 exit()  
 v = int(line[:j])  
 if v not in v\_lst:  
 print(f'Ошибка вершины, строка {i + 1}')  
 exit()  
  
 op = str(line[(j + 1):])  
 try:  
 ops[str(v)] = int(op) if op not in op\_lst else op  
 except:  
 print(f'Ошибка операции, строка {i + 1}')  
 exit()  
 i += 1  
 return ops  
  
  
def check\_operations(graph, ops):  
 for v in graph.keys():  
 if type(ops[str(v)]) == int:  
 if len(graph[v]) == 0:  
 continue  
 print(f'Ошибка операции - {ops[str(v)]}, вершина {v}')  
 exit()  
 elif ops[str(v)] == '+' or ops[str(v)] == '\*':  
 if len(graph[v]) > 1:  
 continue  
 print(f'Ошибка операции - {ops[str(v)]}, вершина {v}')  
 exit()  
 elif ops[str(v)] == 'exp':  
 if len(graph[v]) == 1:  
 continue  
 print(f'Ошибка операции - {ops[str(v)]}, вершина {v}')  
 exit()  
  
  
def dfs\_ops(x, graph, ops, used, vals):  
 used[x] = True  
 tmp = {x: -1}  
 if type(ops[str(x)]) == int:  
 tmp[x] = ops[str(x)]  
 elif ops[str(x)] == '+':  
 tmp[x] = 0  
 elif ops[str(x)] == '\*' or ops[str(x)] == 'exp':  
 tmp[x] = 1  
  
 if not len(graph[x]):  
 vals[x] = tmp[x]  
 return vals  
 for v in graph[x]:  
 if v[1] not in used:  
 val = dfs\_ops(v[1], graph, ops, used, vals)  
 if ops[str(x)] == '+':  
 tmp[x] += val[v[1]]  
 elif ops[str(x)] == '\*':  
 tmp[x] \*= val[v[1]]  
 elif ops[str(x)] == 'exp':  
 tmp[x] = exp(val[v[1]])  
 else:  
 if ops[str(x)] == '+':  
 tmp[x] += vals[v[1]]  
 elif ops[str(x)] == '\*':  
 tmp[x] \*= vals[v[1]]  
 elif ops[str(x)] == 'exp':  
 tmp[x] = exp(vals[v[1]])  
 vals[x] = tmp[x]  
 return vals  
  
  
def evaluating(graph, started\_vertex, operations):  
 used, vals = {}, {}  
 for v in started\_vertex:  
 dfs\_ops(v, graph, operations, used, vals)  
 return vals  
  
  
def main():  
 parser = ArgumentParser()  
 parser.add\_argument('-i', required=1, help='Имя входного файла')  
 parser.add\_argument('-f', required=1, help='Имя файла с операциями')  
 parser.add\_argument('-o', required=1, help='Имя выходного файла')  
 args = parser.parse\_args()  
  
 v\_lst, lst = read\_graph(args.i)  
 start = []  
 for x in v\_lst:  
 if not len(v\_lst[x]):  
 start.append(x)  
  
 check\_cycle(v\_lst, lst)  
 for x in range(1, len(lst) + 1):  
 lst[x][0], lst[x][1] = lst[x][1], lst[x][0]  
  
 rev\_graph = reverse\_graph(v\_lst)  
 graph = [[]] \* (len(v\_lst) + 1)  
  
 k = 0  
 for x in sorted(rev\_graph.keys()):  
 tmp = [[rev\_graph[x][i]] for i in range(len(rev\_graph[x]))]  
 graph[k + 1] = tmp  
 k += 1  
  
 new\_graph = []  
 for i in graph:  
 tmp = [[i[j][0][0], i[j][0][1]] for j in range(len(i))]  
 tmp.sort(key=lambda x: x[0])  
 new\_graph.append(tmp)  
  
 output, fun = check\_cycle2(new\_graph, lst)  
 fun = [y for y in fun if y != '']  
 ops = read\_operations(v\_lst, args.f)  
 new\_fun = []  
 fun\_string = ''  
  
 for x in fun:  
 for char in x:  
 tmp = char  
 if char.isdigit():  
 tmp = ops[str(char)]  
 fun\_string += str(tmp)  
 new\_fun.append(fun\_string)  
 fun\_string = ''  
  
 check\_operations(rev\_graph, ops)  
  
 vals = evaluating(rev\_graph, start, ops)  
 with open(args.o, 'w') as output:  
 for i in range(len(start)):  
 txt = [''.join(fun[i]), ''.join(new\_fun[i]), str(vals[start[i]])]  
 print(\*txt, sep=' = ', file=output)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**Задание 4. Построение многослойной нейронной сети**

**На входе:**

а) 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]]

б) Текстовый файл с входным вектором в формате:

x1, x2, …, xn.

**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:

y1, y2, …, yn.

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

**Результат работы программы:**

Запускаем программу командой:

python main.py -m matrix.txt -o output.txt -i input.txt

Получаем результат (рис. 8 – 11):

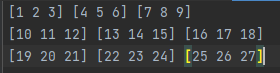
****

Рисунок 8 – Входной файл matrix.txt



Рисунок 9 – Входной файл input.txt



Рисунок 10 – Выходной файл output.txt

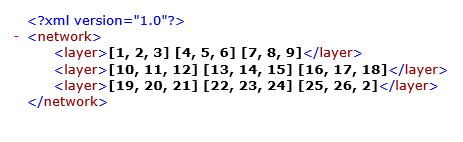


Рисунок 11 – Выходной файл output.xml

**Листинг программы:**

from re import sub  
from xml.dom import minidom  
from argparse import ArgumentParser  
import xml.etree.cElementTree as ET  
  
  
def read\_files(textfile, input):  
 mtx, idx = [], 0  
 with open(textfile, 'r') as rf:  
 v = [int(x) for x in rf.read().split(' ')]  
 with open(input, 'r') as rf:  
 txt = rf.readlines()  
  
 for line in txt:  
 idx += 1  
 line = sub(r'[\[\]]', ' ', line)[1:-2].split(' ')  
 tmp = []  
 for x in line:  
 x = x.split(' ')  
 try:  
 tmp.append([int(i) for i in x])  
 except ValueError:  
 print(f'Ошибка в строке {idx}')  
 exit()  
 if len(x) != len(v):  
 print(f'Ошибка числа компонент нейронов в слоях {idx - 1} - {idx}')  
 exit()  
 mtx.append([tmp])  
  
 return mtx, v  
  
  
def evaluate(mtx, v):  
 ans = []  
 for layer in mtx:  
 tmp = []  
 for x in layer:  
 for neuron in x:  
 val = sum([neuron[i] \* v[i] for i in range(len(v))])  
 val /= (1 + abs(val))  
 tmp.append(val)  
 ans.append(tmp)  
 v = tmp  
 return ans  
  
  
def main():  
 parser = ArgumentParser()  
 parser.add\_argument('-m', required=1, help='Имя входного файла')  
 parser.add\_argument('-i', required=1, help='Имя файла с начальным вектором')  
 parser.add\_argument('-o', required=1, help='Имя выходного файла')  
 args = parser.parse\_args()  
  
 mtx, v = read\_files(args.i, args.m)  
 new\_mtx = evaluate(mtx, v)  
  
 with open(args.o, 'w') as output:  
 for x in new\_mtx[-1]:  
 output.write(str(x) + ' ')  
  
 root = ET.Element('network')  
 for layer in mtx:  
 tmp = ''  
 for x in layer:  
 for y in x:  
 tmp += f'{y} '  
 break  
 ET.SubElement(root, 'layer').text = tmp[:-1]  
  
 ans = minidom.parseString(ET.tostring(root)).toprettyxml(indent='\t')  
 with open('output.xml', 'w') as file:  
 file.write(ans)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**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:

[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.

в) Число итераций обучения (в строке параметров).

**На выходе:**

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

1 : Ошибка1

2 : Ошибка2

...

N : ОшибкаN

**Результат работы программы:**

Запускаем программу командой:

python main.py -i input.txt -o output.txt -d description.txt -n 500 -e 0.02

Получаем результат (рис. 12 – 14):

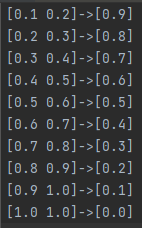
****

Рисунок 12 – Входной файл input.txt



Рисунок 13 – Входной файл description.txt

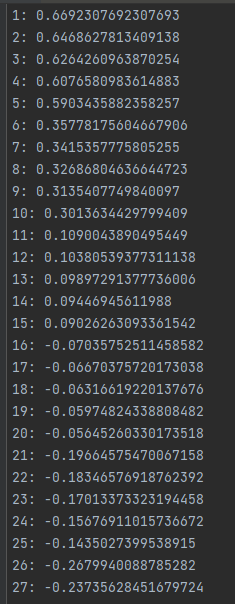


Рисунок 14 – Выходной файл output.txt

**Листинг программы:**

import ast  
import sys  
from argparse import ArgumentParser  
  
  
def read\_files(input\_file, description\_file):  
 try:  
 samples = []  
 with open(input\_file, 'r') as rf:  
 for line in rf:  
 tmp = line.replace('\n', '').split('->')  
 io = [nrn[1:-1].split(' ') for nrn in tmp]  
 samples.append(io)  
 for x in range(len(samples)):  
 for y in range(len(samples[x])):  
 for z in range(len(samples[x][y])):  
 samples[x][y][z] = float(''.join(samples[x][y][z]))  
  
 with open(description\_file, 'r') as rf:  
 nrns = ast.literal\_eval(rf.read())  
  
 return samples, nrns  
 except:  
 print(f'Ошибка чтения файла')  
 exit()  
  
  
class Training:  
 def \_\_init\_\_(self, parameters, neurons, samples):  
 self.params = parameters  
 self.nrns = neurons  
 self.samples = samples  
 self.funs = []  
 self.weights = []  
 for i in range(len(neurons)):  
 self.funs.append([])  
 self.weights.append([])  
  
 def activation(self, x):  
 return x / (1 + abs(x))  
  
 def derivative(self, x):  
 return 1 / pow((1 + abs(x)), 2)  
  
 def perceptron(self, input):  
 ans = input.copy()  
  
 for j in range(len(self.nrns)):  
 n = len(ans)  
 for k in range(len(self.nrns[j])):  
 if n != len(self.nrns[j][k]):  
 print(f'Ошибка - число нейронов не совпадает с длиной массива весов')  
 exit()  
 total = sum([ans[i] \* self.nrns[j][k][i] for i in range(len(self.nrns[j][k]))])  
 self.weights[j].append(total)  
 total = self.activation(total)  
 self.funs[j].append(total)  
 ans.append(total)  
 if n > 0:  
 ans[:n] = []  
 return ans  
  
 def education(self, output):  
 n, eps = self.params[0], self.params[1]  
 ans\_lines = [[] for \_ in range(len(self.samples))]  
  
 for sample in range(len(self.samples)):  
 for count in range(n):  
 input, exp = self.samples[sample][0], self.samples[sample][1]  
 perc = self.perceptron(input)  
  
 if not count % 100:  
 ans\_lines[sample].append(f'{count // 100 + 1 + sample \* n // 100}: {exp[0] - perc[0]}\n')  
  
 sgm = [[] for x in range(len(self.nrns))]  
 for i in range(len(perc)):  
 sgm[len(self.nrns) - 1] = [exp[i] - perc[i]]  
  
 for i in range(len(self.nrns) - 1, 0, -1):  
 for j in range(len(self.nrns[i][i - 1])):  
 total = sum([abs(sgm[i][x]) \* self.nrns[i][x][j] for x in range(len(sgm[i]))])  
 sgm[i - 1].append([total])  
  
 for i in range(len(self.nrns[0])):  
 for w in range(len(self.nrns[0][i])):  
 dw = sgm[0][i] \* self.derivative(self.weights[0][i]) \* self.samples[sample][0][w] \* eps  
 self.nrns[0][i][w] = self.nrns[0][i][w] + dw  
  
 for i in range(1, len(self.nrns)):  
 for j in range(len(self.nrns[i])):  
 for w in range(len(self.nrns[i][j])):  
 dw = sgm[i][j] \* self.derivative(self.weights[i][j]) \* self.funs[i - 1][w] \* eps  
 self.nrns[i][j][w] = self.nrns[i][j][w] + dw  
  
 with open(output, 'w') as file:  
 for line in ans\_lines:  
 for x in line:  
 file.write(x)  
  
  
def main():  
 parser = ArgumentParser()  
 parser.add\_argument('-i', required=1, help='Имя входного файла')  
 parser.add\_argument('-d', required=1, help='Имя файла с начальным вектором')  
 parser.add\_argument('-o', required=1, help='Имя выходного файла')  
 parser.add\_argument('-n', required=1, help='Число эпох')  
 parser.add\_argument('-e', required=1, help='Скорость обучения')  
 args = parser.parse\_args()  
  
 samples, neurons = read\_files(args.i, args.d)  
 n, e = int(args.n), float(args.e)  
 Training([n, e], neurons, samples).education(args.o)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()