Problem Set 2

Collaborated with: Steven Brotz, David Kawashima

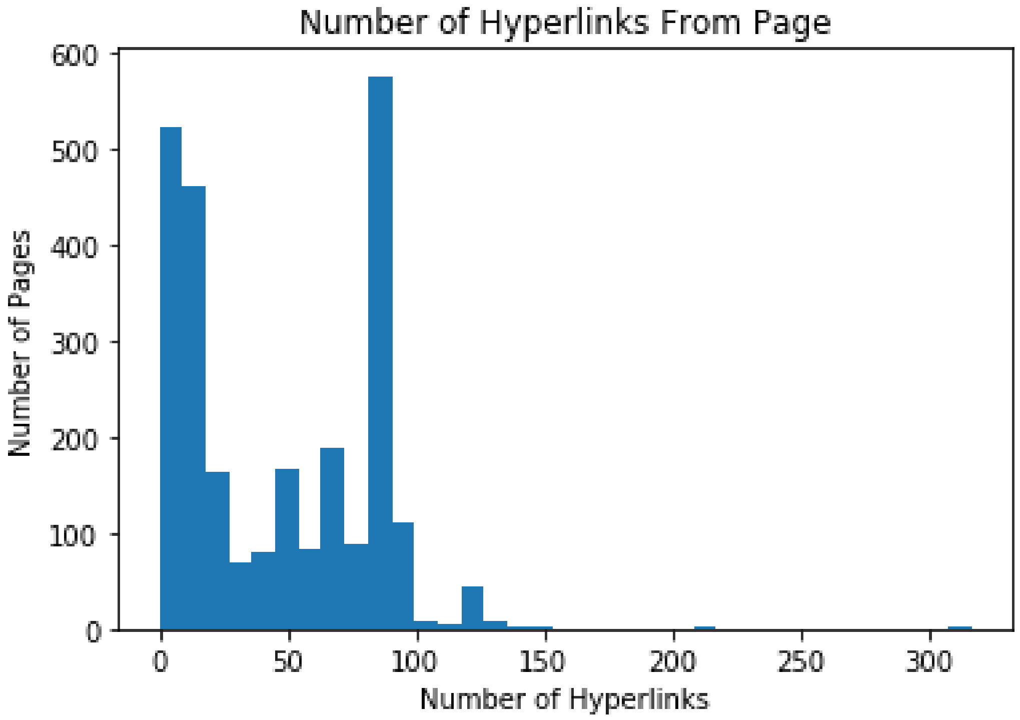
Problem 1

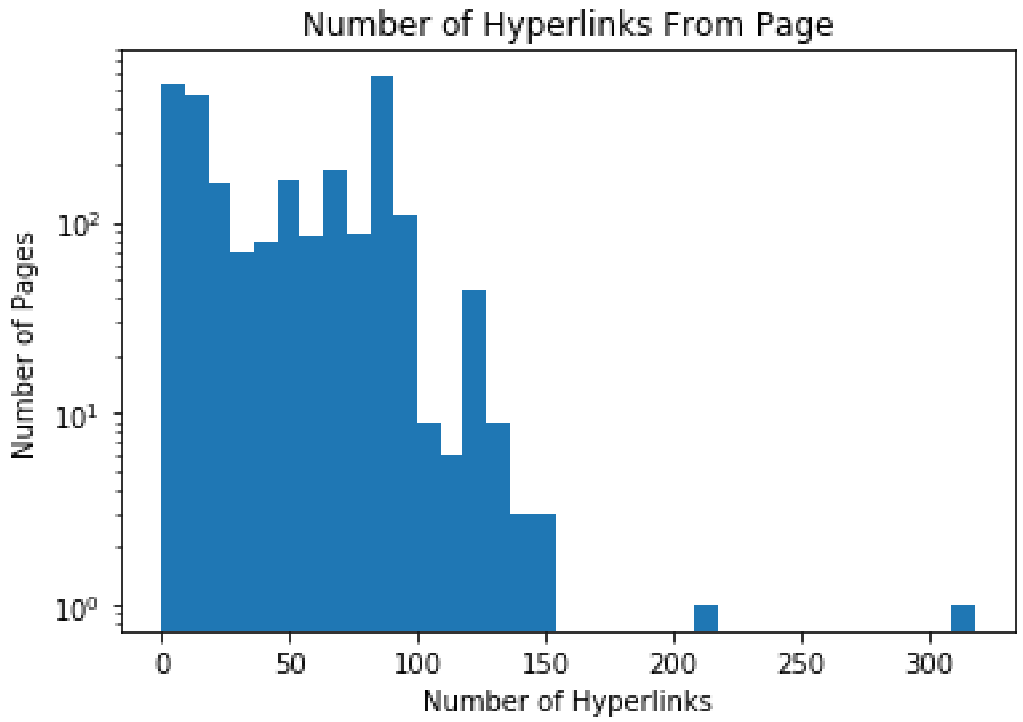
The source code can be found in *web_crawler_v1.py*. To run it, simply type *python web_crawler_v1.py* in the cmd. I used the *networkx* libraries to build the graph, the provided *fetcher3.py* to fetch links from a given URL, and the built-in *queue* libraries for my BFS traversal. The logic for my program which uses a BFS traversal to crawl is described below:

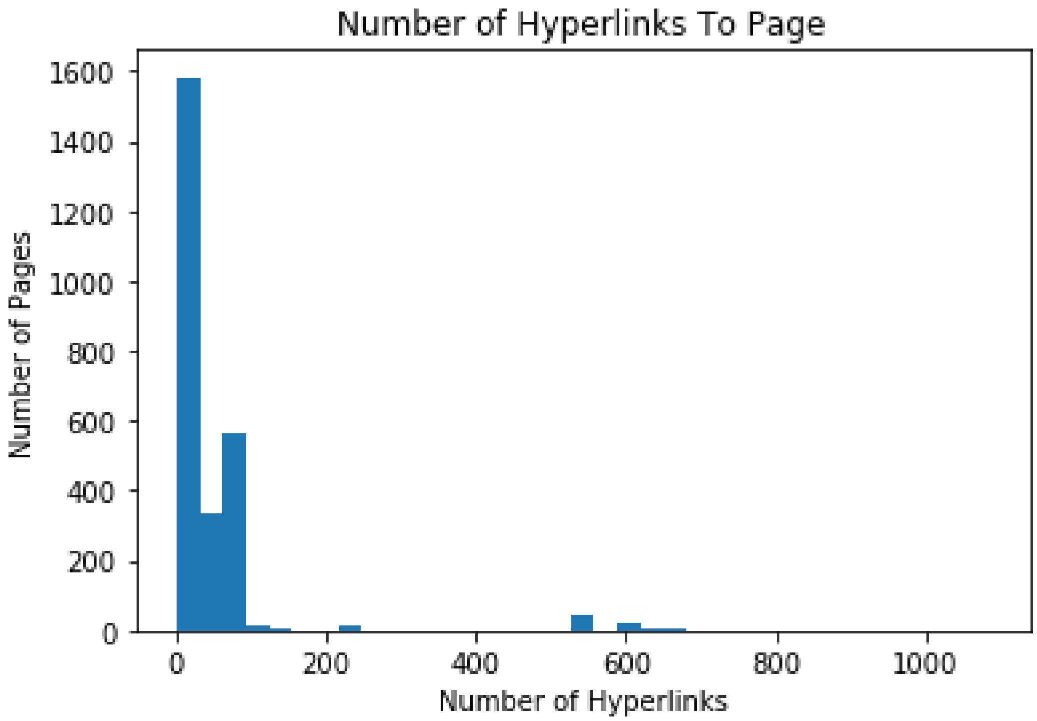
1. Initialize a queue with the starting url of *"http://www.caltech.edu"*.
2. Initialize a "visited" set which contains the urls that have been previously visited.
3. Initialize a counter $count = 0$ that corresponds to the number of urls visited.
4. Initialize a directed graph G
5. While $count < 2500$:
    i. Pop url i from queue.
    ii. If i in visited set, skip to next iteration, else:
    iii. Add url i to visited set
    iv. Fetch list of links from url i
    v. For each link j, add an edge (i, j) to G and add j to queue
    vi. Increment count by 1
6. Filter G as to only contain the nodes that were crawled on (as mentioned on Piazza)
7. Conduct analysis on G (in_degree, out_degree) and make histograms and CCDFs of number of hyperlinks per page and number of hyperlinks that point to page
8. Convert G to undirected graph H for average clustering coefficient, overall clustering coefficient, average diameter, and maximal diameter.

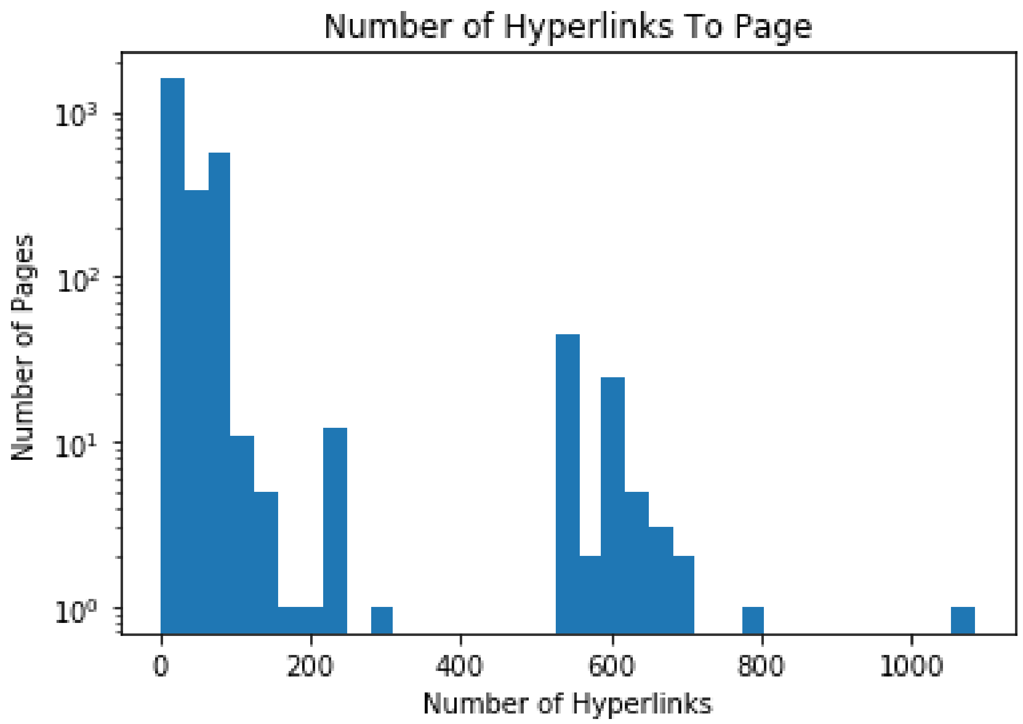
I used a BFS because I thought a BFS would do a better job of getting overall structure of the Caltech domain as oppposed to a DFS. By nature of BFS, we get a better breadth of URLs as opposed to a DFS which priorities URLs that are directly connected to one another on direct paths. I also chose to filter fetched URLs to only get URLs that are in Caltech's domain (more specifically URLs that contained "caltech.edu" in them) and URLs that contained "http" as to avoid downloading multimedia files. One of the reasons why I did such was to avoid crawling websites such as journals that are not in Caltech's domain. Another reason was that due to the nature of the problem in wanting to analyze the structure of web pages on Caltech's domain, it would be better to have a closed graph of URLs where each URL is part of Caltech's domain. However, the disadvantage of doing such is that some web pages may be inaccurately represented, particularly pages that may be really connected to the rest of the web but not to web pages in Caltech's domain. Lastly, some links yielded httpErrors, which likely can be attributed coming across a password-protected site. To circumvent issues arising from these errors, I employed a try, except block to skip over such pages. The advantage of doing such is that it prevents the crawler from getting hung up. In addition, it seems like an intuitive fix; pages that are password-protected should be treated differently than pages that are publicly accessible. The disadvantage of disregarding such websites is that the password-protected section of Caltech's domain is overlooked with this approach. Perhaps the password-protected pages of Caltech's domain are really interconnected, but we are not able to account for this using our methods.

Now, onto our histograms. We have a pair of histograms (normal and log scale) for number of hyperlinks from each page and number of hyperlinks which point to each page.

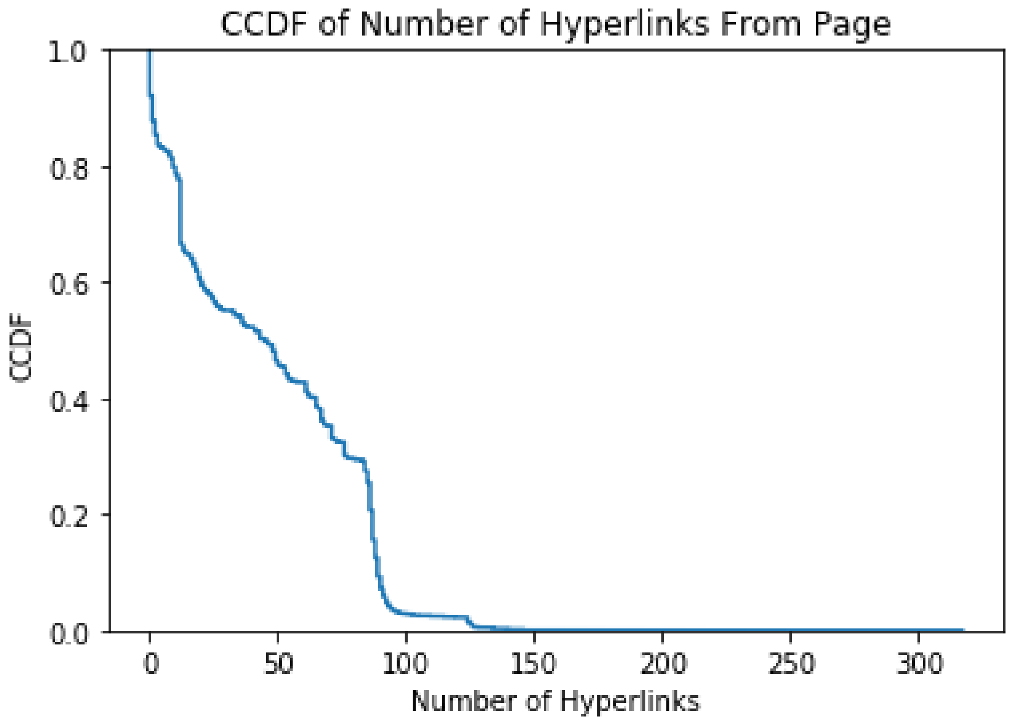


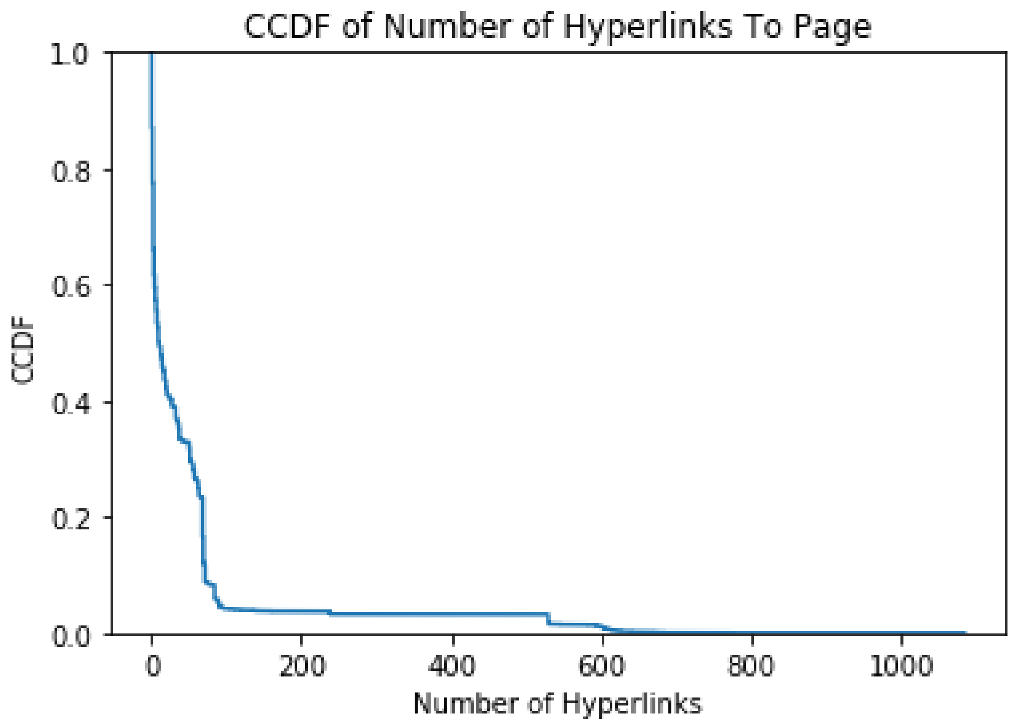

Number of Hyperlinks From Page
Number of Pages
10^2
10^1
10^0
0
50
100
150
200
250
300
Number of Hyperlinks


Number of Hyperlinks To Page
Number of Pages
1600
1400
1200
1000
800
600
400
200
0
0
200
400
600
800
1000
Number of Hyperlinks



Similarly, we have CCDFs for the above quantities.





The average clustering coefficient: 0.7951.
The overall clustering coefficient: 0.3821.
The maximal diameter: 5.
The average distance: 2.6124.

If we compare these metrics with those of problem 3, we see that the diameter for the web crawl is slightly lower. This may be attributed to the fact that our web crawl is limited to Caltech specific pages and from our intuition, we should be able to get from any page to another page by going through a couple of home pages. However, for the FB, the distance measures are still low, but a bit higher than those of the web crawl. This may be because unlike the pages which all come from the same domain, we are not sure if the users are linked in any way (perhaps they are from the same Geographic area) or if they are totally just random users. If we look at the clustering coefficients, both the web crawl and the FB data exhibit high clustering which again, agree with our intuition. Home pages for the web crawl are likely neighbors to many pages and homophily is ever present in any social network. Comparing and contrasting the distributions of number of hyperlinks from page and number of hyperlinks to page, we see that the distribution of number of hyperlinks from page has the vast majority of its data towards the lower end of numbers and only a couple of pages break off and are on the higher end. However, in the case of number of hyperlinks leading to a page, we see a slightly more even distribution; even though there is still a significantly higher proportion of pages with a low number of hyperlinks leading to it, we see more pages having moderate and high number of hyperlinks leading to it (highest is http://www.caltech.edu). This follows our intuition; vast majority of Caltech pages will have about the same low number of hyperlinks going to it, but depending on centrality of a page (whether or not it's a home page i.e. http://www.caltech.edu, http://www.caltech.edu/content/research, http://www.caltech.edu/content/faculty, etc.), we expect to see a bit more variation in number of hyperlinks going to pages. The degree distribution of Facebook's network aligns closely with that of the number of hyperlinks from page(majority of users have low/average number of friends, but there are still a few users that have a very large number of friends).

There are contrasts between the CCDFs however. Between the FromPage CCDF and FB CCDF, we can see from the FromPage CCDF that concentration of web pages having a low number of hyperlinks is higher than the concentration of users having a low number of friends. This makes sense; most web pages have a number of hyperlinks that fit within a well-defined low range, but FB users tend to have more variation in the number of friends they have even if that number is considered low. Comparing the CCDF of the number of hyperlinks to page to number of friends, we see that the CCDF for the number of hyperlinks to page takes a bit longer to reach the x-axis, which makes sense given the bimodal distribution of the number of hyperlinks to page.

Overall, we see that both networks seem to obey the four universal properties. We have reason to believe that both have a "giant" connected component: we can likely get to most Caltech pages from http://www.caltech.edu and Facebook, by nature of an effective social network, is known for interconnecting users. Both seem to have pretty small average distances to suggest small diameters and compactness. From the shapes of degree distributions, we see that we do not have light tails; even though majority of the data is congregated towards the lower end having small degrees/ in edges/ out edges, in all the cases, we have data points and outliers that express a very large degree numbers. Lastly, both have significantly high clustering coefficients, especially compared to the < 0.001 value for clustering coefficients if there was no correlation.

Problem 2

a. We start with:

$$P(D = k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}$$

and taking the limit as $n \to \infty$, we get:

$$\lim_{n\to\infty} P(D = k) = \lim_{n\to\infty} \binom{n-1}{k} p^k (1-p)^{n-1-k}$$

.
Using the substitution $\lambda := (n-1)p$, we get:

$$\lim_{n\to\infty} P(D = k) = \lim_{n\to\infty} \frac{(n-1)!}{k!(n-1-k)!} (\frac{\lambda}{n-1})^k (1 - \frac{\lambda}{n-1})^{n-1-k}$$

Looking at the first term in our product:

$$\frac{(n-1)!}{k!(n-1-k)!} = \frac{(n-1)(n-2)(n-3)...(n-k)}{k!}$$

Note, that as $n \to \infty$, the numerator can be approximated as $(n-1)^k$ and so the above expression can be approximated as $\frac{(n-1)^k}{k!}$. Note that the second term of $(\frac{\lambda}{n-1})^k$ has a denominator that can cancel out with the numerator of the approximated first term. Now for the third term, we use the following approximation: $\lim_{n\to\infty}(1+\frac{x}{n})^n = e^x$ and so

$$\begin{aligned}\lim_{n\to\infty}(1-\frac{\lambda}{n-1})^{n-1-k} &= (1-\frac{\lambda}{n-1})^{n-1}(1-\frac{\lambda}{n-1})^{-k}\\ &= e^{-\lambda}1^{-k}\\ &= e^{-\lambda}\end{aligned}$$

Now, putting this all together, we get:

$$\begin{aligned}\lim_{n\to\infty} P(D=k) &= \frac{(n-1)^k}{k!}(\frac{\lambda}{n-1})^k e^{-\lambda}\\ &= \frac{e^{-\lambda}\lambda^k}{k!}\end{aligned}$$

as desired.

b. We want to calculate the expected number of triangles, denoted $E[T]$ that $G(n,p)$ contains. First, we note that we have $N = \binom{n}{3}$ triples in G. We let $E[t_i]$ be the expected number of triangles triplet i yields which is precisely equal to the probability that t_i has an edge between each of its 3 nodes, which is equal to p^3. Writing this all out and using linearity of expectation, we have:

$$E[T] = \sum_{i=1}^{N} E[t_i] = \sum_{i=1}^{N} p^3 =$$

$$\binom{n}{3}p^3$$

Now, to show that $E[T]$ has a threshold, we let $p = n^{-\alpha}$. Now, taking the $\lim_{n\to\infty} E[T]$:

$$\lim_{n\to\infty}\binom{n}{3}p^3$$

$$= \lim_{n\to\infty}\binom{n}{3}n^{-3\alpha}$$

$$= \lim_{n\to\infty}\frac{(n)(n-1)(n-2)n^{-3\alpha}}{6}$$

$$= \lim_{n\to\infty}\frac{(n)(n-1)(n-2)}{6n^{3\alpha}}$$

From above, we see that if we let $\alpha > 1$, then

$$\lim_{n\to\infty}\frac{(n)(n-1)(n-2)}{6n^{3\alpha}} = 0, \alpha > 1$$

and that if we let $\alpha < 1$, then

$$\lim_{n\to\infty} \frac{(n)(n-1)(n-2)}{6n^{3\alpha}} = \infty, \alpha < 1$$

so we have: if $p < n^{-1}$, then $\lim_{n\to\infty} E[T] = 0$ and if $p > n^{-1}$, then $\lim_{n\to\infty} E[T] = \infty$.

c. Let random variable X be the total number of pairs of nodes in $G(n,p)$ that do NOT share a common neighbor. Thus, to show that the maximal diameter of $G(n,p)$ equals two with a probability of 1 as n becomes large, we can equivalently show that $P(X = 0) \to 1$, $n \to \infty$, or the complementary event, $P(X \geq 1) \to 0$, $n \to \infty$.

By Markov's inequality, $P(X \geq 1) \leq E[X]$. To calculate $E[X]$, we let x_i be an indicator variable for whether or not pair i of nodes do NOT share a common neighbor and $n = (nchoose2)$ be the number of node pairs. Thus, with linearity of expectation, we have:

$$E[X] = \sum_{i=1}^{n} E[x_i] = \binom{n}{2} E[x_i]$$

Now, to calculate $E[x_i]$, because x_i is an indicator variable for whether or not pair i of nodes, which we will say consists of u, v do NOT have any common neighbors, $E[x_i]$ = probability that u and v have no common neighbors. For each other node w, there are three ways in which w can be configured as to prevent w from being a common neighbor of u and v.

1. w is disconnected from u and disconnected from v. This happens with probability $(1-p)^2$.
2. w is connected to u but is disconnected from v. This happens with probability $(1-p)p$.
3. w is connected to v but not connected to u. This happens with probability $p(1-p)$.

Summing the above three options together, we have $(1-p)^2 + p(1-p) + p(1-p) = (1-p^2)$, so for a single node w, there is a $(1-p^2)$ chance that it will be configured as to prevent u and v from having w as a common neighbor. Now, in the total graph of n nodes, we have $(n-2)$ other nodes besides u and v, so the total probability that u and v do not share a common neighbor for the entirety of graph G would be $(1-p^2)^{n-2} = E[x_i]$.
Substituting $E[x_i]$ back into the calculation for $E[X]$, we get:

$$E[X] = \sum_{i=1}^{n} E[x_i] = \binom{n}{2} E[x_i] = \binom{n}{2} (1-p^2)^{n-2}$$

And for large n, we have:

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

and so $\lim_{n\to\infty} P(X \geq 1) = 0$ which proves the complementary event: $\lim_{n\to\infty} P(X = 0) = 1$. Thus, the probability that the number of pairs of nodes that do NOT share a common neighbor is exactly 0 approaches 1 as n grows very large, meaning that the maximal diameter of

$G(n,p)$ equals two with a probability that approaches 1 as n grows very large.

To fully complete this proof however, we need to take care of the extreme case where every node is connected to every node as to make a complete graph and show that as n grows very large, the probability of this happening, goes to 0, because otherwise, the maximal diameter would be 1. The probability of having a complete graph is simply: $p^{\binom{n}{2}}$ because there is a probability $p \in (0,1)$ that any two nodes are connected and for every node to be connected to every other nodes, we want all $\binom{n}{2}$ pairs of nodes to be connected. Taking the limit of this for large n, we get:

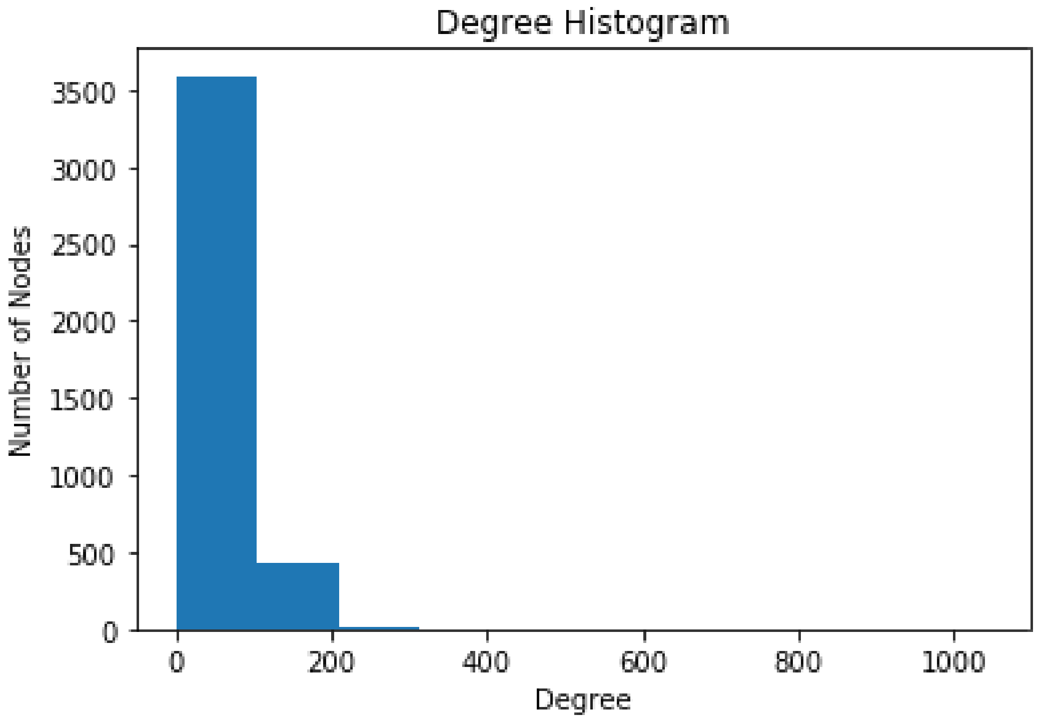
$$\lim_{n\to\infty} p^{\binom{n}{2}} =$$

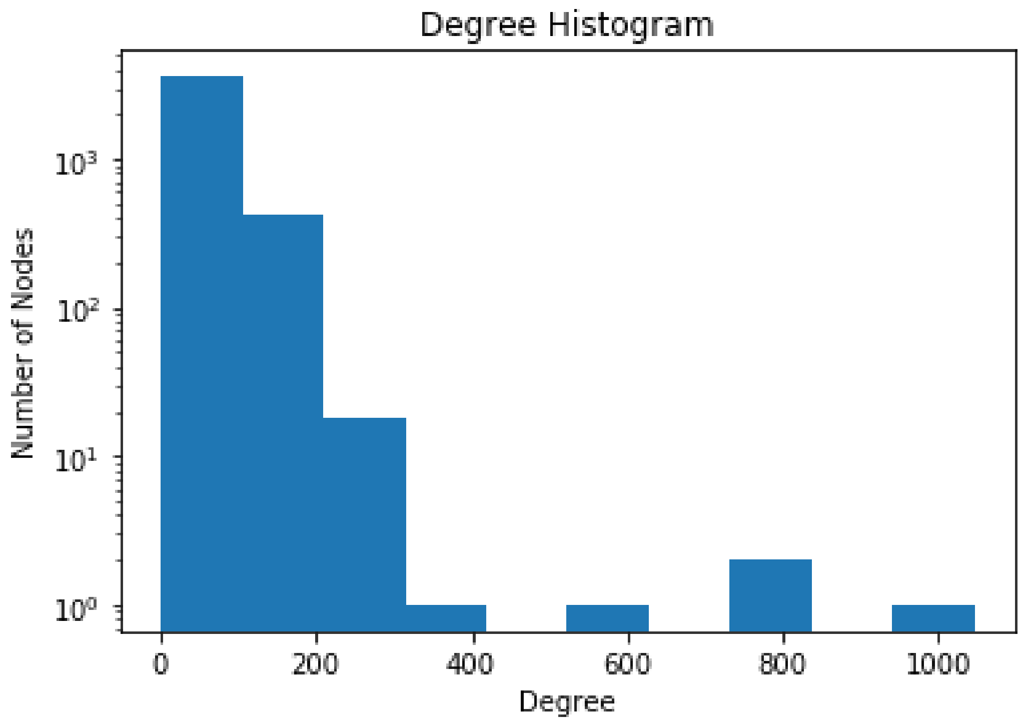
$$\lim_{n\to\infty} p^{\frac{n(n-1)}{2}} = 0$$

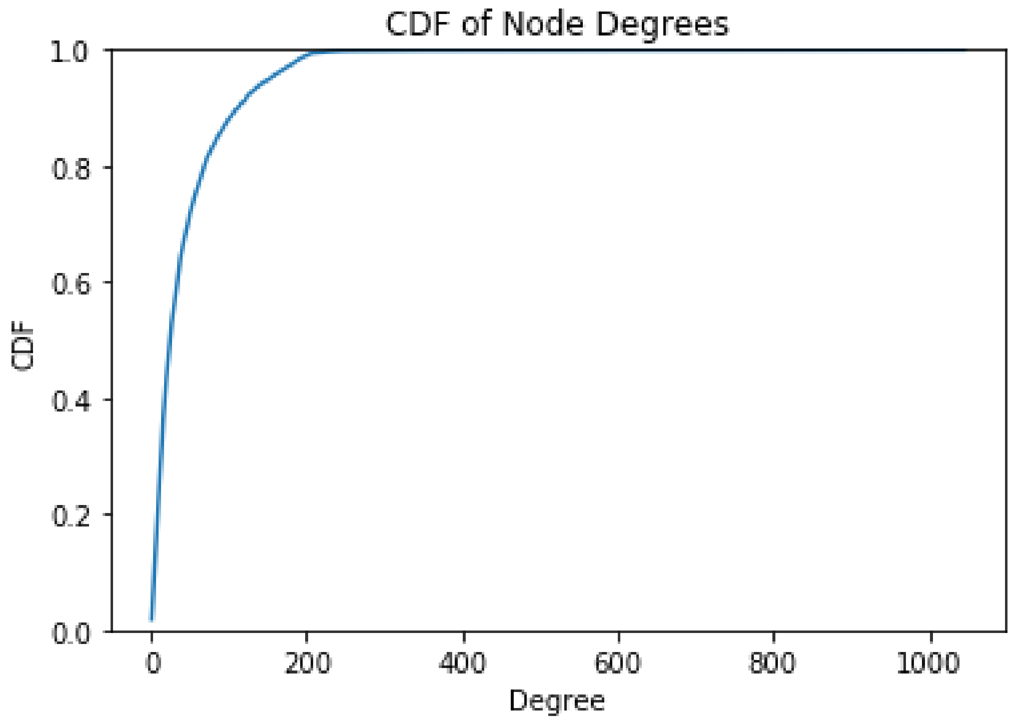
Thus, we have shown that the maximal diameter of $G(n,p)$ equals two with probability that approaches 1 as n becomes large.

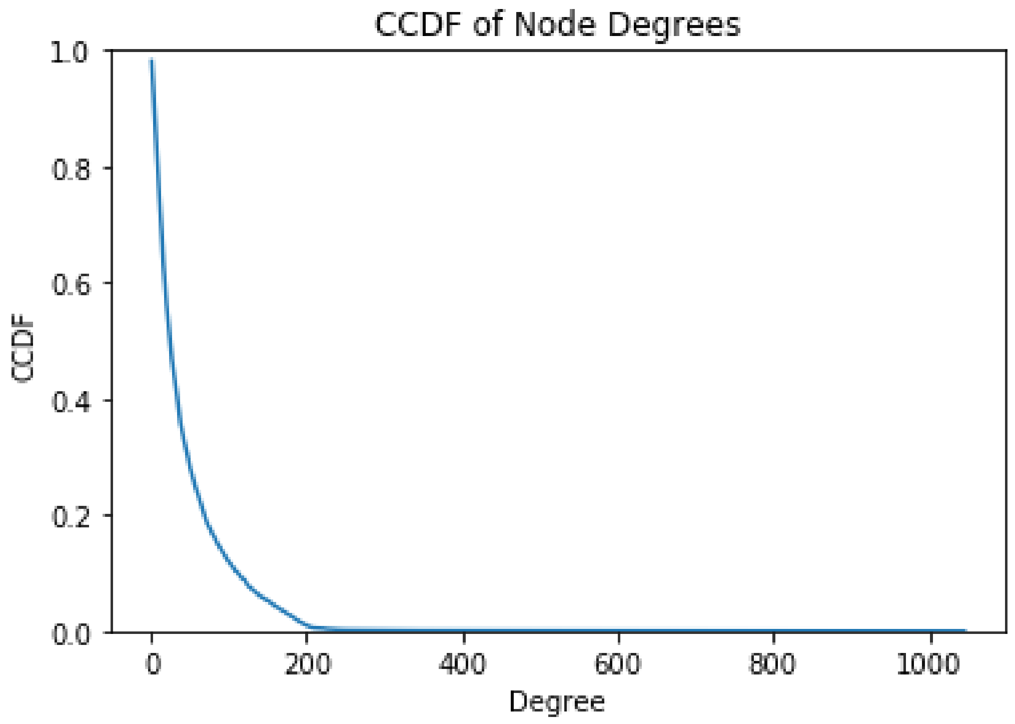
Problem 3

I used the *networkx* Python package to generate the graphs and statistics below. The $G = read\_edgelist('facebook\_edges.txt)$ method was used to read in the file into graph format. $networkX.degree(G).values()$ gives us information about degrees for each node (I included both normal and log scales).




Degree Histogram
Number of Nodes
10^3
10^2
10^1
10^0
0
200
400
600
800
1000
Degree


CDF of Node Degrees
CDF
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000
Degree



The average clustering coefficient: 0.6055.
The overall clustering coefficient: 0.5192.
The maximal diameter: 8.
The average distance: 3.6925.

I used $networkx.average\_clustering(G)$, $networkx.transitivity(G)$, $networkx.diameter(G)$, and $networkx.average\_shortest\_path\_length(G)$, respectively, to generate the statistics above.

Extra credit: In addition to using *networkx*, I went ahead and implemented some of methods required for this problem from scratch. For getting the degree counts, I used a hashmap/dict called *mapping* that maps each node to a list of nodes that it is directly connected to by an edge. For each line of $facebook\_edges.txt$ which is of form $'ab'$, I would append b to $mapping[a]$ and append a to $mapping[b]$ due to symmetry of undirected edges. Now, once we are done reading the file, we simply have to iterate through each key/ node of *mapping* and get the size of the list corresponding to number of nodes each node is adjacent to. The runtime for this is $O(|E|)$ where $|E|$ is the number of edges. This is because for each line of $facebook\_edges.txt$, it is constant time to look up in our dictionary and then amortized $O(1)$ time to append to the list of adjacent nodes. The total number of lines is $O(|E|)$. In addition, realize that in getting the size of corresponding list for each key, we are still doing $O(|E|)$ work because the sum of the sizes of these lists is $O(|E|)$ because a node appears in a list if that node has an edge to another node.

Now, for the maximal diameter and average distance, we can use a BFS level order traversal approach. For maximal diameter, for each node, we start with an initial counter of 0 and an empty queue. Then, we look at which nodes are in its adjacency list and increment the counter by 1 representing that these nodes are 1 edge away. We then look at the neighbors of these nodes (to get nodes that are two edges away from original node)and increment the counter by 1. We keep doing this until our queue is empty, meaning that the original node can no longer reach any more nodes. After doing this for each node, we return the largest counter encountered and that is precisely the maximal diameter. Note, in order to avoid cycles, we have to keep a set of visited nodes and check if each node has been visited before so we do not end up in an infinite cycle. The runtime for this is $O(|E||V|^2)$ because we have $|V|$ nodes and for each node, we have $O(|V|)$ other nodes to calculate distance to and have to traverse $O(|E|)$ edges in the BFS. A similar approach can be used for average distance, but now, we need a dictionary that maps every pair of nodes to the shortest distance of a path between them. Thus, if we calculate the shortest path distance for each pair of nodes, then we are good. We follow the Floyd- Warshall, an $O(|V|^3)$ algorithm to do such.