**Отчет**

Студент: Фридрих Александр Сергеевич

Мною была проделана следующая работа:

1. Написан класс Graph с конструктором, написан генератор, создающий граф (приложение 1, приложение 2).
2. Написан метод, рисующий граф и сохраняющий его в папку (приложение 3).
3. Написано 4 метода, генерирующие разные типы задач с рисунками графов   
   (приложение 4).
4. Весь код тестировался на платформе lmstest2.
5. Выполнял роль лидера в команде (подгонял товарищей по цеху, организовал работу, ставил дедлайны).

*class* Graph:  
 *# пример вызова:  
 # graph = Graph(6, 2, 3) Создаем объект класса граф  
 # tasks = graph.task\_two(5) Генерим 5 задач из второго шаблона  
 # graph.create\_txt\_file(tasks, 1) Создаем txt файл и указываем что нужно рисовать графы  
 # graph.create\_zip\_file("image\_folder", "quiz.txt") Создаем zip архив в формате gift from media formats  
 def \_\_init\_\_*(*self*, n=6, min\_number\_of\_edges\_from\_vertex=1, max\_number\_of\_edges\_from\_vertex=2):  
 *# конструктор  
 """* ***:param*** *n число вершин:* ***:param*** *min\_number\_of\_edges\_from\_vertex: минимальное количество ребер из вершины* ***:param*** *max\_number\_of\_edges\_from\_vertex: максимальное количество ребер из вершины  
 """  
  
 self*.vertices = [value *for* value *in range*(1, n + 1)]  
 *self*.edges = []  
 *self*.min\_number\_of\_edges\_from\_vertex = min\_number\_of\_edges\_from\_vertex  
 *self*.max\_number\_of\_edges\_from\_vertex = max\_number\_of\_edges\_from\_vertex  
 *self*.graph = *dict*()  
 *self*.value = []

*приложение 1*

*def* \_\_graph\_generator(*self*):  
 *"""  
 Метод, который создает граф с минимальным и максимальным количеством ребер из вершин* ***:return****:  
 """* random\_vertices\_one = *self*.vertices.copy()  
 random\_vertices\_two = *self*.vertices.copy()  
 *self*.edges = []  
 random.shuffle(random\_vertices\_one)  
 random.shuffle(random\_vertices\_two)  
  
 *for* i *in range*(*len*(random\_vertices\_one)):  
 count = 0  
 *for* j *in range*(*len*(random\_vertices\_two)):  
 value = (random\_vertices\_one[i], random\_vertices\_two[j])  
 value\_reverse = (random\_vertices\_two[j], random\_vertices\_one[i])  
 *if* count < *self*.min\_number\_of\_edges\_from\_vertex *and* value *not in self*.edges \  
 *and* value\_reverse *not in self*.edges *and* random\_vertices\_one[i] != random\_vertices\_two[j]:  
 *self*.edges.append(value)  
 *self*.value.append(0)  
 count += 1  
 *elif self*.min\_number\_of\_edges\_from\_vertex <= count < *self*.max\_number\_of\_edges\_from\_vertex \  
 *and* value *not in self*.edges *and* value\_reverse *not in self*.edges \  
 *and* random\_vertices\_one[i] != random\_vertices\_two[j]:  
 *if* random.random() >= 0.5:  
 *break  
 self*.edges.append(value)  
 *self*.value.append(0)  
 count += 1  
  
 *self*.graph = {key: value *for* key, value *in zip*(*self*.edges, *self*.value)}

*приложение 2*

*def* paintilovka(*self*, choice\_values: *bool*, choice\_orientation: *bool*):  
 *"""  
 Метод, рисующий граф* ***:param*** *choice\_values: нужно ли рисовать веса на графе* ***:param*** *choice\_orientation: нужна ли ориентация у графа* ***:return****:  
 """* plt.figure(figsize=(4, 2))  
 *# рисуем граф  
 if* choice\_orientation:  
 draw\_graph = nx.DiGraph()  
 *else*:  
 draw\_graph = nx.Graph()  
  
 draw\_graph.add\_nodes\_from(*self*.vertices)  
 *for* key, value *in self*.graph.items():  
 draw\_graph.add\_edge(key[0], key[1])  
  
 pos = nx.spring\_layout(draw\_graph)  
 *# pos = nx.circular\_layout(draw\_graph)  
 # print(pos)* count = 0

nx.draw(draw\_graph, pos=pos, with\_labels=*True*)  
 *if* choice\_values:  
 nx.draw\_networkx\_edge\_labels(draw\_graph, pos=pos,  
 edge\_labels={(key[0], key[1]): value *for* key, value *in self*.graph.items()},  
 font\_color='red')  
  
 *# region сохранение графа* download\_folder = os.path.join(os.path.expanduser("~"), "Downloads", "folder\_tasks")  
 image\_folder = os.path.join(download\_folder, "image\_folder")  
 *if not* os.path.exists(image\_folder):  
 os.makedirs(image\_folder)  
  
 *# Создаем полный путь к файлу* path\_to\_img\_graph = ""  
 *# image\_folder = os.path.join(image\_folder, f"graph {0}.png")* counter\_file\_in\_directory = -1  
 *while True*:  
 counter\_file\_in\_directory += 1  
 path\_to\_img\_graph = os.path.join(image\_folder, f"graph {counter\_file\_in\_directory}.png")  
 *if not* os.path.exists(path\_to\_img\_graph):  
 *break  
  
 # print(path\_to\_img\_graph)* plt.savefig(path\_to\_img\_graph)  
 path\_to\_img\_graph = f"image\_folder/graph {counter\_file\_in\_directory}.png"  
 plt.close()  
 *return* path\_to\_img\_graph  
 *# endregion*

*приложение 3*

*def* task\_two(*self*, number\_of\_tasks, drawing\_graph, number\_forbidden\_answer=3):  
 *"""  
 Граф задан следующим образом: <{1,2,3,4,5,6}-множество вершин; {(1,3),  
 (2,1), (2,5), (3,2), (4,3), (4,5)}-множество дуг>. Выпишите сколько «-1» будет в  
 3-й строке матрицы инциденций.* ***:param*** *number\_of\_tasks: количество задач* ***:return*** *result\_tasks\_massive: массив с задачами в формате (вопрос, ответ)* ***:return*** *drawing\_graph: рисуем или не рисуем граф  
 """* result\_tasks\_massive = []  
 *for* \_ *in range*(number\_of\_tasks):  
 *self*.\_\_graph\_generator()  
 incidence\_matrix = [[0 *for* \_ *in range*(*len*(*self*.edges))] *for* \_ *in range*(*len*(*self*.vertices))]  
 *for* i, (vertex\_one, vertex\_two) *in enumerate*(*self*.edges):  
 incidence\_matrix[vertex\_one - 1][i] = 1  
 incidence\_matrix[vertex\_two - 1][i] = -1  
 row\_index = random.randint(0, *len*(*self*.vertices) - 1)  
 search\_element = random.choice([-1, 0, 1])  
 task\_text = "Граф задан следующим образом: <" + ", ".join(*map*(*str*, *self*.vertices)) + \  
 " -множество вершин; " + ", ".join(*map*(*str*, *self*.edges)) + " -множество дуг>. " \  
 f"Выпишите сколько «{search\_element}» будет в {row\_index + 1}-й строке матрицы инциденций."  
 answer = incidence\_matrix[row\_index].count(search\_element)  
  
 error\_count = 0  
 *while True*:  
 forbidden\_answer = random.sample(*range*(0, 10), number\_forbidden\_answer)  
 *if* answer *not in* forbidden\_answer:  
 *break* error\_count += 1  
 *if* error\_count % 100 == 0:  
 *print*(error\_count)  
  
 *if* drawing\_graph:  
 task\_text += "<br/>graph\_img"  
  
 path\_to\_graph\_img = *self*.paintilovka(*False*, *True*)  
 result\_tasks\_massive.append((task\_text, [answer], forbidden\_answer, path\_to\_graph\_img))  
 *return* result\_tasks\_massive

*def* task\_three(*self*, number\_of\_tasks, drawing\_graph, number\_forbidden\_answer=3):  
 *"""  
 Граф задан следующим образом: <{1,2,3,4,5}-множество вершин; {(1,3), (2,1),  
 (2,5), (3,2), (4,3), (4,5)}-множество дуг>. Какая из последовательностей  
 соответствует столбцу номер 1 матрицы смежностей для данного графа.  
 01000  
 00100  
 01010* ***:param*** *number\_of\_tasks: количество задач* ***:return*** *result\_tasks\_massive: массив с задачами в формате (вопрос, ответ)* ***:return*** *drawing\_graph: рисуем или не рисуем граф  
 """* result\_tasks\_massive = []  
 *for* \_ *in range*(number\_of\_tasks):  
 *self*.\_\_graph\_generator()  
 adjacency\_matrix = [[0 *for* \_ *in range*(*len*(*self*.vertices))] *for* \_ *in range*(*len*(*self*.vertices))]  
 *for* i, (vertex\_one, vertex\_two) *in enumerate*(*self*.edges):  
 adjacency\_matrix[vertex\_one - 1][vertex\_two - 1] = 1  
 adjacency\_matrix[vertex\_two - 1][vertex\_one - 1] = 1  
  
 row\_index = random.randint(0, *len*(*self*.vertices) - 1)  
 task\_text = "Граф задан следующим образом: <" + ", ".join(*map*(*str*, *self*.vertices)) + \  
 " -множество вершин; " + ", ".join(*map*(*str*, *self*.edges)) + " -множество дуг>. " \  
 f"Какая из последовательностей соответствует столбцу номер {row\_index} матрицы " \  
 f"смежностей для данного графа. Отсчет стобцов начинается с 0."  
 *if* drawing\_graph:  
 task\_text += "<br/>graph\_img"  
  
 answer = "".join([*str*(value) *for* value *in* adjacency\_matrix[row\_index]])  
  
 forbidden\_answer = *set*()  
 *while len*(forbidden\_answer) != number\_forbidden\_answer:  
 tmp\_mas = *map*(*str*, [random.choice([0, 1]) *for* \_ *in range*(*len*(*self*.vertices))])  
 gen\_for\_ans = "".join(tmp\_mas)  
 forbidden\_answer.add(gen\_for\_ans)  
 forbidden\_answer = *list*(forbidden\_answer)  
  
 path\_to\_graph\_img = *self*.paintilovka(*False*, *False*)  
 result\_tasks\_massive.append((task\_text, [answer], forbidden\_answer, path\_to\_graph\_img))  
 *return* result\_tasks\_massive

@staticmethod  
*def* task\_four(number\_of\_tasks, number\_forbidden\_answer=3):  
 *"""  
 В однородном графе степень вершины равна 5 Число ребер равно 35. Найдите число вершин.* ***:param*** *number\_of\_tasks: количество задач* ***:return*** *result\_tasks\_massive: массив с задачами в формате (вопрос, ответ)  
 """* result\_tasks\_massive = []  
 *for* \_ *in range*(number\_of\_tasks):  
 vertex\_degree = random.randint(5, 10)  
 answer = random.randint(5, 15)  
 count\_edges = vertex\_degree \* answer  
 task\_text = f"В однородном графе степень вершины равна {vertex\_degree} Число ребер равно {count\_edges}. " \  
 f"Найдите число вершин."  
 forbidden\_answer = [random.choice([answer + i + 1, answer - 1 - i]) *for* i *in range*(number\_forbidden\_answer)]  
  
 result\_tasks\_massive.append((task\_text, [answer], forbidden\_answer))  
 *return* result\_tasks\_massive

*def* task\_eight(*self*, number\_of\_tasks, drawing\_graph: *bool*, number\_forbidden\_answer=3):  
 *"""  
 Граф задан следующим образом: <{1,2,3,4,5,6}-множество вершин; {(1,3),  
 (2,1), (2,5), (3,2), (4,3), (4,5)}-множество дуг>.  
 Сколько ребер останется в графе после удаления вершины 2* ***:param*** *number\_of\_tasks: количество задач* ***:return*** *result\_tasks\_massive: массив с задачами в формате (вопрос, ответ)* ***:return*** *drawing\_graph: рисуем или не рисуем граф  
 """* result\_tasks\_massive = []  
 *for* \_ *in range*(number\_of\_tasks):  
 *self*.\_\_graph\_generator()  
 delete\_vertex = random.choice(*self*.vertices)  
 result\_vertices = *self*.vertices.copy()  
 result\_vertices.remove(delete\_vertex)  
 result\_edges = {(v1, v2) *for* v1, v2 *in self*.edges *if* v1 != delete\_vertex *and* v2 != delete\_vertex}  
  
 task\_text = "Граф задан следующим образом: <" + ", ".join(*map*(*str*, *self*.vertices)) + \  
 " -множество вершин; " + ", ".join(*map*(*str*, *self*.edges)) + " -множество дуг>. Запишите в " \  
 "Сколько ребер останется в графе после удаления вершины " + *str*(  
 delete\_vertex)  
 *if* drawing\_graph:  
 task\_text += "<br/>graph\_img"  
  
 answer\_text = *len*(result\_edges)  
 *while True*:  
 forbidden\_answer = random.sample(*range*(answer\_text // 2, answer\_text \* 2), number\_forbidden\_answer)  
 *if* answer\_text *not in* forbidden\_answer:  
 *break* path\_to\_graph\_img = *self*.paintilovka(*False*, *False*)  
 result\_tasks\_massive.append((task\_text, [answer\_text], forbidden\_answer, path\_to\_graph\_img))  
 *return* result\_tasks\_massive

*приложение 4*