Social Network Analysis Project - Round 1

Team Name - Mr. Robot

Team Members :

1. **Ansh Mehta (17ucs027)**
2. **Shubham Pabuwal (17ucs157)**
3. **Utkarsh Khandelwal (17ucc062)**
4. **Vaibhav Sharma (17ucs174)**

Index

Importing Required Libraries

In [1]:
```
import csv
import networkx as nx
import matplotlib.pyplot as plt
from itertools import combinations
import math
```

1. **CSV - For extracting the dataset**
2. **NetworkX -**
3. **Matplot -**
4. **Math functions**

Creating a Directed Graph for Dataset 1

About the Dataset

Title : **US Airports**
Category : Infrastructure
About : This is the **directed network** of flights between US airports in 2010. Each edge represents a connection from one airport to another, and the weight of an edge shows the number of flights on that connection in the given direction, in 2010.

In [174]:
```
flight = []
with open("out.tsv") as tsvfile:
    tsvreader = csv.reader(tsvfile, delimiter=' ')
    i = 0
    for line in tsvreader:
        if i > 1:
            flight.append((int(line[0]),int(line[1]),int(line[2])))
        i = i+1
```

In [175]:
```
G = nx.DiGraph()
```

In [176]:
```
G.add_weighted_edges_from(flight)
```

In [177]:
```
print(nx.info(G))
```

```
Name:
Type: DiGraph
Number of nodes: 1574
Number of edges: 28236
Average in degree:  17.9390
Average out degree:  17.9390
```

The graph contains 1574 Airports with 28236 connections

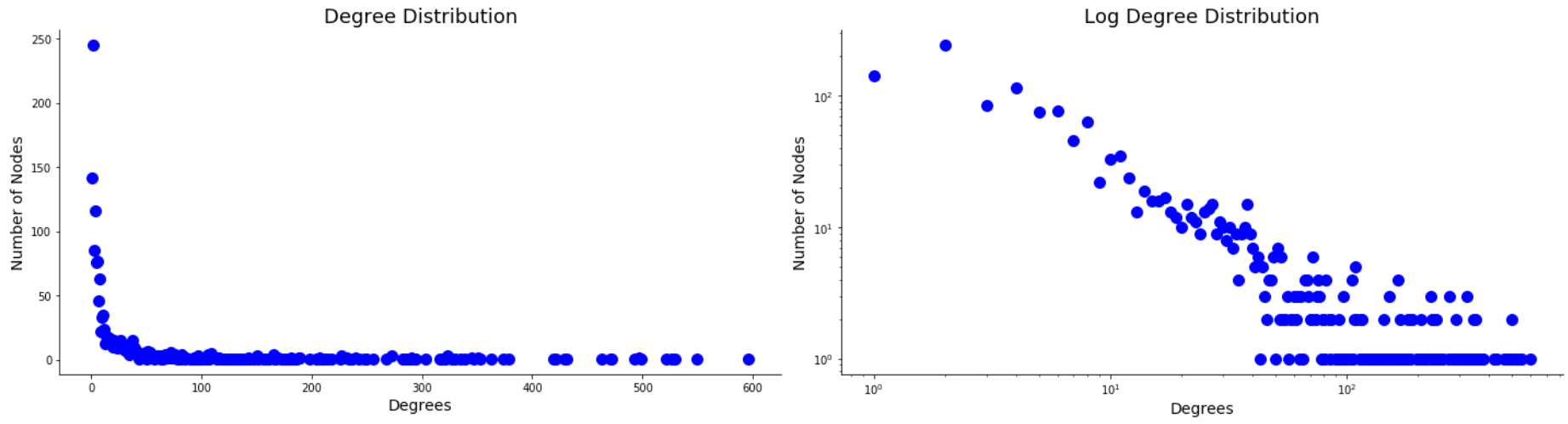
Problem 1 - Dataset - 1

Degree Distribution

In the study of graphs and networks, the degree of a node in a network is the number of connections it has to other nodes and the degree distribution is the probability distribution of these degrees over the whole network.
As we can see form the Graph there are many nodes with very less degree and with some nodes having high degree which is a good example of real type network.

In [178]:
```
d = dict()
for node,degree in nx.degree(G):
    d[degree] = d.get(degree,0)+1
```

In [179]:
```
plt.figure(figsize=(20,10))
plt.subplot(2, 2, 1)
plt.plot(list(d.keys()), list(d.values()),'bo-', linewidth=0, markersize=10)
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

plt.xlabel("Degrees", fontsize=14)
plt.ylabel("Number of Nodes",fontsize=14)
plt.title("Degree Distribution",  fontsize=18)

plt.subplot(2,2,2)
plt.loglog(list(d.keys()), list(d.values()),'bo-', linewidth=0, markersize=10)
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

plt.xlabel("Degrees", fontsize=14)
plt.ylabel("Number of Nodes", fontsize=14)
plt.title("Log Degree Distribution", fontsize=18)

plt.tight_layout()
plt.show()

plt.show()
```



In general, a **power law** is assumed, stating that the number of nodes with n neighbors is proportional to $n^{-\gamma}$ for a constant γ. This can be inspected visually by plotting the degree distribution on a doubly logarithmic scale, on which a power law renders a straight line.

Clustering Coefficients (or Transitivity)

In transitivity, we analyze the linking behavior to determine whether it demonstrates a transitive behavior. In mathematics, for a transitive relation R, aRb ^ bRc --> aRc.

Transitivity is when a friend of my friend is my friend.

Higher Transtivity in a graph resuts in denser graph which in turns is closer to a complete graph.Thus, we can determine how close graphs are to the complete graph by measuring transitivity. This can be performed by measuring

1. Global Clustering Coefficient: computed for a network
2. Local Clustering Coefficient: computed for a node

1. Global Clustering Coefficient

The global clustering coefficient is based on triplets of nodes. A triplet consists of three connected nodes.
The global clustering coefficient is the number of closed triplets (or 3 x triangles) over the total number of triplets (both open and closed).

In [180]:
```
print(nx.transitivity(G))
```

```
0.3532346251940022
```

The **Global Clustering Coefficient** is 0.35, it means that there are several triads present in the Graph.

It is true as there are several routes like

- New York to Chicago
- Chicago to Los Angeles
- Los Angeles to New York

2. Local Clustering Coefficient

The local clustering coefficient of a node in a graph quantifies how close its neighbors are to being a complete graph.
The local clustering coefficient Ci for a vertex Vi is then given by the proportion of links between the vertices within its neighbourhood divided by the number of links that could possibly exist between them.

In [9]:
```
print(nx.average_clustering(G))
```

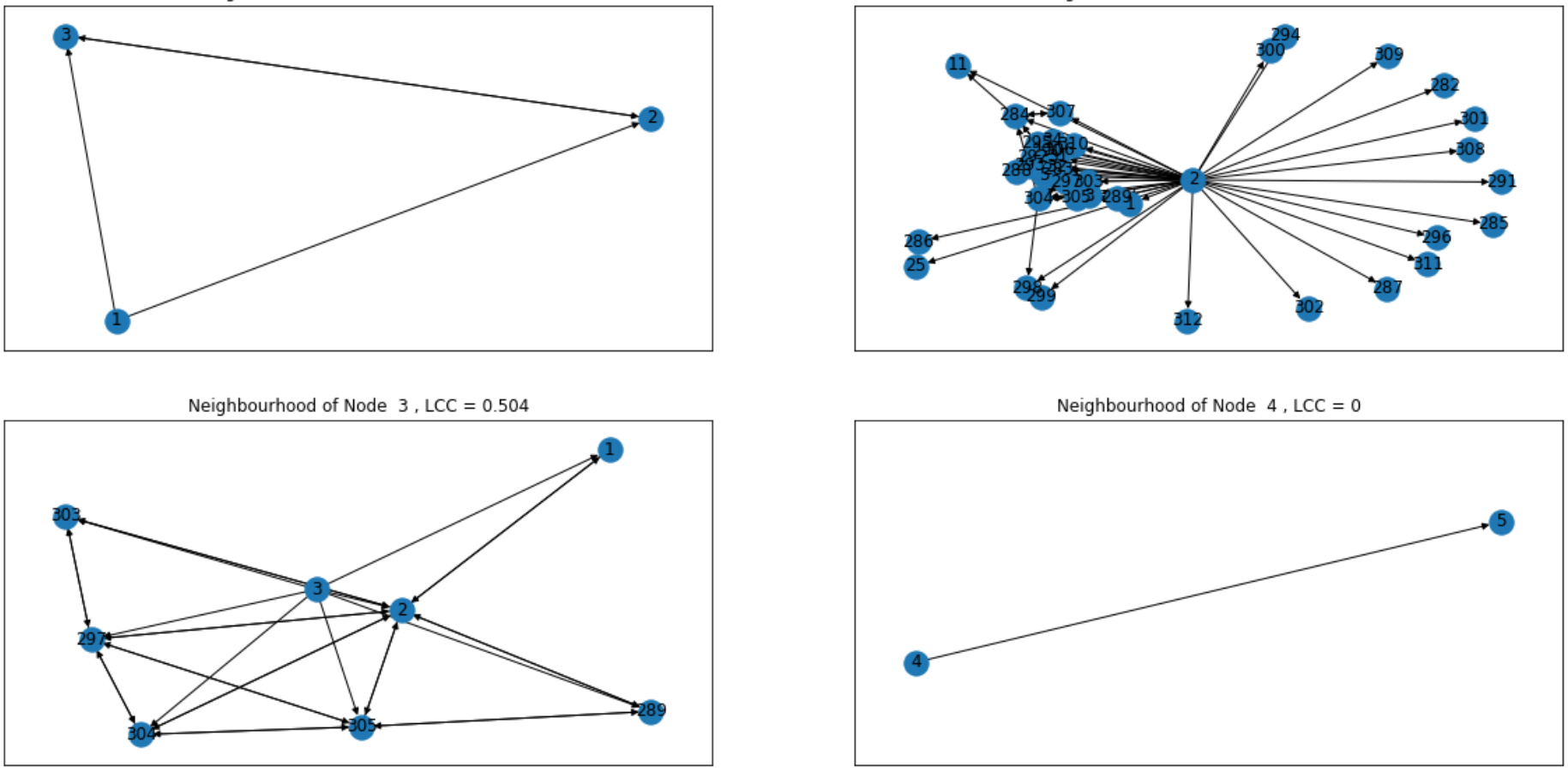
```
0.4885316349607309
```

Average Local Clustering Coefficient is 0.488 ~ 0.5 which means that **most of the airports are highly connected to each other.**

In [10]:
```
clustering_coefficients = list(nx.clustering(G).items())
```

In [11]:
```
plt.figure(figsize=(20,10))
for i in range(1,5):
    plt.subplot(2,2,i)
    plt.title("Neighbourhood of Node  "+ str(i) + " , LCC = " +str(clustering_coefficients[i-1][1])[:5]+"")
    G_node1 = nx.DiGraph()
    G_node1.add_node(i)
    node1_neighb = {}
    for s,t in G.edges():
        if s==i:
            G_node1.add_node(t)
            G_node1.add_edge(s,t)
            node1_neighb[t]=1
    for s,t in G.edges():
        if s in node1_neighb and t in node1_neighb:
            G_node1.add_edge(s,t)
    nx.draw_networkx(G_node1,with_lables=True)
```

```
C:\Users\mahav\Anaconda3\lib\site-packages\networkx\drawing\nx_pylab.py:579: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if not cb.iterable(width):
C:\Users\mahav\Anaconda3\lib\site-packages\networkx\drawing\nx_pylab.py:676: MatplotlibDeprecationWarning:
The iterable function was deprecated in Matplotlib 3.1 and will be removed in 3.3. Use np.iterable instead.
  if cb.iterable(node_size):  # many node sizes
```



These plots shows exmaple of Local clusterings of some Airports. Example : **Plot 1 each node is connected to each other therefore its LCC=1**, and similarly for other airports their neighbourhood is shown

Reciprocity

Reciprocity is a simplified version of transitivity, because it considers closed loops of length 2. It counts number of reciprocal pairs in a graph, i.e. if a flight from A to B exist, then does flight from B to A also exists or not.

$$R = \frac{1}{m} Tr(A^2)$$

where,
m=total no of edges in the graph

$$tr(A) = A_{1,1} + A_{2,2} + A_{3,3} + \ldots + A_{n,n}$$

In [12]:
```
print(nx.reciprocity(G))
```

```
0.7806346508003966
```

78% of the edges are reciprocal.

Centrality Measures

Centrality is concept used to identify how important a node is in a given graph. In this report we would be looking into the following measures -

- Degree Centrality
- Eigenvector Centrality
- PageRank Centrality
- Closeness Centrality
- Betweenness Centrality

1. Degree Centrality

Degree of a node is the number of nodes that it is connected with. As we have taken directed Dataset so there will be two types of degree centality:

1. **In Degree**: no of Incomming edges
2. **Out Degree**: no of Outgoing edges

1.1 Indegree

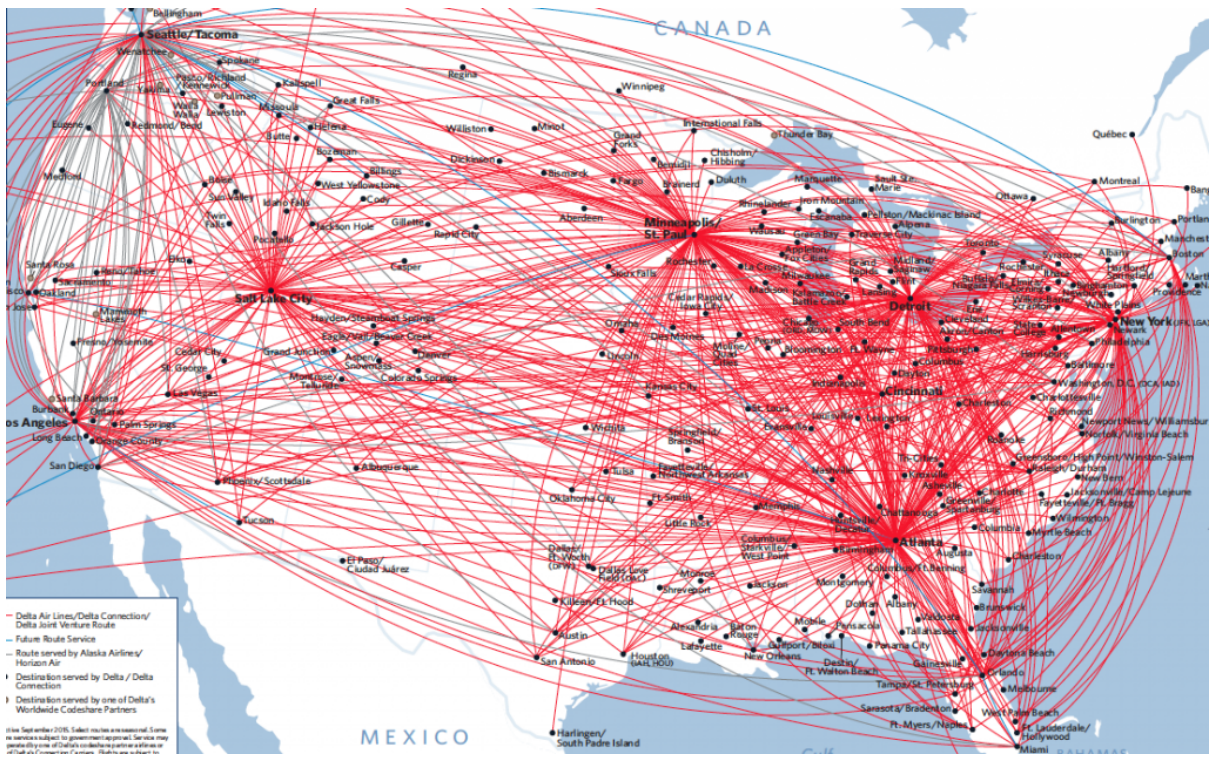
In [13]:
```
in_degrees = list(G.in_degree())
in_degrees.sort(key=lambda x:x[1],reverse=True)
```

In [14]:
```
print("Top 15 - with Maximum In-Degree (AirportId, Indegree)\n\n",*in_degrees[:15])
```

```
Top 15 - with Maximum In-Degree (AirportId, Indegree)

 (46, 294) (88, 270) (69, 261) (74, 256) (165, 256) (150, 244) (159, 240) (174, 240) (147, 239) (317, 232) (57, 227) (6
0, 225) (133, 216) (164, 214) (90, 210)
```

We can see that there are 15 Airports at which daily more than 200 flights from different destinations land in. These Airports can be hubs of major airlines at major US cities wherein flights from teir-2, teir-3 cities fly in.

This is can be verified by looking at the Delta Airlines Routes Map (Source - Google Images)



We can see that there are several hubs from which routes to small cities are flyed.

1.2 Outdegree

In [15]:
```
od = G.out_degree()
out_degrees = list(G.out_degree())
out_degrees.sort(key=lambda x:x[1],reverse=True)
```

In [16]:
```
print("Top 15 - with Maximum Out-Degree (AirportId, Indegree)\n\n",*out_degrees[:15])
```

```
Top 15 - with Maximum Out-Degree (AirportId, Indegree)

 (46, 302) (88, 279) (74, 270) (69, 268) (165, 266) (147, 258) (174, 257) (150, 255) (159, 253) (57, 245) (317, 239) (6
0, 238) (90, 221) (80, 216) (164, 215)
```

We see that almost all the airports which were in Top-15 Indegree list are in Top-15 Outdegree list too, which was expected people fly from/to the hubs.

2. Eigen Vector Centrality

Eigen Vector Centrality incorporates importance of neighbours, i.e if a node in connected to more important nodes in the graph then its importance will also increase.

In [17]:
```
eig_cent = list(nx.eigenvector_centrality(G).items())
eig_cen = [(i[0],i[1]) for i in eig_cent]
eig_cen.sort(key=lambda x:x[1],reverse=True)
top15 = [i[0] for i in eig_cen[:15]]
tt = {}
for i in top15:
    tt[i]=1

hubs_graph = nx.DiGraph()
no_of_edge=0
for s,t in G.edges():
    if s in tt and t in tt:
        hubs_graph.add_nodes_from([s,t])
        hubs_graph.add_edge(s,t)
        no_of_edge+=1
```

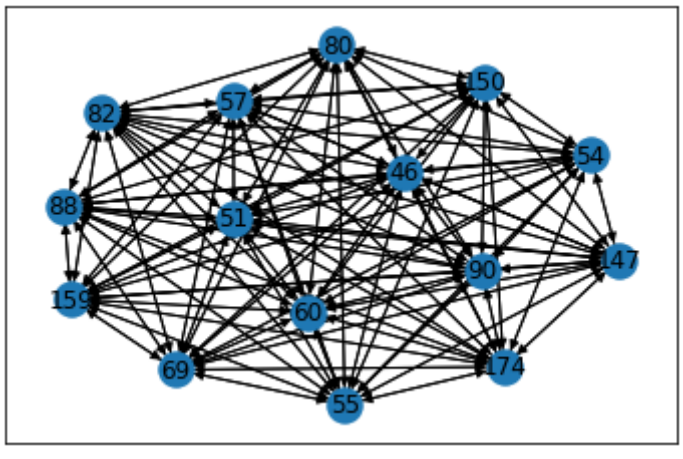
In [18]:
```
print("Top 15 - Eigen Vector Centralities\n\n",*eig_cen[:15])
```

```
Top 15 - Eigen Vector Centralities

 (46, 0.12595638595779174) (88, 0.125870550559244) (60, 0.12306502164951313) (174, 0.12133019341397366) (57, 0.12037076
61371985) (69, 0.11763006740481646) (90, 0.11547863624589512) (159, 0.11482769789356145) (147, 0.11275596704942659) (15
0, 0.11268352070040344) (80, 0.11155992981937837) (54, 0.11030144674673223) (51, 0.10899376028953549) (55, 0.1078568292
8224585) (82, 0.10738904427493143)
```

In eigen-vector centrality too all the major hubs are present, which tells us that all the hubs are well connected or fully connected, i.e. from a hub a there are flights to every other hub. (or they form a complete graph). This can verified from the graph below.

In [19]:
```
print("Connectivity of Top - 15 Betweeness Centrality Airports (Hubs) - Form a Complete Graph")
print("No of Nodes = 15, No. of Edges = ",no_of_edge)
nx.draw_networkx(hubs_graph,with_lables=True)
```

```
Connectivity of Top - 15 Betweeness Centrality Airports (Hubs) - Form a Complete Graph
No of Nodes = 15, No. of Edges =  210
```



Also all some hubs (46,88,60,174) have almost similar centrality value (around 0.125), which tells us that these are major hubs from which you can fly out to almost every city.

We can also see that there are some airports **which didn't had high in/out degree but have high eigen centrality** value. We can reason this as in **eigen centrality** a node as high centrality if it is connected to more important node and in our result we can say nodes like 55,82 are tier-2 cities which have connections to every major hub which makes it too a central figure in the graph.

3. Betweeness Centrality

Betweeness Centrality is by considering how important nodes are in connecting other nodes.

In [20]:
```
bet_cent = list(nx.betweenness_centrality(G).items())
bet_cent.sort(key=lambda x:x[1],reverse=True)
top15 = [i[0] for i in bet_cent[:15]]
tt = {}
for i in top15:
    tt[i]=1
```

In [21]:
```
print("Top 15 - Betweeness Centralities\n\n",*bet_cent[:15])
```

```
Top 15 - Betweeness Centralities

 (32, 0.20193513939310745) (22, 0.0839762325147387) (165, 0.05448819757008784) (74, 0.05437419824941006) (147, 0.053296
313613762115) (10, 0.049815492937504674) (195, 0.04978561494093804) (317, 0.04544366182863709) (174, 0.0454436471837035
35) (46, 0.04491468910843342) (418, 0.035043199307476784) (159, 0.03429095883509927) (69, 0.03426262399760477) (136, 0.
03089744114646743) (150, 0.03034608567065613)
```

Two observations can me made out from the above result.

1. Hubs found in previous centrality measures have relatively low betweeness centrality.
2. Some new Airports are present in Top-15 betweeness centrality list which were not present in other centrality measures we looked at earlier.

For the first observation we can say that hubs have direct connections to most of the nodes.

And for the second we see that Airports like (32) though having less connections to other cities have high betweeness centrality (0.20), i.e. many routes from (s to t) have it in between its shortest path. It can be regarded as a hub at a mid size city at a very strategic location in the US.

4. Closeness Centrality

Closness Centrality measures that the more central nodes are, the more quickly they can reach other nodes, i.e. these nodes should have a smaller average shortest path length to other nodes.

In [22]:
```
close_cent = list(nx.closeness_centrality(G).items())
close_cent.sort(key=lambda x:x[1],reverse=True)
s = 0
for i in close_cent:
    s+=i[1]
print("Average Closness Centrality = ",s/len(close_cent))
print("Average Shortest Path Length = ",len(close_cent)/s)
```

```
Average Closness Centrality =  0.2855955879248404
Average Shortest Path Length =  3.5014546522447256
```

We see that the Average Closeness centrality is about 0.28, that for most of the cities we can reach every other city at at average with 2 stops only.

In [23]:
```
print("Top 15 - Closeness Centralities\n\n",*close_cent[:15])
```

```
Top 15 - Closeness Centralities

 (174, 0.45541835061995506) (165, 0.45096365530903276) (147, 0.4503762718439608) (46, 0.44906023728337774) (88, 0.44616
30744621947) (159, 0.4428772112817174) (164, 0.4423106910242416) (150, 0.4411819875064764) (60, 0.4406197931929925) (19
5, 0.43866334628379433) (90, 0.4376916236812669) (74, 0.43727648777515127) (32, 0.4354866280959709) (57, 0.433847406158
3449) (80, 0.4338474061583449)
```

All the top-15 closness centrality values airport have almost the same centrality values (around 0.43), i.e having only 1 stop at average to reach any other US city. **Example** - To reach a small city in U.S east coast from a major city on the West coast one will only have a single stop-over at a major city on the east coast.

PageRank Centrality

PageRank Centrality divides the value of passed centrality by number of outgoing links (out-degree) from that node such that each connected neighbor gets a fraction of the source node's centrality:

In [24]:
```
pg_rnk = list(nx.pagerank(G).items())
pg_rnk.sort(key=lambda x:x[1],reverse=True)
print("Top 15 - PageRank Centralities\n\n",*pg_rnk[:15])
```

```
Top 15 - PageRank Centralities

 (46, 0.036557637123265) (88, 0.028133191571163077) (165, 0.02555981565498619) (147, 0.024040474283167) (57, 0.02289837
864964141) (195, 0.02248241549925754) (74, 0.022473315975274584) (32, 0.022232513583819415) (159, 0.01794150881033278)
(196, 0.016908247940897157) (317, 0.01684227750178473) (164, 0.016347286676549062) (183, 0.01618638577112354) (174, 0.0
1613477235320718) (55, 0.015040454921553542)
```

Here too we get almost same results, hubs are having the highest centralities.

Problem 1 - Dataset 2

About the Dataset

Title : **Facebook**
About : This dataset consists of 'circles' (or 'friends lists') from Facebook. Facebook data was collected from survey participants using this Facebook app. The dataset includes node features (profiles), circles, and ego networks. This is a Undirected Graph.

In [25]:
```
G2=nx.read_edgelist('facebook_combined.txt',create_using=nx.Graph(),nodetype=int)
```

In [26]:
```
print(nx.info(G2))
```

```
Name:
Type: Graph
Number of nodes: 4039
Number of edges: 88234
Average degree:  43.6910
```

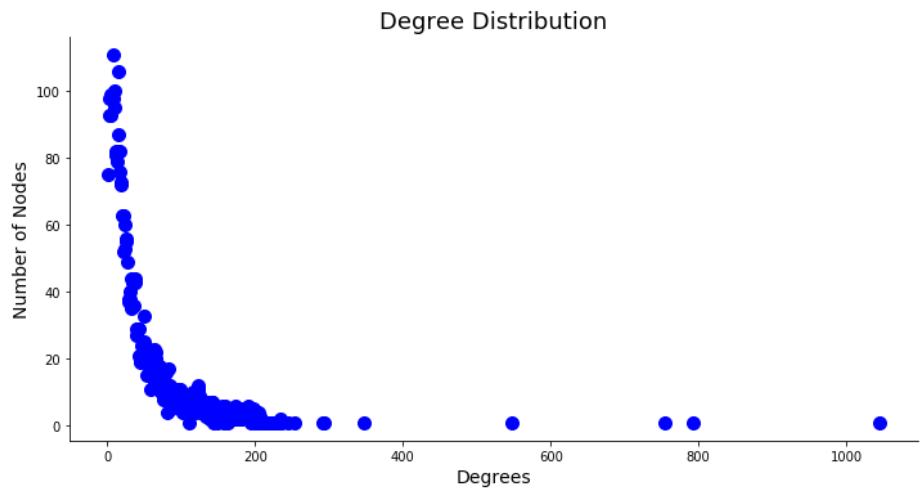
This graph contains 4039 users with 88234 connections among them

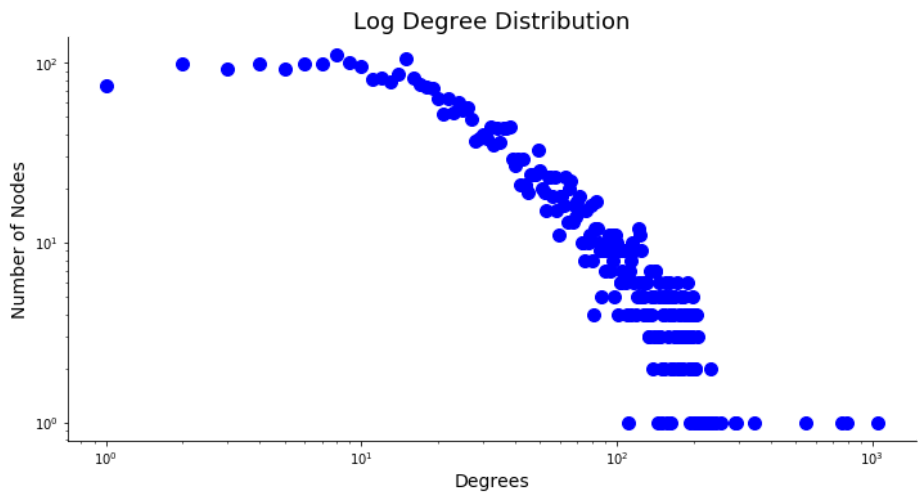
Degree Distribution

In the study of graphs and networks, the degree of a node in a network is the number of connections it has to other nodes and the degree distribution is the probability distribution of these degrees over the whole network.
As we can see form the Graph there are many nodes with very less degree and with some nodes having high degree which is a good example of real type network.

In [27]:
```
d = dict()
for node,degree in nx.degree(G2):
    d[degree] = d.get(degree,0)+1
```

In [28]:
```
plt.figure(figsize=(20,10))
plt.subplot(2, 2, 1)
plt.plot(list(d.keys()), list(d.values()),'bo-', linewidth=0, markersize=10)
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

plt.xlabel("Degrees", fontsize=14)
plt.ylabel("Number of Nodes",fontsize=14)
plt.title("Degree Distribution",  fontsize=18)

plt.subplot(2,2,2)
plt.loglog(list(d.keys()), list(d.values()),'bo-', linewidth=0, markersize=10)
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

plt.xlabel("Degrees", fontsize=14)
plt.ylabel("Number of Nodes", fontsize=14)
plt.title("Log Degree Distribution", fontsize=18)

plt.tight_layout()
plt.show()

plt.show()
```





In general, a **power law** is assumed, stating that the number of nodes with n neighbors is proportional to $n^{-\gamma}$ for a constant γ. This can be inspected visually by plotting the degree distribution on a doubly logarithmic scale, on which a power law renders a straight line.

Clustering Coefficients (or Transitivity)

In transitivity, we analyze the linking behavior to determine whether it demonstrates a transitive behavior. In mathematics, for a transitive relation R, aRb ^ bRc --> aRc.

Transitivity is when a friend of my friend is my friend.

Higher Transtivity in a graph resuts in denser graph which in turns is closer to a complete graph.Thus, we can determine how close graphs are to the complete graph by measuring transitivity. This can be performed by measuring

1. Global Clustering Coefficient: computed for a network
2. Local Clustering Coefficient: computed for a node

1. Global Clustering Coefficient

The global clustering coefficient is based on triplets of nodes. A triplet consists of three connected nodes.
The global clustering coefficient is the number of closed triplets (or 3 x triangles) over the total number of triplets (both open and closed).

In [29]:
```
print(nx.transitivity(G2))
```

```
0.5191742775433075
```

The **Global Clustering Coefficient** is 0.51, it means that there are large number of triads present in the Graph.

2. Local Clustering Coefficient

The local clustering coefficient of a node in a graph quantifies how close its neighbors are to being a complete graph.
The local clustering coefficient Ci for a vertex Vi is then given by the proportion of links between the vertices within its neighbourhood divided by the number of links that could possibly exist between them.

In [30]:
```
print(nx.average_clustering(G2))
```

```
0.6055467186200876
```

Average Local Clustering Coefficient is 0.6 which means that **most of the users are highly connected to each other.**

In [31]:
```
clustering_coefficients = list(nx.clustering(G2).items())
```

Reciprocity

Reciprocity is a simplified version of transitivity, because it considers closed loops of length 2. It counts number of reciprocal pairs in a graph, i.e. if a flight from A to B exist, then does flight from B to A also exists or not.

$$R = \frac{1}{m} Tr(A^2)$$

where,
m=total no of edges in the graph

$$tr(A) = A_{1,1} + A_{2,2} + A_{3,3} + \ldots + A_{n,n}$$

In [32]:
```
print(nx.reciprocity(G2))
```

```
0.0
```

Reciprocity of this graph is 0 as it is a undirected graph

Centrality Measures

Centrality is concept used to identify how important a node is in a given graph. In this report we would be looking into the following measures -

- Degree Centrality
- Eigenvector Centrality
- Closeness Centrality
- Betweenness Centrality

1. Degree Centrality

Degree of a node is the number of nodes that it is connected with. As we have taken directed Dataset so there will be two types of degree centality:

1. **In Degree**: no of Incomming edges
2. **Out Degree**: no of Outgoing edges

In [33]:
```
degrees = list(G2.degree())
degrees.sort(key=lambda x:x[1],reverse=True)
```

In [34]:
```
print("Top 15 - with Maximum In-Degree (User, Indegree)\n\n",*degrees[:15])
```

```
Top 15 - with Maximum In-Degree (User, Indegree)

 (107, 1045) (1684, 792) (1912, 755) (3437, 547) (0, 347) (2543, 294) (2347, 291) (1888, 254) (1800, 245) (1663, 235)
(1352, 234) (2266, 234) (483, 231) (348, 229) (1730, 226)
```

We can see that there are 15 Users which have more than 200 friends on facebook. These users can be famous personalities.

2. Eigen Vector Centrality

Eigen Vector Centrality incorporates importance of neighbours, i.e if a node in connected to more important nodes in the graph then its importance will also increase.

In [35]:
```
eig_cent = list(nx.eigenvector_centrality(G2).items())
eig_cen = [(i[0],i[1]) for i in eig_cent]
eig_cen.sort(key=lambda x:x[1],reverse=True)
top15 = [i[0] for i in eig_cen[:15]]
```

In [36]:
```
print("Top 15 - Eigen Vector Centralities\n\n",*eig_cen[:15])
```

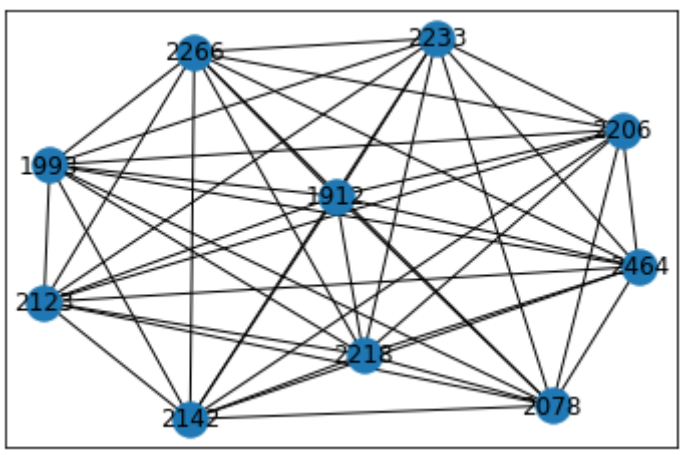
```
Top 15 - Eigen Vector Centralities

 (1912, 0.09540696149067629) (2266, 0.08698327767886553) (2206, 0.08605239270584343) (2233, 0.08517340912756598) (2464,
0.08427877475676092) (2142, 0.08419311897991796) (2218, 0.08415573568055032) (2078, 0.08413617041724979) (2123, 0.08367
141238206226) (1993, 0.0835324284081597) (2410, 0.08351751162148192) (2244, 0.08334186008004286) (2507, 0.0832731156814
4907) (2240, 0.08305685135432213) (2340, 0.08305335409204806)
```

From the top 15 users having largest conncetions wecan see that they also have high eigenvector centrality and it also shows that famous users are also connected with eachother. This can also be seen from the graph below.

In [37]:
```
top15 = [i[0] for i in eig_cen[:10]]
tt = {}
for i in top15:
    tt[i]=1

hubs_graph = nx.Graph()
no_of_edge=0
for s,t in G2.edges():
    if s in tt and t in tt:
        hubs_graph.add_nodes_from([s,t])
        hubs_graph.add_edge(s,t)
        no_of_edge+=1
```

In [38]:
```
print("Connectivity of Top - 15 Betweeness Centrality Users - Form a Complete Graph")
print("No of Nodes = 10, No. of Edges = ",no_of_edge)
nx.draw_networkx(hubs_graph,with_lables=True)
```

```
Connectivity of Top - 15 Betweeness Centrality Users - Form a Complete Graph
No of Nodes = 10, No. of Edges =  45
```



We can also see that there are some Users **which didn't had high degree but have high eigen centrality** value. We can reason this as in **eigen centrality** a node as high centrality if it is connected to more important node which makes it too a central figure in the graph.

3. Betweeness Centrality

Betweeness Centrality is by considering how important nodes are in connecting other nodes.

In [39]:
```
bet_cent = list(nx.betweenness_centrality(G2).items())
bet_cent.sort(key=lambda x:x[1],reverse=True)
print("Top 15 - Betweeness Centralities\n\n",*bet_cent[:15])
```

```
Top 15 - Betweeness Centralities

 (107, 0.4805180785560152) (1684, 0.3377974497301992) (3437, 0.23611535735892905) (1912, 0.2292953395868782) (1085, 0.1
4901509211665306) (0, 0.14630592147442917) (698, 0.11533045020560802) (567, 0.09631033121856215) (58, 0.084360205907964
86) (428, 0.06430906239323866) (563, 0.06278022847240787) (860, 0.05782590687091168) (414, 0.04763337297172344) (1577,
0.03978470502937034) (348, 0.03799809748091909)
```

Two observations can me made out from the above result.

1. Users found in previous centrality measures have relatively low betweeness centrality.
2. Some new Users are present in Top-15 betweeness centrality list which were not present in other centrality measures we looked at earlier.

For the first observation we can say that Users have direct connections to most of the nodes.

And for the second we see that Users like (107) though having less connections to other cities have high betweeness centrality (0.48), i.e. many routes from (s to t) have it in between its shortest path. It can be regarded as a mutual friend between two users.

4. Closeness Centrality

Closness Centrality measures that the more central nodes are, the more quickly they can reach other nodes, i.e. these nodes should have a smaller average shortest path length to other nodes.

In [189]:
```
close_cent = list(nx.closeness_centrality(G2).items())
close_cent.sort(key=lambda x:x[1],reverse=True)
s = 0
for i in close_cent:
    s+=i[1]
print("Average Closness Centrality = ",s/len(close_cent))
```

```
Average Closness Centrality =  0.2761677635668374
```

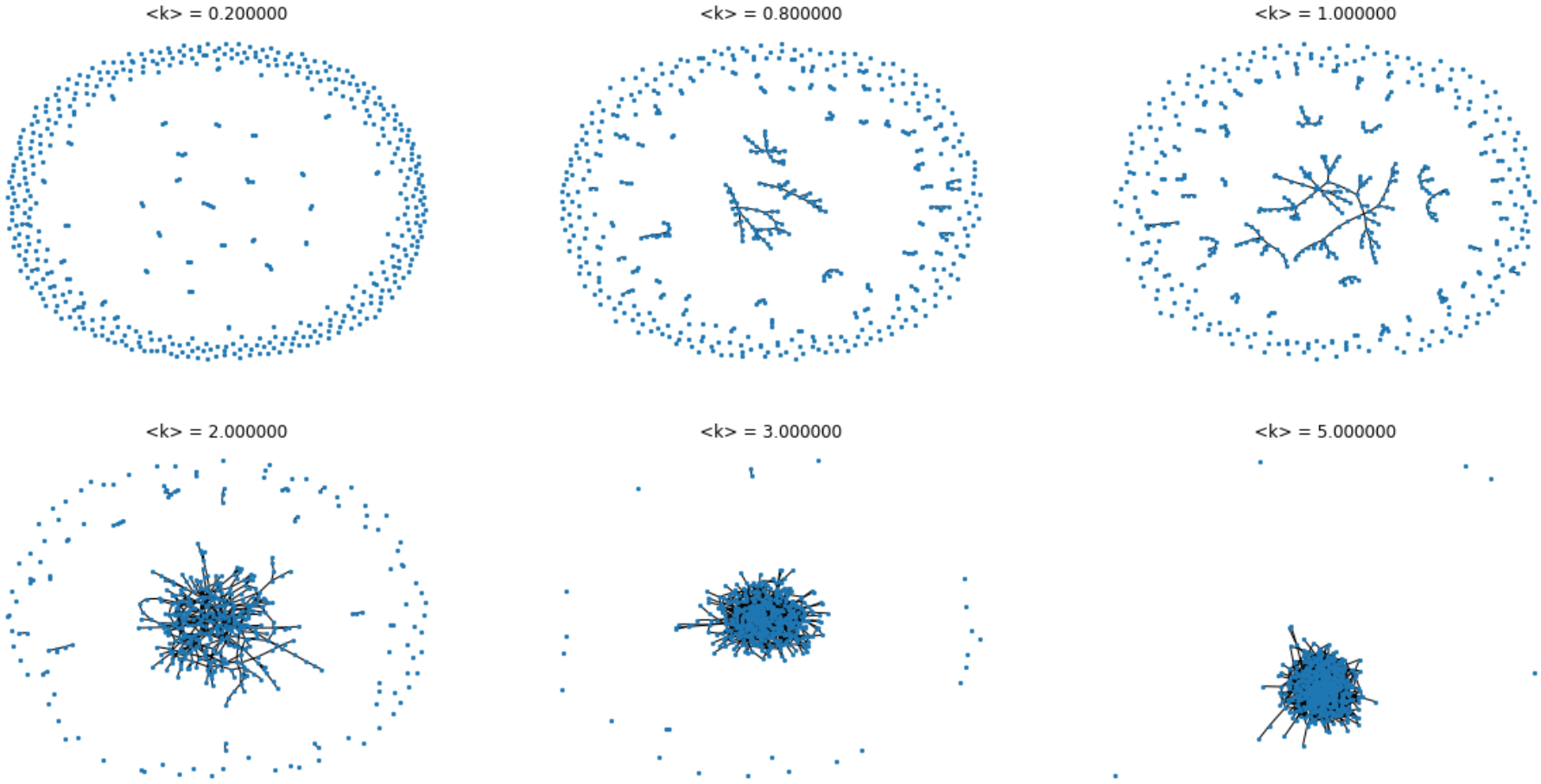
Average Closeness Centrality is 0.27 so we can reach from one user to another with maximum 3 users in between.

Problem 2

Appearance of a Giant Component in Random Network

Figures below shows a random network of 500 nodes in which as we increase the value of k and after it exceeds a critical value large clusters start to form rapidly before that there are many tiny cluster formation.

In [41]:
```
N = 500
k = [0.2,0.8,1,2,3,5]
plt.figure(figsize=(20,10))
pl = 1
for kk in k:
    p = kk/(N-1)
    er_G = nx.erdos_renyi_graph(N,p)
    plt.subplot(2,3,pl)
    plt.title("<k> = %f"%kk)
    pl+=1
    nx.draw(er_G,node_size=5)
```



As we can see increasing the **k** (average degree) for the random network we see bigger and bigger giant component. Initially giant components are sparse trees but as k get > 3 we see a nearly connected graph.

Evolution of a Random Network

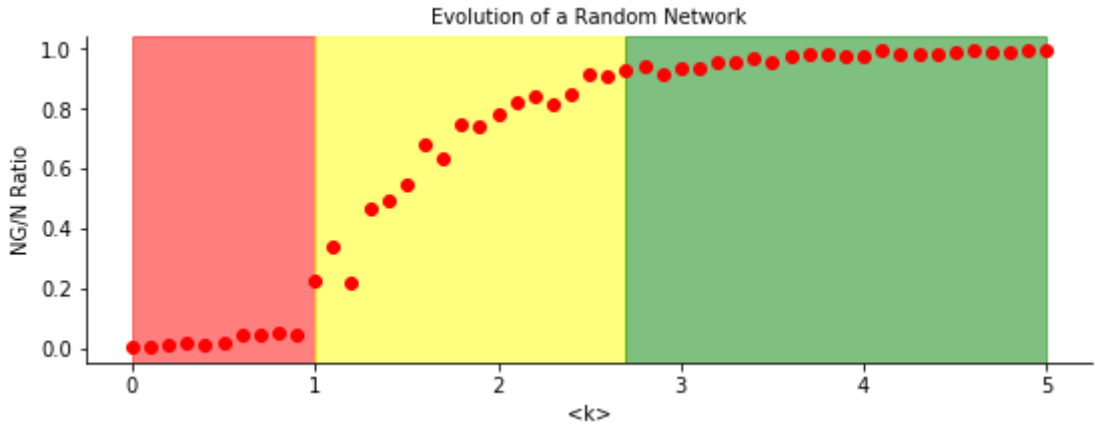
In this section we vary the average degree of each node in a random network and try to figure at what point all the nodes are inside the giant component!

In [42]:
```
list_k = []
ink = 0
while ink<=50:
    list_k.append(ink)
    ink = ink + 1
list_k = [i/10 for i in list_k]
```

In [43]:
```
N = 500
ng_n = {}
for k in list_k:
    p = k/(N-1)
    er_G = nx.erdos_renyi_graph(N,p)
    Gcc = sorted(nx.connected_components(er_G), key=len, reverse=True)
    G0 = er_G.subgraph(Gcc[0])
    ng_n[k]=nx.number_of_nodes(G0)/N
crit_p = 1
conn_reg = math.log(N,10)
```

In [44]:
```
fig,ax = plt.subplots(figsize=(9,3))

ax.plot(list(ng_n.keys()), list(ng_n.values()),'ro-', linewidth=0, markersize=6)

ax.axvspan(0,crit_p,alpha=0.5,color='red')
ax.axvspan(1,conn_reg,alpha=0.5,color='yellow')
ax.axvspan(conn_reg,5,alpha=0.5,color='green')

plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

plt.xlabel("<k>", fontsize=10)
plt.ylabel("NG/N Ratio",fontsize=10)
plt.title("Evolution of a Random Network",  fontsize=10)
```

Out[44]:
```
Text(0.5, 1.0, 'Evolution of a Random Network')
```



Graph Above shows 3 Regions:

1. **Subcritical Regime: Pink region** shown in the figure k varies from 0 to 1. In this region we get tiny clusters. **The reason is that for ‹k› ‹ 1 the largest cluster is a tree with size N_G ~ lnN**, hence its size increases much slower than the size of the network.

2. **Critical Point: Orange Line** in the figure, k=1.The critical point separates the regime where there is not yet a giant component (‹k› ‹ 1) from the regime where there is one (‹k› › 1). Both in the subcritical regime and at the critical point the largest component contains only a vanishing fraction of the total number of nodes in the network.

3. **Supercritical Point: Yellow region** shown in figure, k>1. This regime has the most relevance to real systems, as for the first time we have a giant component that looks like a network. In the supercritical regime numerous isolated components coexist with the giant component. The supercritical regime lasts until all nodes are absorbed by the giant component.
   In the above result the NG/N ratio varies from 0.1 to 0.8, i.e till average degree is about 2.69 (LogN) 80 % of nodes are consumed by the giant component.

4. **Connected Regime: Green region** shown in the figure, K> lnN. In the absence of isolated nodes the network becomes connected. When we enter the connected regime the network is still relatively sparse, as lnN / N → 0 for large N. The network turns into a complete graph only at ‹k› = N - 1.

In summary, the random network model predicts that the emergence of a network is not a smooth, gradual process: The isolated nodes and tiny components observed for small ‹k› collapse into a giant component through a phase transition. As we vary ‹k› we encounter four topologically distinct regimes.

Social Network Analysis Project Round 2

INDEX

PROBLEM 1 - Dataset 1 (US Airports)

About the Dataset

Title : **US Airports**
Category : Infrastructure
About : This is the **directed network** of flights between US airports in 2010. Each edge represents a connection from one airport to another, and the weight of an edge shows the number of flights on that connection in the given direction, in 2010.

Importing dataset and creating a networkx graph

In [197]:
```
flight = []
with open("out.tsv") as tsvfile:
    tsvreader = csv.reader(tsvfile, delimiter=' ')
    i = 0
    for line in tsvreader:
        if i > 1:
            flight.append((int(line[0]),int(line[1]),int(line[2])))
        i = i+1
G1 = nx.DiGraph()
G1.add_weighted_edges_from(flight)
G1 = G1.to_undirected()
print(nx.info(G1))
```

```
Name:
Type: Graph
Number of nodes: 1574
Number of edges: 17215
Average degree:  21.8742
```

The network has **1574 nodes** and **17215 edges**. Since the number of edges are huge (in the order of 10^5) and complexity of calculating the edge-betweenness centrality is **O(nm + n^2logn)** ***(Ref:1)***, so we need to perform sampling.

Finding Giant Component

In [211]:
```
giant_comp = max(nx.connected_components(G1), key=len)
print("No. of Nodes in Graph = ",len(G1.nodes()))
print("No. of Nodes in Giant Component (NG) = ",len(giant_comp))
print("Total no. of Components :",nx.number_connected_components(G1))
print("NG/N = ",len(giant_comp)/len(G1.nodes()))
```

```
No. of Nodes in Graph =  1574
No. of Nodes in Giant Component (NG) =  1572
Total no. of Components : 2
NG/N =  0.9987293519695044
```

Only 2 nodes are not in the giant component, so Ng/N is 0.99

Random Sampling Vertices from the Network

For performing the random sampling one idea is to randomly select a set number of nodes and all the edges connecting those nodes but in that case the structure or characterstics of the network are lost, which will then give wrong results. So to **preserve the network characterstics** we are using **Random Walk Algorithm.** ***(Ref:2)***

We have used a package consisting of Random Walk Algorithm created by a fellow n/w analyst on GitHub. *(Ref:3)*

In [199]:
```
import Graph_Sampling
x = Graph_Sampling.SRW_RWF_ISRW()
G3 = x.random_walk_induced_graph_sampling(G1, 300)
G3 = nx.Graph(G3)
print(nx.info(G3))
```

```
Name:
Type: Graph
Number of nodes: 300
Number of edges: 8742
Average degree:  58.2800
```

Sampling 200 nodes from the network gives us around 2 * 10^3 edges which will suitable for the algorithm.

Finding Communities using Girvan Newman Algorithm

Step 1 : Implementing Girvan NewMan Algorithm

Algorithm

1. Find no. of Connected Components (Subgraphs)
2. While no. of Connected Components are same : Remove the edge having maximum edge-betweenness centrality.

In [200]:
```
def edge_to_remove(G):
    edge_bw_cent = nx.edge_betweenness_centrality(G).items()
    to_remove = max(edge_bw_cent,key=lambda x:x[1])
    return to_remove[0]
```

In [201]:
```
def girvan_newmman(G):
    sub = nx.number_connected_components(G)
    print("No of Subgraphs in the Graph Initially = ",sub)
    sub_c=sub
    print("REMOVED EDGES : ")
    while sub==sub_c:
        rem = edge_to_remove(G)
        G.remove_edge(*rem)
        sub_c = nx.number_connected_components(G)
        print("",rem,end=" ")
    print()
    return G
```

Above 2 Functions are Implementing the Girvan Newman Algorithm

Step 2 : Implementing Community Plotting Function

We have used a code snippet from a stack-overlow user for plotting communitites. **(Ref 4)**

In [272]:
```
from community import *

def plot_community(G3,new_G):
    gn_communities = list(nx.connected_components(new_G))
    gn_dict_communities = {}
    for i, c in enumerate(gn_communities):
        for node in c:
            gn_dict_communities[node] = i + 1
        for node in G3:
            if node not in gn_dict_communities.keys():
                gn_dict_communities[node] = -1

    gn_pos = community_layout(G3, gn_dict_communities)
    from matplotlib import cm
    gn_colors = []
    for node in G3.nodes:
        gn_colors.append(cm.Set1(gn_dict_communities[node]))

    plt.figure(figsize=(5,5))
    nx.draw_networkx_nodes(G3, gn_pos, node_color=gn_colors, node_size=500)
    nx.draw_networkx_edges(G3, gn_pos,alpha=1)
    plt.axis('off')
    plt.show()
```

Step 3 : Finding Communities and Plotting them

In [202]:
```
print("No. of Connected Components = ",nx.number_connected_components(G3))
```

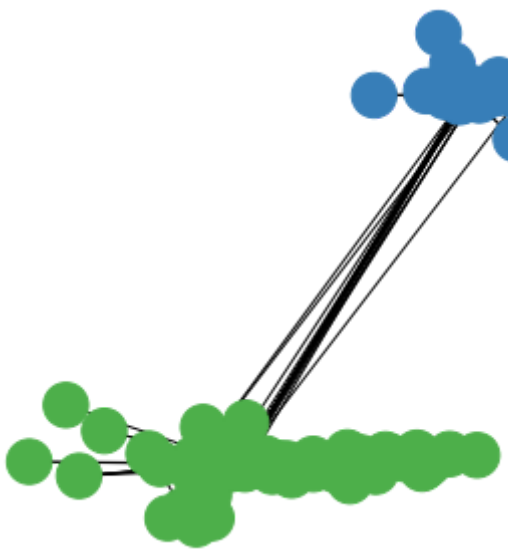
```
No. of Connected Components =  1
```

We see that initially the network has only one connected components, i.e. graph is connected.

1st Iteration

In [169]:
```
new_G = girvan_newmman(nx.Graph(G3))
print("Size of Components in the Graph :",[len(c) for c in sorted(nx.connected_components(new_G), key=len, reverse=True)])
plot_community(G3,new_G)
G3 = nx.Graph(new_G)
```

```
No of Subgraphs in the Graph Initially =  1
REMOVED EDGES :
 (31, 45)  (31, 173)  (31, 158)  (31, 56)  (31, 87)  (31, 89)  (31, 59)  (31, 68)  (31, 149)  (31, 145)  (31, 132)  (3
1, 73)  (31, 82)  (31, 316)  (31, 164)  (31, 146)  (31, 163)  (31, 187)  (31, 136)  (31, 137)  (31, 105)  (31, 78)  (3
1, 202)  (31, 195)  (31, 182)  (31, 112)  (31, 194)  (31, 123)  (31, 96)  (31, 475)  (31, 204)  (539, 31)  (31, 197)
(21, 45)  (31, 496)  (31, 314)  (512, 31)  (31, 276)  (21, 173)  (21, 73)  (21, 146)  (21, 164)  (21, 112)  (21, 194)
(21, 189)  (513, 204)  (13, 204)  (514, 31)  (21, 496)  (21, 314)  (22, 188)  (194, 503)  (519, 194)  (22, 314)  (314,
503)
Size of Components in the Graph : [223, 77]
```



2 Communities are detected in the first step, **with sizes 78 and 72.** We can see that most of the edges removed are from nodes 21 and 31 which we had classified hubs in round 1 of the project. So we may say that these hubs correspond to US East and West coast airports.

2nd Iteration

In [170]:
```
new_G = girvan_newmman(nx.Graph(G3))
print("Size of Components in the Graph :",[len(c) for c in sorted(nx.connected_components(new_G), key=len, reverse=True)
plot_community(G3,new_G)
G3 = nx.Graph(new_G)
```

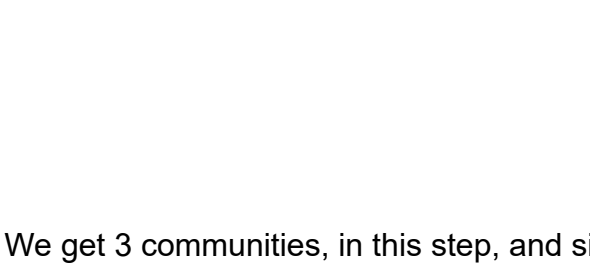
```
No of Subgraphs in the Graph Initially =  2
REMOVED EDGES :
 (123, 125)
Size of Components in the Graph : [222, 77, 1]
```

We get 3 communities, in this step, and similary other steps follow below.

3rd Iteration

In [171]:
```
new_G = girvan_newmman(nx.Graph(G3))
print("Size of Components in the Graph :",[len(c) for c in sorted(nx.connected_components(new_G), key=len, reverse=True)
plot_community(G3,new_G)
G3 = nx.Graph(new_G)
```

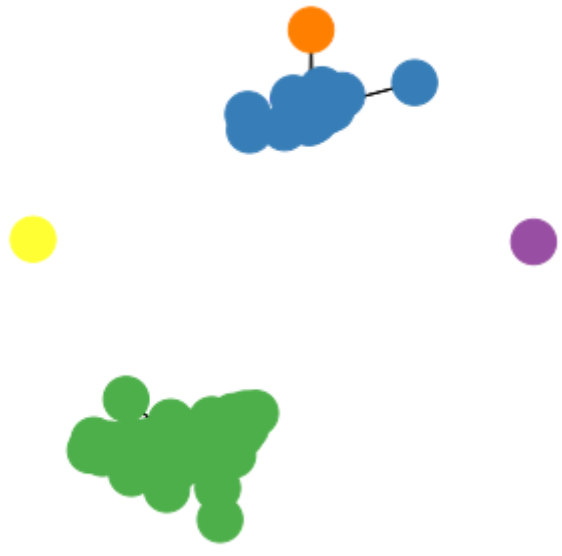
```
No of Subgraphs in the Graph Initially =  3
REMOVED EDGES :
 (1184, 1512)
Size of Components in the Graph : [221, 77, 1, 1]
```



4th Iteration

In [172]:
```
new_G = girvan_newmman(nx.Graph(G3))
print("Size of Components in the Graph :",[len(c) for c in sorted(nx.connected_components(new_G), key=len, reverse=True)
plot_community(G3,new_G)
G3 = nx.Graph(new_G)
```

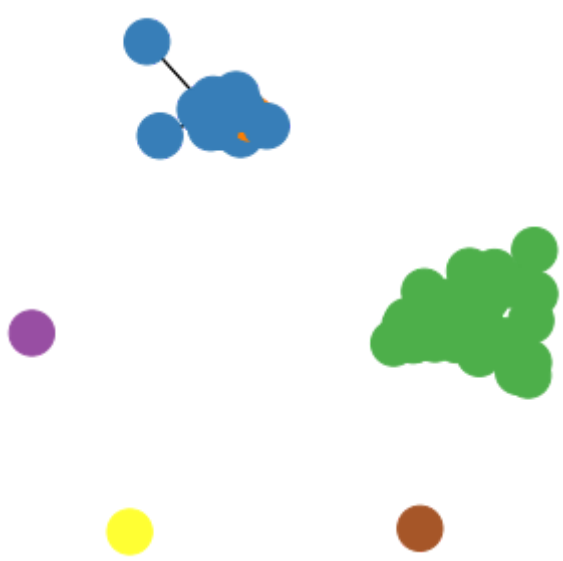
```
No of Subgraphs in the Graph Initially =  4
REMOVED EDGES :
 (57, 985)  (706, 985)
Size of Components in the Graph : [220, 77, 1, 1, 1]
```



5th Iteration

In [173]:
```
new_G = girvan_newmman(nx.Graph(G3))
print("Size of Components in the Graph :",[len(c) for c in sorted(nx.connected_components(new_G), key=len, reverse=True)
plot_community(G3,new_G)
G3 = nx.Graph(new_G)
```

```
No of Subgraphs in the Graph Initially =  5
REMOVED EDGES :
 (770, 314)  (514, 770)
Size of Components in the Graph : [219, 77, 1, 1, 1, 1]
```



PROBLEM 1 - Dataset 2 (Social Circles)

About the Dataset

Title : **Facebook**
About : This dataset consists of 'circles' (or 'friends lists') from Facebook. Facebook data was collected from survey participants using this Facebook app. The dataset includes node features (profiles), circles, and ego networks. This is a Undirected Graph.

Importing dataset and creating a networkx graph

In [257]:
```
G2=nx.read_edgelist('facebook_combined.txt',create_using=nx.Graph(),nodetype=int)
print(nx.info(G2))
```

```
Name:
Type: Graph
Number of nodes: 4039
Number of edges: 88234
Average degree:  43.6910
```

Giant Component

In [258]:
```
giant_comp = max(nx.connected_components(G2), key=len)
print("No. of Nodes in Graph = ",len(G2.nodes()))
print("No. of Nodes in Giant Component (NG) = ",len(giant_comp))
print("Total no. of Components :",nx.number_connected_components(G2))
print("NG/N = ",len(giant_comp)/len(G2.nodes()))
```

```
No. of Nodes in Graph =  4039
No. of Nodes in Giant Component (NG) =  4039
Total no. of Components : 1
NG/N =  1.0
```

All the nodes are inside the giant component, so Ng/N is 1

Random Sampling Vertices from the Network

For performing the random sampling one idea is to randomly select a set number of nodes and all the edges connecting those nodes but in that case the structure or characterstics of the network are lost, which will then give wrong results. So to **preserve the network characterstics** we are using **Random Walk Algorithm.** ***(Ref:2)***

We have used a package consisting of Random Walk Algorithm created by a fellow n/w analyst on GitHub. *(Ref:3)*

In [265]:
```
import Graph_Sampling
x = Graph_Sampling.SRW_RWF_ISRW()
G3 = x.random_walk_induced_graph_sampling(G2, 300)
G3 = nx.Graph(G3)
print(nx.info(G3))
```

```
Name:
Type: Graph
Number of nodes: 300
Number of edges: 3684
Average degree:  24.5600
```

Number of Connected Components Initially

In [266]:
```
print("No. of Connected Components = ",nx.number_connected_components(G3))
```

```
No. of Connected Components =  1
```

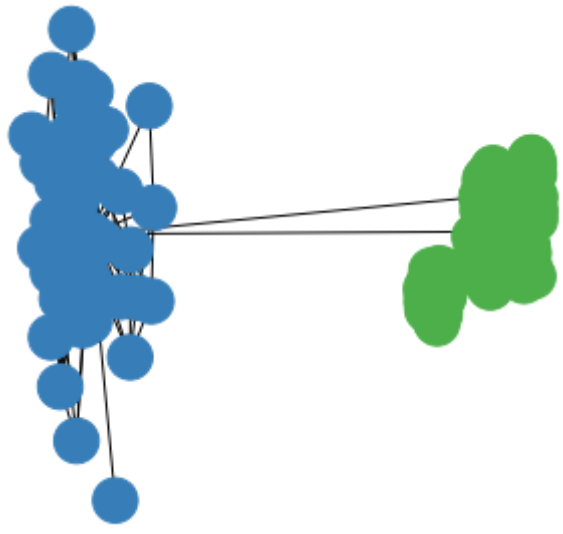
We see that the sampled graph is connected.

Finding Communities using Girvan Newman Algorithm

Iteration 1

In [267]:
```
new_G = girvan_newmman(nx.Graph(G3))
print("Size of Components in the Graph :",[len(c) for c in sorted(nx.connected_components(new_G), key=len, reverse=True)])
plot_community(G3,new_G)
G3 = nx.Graph(new_G)
```

```
No of Subgraphs in the Graph Initially =  1
REMOVED EDGES :
 (0, 107)  (0, 171)
Size of Components in the Graph : [256, 44]
```



Two communites are found in the first step with **256 and 44 nodes**.

Iteration 2

In [268]:
```
new_G = girvan_newmman(nx.Graph(G3))
print("Size of Components in the Graph :",[len(c) for c in sorted(nx.connected_components(new_G), key=len, reverse=True)])
plot_community(G3,new_G)
G3 = nx.Graph(new_G)
```

```
No of Subgraphs in the Graph Initially =  2
REMOVED EDGES :
 (571, 107)  (107, 349)  (107, 679)  (107, 1332)  (571, 1332)  (107, 171)  (679, 171)
Size of Components in the Graph : [219, 44, 37]
```

Three Communities are found in the second iteration. We se that previous green community has split into 2 communites. Now the communites are with **219, 44 and 37 nodes**

Iteration 3

In [269]:
```
new_G = girvan_newmman(nx.Graph(G3))
print("Size of Components in the Graph :",[len(c) for c in sorted(nx.connected_components(new_G), key=len, reverse=True)
plot_community(G3,new_G)
G3 = nx.Graph(new_G)
```

```
No of Subgraphs in the Graph Initially =  3
REMOVED EDGES :
 (107, 365)  (107, 371)  (107, 366)  (107, 350)  (107, 372)  (107, 368)  (107, 359)  (107, 361)  (107, 364)  (349, 366)
(561, 359)  (107, 723)  (1060, 107)  (703, 371)  (107, 1159)  (723, 876)  (723, 1318)  (107, 818)  (1116, 372)  (1581,
366)  (1144, 361)  (107, 487)  (107, 1366)  (838, 487)  (107, 1015)  (1366, 863)  (1279, 366)  (1268, 366)  (1356, 366)
(585, 366)  (1313, 366)  (1068, 366)  (1147, 366)  (1333, 366)  (797, 364)  (1161, 1366)  (610, 487)  (1028, 487)  (70
3, 487)  (1144, 1366)  (359, 1015)  (908, 487)  (1279, 1366)  (1060, 1015)
Size of Components in the Graph : [131, 88, 44, 37]
```


The community with **219 nodes has split up into two with 131 and 88 nodes respectively.**

We also observe that in comparison with the previous dataset more larger communities are found, a reason could be that since this dataset is of a social network, it tends to have close-knitted communities than a dataset of airport and flights.

Iteration 4

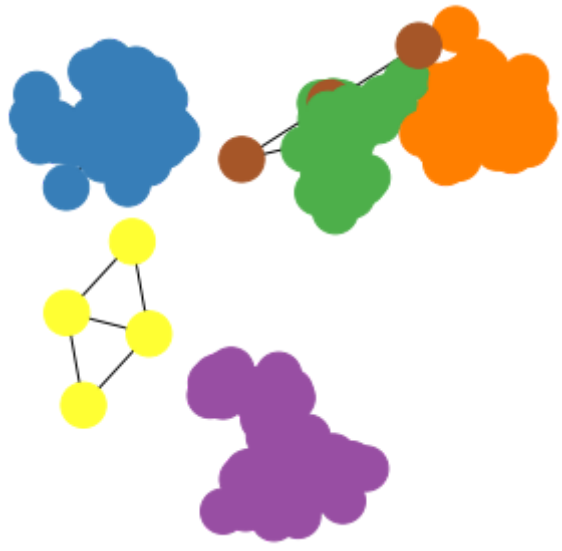
In [270]:
```
new_G = girvan_newmman(nx.Graph(G3))
print("Size of Components in the Graph :",[len(c) for c in sorted(nx.connected_components(new_G), key=len, reverse=True)
plot_community(G3,new_G)
G3 = nx.Graph(new_G)
```

```
No of Subgraphs in the Graph Initially =  4
REMOVED EDGES :
 (349, 365)  (607, 107)  (107, 1167)  (107, 797)  (1116, 107)  (107, 1268)  (1268, 797)  (1116, 797)
Size of Components in the Graph : [127, 88, 44, 37, 4]
```

Iteration 5

In [271]:
```
new_G = girvan_newmman(nx.Graph(G3))
print("Size of Components in the Graph :",[len(c) for c in sorted(nx.connected_components(new_G), key=len, reverse=True)
plot_community(G3,new_G)
G3 = nx.Graph(new_G)
```

```
No of Subgraphs in the Graph Initially =  5
REMOVED EDGES :
 (1651, 349)  (107, 1270)  (107, 1313)  (107, 732)  (107, 1388)  (107, 909)  (107, 1356)  (107, 924)  (1356, 909)  (73
2, 1388)
Size of Components in the Graph : [124, 88, 44, 37, 4, 3]
```



References (For Problem 1)

[1] Performance Analysis of an Algorithm for Computation of Betweenness Centrality https://link.springer.com/chapter/10.1007/978-3-642-21934-4_44 (https://link.springer.com/chapter/10.1007/978-3-642-21934-4_44)

[2] Sampling from Large Graphs https://cs.stanford.edu/people/jure/pubs/sampling-kdd06.pdf (https://cs.stanford.edu/people/jure/pubs/sampling-kdd06.pdf)

[3] Graph Sampling Package https://github.com/Ashish7129/Graph_Sampling (https://github.com/Ashish7129/Graph_Sampling)

[4] Community Plotting Code https://stackoverflow.com/questions/43541376/how-to-draw-communities-with-networkx (https://stackoverflow.com/questions/43541376/how-to-draw-communities-with-networkx)

PROBLEM 2

Create a scale-free network using Net Logo. Apply ICM (Independent Cascade Model) to find the maximum number steps required to get to the maximum number of nodes.

1. CREATING A SCALE FREE NETWORK

For creating a scale free network, we are using **Preferential Attachment Model** *(described by Barabasi and Albert – 1999).* In this model a new node is created at each time step and connected to existing nodes according to *"preferential attachment"* principle.

At a given time step, the ***probability p*** *of creating an edge between an existing node u and the new node is:* $p = [(degree(u) + 1)/(|E| + |V|)]$ *where |V| is a set of nodes and |E| is the set of edges.*

1.1. Implementation

Nodes forming the network will be called ***turtles*** in NetLogo, and each link created has an activation-probability *(set later).*

The first procedure is adding a node *(add-node)*

```
to add-node
  ask links [ set color gray ]
  make-node [end1] of one-of links
  layout
end

to make-node [old-node]
  create-turtles 1
[
    set color red
    if old-node != nobody
    [
        create-link-to old-node [
        set color green
        set weight 0
        ]

        create-link-from old-node [
          set color green
          set weight 0
         ]

        move-to old-node
        fd 8
    ]
 ]
end
```

Code-Snippet 1: Procedures for creating the network

The ***make-node*** procedure takes one input – ***turtle*** to make links with which is randomly selected from all the links present in the network. This selection takes care of the preferential

attachment equation as the node with more links has a higher chance of getting selected to make another link.

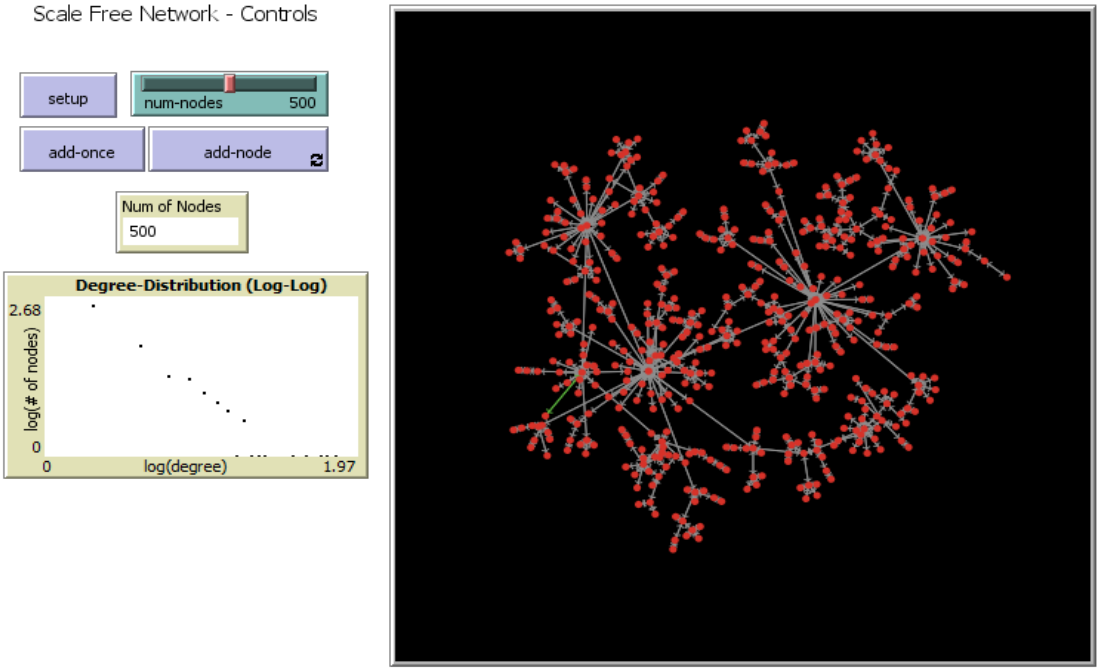


Figure 1 : Scale Free Network creating using Preferential Attachment Mode

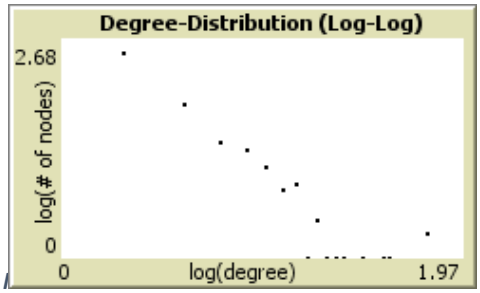


Figure 2: Degree Distribution (LOG-LOG) of the Network

We can also see from the graph plotted in *Figure 2* that the network follows a *power-law distribution.*

2. SETTING UP INFORMATION CASCADE MODEL

2.1. Selecting starting nodes

From the network we randomly select the number of nodes selected in the slider – num-of-starting-nodes and set their color yellow - indicating they are infected.

```
to setup-ICM
  set st_turt n-of starting-nodes-with-info turtles
  ask st_turt [
    set color yellow
  ]
end
```

Code Snippet 2: Selecting starting nodes.

2.2. Setting Activation Probabilities

Activation probabilities $p_{v,w}$ is the probability of a directed link from $v \rightarrow w$ to get activated, i.e. if v is infected then $p_{v,w}$ is the probability of infecting w too.

Now say a node v has $n\ out - links$ to other nodes, then the probability of each link to get activated is $p_{v,i}\ \ for\ all\ i : 0 \leq i \leq n$ and

$$\sum_{i=0}^{n} p_{v,i} = 1 \quad (eq\ 1)$$

We do this in NetLogo by for every node's (turtles) out-links, assigning a random weight from $1\ to\ 100$ and then dividing the weight by the sum of all weights from that node. This ensures that $eq\ 1$ is satisfied.

```
to set-act-prob
  ask turtles [
    let summ 0
    ask my-out-links [
      set weight 1 + random 100
      set summ summ + weight
    ]
    ask my-out-links [
      set weight weight / summ
    ]
  ]
end
```

Code Snippet 3: Setting activation probabilities according to eq:1.

2.3. Cascading Effect

For cascading the information, we perform the following steps:

1. Start with random set of nodes (2.1)
2. Generate a random number from 0 to 1 for each node's out links.
3. If the generated number is less than the activation probability (2.2) the other node of that is also infected (yellow).

$$if\ p < p_{v,w} : w\ gets\ infected\ \ (eq\ 2) \quad Ref\ 1.$$

4. Make a new set of newly infected nodes and continue from step 2 again till there no new nodes.

```
to spread
  let new []
  ask st_turt[
    ask my-out-links[

      let y 0
      let p  random-float 1
      if p < weight [
        ask end2[
          set y color
        ]
        if y != yellow[
         set new lput end2 new
         ask end2[
             set color yellow
           ]
        ]
      ]

    ]
  ]
  set st_turt turtle-set new
  print st_turt

  if length new = 0 [
    let c 0
    ask turtles[
      if color = yellow[
        set c c + 1
      ]
    ]
    user-message (word "Maximum Spread Achieved with " c " nodes infected")

    reset-ticks
    stop
  ]
  update-plots
  tick
end
```

Code Snippet 4: Code for spreading the information in the network.

Following buttons are provided for controlling the above set function and variables.

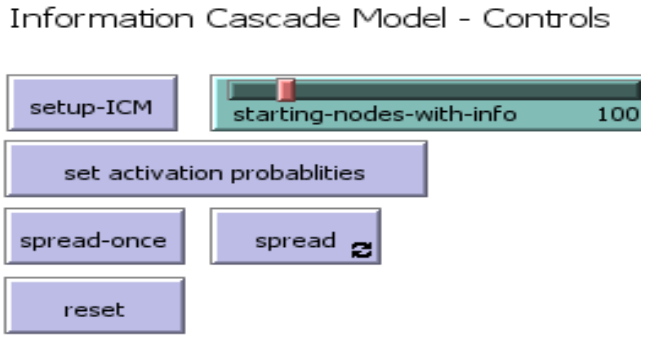


Figure 3: Controls for ICM

3. Simulation – Starting with different set of starting nodes.

3.1. Test 1: $\boldsymbol{count - start - nodes} = 25$

We start with initially 25 nodes infected.

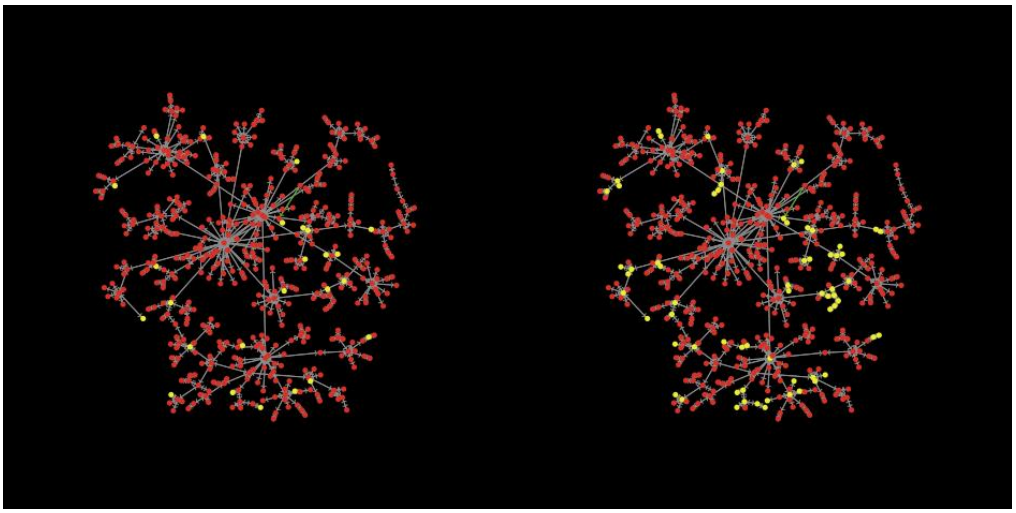


Figure 4: 1st : Initial Stage (25 infected) and 2nd : Maximum spread (66 infected)

Only 66 nodes get infected in a time span of 7 steps

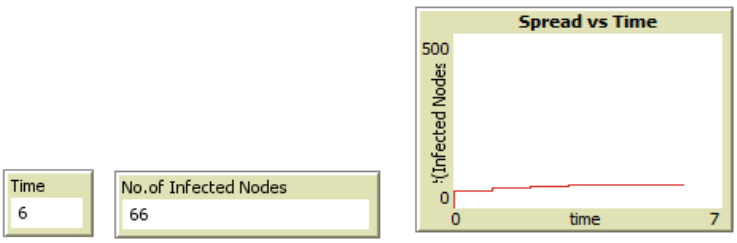


Figure 5: Monitors and Graph showing the spread with 25 nodes initially infected

3.2. Test 2: $\boldsymbol{count - start - nodes} = 25$ (again).

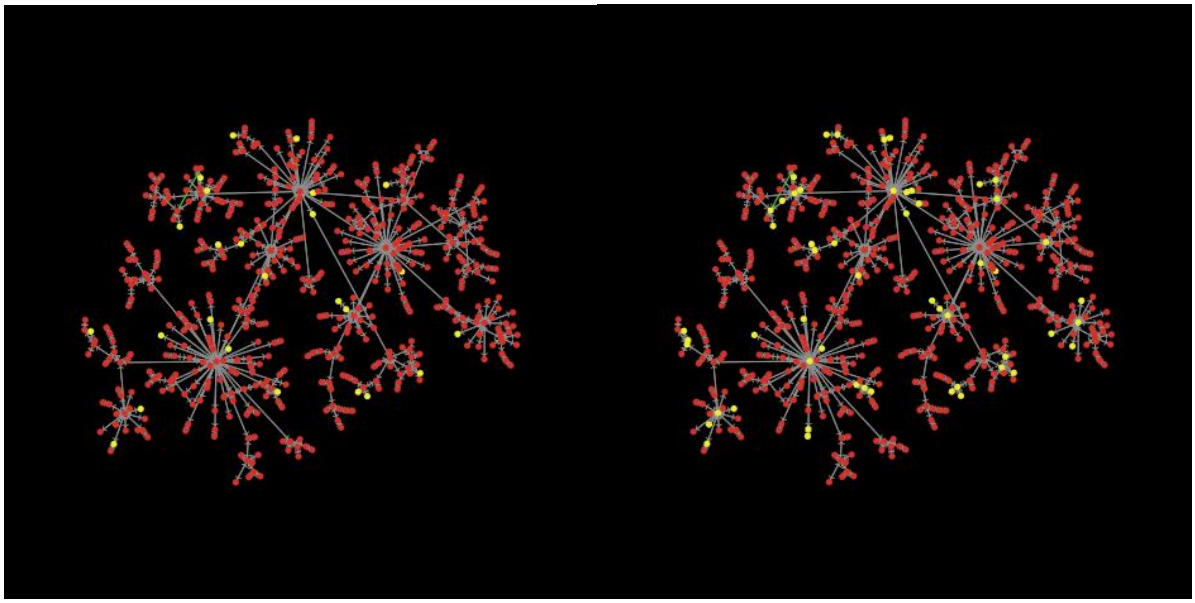


Figure 6: 1st: Initial Stage (25 infected) *2nd : Maximum spread (54 infected)*

Only **54 nodes get infected in a time span of 3 steps**

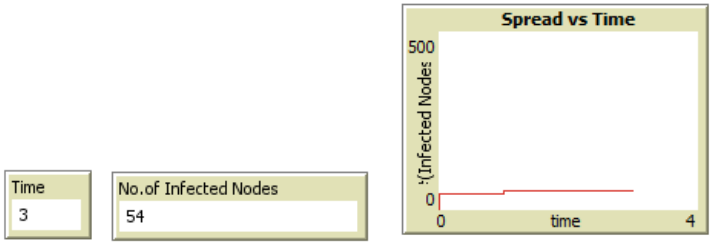


Figure 7: Monitors and Graph showing the spread with 25 nodes initially infected

3.3. Test 3: $\boldsymbol{count - start - nodes} = 75$

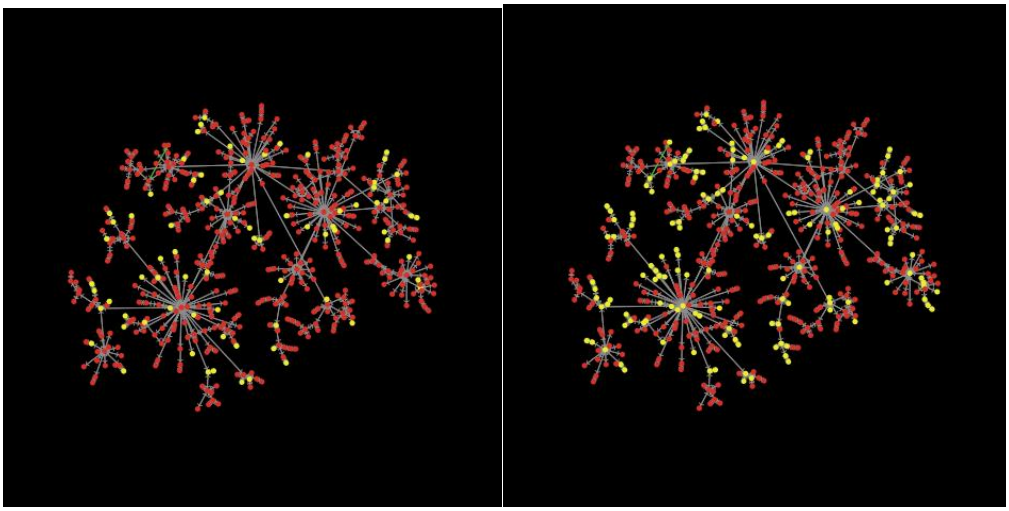


Figure 8: 1st: Initial Stage (75 infected) 2nd : Maximum spread (147 infected)

Only **147 nodes get infected in a time span of 3 steps.**

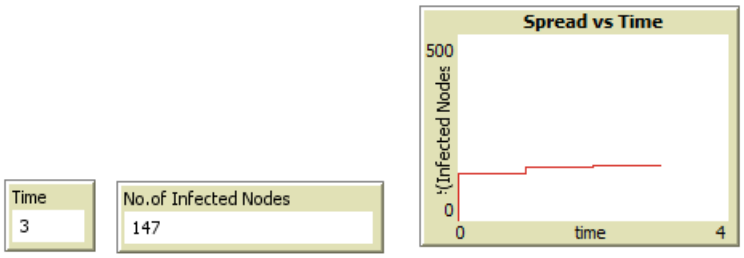


Figure 9: Monitors and Graph showing the spread with 75 nodes initially infected

3.4. Test 4: $count - start - nodes = 150$

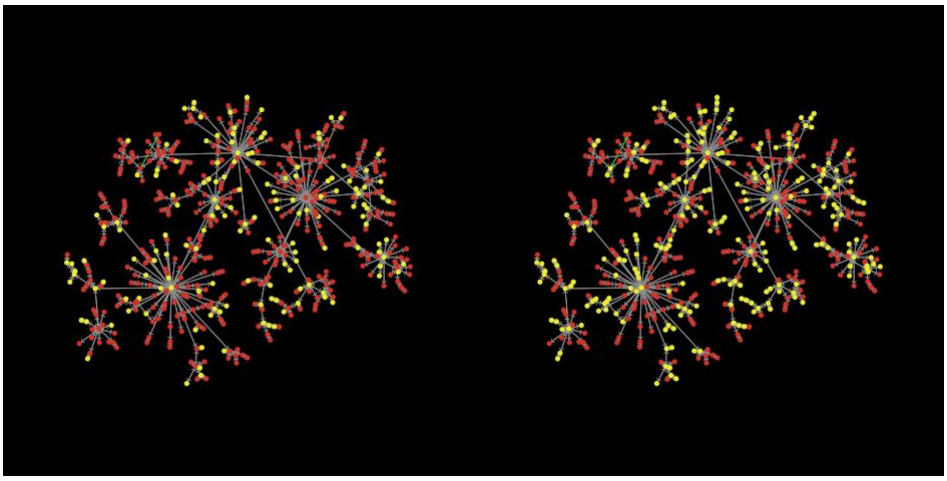


Figure 10: 1st: Initial Stage (150 infected) 2nd : Maximum spread (293 infected)

Only **233 nodes get infected in a time span of 3 steps.**

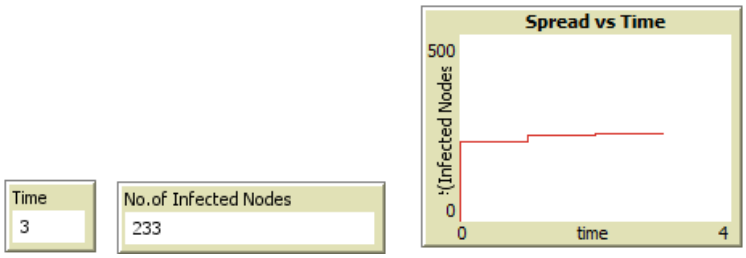


Figure 11: Monitors and Graph showing the spread with 150 nodes initially infected

Following table shows all the tests conducted. For different set and count of initial set of nodes we have tabled down the results.

Table 1 : Summary of Simulations

#(Initial Set)	Sim 1		Sim 2		Sim 3		Sim4		Avg Infected	Avg Time
	Max Infected	Steps	Max Infected	Steps	Max Infected	Steps	Max Infected	Steps		
25	66	3	60	3	64	5	63	2	63.25	3.25
75	147	3	138	4	145	3	158	4	147	3.5
150	233	3	241	3	235	2	242	3	237	2.75
200	292	4	299	3	300	3	297	3	297	3.25

4. OBSERVATION

Very small number of nodes are getting infected through ICM as:

$$\#outlinks(node) = n$$

$$for\ each\ outlink - p_{v,w} \in [0,1]$$

$$Since, \sum p = 1, these\ probablities\ get\ distributed\ among\ n\ links$$

$$So, \boldsymbol{larger\ the\ num\ of\ outlinks\ from\ a\ node, less\ activation\ probablity\ each\ link\ gets.}$$

And since the **EQ 2** states that $\mathbf{p} < \mathbf{p}_{\mathbf{v,w}}$, so for an **influential node** the activation probabilities are already very less so **chance of spreading is also less.**

But if we change the equation to $p \geq p_{v,w}$ results change, information spreads very fast.

Below is a simulation with initially **25 nodes**, and we can see **375 nodes are infected**, using the changed equation $p \geq p_{v,w}$.

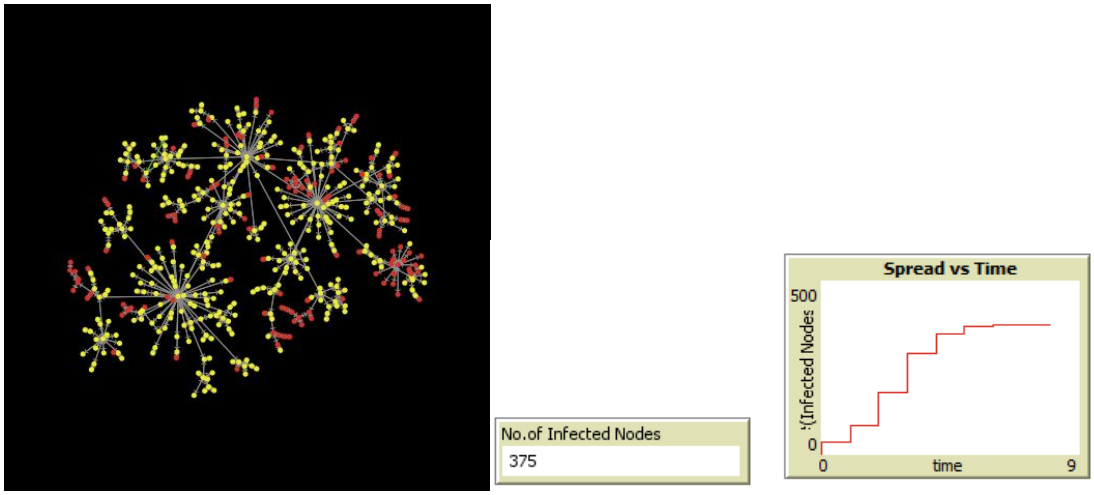


Figure 12 Changing a single operator (>) gives very different results.

The NetLogo file for this project can be found at this Google Drive link
https://drive.google.com/open?id=1ol5tSw7ZbCSDYg6GLM3Lqakr6A-LFKEq

5. REFERENCES (for 2nd Problem)

[1] Social Media Mining *Reza Zafarani Mohammad Ali Abbasi Huan Liu*
[2] Preferential Attachment https://www.sciencedirect.com/topics/computer-science/preferential-attachment.