Network Science Project 1

Name: Francesco Piatti
CID: 01730921

Please enter your name and 8-digit college ID in the cell above

In [36]:

```
# Do not modify this cell or import any other modules
# without the instructor's permission.
# You should run this cell before running the code below.
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import networkx as nx
import scipy.stats #uncomment if anything from scipy is needed
```

Part 1

1) Complete the Python function *generate* in the cell below.

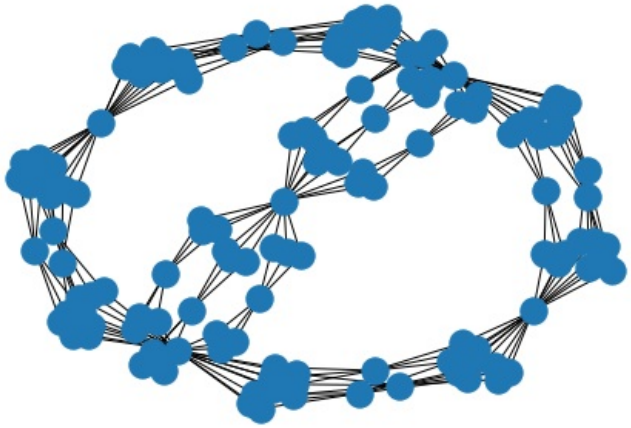
In [37]:

```
def generate(m,T):
    """
    Generate graph based on model given in Project description
    Input:
    m: model parameter; number of length 2 paths replacing each link each iteration
    T: Number of iterations

    Output:
    G: NetworkX graph generated by function
    """
    G = nx.Graph()
    G.add_edge(1,2)          # Iteration - 1

    list1 = []               # Initialize list

    for s in range(2,T+1):                  # for loop representing iterations from 2 to T

        for edge in list(G.edges):          # for loop that chooses one edge at time

            x , y = edge[0], edge[1]        # identify nodes connected by that edge
            t = G.number_of_nodes()         # set a variable that counts the nodes

            for n in range(m):              # for loop that add m edges

                list1.append((t+1,x))       # create two tuples that represent the new edges with a new node
                list1.append((t+1,y))
                t += 1

            G.remove_edge(x,y)              # remove edge
            G.add_edges_from(list1)         # add the edges created (2*m)
            list1.clear()

    return G
```

2) Analyze graphs generated by your function

In [22]:

```
# Here it is presented a graph after 4 iteration with m=3
G = generate(3,4)
nx.draw(G)
```



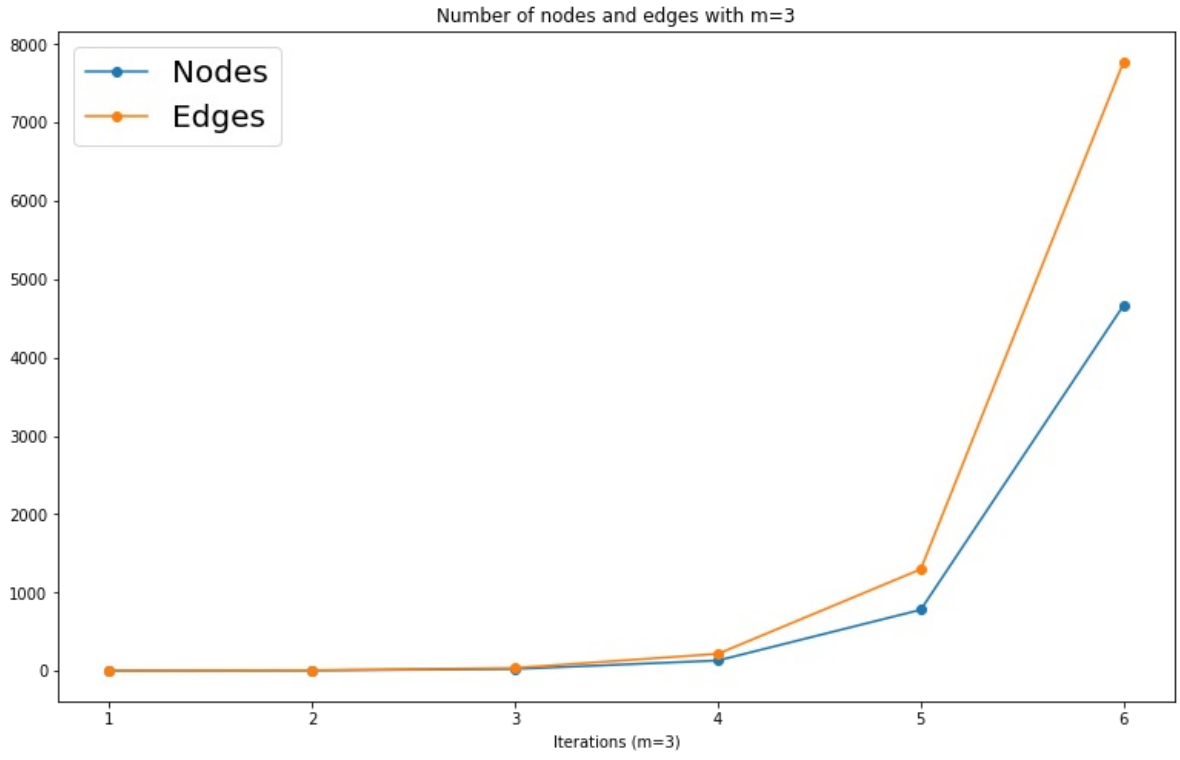
In [38]:

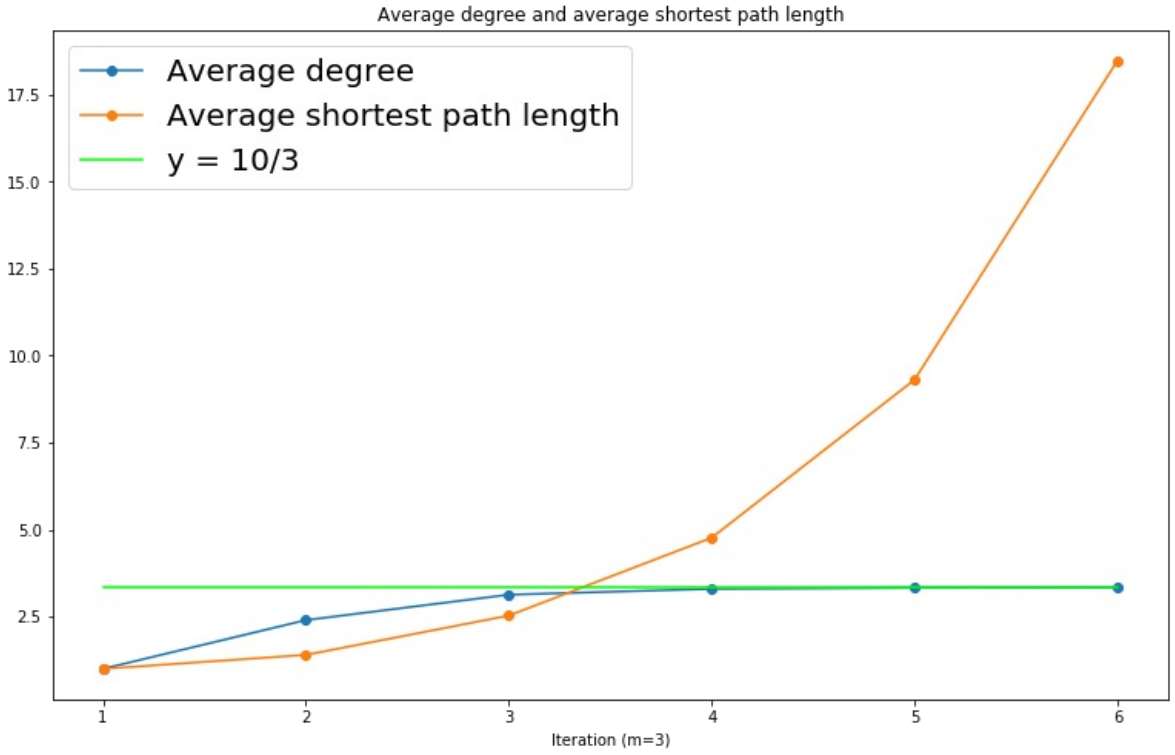
```
def graph_analysis1(G):

    #There is no need to check the number of self-loops and triangles since by construction there are none.

    print("Number of edges:", G.number_of_edges())
    print("Number of nodes:", G.number_of_nodes())
    print("Average degree:", 2*G.number_of_edges()/G.number_of_nodes())    #Evaluated doing 2L/N
    print("Average clustering:", nx.average_clustering(G))
    print("Density:", nx.density(G))
    print("Diameter:", nx.diameter(G))
    print("Average shortest path length:", nx.average_shortest_path_length(G))
    return ""

m=3
A, B, C, D, E, F = generate(m,1), generate(m,2), generate(m,3), generate(m,4), generate(m,5), generate(m,6)
graph_list = [A,B,C,D,E,F]

t=1
for graph in graph_list:
    print("Number of iterations:", t)
    print(graph_analysis1(graph))
    t+=1
```

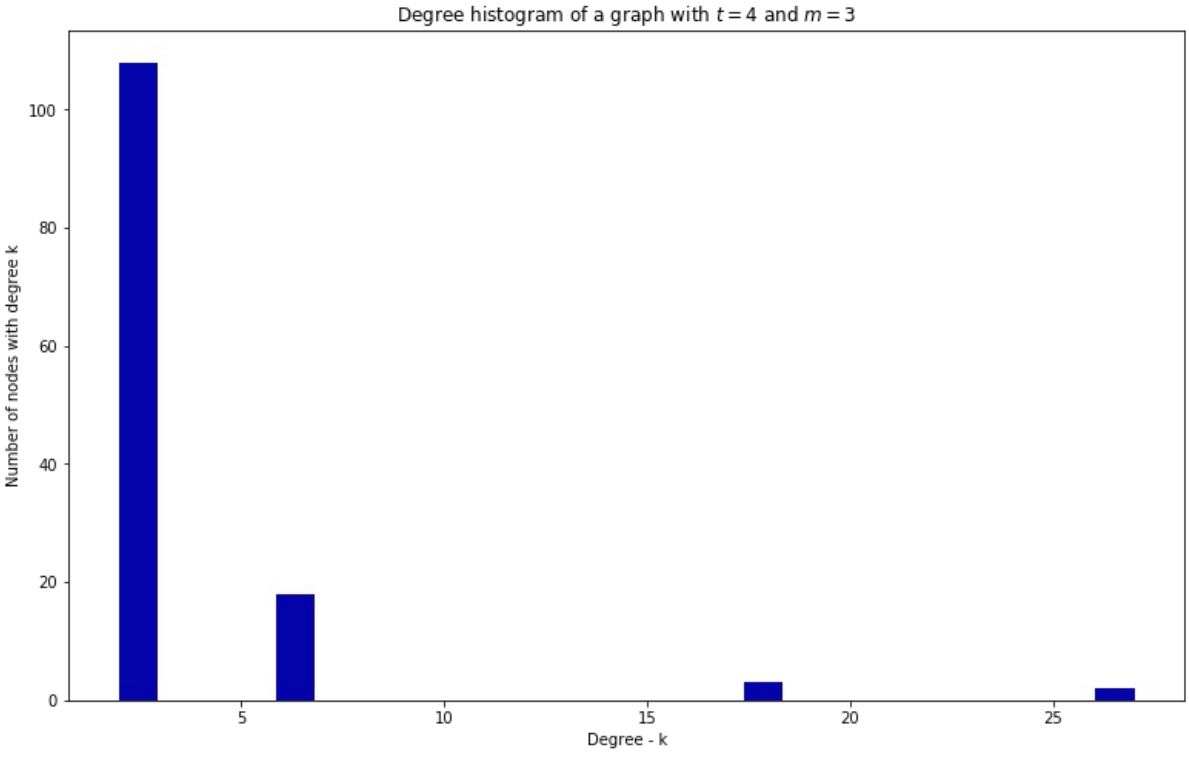
```
Number of iterations: 1
Number of edges: 1
Number of nodes: 2
Average degree: 1.0
Average clustering: 0.0
Density: 1.0
Diameter: 1
Average shortest path length: 1.0

Number of iterations: 2
Number of edges: 6
Number of nodes: 5
Average degree: 2.4
Average clustering: 0.0
Density: 0.6
Diameter: 2
Average shortest path length: 1.4

Number of iterations: 3
Number of edges: 36
Number of nodes: 23
Average degree: 3.130434782608696
Average clustering: 0.0
Density: 0.1422924901185771
Diameter: 4
Average shortest path length: 2.5296442687747036

Number of iterations: 4
Number of edges: 216
Number of nodes: 131
Average degree: 3.2977099236641223
Average clustering: 0.0
Density: 0.02536699941280094
Diameter: 8
Average shortest path length: 4.767116852613036

Number of iterations: 5
Number of edges: 1296
Number of nodes: 779
Average degree: 3.3273427471116817
Average clustering: 0.0
Density: 0.004276790163382624
Diameter: 16
Average shortest path length: 9.309040989205725

Number of iterations: 6
Number of edges: 7776
Number of nodes: 4667
Average degree: 3.332333404756803
Average clustering: 0.0
Density: 0.0007141734686576946
Diameter: 32
Average shortest path length: 18.47820103964774
```

In [39]:

```
#PLOTS

#Number of nodes and edges with m=3
plt.figure(1, figsize=(13,8))
x = np.linspace(1, 6, 6)
nodes = (6**x-1)/10+1.5
edges = 6**(x-1)
plt.plot(x, nodes ,'o-', label = "Nodes")
plt.plot(x, edges, 'o-', label = "Edges")
plt.xlabel("Iterations (m=3)")
plt.title("Number of nodes and edges with m=3")
plt.legend(fontsize=20)
plt.show()

#Average degree for m=3
plt.figure(2, figsize=(13,8))
xlist=list(range(1,len(graph_list)+1))
average_degree_list=[]
diameter_list=[]
shortest_path_average_list=[]
for graph in graph_list:
    average_degree_list.append(2*graph.number_of_edges()/graph.number_of_nodes())
    shortest_path_average_list.append(nx.average_shortest_path_length(graph))
plt.plot(xlist, average_degree_list, 'o-', label="Average degree")
plt.plot(xlist, shortest_path_average_list, 'o-', label="Average shortest path length")
line_list = [10/3 for x in xlist]
plt.plot(xlist, line_list,'-', color="lime", label="y = 10/3")
plt.legend(fontsize=20)
plt.title("Average degree and average shortest path length")
plt.xlabel("Iteration (m=3)")
plt.show()

#Degree distribution
plt.figure(3, figsize=(13,8))
degree_array1 = np.array([D.degree(n) for n in sorted(D.nodes())])
bin_edges = np.linspace(start=(degree_array1.min()), stop=(degree_array1.max()), num= degree_array1.max(), endpoi
nt=True)
n, bins, patches = plt.hist(x=degree_array1, bins=bin_edges, color='#0504aa')
plt.title('Degree histogram of a graph with $t=4$ and $m=3$')
plt.xlabel('Degree - k')
plt.ylabel('Number of nodes with degree k')
plt.show()
```




Average degree and average shortest path length
Average degree
Average shortest path length
y = 10/3
17.5
15.0
12.5
10.0
7.5
5.0
2.5
1
2
3
4
5
6
Iteration (m=3)


Degree histogram of a graph with $t = 4$ and $m = 3$
Number of nodes with degree k
100
80
60
40
20
0
5
10
15
20
25
Degree - k

Discussion

To begin with, it is important to notice that, no matter how many iterations or the m coefficient used, these graphs share some features *by construction*:

1. They are all undirected and unweighted
2. They are all connected
3. The clustering coefficient is always 0 regardless of the degree of the node
4. The number of triangles and self-loops is always 0

As far as the number of nodes and edges is concerned, it is easy to find equations to express them in terms of t and m.

Number of edges: $(2m)^{t-1}$ and thus for $m = 3$, $L(t) = 6^{t-1}$

Number of nodes: here is slightly more complicated. We notice that by construction $N(t+1) = N(t) + m \cdot L(t)$, thus obtaining the simple recurrence relation $N(t+1) = N(t) + m \cdot (2m)^{t-1}$ whose solution, considering that $f(1) = 2$, $\forall m$ is

$$N(t,m) = \frac{(2m)^t - 1}{2(2m-1)} + \frac{3}{2}$$

Which means that for $m = 3$ we have:

$$N(t) = \frac{1}{10}(6^t - 1) + \frac{3}{2}$$

Notice that both edges and nodes increase exponentially with iterations, as shown in *figure 1*.

Degree

The average degree is given by the formula $k = 2L(t)/E(t)$, therefore in the case where

$$k = \frac{2(2m)^{t-1} \cdot 2(2m-1)}{(2m)^t - 1 + 3(2m-1)}$$

for t large, $k \approx 4 - \frac{2}{m}$. In our case, with $m = 3$, we get that $k \to \frac{10}{3}$ (*figure 2*).

When it comes to degree distribution, which you can appreciate in *figure 3*, it is again fairly easy to model it analitically. Notice that for $m = 3$ and $t > 1$ fixed, the degree of a node can assume the values $2 \cdot 3^n$ for $n \in \{0, \dots, t-2\}$ and 3^{t-1}. This is easily explained, since the degree of the initial nodes triplicates every iteration. In fact, at the end only these two nodes will have the maximum degree. It is important to notice that the maximum degree increases exponentially with iterations.

Other features

There are other features that are easy to describe mathematically, for instance one can easily notice that the diameter is

$$d_{max} = 2^{t-1}$$

As diameter increases exponentially with iterations, also the average shortest path length does, as shown again in *figure 2*.

Since this graph is not random, probability generating functions are not required in the analysis.

Part 2

1) Analyze graph corresponding to edge list loaded in cell below

In [40]:

```
# Run this cell to create edge list for graph to be analyzed
E = np.loadtxt('project1.dat', dtype=int)
```

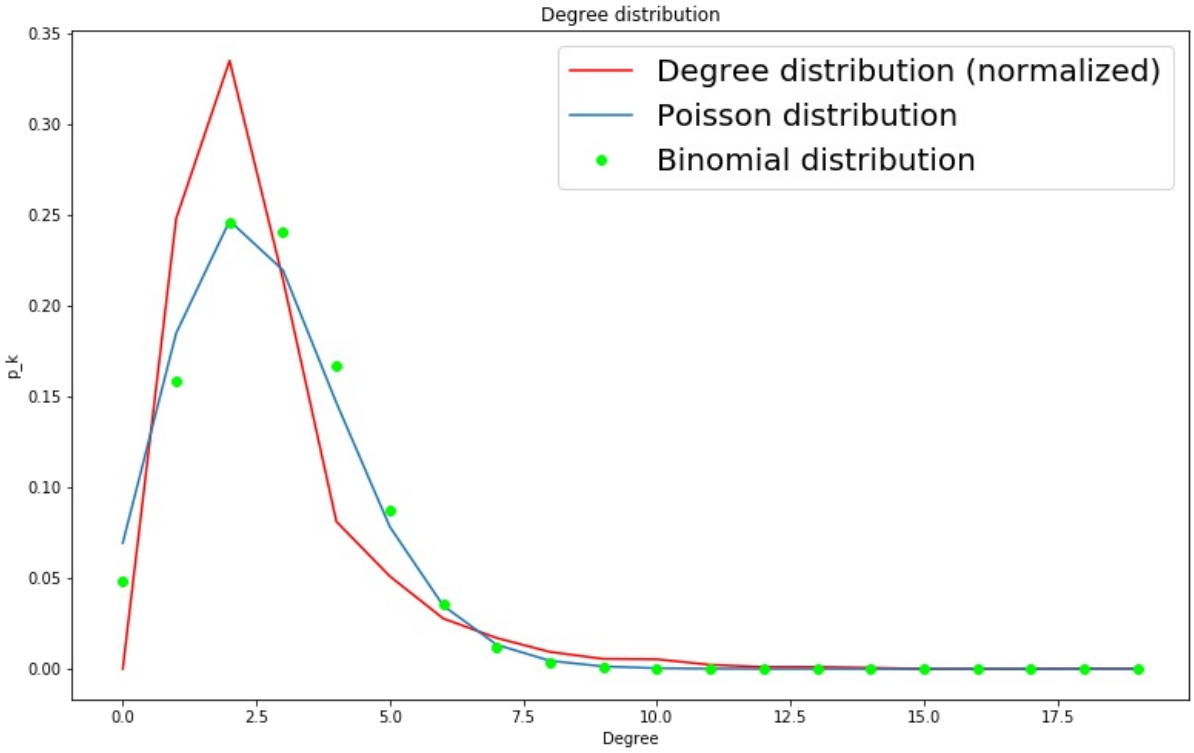
In [41]:

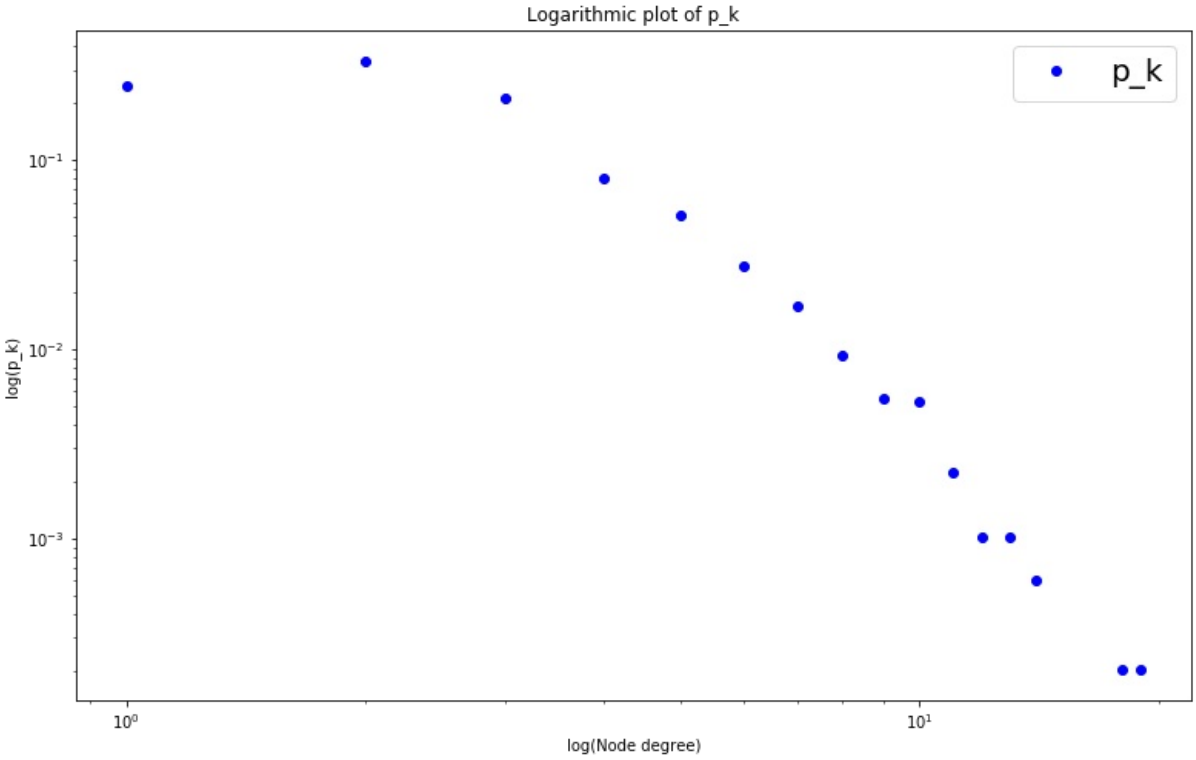
```
G = nx.Graph()
G.add_edges_from(E)

print("The graph is directed:", nx.is_directed(G))
print("The graph is weighted:", nx.is_weighted(G))
print("The graph is connected:", nx.is_connected(G))
print("Number of edges:", G.number_of_edges())
print("Number of nodes:", G.number_of_nodes())
print("Average degree:", 2*G.number_of_edges()/G.number_of_nodes())
print("Average local clustering:", nx.average_clustering(G))
print("Average global clustering:", nx.transitivity(G))
print("Number of self-loops:", nx.number_of_selfloops(G))
print("Number of triangles:", sum([nx.triangles(G,n) for n in sorted(G.nodes())])/3)
print("Density:", nx.density(G))
print("Diameter:", nx.diameter(G))
print("Average shortest path length:", nx.average_shortest_path_length(G))
```

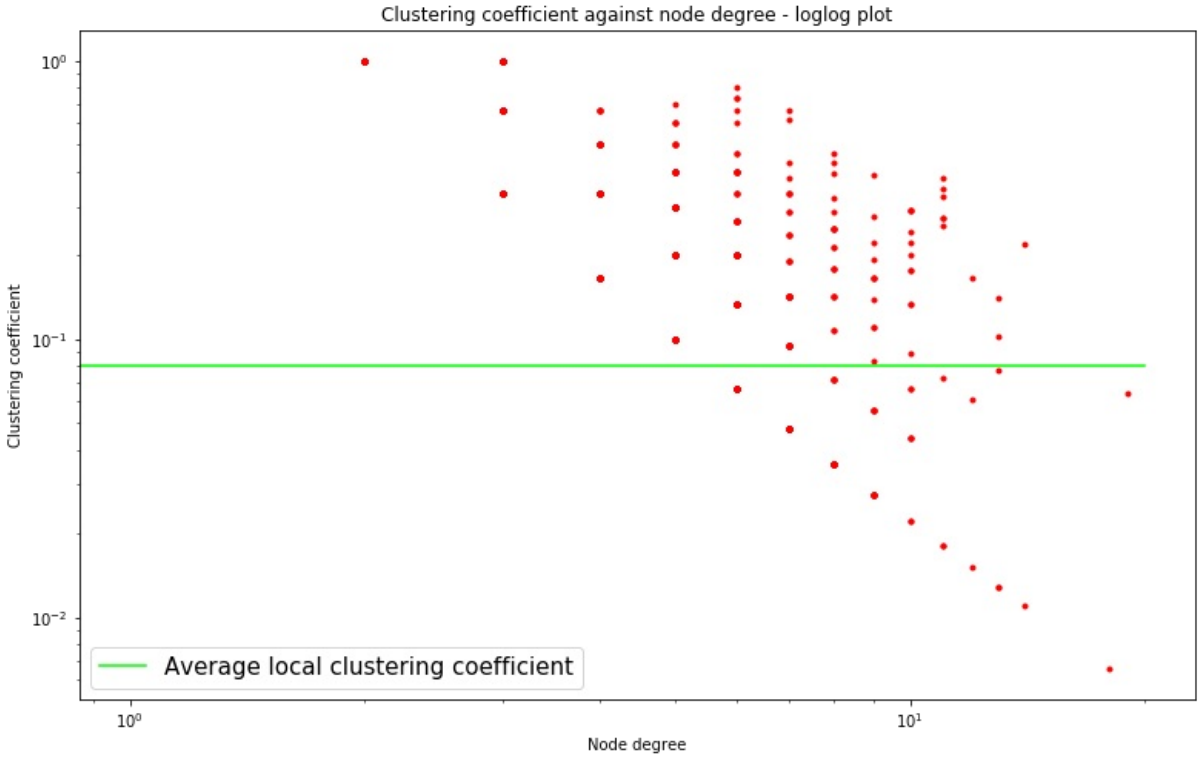
```
The graph is directed: False
The graph is weighted: False
The graph is connected: True
Number of edges: 6594
Number of nodes: 4941
Average degree: 2.66909532483303
Average local clustering: 0.08010361108159714
Average global clustering: 0.10315322452860086
Number of self-loops: 0
Number of triangles: 651.0
Density: 0.0005403026973346214
Diameter: 46
Average shortest path length: 18.989185424445708
```

In [42]:

```
#PLOTS

clustering_list = [nx.clustering(G,n) for n in sorted(G.nodes())]
degree_list = [G.degree(n) for n in sorted(G.nodes())]
degree_array = np.array(degree_list)
average_degree = 2*G.number_of_edges()/G.number_of_nodes()

#Degree distribution compared to poisson and binomial
plt.figure(4, figsize=(13,8))
p_k = np.array(nx.degree_histogram(G))/len(G.nodes())
plt.plot(p_k,'-', color='red', label="Degree distribution (normalized)")
x = np.arange(0,20,1)
y = [scipy.stats.poisson.pmf(n, average_degree) for n in x]
plt.plot(x, y, '-', label="Poisson distribution")
binom = [scipy.stats.binom.pmf(n, 20, average_degree/19) for n in np.arange(0,20,1)]
plt.plot(np.arange(0,20,1), binom, 'o', color="lime", label="Binomial distribution")
plt.title("Degree distribution")
plt.xlabel("Degree")
plt.ylabel("p_k")
plt.legend(fontsize=20)
plt.show()

#Degree distribution logarithmic plot
plt.figure(5, figsize=(13,8))
plt.plot(p_k,'o', color='blue', label="p_k")
plt.title("Logarithmic plot of p_k")
plt.xlabel("log(Node degree)")
plt.ylabel("log(p_k)")
plt.legend(fontsize=20)
plt.loglog()
plt.show()

#Clustering coefficient against node degree
plt.figure(6, figsize=(13,8))
plt.plot(degree_list, clustering_list, '.', color="red")
av_clust_line = [nx.average_clustering(G), nx.average_clustering(G)]
plt.plot([0,20], av_clust_line, '-', color="lime", label="Average local clustering coefficient")
plt.title("Clustering coefficient against node degree - loglog plot")
plt.xlabel("Node degree")
plt.ylabel("Clustering coefficient")
plt.legend(fontsize=15, loc='lower left')
plt.loglog()
plt.show()
```


Degree distribution
Degree distribution (normalized)
Poisson distribution
Binomial distribution
p_k
Degree
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Logarithmic plot of p_k
p_k
log(p_k)
log(Node degree)
10^{-1}
10^{-2}
10^{-3}
10^{0}
10^{1}



Discussion

To begin with, let us consider the outcomes of the *graph_info* function: this unweighted and undirected graph is connected; furthermore, the small clustering coefficient, along with the relatively low average degree and the low density, lead to a large *average shortest path lengh* and diameter (especially when compared to the “6 degrees of Kevin Bacon” game).

Figure 1 portrays the normalized degree distribution compared to a poisson and binomial distribution whose parameters are respectively $k_{average}$ and $(N-1)$, clustering_coefficient. We notice that even if the shape of the degree distribution resembles the probability mass functions, these ones don't seem to represent the graph really well. This may rise the argument that the graph is not random.

In support of this hypothesis, we can look at *figure 3*, which represents the local clustering coefficients in terms of the degree.

In fact, if the graph were totally random, we would not notice any pattern in the $C(k)/k$ plot, which, instead, seem to decrease as the node degree increases. (Recall that in a random network the local clustering coefficient of a node is independent of the node’s degree since $C_i = k_{average}/N$)

The definitive proof of this comes when reading the book by Network Science by Barabasi, which shows in *Table 2.1* that this network (or at least a network with the same number of nodes and edges and the same average degree) represents a **power grid** where nodes represent the power plants or traformers, and edges represent the cables.

Now the outcomes make much more sense: the absense of self-loops, the small density and so on...

In [ ]:

In [ ]: