МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное   
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики  
Кафедра технической кибернетики

**Отчет по лабораторной работе № 4**

Дисциплина: «Анализ социальных сетей»

Тема: «Визуализация связей в виде графов»

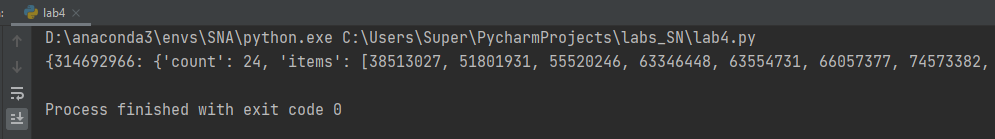
Выполнил: Овчинников И.В.

Группа: 6233-010402D

Самара 2021

Задание № 1. Запрос перечня друзей пользователя

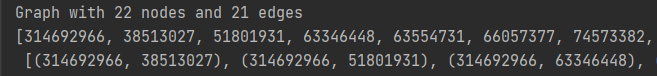
Получение списка друзей пользователя:



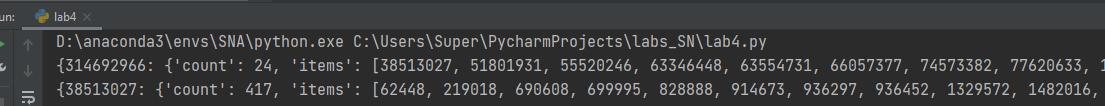
Задание № 2. Формирование графа друзей пользователя

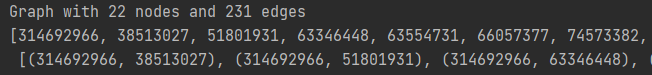
В данном пункте реализуются функции для визуализации графа с помощью библиотеки networkx.

В первом случае создается граф, вершинами которого являются узлы, содержащие id друзей. Ребрами графа являются связи пользователь-друг.



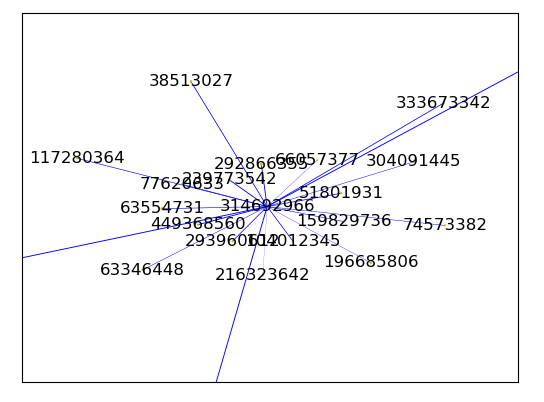
Во втором случае связываются еще и друзья между собой. В этом случае необходимо получить списки друзей друзей пользователя



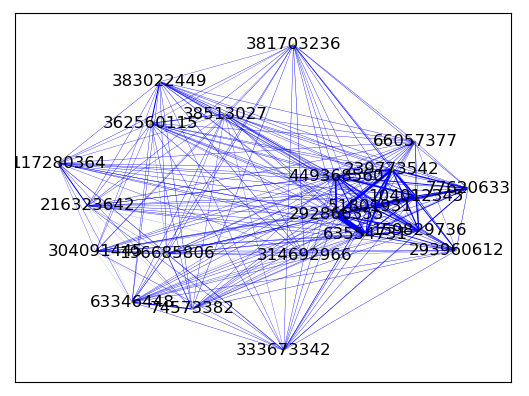


Задание № 3. Визуализация графа друзей пользователя

Визуализация сформированного графа с помощью библиотеки matplotlib для первого случая:

.

Визуализация сформированного графа для второго случая:



Код программы приведен в приложении А.

**Приложение А**

**Код программы**

import vk\_api  
import time  
import pickle  
import networkx as nx  
import numpy as np  
import matplotlib.pyplot as plt  
  
user1 = 314692966  
  
  
def auth\_handler():  
 """ При двухфакторной аутентификации вызывается эта функция.  
 """  
 # Код двухфакторной аутентификации  
 key = input("Enter authentication code: ")  
 # Если: True ‐ сохранить, False ‐ не сохранять.  
 remember\_device = True  
 return key, remember\_device  
  
  
def stop\_f(items):  
 print(items)  
  
  
def get\_groups\_users(friends\_list, tools):  
 friends\_out = {}  
 for friend in friends\_list:  
 try:  
 friends\_out[friend] = tools.get\_all('friends.get', 100, {'user\_id': friend})  
 except Exception:  
 friends\_out[friend] = []  
 time.sleep(1)  
 return friends\_out  
  
  
def main():  
 login, password = 'логин', 'пароль'  
 vk\_session = vk\_api.VkApi(  
 login, password,  
 auth\_handler=auth\_handler # функция для обработки двухфакторной аутентификации  
 )  
 try:  
 vk\_session.auth()  
 except vk\_api.AuthError as error\_msg:  
 print(error\_msg)  
  
 tools = vk\_api.VkTools(vk\_session)  
 friend\_list = []  
 friend\_list.append(user1)  
  
 # friends\_out = get\_groups\_users(friend\_list, tools)  
 #  
 # with open('friends\_out.pkl', 'wb') as output:  
 # pickle.dump(friends\_out, output, pickle.HIGHEST\_PROTOCOL)  
  
 with open('friends\_out.pkl', 'rb') as input:  
 friends\_out = pickle.load(input)  
  
 print(friends\_out)  
  
 # friends\_friends = get\_groups\_users(friends\_out[user1]['items'], tools)  
 #  
 # with open('friends\_friends.pkl', 'wb') as output:  
 # pickle.dump(friends\_friends, output, pickle.HIGHEST\_PROTOCOL)  
  
 with open('friends\_friends.pkl', 'rb') as input:  
 friends\_friends = pickle.load(input)  
  
 print(friends\_friends)  
  
 g = make\_graph(friends\_out, friends\_friends)  
 plot\_graph(g, 500)  
  
  
# def make\_graph(friends\_out, friends\_friends):  
# graph = nx.Graph()  
# graph.add\_node(user1, size=friends\_out[user1]['count'])  
# for i in friends\_out[user1]['items']:  
# try:  
# graph.add\_node(i, size=friends\_friends[i]['count'])  
# intersection = set(friends\_out[user1]['items']).intersection(set(friends\_friends[i]['items']))  
# graph.add\_edge(user1, i, weight=len(intersection))  
# except Exception:  
# print("err")  
# print(graph)  
# print(graph.nodes, "\n", graph.edges)  
# return graph  
  
def make\_graph(friends\_out, friends\_friends):  
 graph = nx.Graph()  
 graph.add\_node(user1, size = friends\_out[user1]['count'])  
 for i in friends\_out[user1]['items']:  
 try:  
 graph.add\_node(i, size = friends\_friends[i]['count'])  
 intersection = set(friends\_out[user1]['items']).intersection(set(friends\_friends[i]['items']))  
 graph.add\_edge(user1, i, weight=len(intersection))  
 except Exception:  
 print("err")  
 for i in range(len(friends\_out[user1]['items'])):  
 id1= friends\_out[user1]['items'][i]  
 for k in range(i+1, len(friends\_out[user1]['items'])):  
 id2= friends\_out[user1]['items'][k]  
 try:  
 intersection = set(friends\_friends[id1]['items']).intersection(set(friends\_friends[id2]['items']))  
 if len(intersection) > 0:  
 graph.add\_edge(id1, id2, weight=len(intersection))  
 except Exception:  
 print("err friend")  
 print(graph)  
 print(graph.nodes, "\n", graph.edges)  
 return graph  
  
  
def plot\_graph(graph, adjust\_nodesize):  
 # pos = nx.drawing.layout.circular\_layout(graph)  
 pos = nx.spring\_layout(graph, k=0.1)  
 # нормализуем размер вершины для визуализации. Оптимальное значение параметра  
 # adjust\_nodesize ‐ от 300 до 500  
  
 nodesize = [graph.nodes[i]['size'] / adjust\_nodesize for i in graph.nodes()]  
 # нормализуем толщину ребра графа. Здесь хорошо подходит  
 # нормализация по Standard Score  
  
 edge\_mean = np.mean([graph.edges[i]['weight'] for i in graph.edges()])  
 edge\_std\_dev = np.std([graph.edges[i]['weight'] for i in graph.edges()])  
 edgewidth = [((graph.edges[i]['weight'] - edge\_mean) / edge\_std\_dev / 2) for i in graph.edges()]  
  
 # создаем граф для визуализации  
 nx.draw\_networkx\_nodes(graph, pos, node\_size=nodesize, node\_color='y', alpha=0.9)  
 nx.draw\_networkx\_edges(graph, pos, width=edgewidth, edge\_color = 'b')  
 nx.draw\_networkx\_labels(graph, pos)  
 # сохраняем и показываем визуализированный граф  
 plt.savefig('saved')  
 plt.show()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()