Graph Algorithm Experiments

Andrew Self

In this project I implemented a C++ algorithm to create Barabasi-Albert graphs. Then I implemented C++ algorithms to get the diameter, clustering coefficient, and degree distribution of graphs of different sizes.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
%matplotlib inline
```

In [4]:
```
def getMeans(df):
    means = []
    vals = pd.unique(df["Size"])
    #print(vals)
    for i in vals:
        temp = df.loc[df['Size'] == i]
        #print(i)
        #print(temp["Time"].mean())
        means.append(temp["Info"].mean())
    return means, vals
```

In [6]:
```
def getLog(sizes):
    #logTime = [math.log2(x) if x > 0.0 else math.log2(0.01) for x in times]
    logSize = [math.log2(x) for x in sizes]
    #return logTime, logSize
    return logSize
```

Barabasi-Albert Graphs

These are graphs that are made by the Barabasi-Albert algorithm, which generates scale-free graphs, meaning they have a power-law degree distribution. Put another way, this means that the more connected a node is, the more likely it is to receive new links.

Diameter

The diameter of a graph is the maximum distance between a pair of vertices.

I calculated this by repeatedly using bfs from one node to another and tracking the max depth of the search.

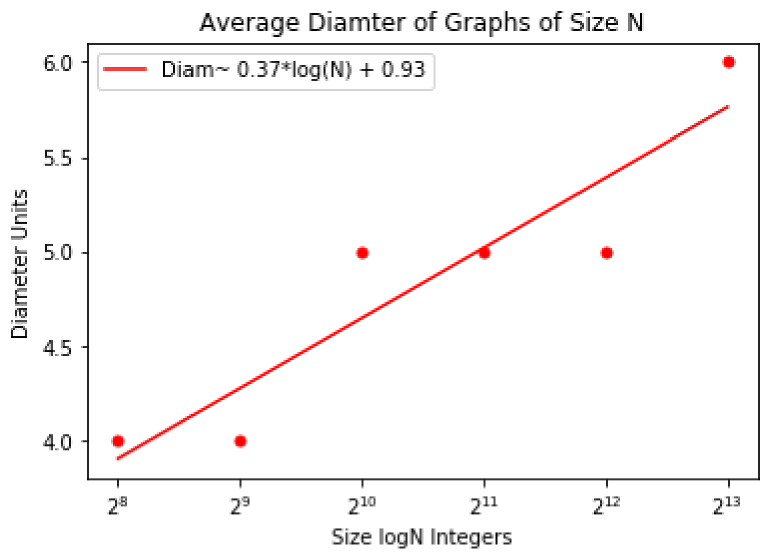
In [8]:
```
df = pd.read_csv('data/diameter.csv', sep=' ')
diams, sizes = getMeans(df)
logSize = getLog(sizes)
m, b = np.polyfit(logSize,diams,1)
print(m,b)
```

```
0.3714285714285712 0.9333333333333337
```

In [13]:
```
plt.semilogx(sizes, diams, '.', basex=2, markersize=10, color='red')
fn = [(m * math.log2(x) + b) for x in sizes]
plt.semilogx(sizes, fn, basex=2, color='red', label='Diam~ 0.37*log(N) + 0.93')
# plt.show()

plt.xlabel('Size logN Graph')
plt.ylabel('Diameter Units')
plt.title('Average Diamter of Graphs of Size N')
plt.legend()
```

Out[13]: <matplotlib.legend.Legend at 0x11a71f710>



As the graph shows, the diameter grows as a function of N. Furthermore, it appears the diameter grows slower than log(N). As N grows from 2^8 to 2^13, the diameter only grows from 4 to 6, which is slower.

Clustering Coefficient

The clustering coefficient of a graph is the a measure of the degree to which nodes in a graph cluster together. I calculate it using the formula 3 * (# of triangles) / (# of 2 edge paths).

The # of 2 edge paths is the sum of deg(v)*deg(v)-1 / 2 for all vertices. The number of triangle can be calculated by making a d-degeneracy ordering and check for the existance of edges with nodes earlier in the ordering.

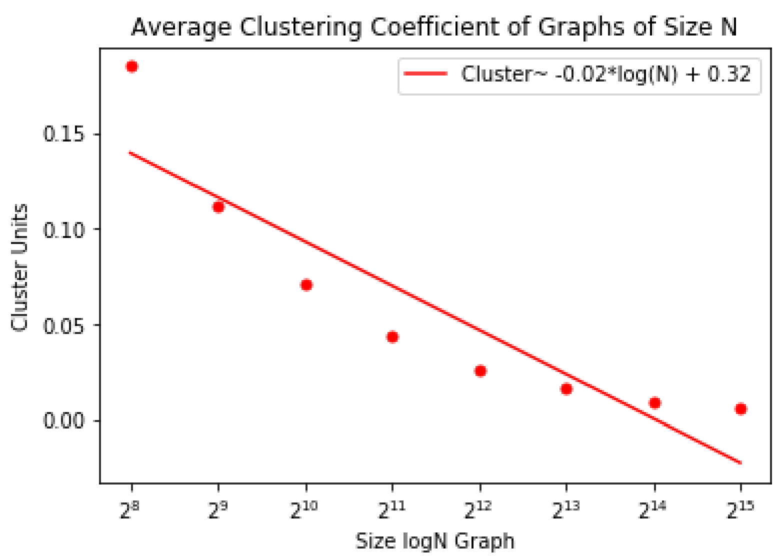
In [19]:
```
df = pd.read_csv('data/cluster.csv', sep=' ')
clust, sizes = getMeans(df)
logSize = getLog(sizes)
m, b = np.polyfit(logSize,clust,1)
print(m,b)
```

```
-0.023210410142857144 0.32552449164285724
```

In [20]:
```
plt.semilogx(sizes, clust, '.', basex=2, markersize=10, color='red')
fn = [(m * math.log2(x) + b) for x in sizes]
plt.semilogx(sizes, fn, basex=2, color='red', label='Cluster~ -0.02*log(N) + 0.32')
# plt.show()

plt.xlabel('Size logN Graph')
plt.ylabel('Cluster Units')
plt.title('Average Clustering Coefficient of Graphs of Size N')
plt.legend()
```

Out[20]: `<matplotlib.legend.Legend at 0x11a8550d0>`



As seen in the graph, the clustering coefficient decreases as a function of N. Also, these values decrease slower than logN. While N goes from 2^8 to 2^15, the cluster coefficient only drops by about 0.14.

In [23]:
```
def getDegDist(df):
    degs = df['Deg']
    nums = df['Num']
    return degs, nums
```

Degree Distribution

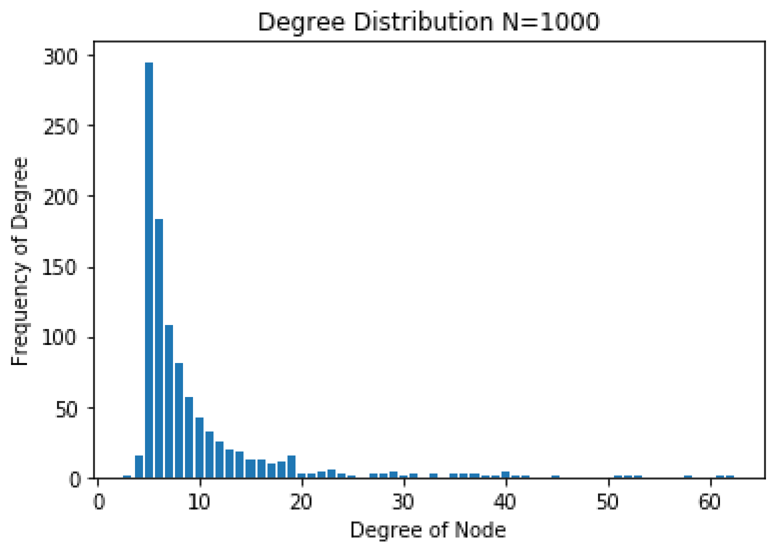
The degree distribution is the number of the nodes for each degree of node in a graph. It can be calculated by using and histogram H of size n and incrementing H[deg(v)] for each vertex.

n = 1000

In [89]:
```
df = pd.read_csv('data/1kB.csv', sep=' ')
degs, nums = getDegDist(df)
plt.bar(degs,nums)

plt.xlabel('Degree of Node')
plt.ylabel('Frequency of Degree')
plt.title('Degree Distribution N=1000')
# plt.legend()
```

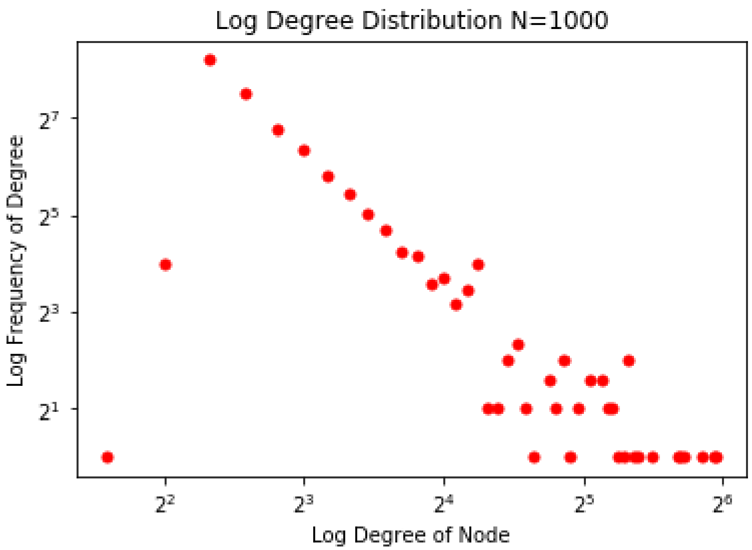
Out[89]: Text(0.5, 1.0, 'Degree Distribution N=1000')



In [68]:
```
plt.loglog(degs, nums, '.', basex=2,basey=2, markersize=10, color='r')

plt.xlabel('Log Degree of Node')
plt.ylabel('Log Frequency of Degree')
plt.title('Log Degree Distribution N=1000')
```

Out[68]:
```
Text(0.5, 1.0, 'Log Degree Distribution N=1000')
```



There is a power law.

In [85]:
```
logD = getLog(degs)
logN = getLog(nums)
m, b = np.polyfit(logD,logN,1)
print("Slope/Exponent:  ", m)
```

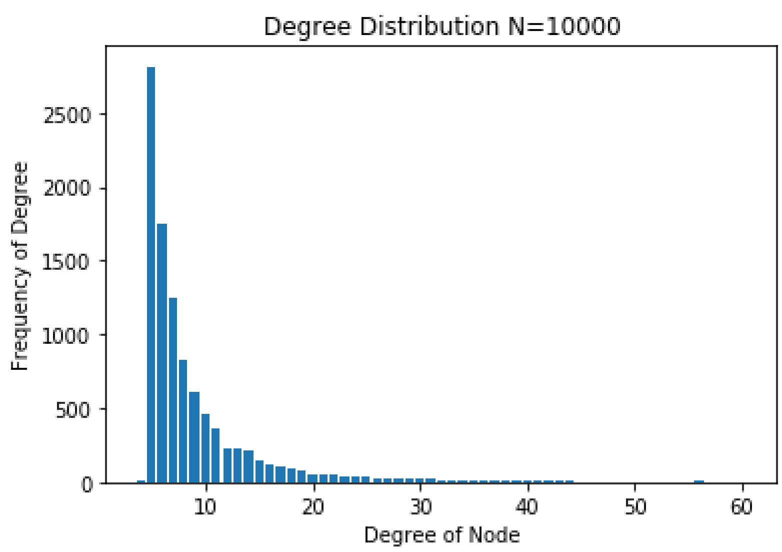
```
Slope/Exponent:   -1.6474096312536026
```

n = 10000

In [75]:
```
df = pd.read_csv('data/10kB.csv', sep=' ')
degs2, nums2 = getDegDist(df)
plt.bar(degs2,nums2)

plt.xlabel('Degree of Node')
plt.ylabel('Frequency of Degree')
plt.title('Degree Distribution N=10000')
# plt.legend()
```

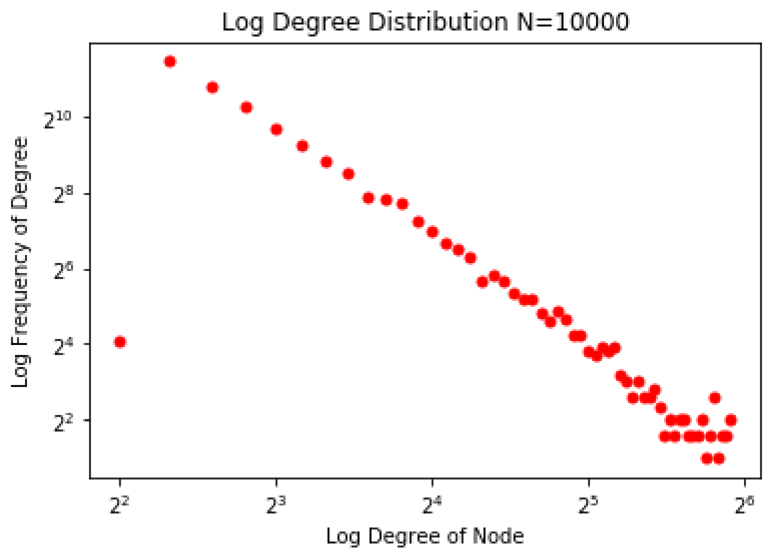
Out[75]:
```
Text(0.5, 1.0, 'Degree Distribution N=10000')
```



In [76]:
```
plt.loglog(degs2, nums2, '.', basex=2,basey=2, markersize=10, color='r')

plt.xlabel('Log Degree of Node')
plt.ylabel('Log Frequency of Degree')
plt.title('Log Degree Distribution N=10000')
```

Out[76]:
```
Text(0.5, 1.0, 'Log Degree Distribution N=10000')
```



There is a power law.

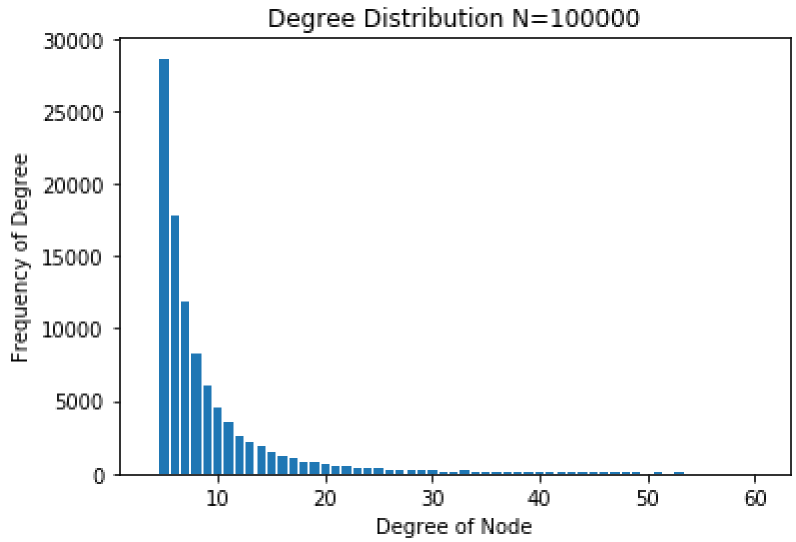
In [86]:
```
logD2 = getLog(degs2)
logN2 = getLog(nums2)
m, b = np.polyfit(logD2,logN2,1)
print("Slope/Exponent:  ", m)
```

```
Slope/Exponent:   -2.526245303416841
```

n = 100000

In [73]:
```
df = pd.read_csv('data/100kB.csv', sep=' ')
degs3, nums3 = getDegDist(df)
plt.bar(degs3,nums3)
# print(degs3)
plt.xlabel('Degree of Node')
plt.ylabel('Frequency of Degree')
plt.title('Degree Distribution N=100000')
# plt.legend()
```

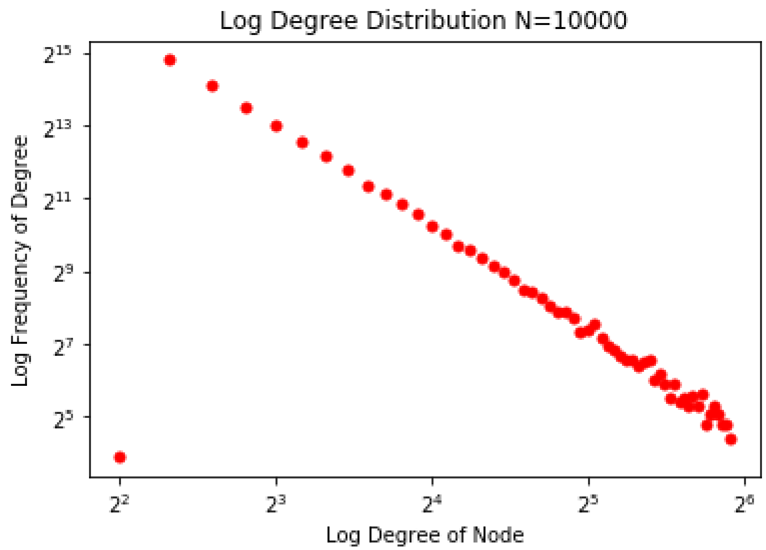
Out[73]:
```
Text(0.5, 1.0, 'Degree Distribution N=100000')
```



In [74]:
```
plt.loglog(degs3, nums3, '.', basex=2,basey=2, markersize=10, color='r')

plt.xlabel('Log Degree of Node')
plt.ylabel('Log Frequency of Degree')
plt.title('Log Degree Distribution N=10000')
```

Out[74]:
```
Text(0.5, 1.0, 'Log Degree Distribution N=10000')
```



There is a power law.

In [87]:
```
logD3 = getLog(degs3)
logN3 = getLog(nums3)
m, b = np.polyfit(logD3,logN3,1)
print("Slope/Exponent:  ", m)
```

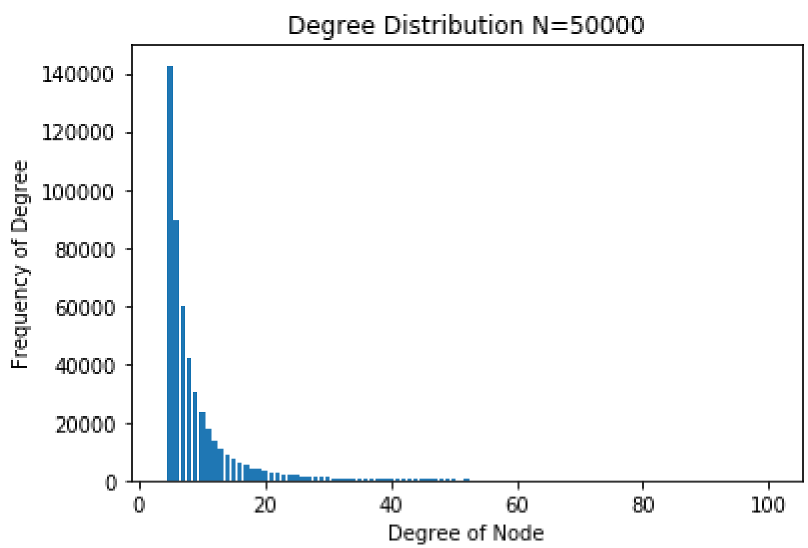
```
Slope/Exponent:   -2.270125737569347
```

n = 500000

In [80]:
```
df = pd.read_csv('data/500kB.csv', sep=' ')
degs4, nums4 = getDegDist(df)
plt.bar(degs4,nums4)

plt.xlabel('Degree of Node')
plt.ylabel('Frequency of Degree')
plt.title('Degree Distribution N=50000')
# plt.legend()
```

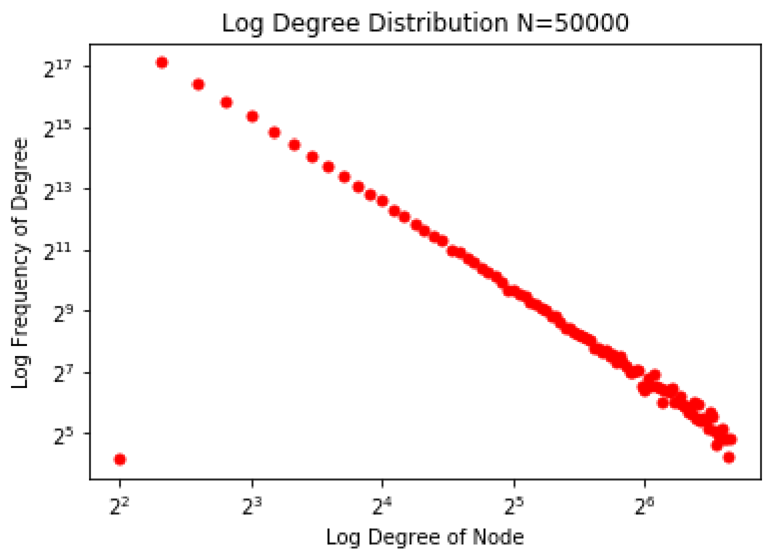
Out[80]:
```
Text(0.5, 1.0, 'Degree Distribution N=50000')
```



In [81]:
```
plt.loglog(degs4, nums4, '.', basex=2,basey=2, markersize=10, color='r')

plt.xlabel('Log Degree of Node')
plt.ylabel('Log Frequency of Degree')
plt.title('Log Degree Distribution N=50000')
```

Out[81]:
```
Text(0.5, 1.0, 'Log Degree Distribution N=50000')
```



There is a power law.

In [88]:
```
logD4 = getLog(degs4)
logN4 = getLog(nums4)
m, b = np.polyfit(logD4,logN4,1)
print("Slope/Exponent:  ", m)
```

```
Slope/Exponent:   -2.483308133140854
```

From these graphs we can see that the degree distribution maintins a power law

as the number of vertices grows.

In [ ]: