In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# Load Data
Diameter = np.genfromtxt('Diameter.csv', delimiter=',')
Clustering_coefficient = np.genfromtxt('Clustering Coefficient.csv', deli
miter=',')
Degree_dist = np.genfromtxt('Degree Distribution.csv', delimiter=',')
```

In [3]:
```
# Print out one set of data for example
print(Diameter)
```

```
[[ 3.    2.02]
 [ 4.1   2.82]
 [ 5.5   3.14]
 [ 7.4   3.92]
 [ 8.    3.81]
 [ 9.1   3.82]
 [10.9   5.01]
 [11.2   5.04]
 [12.    4.97]
 [13.5   5.32]
 [14.8   5.81]
 [15.2   5.74]
 [16.4   6.01]
 [17.1   6.04]
 [18.1   7.04]
 [19.    6.98]
 [20.    7.02]
 [22.4   7.24]]
```

The first columne represents the input size on log scale of the data (data size = 2^(value of the first colume)), and the second colume represents the value of the characteristics of the graph.

Introduction

In this project, we try to experimentally determine different characteristics of large-scale random graphs. These graph characteristics are diameter, the clustering-coefficient, and the degree-distribution. Diameter of a graph refers to the longest path in the graph. Clustering-coefficient of a graph measures how the nodes in the graph tend to cluster together. Degree-distribution of a graph is the probability distribution of the degrees of the nodes in the graph, which determines what extent the nodes are connected to each other. We intend to determine the correlation between the graph size (n) and these characteristics. I have done five trials of random graphs with increasing sizes, and the results of these trials are averaged to minimize variance.

The random graphs are generated by an algorithm called Barabasi-Albert model, which creates a pseudo-random graph that is similar to a real-world graph.

Barabasi-Albert Model pseudo-code

In [ ]:
```
Barabasi-Albert(n, d):
```

```
    G - undirected unweighted graph with two nodes connected to each othe
r
    for i from 0 to n:
        add a node v to G
        for j from 0 to d:
            for all nodes k (expect for v) in G:
                p = degree of k / sum of all degrees
                connect v to k with probaility p
    return G
```

Diameter pseudo-code

In [ ]:
```
Diameter(G):
    Dmax - 0
    longest_path = get_longest_path(random node in the graph)
    while(longest_path.dist > Dmax):
        Dmax = longest_path.dist
        longest_path = get_longest_path(longest_path.destination)
    return Dmax
```

Clustering-coefficient pseudo-code

In [ ]:
```
Clustering_coefficient(G):
    return 3 * triangles(G) / two_edge_paths(G);

triangles(G):
    triangles - 0
    for vertex v in G:
        for every distinct pair of neighbouring nodes u,w of v:
            if u, w have higher degree than v (tie-breaker when equal):
                if u,w are connected:
                    triangles++
    return triangles

two_edge_paths(G):
    two_edge_paths - 0
    for vertex v in G:
        two_edge_paths += degree of v * (degree of v - 1) / 2
    return two_edge_paths
```

Degree-distribution pseudo-code

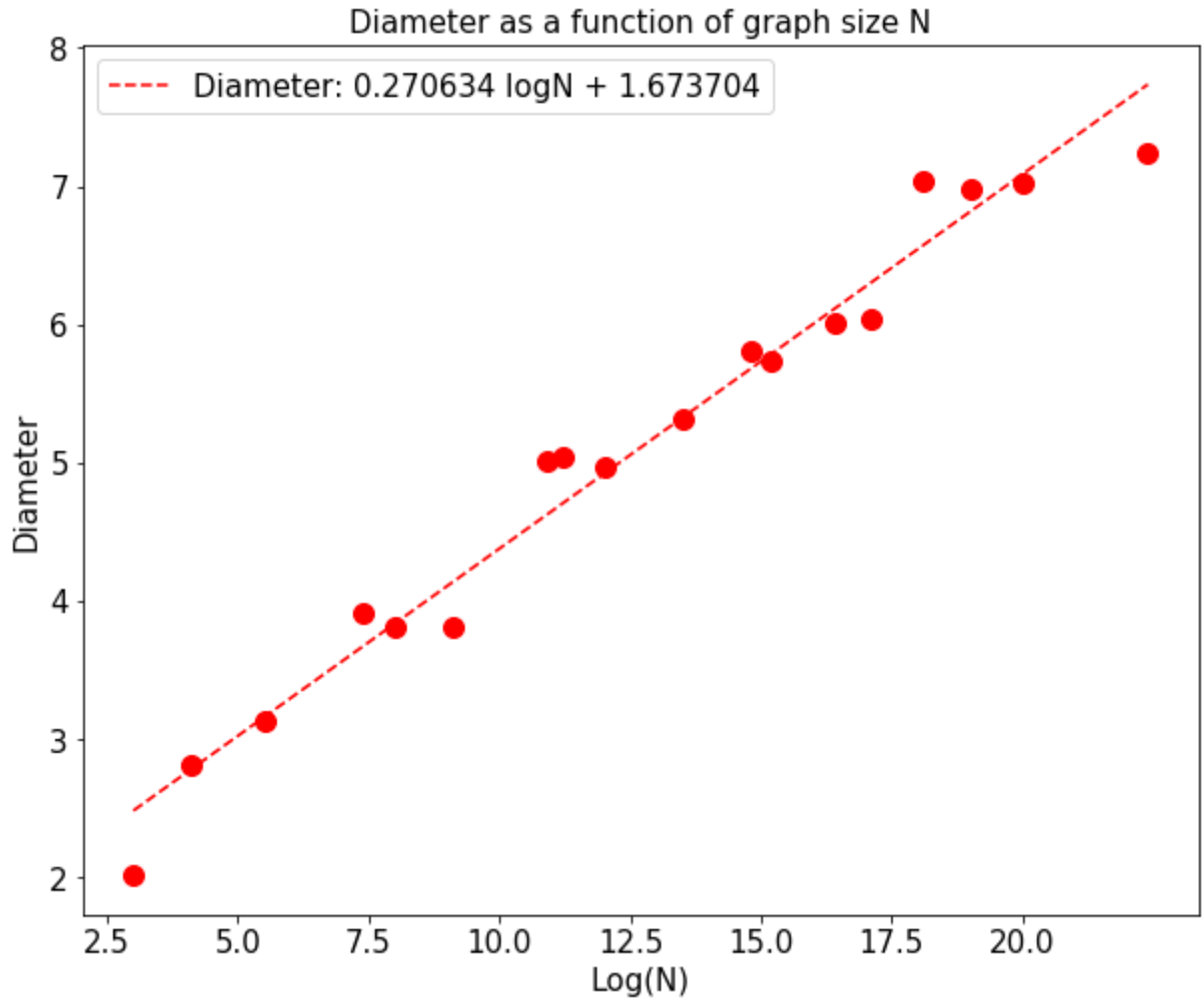
In [ ]:
```
Degree_distribution(G):
    dist - {}
    for vertex v in G:
        dist[degree of v]++
    return dist
```

Results and Plots

Diameter

In [5]:
```
# Plotting Diameter
f, ax = plt.subplots(1,1, figsize=(10,8))


m1, b1 = np.polyfit(Diameter[:,0], Diameter[:,1], 1)

plt.plot(Diameter[:,0], Diameter[:,1], '.',
```

```
            markersize=20, color='red')

plt.plot(Diameter[:,0], m1*Diameter[:,0]+b1,
         linestyle='--', label=f'Diameter: {m1:4f} logN + {b1:4f}', color
='red')

plt.xlabel('Log(N)', fontsize=15)
plt.ylabel('Diameter', fontsize=15)
plt.xticks(np.arange(2.5, 22.5, 2.5), fontsize=15)
plt.yticks(fontsize=15)
plt.title('Diameter as a function of graph size N', fontsize=15)
plt.legend(fontsize=15)
plt.show()
```

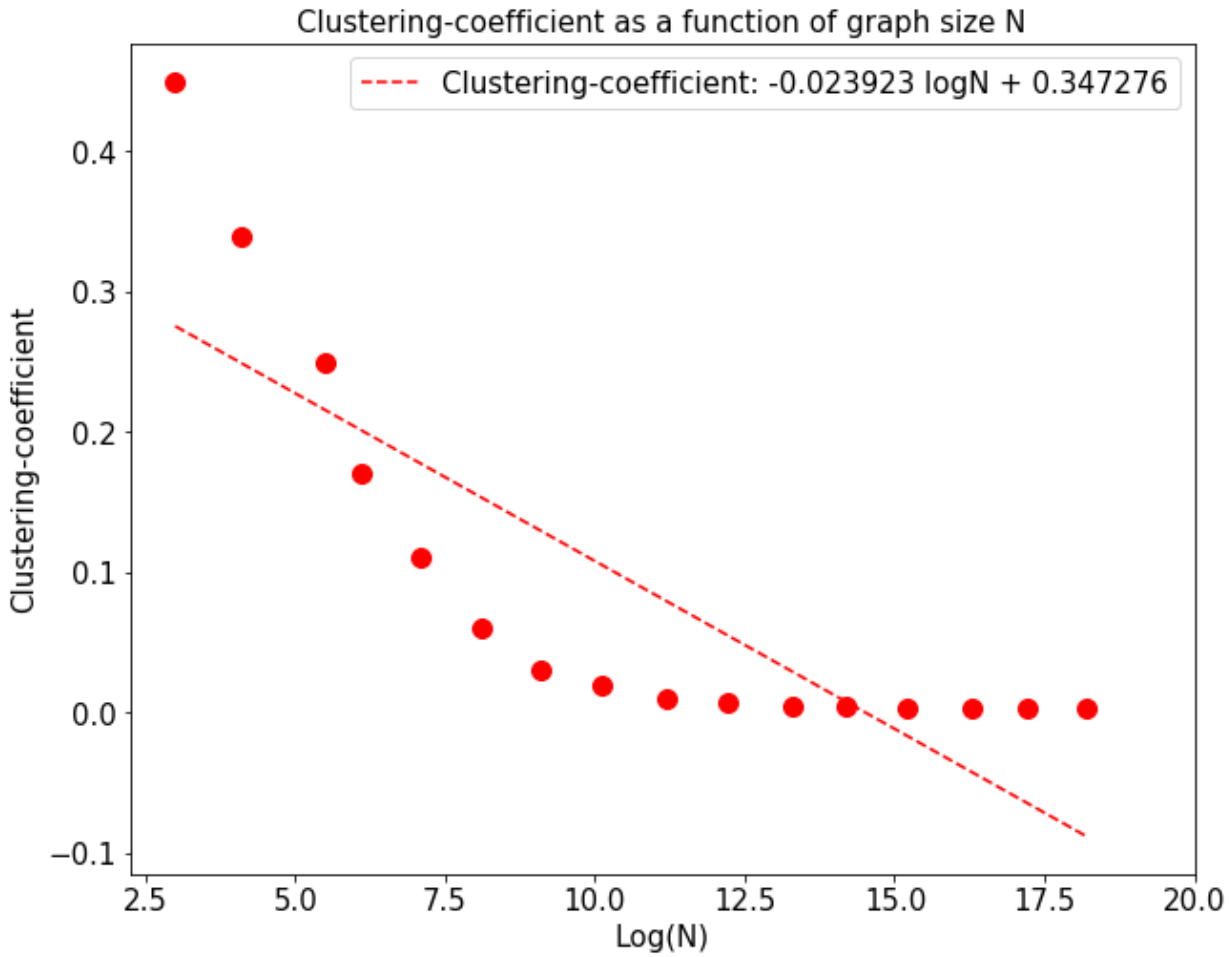


Comments:

In a lin-log plot, the results can be nicely fit with a linear regression line which has a possitive slope. Therefore, the correlation between diameter and data size of a graph can be represents as Diameter = c * log(N). This means the diameter of a graph is proportional to the log of its size. C is a constant which is 0.270634 in this case.

Clustering-coefficient

In [6]:
```
# Plotting Clustering-coefficient
f, ax = plt.subplots(1,1, figsize=(10,8))


m1, b1 = np.polyfit(Clustering_coefficient[:,0], Clustering_coefficient
[:,1], 1)

plt.plot(Clustering_coefficient[:,0], Clustering_coefficient[:,1], '.',
          markersize=20, color='red')

plt.plot(Clustering_coefficient[:,0], m1*Clustering_coefficient[:,0]+b1,
         linestyle='--', label=f'Clustering-coefficient: {m1:4f} logN +
{b1:4f}', color='red')

plt.xlabel('Log(N)', fontsize=15)
plt.ylabel('Clustering-coefficient', fontsize=15)
plt.xticks(np.arange(2.5, 22.5, 2.5), fontsize=15)
plt.yticks(fontsize=15)
plt.title('Clustering-coefficient as a function of graph size N', fontsiz
e=15)
plt.legend(fontsize=15)
plt.show()
```

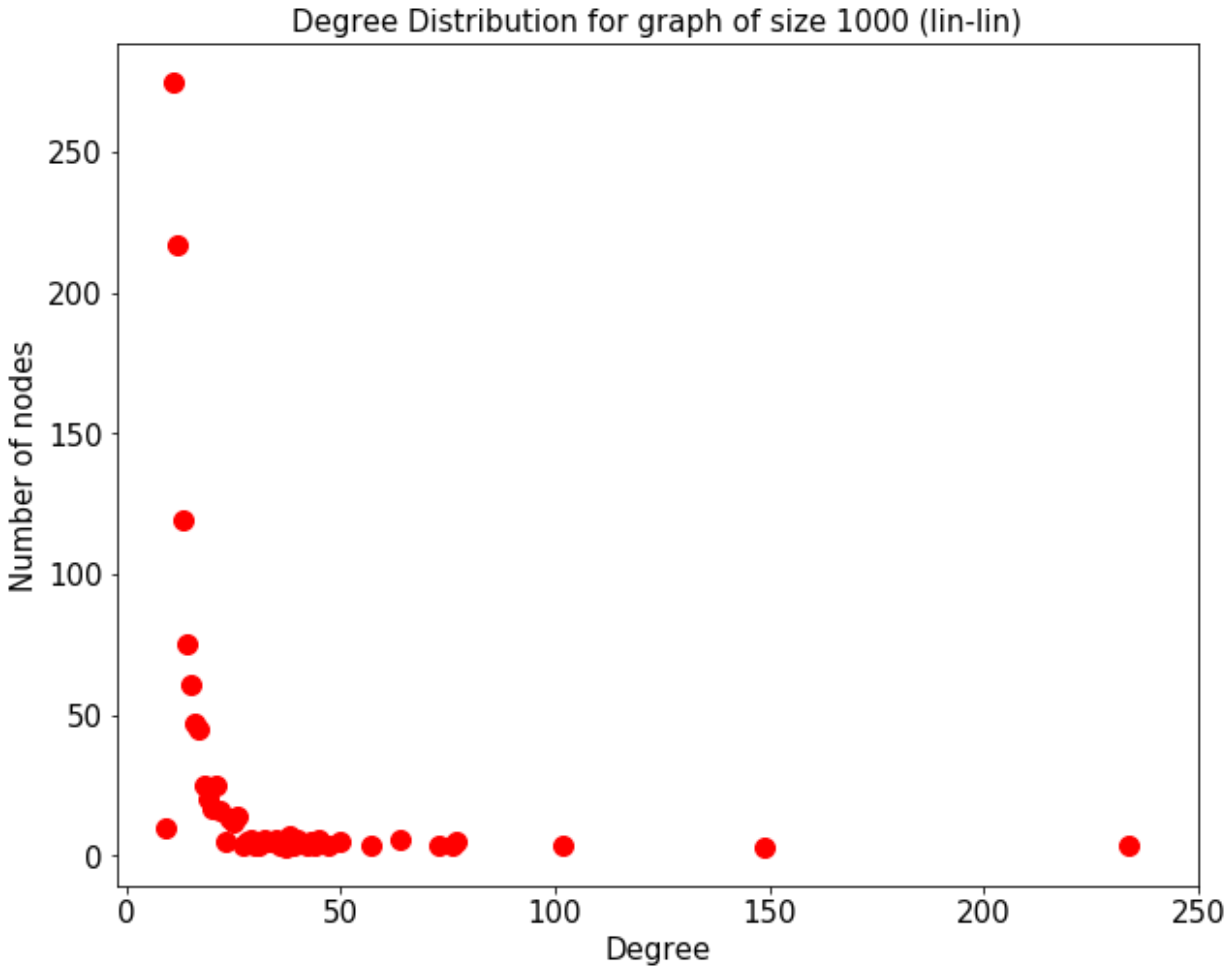


Comments:

In a lin-log plot, the results can not be fit with a linear regression line. However, we can see that the clustering-coefficient of the graph is decreasing as the size of the graph is increasing, and the rate of the decrease is also decreasing as the size of the graph is increasing. Therefore, the correlation between clustering-coefficient and data size of a graph can be represents as Clustering-coefficient = c * log(1/N). This means the clustering-coefficient of a graph is proportional to log(1/N), which is slowlier than log(N).

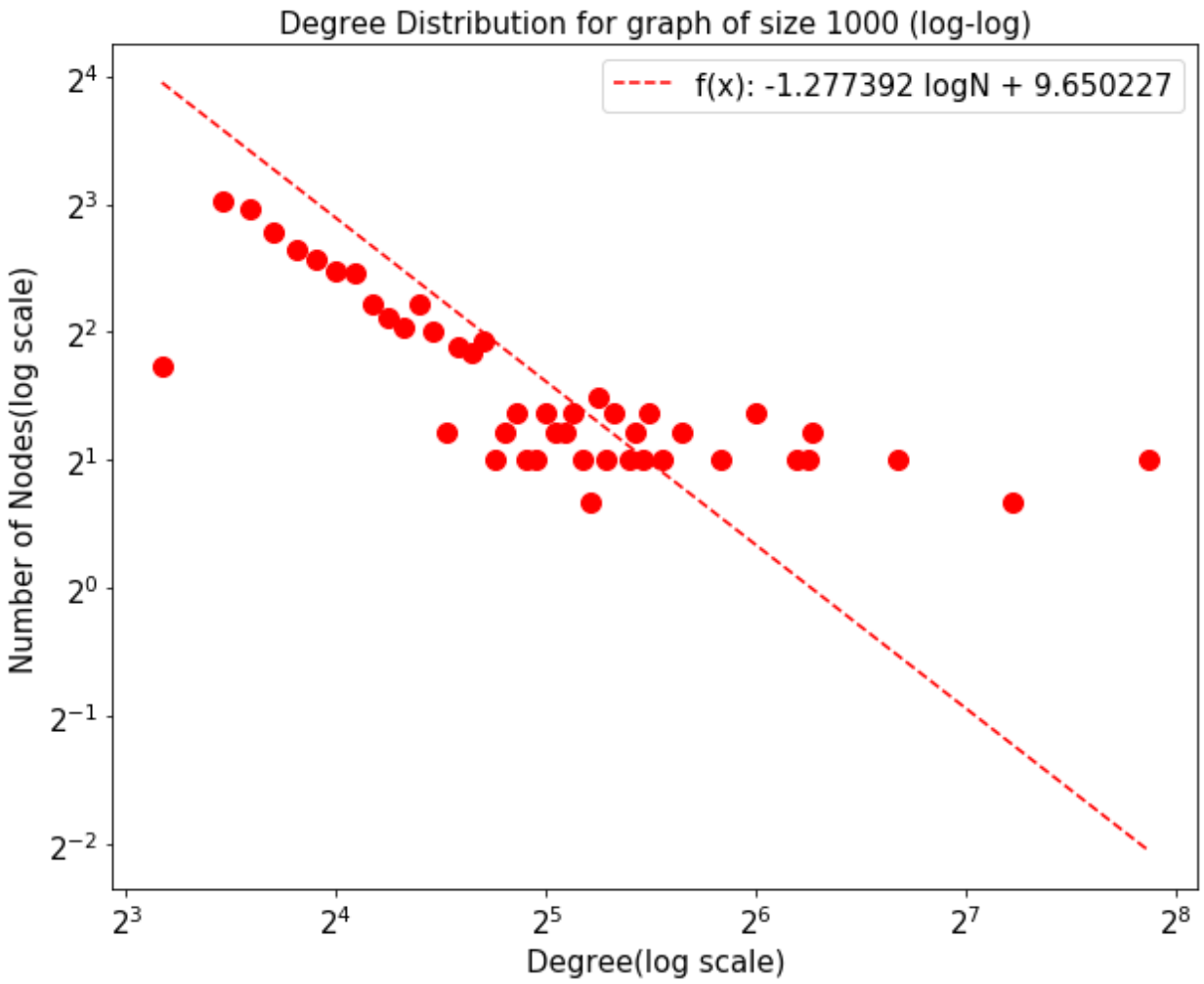
Degree-distributions

In [7]:
```
# Plotting Degree-distribution(lin-lin)
# N = 1000
f, ax = plt.subplots(1,1, figsize=(10,8))

plt.plot(Degree_dist[0:45,0], Degree_dist[0:45,1], '.',
           markersize=20, color='red')

plt.xlabel('Degree', fontsize=15)
plt.ylabel('Number of nodes', fontsize=15)
plt.xticks(np.arange(0, 300, 50), fontsize=15)
plt.yticks(fontsize=15)
plt.title('Degree Distribution for graph of size 1000 (lin-lin)', fontsiz
e=15)
plt.show()
```



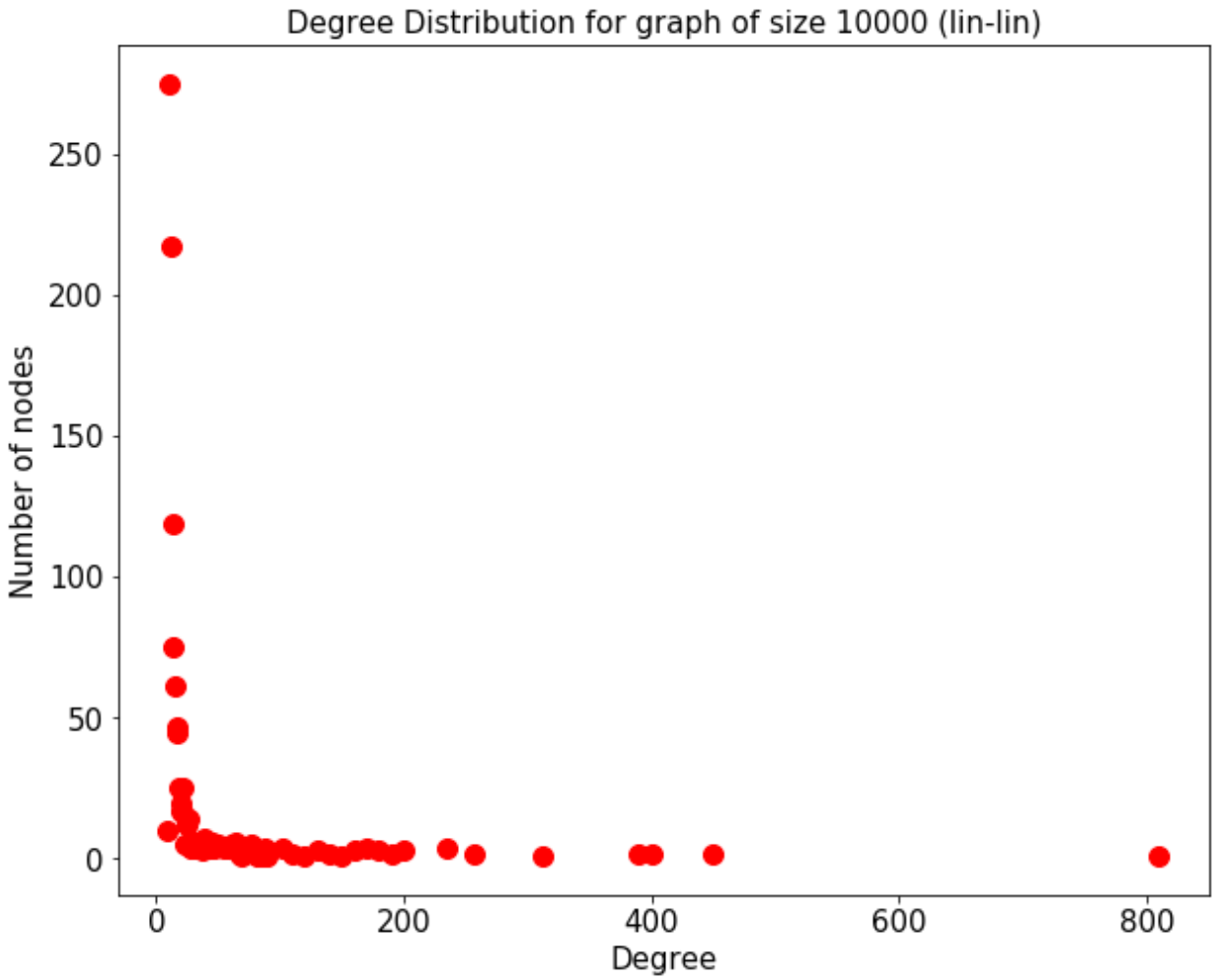
In [8]:
```
# Plotting Degree-distribution (log-log)
f, ax = plt.subplots(1,1, figsize=(10,8))

m1, b1 = np.polyfit(np.log2(Degree_dist[0:45,0]), np.log2(Degree_dist[0:4
5,1]), 1)

plt.loglog(Degree_dist[0:45,0], np.log2(Degree_dist[0:45,1]), '.',basex=2
, basey=2,
           markersize=20, color='red')

plt.loglog(Degree_dist[0:45,0], 2**(m1*np.log2(Degree_dist[0:45,0])+b1),b
asex=2, basey=2,
         linestyle='--', label=f'f(x): {m1:4f} logN + {b1:4f}', color='re
d')

plt.xlabel('Degree(log scale)', fontsize=15)
plt.ylabel('Number of Nodes(log scale)', fontsize=15)
```

```
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.title('Degree Distribution for graph of size 1000 (log-log)', fontsiz
e=15)
plt.legend(fontsize=15)
plt.show()
```



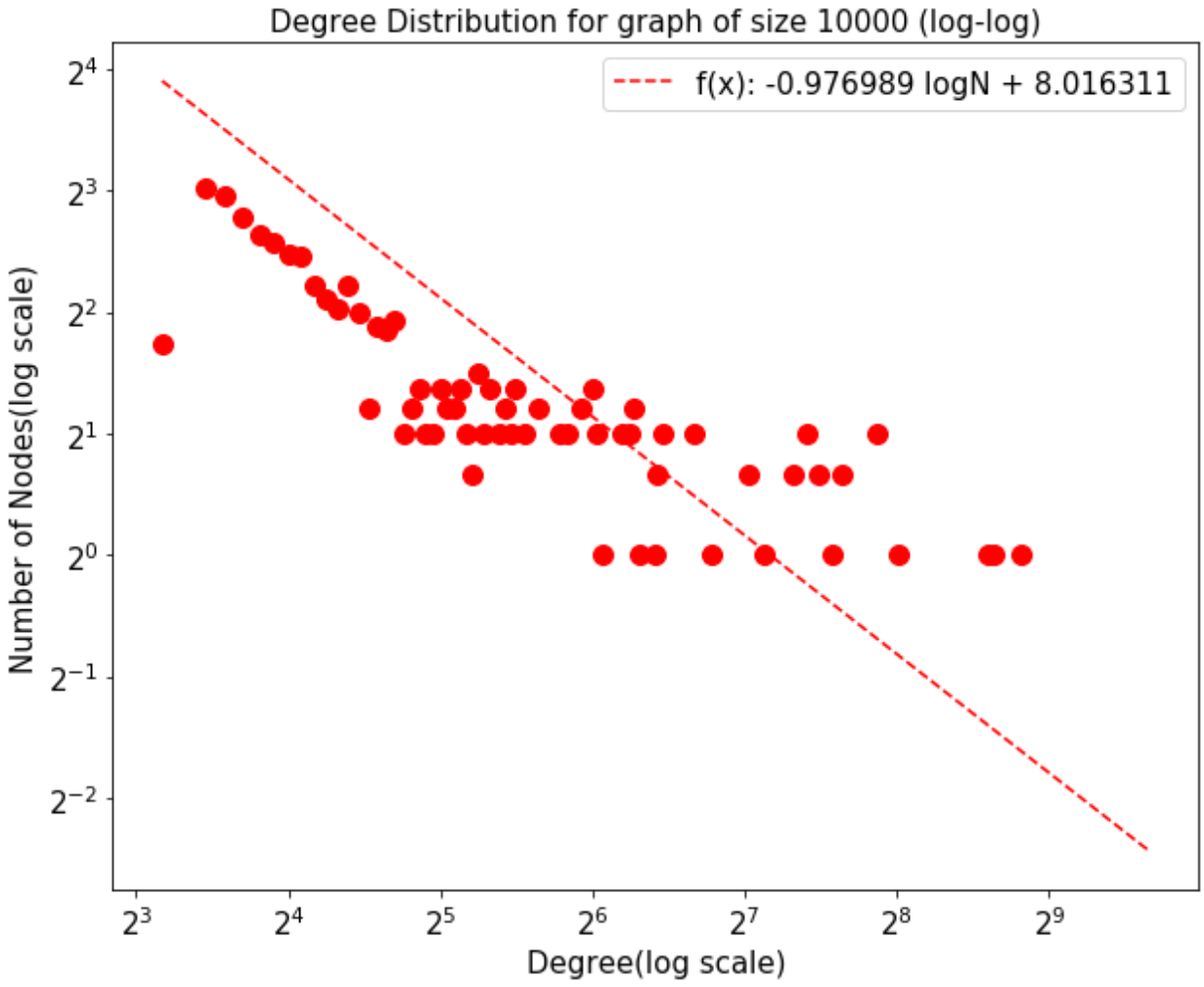
In [9]:
```
# Plotting Degree-distribution(lin-lin)
# N = 10000
f, ax = plt.subplots(1,1, figsize=(10,8))

plt.plot(Degree_dist[45:119,0], Degree_dist[45:119,1], '.',
          markersize=20, color='red')

plt.xlabel('Degree', fontsize=15)
plt.ylabel('Number of nodes', fontsize=15)
```

```
plt.xticks(np.arange(0, 1000, 200), fontsize=15)
plt.yticks(fontsize=15)
plt.title('Degree Distribution for graph of size 10000 (lin-lin)', fontsi
ze=15)
plt.show()
```



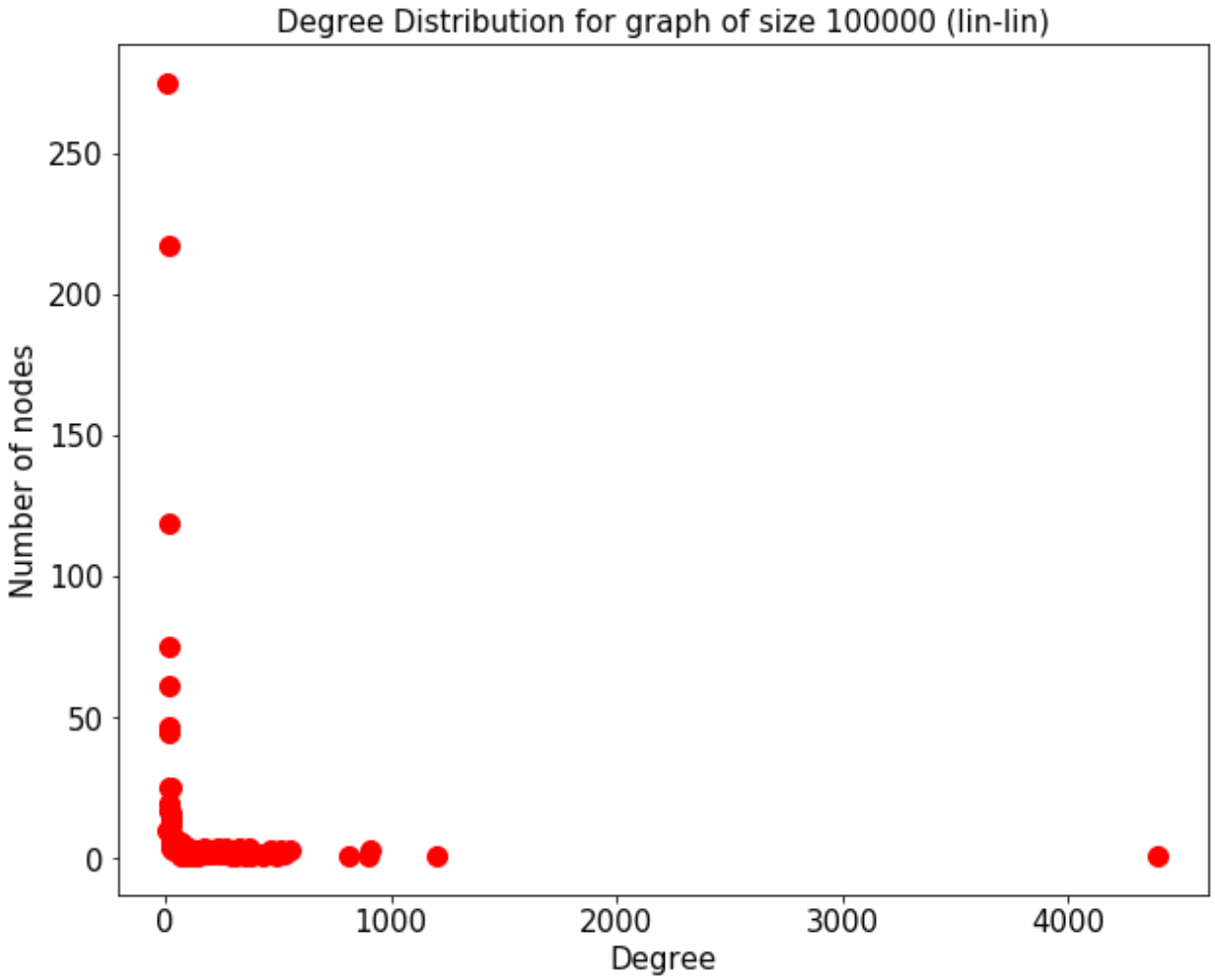
In [10]:
```
# Plotting Degree-distribution (log-log)
f, ax = plt.subplots(1,1, figsize=(10,8))

m1, b1 = np.polyfit(np.log2(Degree_dist[45:119,0]), np.log2(Degree_dist[4
5:119,1]), 1)

plt.loglog(Degree_dist[45:119,0], np.log2(Degree_dist[45:119,1]), '.',bas
ex=2, basey=2,
           markersize=20, color='red')

plt.loglog(Degree_dist[45:119,0], 2**(m1*np.log2(Degree_dist[45:119,0])+b
1),basex=2, basey=2,
         linestyle='--', label=f'f(x): {m1:4f} logN + {b1:4f}', color='re
d')

plt.xlabel('Degree(log scale)', fontsize=15)
plt.ylabel('Number of Nodes(log scale)', fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
```

```
plt.title('Degree Distribution for graph of size 10000 (log-log)', fontsi
ze=15)
plt.legend(fontsize=15)
plt.show()
```



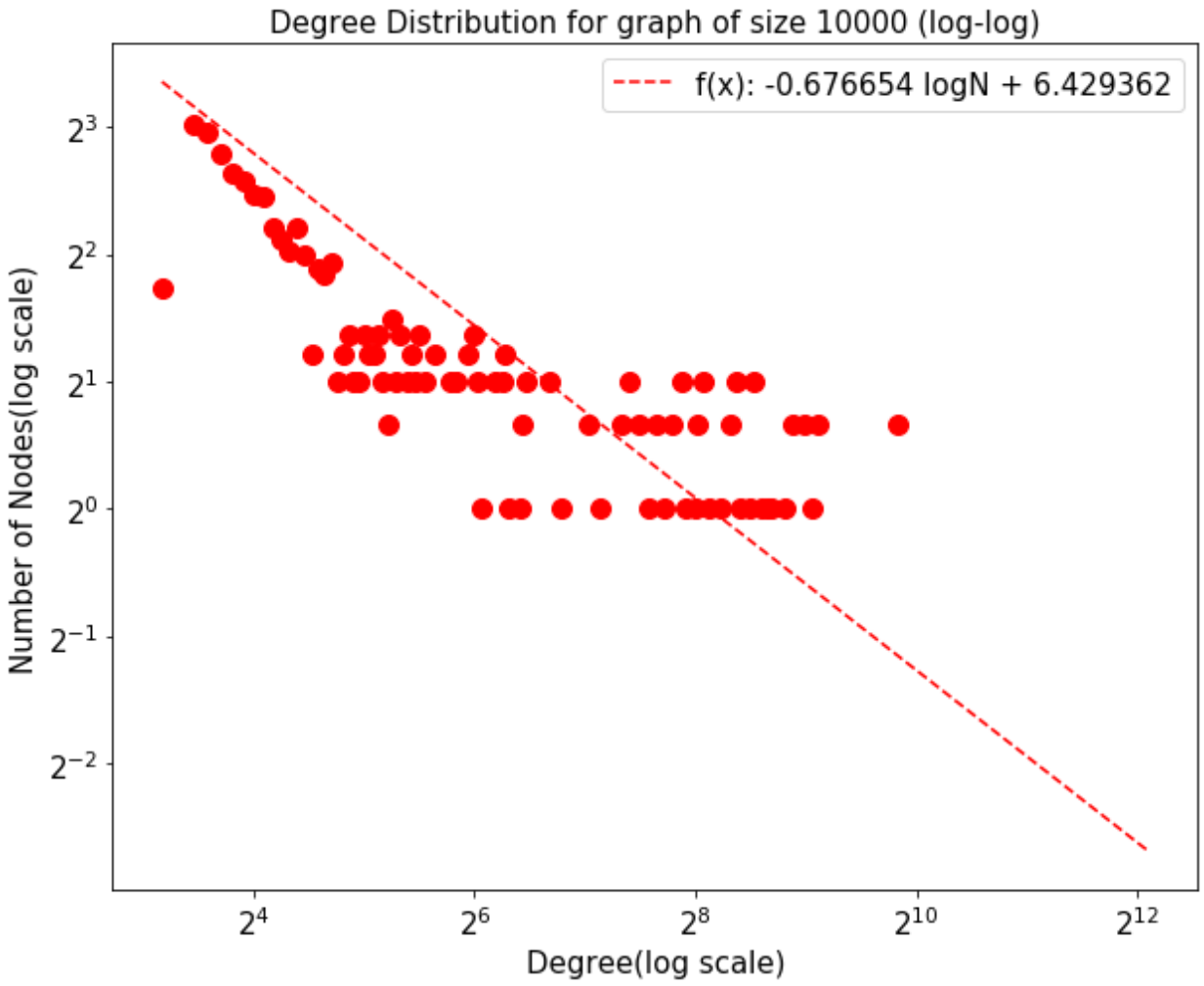
In [11]:
```
# Plotting Degree-distribution(lin-lin)
# N = 100000
f, ax = plt.subplots(1,1, figsize=(10,8))

plt.plot(Degree_dist[119:219,0], Degree_dist[119:219,1], '.',
           markersize=20, color='red')

plt.xlabel('Degree', fontsize=15)
plt.ylabel('Number of nodes', fontsize=15)
plt.xticks(np.arange(0, 5000, 1000), fontsize=15)
```

```
plt.yticks(fontsize=15)
plt.title('Degree Distribution for graph of size 100000 (lin-lin)', fonts
ize=15)
plt.show()
```



In [12]:
```
# Plotting Degree-distribution (log-log)
f, ax = plt.subplots(1,1, figsize=(10,8))

m1, b1 = np.polyfit(np.log2(Degree_dist[119:219,0]), np.log2(Degree_dist[
119:219,1]), 1)

plt.loglog(Degree_dist[119:219,0], np.log2(Degree_dist[119:219,1]), '.',b
asex=2, basey=2,
           markersize=20, color='red')

plt.loglog(Degree_dist[119:219,0], 2**(m1*np.log2(Degree_dist[119:219,0])
+b1),basex=2, basey=2,
         linestyle='--', label=f'f(x): {m1:4f} logN + {b1:4f}', color='re
d')

plt.xlabel('Degree(log scale)', fontsize=15)
plt.ylabel('Number of Nodes(log scale)', fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
```

```
plt.title('Degree Distribution for graph of size 10000 (log-log)', fontsi
ze=15)
plt.legend(fontsize=15)
plt.show()
```



Comments:

From the plots above, we can see that the degree distribution follows the power law distribution. Most of the degrees falls into the bins on the left side, and there is a long tail to the right. Therefore, we say the this random graph follows the power law distributon. The linear regression on the log-log plot can be considered fitting the data. The slope of these lines can be seen on the graph.

Conclusion:

Overall, we have discovered all of the characteristics of random graphs creating by Barabasi-Albert model. Diameter of the graph is proportional to log(N), clustering-coefficient is propotional to log(1/N), and the degree distribution of the graph follows power law distribution.

In [ ]: