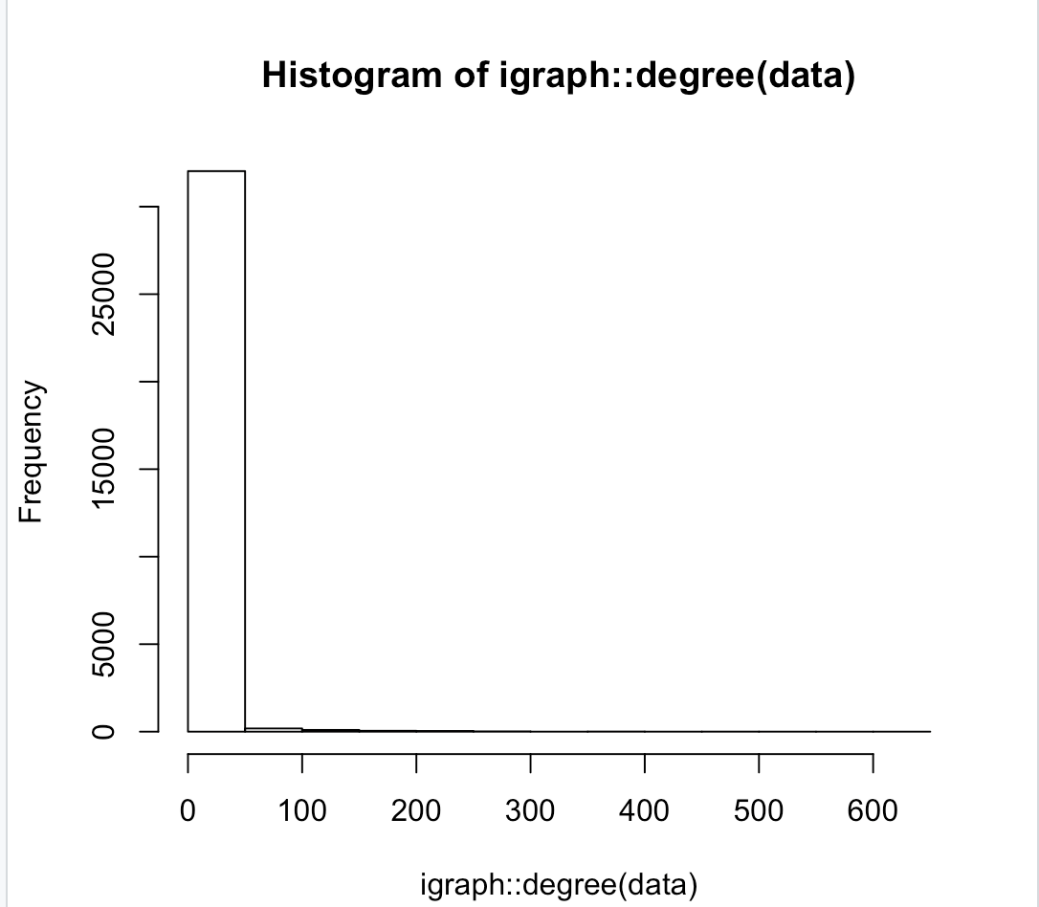
**CS3907: Big Data and Analytics - Project #1 Write Up**

As a team, we began the project by first creating a repository on Github and uploading the data from blackboard. We placed all these files (the data, our own R file, that interacts with the data, and write-up file) into our shared repository. Our next step was to read the graph through the R function “read.graph” and begin our analysis. We then looked at 10 different R functions in order to better grasp what the data was showing us. From looking at the graphs connectivity, its diameter, and neighboring nodes, we were able to figure out different characteristics of the data. Finally, through our research, we were then able to determine who the central person(s) is, the longest path, the largest clique, ego, and power centrality.

**# 3**

As far as simplification goes, our group felt that just using the simplify command was enough.To make it easier to use, we saved the simplified graph as a Rdata file. This allowed us to interact with the graph by referring to data.Rdata and made subsequent analysis faster and more straightforward. As per number 3 of the project specification, we explored some of the methods that were demoed in the lecture slides. This included methods like centr\_betw, edge\_density, shortest\_paths, and plotting a histogram of the degrees. These methods provided us with insight into the data, and we also discovered a potential error in the lecture slides: when using shortest\_paths like the slides shows there is an error, the definition of the method may have changed. Here is an image of the histogram generated of node degrees

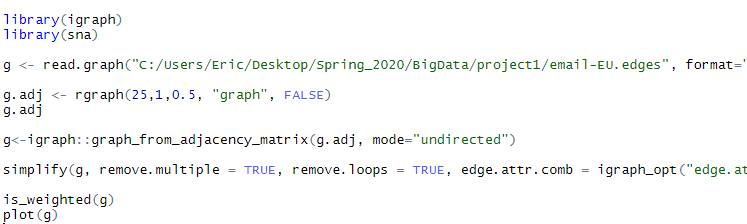


Below is the set of 10 functions we utilized in order to figure out how each of the 25 inputs is connected and how they interacted with one another.

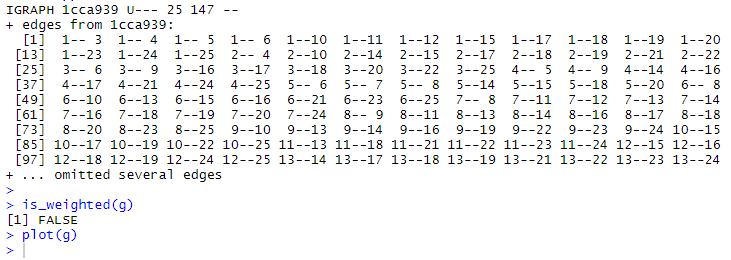
**Function 1:** is\_weighted

This function allows users to determine if an inputted graph, either directed or undirected, has an edge attribute, which identifies if it is weighted or not. We first simplified the graph through the simplify function. This function removes multiples and loops. The outputted results are pictured below, showing that our graph was not weighted.

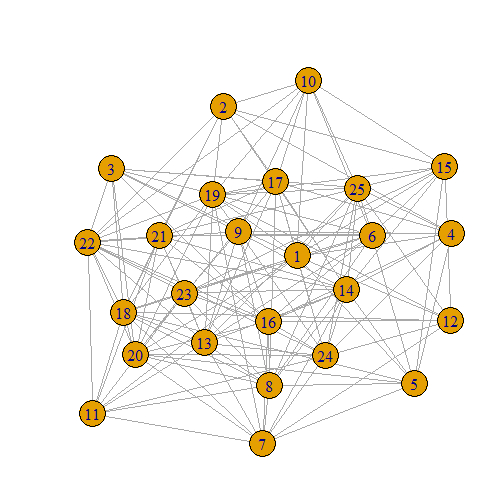
Code:



Result:



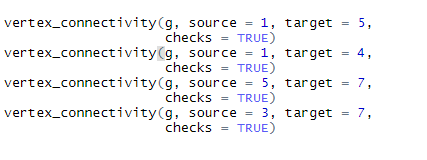
Graph:



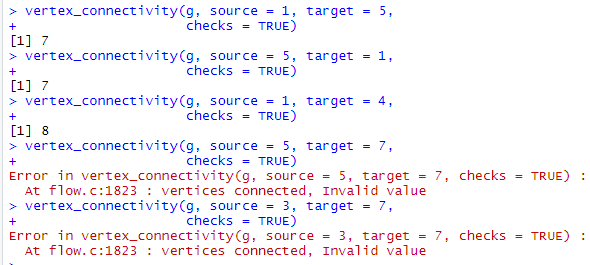
**Function 2:** vertex\_connectivity

This function analyzes the connection between a source and target node in a directed graph, to identify how many nodes need to be removed in order for there to be no direct paths between the source and target node. We tested this function with several nodes. The output showed that the connectivity of nodes one and five is seven. This means that seven nodes would need to be removed in order for there to be no direct paths from one to five.

Code:



Result:



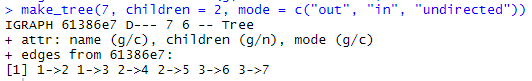
**Function 3:** make\_tree

Given a number of vertexes, this function will produce a tree and print relevant associations.

Code:



Result:



**Function 4:** Isomorphic

This function is also self-explanatory. It determines whether two groups correspond to each other in a manner that is one-to-one (whether they are isomorphic). The output was True.

**Code:**



**Result:**



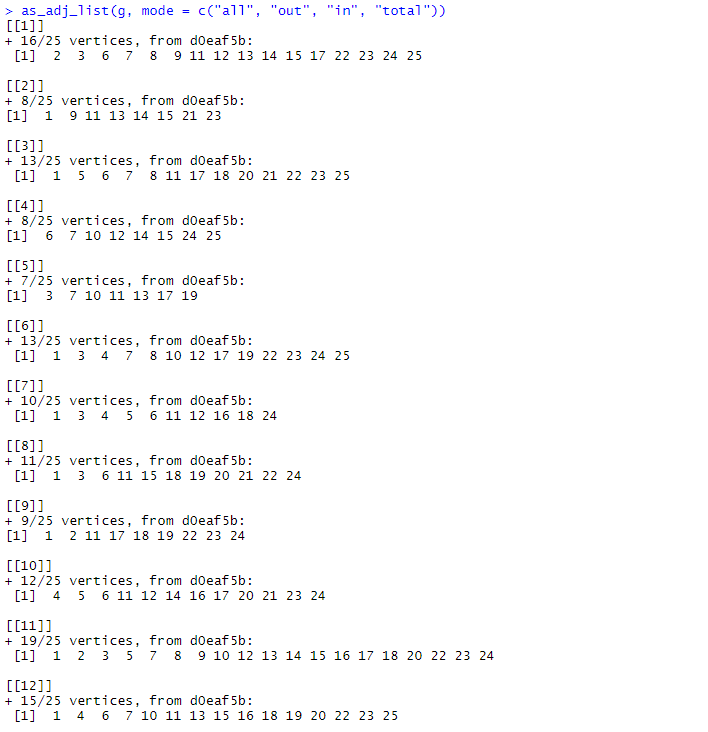
**Function 5:** as\_adj\_list

This function creates an adjacency list from the inputted graph. The result returns all of the neighbors of each node in the graph. This accounts for outgoing edges and incoming edges since this is a directed graph.

Code:



Result:



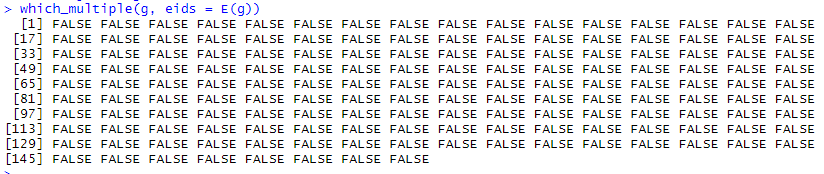
**Function 6:** which\_multiple

This function returns if there are any duplicate connections within the graph. The result was false for every connection, meaning that there were no connections, connected more than once.

Code:



Result:



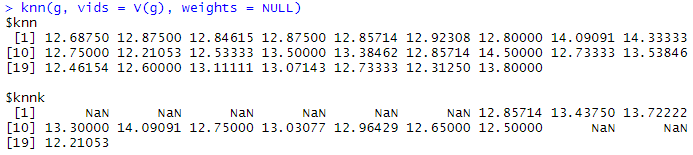
**Function 7:** knn

The function, knn, returns the average nearest neighbor degree.

Code:



Result:



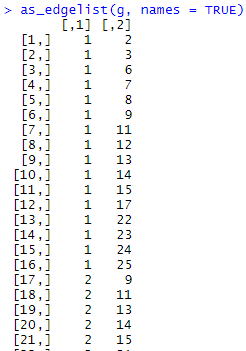
**Function 8:** as\_edgelist

This function outputs the neighbors of each node in a list format. This is similar to function five.

Code:



Result:



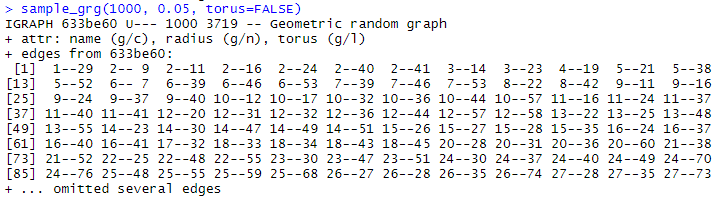
**Function 9:** sample\_grg

This function generates a random graph using points of a unit square.

Code:



Result:



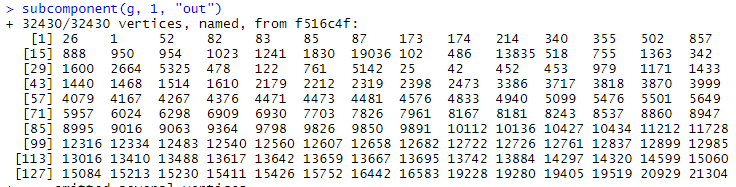
**Function 10:** subcomponent

This function takes in a vertex and returns an adjacency list of all vertexes that are reachable from that point. It can also be reversed. This is less flashy for a connected graph.

Code:



Result:

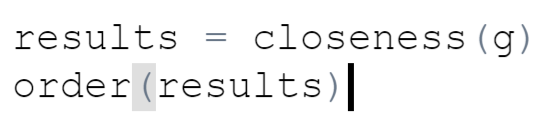


For getting the answers to who the central person(s) is, the longest path, the largest clique, ego, and power centrality, we looked to five different R functions that will be described below, with an explanation, and answer. These functions were included in the packages that were installed.

**Central Person(s):**

In order to find the central persons, we used the closeness centrality measure as provided by closeness() by igraph. This measure is based on the number of steps required to access every other node. Our results show node 1 as the most central node.

Code:



Result:

[1] 1 451 2025 2940 12316 14253 17764 18618 22535

[10] 23080 26224 29321 13511 20107 21268 21618 23555 4948

[19] 4952 5601 8074 8995 14794 19390 19815 4347 25129

[28] 5742 5752 5809 27207 8428 16732 9476 3870 6276

[37] 7529 8593 9179 15361 15954 16876 18566 18897 20745

[46] 22578 6571 6992 12253 28441 4232 4661 6152 11739

[55] 14006 3401 6285 9570 12169 12441 17148 17190 17460

[64] 17731 22059 22354 22565 22675 23248 1436 1812 5286

[73] 16064 18164 26188 10637 24227 778 864 872 1251

[82] 1671 3246 3952 4148 4260 4708 5506 6608 7822

[91] 8431 11968 11969 11971 12003 12033 12046 12061 12067

[100] 12181 12531 13026 14520 14546 16207 20590 22743 22865

[109] 6746 6751 6763 6784 6795 1542 3981 10332 11216

[118] 15958 23619 28669 31507 17275 10801 10935 11832 23117

[127] 31702 127 15122 16657 17164 17247 17503 17603 17733

[136] 13335 13634 13953 23456 25486 21817 22254 2524 113

[145] 1558 3494 3541 4351 4432 4884 5451 6284 6457

[154] 7144 7420 7535 8488 9679 9830 10079 10940 11046

[163] 11462 11756 11937 13438 13504 14009 14010 14451 14725

[172] 14850 15205 16407 17690 17692 17863 18911 20040 20382

[181] 22662 22741 23025 292 2739 92 1747 1872 2547

[190] 2626 2794 3335 3761 3855 4569 4861 5271 7134

[199] 7526 7667 14688 3390 4305 4960 9192 13672 14013

[208] 15732 16943 18968 19532 19779 21920 23893 25196 26265

[217] 27258 27925 29519 10573 12478 14460 15963 17979 21352

[226] 22229 2891 5843 5844 6019 6825 6978 10962 10963

[235] 14936 18067 18920 19324 19893 24849 27691 31528 31923

[244] 20962 24211 1730 3854 12487 15780 17480 17985 18850

[253] 20875 24262 27264 16734 26557 20142 11459 19937 23024

[262] 24671 24166 16287 22734 7235 6546 14287 20570 20851

[271] 21396 437 6381 7868 9921 10928 11224 11797 11986

[280] 14206 14702 17819 17907 22331 24379 193 14134 18066

[289] 17104 32059 32161 6194 6344 8955 9390 11206 15328

[298] 15334 15799 18977 20020 21358 1744 8856 696 2914

[307] 3382 4215 10349 11475 11710 15736 15873 16242 16730

[316] 20714 20784 26147 26422 30681 31248 31976 2816 14926

[325] 19664 23730 31260 4562 4563 4564 11669 6583 28699

[334] 1649 4375 2874 2924 3822 6739 6897 7771 9060

[343] 9640 9741 10404 11953 12801 14699 15633 17997 19855

[352] 20941 22337 22805 22807 23640 23651 24874 25443 25532

[361] 26279 27721 27890 28206 28727 29446 30213 30300 30779

[370] 30947 31100 31325 31793 8508 62 4348 9399 9497

[379] 9042 25545 25882 26349 26705 26804 26814 27187 29437

[388] 30301 32242 7148 2910 7512 26696 7525 8460 26900

[397] 26985 27043 30172 30230 5706 2852 6022 6564 8479

[406] 8664 8714 8768 13984 14197 19756 20165 20288 24631

[415] 27398 27400 28027 28676 28870 30205 30824 31901 31943

[424] 31953 31956 32221 32258 1583 14096 16306 6987 2470

[433] 8134 12847 15249 16139 20754 21656 23185 23458 23665

[442] 25522 26629 28069 28353 28867 29763 20353 20354 20602

[451] 21069 1932 8800 15299 16138 17357 25014 28551 29801

[460] 17209 580 655 2513 2567 4399 13070 15789 22221

[469] 23600 28596 30902 7070 8088 14302 14959 14963 15259

[478] 16422 17155 17456 18297 18608 20708 21328 21937 21983

[487] 21990 22061 25210 27832 27930 27937 27938 29170 29279

[496] 29294 31028 31054 12639 7983 10436 15514 17040 18461

[505] 19186 19928 21095 22344 763 1774 3913 7470 7959

[514] 8379 11997 16680 19916 19931 20255 20612 22026 22103

**Longest Path:**

The function used to extract the longest path from the graph by using the path that has the largest diameter. The result outputted is 9, highlighting that the longest path goes through 9 nodes.

Code:



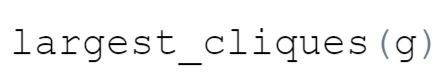
Result:



**Largest Clique:**

A clique is a completely connected subgraph. In this, every node is connected to every other node. These cliques have a strong connection, as every node is connected. The results show that there are 24 cliques all of size 12.

Code:



Result:

[[1]]

+ 12/32431 vertices, from 9387f47:

[1] 951 215 175 84 83 88 356 503 454 280 96 889

[[2]]

+ 12/32431 vertices, from 9387f47:

[1] 951 215 175 84 83 88 356 503 454 280 96 104

[[3]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 6713 326 30 503 84 224 1024 12888

[11] 174 682

[[4]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 224 326 84 503 30 1024 682 174

[11] 1023 12888

[[5]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 224 326 84 503 30 1024 83 1023

[11] 12888 8014

[[6]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 224 326 84 503 30 1024 83 1023

[11] 12888 174

[[7]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 215 560 682 12888 30 174 326 503

[11] 1023 1024

[[8]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 215 84 503 682 174 326 1023 1024

[11] 30 12888

[[9]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 215 84 503 83 326 1024 1023 30

[11] 12888 8014

[[10]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 215 84 503 83 326 1024 1023 30

[11] 12888 174

[[11]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 215 84 503 83 88 1024 1023 8014

[11] 30 12888

[[12]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 215 84 503 83 88 1024 1023 174 356 280

[[13]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 215 84 503 83 88 1024 1023 174

[11] 30 12888

[[14]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 215 84 503 83 88 1024 1023 104 280 356

[[15]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 215 84 503 83 88 1024 96 104 280 356

[[16]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 215 84 503 83 88 454 356 280 104 96

[[17]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 215 84 503 83 88 454 356 280 53 174

[[18]]

+ 12/32431 vertices, from 9387f47:

[1] 103 175 215 84 503 83 88 454 356 27 53 174

[[19]]

+ 12/32431 vertices, from 9387f47:

[1] 2 175 84 83 224 326 503 30 1024 1023

[11] 12888 8014

[[20]]

+ 12/32431 vertices, from 9387f47:

[1] 2 175 84 83 224 326 503 30 1024 1023

[11] 12888 174

[[21]]

+ 12/32431 vertices, from 9387f47:

[1] 2 175 84 83 215 503 326 30 1024 1023

[11] 12888 8014

[[22]]

+ 12/32431 vertices, from 9387f47:

[1] 2 175 84 83 215 503 326 30 1024 1023

[11] 12888 174

[[23]]

+ 12/32431 vertices, from 9387f47:

[1] 2 175 84 83 215 503 88 1024 30 1023

[11] 12888 8014

[[24]]

+ 12/32431 vertices, from 9387f47:

[1] 2 175 84 83 215 503 88 1024 30 1023

[11] 12888 174

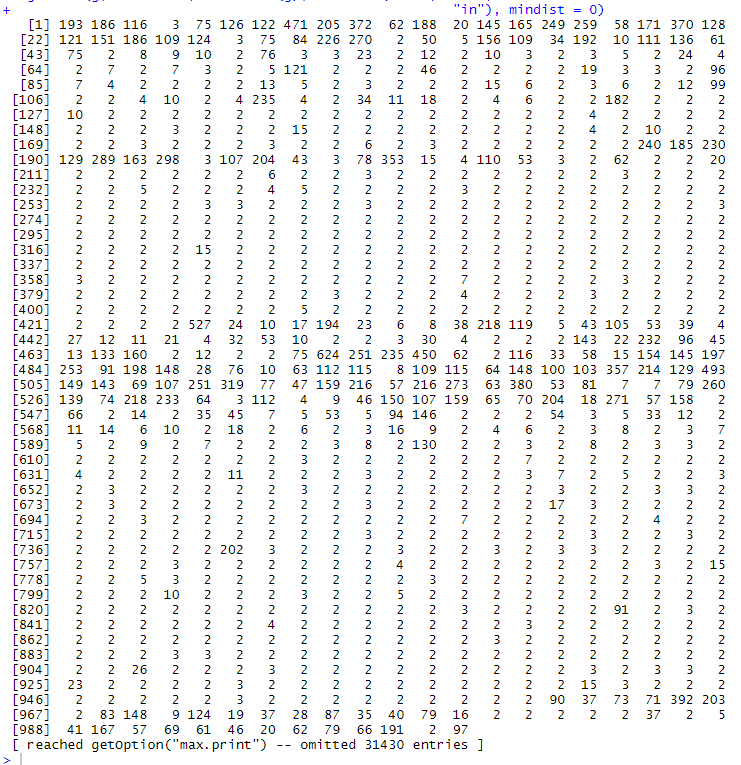
**Ego:**

Ego focuses upon seeing which nodes are directly and indirectly connected to one another. For example, if a node has a direct link to one node or if a node goes through another node to get to the target node. Below is the result of all the connections of nodes to one another.

Code:



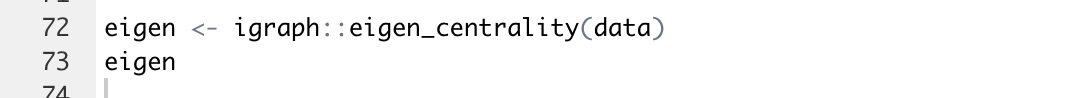
Result:



**Power Centrality:**

For power centrality we decided to use eigenvector centrality. According to the igraph documentation, this process essentially assigns each node a score based on the first eigenvector of the graph adjacency matrix. Nodes with high scores are highly connected to other nodes with lots of connections. The return of the igraph function gives a vector containing the scores, as well as the eigenvalue corresponding to the vector. Below is a screenshot of the begging of the scores, as well as the value. If you sort the scores, interestingly there is a node with a value of 1.

Code:



Result:

