ДТЗ Ботуева ПМ18-4 Вариант 6

Задача #6 (3.3)

Реализовать функцию пересвязывания связей в произвольной сети с заданной вероятностью p. Вероятность p разыгрывается для каждой связи, если принято решение о пересвязывании, один из двух узлов данной связи меняется на произвольный узел сети (нужна проверка на попытку создать повторную связь). Сеть для пересвязывания и p - параметры функции, пересвязанная сеть - возвращаемое значение.

Выбрать 2 реальные сети сравнимого размера с большой и маленькой среденей длиной пути. С помощью 3.3.5. провести для этих 2 сетей последовательность пересвязываний (не менее 15) с p возрастающим в геометрической прогерссии. Реализовать последовательность пересвязываний так, чтобы следующее пересвязывание (с большей вероятностью p) включало все передыдущие резульаты пересвязывания. Для каждой сети построить на одном графике относительные изменения коэффициента кластеризации и средней длины пути (график, аналогичный приведенному в лекции).

```
import networkx as nx
import random
import matplotlib.pyplot as plt
import numpy as np
```

```
def HasLink(network, node1, node2):
    ed = list(network.edges())
    if ((node1, node2) in ed) or ((node2, node1) in ed):
        return True
    return False

def Rewire(network, node1, node2, node3):
    network.remove_edge(node2, node3)
    network.add_edge(node1, node2)
    return network

def NumberOfConnectComponents(network):
    V = network.nodes()
    visited = {i: False for i in V}
    count = 0
    for v in V:
        if (visited[v] == False):
            DFSUtil(network, v, visited)
            count += 1
    return count

def DFSUtil(network, v, visited):
    visited[v] = True
    for i in list(network.adj[v]):
        if (not visited[i]):
            DFSUtil(network, i, visited)

def Relinkage(network, p):
    for u, v in network.edges():
        p_ = random.uniform(0, 1)
        if p_ >= p:
            nodeA = random.choice(list(G.nodes()))
            nodeB = random.choice([u, v])
            nodeC = u if v == nodeB else v
            if (nodeB != nodeA) and not (HasLink(network, nodeB, nodeA)) and (G.deg
                Rewire(network, nodeA, nodeB, nodeC)
            if NumberOfConnectComponents(network) > 1:
                Rewire(network, nodeC, nodeB, nodeA)
    return network
```

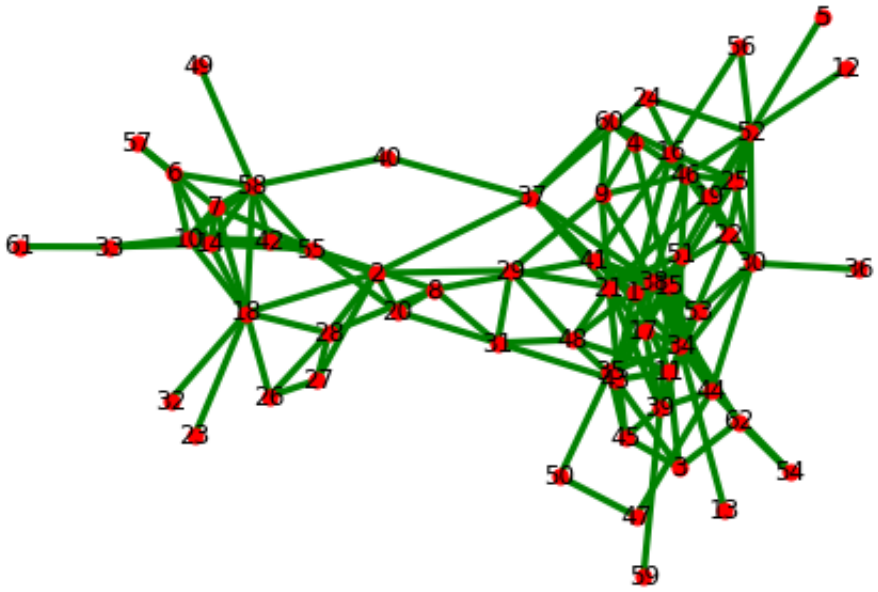
```
G = nx.read_adjlist('out.dolphins')
G.remove_edges_from([('%', 'sym'), ('%', 'unweighted')])
G.remove_nodes_from(['%', 'sym', 'unweighted'])
print(list(G.nodes))
print(list(G.edges))
```

```
['9', '4', '10', '6', '7', '11', '1', '3', '14', '15', '16', '17', '18', '2',
[('9', '4'), ('9', '21'), ('9', '29'), ('9', '38'), ('9', '46'), ('9', '60'),
```

```
clust = nx.average_clustering(G)
av_len = nx.average_shortest_path_length(G)
print('Начальные данные C = {}, l = {}'.format(clust, av_len))
```

```
Начальные данные C = 0.25895824605502027, l = 3.3569539925965097
```

```
#plt.figure(figsize = (8,7))
pos = nx.spring_layout(G, seed = 100)
#degrees = [G.degree(n) for n in G.nodes()]
options = {
    "node_color": 'red',
    "edge_color": "green",
    "width": 3.2,
    "node_size": 50,
    "with_labels": True
}
nx.draw(G,pos, **options)
```



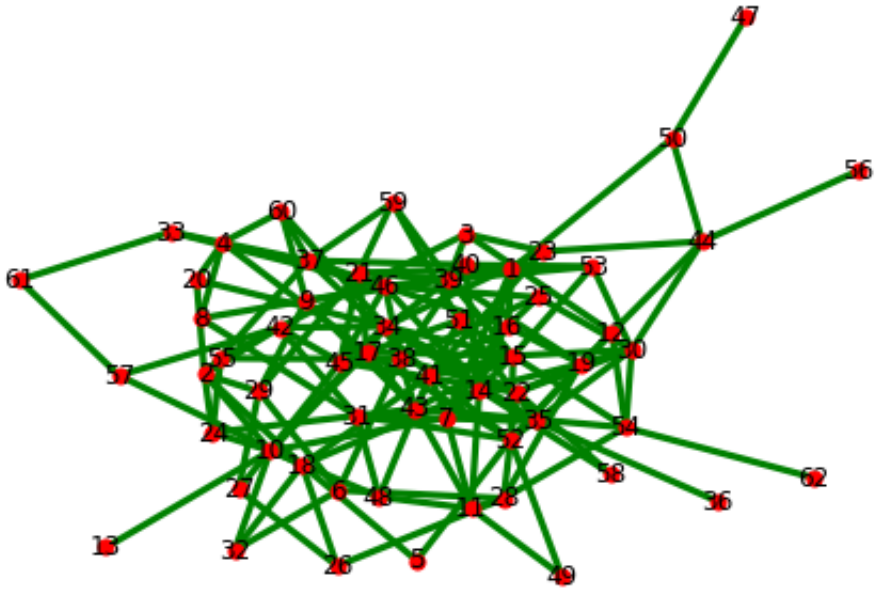
```
Relinkage(G, 0.3)
```

```
<networkx.classes.graph.Graph at 0x7f95a4caf490>
```

```
clust = nx.average_clustering(G)
av_len = nx.average_shortest_path_length(G)
print('Начальные данные C = {}, l = {}'.format(clust, av_len))
```

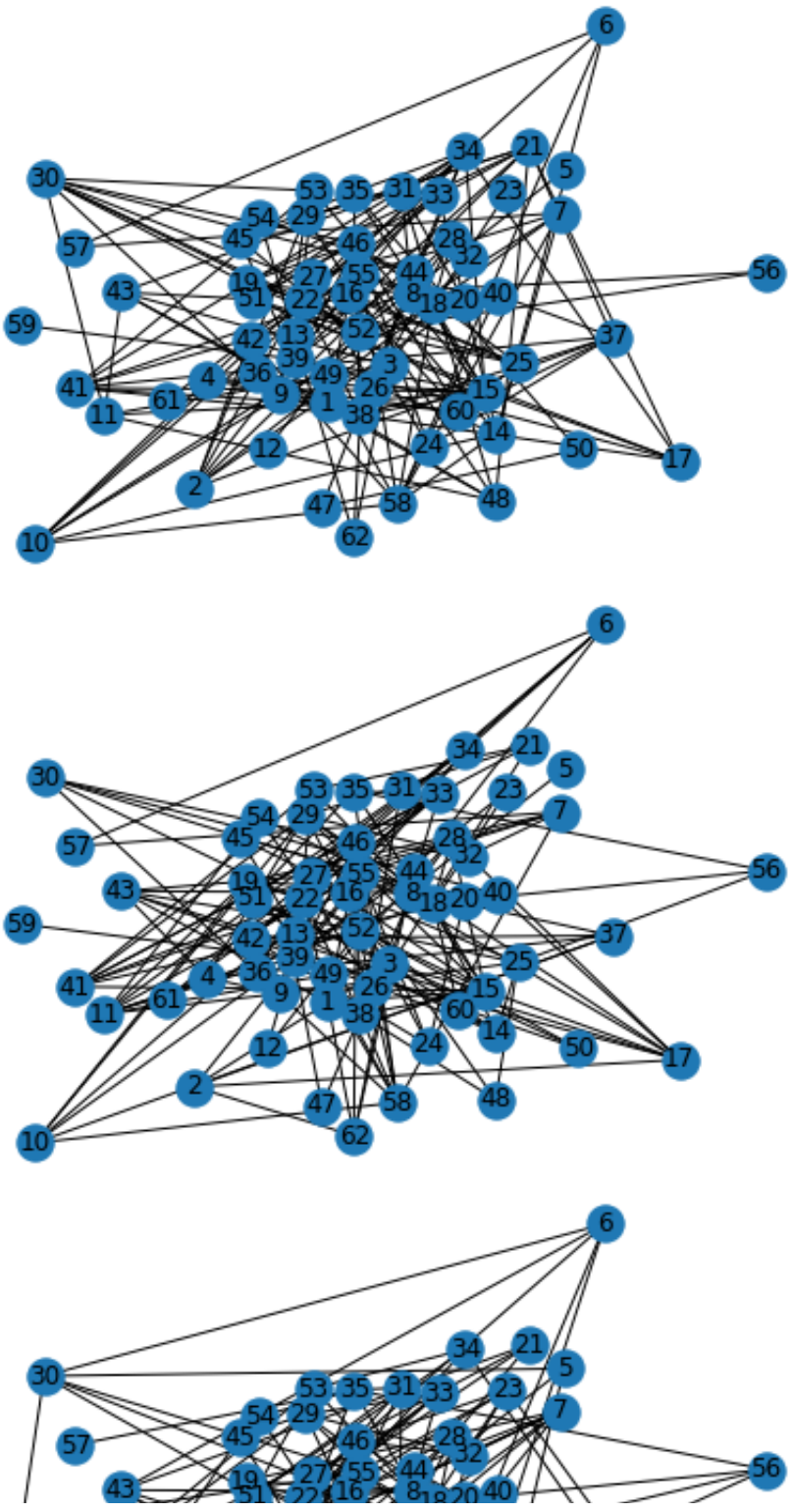
```
Начальные данные C = 0.08535935390774103, l = 2.730830248545743
```

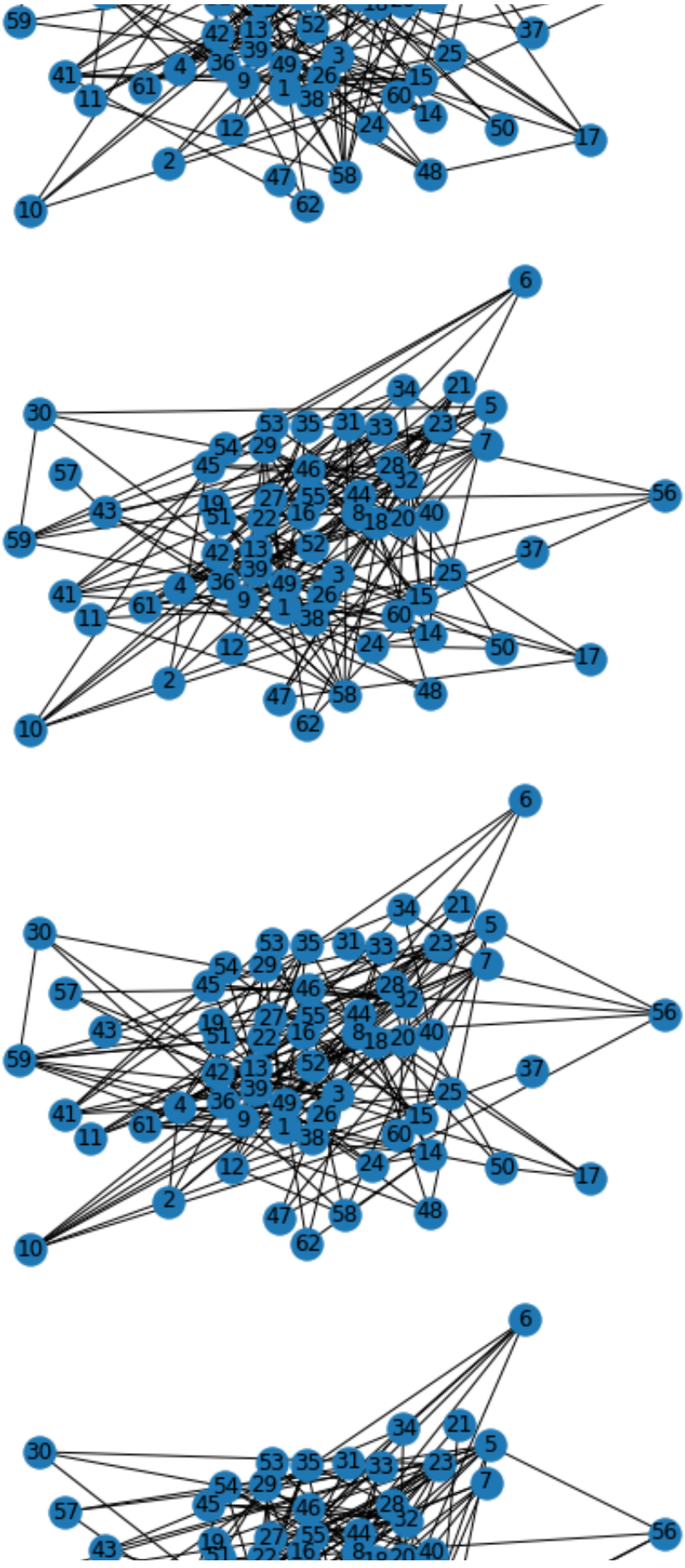
```
#plt.figure(figsize = (8,7))
pos = nx.spring_layout(G, seed = 100)
#degrees = [G.degree(n) for n in G.nodes()]
options = {
    "node_color": 'red',
    "edge_color": "green",
    "width": 3.2,
    "node_size": 50,
    "with_labels": True
}
nx.draw(G, pos, **options)
```

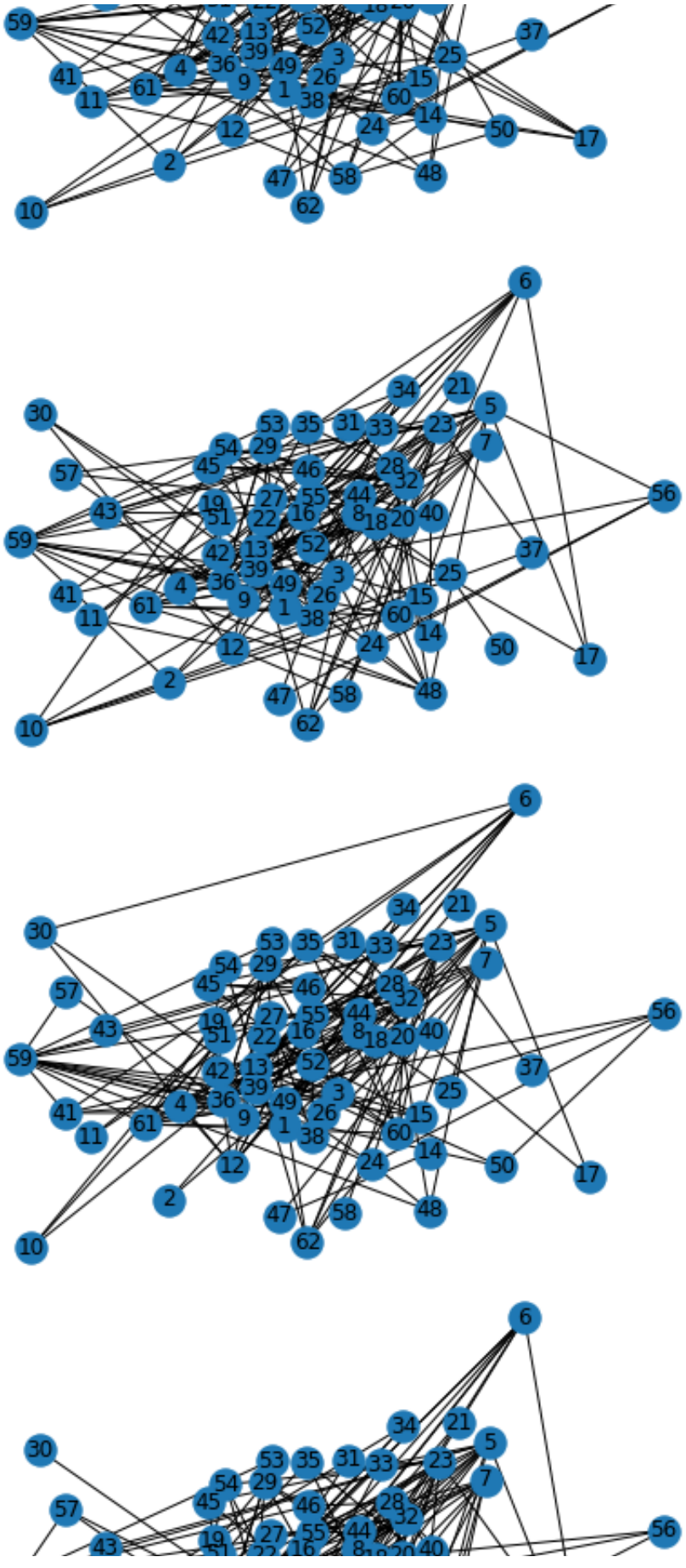


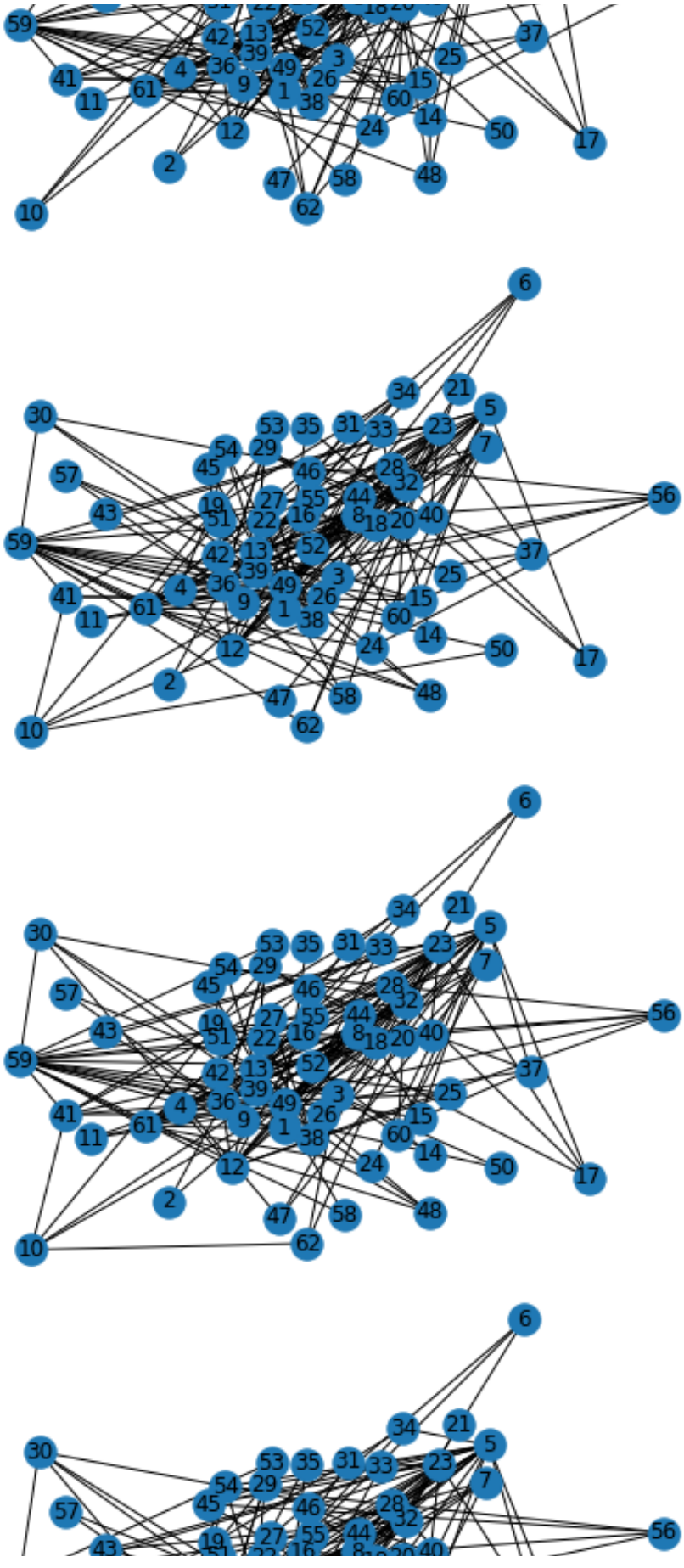
```
G = nx.read_adjlist('out.dolphins')
G.remove_edges_from([('%', 'sym'), ('%', 'unweighted')])
G.remove_nodes_from(['%', 'sym', 'unweighted'])
n = 15
d = 1.165
p = 0.1
clusts, av_lens, ps = [], [], []
H = [G.copy()] * n
for i in range(n):
    fig, ax = plt.subplots(nrows= 1, ncols = 1, figsize = (7,5))
    nx.draw(H[i], pos, with_labels=True)
    Relinkage(H[i], p)
    clusts.append(nx.average_clustering(H[i]))
    av_lens.append(nx.average_shortest_path_length(H[i]))
    # Relinkage(G, p)
```

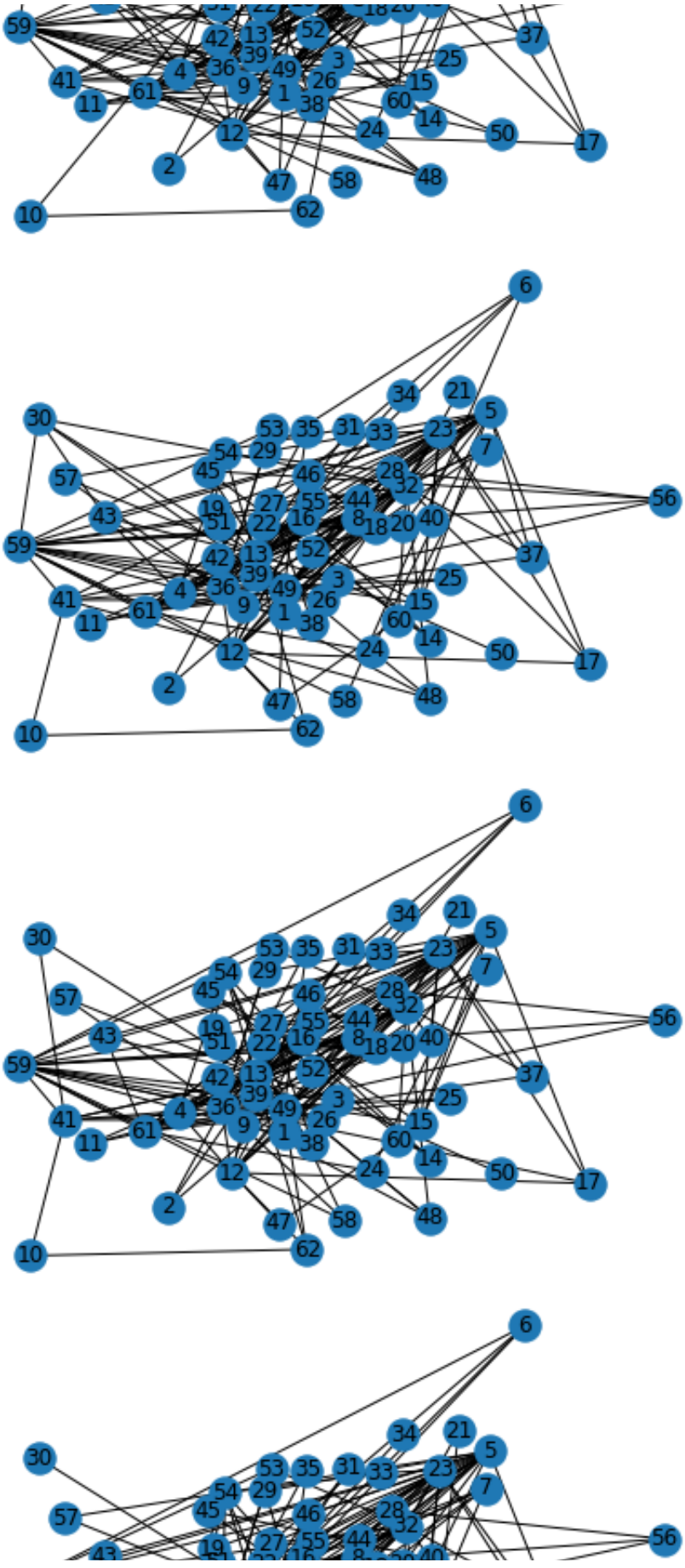
```
    # clusts.append(nx.average_clustering(G))
    # av_lens.append(nx.average_shortest_path_length(G))
    ps.append(p)
    #print(nx.average_clustering(G), len(list(G.edges())))
    p = p * d
ps = np.log(ps)
```

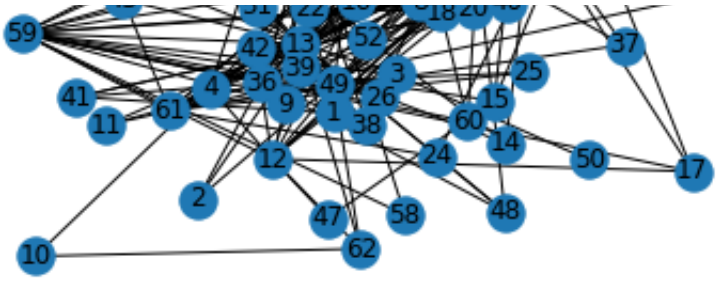



59 42 13 52 37 39 3 25 41 4 36 49 26 15 61 9 1 38 60 11 14 12 24 50 17 2 47 58 48 10 62
6 34 21 5 30 53 35 31 33 23 7 54 29 45 46 28 32 57 56 43 19 27 55 44 51 22 16 8 18 20 40 59 42 13 52 37 39 3 25 41 4 36 49 26 15 61 9 1 38 60 11 14 12 24 50 17 2 47 58 48 10 62
6 34 21 5 30 53 35 31 33 23 7 54 29 45 46 28 32 57 56 43 19 27 55 44 51 22 16 8 18 20 40 59 42 13 52 37 39 3 25 41 4 36 49 26 15 61 9 1 38 60 11 14 12 24 50 17 2 47 58 48 10 62
6 34 21 5 30 53 35 31 33 23 7 54 29 45 46 28 32 57 56 43 19 27 55 44 51 22 16 8 18 20 40

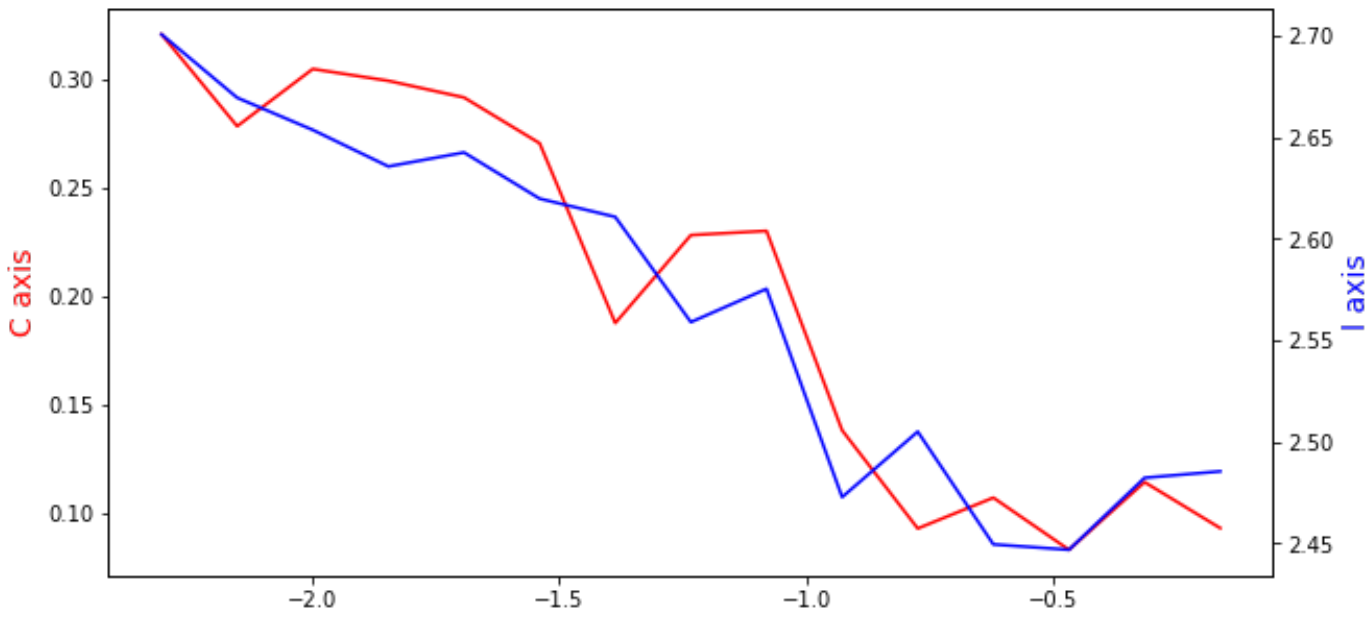

59 42 13 52 37
39 3 25
4 36 49 26 15
41 9 1 38 60
61 14
11 12 24 50
17
2
47 58 48
10 62
6
34 21
30 5
53 35 31 33 23
54 29 7
57 45 46 28 32
56
43 19 27 55 44
51 22 16 8 18 20 40
59 42 13 52 37
39 3 25
4 36 49 26 15
41 9 1 38 60
61 14
11 12 24 50
17
2
47 58 48
10 62
6
34 21
30 5
53 35 31 33 23
54 29 7
57 45 46 28 32
56
43 19 27 55 44
51 22 16 8 18 20 40
59 42 13 52 37
39 3 25
4 36 49 26 15
41 9 1 38 60
61 14
11 12 24 50
17
2
47 58 48
10 62
6
34 21
30 5
53 35 31 33 23
54 29 7
57 45 46 28 32
56
43


59 42 13 52 37 39 3 25 4 36 49 41 9 26 15 61 1 60 11 38 14 12 24 50 17 2 47 58 48 10 62
6 34 21 30 53 35 31 33 23 5 54 29 7 45 46 28 57 32 56 19 27 55 44 43 51 22 16 8 18 20 40 59 42 13 52 37 39 3 25 4 36 49 41 9 26 15 61 1 60 11 38 14 12 24 50 17 2 47 58 48 10 62
6 34 21 30 53 35 31 33 23 5 54 29 7 45 46 28 57 32 56 19 27 55 44 43 51 22 16 8 18 20 40 59 42 13 52 37 39 3 25 4 36 49 41 9 26 15 61 1 60 11 38 14 12 24 50 17 2 47 58 48 10 62
6 34 21 30 53 35 31 33 23 5 54 29 7 45 46 28 57 32 56 19 27 55 44 43 51 22 16 8 18 20 40


59 42 13 52 37
39 3 25
4 36 49 26 15
41 9 1 60
61 38 14
11 12 24 50
17
2 47 58 48
10 62
6
34 21
30 5
53 35 31 33 23 7
54 29
45 46 28 32 56
57 27 55 44
43 19 16 8 20 40
51 22 18
59 42 13 52 37
39 3 25
4 36 49 26 15
41 9 1 60
61 38 14
11 12 24 50
17
2 47 58 48
10 62
6
34 21
30 5
53 35 31 33 23 7
54 29
45 46 28 32 56
57 27 55 44
43 19 16 8 20 40
51 22 18
59 42 13 52 37
39 3 25
4 36 49 26 15
41 9 1 60
61 38 14
11 12 24 50
17
2 47 58 48
10 62
6
34 21
30 5
53 35 31 33 23 7
54 29
45 46 28 32 56
57 27 55 44
43 19




```
fig, ax = plt.subplots(nrows= 1, ncols = 1, figsize = (10,5))
ax.plot(ps, clusts, label = 'C', color = 'red')
ax.set_ylabel("C axis",color="red",fontsize=14)
ax2=ax.twinx()
ax2.plot(ps, av_lens, label = 'l', color = "blue")
ax2.set_ylabel("l axis",color="blue",fontsize=14)
# ax.set_xlabel("p * 100",color="green",fontsize=14)
# ax.set_xlim([0, 1])
# ax.set_ylim([0, 0.2])
# ax2.set_ylim([2.1, 2.7])
plt.show()
```

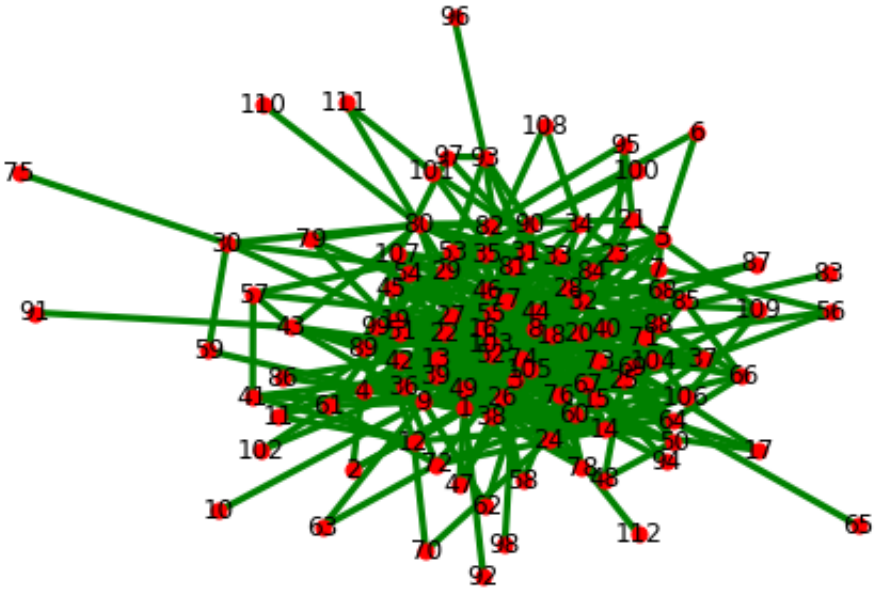


Start coding or generate with AI.

```
G = nx.read_adjlist('out.adjnoun_adjacency_adjacency')
G.remove_edges_from([('%', 'sym'), ('%', 'unweighted')])
G.remove_nodes_from(['%', 'sym', 'unweighted', '425'])
print(list(G.nodes))
print(list(G.edges))
```

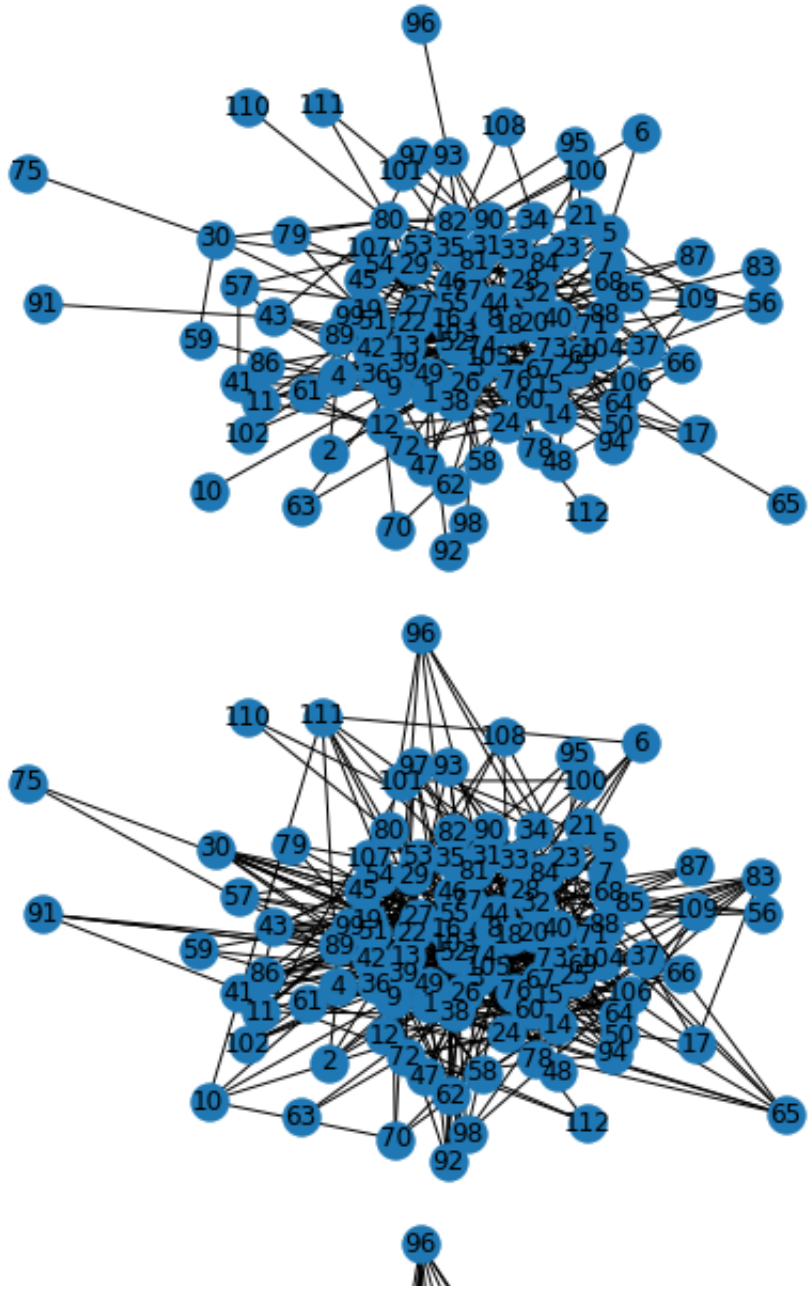
```
['112', '1', '2', '3', '9', '14', '18', '19', '20', '29', '42', '46', '47', '5
[('112', '3'), ('1', '2'), ('1', '3'), ('1', '9'), ('1', '14'), ('1', '18'),
```

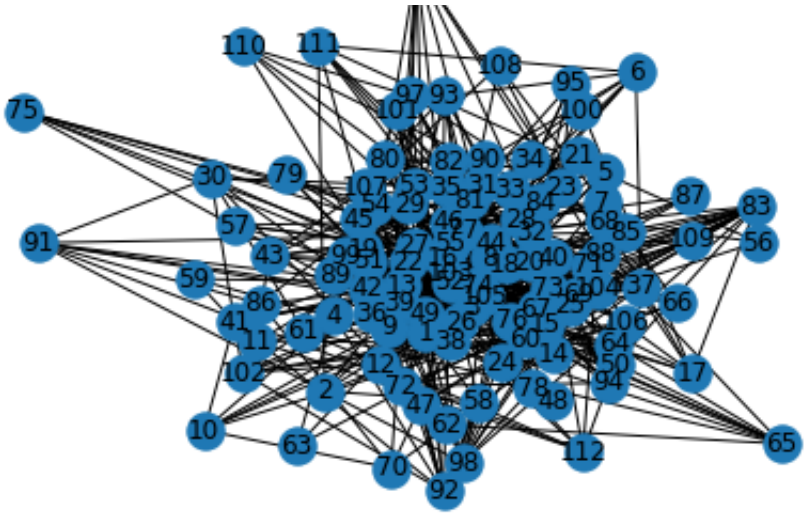
```
#plt.figure(figsize = (8,7))
pos = nx.spring_layout(G, seed = 100)
#degrees = [G.degree(n) for n in G.nodes()]
options = {
    "node_color": 'red',
    "edge_color": "green",
    "width": 3.2,
    "node_size": 50,
    "with_labels": True
}
nx.draw(G,pos, **options)
```

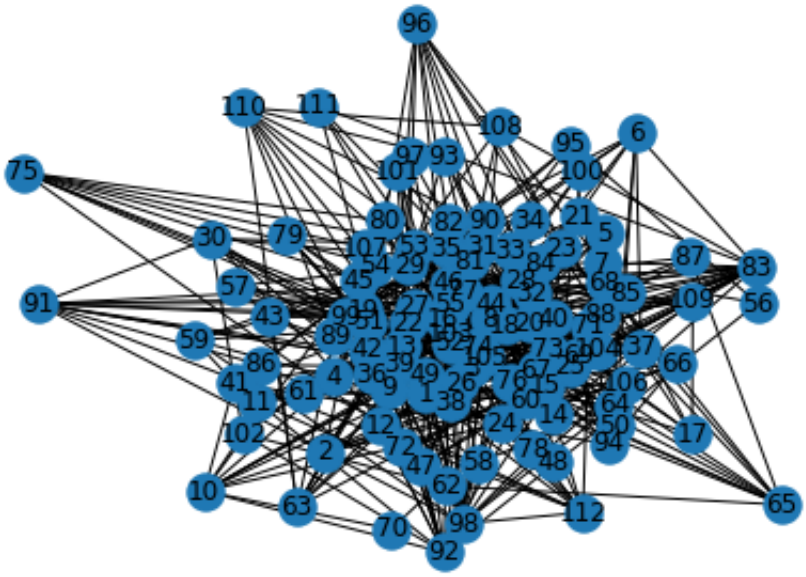


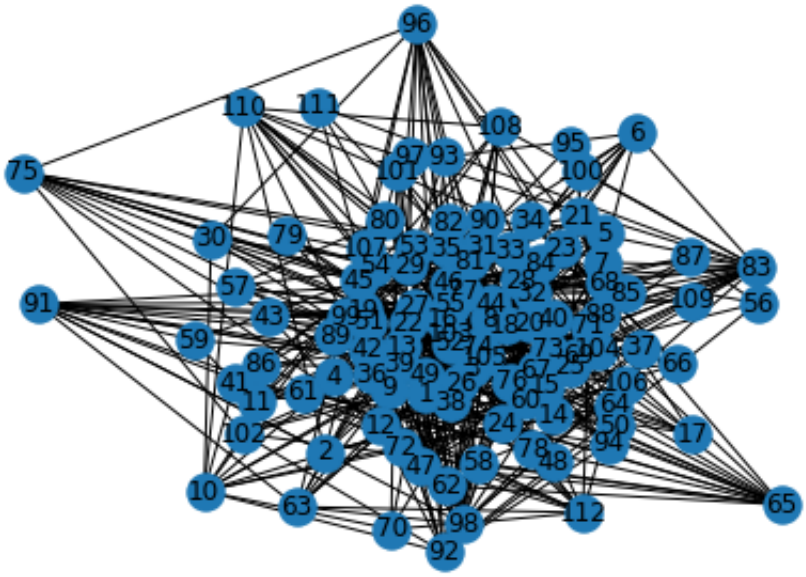
```
n = 15
d = 1.165
p = 0.1
clusts, av_lens, ps = [], [], []
H = [G.copy()] * n
```

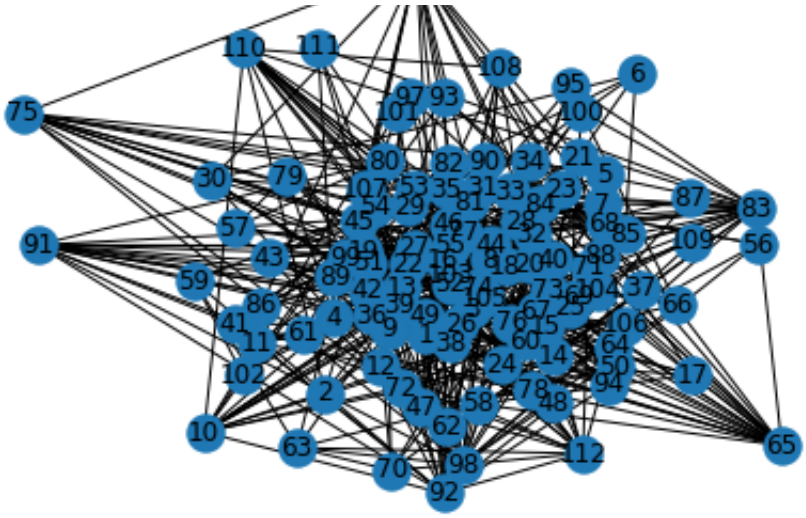
```
for i in range(n):
    fig, ax = plt.subplots(nrows= 1, ncols = 1, figsize = (7,5))
    nx.draw(H[i], pos, with_labels=True)
    Relinkage(H[i], p)
    clusts.append(nx.average_clustering(H[i]))
    av_lens.append(nx.average_shortest_path_length(H[i]))
    # Relinkage(G, p)
    # clusts.append(nx.average_clustering(G))
    # av_lens.append(nx.average_shortest_path_length(G))
    ps.append(p)
    #print(nx.average_clustering(G), len(list(G.edges())))
    p = p * d
```

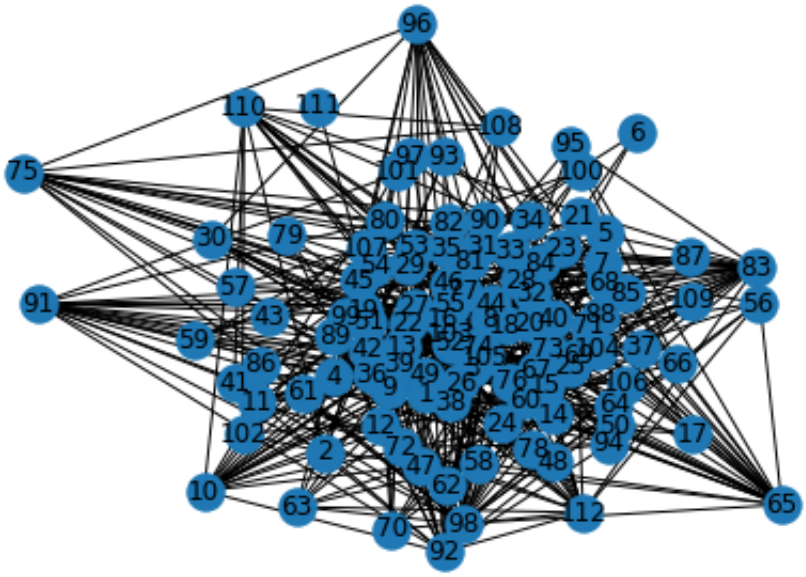


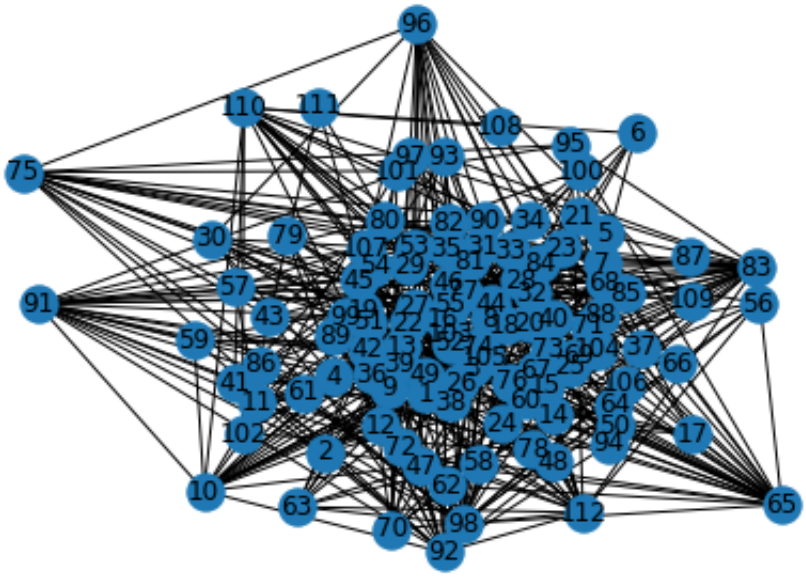


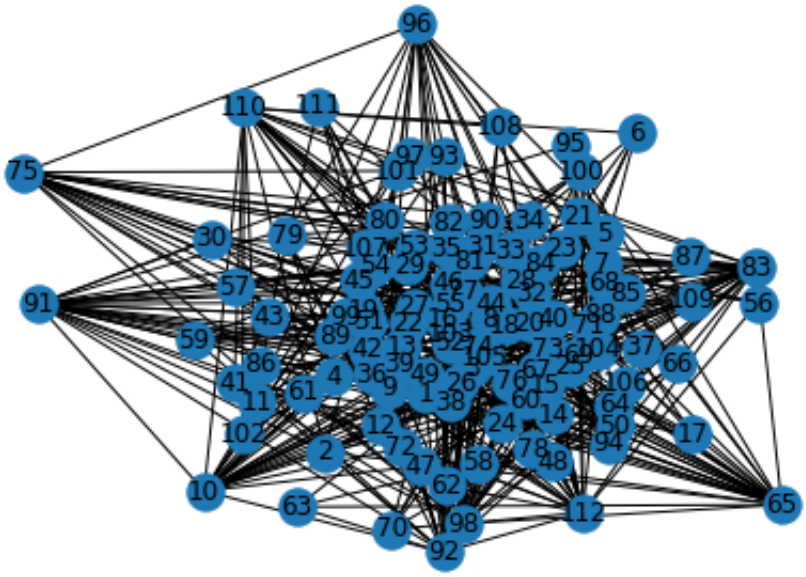


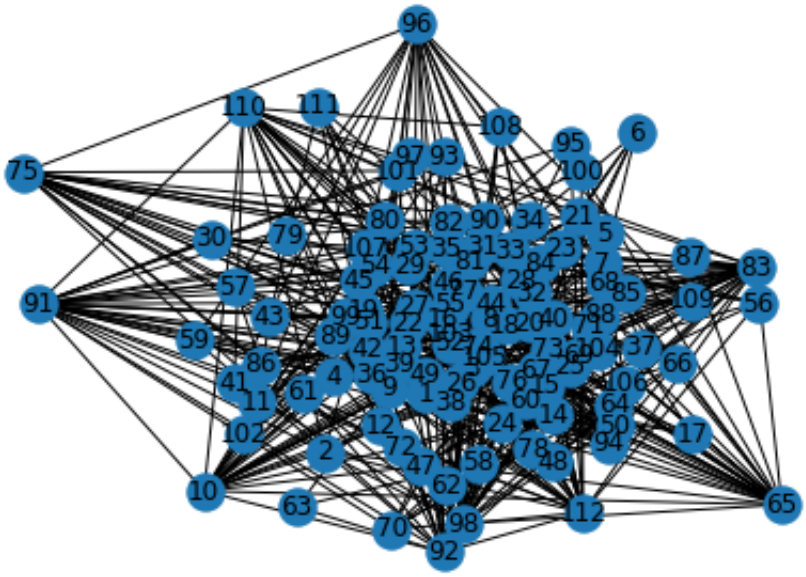


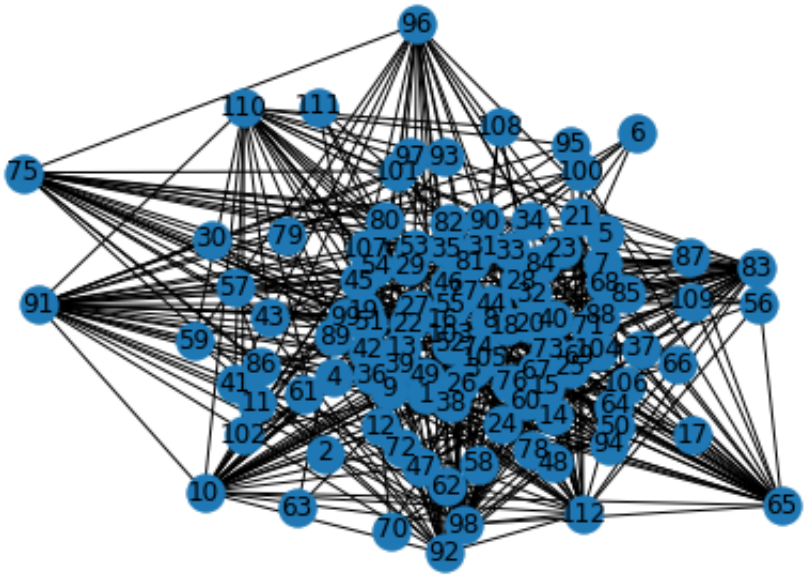


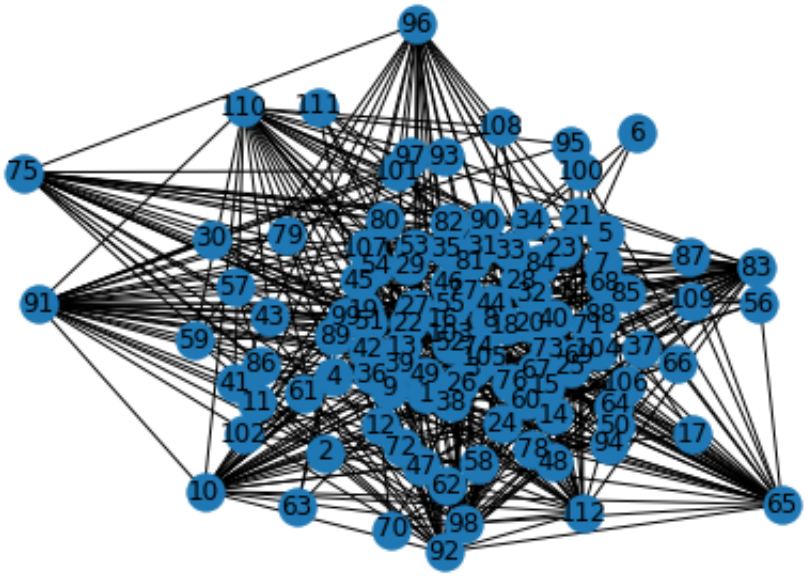




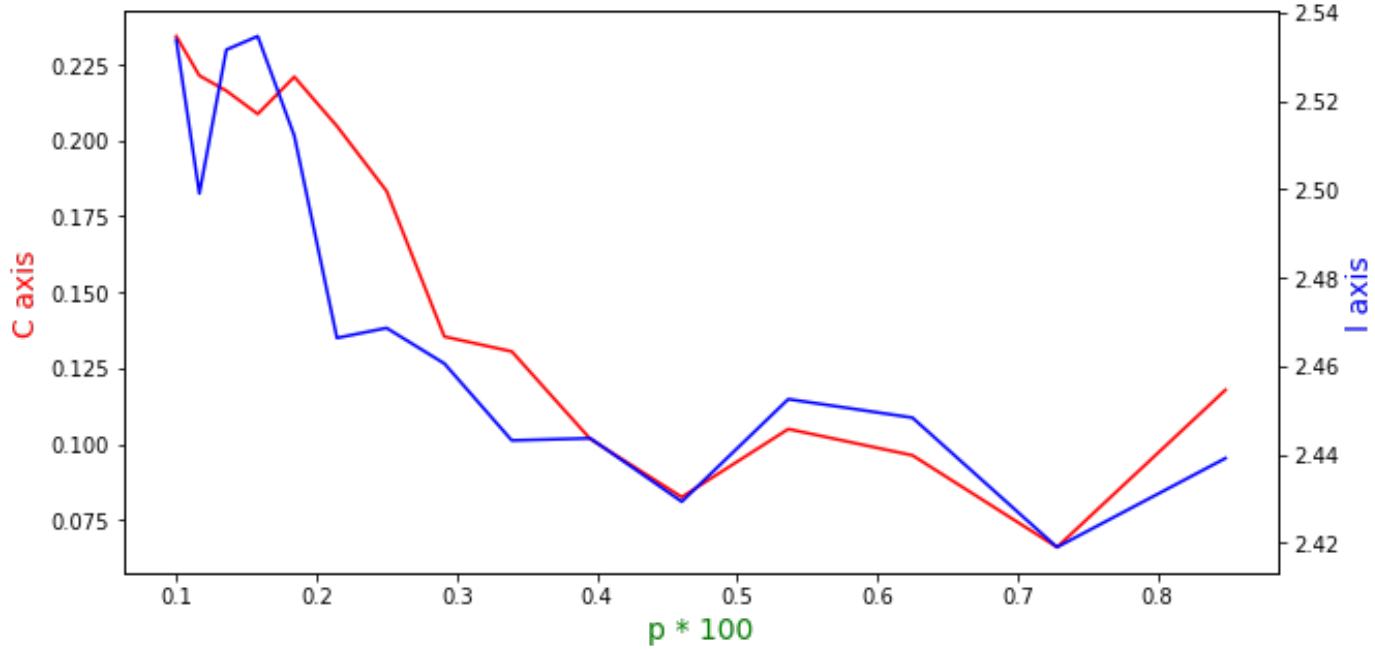








```
fig, ax = plt.subplots(nrows= 1, ncols = 1, figsize = (10,5))
ax.plot(ps, clusts, label = 'C', color = 'red')
ax.set_ylabel("C axis",color="red",fontsize=14)
ax2=ax.twinx()
ax2.plot(ps, av_lens, label = 'l', color = "blue")
ax2.set_ylabel("l axis",color="blue",fontsize=14)
ax.set_xlabel("p * 100",color="green",fontsize=14)
plt.show()
```



Start coding or generate with AI.