Problem1

January 25, 2023

```
In [2]: import fetcher3 as fetch
        import networkx as nx
        import matplotlib.pyplot as plt
        import time
        from tqdm import tqdm
        import numpy as np
        import pickle
```

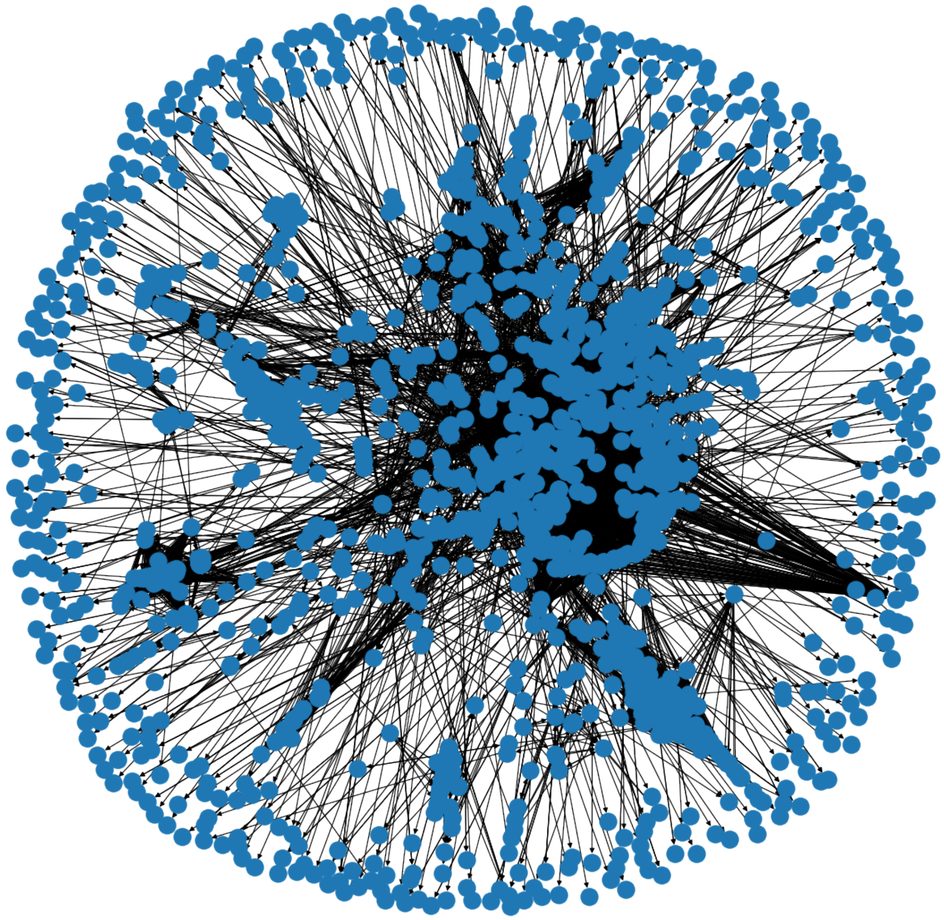
```
In [3]: def save_graph(graph,file_name):
            #initialze Figure
            plt.figure(num=None, figsize=(20, 20), dpi=80)
            plt.axis('off')
            fig = plt.figure(1)
            pos = nx.spring_layout(graph)
            nx.draw_networkx_nodes(graph,pos)
            nx.draw_networkx_edges(graph,pos)
            # nx.draw_networkx_labels(graph,pos)


            plt.savefig(file_name + ".jpg",bbox_inches="tight")
            with open(file_name + ".pickle", 'wb') as handle:
                pickle.dump(graph, handle)
```

```
In [ ]:
```

```
In [8]: # Does not do the "only add ones you crawl"
        stack = [("", "https://www.caltech.edu/")]
        visited = set()
        G = nx.DiGraph()
        with tqdm(total=2000) as pbar:
            while stack and G.number_of_nodes() < 2000:
                parent, url = stack.pop()
                if url in visited:
                    if parent:
                        G.add_edge(parent, url)
                    continue
                visited.add(url)
```

```python
            G.add_node(url)
            pbar.update(1)
            if parent:
                G.add_edge(parent, url)
            links = fetch.fetch_links(url)
            if not links:
                continue
            links = list(filter(lambda x: "caltech.edu" in x, links))
            edges = [(url, link) for link in links]
            stack.extend(edges)

            # print("num_nodes: {}".format(G.number_of_nodes()))
            # time.sleep(0.5)

    save_graph(G,"web_graph_full")
```

```
100%|| 2000/2000 [11:58<00:00,  2.78it/s]
```



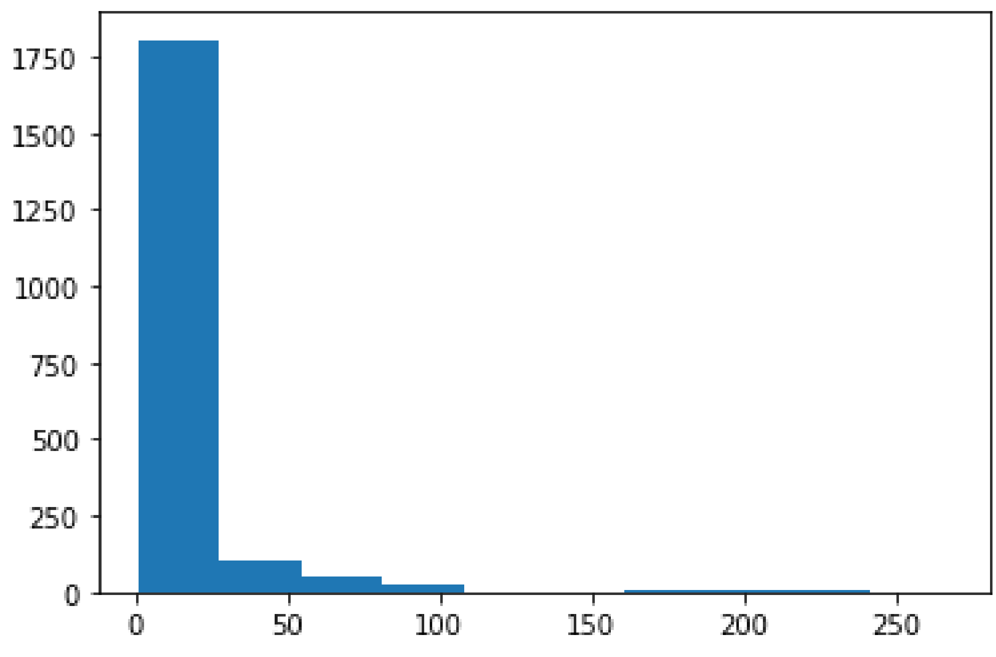
The selection policy is primarily whether the domain is caltech.edu. Besides that, I make sure I don't re-crawl already visited nodes, but if a node I visited shows up again I will add the theoretical new edge, since if I'm seeing a node again it's because it's from a parent that links to it that did not exist before, so the edge is new. If a node does not have any links, it's a leaf and I continue.

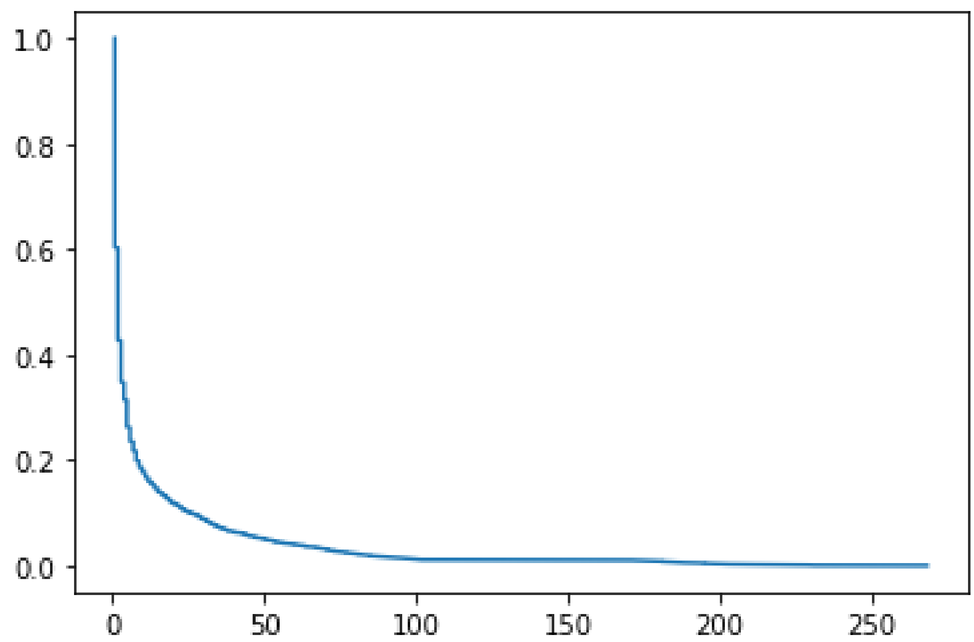
Advantages: Simple, keeps us within the caltech domain for the most part (aside from edge cases of a query having caltech.edu within it which shouldn't really happen?)

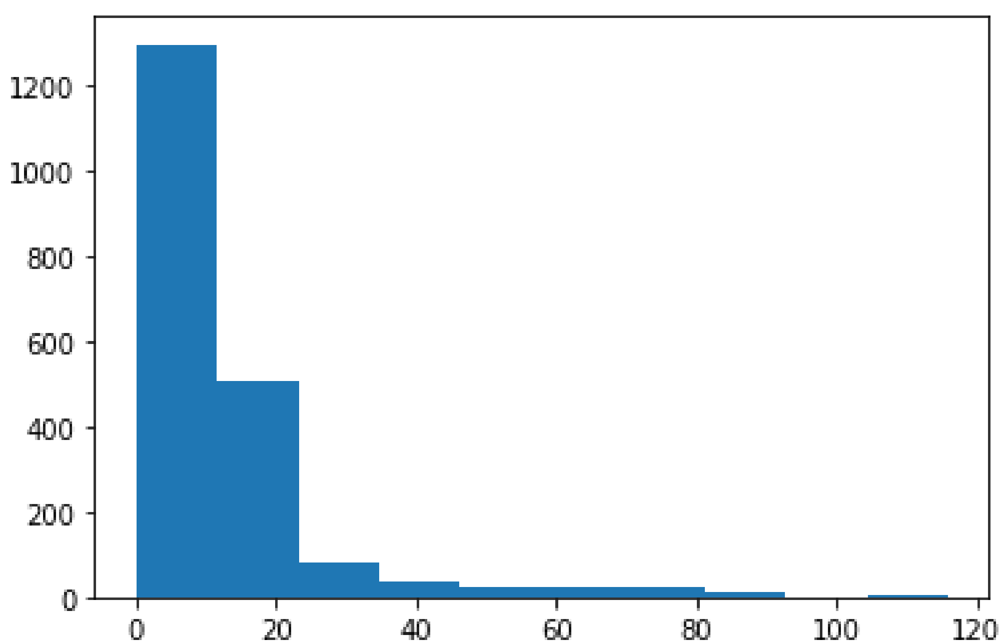
Disadvantages: Sees plenty of useless pages, keeps revisiting certain pages (division home pages, actual home page, etc.), does not properly filter out dynamic pages (there are enough other pages it's not a problem with only 2000 nodes). It also does not deal with http vs https and caltech.edu vs caltech.edu/ and similar cases.

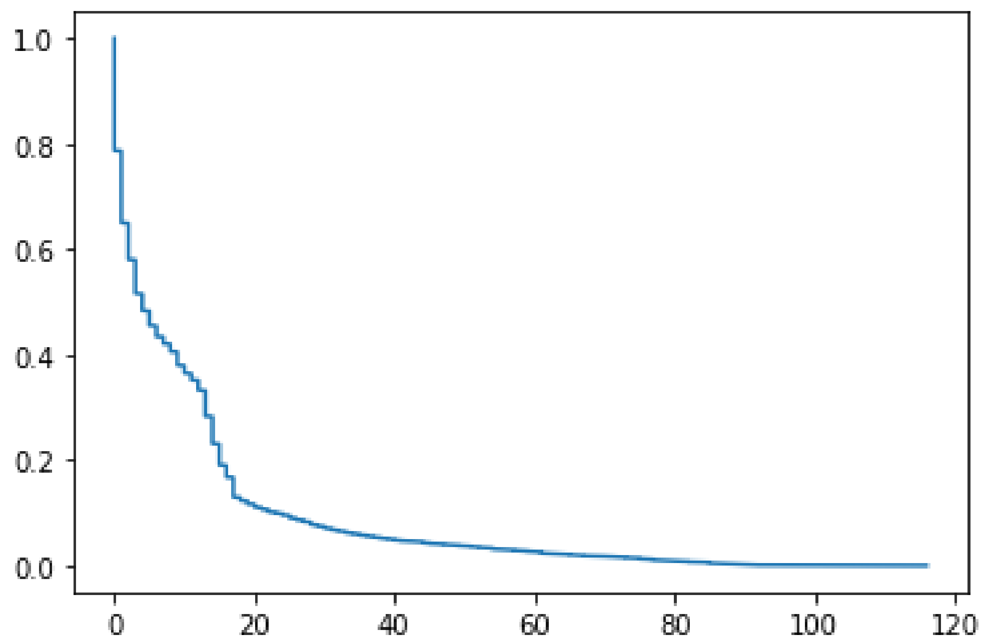
```
In [12]: def plot_in_degree_dist(G):
             in_degrees = np.array([G.in_degree(n) for n in G.nodes()])
             plt.hist(in_degrees)
             plt.show()
```

```
    x = np.concatenate([np.sort(list(in_degrees))])
    plt.plot(x, np.linspace(1, 0, len(x)))
    plt.show()

def plot_out_degree_dist(G):
    out_degrees = np.array([G.out_degree(n) for n in G.nodes()])
    plt.hist(out_degrees)
    plt.show()

    # cdf = out_degrees.cumsum() / out_degrees.sum()
    # ccdf = 1 - cdf
    # plt.plot(range(len(out_degrees)), ccdf)
    # plt.show()
    x = np.concatenate([np.sort(list(out_degrees))])
    plt.plot(x, np.linspace(1, 0, len(x)))
    plt.show()

plot_in_degree_dist(G)
plot_out_degree_dist(G)
```




1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250


1200
1000
800
600
400
200
0
0
20
40
60
80
100
120




```
In [10]: undirected_G = G.to_undirected()
         avg_cluster = nx.average_clustering(undirected_G)
         overall_cluster = nx.transitivity(undirected_G)

         print("avg clustering coeff: {}\n overall clustering coeff: {}".format(avg_cluster, o
```

```
avg clustering coeff: 0.4666830625426355
 overall clustering coeff: 0.5191001325426614
```

```
In [11]: max_diameter = nx.diameter(undirected_G)
         avg_diameter = nx.average_shortest_path_length(undirected_G)

         print("avg diameter: {}\n max diameter: {}".format(avg_diameter, max_diameter))
```

```
avg diameter: 5.6658024012006
 max diameter: 18
```

My clustering coefficients are pretty similar to problem 2's in that both graphs are very highly clustered, but mine is slightly less so. The average and max diameters are also extremely similar. Both graphs definitely have a large component, and ahve small diameters. They also definitely have high clustering coefficients and are heavy tailed.

Problem2

January 25, 2023

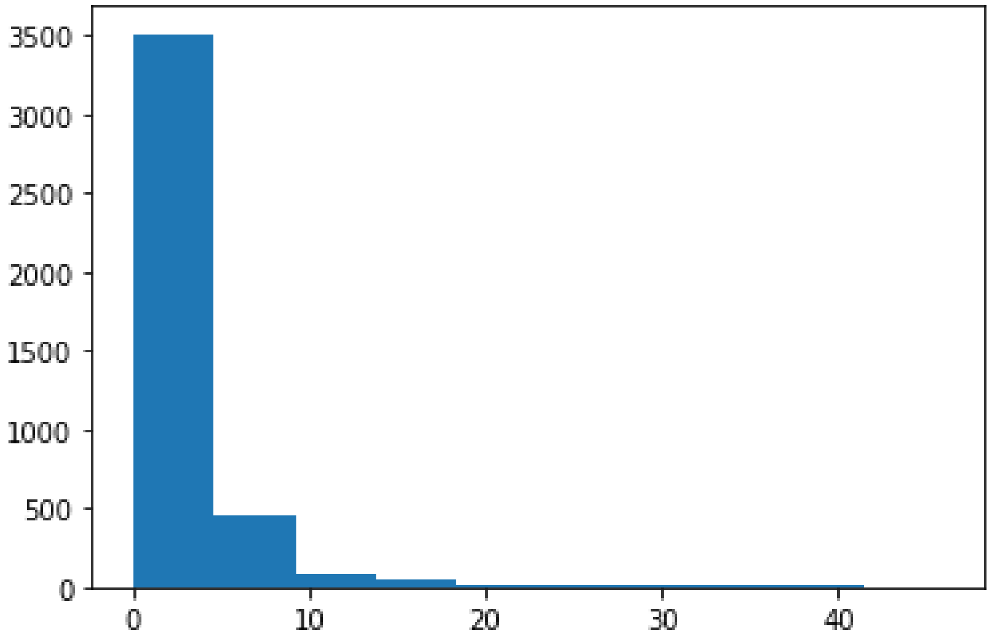
```
In [40]: import networkx as nx
         import matplotlib.pyplot as plt
         import numpy as np
         import collections
```

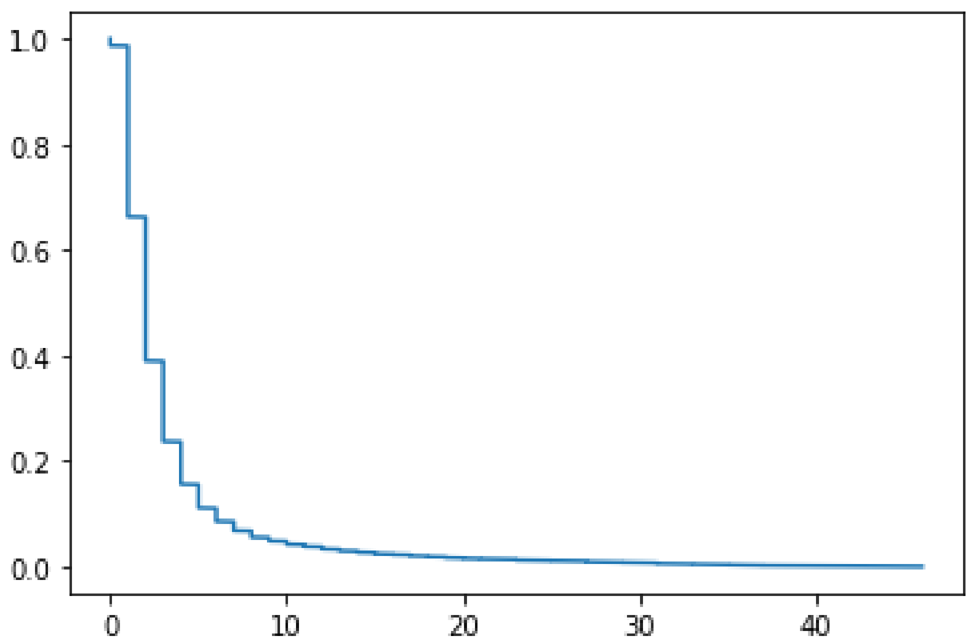
```
In [41]: G = nx.DiGraph()
         with open("gr_qc_coauthorships.txt", "r") as file:
             lines = file.read().splitlines()
             for line in lines:
                 v1, v2 = line.split(' ')
                 G.add_nodes_from([v1, v2])
                 G.add_edge(v1, v2)
```

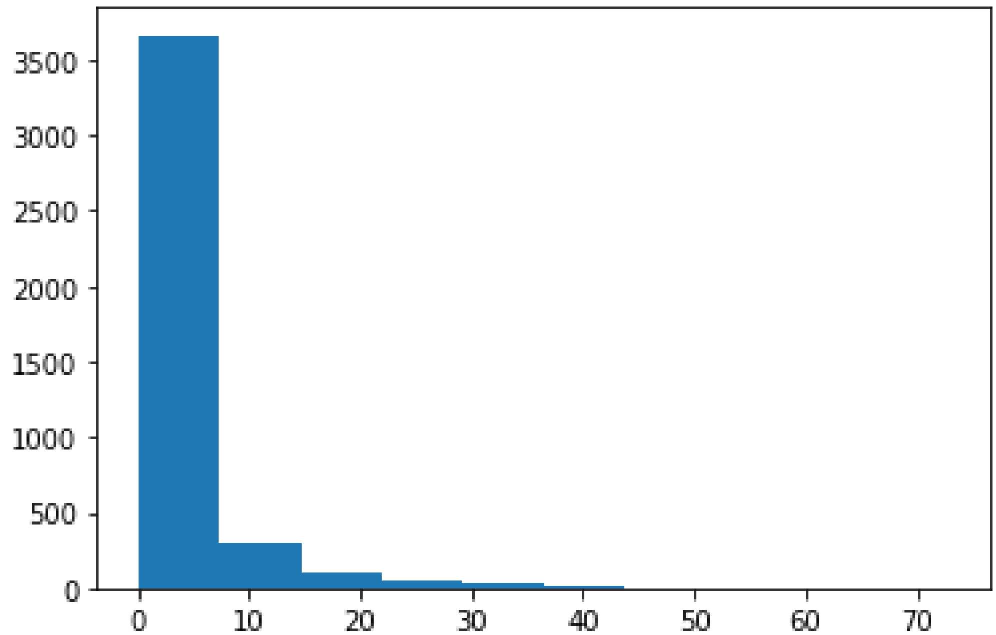
1 A.

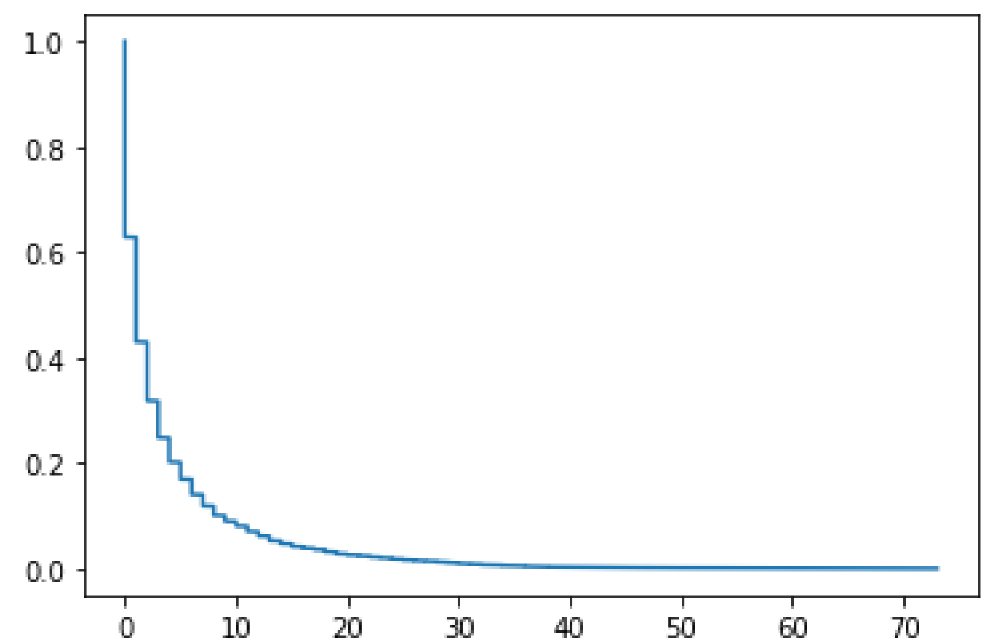
```
In [45]: def plot_in_degree_dist(G):
             in_degrees = np.array([G.in_degree(n) for n in G.nodes()])
             plt.hist(in_degrees)
             plt.show()

             x = np.concatenate([np.sort(list(in_degrees))])
             plt.plot(x, np.linspace(1, 0, len(x)))
             plt.show()

         def plot_out_degree_dist(G):
             out_degrees = np.array([G.out_degree(n) for n in G.nodes()])
             plt.hist(out_degrees)
             plt.show()

             # cdf = out_degrees.cumsum() / out_degrees.sum()
             # ccdf = 1 - cdf
             # plt.plot(range(len(out_degrees)), ccdf)
             # plt.show()
             x = np.concatenate([np.sort(list(out_degrees))])
             plt.plot(x, np.linspace(1, 0, len(x)))
             plt.show()
```

```
plot_in_degree_dist(G)
plot_out_degree_dist(G)
```









In [43]:
```
undirected_G = G.to_undirected()
avg_cluster = nx.average_clustering(undirected_G)
overall_cluster = nx.transitivity(undirected_G)
```

```
max_diameter = nx.diameter(undirected_G)
avg_diameter = nx.average_shortest_path_length(undirected_G)

print("avg clustering coeff: {}\n overall clustering coeff: {}".format(avg_cluster, o
print("avg diameter: {}\n max diameter: {}".format(avg_diameter, max_diameter))
```

```
avg clustering coeff: 0.5568782161697919
 overall clustering coeff: 0.6288944756689877
avg diameter: 6.049380016182999
 max diameter: 17
```

2 B

```
In [44]: num_triangles = np.sum(list(nx.triangles(undirected_G).values()))/3
         print("Num triangles: {}".format(num_triangles))
         n = G.number_of_nodes()
         p = (6 * num_triangles / (n * (n-1) * (n-2))) ** (1/3)
         print("p = {}".format(p))
```

```
Num triangles: 47779.0
p = 0.015861688593415416
```

3 C

We expect the erdos graph to take on a binomial distribution, but this graph takes on a heavy-tailed distribution more, so it is not a great model. We do not need the histogram to conclude this, as the random graph is an evenly distributed sample of edges between nodes, whereas in the real world they are note quite random and some nodes have many more connections than others based on other properties.

Set 2

3 a i) $\pi(n)\ \frac{1}{n^3}$

$\omega(\pi(n))$

$p > \frac{1}{p}$

$\boxed{E[T]} = \frac{n(n-1)(n-2)}{6} \cdot p^3 \to \infty$

$\Rightarrow 0$

$p < \frac{1}{n}$

$O(\pi(n))$

take $\pi(n) = \frac{1}{n}$. for $\omega(\pi(n))$, we take $p > \frac{1}{n}$ in the limit of $n \to \infty$. Thus, since we have $E[T] = \frac{n(n-1)(n-2)}{6} p^3$,

we know in the limit of $n \to \infty$, $p^3 > n^3$, which means we have an unbounded $E[T] \Rightarrow E[T] \to \infty$

For $O(\pi(n))$, take $p < \frac{1}{n}$ in the limit of $n \to \infty$. Thus, since we have $E[T] = \frac{n(n-1)(n-2)}{6} p^3$, in the limit of $n \to \infty$, our function is dominated by p^3, which by O notation means $E[T] \to 0$ in the limit of $n \to \infty$

ii) Enumerate all $\triangle$

$$\mathrm{Var}[T] = E[T^2] - E[T]^2$$
$$= \underbrace{\binom{n}{3}p^3 + \binom{6}{3}\binom{n}{6}p^6 + \binom{4}{3}\binom{3}{2}\binom{n}{4}p^5 + \binom{n}{5}\binom{5}{3}\binom{3}{1}p^6}_{E[T^2]} - \underbrace{\binom{n}{3}^2 p^6}_{E[T]^2}$$

$$\pi(n) = \frac{1}{n}$$

So if $p(n) = \pi(n)\log(n)$, we can cancel $\binom{6}{3}\binom{n}{6}p^6$ and $-\binom{n}{3}^2 p^6$ in the limit of n. the p^5 and p^6 terms left are then dominated by $\frac{1}{n}$, so they $\to 0$ as $n \to \infty$. Then, all that is left is the term w/ p^3, which in the limit of n, the n terms cancel leaving $\log^3(n)$. Thus, it must be that $\mathrm{var}[T] \in \Theta(\log^3(n))$

$$\Pr(T=0) \le P(|T-\mu| \ge k) \quad k = \mu$$
$$\le \frac{\sigma^2}{k^2}$$
$$\le \frac{\log^3(n)}{\left(\binom{n}{3}p^3\right)^2}$$
$$\le \frac{\log^3(n)}{\left(\frac{n^3}{n^3}\cdot\log^3\right)^2} \text{ in the limit of } n\to\infty$$
$$\le \frac{\log^3(n)}{(\log^6(n))}$$
$$\le 0 \text{ in limit } n \to \infty$$

So $\Pr(T=0) \in o(1)$

$$\Pr(T>0) = 1 - \Pr(T=0)$$

b/c $\Pr(T=0) \in o(1)$, as $n \to \infty$, $\Pr(T=0) \to 0$, so

$$\lim_{n\to\infty} \Pr(T>0) = 1 - \lim_{n\to\infty} \Pr(T=0) = 1 - 0 = 1 \checkmark$$

3b. $\Pr(d=2) = 1$

$$\lim_{n\to\infty} \Pr(d=1) + \Pr(d=2) + \Pr(d>2) = 1$$

$p^{\binom{n}{2}} \to 0$ — Probability of fully connected G.

$\binom{n}{2}(1-p)(1-p^2)^{n-2} \to 0$

pick 2 vertices | prob they don't share edge | prob both the other n-2 vertices aren't connected to both chosen vertices

Thus, the 2 other possibilities' $\Pr \to 0$ as $n \to \infty$, so $\Pr(d=2) = 1$ must be true.

Problem4

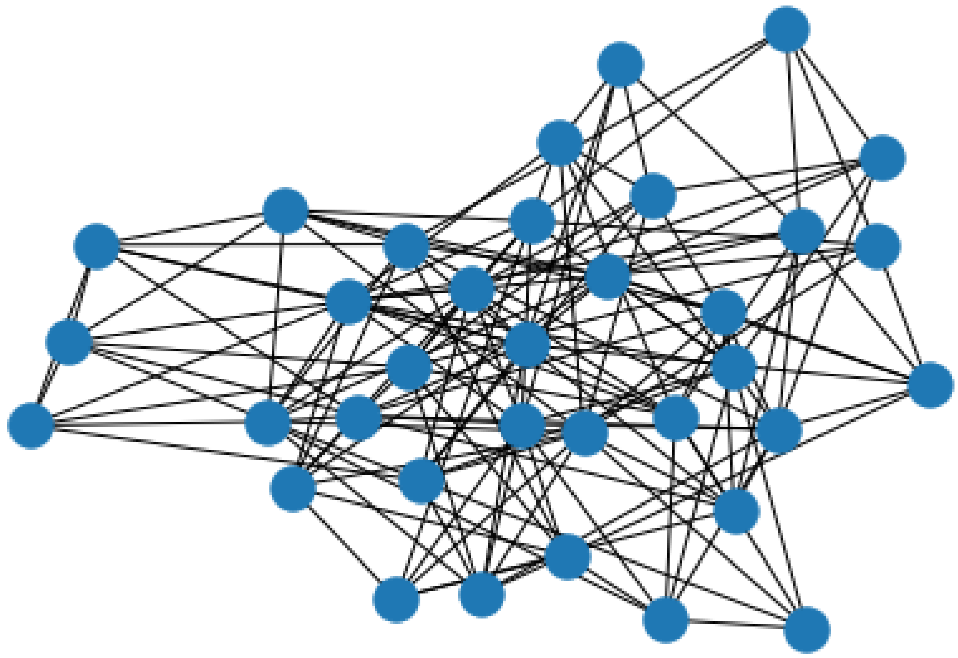
January 25, 2023

```
In [121]: import networkx as nx
          import matplotlib.pyplot as plt
          import numpy as np
          import pickle
```

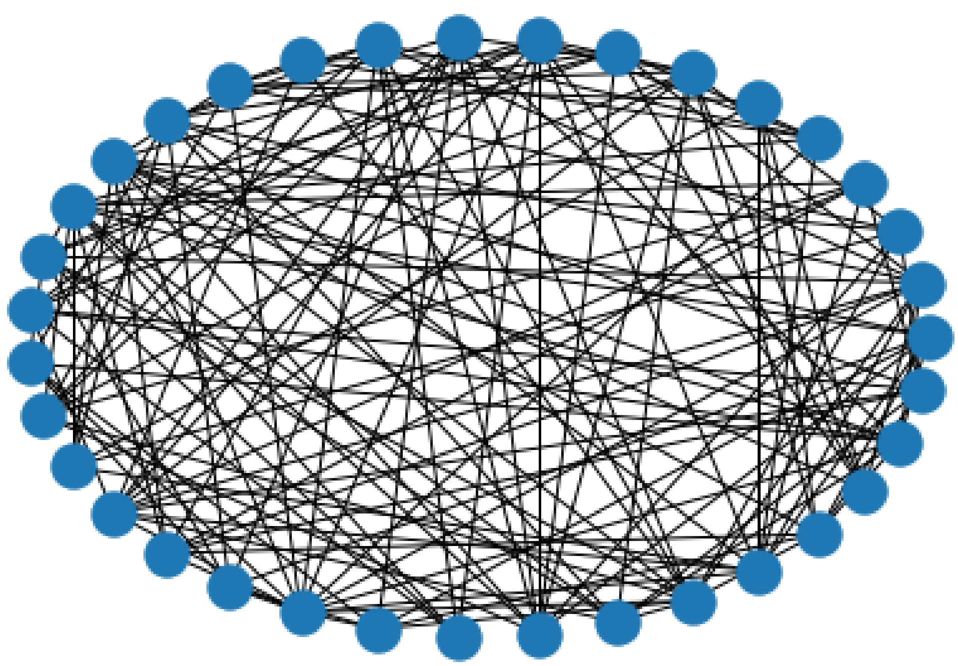
1 A.

```
In [122]: G = nx.Graph()
          n = 35
          p = 0.25
          for i in range(n):
              G.add_node(i)
          for i in range(n):
              for j in range(i+1, n, 1):
                  if np.random.random() <= p:
                      G.add_edge(i, j)
```

```
In [123]: nx.draw(G)
```



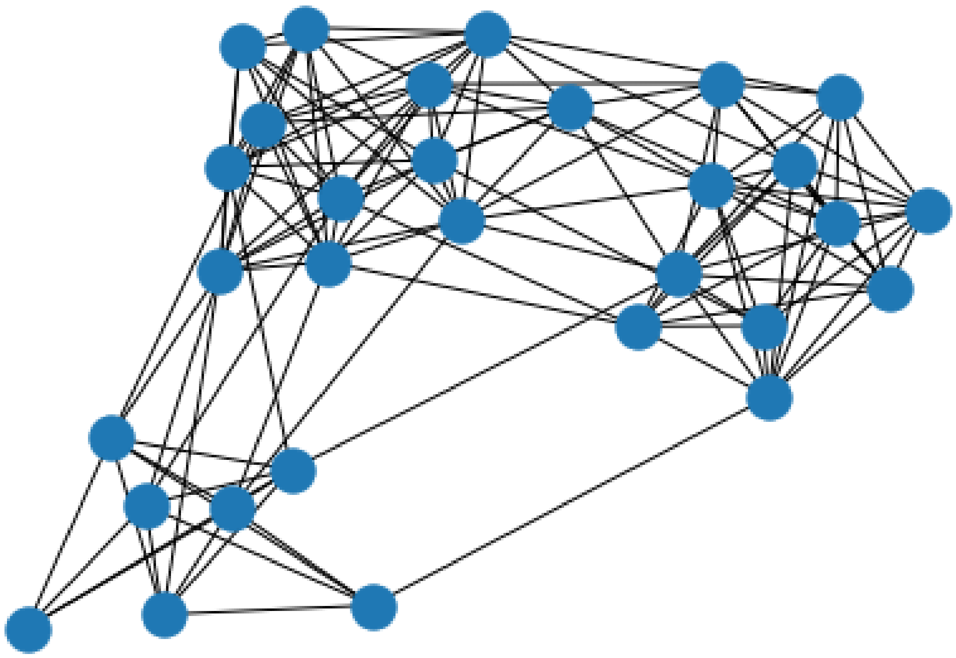
In [124]: nx.draw_circular(G)



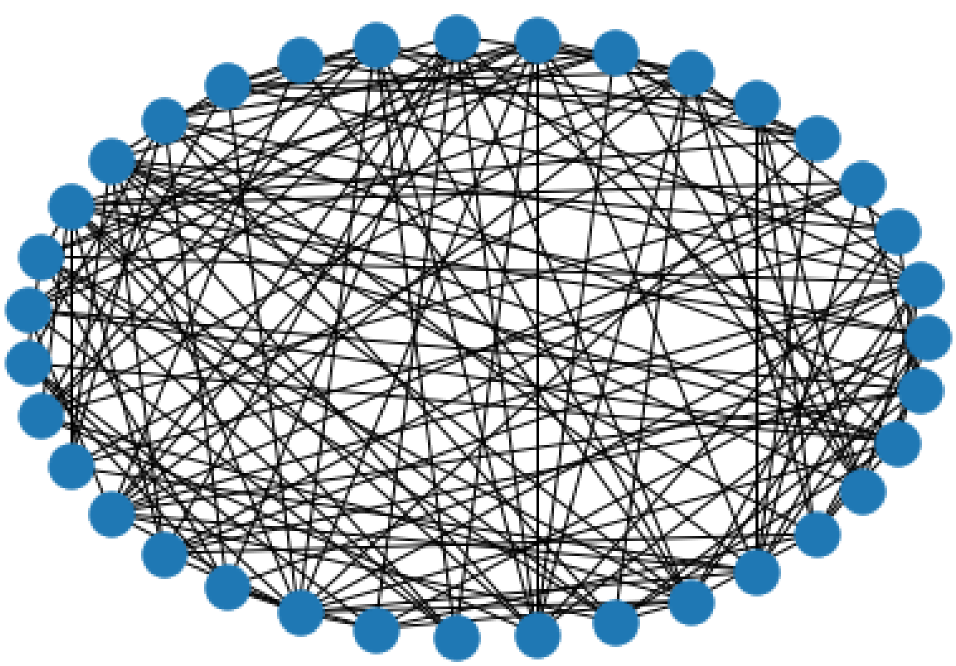
2 B

In [98]:
```
n = 30
k = 3
A = 0.7
B = 0.1
W = [[A, B, B], [B, A, B], [B, B, A]]

G = nx.Graph()
labels = [0, 1, 2]

for i in range(n):
    label = np.random.choice(labels)
    G.add_node(str(i) + " " + str(label))

nodes = list(G.nodes)
for i in range(n):
    for j in range(i + 1, n, 1):
        node1 = nodes[i]
        node2 = nodes[j]
        label1 = node1.split(" ")[1]
        label2 = node2.split(" ")[1]
        p = W[int(label1)][int(label2)]
        if np.random.random() < p:
            G.add_edge(node1, node2)
```

In [89]:
```
nx.draw(G)
```



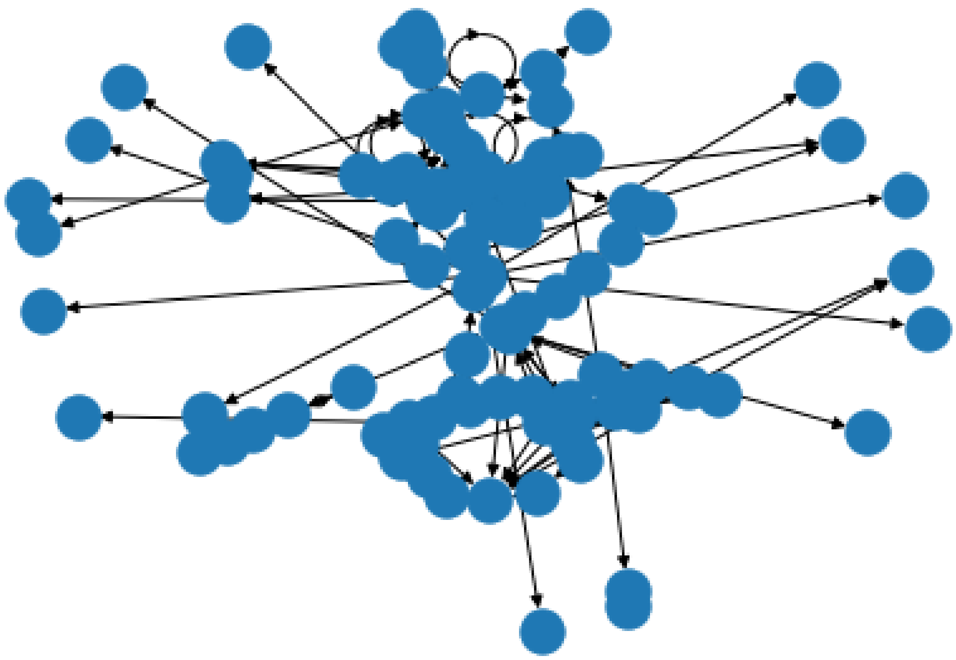
```
In [131]: nx.draw_circular(G)
```



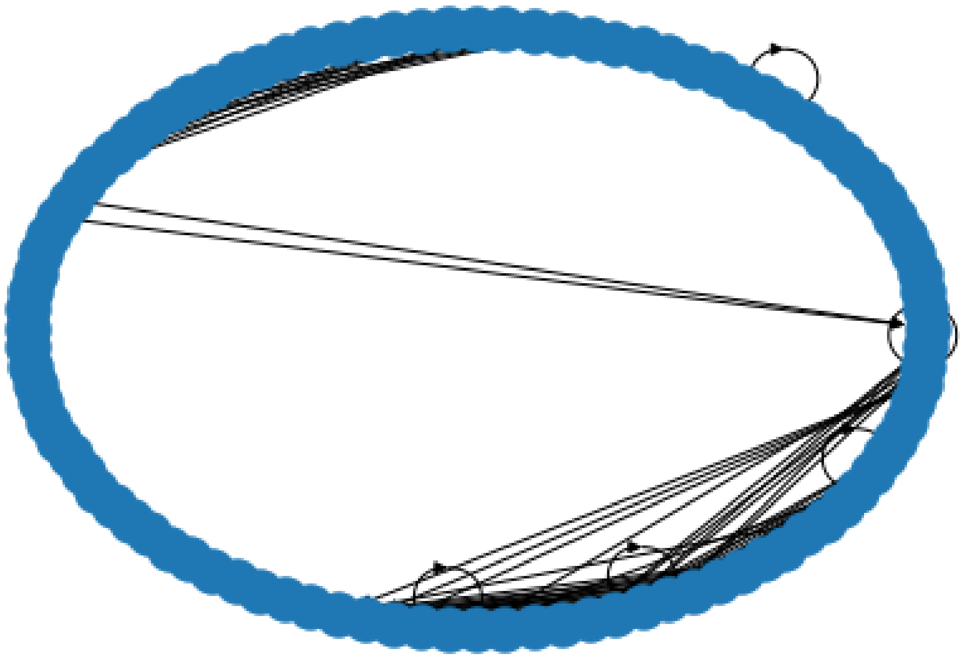
3 C

```
In [100]: with open("web_graph_full.pickle", "rb") as graph_file:
              web_graph_100 = pickle.load(graph_file)
          web_graph_100.remove_nodes_from(list(web_graph_100.nodes)[100:])
```

```
In [101]: nx.draw(web_graph_100)
```

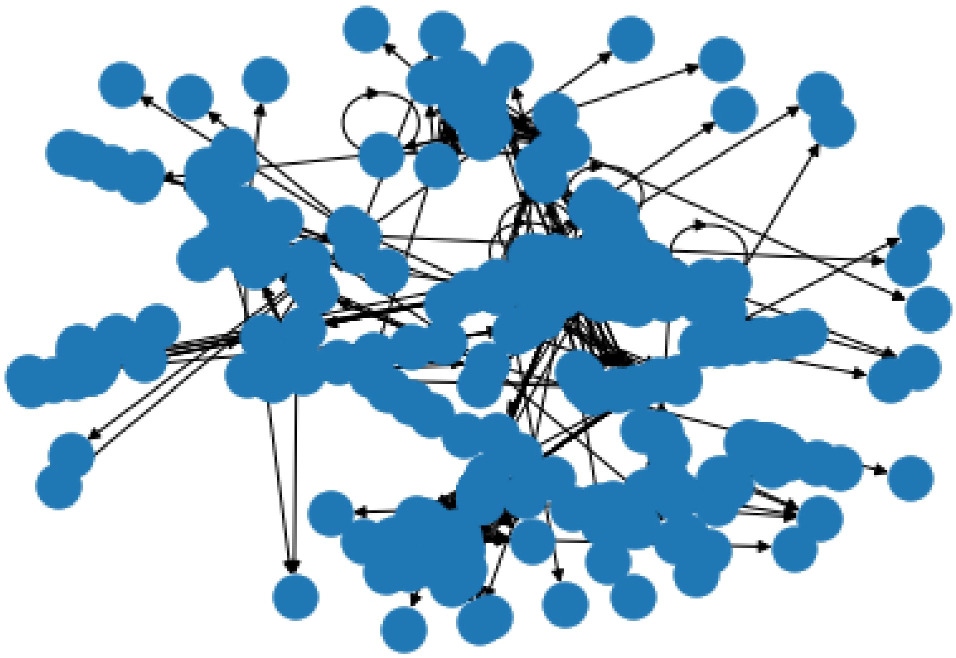


```
In [126]: nx.draw_circular(web_graph_100)
```

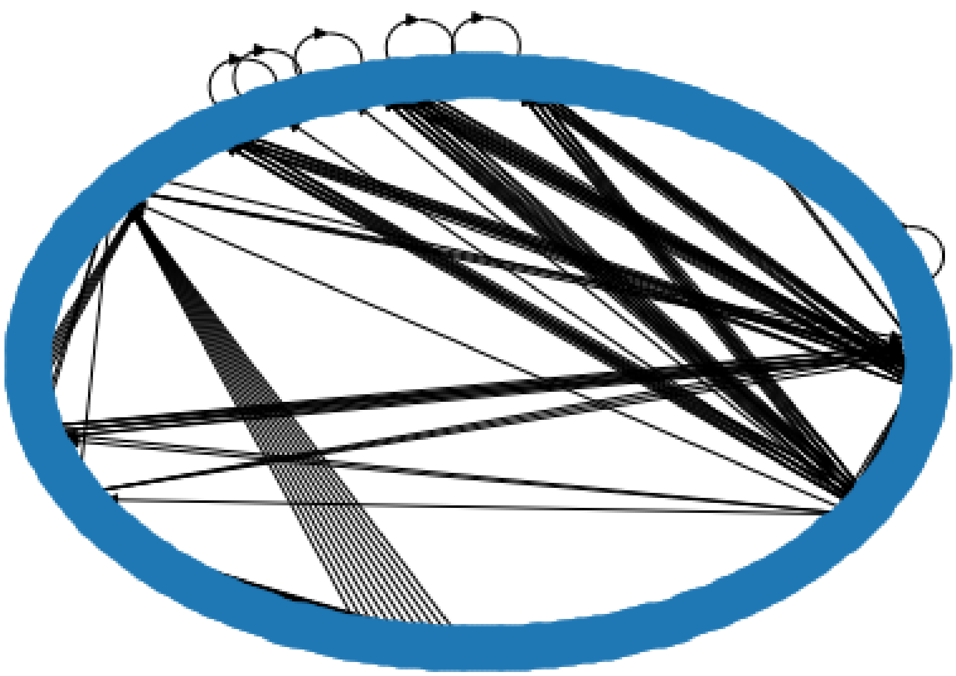


In [104]:
```
with open("web_graph_full.pickle", "rb") as graph_file:
    web_graph_300 = pickle.load(graph_file)
web_graph_300.remove_nodes_from(list(web_graph_300.nodes)[300:])
```

In [128]:
```
nx.draw(web_graph_300)
```



```
In [129]: nx.draw_circular(web_graph_300)
```



The default draw is the spring visualization, which tends to show clusters well by increasing spring force on nodes with lots of edges. The nodes look very clustered there, while the circular visualization lets us see the density in edges between clusters pretty well, and the presence of smaller clusters better. I used these 2 visualizations for all of these.

For the erdos graph, we see that it is truly random. There are no outstanding clusters and the edge density seems pretty even in the circular graph as well.

for the SBM graph, we see the 3 clusters we should see with the spring graph, but we see that the distribution of edge densities is pretty random still.

In both the web graphs, we see some big distinct clusters with the spring visualization, but we also see pretty numerous smaller clusters with large edge densities, or at the very least can see the connections between highly connected nodes such as home pages.