

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

студентки 5 курса 531 группы

факультета компьютерных наук и информационных технологий

Змеевой Вероники Александровны

фамилия, имя, отчество

Преподаватель

аспирант

А. С. Никитина

подпись, дата

Саратов 2024

СОДЕРЖАНИЕ

1. Создание ориентированного графа	3
1.1 Описание задания 1	3
1.2 Тестирование задания 1	4
2. Создание функции по графу	11
2.1 Описание задания 2	11
2.2 Тестирование задания 2	11
3. Вычисление значения функции на графе	20
3.1 Описание задания 3	20
3.2 Тестирование задания 3	21
4. Построение многослойной нейронной сети	29
4.1 Описание задания 4	29
4.2 Тестирование задания 4	30
5. Построение многослойной нейронной сети	33
5.1 Описание задания 5	33
5.2 Тестирование задания 5	33
ПРИЛОЖЕНИЯ	42
Приложение 1 – задание 1	42
Приложение 2 – задание 2	45
Приложение 3 – задание 3	48
Приложение 4 – задание 4	54
Приложение 5 – задание 5	56

1. Создание ориентированного графа

1.1 Описание задания 1

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

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

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

На выходе:

а) Ориентированный граф с 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.

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>
```

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

1.2 Тестирование задания 1

Пример входного файла:

`(v1, v3, 1), (v2, v3, 2)`

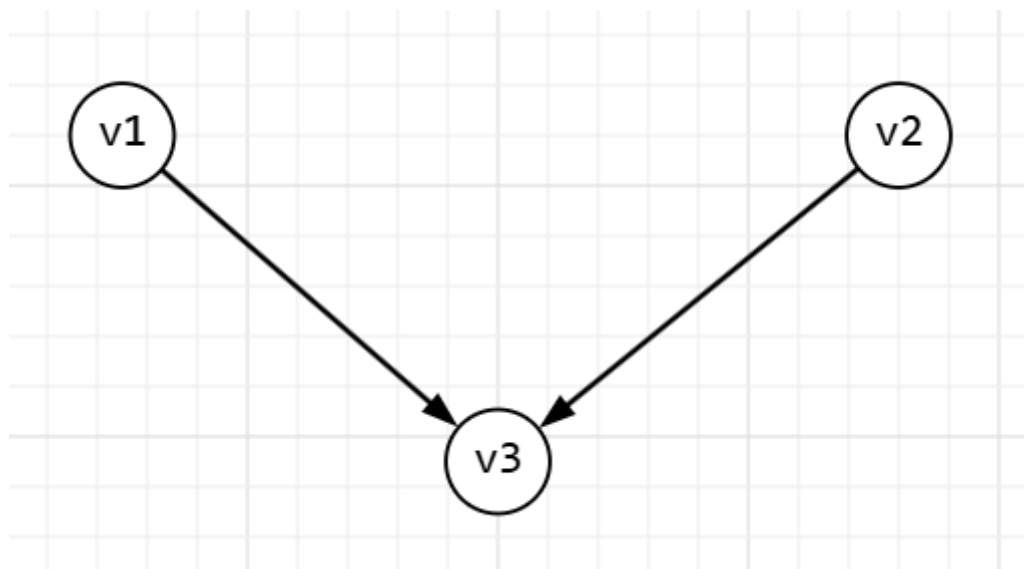


Рисунок 1 – граф по описанию

Пример работы программы:

```
D:\python\нс>python nntask1.py input1=test1.txt
Файл для вывода не был введен.
Поэтому был установлен файл по умолчанию (output1.json)
Граф был успешно записан в файл
```

Рисунок 2 – запуск программы в консоли

```

hc > {} output1.json > ...
1  {
2    "graph": {
3      "vertex": [
4        "v1",
5        "v2",
6        "v3"
7      ],
8      "arc": [
9        {
10         "from": "v1",
11         "to": "v3",
12         "order": 1
13       },
14       {
15         "from": "v2",
16         "to": "v3",
17         "order": 2
18       }
19     ]
20   }
21 }

```

Рисунок 3 – результат работы программы

Пример входного файла:

(v1, v2, 1), (v3, v4, 1), (v2, v5, 1), (v5, v6, 1), (v5, v7, 1), (v4, v8,
1), (v4, v9, 1), (v9, v6, 2)

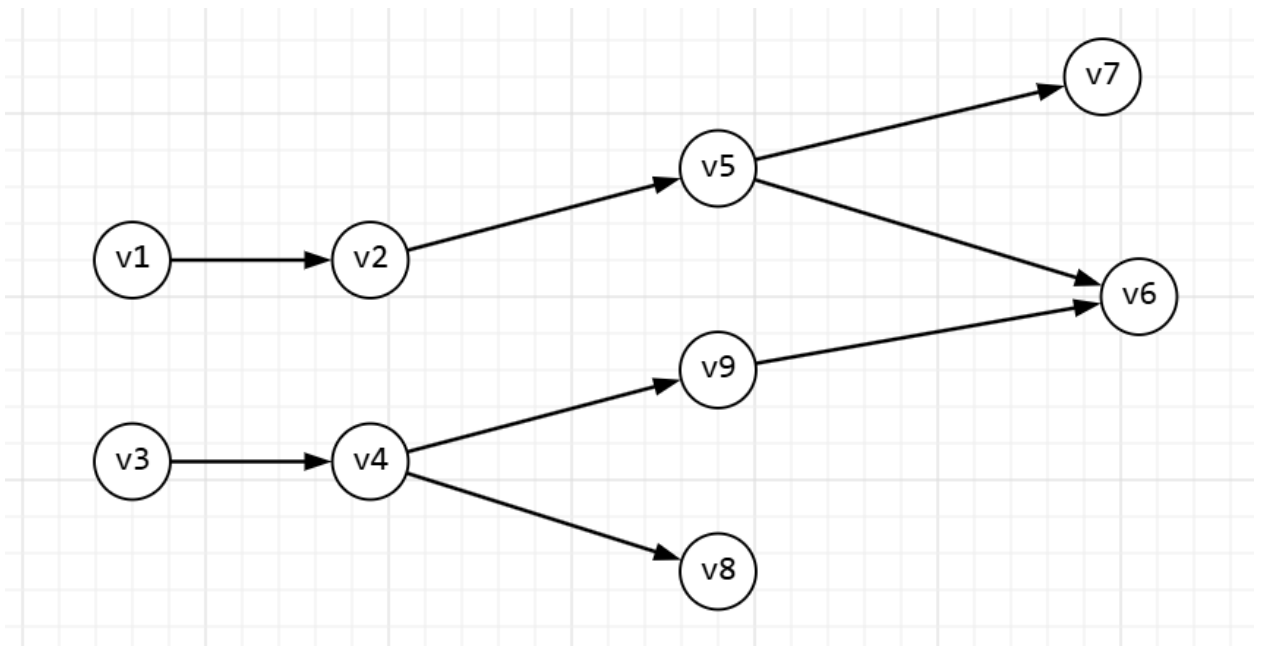


Рисунок 4 – граф по описанию

Пример работы программы:

```
D:\python\nc>python nntask1.py input1=test2.txt
Файл для вывода не был введен.
Поэтому был установлен файл по умолчанию (output1.json)
Граф был успешно записан в файл
```

Рисунок 5 – запуск программы в консоли

```
1  {
2    "graph": {
3      "vertex": [
4        "v1",
5        "v2",
6        "v3",
7        "v4",
8        "v5",
9        "v6",
10       "v7",
11       "v8",
12       "v9"
13     ],
14     "arc": [
15       {
16         "from": "v1",
17         "to": "v2",
18         "order": 1
19       },
20       {
21         "from": "v3",
22         "to": "v4",
23         "order": 1
24       },
25       {
26         "from": "v2",
27         "to": "v5",
28         "order": 1
29     },
```

```

30      {
31          "from": "v5",
32          "to": "v6",
33          "order": 1
34      },
35      {
36          "from": "v5",
37          "to": "v7",
38          "order": 1
39      },
40      {
41          "from": "v4",
42          "to": "v8",
43          "order": 1
44      },
45      {
46          "from": "v4",
47          "to": "v9",
48          "order": 1
49      },
50      {
51          "from": "v9",
52          "to": "v6",
53          "order": 2
54      }
55  ]
56  }
57  }

```

Рисунок 6 – результат работы программы

Пример входного файла:

(v1, v2, 1), (v1, v2, 1)

Пример работы программы:

```

D:\python\нс>python nntask1.py input1=test3.txt
Файл для вывода не был введен.
Поэтому был установлен файл по умолчанию (output1.json)
Некорректный ввод данных.
Проверьте уникальность порядковых номеров у каждой из дуг.

```


Рисунок 7 – запуск программы в консоли

Пример входного файла:

$(v1, v2, 1), (v1, v2, a)$

Пример работы программы:

```
D:\python\нс>python nntask1.py input1=test4.txt
Файл для вывода не был введен.
Поэтому был установлен файл по умолчанию (output1.json)
Некорректный ввод данных.
Проверьте уникальность порядковых номеров у каждой из дуг.
```

Рисунок 8 – запуск программы в консоли

Пример входного файла:

$(a, b, 1), (c, d, 1), (b, e, 1), (d, e, 2), (e, f, 1)$

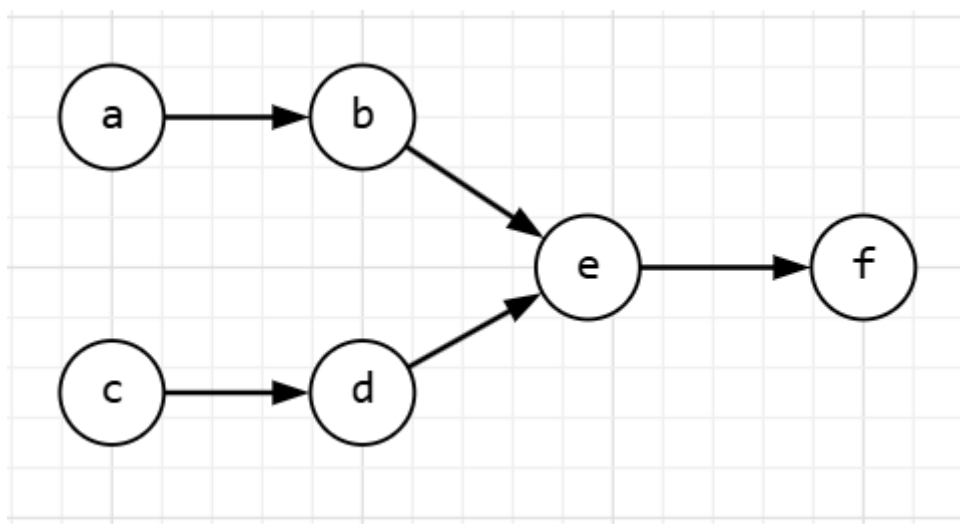


Рисунок 9 – граф по описанию

Пример работы программы:

```
D:\python\нс>python nntask1.py input1=test5.txt
Файл для вывода не был введен.
Поэтому был установлен файл по умолчанию (output1.json)
Граф был успешно записан в файл
```

Рисунок 10 – запуск программы в консоли

```

hc > {} output1.json > {} graph > [ ] arc
1  {
2  "graph": {
3    "vertex": [
4      "a",
5      "b",
6      "c",
7      "d",
8      "e",
9      "f"
10   ],
11   "arc": [
12     {
13       "from": "a",
14       "to": "b",
15       "order": 1
16     },
17     {
18       "from": "c",
19       "to": "d",
20       "order": 1
21     },
22     {
23       "from": "b",
24       "to": "e",
25       "order": 1
26     },
27     {
28       "from": "d",
29       "to": "e",
30       "order": 2
31     },
32     {
33       "from": "e",
34       "to": "f",
35       "order": 1
36     }
37   ]
38 }
39 }

```

Рисунок 11 – результат работы программы

Пример входного файла:

(a, b, 1), (c, d, 1), (b, e, 1), (d, e, 1), (e, f, 1)

Пример работы программы:

```
D:\python\nc>python nntask1.py input1=test6.txt
Файл для вывода не был введен.
Поэтому был установлен файл по умолчанию (output1.json)
В графе некорректно заданы номера!
Проверьте уникальность номеров.
```

Рисунок 12 – запуск программы в консоли

2. Создание функции по графу

2.1 Описание задания 2

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

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

$$A_1(B_1(C_1(\dots), \dots, C_m(\dots)), \dots, B_n(\dots))$$

Способ проверки результата:

- а) выгрузка в текстовый файл результата преобразования графа в имя функции.
- б) сообщение о наличии циклов в графе, если они присутствуют.

2.2 Тестирование задания 2

Пример входного файла:

```
нс > {} test1.json > ...
1  {
2    "graph": {
3      "vertex": [
4        "v1",
5        "v2",
6        "v3"
7      ],
8      "arc": [
9        {
10         "from": "v1",
11         "to": "v3",
12         "order": 1
13       },
14       {
15         "from": "v2",
16         "to": "v3",
17         "order": 2
18       }
19     ]
20   }
21 }
```

Рисунок 13 – входной файл

Пример работы программы:

```
D:\python\нс>python nntask2.py input1=test1.json
Файл для вывода не был введен.
Поэтому был установлен файл по умолчанию (func.txt)
Линейное представление функции сохранено в файл func.txt.
```

Рисунок 14 – запуск программы в консоли

Линейное представление функции: $v_3(v_1(), v_2())$

Пример входного файла:

HC > {} test2.json > ...

```
1  {
2    "graph": {
3      "vertex": [
4        "v1",
5        "v2",
6        "v3",
7        "v4",
8        "v5",
9        "v6",
10       "v7",
11       "v8",
12       "v9"
13     ],
14     "arc": [
15       {
16         "from": "v1",
17         "to": "v2",
18         "order": 1
19       },
20       {
21         "from": "v3",
22         "to": "v4",
23         "order": 1
24       },
25       {
26         "from": "v2",
27         "to": "v5",
28         "order": 1
29       },
30       {
31         "from": "v5",
32         "to": "v6",
33         "order": 1
34       },
35       {
36         "from": "v5",
37         "to": "v7",
38         "order": 1
```

```

38         "order": 1
39     },
40     {
41         "from": "v4",
42         "to": "v8",
43         "order": 1
44     },
45     {
46         "from": "v4",
47         "to": "v9",
48         "order": 1
49     },
50     {
51         "from": "v9",
52         "to": "v6",
53         "order": 2
54     }
55 ]
56 }
57 }

```

Рисунок 15 – входной файл

Пример работы программы:

```

D:\python\нс>python nntask2.py input1=test2.json
Файл для вывода не был введен.
Поэтому был установлен файл по умолчанию (func.txt)
Линейное представление функции сохранено в файл func.txt.

```

Рисунок 16 – запуск программы в консоли

Линейное представление функции: $v_6(v_5(v_2(v_1()))), v_9(v_4(v_3()))), v_7(v_5(v_2(v_1()))), v_8(v_4(v_3()))$

Пример входного файла:

```
нс > {} test3.json > {} graph > [ ] arc > {} 1
1  {
2    "graph": {
3      "vertex": [
4        "v1",
5        "v2",
6        "v3"
7      ],
8      "arc": [
9        {
10         "from": "v1",
11         "to": "v2",
12         "order": 1
13       },
14       {
15         "from": "v3",
16         "to": "v1",
17         "order": 1
18       }
19     ]
20   }
21 }
```

Рисунок 17 – входной файл

Пример работы программы:

```
D:\python\нс>python nntask2.py input1=test3.json
Файл для вывода не был введен.
Поэтому был установлен файл по умолчанию (func.txt)
Линейное представление функции сохранено в файл func.txt.
```

Рисунок 18 – запуск программы в консоли

Линейное представление функции: $v_2(v_1(v_3()))$

Пример входного файла:

```
ntask2.py test4.json X
nc > {} test4.json > ...
1 {
2   "graph": {
3     "vertex": [
4       "v1",
5       "v2",
6       "v3"
7     ],
8     "arc": [
9       {
10        "from": "v1",
11        "to": "v2",
12        "order": 1
13      },
14      {
15        "from": "v2",
16        "to": "v3",
17        "order": 1
18      },
19      {
20        "from": "v3",
21        "to": "v1",
22        "order": 1
23      }
24    ]
25  }
26 }
```

Рисунок 19 – входной файл

Пример работы программы:

```
D:\python\nc>python nntask2.py input1=test4.json
Файл для вывода не был введен.
Поэтому был установлен файл по умолчанию (func.txt)
В графе обнаружен цикл.
Не получилось считать граф с файла test4.json
```

Рисунок 20 – запуск программы в консоли

Пример входного файла:

```
nntask2.py test5.json x
нс > {} test5.json > ...
1 {
2   "graph": {
3     "vertex": [
4       "A",
5       "B",
6       "C",
7       "D",
8       "E",
9       "F",
10      "G"
11    ],
12    "arc": [
13      {
14        "from": "A",
15        "to": "D",
16        "order": 1
17      },
18      {
19        "from": "A",
20        "to": "D",
21        "order": 2
22      },
23      {
24        "from": "B",
25        "to": "E",
26        "order": 1
27      },
28    ]
  }
```

```

29     "from": "C",
30     "to": "E",
31     "order": 2
32 },
33 {
34     "from": "D",
35     "to": "G",
36     "order": 1
37 },
38 {
39     "from": "E",
40     "to": "F",
41     "order": 1
42 },
43 {
44     "from": "F",
45     "to": "G",
46     "order": 2
47 }
48 ]
49 }
50 }

```

Рисунок 21 – входной файл

Пример работы программы:

```

D:\python\нс>python nntask2.py input1=test5.json
Файл для вывода не был введен.
Поэтому был установлен файл по умолчанию (func.txt)
Линейное представление функции сохранено в файл func.txt.

```

Рисунок 22 – запуск программы в консоли

Линейное представление функции: $G(D(A(), A()), F(E(B(), C()))))$

Пример входного файла:

```
nntask2.py test6.json ×
nc > {} test6.json > ...
1  {
2    "graph": {
3      "vertex": [
4        "v1",
5        "v2",
6        "v3",
7        "v4",
8        "v5",
9        "v6"
10     ],
11     "arc": [
12       {
13         "from": "v1",
14         "to": "v4",
15         "order": 1
16       },
17       {
18         "from": "v2",
19         "to": "v4",
20         "order": 2
21       },
22       {
23         "from": "v4",
24         "to": "v5",
25         "order": 1
26       },
27       {
28         "from": "v3",
29         "to": "v5",
30         "order": 2
31       },
32       {
33         "from": "v5",
34         "to": "v6",
35         "order": 1
36       }
37     ]
38   }
```

Рисунок 23 – входной файл

Пример работы программы:

```
D:\python\nc>python nntask2.py input1=test6.json
Файл для вывода не был введен.
Поэтому был установлен файл по умолчанию (func.txt)
Линейное представление функции сохранено в файл func.txt.
```

Рисунок 24 – запуск программы в консоли

Линейное представление функции: $v_6(v_5(v_4(v_1()), v_2()), v_3())$

3. Вычисление значения функции на графе

3.1 Описание задания 3

На входе:

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

б) Текстовый файл соответствий арифметических операций именам вершин:

$$\begin{cases} \\ a_1 : \text{операция}_1 \\ a_2 : \text{операция}_2 \\ \dots \\ a_n : \text{операция}_n \\ \end{cases}$$

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

Допустимы следующие символы операций:

+ – сумма значений,

***** – произведение значений,

exp – экспонирование входного значения,

число – любая числовая константа.

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

Способ проверки результата: результат вычисления, выведенный в файл.

3.2 Тестирование задания 3

Пример входного файла:

```
(v1, v3, 1), (v2, v3, 2)

{
  "v1" : 512,
  "v2" : 2,
  "v3" : "*"
}
```



Рисунок 25 – json представление входного файла

Линейное представление функции: $v_3(v_1(), v_2())$

Пример работы программы:

```
D:\python\nc>python nntask3.py input1=test1.txt oper1=oper.json output1=out.txt
Граф был успешно записан в файл
Значение функции, построенной по графу test1.txt и файлу oper.json сохранено в out.txt.
```

Рисунок 26 – запуск программы в консоли

Вычисленное значение: 1024

Пример входного файла:

(v1, v4, 1), (v2, v4, 2), (v2, v5, 1), (v3, v5, 2), (v4, v6, 1),
(v5, v6, 2), (v6, v7, 1)

```
{
    "v1" : 3,
    "v2" : 2,
    "v3" : 5,
    "v4" : "*",
    "v5" : "+",
    "v6" : "+",
    "v7" : "exp"
}
```

HC > {} test7.json > ...

```
1  {
2    "graph": {
3      "vertex": [
4        "v1",
5        "v2",
6        "v3",
7        "v4",
8        "v5",
9        "v6",
10       "v7"
11     ],
12     "arc": [
13       {
14         "from": "v1",
15         "to": "v4",
16         "order": 1
17       },
18       {
19         "from": "v2",
20         "to": "v4",
21         "order": 2
22       },
23       {
24         "from": "v2",
25         "to": "v5",
26         "order": 1
27       },
28       {
29         "from": "v3",
30         "to": "v5",
31         "order": 2
32       },
33       {
```

```

34     "from": "v4",
35     "to": "v6",
36     "order": 1
37 },
38 {
39     "from": "v5",
40     "to": "v6",
41     "order": 2
42 },
43 {
44     "from": "v6",
45     "to": "v7",
46     "order": 1
47 }
48 ]
49 }
50 }

```

Рисунок 27 – json представление входного файла

Линейное представление функции: $v_7(v_6(v_4(v_1(), v_2()), v_5(v_2(), v_3())))$

Пример работы программы:

```

D:\python\nc>python nntask3.py input1=test7.txt oper1=oper.json output1=out.txt
Граф был успешно записан в файл
Значение функции, построенной по графу test7.txt и файлу oper.json сохранено в out.txt.

```

Рисунок 28 – запуск программы в консоли

Вычисленное значение: 442413.3920089205

Пример входного файла:

$(v_1, v_2, 1), (v_1, v_2, 2), (v_2, v_6, 1), (v_3, v_5, 1), (v_4, v_5, 2), (v_6, v_7, 1), (v_5, v_7, 2)$

```

{
    "v1" : 1,
    "v2" : "+",
    "v3" : 5,

```



```
    "v4" : 12,  
    "v5" : "*",  
    "v6" : "exp",  
    "v7" : "+"  
}
```

HC > {} test8.json > ...

```
1  {
2    "graph": {
3      "vertex": [
4        "v1",
5        "v2",
6        "v3",
7        "v4",
8        "v5",
9        "v6",
10       "v7"
11     ],
12     "arc": [
13       {
14         "from": "v1",
15         "to": "v2",
16         "order": 1
17       },
18       {
19         "from": "v1",
20         "to": "v2",
21         "order": 2
22       },
23       {
24         "from": "v2",
25         "to": "v6",
26         "order": 1
27       },
28       {
29         "from": "v3",
30         "to": "v5",
31         "order": 1
32       },
33     ]
  }
```

```

34     "from": "v4",
35     "to": "v5",
36     "order": 2
37 },
38 {
39     "from": "v6",
40     "to": "v7",
41     "order": 1
42 },
43 {
44     "from": "v5",
45     "to": "v7",
46     "order": 2
47 }
48 ]
49 }
50 }

```

Рисунок 27 – json представление входного файла

Линейное представление функции: $v_7(v_6(v_2(v_1()), v_1()), v_5(v_3(), v_4()))$

Пример работы программы:

```

D:\python\нс>python nntask3.py input1=test8.txt oper1=oper.json output1=out.txt
Граф был успешно записан в файл
Значение функции, построенной по графу test8.txt и файлу oper.json сохранено в out.txt.

```

Рисунок 28 – запуск программы в консоли

Вычисленное значение: 67.38905609893065

Пример входного файла:

```

(v1,v4,1), (v2,v4,2), (v4,v5,1), (v3,v5,2), (v5,v6,1)

{
  "v1" : 1,
  "v2" : 10,
  "v3" : 5,
  "v4" : "+",
  "v5" : "*",
  "v6" : "exp"
}

```

```
nntask2.py  {} test6.json  X
nc > {} test6.json > ...
1  {
2    "graph": {
3      "vertex": [
4        "v1",
5        "v2",
6        "v3",
7        "v4",
8        "v5",
9        "v6"
10     ],
11     "arc": [
12       {
13         "from": "v1",
14         "to": "v4",
15         "order": 1
16       },
17       {
18         "from": "v2",
19         "to": "v4",
20         "order": 2
21       },
22       {
23         "from": "v4",
24         "to": "v5",
25         "order": 1
26       },
27       {
28         "from": "v3",
29         "to": "v5",
30         "order": 2
31       },
32       {
33         "from": "v5",
34         "to": "v6",
35         "order": 1
36       }
37     ]
38   }
```

Рисунок 29 – json представление входного файла

Линейное представление функции: $v_6(v_5(v_4(v_1()), v_2()), v_3())$

Пример работы программы:

```
D:\python\nc>python nntask3.py input1=test6.txt oper1=oper.json output1=out.txt
Граф был успешно записан в файл
Значение функции, построенной по графу test6.txt и файлу oper.json сохранено в out.txt.
```

Рисунок 30 – запуск программы в консоли

Вычисленное значение: $7.694785265142018e+23$

4. Построение многослойной нейронной сети

4.1 Описание задания 4

На входе:

- а) Текстовый файл с набором матриц весов межнейронных связей:

$M1 : [M1[1,1], M1[1,2], \dots, M1[1,n]], \dots, [M1[m,1], M1[m,2], \dots, M1[m,n]]$

$M2 : [M2[1,1], M2[1,2], \dots, M2[1,n]], \dots, [M2[m,1], M2[m,2], \dots, M2[m,n]]$

...

$Mp : [Mp[1,1], Mp[1,2], \dots, Mp[1,n]], \dots, [Mp[m,1], Mp[m,2], \dots, Mp[m,n]]$

- б) Текстовый файл с входным вектором в формате:

$$x_1, x_2, \dots, x_n.$$

На выходе:

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

Файл с выходным вектором – результатом вычислений НС в формате:

$$y_1, y_2, \dots, y_n.$$

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

4.2 Тестирование задания 4

Пример входного файла:

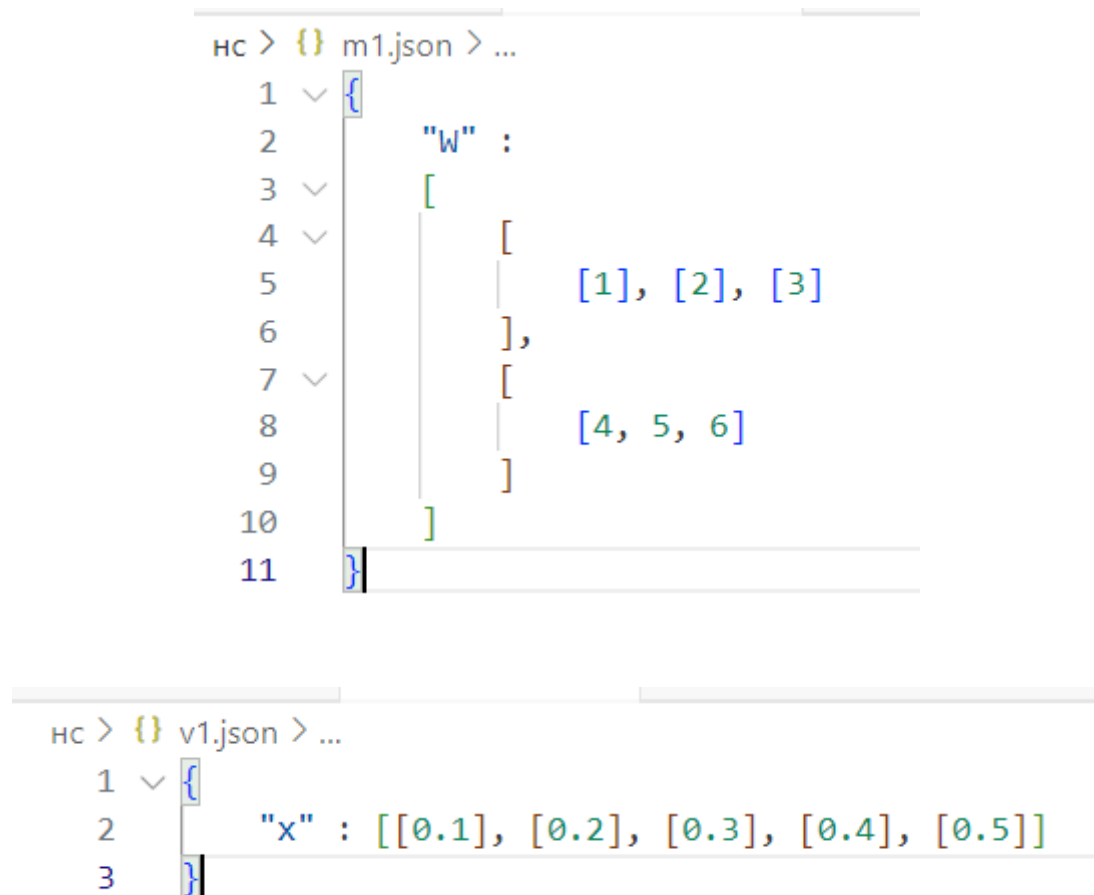


Рисунок 31 – json представления входных файлов матрицы и вектора

Пример работы программы:

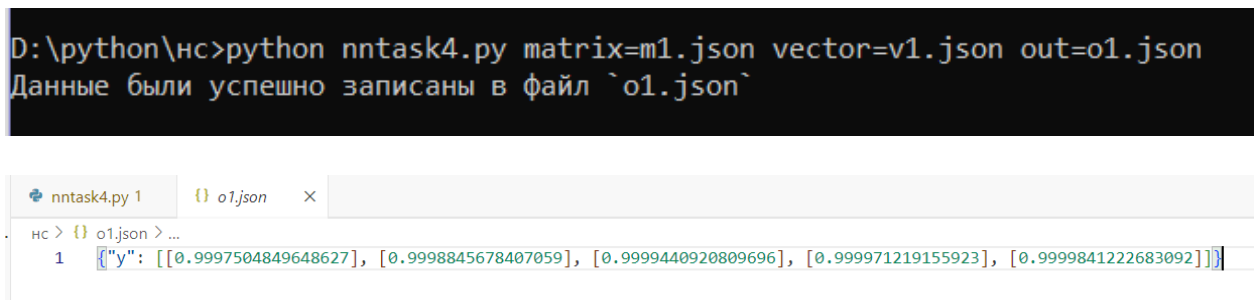


Рисунок 32 – запуск программы в консоли и результирующий вектор

Пример входного файла:

```
nntask4.py 1  {} m2.json  X
нс > {} m2.json > ...
1  {
2      "W" :
3      [
4          [
5              [1], [2], [3]
6          ],
7          [
8              [4, 5, 6, 7]
9          ]
10     ]
11 }
```

```
nntask4.py 1 X  {} m2.json  {} v2.json  X
нс > {} v2.json > ...
1  {
2      "x" : [[0.1, 0.2, 0.3], [0.2, 0.3, 0.4]]
3  }
```

Рисунок 33 – json представления входных файлов матрицы и вектора

Пример работы программы:

```
D:\python\nс>python nntask4.py matrix=m2.json vector=v2.json out=o2.json
shapes (3,1) and (3,) not aligned: 1 (dim 1) != 3 (dim 0): проверьте размерность слоя #1 и вектора x!
Умножение выполнить не удалось из-за некорректно заданных размерностей слоя #1 и вектора x.
```

Рисунок 34 – запуск программы в консоли

Пример входного файла:

```

нс > {} m3.json > ...
1  {
2    "w" :
3    [
4      [
5        [0.47519493033675375, 0.015705490366171526, 0.9433818257724572],
6        [0.48092032736144574, 0.13929695479782134, 0.6869903232566065],
7        [0.436988975888717, 0.20037642195993755, 0.17561406275527947]
8      ],
9      [
10     [0.042224071742743785, 0.15331022315027187, 0.464635658411239],
11     [0.6000159964796773, 0.22606113281552231, 0.5301212736820182],
12     [0.19651133783303198, 0.7498835958139106, 0.28721556978456597]
13   ],
14   [
15     [0.11837615025116721, 0.00927217999098906, 0.7504596929897048],
16     [0.5675946231090779, 0.9748635791740536, 0.30501309542663524],
17     [0.8574872089946126, 0.3047120321509168, 0.3376899733092712]
18   ]
19 ]
20 }
21

```

```

нс > {} v3.json > ...
1  {
2    "x" : [[2, 3, 5]]
3  }
4
5

```

Рисунок 35 – json представления входных файлов матрицы и вектора

Пример работы программы:

```

D:\python\нс>python nntask4.py matrix=m3.json vector=v3.json out=o3.json
Данные были успешно записаны в файл `o3.json`

```

```

нс > {} o3.json > ...
1  {"y": [[0.6595385964875882, 0.7974622186248654, 0.7420550486830448]]}

```

Рисунок 36 – запуск программы в консоли и результирующий вектор

5. Построение многослойной нейронной сети

5.1 Описание задания 5

На входе:

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

б) Текстовый файл с обучающей выборкой:

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

...

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

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

в) Число итераций обучения (в строке параметров).

На выходе:

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

1 : Ошибка1

2 : Ошибка2

...

N : Ошибка N

5.2 Тестирование задания 5

Пример входного файла:

```

hc > {} m1.json > ...
1  {
2      "w" :
3      [
4          [
5              [0.25, 0.75],
6              [0.1, 0.33],
7              [0.12, 0.56],
8              [0.31, 0.2]
9          ],
10         [
11             [0.3, 0.4, 0.7, 0.2]
12         ]
13     ]
14 }

```

```

hc > {} t1.json > ...
1  {
2      "in" :
3      [
4          [0.5, 0.5]
5      ],
6      "out" :
7      [
8          [0.125]
9      ]
10 }

```

```

hc > {} p1.json > ...
1  {
2      "n" : 25,
3      "lrate" : 0.8
4  }

```

Рисунок 37 – json представления входных файлов матрицы и вектора

Пример работы программы:

```

D:\python\nc>python nntask5.py matrix=m1.json param=p1.json train=t1.json out=o1.txt

Матрица весов W1 (1-й слой): [[0.25 0.75]
[0.1 0.33]
[0.12 0.56]
[0.31 0.2 ]]

Матрица весов W2 (2-й слой): [[0.3 0.4 0.7 0.2]]

Количество итераций: 25

Скорость обучения: 0.8
C:\Users\nika1\AppData\Local\Programs\Python\Python39-32\lib\site-packages\numpy\lib\function_base.py:5291: VisibleDepre
cationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays
with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating
the ndarray.
  arr = asarray(arr)

Данные успешно записаны в файл o1.txt

```

рис. 38 – вывод

1	При i = 1 значения функции ошибок: 0.5920286381785721
2	При i = 2 значения функции ошибок: 0.564556624198004
3	При i = 3 значения функции ошибок: 0.5356878665544065
4	При i = 4 значения функции ошибок: 0.5059279188227961
5	При i = 5 значения функции ошибок: 0.475859207517653
6	При i = 6 значения функции ошибок: 0.4460783625238085
7	При i = 7 значения функции ошибок: 0.4171313362852527
8	При i = 8 значения функции ошибок: 0.38946268901252945
9	При i = 9 значения функции ошибок: 0.3633890486436304
10	При i = 10 значения функции ошибок: 0.33909736390467016
11	При i = 11 значения функции ошибок: 0.31666136434183456
12	При i = 12 значения функции ошибок: 0.2960671253413246
13	При i = 13 значения функции ошибок: 0.27723999997032706
14	При i = 14 значения функции ошибок: 0.26006817779911107
15	При i = 15 значения функции ошибок: 0.24442094559278993
16	При i = 16 значения функции ошибок: 0.23016158607768006
17	При i = 17 значения функции ошибок: 0.21715575937730197
18	При i = 18 значения функции ошибок: 0.20527647126291154
19	При i = 19 значения функции ошибок: 0.1944066532338145
20	При i = 20 значения функции ошибок: 0.1844401731985894
21	При i = 21 значения функции ошибок: 0.17528187540797485
22	При i = 22 значения функции ошибок: 0.16684706152777856
23	При i = 23 значения функции ошибок: 0.15906068306464283
24	При i = 24 значения функции ошибок: 0.15185641492154667
25	При i = 25 значения функции ошибок: 0.14517571190179762
26	

Рисунок 38 – запуск программы в консоли и результирующий файл

Пример входного файла:

```

нс > {} m2.json > ...
1  ∨ {
2      "W" :
3      [
4          [
5              [1], [2], [3]
6          ],
7          [
8              [4, 5, 6]
9          ]
10     ]
11 }

```

```

нс > {} t2.json > ...
1  ∨ {
2      "in" :
3      [
4          [0.1], [0.2], [0.3], [0.4]
5      ],
6      "out" :
7      [
8          [0.9], [0.8], [0.7], [0.6]
9      ]
10 }

```

```

нс > {} p2.json > ...
1  {
2      "n" : 100,
3      "lrate" : 0.6
4  }

```

Рисунок 39 – json представления входных файлов матрицы и вектора

Пример работы программы:

```
D:\python\нс>python nntask5.py matrix=m2.json param=p2.json train=t2.json out=o2.txt
```

```
Матрица весов W1 (1-й слой): [[1]  
[2]  
[3]]
```

```
Матрица весов W2 (2-й слой): [[4 5 6]]
```

```
Количество итераций: 100
```

```
Скорость обучения: 0.6
```

```
C:\Users\nikal\AppData\Local\Programs\Python\Python39-32\lib\site-packages\numpy\lib\function_base.py:5291: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.  
arr = asarray(arr)
```

```
Данные успешно записаны в файл o2.txt
```

```
D:\python\нс>
```

```
нс > ≡ o2.txt
1 При i = 1 значения функции ошибок: 0.24965137935543213
2 При i = 2 значения функции ошибок: 0.24944708478343486
3 При i = 3 значения функции ошибок: 0.2494469476387328
4 При i = 4 значения функции ошибок: 0.24944681042609376
5 При i = 5 значения функции ошибок: 0.24944667314546723
6 При i = 6 значения функции ошибок: 0.2494465357968028
7 При i = 7 значения функции ошибок: 0.24944639838004992
8 При i = 8 значения функции ошибок: 0.24944626089515787
9 При i = 9 значения функции ошибок: 0.24944612334207614
10 При i = 10 значения функции ошибок: 0.24944598572075405
11 При i = 11 значения функции ошибок: 0.24944584803114084
12 При i = 12 значения функции ошибок: 0.2494457102731858
13 При i = 13 значения функции ошибок: 0.24944557244683785
14 При i = 14 значения функции ошибок: 0.24944543455204632
15 При i = 15 значения функции ошибок: 0.24944529658876027
16 При i = 16 значения функции ошибок: 0.24944515855692864
17 При i = 17 значения функции ошибок: 0.24944502045650035
18 При i = 18 значения функции ошибок: 0.2494448822874244
19 При i = 19 значения функции ошибок: 0.2494447440496496
20 При i = 20 значения функции ошибок: 0.24944460574312483
21 При i = 21 значения функции ошибок: 0.24944446736779877
22 При i = 22 значения функции ошибок: 0.2494443289236201
23 При i = 23 значения функции ошибок: 0.24944419041053753
24 При i = 24 значения функции ошибок: 0.24944405182849969
25 При i = 25 значения функции ошибок: 0.24944391317745518
26 При i = 26 значения функции ошибок: 0.24944377445735227
27 При i = 27 значения функции ошибок: 0.24944363566813968
28 При i = 28 значения функции ошибок: 0.2494434968097656
29 При i = 29 значения функции ошибок: 0.2494433578821786
30 При i = 30 значения функции ошибок: 0.24944321888532675
31 При i = 31 значения функции ошибок: 0.24944307981915845
32 При i = 32 значения функции ошибок: 0.24944294068362186
33 При i = 33 значения функции ошибок: 0.24944280147866513
34 При i = 34 значения функции ошибок: 0.24944266220423628
35 При i = 35 значения функции ошибок: 0.24944252286028337
36 При i = 36 значения функции ошибок: 0.2494423834467545
```

Рисунок 40 – запуск программы в консоли и результирующий файл

Пример входного файла:

HC > {} m3.json > ...

```
1  {  
2    "w" :  
3    [  
4      [  
5        [0.47519493033675375, 0.015705490366171526, 0.9433818257724572],  
6        [0.48092032736144574, 0.13929695479782134, 0.6869903232566065],  
7        [0.436988975888717, 0.20037642195993755, 0.17561406275527947]  
8      ],  
9      [  
10       [0.042224071742743785, 0.15331022315027187, 0.464635658411239],  
11       [0.6000159964796773, 0.22606113281552231, 0.5301212736820182],  
12       [0.19651133783303198, 0.7498835958139106, 0.28721556978456597]  
13     ],  
14     [  
15       [0.11837615025116721, 0.00927217999098906, 0.7504596929897048],  
16       [0.5675946231090779, 0.9748635791740536, 0.30501309542663524],  
17       [0.8574872089946126, 0.3047120321509168, 0.3376899733092712]  
18     ]  
19   ]  
20 }  
21
```

```
1  {  
2    "in" :  
3    [  
4      [-4, 1, 5],  
5      [7, -1, -4],  
6      [4, 14, 10],  
7      [-8, -18, 6]  
8    ],  
9    "out" :  
10   [  
11     [0, 0, 0],  
12     [1, 1, 1],  
13     [1, 1, 1],  
14     [0, 0, 0]  
15   ]  
16 }
```

```
нс > {} p3.json > ...  
1  ∨ {  
2    "n" : 25,  
3    "lr" : 0.8  
4  }
```

Рисунок 41 – json представления входных файлов матрицы и вектора

Пример работы программы:

```
D:\python\нс>python nntask5.py matrix=m3.json param=p3.json train=t3.json out=o3.txt  
  
Матрица весов W1 (1-й слой): [[0.47519493 0.01570549 0.94338183]  
[0.48092033 0.13929695 0.68699032]  
[0.43698898 0.20037642 0.17561406]]  
  
Матрица весов W2 (2-й слой): [[0.04222407 0.15331022 0.46463566]  
[0.600016 0.22606113 0.53012127]  
[0.19651134 0.7498836 0.28721557]]  
  
Матрица весов W3 (3-й слой): [[0.11837615 0.00927218 0.75045969]  
[0.56759462 0.97486358 0.3050131 ]  
[0.85748721 0.30471203 0.33768997]]  
  
Количество итераций: 25  
  
Скорость обучения: 0.8  
  
Данные успешно записаны в файл o3.txt
```


nc > ≡ o3.txt

```
1 При i = 1 значения функции ошибок: 0.2000546943212353
2 При i = 2 значения функции ошибок: 0.16991811766496046
3 При i = 3 значения функции ошибок: 0.14367987909005
4 При i = 4 значения функции ошибок: 0.12187406561044854
5 При i = 5 значения функции ошибок: 0.10460456977334781
6 При i = 6 значения функции ошибок: 0.09148512501723877
7 При i = 7 значения функции ошибок: 0.08166117841920054
8 При i = 8 значения функции ошибок: 0.074299427751067
9 При i = 9 значения функции ошибок: 0.06879536416411199
10 При i = 10 значения функции ошибок: 0.06470762781458543
11 При i = 11 значения функции ошибок: 0.06169847626215891
12 При i = 12 значения функции ошибок: 0.05950232168759868
13 При i = 13 значения функции ошибок: 0.05790711245965613
14 При i = 14 значения функции ошибок: 0.05674153704416596
15 При i = 15 значения функции ошибок: 0.055865154667598727
16 При i = 16 значения функции ошибок: 0.05515983258548446
17 При i = 17 значения функции ошибок: 0.05452110342711587
18 При i = 18 значения функции ошибок: 0.05384798753906572
19 При i = 19 значения функции ошибок: 0.05303059074710573
20 При i = 20 значения функции ошибок: 0.05194027245813776
21 При i = 21 значения функции ошибок: 0.050443783040230866
22 При i = 22 значения функции ошибок: 0.04847271408384185
23 При i = 23 значения функции ошибок: 0.04610900504916342
24 При i = 24 значения функции ошибок: 0.04355683640361677
25 При i = 25 значения функции ошибок: 0.041013965010396336
26
```

Рисунок 42 – запуск программы в консоли и результирующий файл

ПРИЛОЖЕНИЯ

Приложение 1 – задание 1

```
import sys,re
import json

class Graph:

    def __init__(self) -> None:
        self.vertex = set() # множество для хранения всех вершин графа
        self.adjc = {} # словарь для хранения списка дуг, исходящих из каждой
        вершины
        self.check = {} # словарь для хранения списка дуг, входящих в каждую
        вершину

    def graph_construction(self, data : list) -> None:
        for el in data:
            # Добавление вершин в множество vertex
            self.vertex.add(el[0])
            self.vertex.add(el[1])
            # Добавление входящей дуги в словарь check
            if el[1] not in self.check:
                self.check[el[1]] = []
            self.check[el[1]].append((el[0], int(el[2])))
            # Добавление исходящей дуги в словарь adjc
            if el[0] not in self.adjc:
                self.adjc[el[0]] = []
            self.adjc[el[0]].append((el[1], int(el[2])))
            # Сортировка списка вершин
            self.vertex = list(self.vertex)
            self.vertex.sort()
            # Проверка уникальности номеров входящих дуг для каждой вершины
            for _, vs in self.check.items(): # Проход по каждой вершине в словаре
                check
                n = len(vs)
                tmp = [''] * n
                for el, k in vs:
                    t = (k - 1) % n
                    if tmp[t] == el:
                        tmp[t + 1] = el
                    else:
                        tmp[t] = el
                test = set()
                for el in tmp:
                    test.add(el)
                    if el == '':
                        self.vertex = [-1]
                        print('В графе некорректно заданы номера!\nПроверьте
уникальность номеров.')
```

```
        return
```

```
class GraphCreation:
```

```
    def __init__(self, input, output) -> None:
        self._in = input
        self._out = output
        self.data = None
        self.graph = None
        self.g = Graph()

    def check_params(self) -> bool:
        if self._in is None:
            return False
        if self._out is None:
            self._out = "output.json"
            print(f"Файл для вывода не был введен.\n"
                  f"Поэтому был установлен файл по умолчанию ({self._out})")
        return True

    def read_from_file(self) -> bool:
        try:
            f = open(self._in, 'r', encoding='utf-8')
            self.data = f.read()
            f.close()
        except:
            print(f"Файла `{self._in}` не существует. "
                  "Проверьте корректность имени файла.")
            return False
        return True

    def data_parser(self) -> None:
        rawdata = self.data
        self.data = []
        cnt = 0
        tmp = []
        for el in re.split(r"\W+", rawdata):
            if el.isalnum():
                cnt += 1
                tmp.append(el)
                if cnt == 3:
                    if not tmp[2].isdigit():
                        self.data = []
                        return
                    cnt = 0
                    self.data.append(tmp)
                    tmp = []
```

```

def check_data(self) -> bool:
    if self.data == []:
        return False
    arcs_cnt = {}
    arcs = {}
    for el in self.data:
        s = f'{el[0]}-{el[1]}'
        if s not in arcs and s not in arcs_cnt:
            arcs_cnt[s] = 0
            arcs[s] = set()
        arcs_cnt[s] += 1
        arcs[s].add(el[2])
    for k in arcs.keys():
        if len(arcs[k]) != arcs_cnt[k]:
            return False
    return True

def get_graph(self) -> None:
    self.g.graph_construction(self.data)
    arc = []
    for el in self.g.adjc.keys():
        for tpl in self.g.adjc[el]:
            arc.append({ "from" : el
                        , "to" : tpl[0]
                        , "order" : tpl[1]})
    self.graph = {"graph" : {"vertex" : self.g.vertex, "arc" : arc}}

def write_to_file(self) -> bool:
    try:
        with open(self._out, 'w') as f:
            f.write(json.dumps(self.graph, indent=2))
    except:
        print(f"Не удалось записать в файл `{self._out}`.\n")
        return False
    return True

def graph_creation(self) -> None:
    if self.check_params():
        if self.read_from_file():
            self.data_parser()
            if not self.check_data():
                print('Некорректный ввод данных.\n'
                      'Проверьте уникальность порядковых номеров '
                      'у каждой из дуг.')
                return
            self.get_graph()
            if self.g.vertex == [-1]:
                return

```

```

        if self.write_to_file():
            print(f"Граф был успешно записан в файл")

def args_parser(argv : list) -> None:
    global graphs
    in1, in2, out1, out2 = None, None, None, None
    for el in argv:
        if "input1=" in el:
            in1 = el[el.find("=") + 1:]
        elif "input2=" in el:
            in2 = el[el.find("=") + 1:]
        elif "output1=" in el:
            out1 = el[el.find("=") + 1:]
        elif "output2=" in el:
            out2 = el[el.find("=") + 1:]
    if (in1 is None) and (in2 is None):
        print("Для корректной работы программы необходимо"
              " добавить в качестве аргументов названия файлов"
              " для ввода и вывода.\n"
              "Пример: input1=in.txt output1=out.json\n")
        return False
    if in1 is not None:
        if out1 is None:
            out1 = f"output1.json"
            print(f"Файл для вывода не был введен.\n"
                  f"Поэтому был установлен файл по умолчанию ({out1})")
            graphs.append(GraphCreation(in1, out1))
    if in2 is not None:
        if out2 is None:
            out2 = f"output2.json"
            print(f"Файл для вывода не был введен.\n"
                  f"Поэтому был установлен файл по умолчанию ({out2})")
            graphs.append(GraphCreation(in2, out2))
    return True

def main() -> None:
    args_parser(sys.argv)
    for g in graphs:
        g.graph_creation()

if __name__ == "__main__":
    graphs = []
    main()

```

Приложение 2 – задание 2

```

import json
import sys,re

```

```

def args_parser(argv : list) -> None:
    global graphs
    in1, out1 = None, None
    for el in argv:
        if "input1=" in el:
            in1 = el[el.find("=") + 1:]
        if "output1=" in el:
            out1 = el[el.find("=") + 1:]
    if (in1 is None):
        print("Для корректной работы программы необходимо"
              " добавить в качестве аргументов названия файлов"
              " для ввода и вывода.\n"
              "Пример: input1=in.json output1=out.txt\n")
        return False
    if in1 is not None:
        if out1 is None:
            out1 = f"func.txt"
            print(f"Файл для вывода не был введен.\n"
                  f"Поэтому был установлен файл по умолчанию ({out1})")
        return in1, out1

```

```

class Node:
    def __init__(self, node, parents=None, child=None) -> None:
        self.node = node
        self.from_nodes = [] if parents is None else parents
        self.to_nodes = [] if child is None else child

```

```

def get_graph_from_json(json_graph):

    with open(json_graph, "r") as f:
        data = json.load(f)

    graph_data = data['graph']

    graph = {}
    nodes = {}

    for vertex in graph_data['vertex']:
        graph[vertex] = []
        nodes[vertex] = Node(vertex)

    for arc in graph_data['arc']:
        from_vertex = arc['from']
        to_vertex = arc['to']
        graph[from_vertex].append(to_vertex)
        nodes[from_vertex].to_nodes.append(to_vertex)
        nodes[to_vertex].from_nodes.append(from_vertex)

    if cycle_check(graph):

```

```

        print("В графе обнаружен цикл.")
        raise RuntimeError("Цикл в графе.")

    return graph, nodes

def cycle_check(graph):
    path = set()
    def visit(vertex):
        path.add(vertex)
        for neighbour in graph.get(vertex, ()):
            if neighbour in path or visit(neighbour):
                return True
        path.remove(vertex)
        return False
    return any(visit(v) for v in graph)

def get_prefix_func(graph, output):

    def iterates(cur_node, nodes):
        from_nodes = [iterates(nodes[p], nodes)
                       for p in cur_node.from_nodes]
        return f'{cur_node.node}({", ".join(from_nodes)})'

    graph, nodes = graph
    root = None
    result = []
    for vertex in graph.keys():
        if not graph[vertex]:
            root = vertex
            result.append(iterates(nodes[root], nodes))

    with open(output, 'w') as file:
        file.write(", ".join(result))
    print(f"Линейное представление функции сохранено в файл {output}.")

def main():

    try:
        input, output = args_parser(sys.argv)
    except:
        print("Ошибка чтения аргументов!")
        return 0

    try:
        graph = get_graph_from_json(input)
    except:
        print("Не получилось считать граф с файла", input)
        return 0

    try:
        get_prefix_func(graph, output)

```

```

except:
    print("Не получилось составить префиксную функцию")
    return 0

if __name__ == "__main__":
    main()

```

Приложение 3 – задание 3

```

import sys, re
import json
import math
import os

def sum(x, y):
    return x + y

def multiply(x, y):
    return x * y

def exponent(x):
    return math.exp(x)

STRING_TO_OPERATION = {
    "+": "sum",
    "*": "multiply",
    "exp": "exponent"
}

class GraphCreation:

    def __init__(self, input, output) -> None:
        self._in = input
        self._out = output
        self.data = None
        self.graph = None
        self.g = Graph()

    def check_params(self) -> bool:
        if self._in is None:
            return False
        if self._out is None:
            self._out = f"output.json"
            print(f"Файл для вывода не был введен.\n"
                  f"Поэтому был установлен файл по умолчанию ({self._out})")

```



```

        return True

def read_from_file(self) -> bool:
    try:
        f = open(self._in, 'r', encoding='utf-8')
        self.data = f.read()
        f.close()
    except:
        print(f"Файла `{self._in}` не существует. "
              "Проверьте корректность имени файла.")
        return False
    return True

def data_parser(self) -> None:
    rawdata = self.data
    self.data = []
    cnt = 0
    tmp = []
    for el in re.split(r"\W+", rawdata):
        if el.isalnum():
            cnt += 1
            tmp.append(el)
            if cnt == 3:
                if not tmp[2].isdigit():
                    self.data = []
                    return
                cnt = 0
                self.data.append(tmp)
                tmp = []

def check_data(self) -> bool:
    if self.data == []:
        return False
    arcs_cnt = {}
    arcs = {}
    for el in self.data:
        s = f'{el[0]}-{el[1]}'
        if s not in arcs and s not in arcs_cnt:
            arcs_cnt[s] = 0
            arcs[s] = set()
            arcs_cnt[s] += 1
            arcs[s].add(el[2])
    for k in arcs.keys():
        if len(arcs[k]) != arcs_cnt[k]:
            return False
    return True

```

```

def get_graph(self) -> None:
    self.g.graph_construction(self.data)
    arc = []
    for el in self.g.adjc.keys():
        for tpl in self.g.adjc[el]:
            arc.append({ "from" : el
                        , "to" : tpl[0]
                        , "order" : tpl[1]})
    self.graph = {"graph" : {"vertex" : self.g.vertex, "arc" : arc}}

```

```

def write_to_file(self) -> bool:
    try:
        with open(self._out, 'w') as f:
            f.write(json.dumps(self.graph, indent=2))
    except:
        print(f"Не удалось записать в файл `{self._out}`.\n")
        return False
    return True

```

```

def graph_creation(self) -> None:
    if self.check_params():
        if self.read_from_file():
            self.data_parser()
            if not self.check_data():
                print('Некорректный ввод данных.\n'
                      'Проверьте уникальность порядковых номеров '
                      'у каждой из дуг.')
                return
            self.get_graph()
            if self.g.vertex == [-1]:
                return
            if self.write_to_file():
                print(f"Граф был успешно записан в файл")

```

```

class Graph:

```

```

    def __init__(self) -> None:
        self.vertex = set() # множество для хранения всех вершин графа
        self.adjc = {} # словарь для хранения списка дуг, исходящих из каждой
        вершины
        self.check = {} # словарь для хранения списка дуг, входящих в каждую
        вершину

```

```

    def graph_construction(self, data : list) -> None:
        for el in data:
            # Добавление вершин в множество vertex
            self.vertex.add(el[0])
            self.vertex.add(el[1])

```

```

        # Добавление входящей дуги в словарь check
        if el[1] not in self.check:
            self.check[el[1]] = []
        self.check[el[1]].append((el[0], int(el[2])))
        # Добавление исходящей дуги в словарь adjc
        if el[0] not in self.adjc:
            self.adjc[el[0]] = []
        self.adjc[el[0]].append((el[1], int(el[2])))
    # Сортировка списка вершин
    self.vertex = list(self.vertex)
    self.vertex.sort()
    # Проверка уникальности номеров входящих дуг для каждой вершины
    for _, vs in self.check.items(): # Проход по каждой вершине в словаре
check
        n = len(vs)
        tmp = [''] * n
        for el, k in vs:
            t = (k - 1) % n
            if tmp[t] == el:
                tmp[t + 1] = el
            else:
                tmp[t] = el
        test = set()
        for el in tmp:
            test.add(el)
            if el == '':
                self.vertex = [-1]
                print('В графе некорректно заданы номера!\nПроверьте
уникальность номеров.')
                return

def args_parser(argv : list) -> None:
    global graphs
    in1, op1, out1 = None, None, None
    for el in argv:
        if "input1=" in el:
            in1 = el[el.find("=") + 1:]
        elif "oper1=" in el:
            op1 = el[el.find("=") + 1:]
        elif "output1" in el:
            out1 = el[el.find("=") + 1:]
    if (in1 is None):
        print("Для корректной работы программы необходимо"
              " добавить в качестве аргументов названия файлов"
              " для ввода и вывода.\n"
              "Пример: input1=in.txt oper1=operations.json output1=out.txt\n")
        return False
    if op1 is not None:
        graphs.append(GraphCreation(in1, 'tmp.json'))
        return in1, op1, out1
    else:

```

```
print('Необходимо также ввести название файла, в котором содержатся  
операции')
```

```
'\nНапример oper1=op.json')
```

```
class Node:
```

```
def __init__(self, node, parents=None, child=None) -> None:  
    self.node = node  
    self.from_nodes = [] if parents is None else parents  
    self.to_nodes = [] if child is None else child
```

```
def get_graph_from_json(json_graph):
```

```
    with open(json_graph, "r") as f:  
        data = json.load(f)
```

```
    graph_data = data['graph']
```

```
    graph = {}  
    nodes = {}
```

```
    for vertex in graph_data['vertex']:  
        graph[vertex] = []  
        nodes[vertex] = Node(vertex)
```

```
    for arc in graph_data['arc']:  
        from_vertex = arc['from']  
        to_vertex = arc['to']  
        graph[from_vertex].append(to_vertex)  
        nodes[from_vertex].to_nodes.append(to_vertex)  
        nodes[to_vertex].from_nodes.append(from_vertex)
```

```
    if cycle_check(graph):  
        print("В графе обнаружен цикл.")  
        raise RuntimeError("Цикл в графе.")
```

```
    return graph, nodes
```

```
def cycle_check(graph):
```

```
    path = set()  
    def visit(vertex):  
        path.add(vertex)  
        for neighbour in graph.get(vertex, ()):   
            if neighbour in path or visit(neighbour):  
                return True  
        path.remove(vertex)  
        return False  
    return any(visit(v) for v in graph)
```

```
def get_prefix_func(graph):
```

```
    def iterates(cur_node, nodes):
```

```

        from_nodes = [iterates(nodes[p], nodes)
                        for p in cur_node.from_nodes]
        return f'{cur_node.node}({", ".join(from_nodes)})'

graph, nodes = graph
root = None
result = []
for vertex in graph.keys():
    if not graph[vertex]:
        root = vertex
        result.append(iterates(nodes[root], nodes))

return(", ".join(result))

def evaluate_graph(graph_string, ops):
    graph_string = graph_string.replace("()", "")
    for cur_op in ops:
        operation = None
        if ops[cur_op] in STRING_TO_OPERATION.keys():
            operation = STRING_TO_OPERATION[ops[cur_op]]
        else:
            operation = str(ops[cur_op])
        graph_string = graph_string.replace(cur_op, operation)
    return eval(graph_string)

def main():
    try:
        graph_input, operations_path, output = args_parser(sys.argv)
    except:
        print("Ошибка чтения аргументов!")
        return 0

    try:
        for g in graphs:
            g.graph_creation()
    except:
        print("Не удалось создать граф с помощью входного файла.")
        return 0

    graph = get_graph_from_json("tmp.json")
    os.remove("tmp.json")
    lin_interpretation = get_prefix_func(graph)

    try:
        with open(operations_path, 'r') as file:
            operations_dict = file.read()
            operations_dict = eval(operations_dict)
            result = evaluate_graph(lin_interpretation, operations_dict)
            with open(output, 'w') as file:

```

```

        file.write(str(result))
    print(f"Значение функции, построенной по графу {graph_input} и файлу" +
          f" {operations_path} сохранено в {output}.")
except:
    print("Ошибка сопоставления операций с описанием графа!")
    return 0

if __name__ == "__main__":
    graphs = []
    main()

```

Приложение 4 – задание 4

```

import autograd.numpy as np
import json, sys

class FeedForward:

    def __init__(self, ws) -> None:
        self.ws = ws
        self.n = len(ws)

    def sigmoid(self, x : float) -> float:
        return 1.0 / (1.0 + np.exp(-x))

    def go_forward(self, x : list) -> (list, list):
        try:
            sum = np.dot(self.ws[0], x)
            y = np.array([self.sigmoid(x) for x in sum])
        except ValueError as e:
            print(f"{e}: проверьте размерность слоя #1 и вектора x!\n"
                  f"Умножение выполнить не удалось из-за некорректно заданных"
                  f" размерностей"
                  f" слоя #1 и вектора x.")
            return [-1]
        for i in range(1, self.n):
            try:
                sum = np.dot(self.ws[i], y)
                y = np.array([self.sigmoid(x) for x in sum])
            except ValueError as e:
                print(f"{e}: проверьте размерность слоя #{i + 1} и вектора y!\n"
                      f"Умножение выполнить не удалось из-за некорректно заданных"
                      f" размерностей"
                      f" слоя #{i + 1} и вектора y.")
                return [-1]
        return y

```

```

def get_result(self, xs : list):
    ys = []
    for x in xs:
        new_val = list(self.go_forward(np.array(x)))
        if new_val == [-1]:
            ys.clear()
            break
        ys.append(new_val)
    return ys

def read_json_file(name) -> dict:
    try:
        f = open(name)
        data = json.load(f)
        f.close()
    except:
        print(f"Файла `{name}` не существует. "
              "Проверьте корректность имени файла.")
        exit(0)
    return data

def args_parser(argv : list) -> None:
    in1, in2, out = None, None, None
    for el in argv:
        if "matrix=" in el:
            in1 = el[el.find("=") + 1:]
        elif "vector=" in el:
            in2 = el[el.find("=") + 1:]
        elif "out=" in el:
            out = el[el.find("=") + 1:]

    if (in1 is None) or (in2 is None):
        print("Для корректной работы программы необходимо"
              " добавить в качестве аргументов названия файлов"
              " формата JSON для следующих параметров:\n"
              "matrix= -- файл, где лежат матрицы весов\n"
              "vector= -- файл с входными параметрами\n")
        exit(0)
    return in1, in2, out

def write_to_file(data, filename) -> bool:
    try:
        with open(filename, 'w') as f:
            json.dump(data, f)
    except:
        print(f'Ошибка записи данных в файл `{filename}`')

```

```

        exit(0)
    else:
        print(f'Данные были успешно записаны в файл `{filename}`')

def main():
    mtrx, vec, out = args_parser(sys.argv)
    if out is None:
        print("Отсутствует название файла для вывода.\n"
              "Файл для вывода был выбран по умолчанию (output.txt)")
        out = "output.txt"
    WS = read_json_file(mtrx)
    XS = read_json_file(vec)
    ws = []
    if "W" not in WS.keys():
        print('\nНекорректный ввод данных')
        return
    if "x" not in XS.keys():
        print('\nНекорректный ввод данных')
        return
    for w in WS['W']:
        ws.append(np.array(w))
    c = FeedForward(ws)
    ys = c.get_result(XS['x'])
    if ys != []:
        write_to_file({'y' : ys}, out)

if __name__ == '__main__':
    main()

```

Приложение 5 – задание 5

```

import autograd.numpy as np
import json, sys

class BackPropagation:

    def __init__(self, ws, n, lrate) -> None:
        self.ws = ws
        self.len_ws = len(ws)
        self.n = n
        self.lrate = lrate
        self.messages = None

    def sigmoid(self, x : float) -> float:
        return 1.0 / (1.0 + np.exp(-x))

    def d_sigmoid(self, y : float) -> float:
        return y * (1.0 - y)

```



```

def go_forward(self, x : list) -> (list, list):
    ys = []
    try:
        sum = np.dot(self.ws[0], x)
        y = np.array([self.sigmoid(x) for x in sum])
        ys.append(y)
    except ValueError as e:
        print(f"{e}: проверьте размерность слоя #1 и вектора x!\n"
              f"Умножение выполнить не удалось из-за некорректно заданных
размерностей"
              f" слоя #1 и вектора x.")
        exit(0)
    for i in range(1, self.len_ws):
        try:
            sum = np.dot(self.ws[i], y)
            y = np.array([self.sigmoid(x) for x in sum])
            ys.append(y)
        except ValueError as e:
            print(f"{e}: проверьте размерность слоя #{i + 1} и вектора y!\n"
                  f"Умножение выполнить не удалось из-за некорректно заданных
размерностей"
                  f" слоя #{i + 1} и вектора y.")
            exit(0)
    return ys

def train(self, x_vec : list, y_true : list) -> None:
    m = len(x_vec)
    self.messages = []
    for it in range(1, self.n + 1):
        errors = []
        for k in range(m):
            ys = self.go_forward(np.array(x_vec[k]))
            ys = np.insert(ys, 0, np.array(x_vec[k]), axis=0)
            if len(ys[-1]) != len(y_true[k]):
                print(f'\nРазмерность выходного значения алгоритма\n'
                      f'не совпадает с размерностью желаемого выходного
состояния {y_true[k]}')
                exit(0)
            j = len(ys) - 2
            e = ys[-1] - y_true[k]
            grad = e * self.d_sigmoid(ys[-1])
            self.ws[-1] = self.ws[-1] - self.lrate * ys[j] * grad
            for i in range(self.len_ws - 2, -1, -1):
                j -= 1
                grad = self.ws[i + 1] * grad * self.d_sigmoid(ys[j + 1])
                for t in range(len(self.ws[i])):
                    self.ws[i][t, :] = self.ws[i][t, :] - ys[j] * grad[:, t]
            * self.lrate
            errors.append(sum(e) / len(e))

```

```

        self.messages.append(f"При i = {it} значения функции ошибок:
{sum(errors) / len(errors)}\n")

def read_json_file(name) -> dict:
    try:
        f = open(name)
        data = json.load(f)
        f.close()
    except:
        print(f"Файла `{name}` не существует. "
              "Проверьте корректность имени файла.")
        exit(0)
    return data

def args_parser(argv : list) -> None:
    in1, in2, in3, out = None, None, None, None
    for el in argv:
        if "matrix=" in el:
            in1 = el[el.find("=") + 1:]
        elif "param=" in el:
            in2 = el[el.find("=") + 1:]
        elif "train=" in el:
            in3 = el[el.find("=") + 1:]
        elif "out=" in el:
            out = el[el.find("=") + 1:]

    if (in1 is None) or (in2 is None) or (in3 is None):
        print("Для корректной работы программы необходимо"
              " добавить в качестве аргументов названия файлов"
              " формата JSON для следующих параметров:\n"
              "matrix= -- файл, где лежат матрицы весов\n"
              "param= -- файл с параметром n (количество итераций)\n"
              "train= -- файл с входными и выходными параметрами\n")
        exit(0)
    return in1, in2, in3, out

def write_inf(mes : list, filename):
    mes = "".join(mes)
    try:
        f = open(filename, 'w')
        f.write(mes)
        f.close()
    except:
        print(f'Ошибка записи данных в файл {filename}')
        exit(0)
    else:
        print(f'\nДанные успешно записаны в файл {filename}')

def main():
    # elements = []

```

```

mtrx, par, train, out = args_parser(sys.argv)
if out is None:
    print("Отсутствует название файла для вывода.\n"
          "Файл для вывода был выбран по умолчанию (output.txt)")
    out = "output.txt"
mtrx = read_json_file(mtrx)
par = read_json_file(par)
train = read_json_file(train)
if "W" not in mtrx.keys():
    print('\nНекорректный ввод данных!\n'
          'Не удалось считать матрицы весов.')
    return
if "n" not in par.keys() or "lr" not in par.keys():
    print('\nНекорректный ввод данных!\n'
          'Не удалось считать параметры')
    return
if "in" not in train.keys() or "out" not in train.keys():
    print('\nНекорректный ввод данных!\n'
          'Не удалось считать входные/выходные значения')
    return
ws = []
for w in mtrx['W']:
    ws.append(np.array(w))
for i in range(len(ws)):
    print(f"\nМатрица весов W{i + 1} ({i + 1}-й слой): {ws[i]}")
print(f"\nКоличество итераций: {par['n']}")
print(f"\nСкорость обучения: {par['lr']}")
bp = BackPropagation(ws, par['n'], par['lr'])
xs = np.array(train["in"])
ys = np.array(train["out"])
bp.train(xs, ys)
write_inf(bp.messages, out)
if __name__ == '__main__':
    main()

```