WSWT – Home Assignment 1.1

Gökhan Polat, M#0830690

Task 1: Crawling a TWO-MODE NETWORK

My decision was two crawl user and their top artists. Firstly I began with a **well-known artist** and not with a user for the risk that it couldn't be existing anymore. From my favorite artists I requested fans and added some of them (**randomly**) to the networkX graph as edges. After crawling from user in the graph I also added the artists to the graph with the user as edges.

After crawling from both sides I checked routinely whether the minimum size of fans and artists (**100**) is reached. If that was the case the second check were the average degrees which should have a particular rate (**> 1.5**). Finally if the graph was **connected** the main loop stops and the data are stored to files.

Task 2: Crawling a ONE-MODE NETWORK

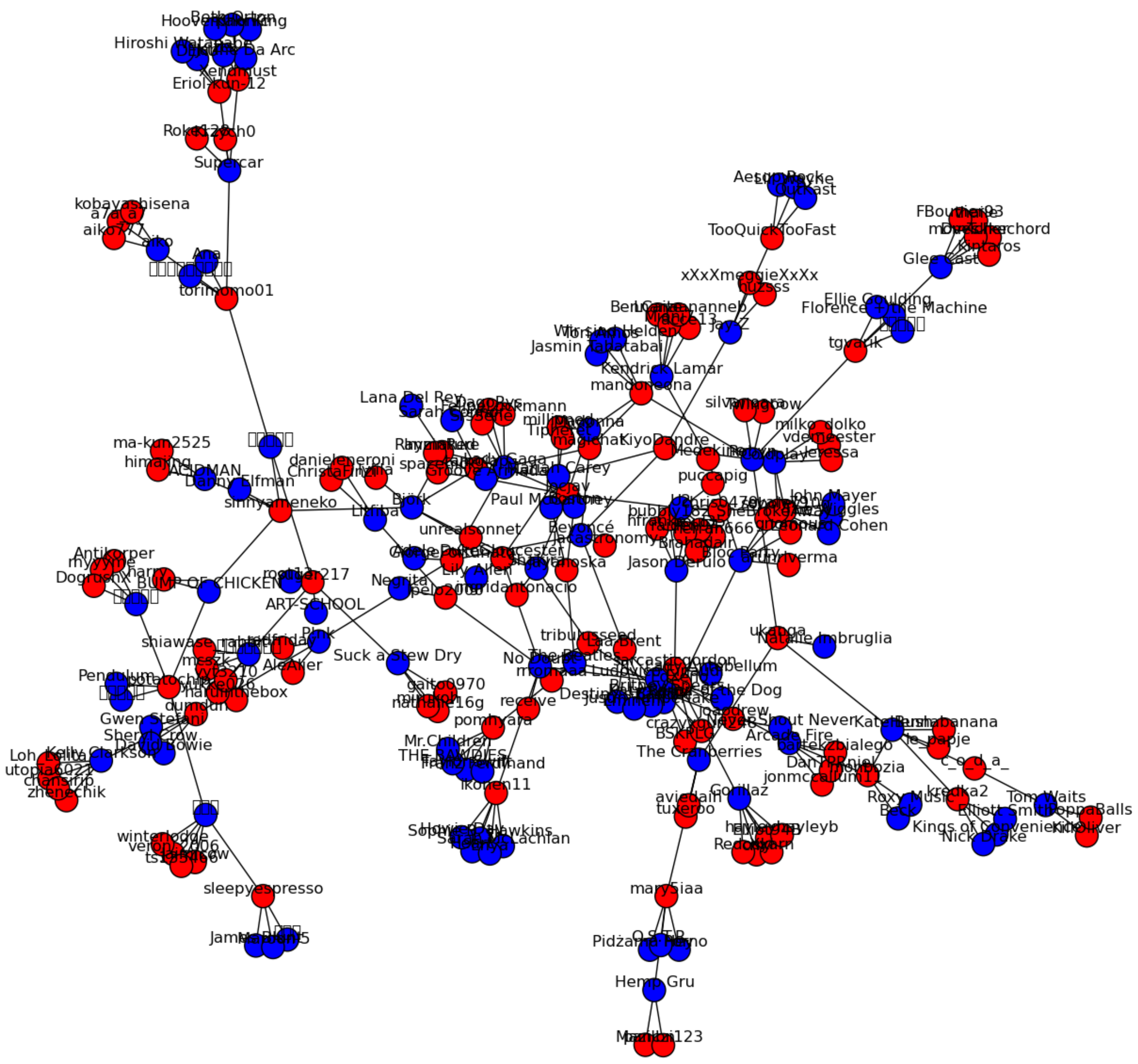
The strategy and the algorithm is similar to my strategy as in two-mode network crawler.
First step was to crawl from a **hard given user** his friends and save them to the networkX graph and other data structures. The crawler continues then by taking any random user from the graph and requesting his friend and adding them to the network. The finishing strategy was the same as in Task 1. After reaching all of the conditions the crawler stops and saves data to the files.

Task 3: Plotting and statistics of two-mode networks

3.1. plot/plot.py

The Plot shows two mode network and is **connected** because none of the nodes is separate (no island).

- Red colored nodes represents the **actors** (124).
- Blue colored nodes represents the **events** (102).



plot/visualNetworkA.png

3.2. isConnected2.m

```
connected2 =  1
```

The octave function returns 1 applied on network A.
The first step is two check whether there is a column or row only filled with zeros. This would mean that there is a node which has no edges to other nodes. The result would be not connected immediate. For checking of the connectivity I used two functions of the set theory where it was possible to check if any edge i**ntersects** with other nodes. In that case I only added the nodes by the **union** function. After every sub iteration I checked the size of set (nodes) which should be equal to the size of rows + columns.

The pseudo code below represents nearly my developed algorithm:

```
set = [1,2] // initialize set by any edge
for i → m(1)
     for j → m(2)
     if m(1,2)
          if(intersect(m(1,2), set)
          set = union(set, m(1,2))
     if(size(nodes) == size(set))
          return 1;
```

3.3. calcParticipationRates.m

```
av_rate_of_participation =  1.9113
av_size_of_events =  2.3235
```

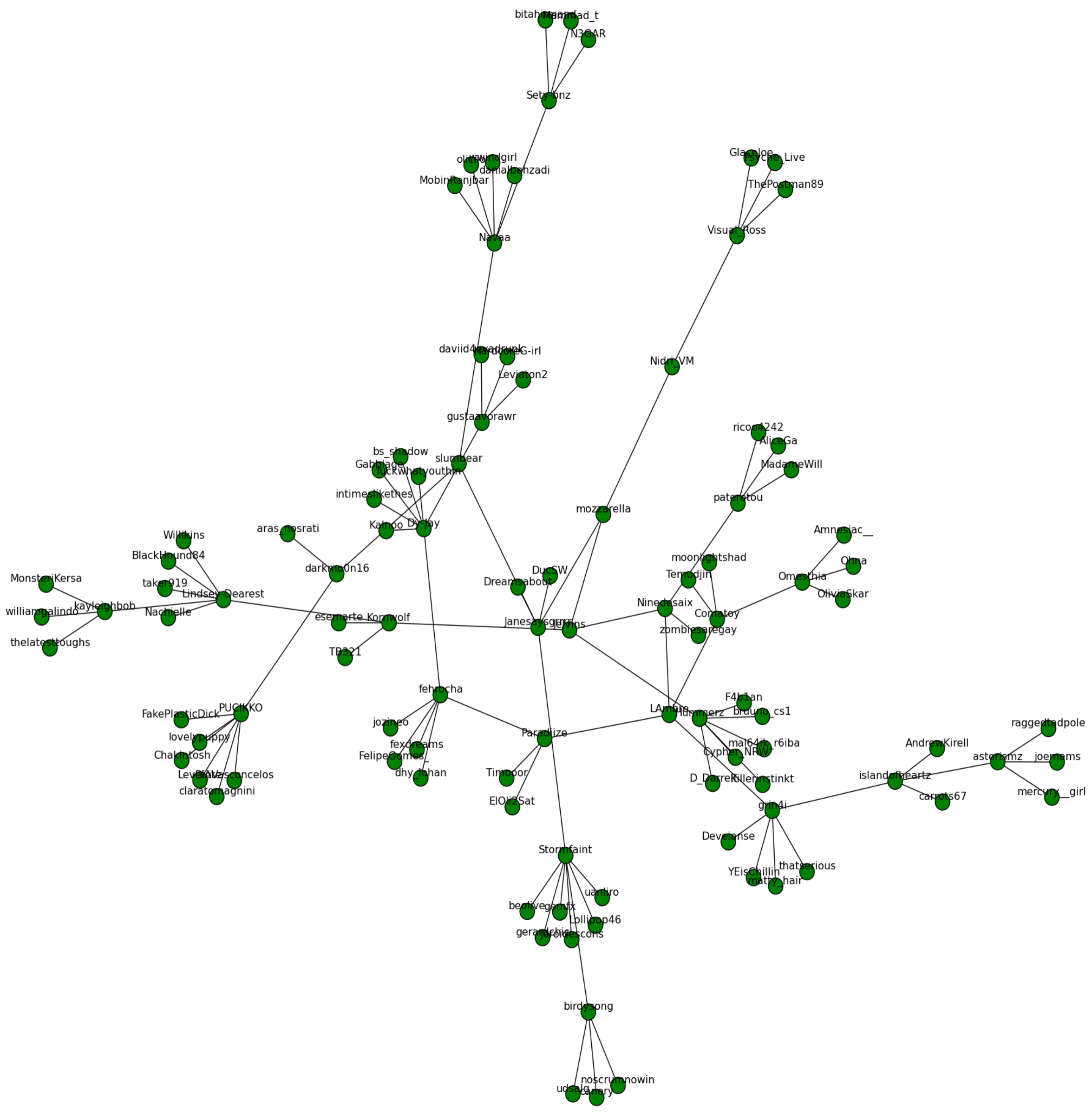
Average size of events → number of edges : events
Average size of participation → number of edges : actors

Task 4: Plotting and statistics of one-mode networks

4.1. plot/plot.py

The Plot shows one mode network and is **connected** because none of the nodes is separate (no island).

- Green nodes represents **friends** (103).



plot/visualNetworkA.png

4.2. isConnected1.m

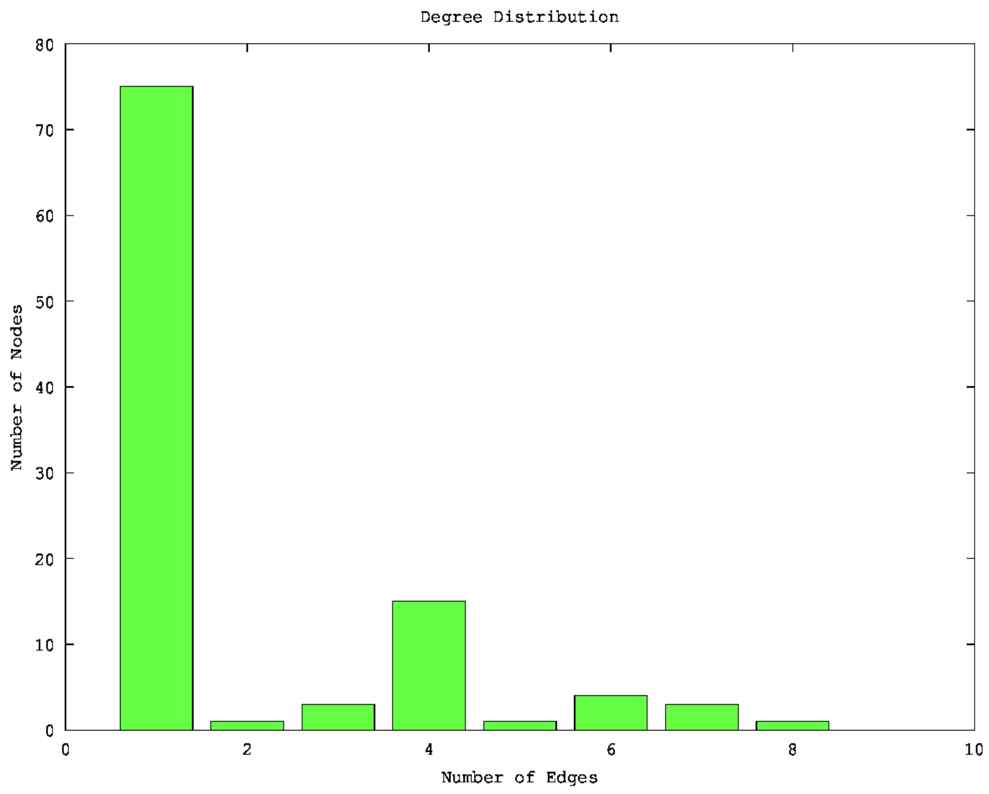
```
connected1 = 1
```

The octave function returns 1 applied on network B.
The algorithm works based to the algorithm as Task 3.2.
Check **symmetry** by evaluating the **diagonal matrix** which should contain only zeros.
Second step is to check if M(m,n) is equal to M(n,m) which represents the same edges navigating from actor row or event column. Is that not the case then the matrix is corrupted.
Connectivity is checked by using **Disjoint-set Data Structures**.
At first there should be an edge in the edges sets with and after every iteration over the whole network once can check if the current edge **intersects** with the edges sets. In that case the new edge is added to the data set by **union**. If the size entries in the data set equals the size of the network rows / columns the network is connected.

The pseudo code below represents nearly my developed algorithm:

```
set = [1,2] // initialize set by any edge
for i → m(1)
     for j → m(2)
     if m(1,2)
          if(intersect(m(1,2), set)
          set = union(set, m(1,2))
     if(size(nodes) == size(set))
          return 1;
```

4.3. degreeDistribution.m



Degree distribution is calculated by firstly summing up all vectors (rows) and saving them to an own degrees vector. Finally I calculated which degree appears how many times and saved them in a vector called histogram for plotting.

```
degree_vector =

 Columns 1 through 21:

  75   1   3  15   1   4   3   1   0   0   0   0   0   0   0   0   0   0   0   0   0

 Columns 22 through 42:

   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

 Columns 43 through 63:

   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

 Columns 64 through 84:

   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

 Columns 85 through 103:

   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

4.4. getNumEdges.m & smallWorld.m

```
num_edges =  102
L =  6.6587
C = 0
```

Number of edges → sum of the edges (ones) in the **triangular matrix**.

Small World?
The degree distribution is similar to a **long tail curve**, this means that there are a lot of nodes which only have one node.

Condition for Small World:

if ((L > Lrandom) & (C >> Crandom))

YES, my network is a small world network. The average path length is about 6.65 which means that the distance between two nodes is not so far (**six degrees of separation**). Due to the fact that Cluster Coefficient is 0 (zero) the probability of being a small world is therefore much more higher. There are no **caves**.