

Sysoev Anatoly
Tomaschuk Korney
Chabrova Tatiana




```
import requests
import vk
import math
import time
import json

app_id = "7044123"
token = "6992e28f6992e28f6992e28fce69f99e94669926992e28f348f1922f8afaa0f540cc120"

# function which returns the number of members in a group
def number_of_members(group_id):
    payload = {'access_token' : token, 'v' : '5.61', 'group_id' : str(group_id),
              'fields' : 'members_count'}
    r = requests.get('https://api.vk.com/method/groups.getById', params=payload)
    number = r.json()['response'][0]['members_count']
    return number

# function which returns id of each memeber of a group
def get_members(group_id):
    res = []
    num = number_of_members(group_id)
    offset = 0
    for i in range(math.ceil(num / 1000)):
        payload = {'access_token' : token, 'v' : '5.61', 'group_id' : str(group_id), 'count' : '1000', 'offset' : str(offset)}
        r = requests.get('https://api.vk.com/method/groups.getMembers', params=payload)
        res += r.json()['response']['items']
        offset += 1000
    return res

# write ids of all users to txt file
users = get_members("30666517")
f = open("users.txt", "w")
print(users, file=f)
f.close()
```

ID приложения	7044123
Защищённый ключ	pum6TdH9luUlrIbTubTp 
Сервисный ключ доступа	6992e28f6992e28f6992e28fce69f99 
Состояние	Приложение включено и видно всем 
Первый запрос к API	

— — — — —

1.Graph on groups

Basing on the list of members we have created a **graph**, where each node was a **user** and every edge represented that two users had **more than 5 communities** in common.

```
# creating the graph of users by the rule of connection:
# users have more than 5 communities in common
# using NetWorkX library to visualize the graph
people = {}
G = nx.Graph()
for u in member_ids:
    people[u] = []
for u1 in member_ids:
    for u2 in member_ids:
        G.add_node(u1)
        G.add_node(u2)
        if u1 != u2:
            s1 = set(members[u1])
            s2 = set(members[u2])
            inter = s1.intersection(s2)
            if len(inter) > 5:
                G.add_edge(u1, u2)
                people[u1].append(u2)
```

After we have got the graph, we need to detect the strongest connected component got Karger's algorithm

```
# keeping only the largest connected component
pathes = []
cc = []
for i in people:
    pathes.append(dfs(people, i, path = set()))
for i in pathes:
    if len(i) > len(cc):
        cc = i
connected_component = {}
for i in cc:
    connected_component[i] = people[i]
```

Then we implemented the Karger's algorithm

In order to check hypothesis and to show dependencies we created two subgroups

```
# creating two groups for KARGER
new_g = change_graph(connected_component)
while len(new_g) > 2:
    contract_edge(choose_random_edge(new_g), new_g)
```

```
keys = list(new_g.keys())
group1 = list(keys[0])
group2 = list(keys[1])
```

Creating hypothesis: The first - percentage of male and the second - common country, third - friendship

```
def part_of_male(array):
    l = len(array)
    c = 0
    for i in array:
        if sex(i) == 0:
            l -= 1
        if sex(i) == 2:
            c += 1
    return (c / l) * 100

def H1(group1, group2):
    v1 = part_of_male(group1)
    v2 = part_of_male(group2)
    if abs((v1 + v2) - 100) < 30:
        return True
    return False

def common_country(array):
    c = {}
    l = len(array)
    for i in array:
        co = country(i)
        if not co in c:
            c[co] = 0
        c[co] += 1
    for i in c:
        if c[i] == l / 2:
            return i
    return 0

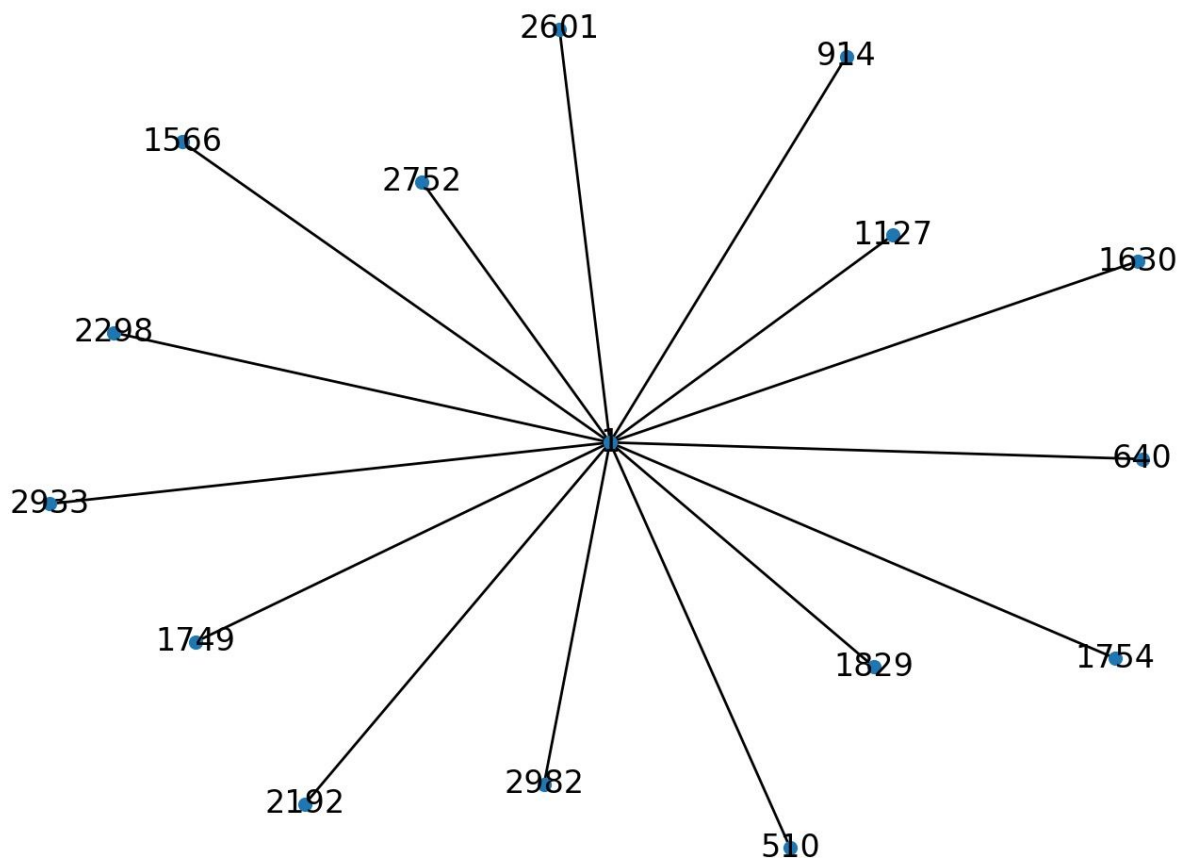
def H2(group1, group2):
    if common_country(group1) and common_country(group2):
        if common_country(group1) != common_country(group2):
            return True
    return False
```

After all we checked the hypothesis

```
# HYPOTHESIS 1 sex division
if H1(group1, group2):
    print("H1 is TRUE")
else:
    print("H1 is FALSE")

# HYPOTHESIS 2 common country
if H2(group1, group2):
    print("H2 is TRUE")
else:
    print("H2 is FALSE")
```

After that we ran the Louvain algorithm and drew the graph



```
# Louvain
partition = community.best_partition(G)
new = nx.Graph()
new.add_edges_from([pair for pair in partition.items() if pair[1]==1])

# we have got 4 groups of nodes. Lets check our hypothesis on them:

#groups
group1 = ['510', '640', '914', '1829', '1843', '2192', '2298', '2601', '2752', '2933', '3427']
group2 = ['510', '640', '914', '1127', '1566', '1630', '1749', '1754', '1829', '2192', '2298', '2601', '2752', '2933', '2982']
group3 = ['510', '640', '914', '1127', '1754', '1829', '2192', '2298', '2601', '2752', '2756', '2933']
group4 = ['510', '640', '914', '1211', '1607', '1754', '1829', '1843', '2192', '2298', '2601', '2752']

groups = [group1, group2, group3, group4]
for i in groups:
    if luiH1(i):
        print("H1 is TRUE")
    else:
        print("H1 is FALSE")

for i in groups:
    if luiH2(i):
        print("H2 is TRUE")
    else:
        print("H2 is FALSE")
```

Finally we checked the hypothesis on the groups we got after this algorithm implementation

Conclusion:

Karger: here only the first hypothesis worked, it really turned out that in the first group the majority of members was male and in the other - female. Other hyp-s were incorrect.

Louvain: here we had the same result. Only for one group it turned out to be that the majority of its users were of one gender.

2.Graph on hometowns

We already had list of ids of members of the group. We had to implement function which collect information about hometowns of each user, check if it is and existing town in Russia and write users with their hometowns in txt file.

```
import requests
import vk
import json

token= "6992e28f6992e28f6992e28fce69f99e94669926992e28f348f1922f8afaa0f540cc120"
people = []
cities = []
petersburg = ["санкт-петербург", "питер", "spb", "cnб", "peter", "saint-petersburg", "st-petersburg", "петербург"]
moscow = ["москва", "moscow"]
with open("cities.txt", "r") as c:
    for line in c.readlines():
        line = line.strip()
        cities.append(line.lower())
with open("users.txt", "r") as file:
    for line in file.readlines():
        line = line.strip()
        line = line.split()
        i = 0
        for user in line:
            user = user[:-1]
            payload = {'access_token' : token, 'v' : '5.52', 'user_ids' : str(user), 'fields': 'home_town'}
            r = requests.get('https://api.vk.com/method/users.get', params=payload)
            info = r.json()["response"][0]
            if "home_town" in info:
                town = info["home_town"].lower()
                if town == " " or town == "":
                    continue
                if town in petersburg:
                    town = "санкт-петербург"
                if town in moscow:
                    town = "москва"
                if town not in cities:
                    continue
            else:
                continue
            name = info["first_name"]
            surname = info["last_name"]
            person = []
            person.append(user)
            person.append(town)
            people.append(person)
            i += 1
            if i == 1000:
                break
fout = open("t.txt", "w")
print(people, file = fout)
fout.close()
```

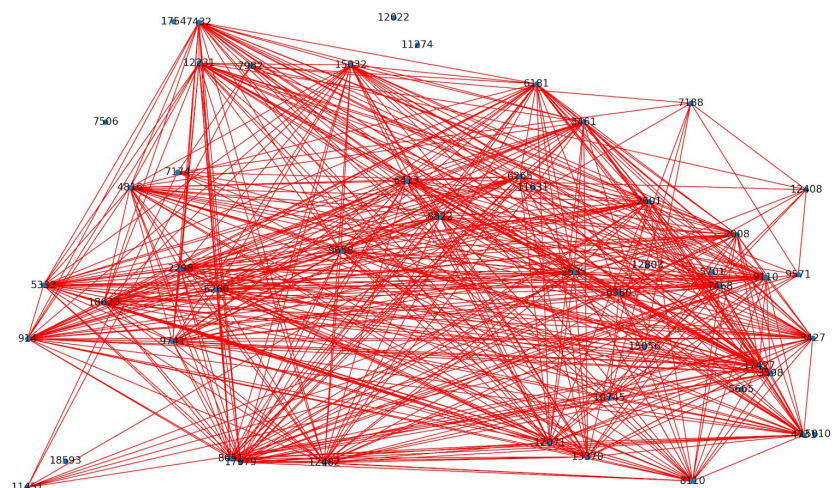

To draw this graph we had to create dictionary, where each node is a key in dictionary and its value is an array of connected nodes.

```
# parsing file with members of group and creating list of them
people = []
with open("t.txt", "r") as f:
    i = 0
    for i in f.readlines():
        i = i.replace("[", "")
        i = i.replace("'", "")
        i = i.split("]")
        k = 0
        for person in i:
            person = person.replace(" ", "")
            person = person.split(",")
            # was 4
            if len(person) != 3:
                continue
            people.append(person[1:])

# creating dictionary which represents the graph
g = {}
for person1 in people:
    g[person1[0]] = []
    for person2 in people:
        if person1[1] == person2[1] and person1 != person2:
            g[person1[0]].append(person2[0])
```

Then we used `networkx` and `matplotlib` libraries to draw the graph.

```
# drawing the graph (nodes are people; nodes are connected if poeple are from the same city)
graph = nx.Graph()
for person in people:
    graph.add_node(person[0])
for person1 in people:
    for person2 in people:
        if (person1[1] == person2[1]):
            graph.add_edge(person1[0], person2[0])
nx.draw(graph, node_size=50, with_labels=True, font_size=10)
plt.show()
```



To apply Karger's algorithm to the graph, the graph has to be connected. Thus, firstly we used to DFS algorithm and obtained subgraph (biggest connected component) and then applied Karger's to it.

```
# applying dfs algorithym to find all connected components and idetify the biggest one
pathes = []
cc = []
for i in g:
    pathes.append(dfs(g, i, path = set()))

for i in pathes:
    if len(i) > len(cc):
        cc = i

# creating subgraph
connected_component = {}
for i in cc:
    connected_component[i] = g[i]

# change form of the graph to apply karger's algorithym
new_g = change_graph(connected_component)

# applying karger's algorithym
while len(new_g) > 2:
    contract_edge(choose_random_edge(new_g), new_g)

# writting result in file
file = open("karger.txt", 'a')
for i in new_g:
    print(i, file = file)
    mincut = len(new_g[i])
print(mincut)
file.close()
```

Finally, we obtained two communities which we will check for hypothesis.

Firstly, we decided to check if majority of people in communities are the same gender. For this graph it turned out to be false.

Sendodly, we assumed that people in communities may be friends to each other. This hypothesis also appeared to be not true.

Finally, checked if people in one community have common groups and it turned out to be true.

All people in both communities have at least one common group.

Then we applied Luovain algorithm to the graph on hometowns and it appeared to produce the same partitions. Such situation appeared because in this graph in connected components all vertices are connected with each other. Thus, there was no sense to check our hypothesis on the communities of this graph.

