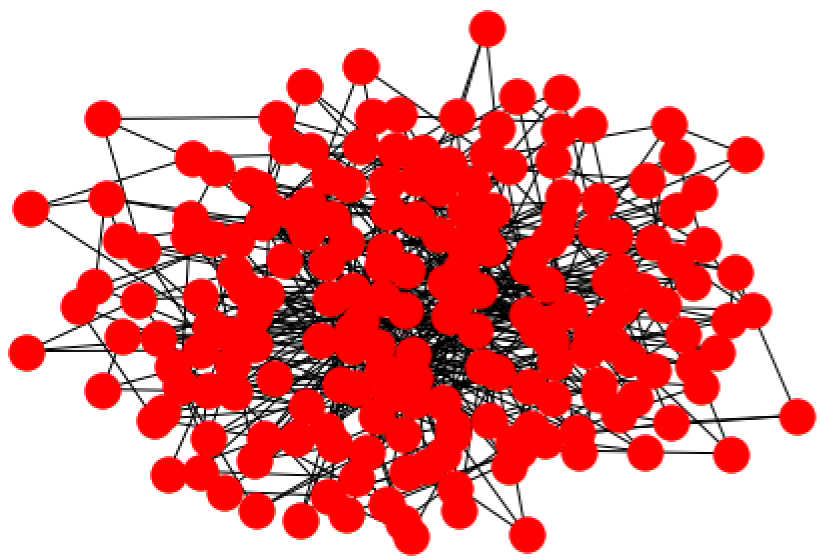
11. Degree Analysis

In [1]:
```
%matplotlib inline
import networkx as nx
import matplotlib.pylab as plt
```

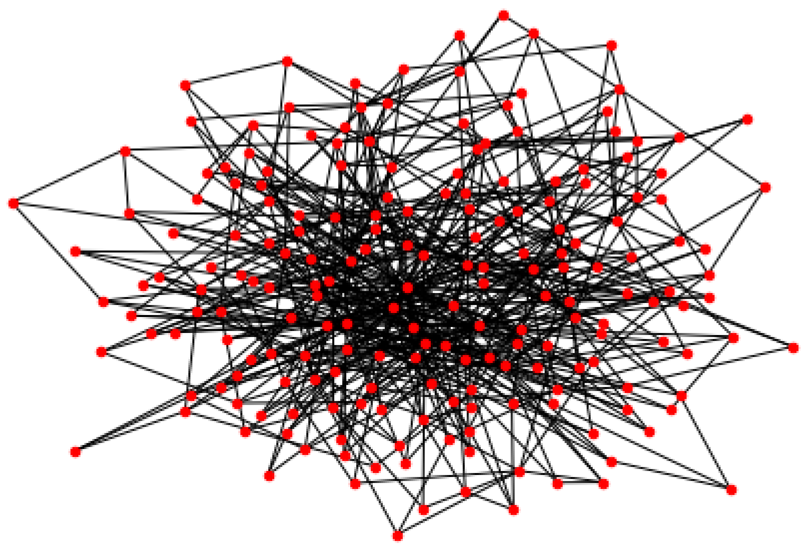
In [2]:
```
G = nx.read_gml('Graphs/rand200.gml')
```

In [3]:
```
nx.draw(G)
```



- The overall graph is hidden by the too-big-nodes.
- Make the nodes smaller.
- In complex network analysis, we handle hundreds and thousands of nodes.
- Smaller nodes are preferred to see the overall graph.

In [4]:
```
nx.draw(G,node_size=20)
```



Degree Histogram

Histogram of vertex degree:

$$h(d) = |\{v \mid \delta(v) = d, v \in V(G)\}|$$

$h(d)$ is the number of vertices having degree d.

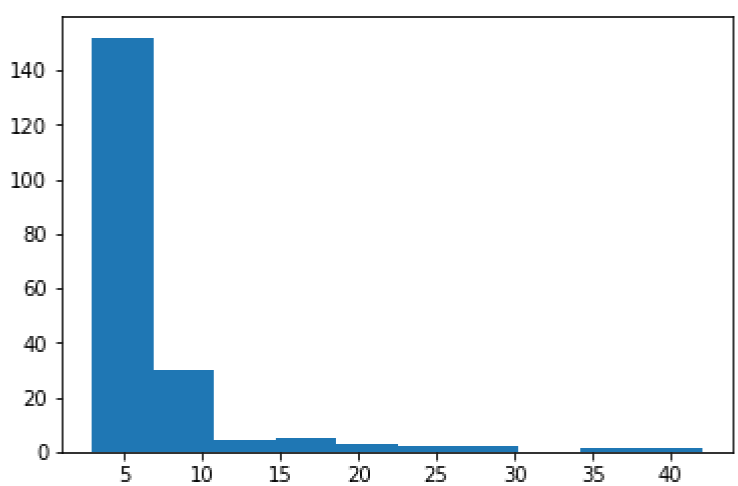
Normalized histogram of vertex degree:

$$p_{DEG} = h(d)/n$$

where n is the number of vertices in the graph.

In [5]:
```
D = dict(nx.degree(G)).values()
plt.hist(D)
```

Out[5]:
```
(array([152.,  30.,   4.,   5.,   3.,   2.,   2.,   0.,   1.,   1.]),
 array([ 3. ,  6.9, 10.8, 14.7, 18.6, 22.5, 26.4, 30.3, 34.2, 38.1, 42. ]),
 <a list of 10 Patch objects>)
```

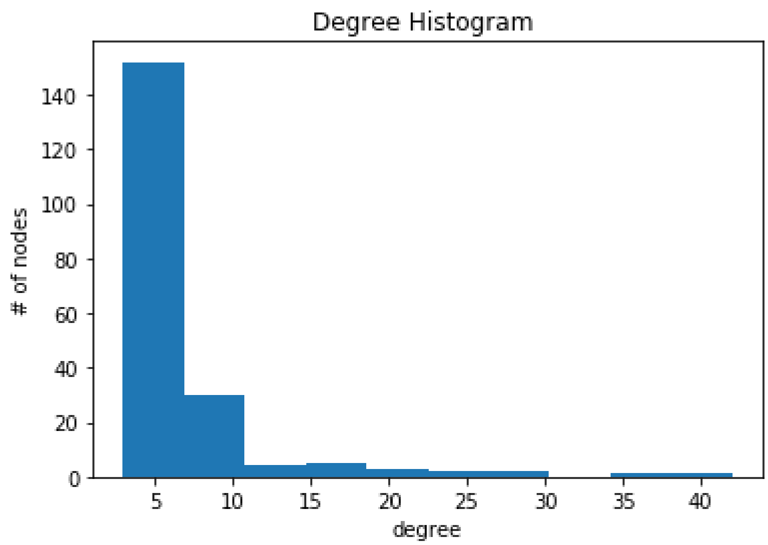


For any chart, give hints on

- what the chart for
- what each axis for

In [6]:
```
plt.hist(D)
plt.title("Degree Histogram ")
plt.ylabel("# of nodes")
plt.xlabel("degree")
```

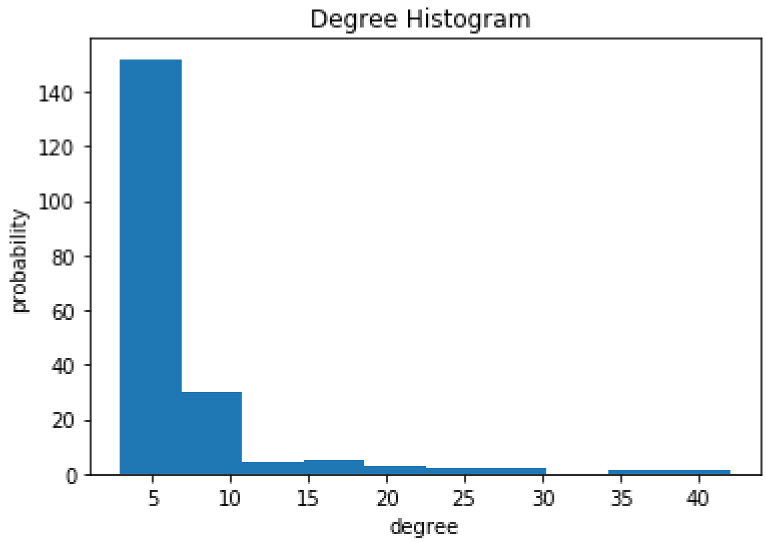
Out[6]: Text(0.5,0,'degree')



For saving lines of code, make a function to display title, ylabel, and xlabel

In [7]:

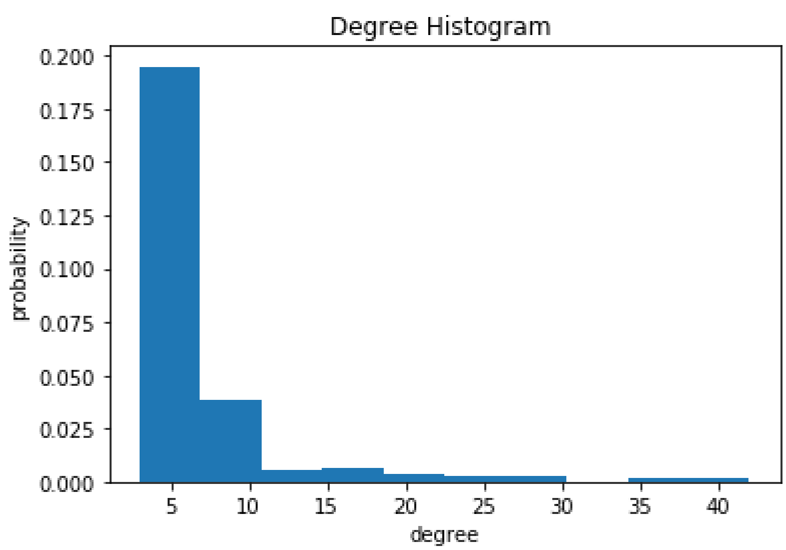
```python
def legend(title, ylabel, xlabel):
    plt.title(title)
    plt.ylabel(ylabel)
    plt.xlabel(xlabel)

plt.hist(D)
legend("Degree Histogram ", "probability", "degree")
```



- we will use this function from now on.
- In many case, a probability density function is preferred.

In [8]:

```python
# normalized histogram, ie. probability density function
plt.hist(D, density=True)
legend("Degree Histogram ", "probability", "degree")
```

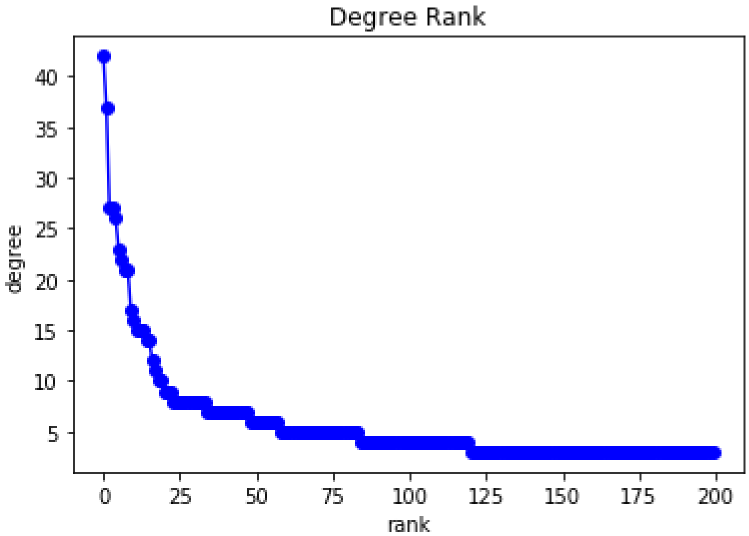


Rank of Vertex Degrees

- sort the nodes on non-decreasing order of degrees.

In [9]:

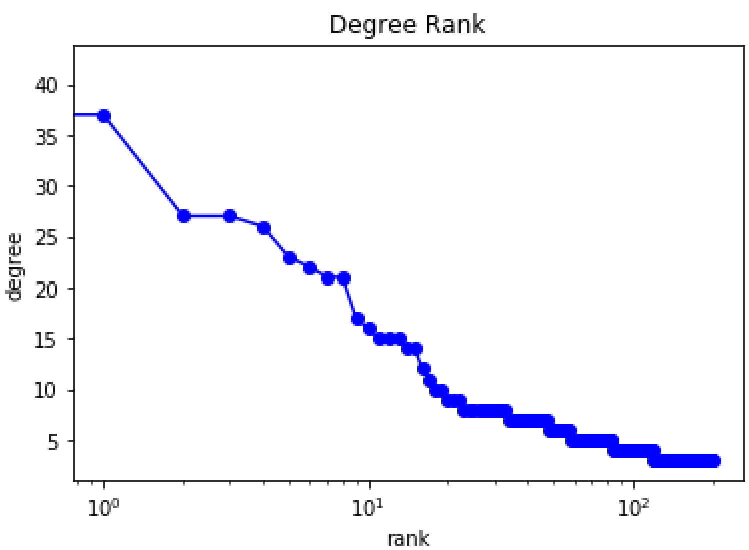
```
D = sorted(dict(nx.degree(G)).values(),reverse=True) # degree sequence
plt.plot(D,'b-',marker='o')
legend("Degree Rank", "degree", "rank")
```



- What is the relationship between x-axis and y-axis?
- It looks logarithmic. Really?
- Try log-scale on x-axis.

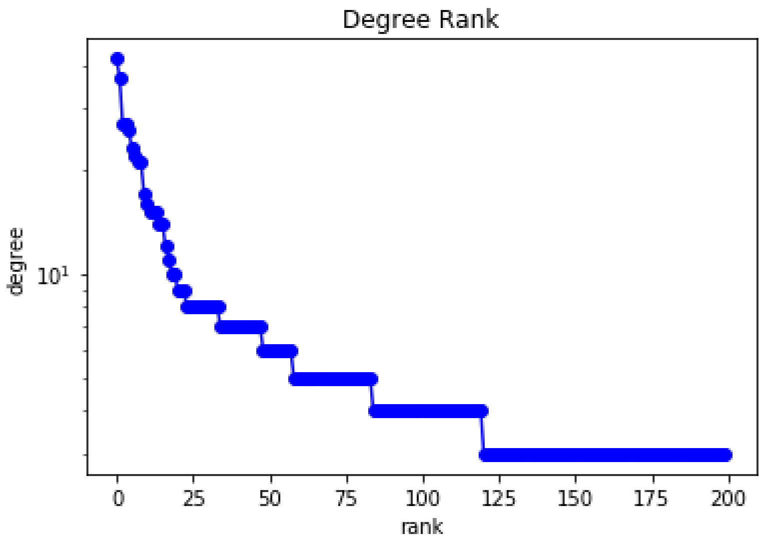
In [10]:

```
plt.semilogx(D,'b-',marker='o')
legend("Degree Rank", "degree", "rank")
```



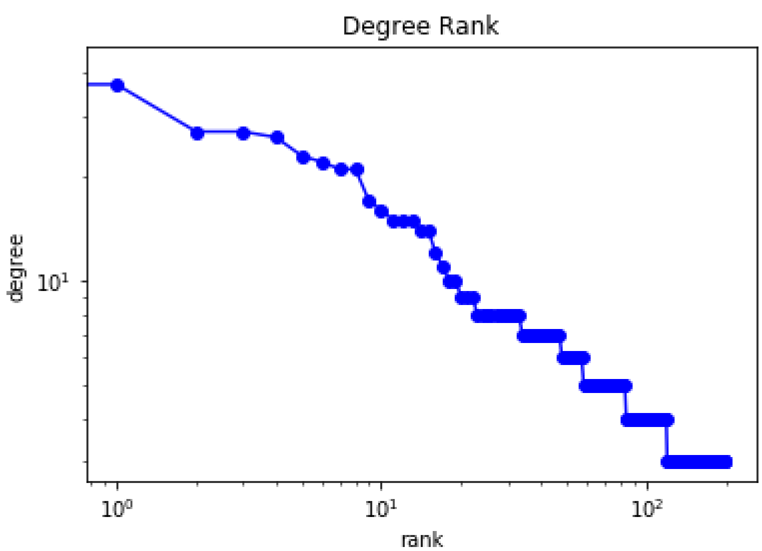
- Maybe logarithmic.
- But, it is still convex.
- Try log-scale on y-axis.

In [11]:
```
plt.semilogy(D,'b-',marker='o')
legend("Degree Rank", "degree", "rank")
```



- But, it is still convex.
- Try log-scale on *both x-axis and y-axis*

In [12]:
```
plt.loglog(D,'b-',marker='o')
legend("Degree Rank", "degree", "rank")
```



- Eureka!! it is much more linear.

Degree Correlations

$$r_{deg}(G) = \frac{\sum_{(i,j)\in E}(d_i - \bar{d})(d_j - \bar{d})}{\sum_{i\in V}(d_i - \bar{d})^2}$$

In [13]:
```
def degree_corr(G):
    import numpy as np
    D = dict(nx.degree(G))
    Dmean = np.mean(list(D.values()))
    sum_e = sum_v = 0.0

    for u,v in G.edges():
        if u < v:
            sum_e += (D[u] - Dmean)*(D[v] - Dmean)

    for u in G.nodes():
        sum_v += (D[u] - Dmean)**2

    return sum_e / sum_v

G = nx.read_gml('Graphs/rand200.gml')
print (degree_corr(G))
```

```
1.3307528123274153
```

Assortativity Mixing

Pearson correlation coefficient is defined as

$$r = \frac{\sum_{jk} jk(e_{jk} - q_j q_k)}{\sigma_q^2}$$

In [14]:
```
nx.degree_assortativity_coefficient(G)
```

Out[14]: `-0.12442103054340635`

In [15]:
```
nx.degree_pearson_correlation_coefficient(G)
# Need SciPy requiring Lapack/blas requiring ...
```

Out[15]: `-0.1244210305434064`

In [16]:
```
nx.degree_pearson_correlation_coefficient??
```

In [17]:
```
nx.degree_assortativity_coefficient??
```

Assortativity Coefficients

network	n	r
physics co-authorship	52,000	0.363
biology co-authorship	1,520,251	0.127
mathematics co co-authorship	253,339	0.120
film actor collaboration	339,913	0.208
company directors	7,673	0.276

Internet	10,697	-0.189
WWW	269,504	-0.065
protein interaction	2,115	-0.156
neural network	307	-0.163
food web	92	-0.276
random graph		0
callaway et al.		$\delta/(1+2\delta)$
Barabasi and Albert		0

from Newman, Mark EJ. "Assortative mixing in networks." Physical review letters 89.20 (2002): 208701.

In [18]:
```
# draw graph in inset
%matplotlib inline
import networkx as nx
import matplotlib.pylab as plt
import numpy as np
from scipy import stats
import math

def mini_draw(G, pos=[1.0, 0.0, 1.0, 1.0]):
    plt.axes(pos)
#    c=sorted(nx.connected_component_subgraphs(G), key = len, reverse=True)[
    layout=nx.spring_layout(G)
    plt.axis('off')
    nx.draw_networkx_nodes(G,layout,node_size=20)
    nx.draw_networkx_edges(G,layout,alpha=0.4)

def degree_histogram(G, mini=False):
    D = list(dict(nx.degree(G)).values())
    x = plt.hist(D)
    plt.title("Degree Histogram ")
    plt.ylabel("# of nodes")
    plt.xlabel("degree")
    if mini:
        mini_draw(G)
    plt.show()
```

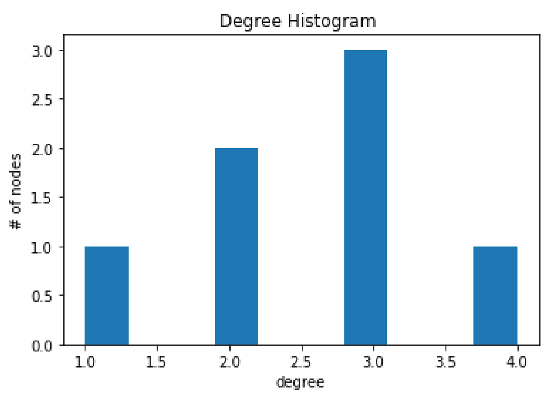
Graph Regularity, or Graph Randomness

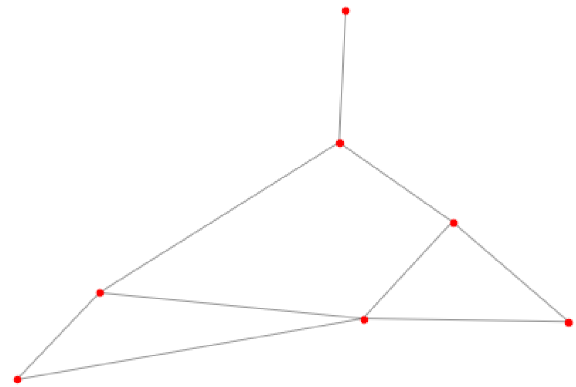
Let us compare the following 4 graphs.

In [19]:

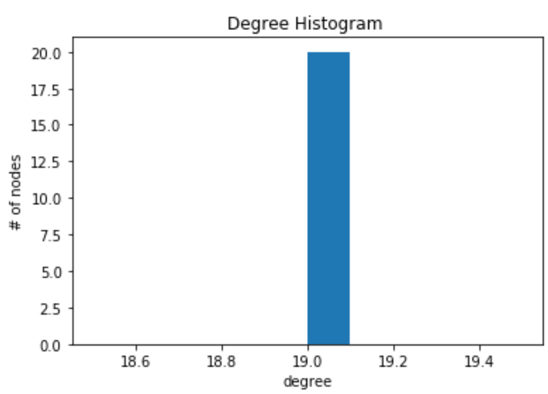
```python
G = {}
filenames = ['Graphs/g35.gml', 'Graphs/k_20.gml', 'data/AttMpls.gml', 'Graph
for k in range(4):
    G[k] = nx.read_gml(filenames[k])
    print ("\t\t\t\t-------- %s --------" % (filenames[k]))
    degree_histogram(G[k], True)
```

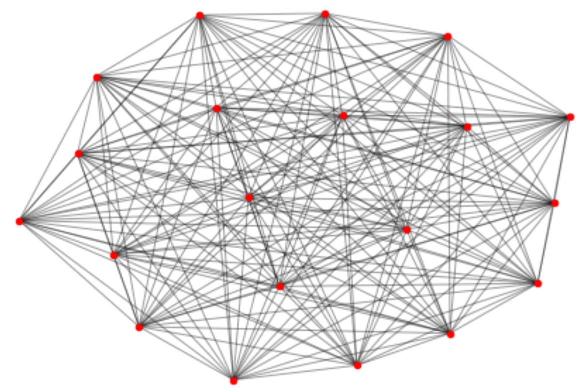
```
-------- Graphs/g35.gml --------
```



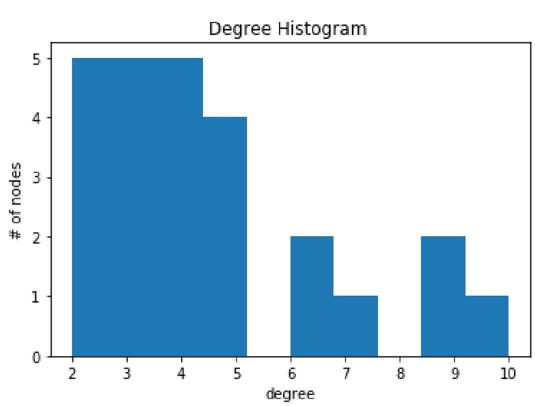


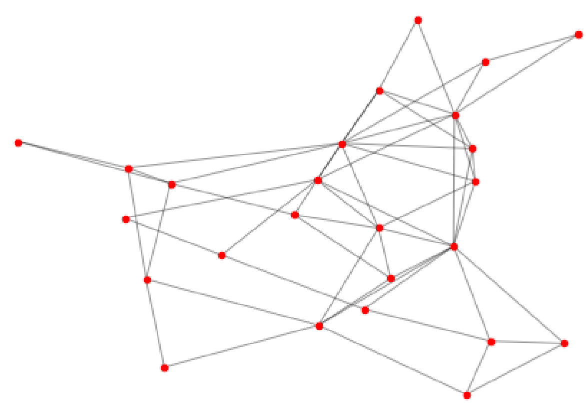
```
-------- Graphs/k_20.gml --------
```



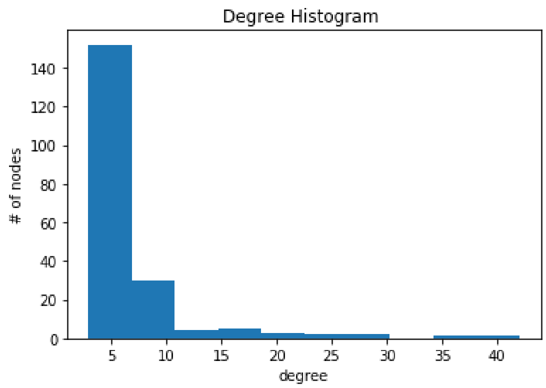


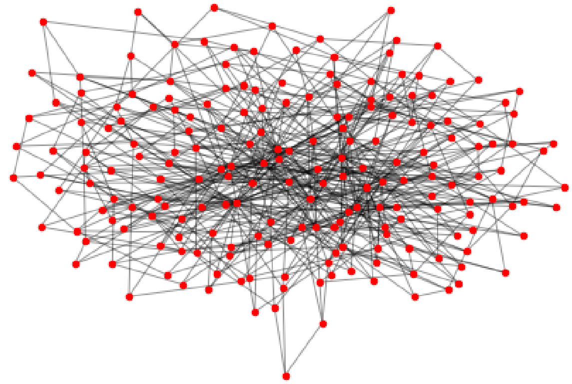
```
-------- data/AttMpls.gml --------
```





```
-------- Graphs/rand200.gml --------
```





Entropy

Entropy is a measure of the number of possible arrangements the atoms in a system can have. In this sense, entropy is a measure of uncertainty or randomness.
Refer thermodynamic entropy in [https://simple.wikipedia.org/wiki/Thermodynamic_entropy (https://simple.wikipedia.org/wiki/Thermodynamic_entropy)](https://simple.wikipedia.org/wiki/Thermodynamic_entropy)

Entropy in Statistics

The information gain is a measure of the probability with which a certain result is expected to happen. In the context of a coin flip, with a 50-50 probability, the entropy is the highest value of 1. It does not involve information gain because it does not incline towards a specific result more than the other. If there is a 100-0 probability that a result will occur, the entropy is 0.
In general, Let X be a discrete random variable with possible values $\{x_1, x_2, \cdots, x_n\}$ and probability mass function $P(x_i)$. The Shannon entropy H is defined as:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log P(x_i)$$

Example) In the above graph, degree sequence X, degree distribution D, or the Probability mass function $P(X)$

$$\begin{aligned} X &= [3, 3, 1, 3, 2, 2, 4] \\ D &= \{(1, 1), (2, 2), (3, 3), (4, 1)\} \\ P(x) &= [1/7, 2/7, 3/7, 1/7] = [0.143, 0.286, 0.428, 0.143] \end{aligned}$$

The entropy $H(X)$ is given as

$$\begin{aligned} H(X) &= -(0.143 \times \log 0.143 + 0.286 \times \log 0.286 + 0.428 \times \log 0.428 + 0.143 \times \log 0.143) \\ &= 0.278 + 0.358 + 0.363 + 0.278 \\ &= 1.277 \end{aligned}$$

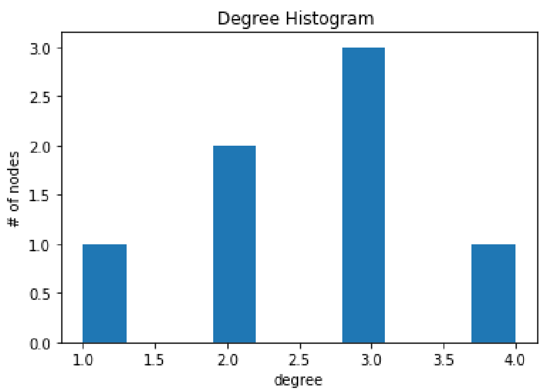
The normalized entropy $H_n(X)$ is given as

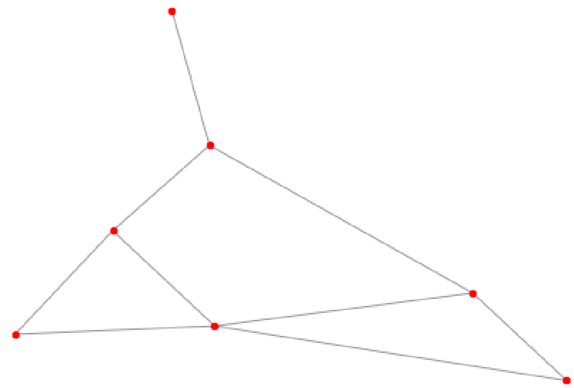
$$H_n(X) = -\sum_{i=1}^{n} \frac{P(x_i) \log P(x_i)}{\log n} = \frac{H(X)}{\log n}$$

In [23]:
```
def degree_entropy(G, normed=False):
    P = np.bincount(list(dict(nx.degree(G)).values()))
    NP = np.nonzero(P)
    N = NP[0].size
    H = stats.entropy(P[NP])
    if normed and N > 1:
        return H / math.log(N)
    else:
        return H
```

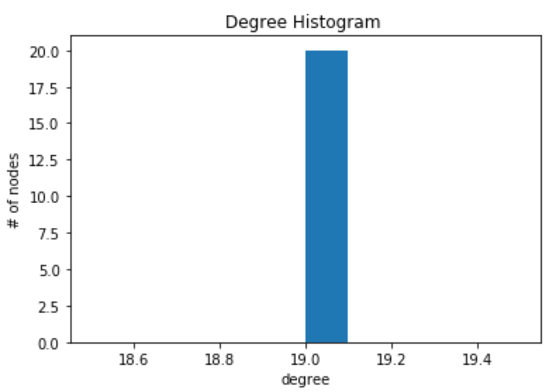
In [24]:
```
for x in G:
    print ('----------------- G[%d], %s' % (x, filenames[x]))
    print ('H[%d]  =' % x, degree_entropy(G[x]))
    print ('Hn[%d] =' % x, degree_entropy(G[x], True))
    degree_histogram(G[x], True)
```

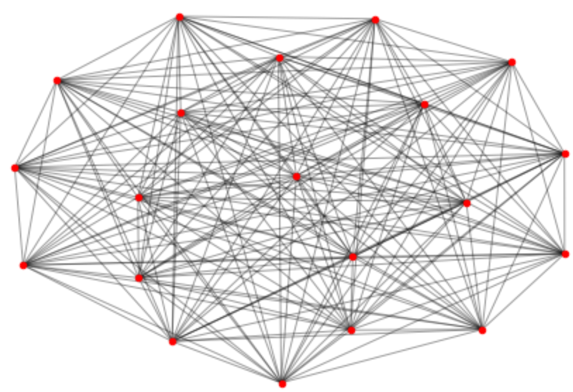
```
----------------- G[0], Graphs/g35.gml
H[0]  = 1.277034259466139
Hn[0] = 0.9211854965885543
```



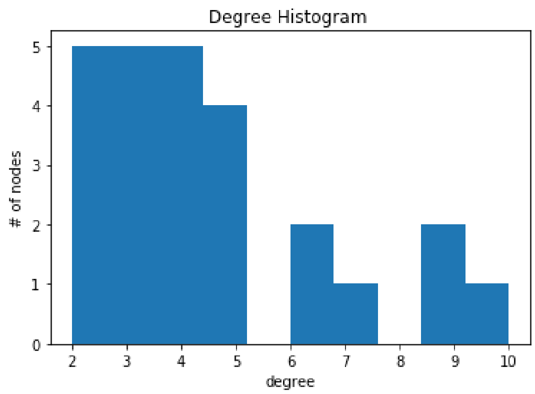


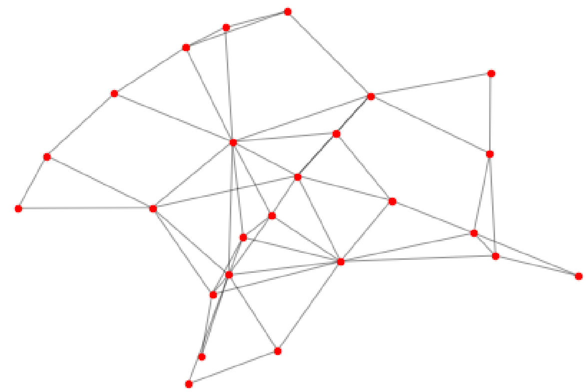
```
----------------- G[1], Graphs/k_20.gml
H[1]  = 0.0
Hn[1] = 0.0
```





```
----------------- G[2], data/AttMpls.gml
H[2]  = 1.920502430738967
Hn[2] = 0.9235664442807693
```





```
----------------- G[3], Graphs/rand200.gml
H[3]  = 1.9844889902044696
Hn[3] = 0.651822750791876
```