Federal Autonomous Educational Institution of Higher Professional Education

National Research University “Higher School of Economics”

Faculty of computer science

Educational program Data science and busyness analytics

Baccalaureate

**01.03.02 Applied Mathematics and Information Science**

**Report**

**Hands-on training**

Conducted by:

*StepanovYaroslav*

*Lobachev Nikita*

*Andreev Mikhail*

**Verified by:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(occupation, full name, chief of organization/ NRU HSE)*

*\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*(grade) (signature)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_*

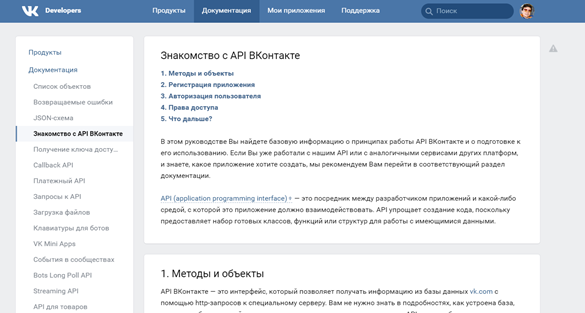
*(dates)*

Community detection in VK

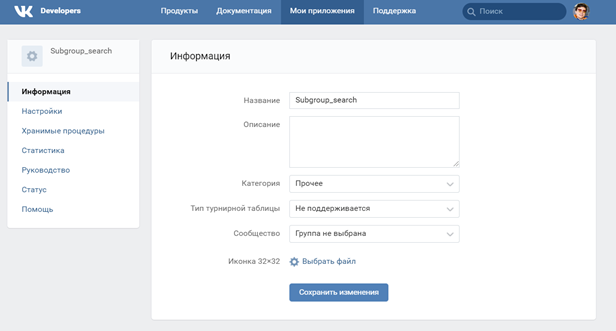
From the very beginning of our project, we faced some troubles related with managing our goals. After a short brainstorming, we have decided to look for the sub communities in a group of users that are subscribed to one public in the social network. Subsequently we decided to connect our results with closed communities of friends in that very public.

**Obtaining data**

All the data present in the project was taken from official VK API service.



First step was to get used to the main features of its documentation. Then following all the instructions given, you have to create app that will have its own access rights.



From this point, you can get the access token, which is the key that is required to send the request to VK database. After that, reading of “request” library is required as well as documentation of methods. Theoretically, it is possible to work with VK API only using address string, but for comfort of usage, it is better to work through some IDE. All work here is in sending some message to VK servers and getting the answer. Key point here are parameters of your message. Every method of this system might have its own parameters; every method is proceed as an URL ending. Library request allows us to send those parameters as arguments of a function; all the parameters have to be in a form of tuples. 

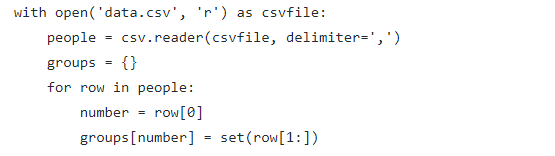
Problem that could appear if we take a bigger group is that VK API requests have limitations for developers. With ordinary account, one can get only 1000 objects from 1 request and it is framed by not more than 3 requests per second. It is done to protect servers from attacks. However if app is installed by certain amount of users, developer will get more rights. We just used a pause for one second after every third request.

Another important thing is that all data that is obtained is in JSON format initially and as it is easier to work with it in another format, it was translated to CSV.

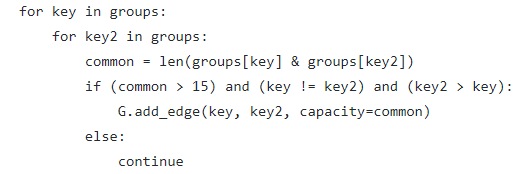
All the information was collected from <https://vk.com/sashershakov> group.

**Creating a graph**

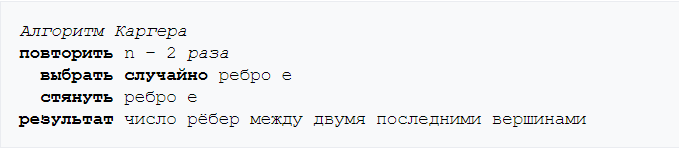
After collecting data from a VK community, we got a CSV file, where the first ID was a user and others in a row were groups for which he or she was subscribed.



Then, using NetworkX library we created a corresponding graph. As it can be seen in the code if there were more than 15 common groups between users, we added an edge between the corresponding people.

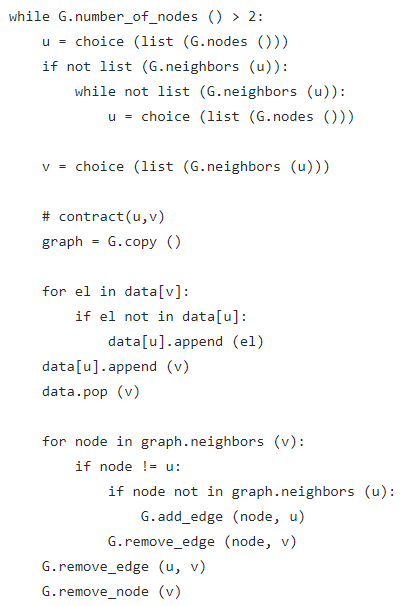


**Karger’s algorithm**

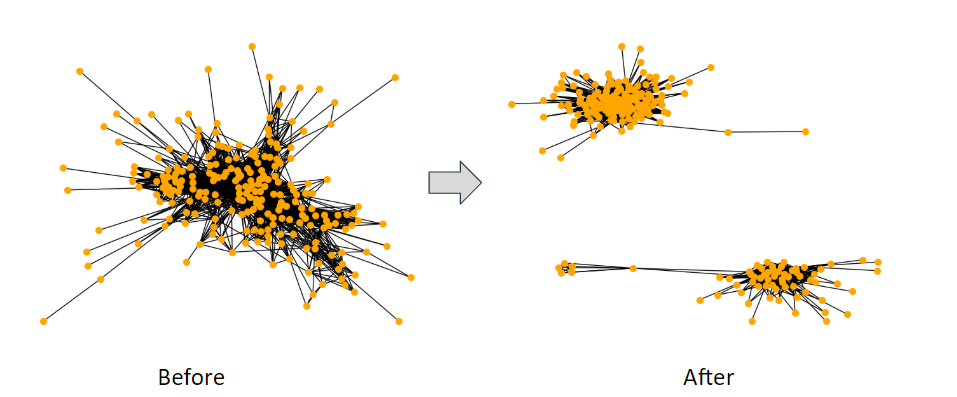


Using Karger’s algorithm we could find two distinct subcommunities.

While there were more than nodes in the graph, we randomly contracted them and finally obtained a minimum cut and consequently two subcommunities. There appeared a problem that we could not get information about contracted nodes in a multi-node if we used built-in NetworkX functions “contract\_node” or “contract\_edge”. Therefore, we decided to use dictionary to solve the issue and we saved information about all the nodes, which were contracted there.



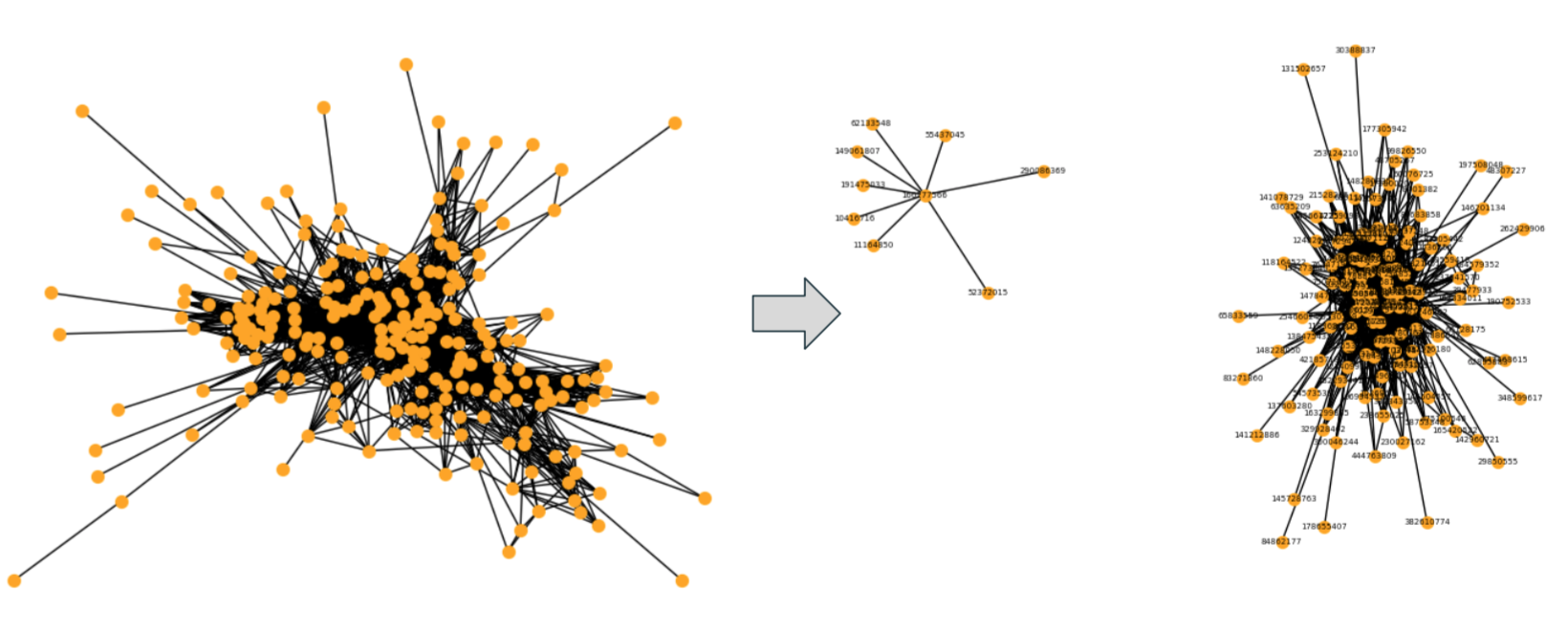
It is worth mentioning that Karger’s algorithm doesn’t give a good result immediately after just one usage as it is random. Therefore, we executed it approximately 100 times to achieve the best outcome.



**NetworkX minimum\_cut function**

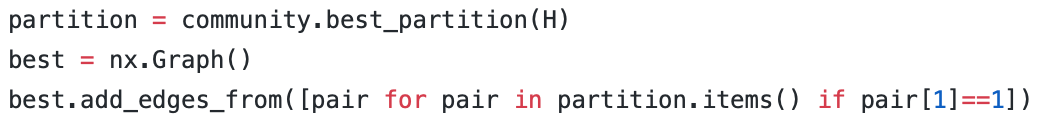
We can also find two distinct subcommunities using a function called “minimum\_cut” from NetworkX package. However, there is a problem with using this function, because we have to have “source” and “sink” as required parameters. So, we should randomly choose two nodes to be source and sink. After its work function will return two values: integer “cut\_value”,which is value of the minimum cut, and pair of sets “partition”,which represents nodes that define a minimum cut. In our case, we need only second return value. After getting two pairs of distinct nodes, we simply need to delete edges between nodes from different groups. However, in most cases one of the sets from “partition” consists only from one element, which in fact can’t be really considered as a subcommunity, so we have to choose new source and sink and repeat the procedure. As it requires time, the algorithm is working relatively slowly.

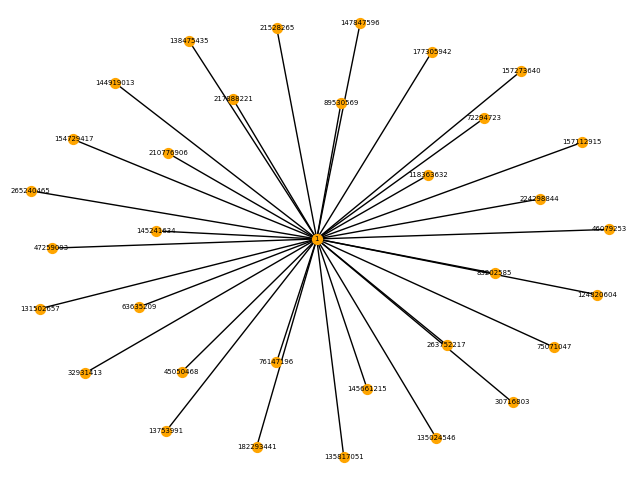
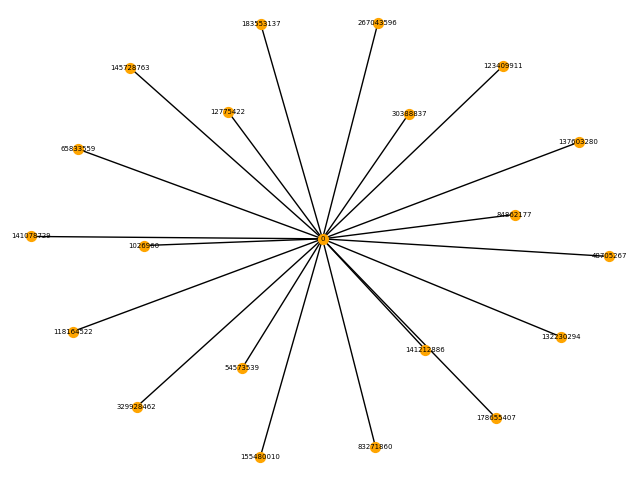
****

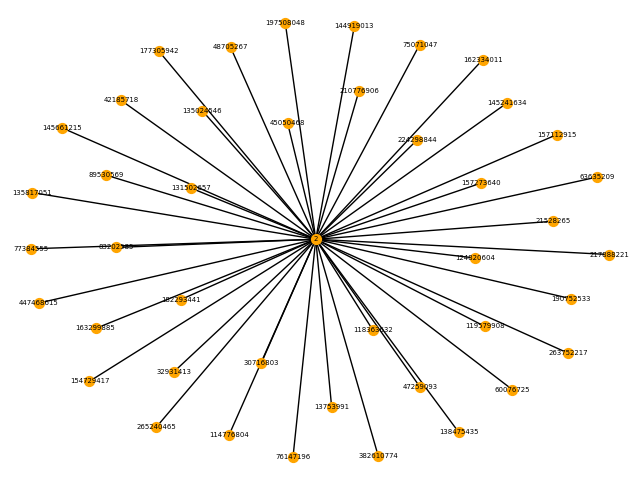
****

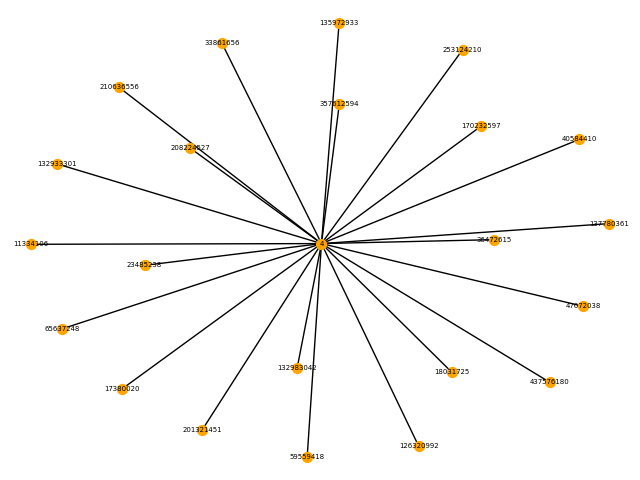
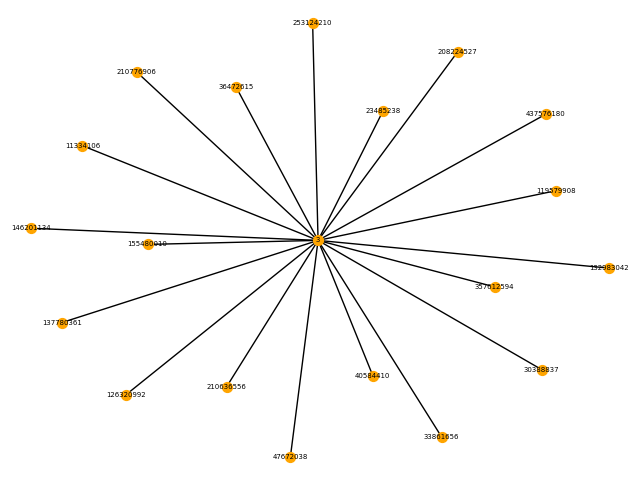
**Best Partition method**

We can get subcommunities using “best\_partition” from community,which uses Louvain method to get subcommunities. The method returns a dictionary “partition” with communities as values. In our particular case, there are 5 subcommunities.



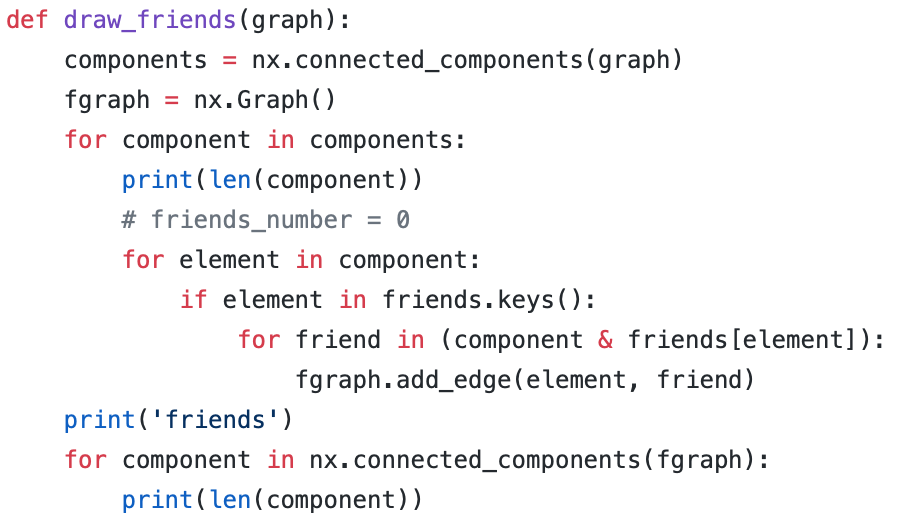


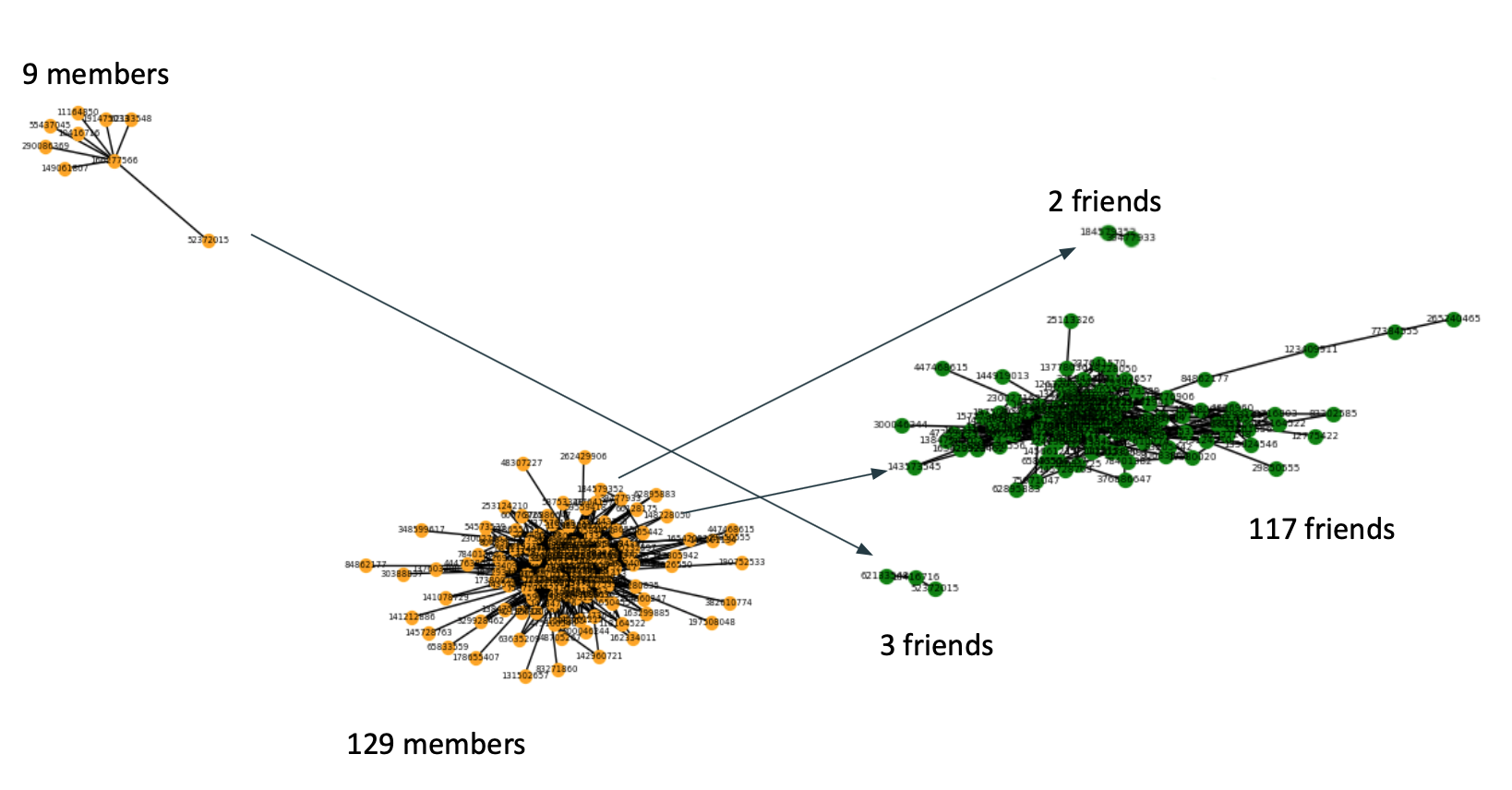




**Dependency between obtained subcommunities and friendship**

After we achieved different subcommunities, we wanted to figure out dependency between two people being members of one subcommunity and them being friends. To do so, we analyse each node of connected components of our final graph and check whether other nodes of this connected component are friends with the node. If it is so,we draw an edge in our new graph of friends.





From the graph it can be seen, that people in one subcommunity are tend to be friends with at least one member.