test_on_networkx

November 3, 2018

```
In [96]: #You are using version 2.0 of networkx. Which changed from using a dict for G.degree(
         import networkx as nx
         import os
         import numpy as np
         import pandas as pd
         %matplotlib inline
         import matplotlib.pyplot as plt
         import time
         #set current directory
         os.chdir("/Data")
```

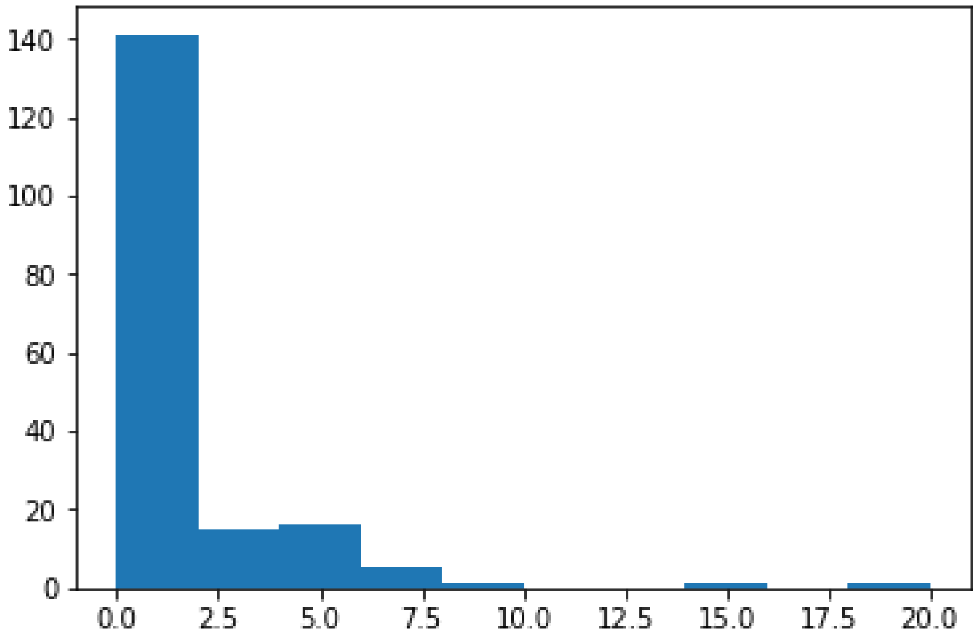
```
In [97]: print(ctime())
         problock1=pd.read_csv("processed_400600.csv")
         problock2=pd.read_csv("processed_398049.csv")
         problock3=pd.read_csv("processed_400051.csv")
         problock_Feb=pd.concat([problock2,problock3,problock1])
         problock_Jan=pd.read_csv("processed_to")
         print(ctime())
```

```
In [98]: problock_Jan=problock_Jan[["fromAddress","ToAddress"]]
         problock_Feb=problock_Feb[["fromAddress","ToAddress"]]
```

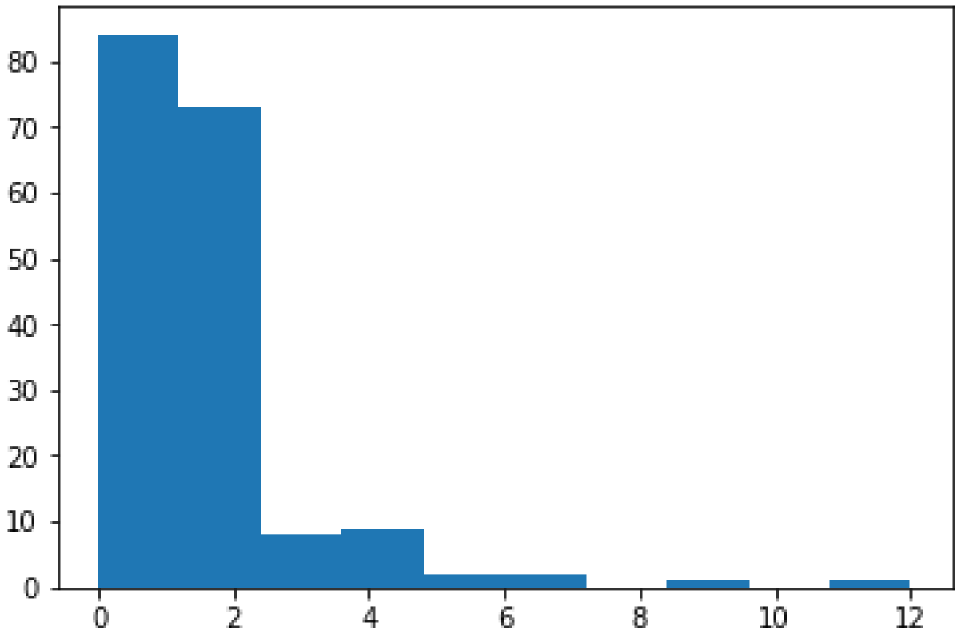
```
Out[98]:                           fromAddress                           ToAddress  \
         0  1KbqoXZcgMDoU7CY8k2hqwrVRGDEqydWM3   1szTHbdCFLY9WPyoMyH5jtKW4pepR6DLx
         1  1KbqoXZcgMDoU7CY8k2hqwrVRGDEqydWM3  17aaSV8mHKgc9Q9Se4Ayr9NpVHAng9sQNp
         2  1Q4LDKaENhAe7SGfMMo1VdWZQGEFANS7JJ  1LpoamuxjMnq8cT5BcHwedPFjVBeW4sGYb
         3  1Q4LDKaENhAe7SGfMMo1VdWZQGEFANS7JJ  1HGJezuLBYr5Tr1zfpbQMUckLPjsyCkZdj
         4  1JpooykQUintEJmi5BkR3SoczFwuYQ33cw  1JpqZ94aYBLZnEHD27m6Ahgahks29orpuz

                 aggcoin
         0  1.398561e+12
         1  1.398561e+12
         2  1.152116e+10
         3  1.152116e+10
         4  6.338635e+09
```

```
In [100]: print(problock_Jan.shape)
          print(problock_Feb.shape)
```

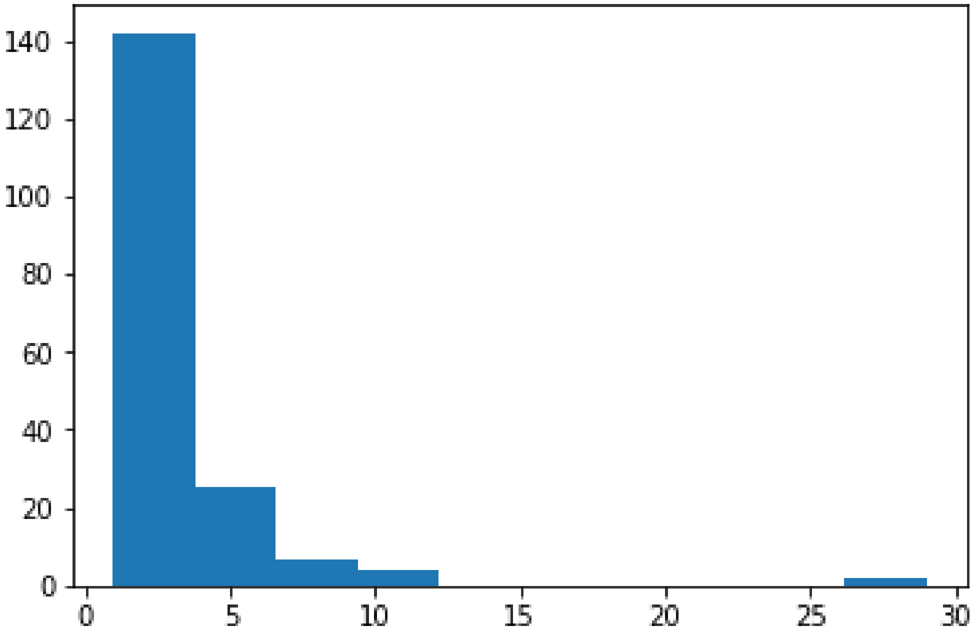
```
Out[100]: (288, 3)

In [65]: #create Jan graph
         edges_Jan = [tuple(x) for x in problock_Jan.to_records(index=False)]
         df1 = problock_Jan.stack().reset_index(drop=True, level=1).reset_index(name='Nodes')
         nodes0 = df1['Nodes'].tolist()
         nodes_Jan = list(set(nodes0))
         G_Jan = nx.DiGraph()
         #remove self-loop
         G_Jan.add_nodes_from(nodes_Jan)
         G_Jan.add_edges_from(edges_Jan)

In [66]: #create Feb graph
         edges_Feb = [tuple(x) for x in problock_Feb.to_records(index=False)]
         df2 = problock_Feb.stack().reset_index(drop=True, level=1).reset_index(name='Nodes')
         nodes1 = df2['Nodes'].tolist()
         nodes_Feb = list(set(nodes1))
         G_Feb = nx.DiGraph()
         #remove self-loop
         G_Feb.add_nodes_from(nodes_Feb)
         G_Feb.add_edges_from(edges_Feb)

In [68]: #plot(g_problock,layout=layout_with_gem,main="gem layout")    10:56-11:36 It might be
         #plt.plot(g_problock,layout=nx.kamada_kawai_layout(g_problock))
         #nx.draw(G,node_color="skyblue", pos=nx.fruchterman_reingold_layout(G))
         #plt.title("fruchterman_reingold")
         #nx.draw(G,node_color="skyblue", pos=nx.kamada_kawai_layout(G))
         #plt.title("kamada_kawai")
         #Layout algorithm:
         #circular_layout(g_problock)
         #kamada_kawai_layout(g_problock)

In [77]: in_degree_sequence = sorted([d for n, d in G.in_degree()], reverse=True)
         plt.hist(in_degree_sequence)

Out[77]: (array([141.,  15.,  16.,   5.,   1.,   0.,   0.,   1.,   0.,   1.]),
          array([ 0.,  2.,  4.,  6.,  8., 10., 12., 14., 16., 18., 20.]),
          <a list of 10 Patch objects>)
```



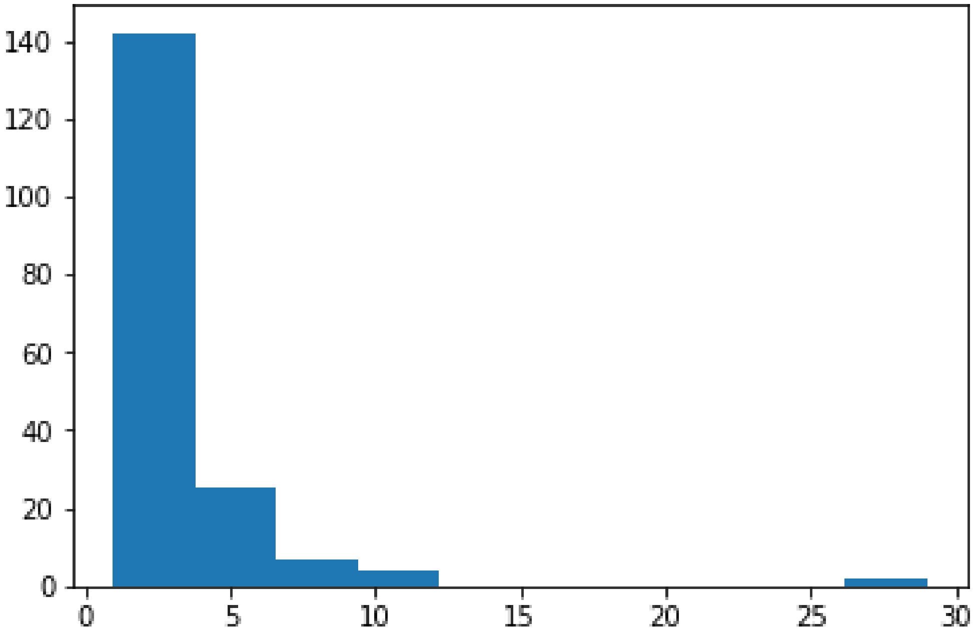
```
In [78]: out_degree_sequence=sorted([d for n, d in G.out_degree()], reverse=True)
         plt.hist(out_degree_sequence)

Out[78]: (array([84., 73.,  8.,  9.,  2.,  2.,  0.,  1.,  0.,  1.]),
          array([ 0. ,  1.2,  2.4,  3.6,  4.8,  6. ,  7.2,  8.4,  9.6, 10.8, 12. ]),
          <a list of 10 Patch objects>)
```



```
In [79]: out_degree_sequence=sorted([d for n, d in G.degree()], reverse=True)
         plt.hist(out_degree_sequence)

Out[79]: (array([142.,  25.,   7.,   4.,   0.,   0.,   0.,   0.,   0.,   2.]),
          array([ 1. ,  3.8,  6.6,  9.4, 12.2, 15. , 17.8, 20.6, 23.4, 26.2, 29. ]),
          <a list of 10 Patch objects>)
```

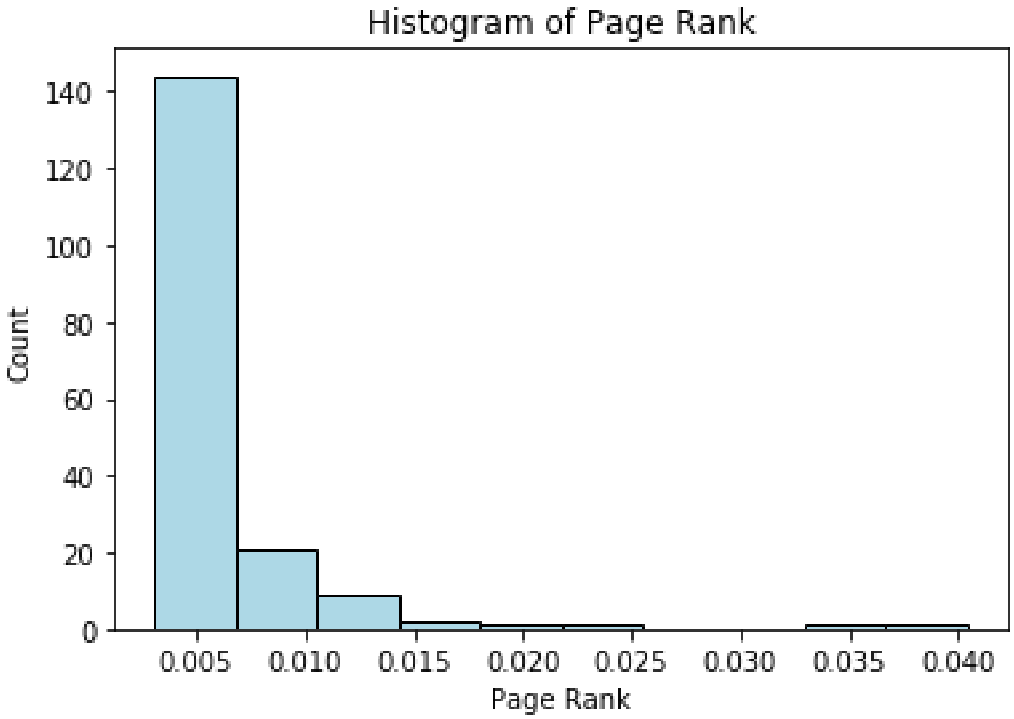


```
In [80]: #in-degree, out-degree and total degree
         #nx.degree(g_problock)
         #degree_sequence=sorted(nx.degree(g_problock).values(),reverse=True)
         #g_problock.out_degree(from_list)
         degrees = [val for (node, val) in G.degree()]
         degree_sequence=sorted(degrees)
         plt.hist(degree_sequence)
         degree_sequence[-10:-1]

Out[80]: [7, 7, 9, 9, 10, 11, 12, 12, 27]
```



```
In [81]: degree_sequence
         sum([val == 1 for val in degree_sequence])

Out[81]: 63

In [82]: #compute pagerank of nodes in the graph
         #nx.pagerank(g_problock)
         #google_matrix(g_problock)
         #The eigenvector calculation uses power iteration with a SciPy sparse matrix represen
         #nx.pagerank_scipy(g_problock)

In [83]: page_btc=nx.pagerank_scipy(G)       #42-43
         pagerank=sorted(page_btc.values(),reverse=True)
         pagerank=list(pagerank)
         pagerank[1:10]

Out[83]: [0.036258930422990214,
          0.022316157043439595,
          0.019694000254336298,
          0.017385470253844286,
          0.01596517251357888,
          0.013894205709246241,
          0.01325963999715457,
          0.013013586341061052,
          0.01287968810961438]
```

```
In [84]: plt.hist(pagerank,color = "lightblue", ec="black")
         plt.title("Histogram of Page Rank")
         plt.ylabel("Count")
         plt.xlabel("Page Rank")

Out[84]: Text(0.5,0,'Page Rank')
```



```
In [85]: density = nx.density(G)
         print(nx.info(G))
         print("Network density:", density)

Name:
Type: DiGraph
Number of nodes: 180
Number of edges: 251
Average in degree:   1.3944
Average out degree:   1.3944
('Network density:', 0.007790192427063935)
```

```
In [86]: #nx.shortest_path(G)
```

ego graph

#RuntimeWarning: invalid value encountered in sqrt distance = np.sqrt((delta**2).sum(axis=0)) 1:46-

1 find node with largest degree

node_and_degree=G.degree() #(largest_hub,degree)=sorted(node_and_degree.items(),key=itemgetter(1))[-1] cannot be used in degree view largest_degree=max([val for (node, val) in G.degree()]) largest_hub=[node for (node, val) in G.degree() if val==largest_degree] # Create ego graph of main hub hub_ego=nx.ego_graph(G,largest_hub[0]) # Draw graph pos=nx.spring_layout(hub_ego) nx.draw(hub_ego,pos,node_color='b',node_size=50,with_labels=False) # Draw ego as large and red nx.draw_networkx_nodes(hub_ego,pos,nodelist=[largest_hub[0]],node_size=300,node_color='r') nx.draw_networkx_nodes(hub_ego,pos,['1dice97ECuByXAvqXpaYzSaQuPVvrtmz6'],node_size=300,node_color #plt.savefig('ego_graph.png') plt.show()

test_all = [nx.single_source_shortest_path_length(G,key) for key in nodes] new_list = [(val) for dic in test_all for key,val in dic.items()] plt.hist(new_list, bins = 30) plt.xlabel('All shortest paths') plt.ylabel('Frequency')

Now ego-centric analysis - select an arbitray node- 1KXZ, go to order 3.

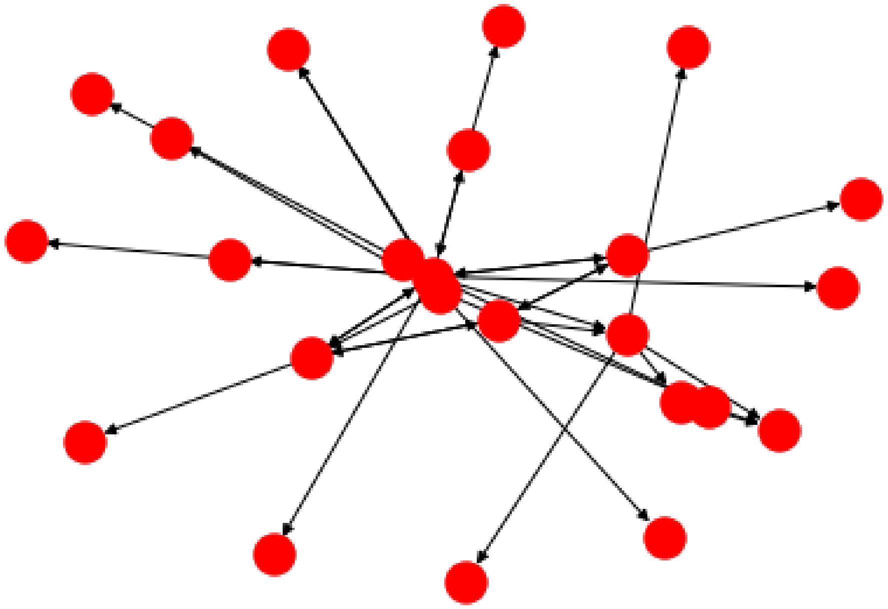
```
ego_russian_1<-make_ego_graph(g_problock,order=1,"1LQv8aKtQoiY5M5zkaG8RWL7LMwNzVaVqR")
plot(ego_russian_1[[1]],main="Russian Order 2 Ego")
ego_russian_2<-make_ego_graph(g_problock,order=2,"1LQv8aKtQoiY5M5zkaG8RWL7LMwNzVaVqR")
plot(ego_russian_2[[1]],main="Russian 1LQv8aKtQoi..VaVqR, Order 2 Ego")
ego_russian_3<-make_ego_graph(g_problock,order=3,"1LQv8aKtQoiY5M5zkaG8RWL7LMwNzVaVqR")
plot(ego_russian_3[[1]],main="Russian 1LQv8aKtQoi..VaVqR Order 3 Ego")
plot(ego_russian_3[[1]],vertex.label=NA,main="Russian 1LQv8aKtQoi..VaVqR Order 3 Ego")
```

Now compute ego with loop and mple

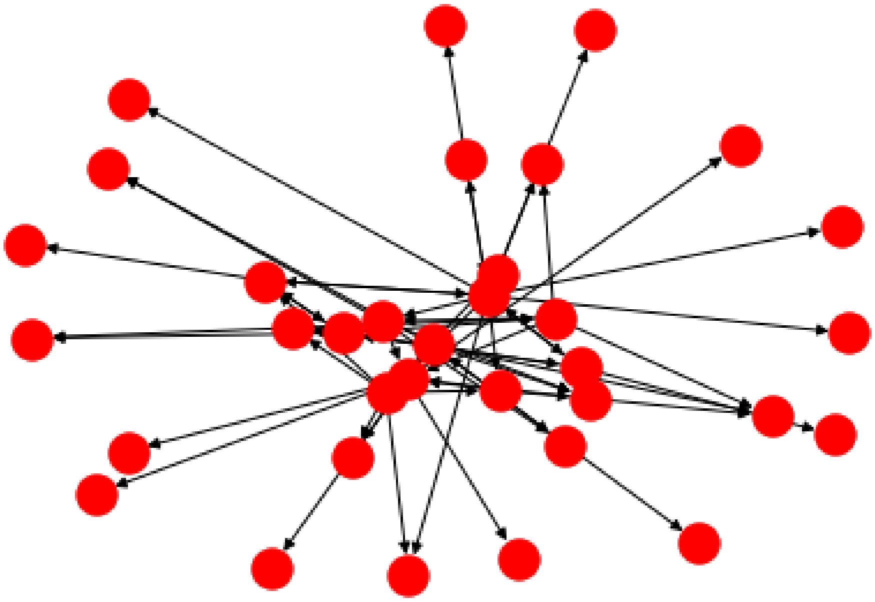
```
ego_russian_1_lp_mple<-make_ego_graph(g_problock_lp_mple,order=1,"1LQv8aKtQoiY5M5zkaG8RWL7LMwN
plot(ego_russian_1_lp_mple[[1]],main="Russian Order 2 Ego")
ego_russian_2_lp_mple<-make_ego_graph(g_problock_lp_mple,order=2,"1LQv8aKtQoiY5M5zkaG8RWL7LMwN
plot(ego_russian_2_lp_mple[[1]],main="Russian 1LQv8aKtQoi..VaVqR, Order 2 Ego",layout=layout_w
ego_russian_3_lp_mple<-make_ego_graph(g_problock_lp_mple,order=3,"1LQv8aKtQoiY5M5zkaG8RWL7LMwN
plot(ego_russian_3_lp_mple[[1]],main="Russian 1LQv8aKtQoi..VaVqR Order 3 Ego")
plot(ego_russian_3_lp_mple[[1]],vertex.label=NA,main="Russian 1LQv8aKtQoi..VaVqR Order 3 Ego",
plot(ego_russian_3_lp_mple[[1]],vertex.label=NA,main="Russian 1LQv8aKtQoi..VaVqR Order 3 Ego",
save.image("/Users/siddharthadalal/Dropbox/Columbia/Columbia_Courses/APAN/APAN_Blockchain_Cour
```

ego_graph(G,"1LQv8aKtQoiY5M5zkaG8RWL7LMwNzVaVqR", radius=1) hub_ego=nx.ego_graph(G,"1LQv8aKtQoiY5M5zkaG8RWL7LMwNzVaVqR", radius=1) plt.plot(hub_ego[0],main="Russian Order 2 Ego")

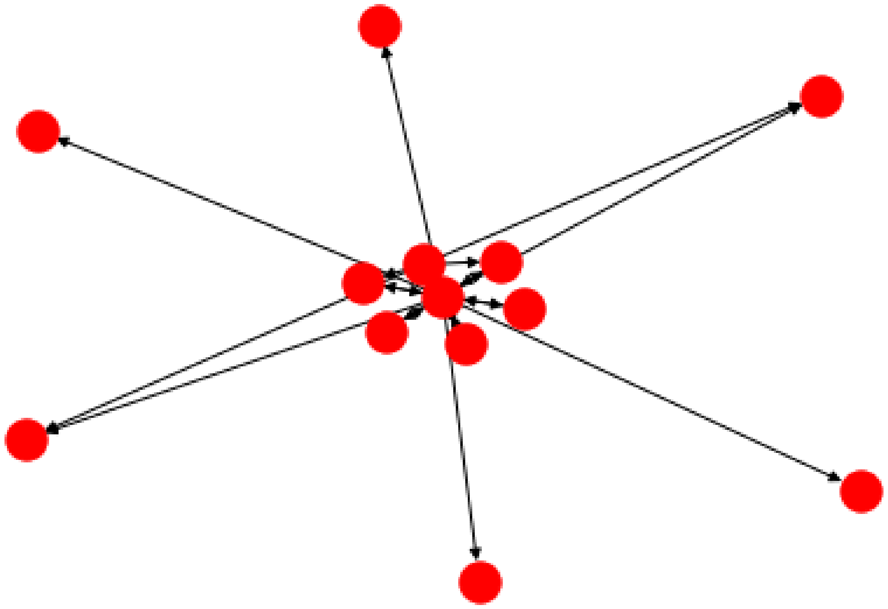
```
In [88]: # create an ego-graph for some node
         node ="1LQv8aKtQoiY5M5zkaG8RWL7LMwNzVaVqR"
         ego_graph = nx.ego_graph(G,node, radius=2)
         # plot to check
         nx.draw(ego_graph); plt.show()
```



In [89]:
```
# create an ego-graph for some node
node = "1LQv8aKtQoiY5M5zkaG8RWL7LMwNzVaVqR"
ego_graph = nx.ego_graph(G,node, radius=3)

# plot to check
nx.draw(ego_graph); plt.show()
```

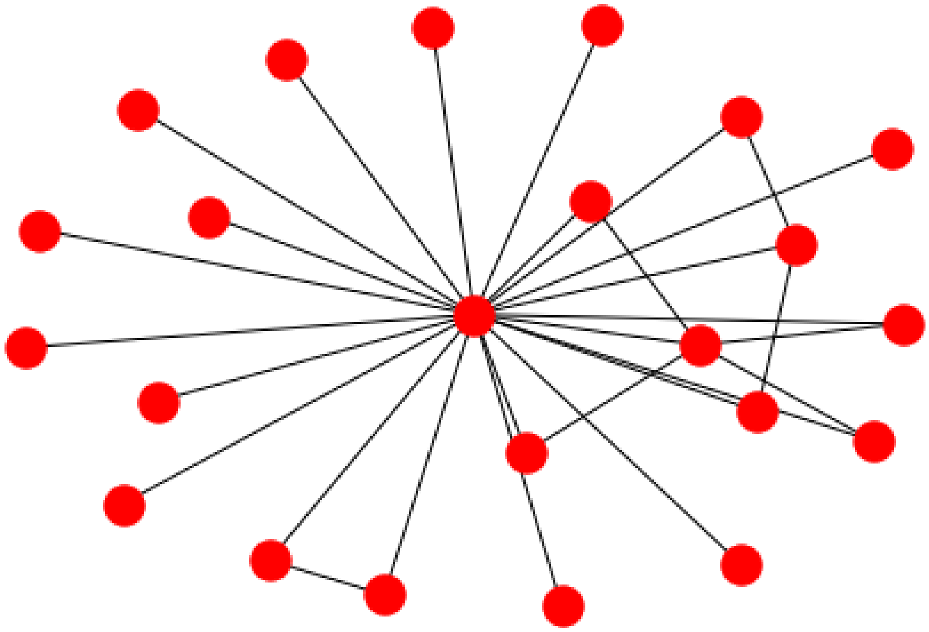


```
In [90]: # create an ego-graph for some node
         node = "1LQv8aKtQoiY5M5zkaG8RWL7LMwNzVaVqR"
         ego_graph = nx.ego_graph(G,node, radius=1)

         # plot to check
         nx.draw(ego_graph); plt.show()
```

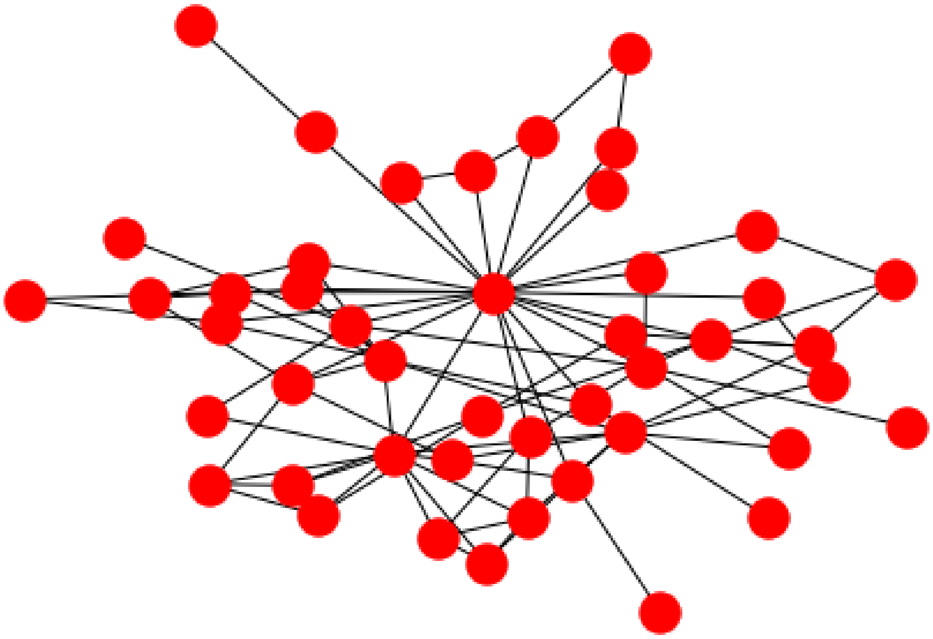


#find node with largest degree largest_degree=max([val for (node, val) in G.degree()]) largest_hub=[node for (node, val) in G.degree() if val==largest_degree] #largest_hub, degree) = sorted(node_and_degree, reverse=True)[-1] #Create ego graph of main hub hub_ego = nx.ego_graph(G, largest_hub[0]) #Draw graph pos = nx.spring_layout(hub_ego) nx.draw(hub_ego, pos, node_color='b', node_size=50, with_labels=False) #Draw ego as large and red nx.draw_networkx_nodes(hub_ego, pos, nodelist=[largest_hub[0]], node_size=300, node_color='r') plt.show()

```
In [92]: # create an ego-graph for some node
         node = "1LQv8aKtQoiY5M5zkaG8RWL7LMwNzVaVqR"
         ego_graph = nx.ego_graph(G1,node, radius=1)

         # plot to check
         nx.draw(ego_graph); plt.show()
```



```
In [94]: # create an ego-graph for some node
         node = "1LQv8aKtQoiY5M5zkaG8RWL7LMwNzVaVqR"
         ego_graph = nx.ego_graph(G1,node, radius=2)

         # plot to check
         nx.draw(ego_graph); plt.show()
```



```
In [95]: # create an ego-graph for some node
         node = "1LQv8aKtQoiY5M5zkaG8RWL7LMwNzVaVqR"
         ego_graph = nx.ego_graph(G1,node, radius=3)

         # plot to check
         nx.draw(ego_graph); plt.show()
```