Social Network Analysis

William Cullian

Overview

This project will analyze Facebook social network data using a graph data structure. I will analyze sub communities. In the first part of the project I will analyze a user's egonet to determine the strength the user's friend group(egonet) by the average number of connections per user. I will also examine the larger egonets created by going one more hop out from the user, up to 3 hops out In the second part of the project I will attempt to find sub communities of friends in the data and attempt to identify the most influential users in the sub communities. I will use DFS to find any isolated sub communities, then use BFS to find the betweenness of edges and start eliminating the edges with the highest betweenness, until I have suitably small sub communities. Then I will print the stats.

Data

UCSD provided facebook data(File: facebook_UCSD.txt). This is an undirected graph containing 14947 facebook users and 886,442 friendships.
For testing I will use some smaller data sets that I created.

Easy Question:

Find the egonet of a user and rate its strength by number of connections divided by number of people. This will give me the average number of connections per user. I will also create larger egonets by going up to 3 hops out and compare the connectivity average as the egonet grows.

Hard Question:

Find sub communities in the data by removing edges that have maximum betweenness until I have a reasonable number of groups. Then examine the subgroups for users with maximum number of connections within the group. Then print stats for the groups including egonets with stats for the highest degree users.

Algorithms and Data Structures:

The network has been laid out as a classic graph using an adjacency list. Each individual in the graph is a vertex and an edge between vertices represents a friendship. Each vertex(user) has a list of edges to people he/she are friends with. The edges are directed, so a complete friendship consists of two edges, one from A to B and one from B to A. I will create an Egonet subclass to hold the egonets generated. It will have fields for the the center user number of the egonet as well as the depth and strength rating. For the hard question I will have to add fields to the UserNode as well as the FriendshipEdges, in order to implement brandes algorithm to find the centrality of betweenness.

For the easy question I will use a breadth first search to create an egonet for the user. I will modify BFS to use a queue of queues to separate the layers as we go out 1 hop then 2 hops, … etc. Then I will calculate the average number of connections per user. I will do this for 1, 2, and 3 hops out and then I will print stats showing the sizes, depth, density, connectivity average and degree distribution for each.

For the hard question, first, I will do Depth First searches starting from the first node and keeping a list of visited nodes. Then I will continue DFS from all unvisited nodes, keeping a list of the sub graphs created. This list will be the initial isolated communities in the data set. Then I will go thru communities larger than 300 users and try to separate them using breadth first search to find shortest paths from every node to every other node and use these to compute the betweenness of the edges and use the edges with highest betweenness to split communities into groups smaller than 300. Then I will attempt to identify the leaders in the group, by the highest count adjacency lists(degree) in the group.
Then I will print the group stats, for each sub community, including sizes, depth, density, connectivity average and degree distribution for each., as well as the egonets of the 3 highest degree leaders in each group.

<u>Algorithms</u>:

<u>Easy Question(egonets)</u>
Input: Specific User (u), depth to go to, I will use 3
Output: A printout showing the size and connectivity of the DEPTH egonets of the user u and also returns a list of DEPTH egonets, one for each hop out.

Main method:

```
evaluateEgonets(u, DEPTH)
    Create a return list of egonets of u (getEgonets(u, DEPTH))
    print the user u the egonet is for
    For each egonet x in the List:
        print the egonet number(number of hops), the average number of friends per person,
        the number of vertices, the number of edges, maximum possible friendships, density
        and degree distribution.

    return the return list
```

Egonet method(variation of BFS with queue of queues to keep track of depth and no parent map):

```
getEgonets(u, DEPTH)
    check if u exists in graph
    Initialize: queueOfQueues, queue, visited hashset, list of egonets to return, depth counter to 0
    Create queue and enque u into it end enque it into queueOfQueues
    add u to visited
    create empty egonet E with u as center
    while(there is still a queue in the queueOfQueues and depth counter < DEPTH)
        add one to depth counter
        deque queue currQueue from queueOfQueues
        create nextQueue to hold next hop out nodes
        while( currQueue is not empty)
            deque curr from currQueue
            for each of curr’s neighbors n not in visited set
                add n to visited
```

```
                    enque n into nextQueue
                    add n to egonet

               enque nextQueue to queueOfQueues
               fill in egonet's edges
               set egonet depth
               add copy of egonet to list of egonets
     return list of egonets
```

Answer:

```
Making new maps...DONE.
Loading the maps...DONE.
TEST MAP 1 SOLO ---------------------

PRINT EGONETS OF 10 TO DEPTH 3 ---------------

TEST MAP 2 SPARSE ---------------------

PRINT EGONETS OF 7 TO DEPTH 3 ---------------

Egonet for user: 7
Egonet depth: 1
Average number of friends per person: 0.75
Number of people in graph: 4
Number of friendships in graph: 3
Maximum possible number of friendships in graph: 6
Density of graph(num edges/ max num edges): 1.0
Degree Distribution:
Number of users with degree 1: 3
Number of users with degree 3: 1

Egonet for user: 7
Egonet depth: 2
Average number of friends per person: 1.1
Number of people in graph: 10
Number of friendships in graph: 11
Maximum possible number of friendships in graph: 45
Density of graph(num edges/ max num edges): 0.4888888888888889
Degree Distribution:
Number of users with degree 1: 2
Number of users with degree 2: 4
Number of users with degree 3: 4

Egonet for user: 7
Egonet depth: 3
```

```
Average number of friends per person: 1.2142857142857142
Number of people in graph: 14
Number of friendships in graph: 17
Maximum possible number of friendships in graph: 91
Density of graph(num edges/ max num edges): 0.37362637362637363
Degree Distribution:
Number of users with degree 2: 8
Number of users with degree 3: 6

TEST MAP 3 MEDIUM ---------------------

PRINT EGONETS OF 7 TO DEPTH 3 ---------------

Egonet for user: 7
Egonet depth: 1
Average number of friends per person: 1.0
Number of people in graph: 5
Number of friendships in graph: 5
Maximum possible number of friendships in graph: 10
Density of graph(num edges/ max num edges): 1.0
Degree Distribution:
Number of users with degree 1: 2
Number of users with degree 2: 2
Number of users with degree 4: 1

Egonet for user: 7
Egonet depth: 2
Average number of friends per person: 1.4545454545454546
Number of people in graph: 11
Number of friendships in graph: 16
Maximum possible number of friendships in graph: 55
Density of graph(num edges/ max num edges): 0.5818181818181818
Degree Distribution:
Number of users with degree 2: 4
Number of users with degree 3: 5
Number of users with degree 4: 1
Number of users with degree 5: 1

Egonet for user: 7
Egonet depth: 3
Average number of friends per person: 1.5714285714285714
Number of people in graph: 14
Number of friendships in graph: 22
Maximum possible number of friendships in graph: 91
Density of graph(num edges/ max num edges): 0.4835164835164835
Degree Distribution:
```

```
Number of users with degree 2: 3
Number of users with degree 3: 7
Number of users with degree 4: 3
Number of users with degree 5: 1

TEST MAP 4 FULL ---------------------

PRINT EGONETS OF 4 TO DEPTH 3 ---------------

Egonet for user: 4
Egonet depth: 1
Average number of friends per person: 2.0
Number of people in graph: 5
Number of friendships in graph: 10
Maximum possible number of friendships in graph: 10
Density of graph(num edges/ max num edges): 2.0
Degree Distribution:
Number of users with degree 4: 5

Egonet for user: 4
Egonet depth: 2
Average number of friends per person: 2.0
Number of people in graph: 5
Number of friendships in graph: 10
Maximum possible number of friendships in graph: 10
Density of graph(num edges/ max num edges): 2.0
Degree Distribution:
Number of users with degree 4: 5

FACEBOOK UCSD DATA---------------------

PRINT EGONETS OF 94 TO DEPTH 3 ---------------

Egonet for user: 94
Egonet depth: 1
Average number of friends per person: 5.75
Number of people in graph: 80
Number of friendships in graph: 460
Maximum possible number of friendships in graph: 3160
Density of graph(num edges/ max num edges): 0.2911392405063291
Degree Distribution:
Number of users with degree 1: 3
Number of users with degree 2: 2
Number of users with degree 3: 7
Number of users with degree 4: 5
Number of users with degree 5: 6
Number of users with degree 6: 8
Number of users with degree 7: 5
Number of users with degree 8: 3
Number of users with degree 9: 6
Number of users with degree 10: 1
Number of users with degree 11: 3
Number of users with degree 12: 3
Number of users with degree 13: 5
Number of users with degree 14: 5
Number of users with degree 15: 2
Number of users with degree 16: 2
Number of users with degree 17: 1
Number of users with degree 18: 2
Number of users with degree 22: 1
Number of users with degree 25: 3
Number of users with degree 26: 1
Number of users with degree 28: 3
Number of users with degree 32: 1
Number of users with degree 34: 1
Number of users with degree 79: 1

Egonet for user: 94
Egonet depth: 2
Average number of friends per person: 27.650602409638555
Number of people in graph: 3320
Number of friendships in graph: 91800
Maximum possible number of friendships in graph: 5509540
Density of graph(num edges/ max num edges): 0.03332401616105882
Degree Distribution:

Number of users with degree 1: 6
Number of users with degree 2: 12
Number of users with degree 3: 15
Number of users with degree 4: 12
```

To save space I have omitted the middle of this output

```
Number of users with degree 258: 1
Number of users with degree 260: 1
Number of users with degree 263: 1
Number of users with degree 328: 1
Number of users with degree 397: 1
Number of users with degree 672: 1

Egonet for user: 94
Egonet depth: 3
```

```
Average number of friends per person: 31.986387587822016
Number of people in graph: 13664
Number of friendships in graph: 437062
Maximum possible number of friendships in graph: 93345616
Density of graph(num edges/ max num edges): 0.009364381933051896
Degree Distribution:
Number of users with degree 1: 85
Number of users with degree 2: 117
Number of users with degree 3: 147
Number of users with degree 4: 126
```

To save space I have omitted the middle of this output

```
Number of users with degree 409: 1
Number of users with degree 413: 1
Number of users with degree 414: 1
Number of users with degree 421: 1
Number of users with degree 424: 1
Number of users with degree 427: 1
Number of users with degree 431: 1
Number of users with degree 435: 1
Number of users with degree 486: 1
Number of users with degree 488: 1
Number of users with degree 555: 1
Number of users with degree 567: 1
Number of users with degree 586: 1
Number of users with degree 801: 1
Number of users with degree 1262: 1
Number of users with degree 2165: 1
```

<u>Hard question(communities)</u>

Input: The graph of the community and the maximum community size for the sub-graphs.
Output: A printout showing the initial separation of the community and the stats for each group.
Then a printout showing the same for the graph after it has been broken into smaller communities.

Main method:

Input: a SocialNetworkGraph, maxSize
Output: Stats on Graph and on sub communities in Graph smaller than maxSize

```
analyzeCommunities(maxSize)
    print stats on initial Graph
    Create an initial list of sub-graphs of isolated communities IC, that exist within the community
        by calling findIsolatedCommunities() and print initial stats
    remove all communities larger than maxSize and place in stack STACK
    while STACK is not empty
        deque S from STACK
        While S is larger than maxSize
            Use breakStrongestLink() on S to remove the edge with highest betweenness
            call  findIsolatedCommunities() on S
            add any sub communities smaller than maxSize to IC
            push the sub communities larger than maxSize back on the STACK
    Print stats
```

Isolated community method:

Input: A Graph object
Output: List of sub-graphs of GRAPH

```
findIsolatedCommunities(GRAPH)
    Initialize : Create stack of all users STK, visited hashset VH, list of sub-graphs to return RET
        create empty sub-graph SG
    while there are still users in  STK
        pop user U from STK
        if U is already visited skip it
        else add U to visited VH
        create new sub-graph SG
        call DFSCapture(U, VH, SG)
        put SG from DFSCapture into RET
    return RET
```

Input: User, Visited list, Sub-graph
Output: visited list is updated and sub graph is completed

```
DFSCapture(U, VH, SG)
    add U to SG
    for each of U’s friends F not in VH
        add F to VH
        DFSCapture(F, VH, SG)
    add edges to SG
```

Betweenness method:

Input: A Graph object
Output: A Graph with its strongest betweenness link removed

```
breakStrongestLink(GRAPH)
    computeBetweenness(BSG)
    removeLargestFlow(BSG)

computeBetweenness(GRAPH)
    for all nodes S in GRAPH
        reset nodes
        BFS of GRAPH starting at S
        compute number of shortest paths from S to each other node
        trace paths back and distribute flow to the edges along the paths

removeLargestFlow(GRAPH)
    remove edge with largest flow
```

Answer:

Output from facebook_1000.txt, the full facebook graph was taking too long. Big O |VE|, C'Est la Vie.

```
Initial Graph:
Number of people in graph: 783
Number of friendships in graph: 1892
Maximum possible number of friendships in graph: 306153
Density of graph(num edges/ max num edges): 0.006179916577658883
Degree Distribution:
Number of users with degree 1: 130
Number of users with degree 2: 119
Number of users with degree 3: 121
Number of users with degree 4: 91
Number of users with degree 5: 69
Number of users with degree 6: 63
Number of users with degree 7: 39
Number of users with degree 8: 38
Number of users with degree 9: 23
Number of users with degree 10: 21
Number of users with degree 11: 14
Number of users with degree 12: 16
Number of users with degree 13: 6
Number of users with degree 14: 9
Number of users with degree 15: 5
Number of users with degree 16: 4
Number of users with degree 17: 6
Number of users with degree 18: 2
Number of users with degree 19: 2
Number of users with degree 20: 1
Number of users with degree 21: 2
Number of users with degree 22: 1
Number of users with degree 23: 1
Degree with most users: 130 users with degree 1

Initial communities:

Number of sub-communities in this graph: 1

Community #1
Number of people: 783
Number of friendships 1892
Edge with highest flow Edge [from=823, to=371]
Node with highest centrality 702

Egonets for 702 to depth 1

Egonet for user: 702
Egonet depth: 1
Average number of friends per person: 0.0
Number of people in graph: 2
Number of friendships in graph: 1
Maximum possible number of friendships in graph: 1
Density of graph(num edges/ max num edges): 1.0
Degree Distribution:
Number of users with degree 1: 2
Degree with most users: 2 users with degree 1

Node with highest degree 687

Egonets for 687 to depth 1

Egonet for user: 687
Egonet depth: 1
Average number of friends per person: 1.0416666666666667
```

```
Number of people in graph: 24
Number of friendships in graph: 51
Maximum possible number of friendships in graph: 276
Density of graph(num edges/ max num edges): 0.18478260869565216
Degree Distribution:
Number of users with degree 1: 4
Number of users with degree 2: 6
Number of users with degree 3: 4
Number of users with degree 4: 2
Number of users with degree 5: 4
Number of users with degree 6: 1
Number of users with degree 8: 1
Number of users with degree 9: 1
Number of users with degree 23: 1
Degree with most users: 6 users with degree 2


Final breakdown of communities smaller than 700:

Number of sub-communities in this graph: 12

Community #1
Number of people: 11
Number of friendships 14
Edge with highest flow Edge [from=915, to=641]
Node with highest centrality 297

Egonets for 297 to depth 1

Egonet for user: 297
Egonet depth: 1
Average number of friends per person: 0.3333333333333333
Number of people in graph: 3
Number of friendships in graph: 2
Maximum possible number of friendships in graph: 3
Density of graph(num edges/ max num edges): 0.6666666666666666
Degree Distribution:
Number of users with degree 1: 2
Number of users with degree 2: 1
Degree with most users: 2 users with degree 1

Node with highest degree 999

Egonets for 999 to depth 1

Egonet for user: 999
Egonet depth: 1
Average number of friends per person: 0.5714285714285714
Number of people in graph: 7
Number of friendships in graph: 9
Maximum possible number of friendships in graph: 21
Density of graph(num edges/ max num edges): 0.42857142857142855
Degree Distribution:
Number of users with degree 1: 2
Number of users with degree 2: 2
Number of users with degree 3: 2
Number of users with degree 6: 1
Degree with most users: 2 users with degree 1

Community #2
Number of people: 11
Number of friendships 13
Edge with highest flow Edge [from=631, to=998]
Node with highest centrality 200

Egonets for 200 to depth 1

Egonet for user: 200
Egonet depth: 1
Average number of friends per person: 0.0
Number of people in graph: 2
Number of friendships in graph: 1
Maximum possible number of friendships in graph: 1
Density of graph(num edges/ max num edges): 1.0
Degree Distribution:
Number of users with degree 1: 2
Degree with most users: 2 users with degree 1

Node with highest degree 998

Egonets for 998 to depth 1

Egonet for user: 998
Egonet depth: 1
Average number of friends per person: 0.5714285714285714
Number of people in graph: 7
Number of friendships in graph: 8
Maximum possible number of friendships in graph: 21
Density of graph(num edges/ max num edges): 0.38095238095238093
Degree Distribution:
Number of users with degree 1: 3
Number of users with degree 2: 2
Number of users with degree 3: 1
Number of users with degree 6: 1
Degree with most users: 3 users with degree 1

Community #3
Number of people: 10
Number of friendships 15
Edge with highest flow Edge [from=878, to=45]
Node with highest centrality 419

Egonets for 419 to depth 1

Egonet for user: 419
Egonet depth: 1
Average number of friends per person: 0.0
Number of people in graph: 2
Number of friendships in graph: 1
Maximum possible number of friendships in graph: 1
Density of graph(num edges/ max num edges): 1.0
Degree Distribution:
Number of users with degree 1: 2
Degree with most users: 2 users with degree 1

Node with highest degree 878

Egonets for 878 to depth 1

Egonet for user: 878
Egonet depth: 1
Average number of friends per person: 0.8333333333333334
Number of people in graph: 6
Number of friendships in graph: 11
Maximum possible number of friendships in graph: 15
Density of graph(num edges/ max num edges): 0.7333333333333333
Degree Distribution:
Number of users with degree 3: 3
Number of users with degree 4: 2
Number of users with degree 5: 1
Degree with most users: 3 users with degree 3

Community #4
Number of people: 6
Number of friendships 9
Edge with highest flow Edge [from=312, to=669]
Node with highest centrality 669
```

```
Egonets for 669 to depth 1

Egonet for user: 669
Egonet depth: 1
Average number of friends per person: 0.6
Number of people in graph: 5
Number of friendships in graph: 7
Maximum possible number of friendships in graph: 10
Density of graph(num edges/ max num edges): 0.7
Degree Distribution:
Number of users with degree 2: 2
Number of users with degree 3: 2
Number of users with degree 4: 1
Degree with most users: 2 users with degree 2

Node with highest degree 996

Egonets for 996 to depth 1

Egonet for user: 996
Egonet depth: 1
Average number of friends per person: 0.6
Number of people in graph: 5
Number of friendships in graph: 7
Maximum possible number of friendships in graph: 10
Density of graph(num edges/ max num edges): 0.7
Degree Distribution:
Number of users with degree 2: 2
Number of users with degree 3: 2
Number of users with degree 4: 1
Degree with most users: 2 users with degree 2

Community #5
Number of people: 7
Number of friendships 8
Edge with highest flow Edge [from=995, to=266]
Node with highest centrality 995

Egonets for 995 to depth 1

Egonet for user: 995
Egonet depth: 1
Average number of friends per person: 0.4
Number of people in graph: 5
Number of friendships in graph: 5
Maximum possible number of friendships in graph: 10
```

```
Density of graph(num edges/ max num edges): 0.5
Degree Distribution:
Number of users with degree 1: 2
Number of users with degree 2: 2
Number of users with degree 4: 1
Degree with most users: 2 users with degree 1

Node with highest degree 995

Egonets for 995 to depth 1

Egonet for user: 995
Egonet depth: 1
Average number of friends per person: 0.4
Number of people in graph: 5
Number of friendships in graph: 5
Maximum possible number of friendships in graph: 10
Density of graph(num edges/ max num edges): 0.5
Degree Distribution:
Number of users with degree 1: 2
Number of users with degree 2: 2
Number of users with degree 4: 1
Degree with most users: 2 users with degree 1

Community #6
Number of people: 5
Number of friendships 4
Edge with highest flow Edge [from=236, to=994]
Node with highest centrality 994

Egonets for 994 to depth 1

Egonet for user: 994
Egonet depth: 1
Average number of friends per person: 0.25
Number of people in graph: 4
Number of friendships in graph: 3
Maximum possible number of friendships in graph: 6
Density of graph(num edges/ max num edges): 0.5
Degree Distribution:
Number of users with degree 1: 3
Number of users with degree 3: 1
Degree with most users: 3 users with degree 1

Node with highest degree 994
```

```
Egonets for 994 to depth 1

Egonet for user: 994
Egonet depth: 1
Average number of friends per person: 0.25
Number of people in graph: 4
Number of friendships in graph: 3
Maximum possible number of friendships in graph: 6
Density of graph(num edges/ max num edges): 0.5
Degree Distribution:
Number of users with degree 1: 3
Number of users with degree 3: 1
Degree with most users: 3 users with degree 1

Community #7
Number of people: 4
Number of friendships 3
Edge with highest flow Edge [from=993, to=391]
Node with highest centrality 993

Egonets for 993 to depth 1

Egonet for user: 993
Egonet depth: 1
Average number of friends per person: 0.3333333333333333
Number of people in graph: 3
Number of friendships in graph: 2
Maximum possible number of friendships in graph: 3
Density of graph(num edges/ max num edges): 0.6666666666666666
Degree Distribution:
Number of users with degree 1: 2
Number of users with degree 2: 1
Degree with most users: 2 users with degree 1

Node with highest degree 993

Egonets for 993 to depth 1

Egonet for user: 993
Egonet depth: 1
Average number of friends per person: 0.3333333333333333
Number of people in graph: 3
Number of friendships in graph: 2
Maximum possible number of friendships in graph: 3
Density of graph(num edges/ max num edges): 0.6666666666666666
Degree Distribution:
```

```
Number of users with degree 1: 2
Number of users with degree 2: 1
Degree with most users: 2 users with degree 1

Community #8
Number of people: 15
Number of friendships 16
Edge with highest flow Edge [from=153, to=992]
Node with highest centrality 777

Egonets for 777 to depth 1

Egonet for user: 777
Egonet depth: 1
Average number of friends per person: 0.25
Number of people in graph: 4
Number of friendships in graph: 3
Maximum possible number of friendships in graph: 6
Density of graph(num edges/ max num edges): 0.5
Degree Distribution:
Number of users with degree 1: 3
Number of users with degree 3: 1
Degree with most users: 3 users with degree 1

Node with highest degree 793

Egonets for 793 to depth 1

Egonet for user: 793
Egonet depth: 1
Average number of friends per person: 0.42857142857142855
Number of people in graph: 7
Number of friendships in graph: 6
Maximum possible number of friendships in graph: 21
Density of graph(num edges/ max num edges): 0.2857142857142857
Degree Distribution:
Number of users with degree 1: 6
Number of users with degree 6: 1
Degree with most users: 6 users with degree 1

Community #9
Number of people: 6
Number of friendships 6
Edge with highest flow Edge [from=989, to=508]
Node with highest centrality 360
```

```
Egonets for 360 to depth 1

Egonet for user: 360
Egonet depth: 1
Average number of friends per person: 0.3333333333333333
Number of people in graph: 3
Number of friendships in graph: 2
Maximum possible number of friendships in graph: 3
Density of graph(num edges/ max num edges): 0.6666666666666666
Degree Distribution:
Number of users with degree 1: 2
Number of users with degree 2: 1
Degree with most users: 2 users with degree 1

Node with highest degree 989

Egonets for 989 to depth 1

Egonet for user: 989
Egonet depth: 1
Average number of friends per person: 0.4
Number of people in graph: 5
Number of friendships in graph: 5
Maximum possible number of friendships in graph: 10
Density of graph(num edges/ max num edges): 0.5
Degree Distribution:
Number of users with degree 1: 2
Number of users with degree 2: 2
Number of users with degree 4: 1
Degree with most users: 2 users with degree 1

Community #10
Number of people: 6
Number of friendships 7
Edge with highest flow Edge [from=91, to=555]
Node with highest centrality 230

Egonets for 230 to depth 1

Egonet for user: 230
Egonet depth: 1
Average number of friends per person: 0.3333333333333333
Number of people in graph: 3
Number of friendships in graph: 3
Maximum possible number of friendships in graph: 3
Density of graph(num edges/ max num edges): 1.0
```

```
Degree Distribution:
Number of users with degree 2: 3
Degree with most users: 3 users with degree 2

Node with highest degree 91

Egonets for 91 to depth 1

Egonet for user: 91
Egonet depth: 1
Average number of friends per person: 0.6
Number of people in graph: 5
Number of friendships in graph: 6
Maximum possible number of friendships in graph: 10
Density of graph(num edges/ max num edges): 0.6
Degree Distribution:
Number of users with degree 1: 1
Number of users with degree 2: 2
Number of users with degree 3: 1
Number of users with degree 4: 1
Degree with most users: 2 users with degree 2

Community #11
Number of people: 8
Number of friendships 11
Edge with highest flow Edge [from=987, to=646]
Node with highest centrality 646

Egonets for 646 to depth 1

Egonet for user: 646
Egonet depth: 1
Average number of friends per person: 0.6
Number of people in graph: 5
Number of friendships in graph: 6
Maximum possible number of friendships in graph: 10
Density of graph(num edges/ max num edges): 0.6
Degree Distribution:
Number of users with degree 1: 1
Number of users with degree 2: 2
Number of users with degree 3: 1
Number of users with degree 4: 1
Degree with most users: 2 users with degree 2

Node with highest degree 987
```

```
Egonets for 987 to depth 1

Egonet for user: 987
Egonet depth: 1
Average number of friends per person: 0.6666666666666666
Number of people in graph: 6
Number of friendships in graph: 9
Maximum possible number of friendships in graph: 15
Density of graph(num edges/ max num edges): 0.6
Degree Distribution:
Number of users with degree 2: 2
Number of users with degree 3: 3
Number of users with degree 5: 1
Degree with most users: 3 users with degree 3

Community #12
Number of people: 694
Number of friendships 1535
Edge with highest flow Edge [from=167, to=170]
Node with highest centrality 167

Egonets for 167 to depth 1

Egonet for user: 167
Egonet depth: 1
Average number of friends per person: 0.375
Number of people in graph: 8
Number of friendships in graph: 7
Maximum possible number of friendships in graph: 28
Density of graph(num edges/ max num edges): 0.25
Degree Distribution:
Number of users with degree 1: 7
Number of users with degree 7: 1
Degree with most users: 7 users with degree 1

Node with highest degree 687

Egonets for 687 to depth 1

Egonet for user: 687
Egonet depth: 1
Average number of friends per person: 1.0
Number of people in graph: 22
Number of friendships in graph: 45
Maximum possible number of friendships in graph: 231
Density of graph(num edges/ max num edges): 0.19480519480519481
```

Degree Distribution:
Number of users with degree 1: 4
Number of users with degree 2: 6
Number of users with degree 3: 4
Number of users with degree 4: 1
Number of users with degree 5: 3
Number of users with degree 6: 1
Number of users with degree 8: 2
Number of users with degree 21: 1
Degree with most users: 6 users with degree 2

Algorithm Analysis, Limitations, Risk:
Easy Question:
Let V be the number of friends of a given user (this could be as large as number of vertices - 1 in the graph, then there would be only one egonet).
The main method just loops thru DEPTH egonets and prints stats on them, that is O(1).
The egonet method loops thru each queue in a queue of queues, all the queues together will never have more that V elements, but I will also be looping thru E edges which makes it O(|V| + |E|).
Adding a egonet to the return list is also O(1) because I use a LinkedList for my return list.

Hard Question:
First I will have to check for disjointed sets to find communities that are already separated. I can use a variation of DFS similar to Strongly Connected Components algorithm for this. I don't think I will need to do more than one search per connected group to accomplish this, which eliminates the transpose and the second search. That will be O of |V| + |E|.
Then, I am attempting to use brandes algorithm to find the betweenness of the edges to aid in separating communities. This is Big O of |VE|. Unfortunately this will take a while to run on my large data set.
Every time I eliminate an edge I will need to check the community for disjointed sets(separate sub communities) again. This is V + E again, the V's and E's of the sub communities will add up to the whole at worst.
This is all linear time big O of V + E, except the betweenness algorithm, which is VE. So, overall it is running in big O of VE. The data set has 14947 facebook users and 886,442 friendships, so it may take a while to run, but I think it will work.
The hard question takes some time to run, but my computer is old and slow. I think it could still be optimized to run faster, but I am out of time.

Correctness verification (i.e. testing):
I created four small data sets. The first was a small network I'd used to develop the algorithm. The second was still small but very sparse, to see if all edges got caught. The third was small but completely connected to test for only one egonet. The fourth was just a single vertex to test a corner case. I calculated the stats on paper and my first test didn't come out right, forcing me to make some corrections in the counting mechanism for adding edges. The single case also forced me to make a a

correction. After that the stats added up and all cases worked. I tried it on the large data set and it didn't crash. Viola!
For the hard question I added two new testmaps, one with isolated groups, one with isolated individuals. This is to test to make sure I find all isolated groups and people. I also made a small one and calculated edge flow and betweenness centrality by hand to verify my results.

Reflection:
After starting I soon realized that I forgot a few things in my algorithm. I reexamined some test cases on paper and rewrote my algorithm to solve problems with the queue of queues. I also realized that I could have made it easier and more efficient by eliminating the edge class and just using a hashset for the edges. I decided to leave it the way it was in case I need the edge class on the harder question. I did optimize by making the linked list in UserNode into a hashset. I also added compareTo methods to make comparisons easier. I ended up using the edge class on the hard question to store shortest path pairs and in a hashmap for edge flow values that can be looked up by edge.
I also forgot that I could extend the SocialNetworkGraph class for the Egonet class. I created seperate classes. I changed that and now Egonet extends SocialNetworkGraph. I then did the same thing for the hard question and created the CentralityGraph class.
I originally was generating a inter connectivity rating for the egonets by dividing the number of edges by the maximum number of edges. After further reflection I decided this was not good since the denominator was increasing exponentially as the number of people increased. I decided to create an inner connectivity average. This is the average number of connections per user in the egonet. Later I found out that my so called inter connectivity rating was actually called graph density and decided to include it in my stats.
After studying brandes algorithm I realized that by adding fields to the UserNode I could tell how many hops out I was just by the distance of the node from the start, this may have been better than the queue of queues I used in the egonet, but I put too much work in on those to change them now.
After all the web surfing for brandes algorithm I also learned about some new stats to print for my graphs. Graph Density and Graph Degree Distribution. Both good representations of how connected a graph is.
I skipped ahead to Leo Porter's video for data presentation and got the idea for creating a data class to hold the data for the algorithms. I ended up creating a lot of different data classes and sub-classes for graphs, nodes and edges. I ended up with more than 20 and got very confused with all the polymorphism and inheritance. I didn't really like it as well, it made it harder to see what fields the class had and adding classes making things more complicated. I consolidated them and have only 7 classes and 2 interfaces now.
When printing the hashmap for the degree distribution, it was not coming out in order, thats when I discovered treemap, a sorted hashmap. It sacrifices some speed so it would print out in order.
I just did things the way I did, because I never built anything from scratch before. There are still things that can be optimized, but I just wanted something that worked first. I would like to make it more robust by adding exception handling and would like to get some practice making junit tests for it.