Topic: Networks by James Mastran

MATH!

Networks can represent a lot of real-world entities and can explain certain phenomena. Few things to note before jumping in:

"g.smallWorldNetwork(L,Z,p)"

-First argument, the L, is the number of nodes in the diagram.

-Second argument, the Z, is the number of immediate neighbors each node is connected to.

-Third argument, the p, determines the number of random shortcuts added between nodes according to the formula: number of additional shortcuts = p * L * Z / 2

-If L is the number of nodes in the diagram and since our diagrams have only, at a minimum, 2 connected neighbors, then that means the maximum distance between any two nodes is L/2, in the case that Z = 2 and p = 0.

-Not as important, but if there at L nodes, then there are L - 1 number of edges, in the case that Z = 2 and p = 0.

-Edges are what connect two nodes. In our examples, edges can represent relationships.

-This is an undirected graph/network, which means that if Node X is connected to Node Y, then Node Y is connected to Node X. This is not true in all networks, but it is true for most that we are exploring today.

-We will be adding shortcuts throughout our networks which will connect non-adjacent nodes. If applied, the depth of a shortcut describes how many nodes are in between two nodes that are connected by a shortcut. For example, Node 0 to Node 10 has a depth of 10.

-When we discuss the length between two nodes, we are talking about the number of steps/edges that must be taken to get from Node X to Node Y.

-Nodes can represent people. Nodes are connected by lines, which we call edges. Edges represent relationships between the nodes, such as friendships.

-We use Breadth-First Search for some of our algorithms which is: "Breadth-first search is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root, and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level" (https://en.wikipedia.org/wiki/Breadth-first_search (https://en.wikipedia.org/wiki/Breadth-first_search)).

Where Did I Get the Information from?

Part one:
http://pages.physics.cornell.edu/~myers/teaching/ComputationalMethods/ComputerExercises/I (http://pages.physics.cornell.edu/~myers/teaching/ComputationalMethods/ComputerExercises/

Part two:
http://pages.physics.cornell.edu/~myers/teaching/ComputationalMethods/ComputerExercises/S
(http://pages.physics.cornell.edu/~myers/teaching/ComputationalMethods/ComputerExercises/
and
http://pages.physics.cornell.edu/~myers/teaching/ComputationalMethods/ComputerExercises/S
(http://pages.physics.cornell.edu/~myers/teaching/ComputationalMethods/ComputerExercises/

What did I do and what did I implement?

Basically, updated all the code in MastranGraph3.py and then made this report document from it. None of the methods in MastranGraph3.py had any code in it. Just comments. I followed the Cornell links above as a guide to my project. Because there is not too much math involved, more than discussed above, I will write about here how I implemented the code. For most of the implementation, I did not always follow the comments. I basically implemented all the code for the following methods (they can be found in MastranGraph3.py):

HasNode(self, node):

"Returns True if the graph contains the specified node, and False otherwise. Check directly to see if the dictionary of connections contains the node, rather than (inefficiently) generating a list of all nodes and then searching for the specified node."

I coded this method so that it looks through the dictionary of nodes to see if the node is in the dictionary of a particular graph.

AddNode(self, node):

"Uses HasNode(node) to determine if node has already been added."

First, I check the dictionary to see if the node has been added, if not then the node is added to the dictionary as not connected to any other node.

AddEdge(self, node1, node2):

"Add node1 and node2 to network first Adds new edge (appends node2 to connections[node1] and vice-versa, since it's an undirected graph) Do so only if old edge does not already exist (node2 not in connections[node1])"

This was implemented such that it adds node1 and node2 through AddNode(node1) and AddNode(node2). Then, we add that node1 is connected to node2 if it is already inserted. Likewise, we add that node2 is connected to node1 if absent.

GetNodes(self)

"g.GetNodes() returns all nodes (keys) in connections"

Returns the dictionary without including the edges. I used a for loop to append all nodes in dictionary to an array that begins empty.

GetNeighbors(self, node): "g.GetNeighbors(node) returns a copy of the list of neighbors of the specified node. A copy is returned (using the [:] operator) so that the user does not inadvertently change the neighbor list."

I use a for loop to go through the dictionary, looking for the specified node and then return once found, otherwise return null.

smallWorldNetwork(L,Z,p, show = True): This creates an L-node UndirectedGraph, where the nodes are arranged in a circle (ring), and then uses the imported NetGraphics module to layout and display the graph. The graph is returned from the function. Z indicates the number of immediate and adjacent neigbors that are connected to each node (minimum of 2). p determines the number of random and additional relationships/edges drawn according to the formula:

$$p * \frac{L}{2} * Z$$

So in this way:

L is the number of nodes in the network

Z is how the ring is formed

p determines number of additional and random shortcuts

I did this as a 3 step process. First, I added all the necessary nodes to the graph. Then I added immediate connections. Then I added the random edges. For more investigation, I added:

smallWorldNetworkLong

In this Long version, setting the depth sets the minimum 'depth' of a shortcut. This depth indicates the number of nodes separating two nodes that are to be connected by a shortcut edge.

smallWorldNetworkShort

In this Short version, setting the depth sets the maximum 'depth' of a shortcut. This depth indicates the number of nodes separating two nodes that are to be connected by a shortcut edge.

smallWorldNetworkSpecific

In this Specific version, setting the depth sets the specific 'depth' of a shortcut. This depth indicates the number of nodes separating two nodes that are to be connected by a shortcut edge.

FindPathLengthsFromNode(graph, node):

"Breadth--first search. See "Six degrees of separation" exercise from Sethna's book. Use a dictionary to store the distance to each node visited. Keys in the dictionary thus serve as markers of nodes that have already been visited, and should not be considered again."

I create a current shell which starts with only the specified node. There is a variable l that is initialized to 0. On each iteration, l is increased, all nodes in current shell are moved to the dictionary and the distance to these nodes are set to be equal to l. Additionally, after every pass current shell gets set to next shell, which includes all the nodes that are connected to every node in current shell. Next shell is updated on each pass. This continues until current shell is empty.

To explain further: first iteration we look at nodes a distance of 1 away. We add them to the array as (distance, point) where point is (start node, end node). Then we repeat this for a distance of 2 away, then 3 away, then 4 away, ... L/2 away. Each time we add every unvisited node to the array with the corresponding number of steps to get to the end node. Why do we only add unvisited nodes? Simple, if it is already added, then we know we cannot find a shorter path on the next iteration. That is the beauty of breadth first search.

FindAllPathLengths calls FindPAthLengthsFromNode over and over until we have an array with every node connected to every other node.

FindAllPathLengths(graph, show=False):

Is done by repetitively calling FindPathLengthsFromNode to generate a "list of all lengths (one per pair of nodes in the graph)"

FindAveragePathLength(graph, include0 = True):

"Averages path length over all pairs of nodes"

This is mainly done with np.average(arg)

FindAverageAveragePathLength(graph):

This returns the average path length over all pairs of nodes as well as the standard deviation as calculated through the wiki page:

https://en.wikipedia.org/wiki/Standard_deviation (https://en.wikipedia.org/wiki/Standard_deviation)

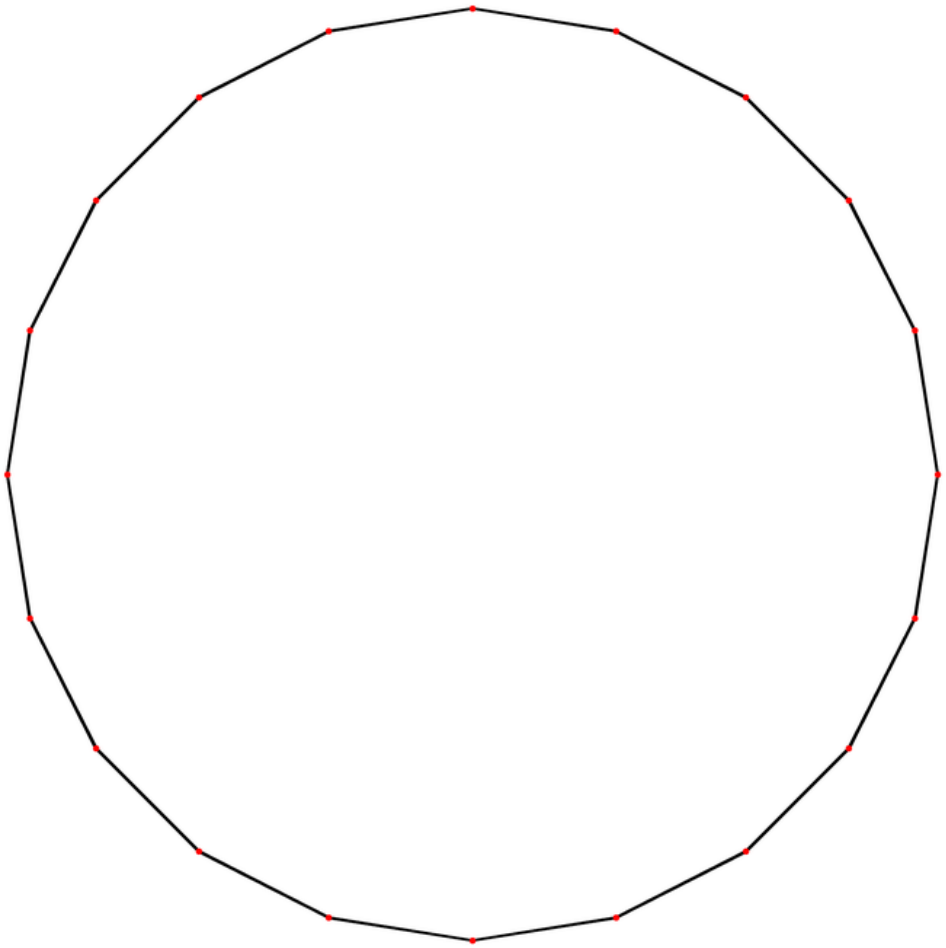
getDist(L, start, end):

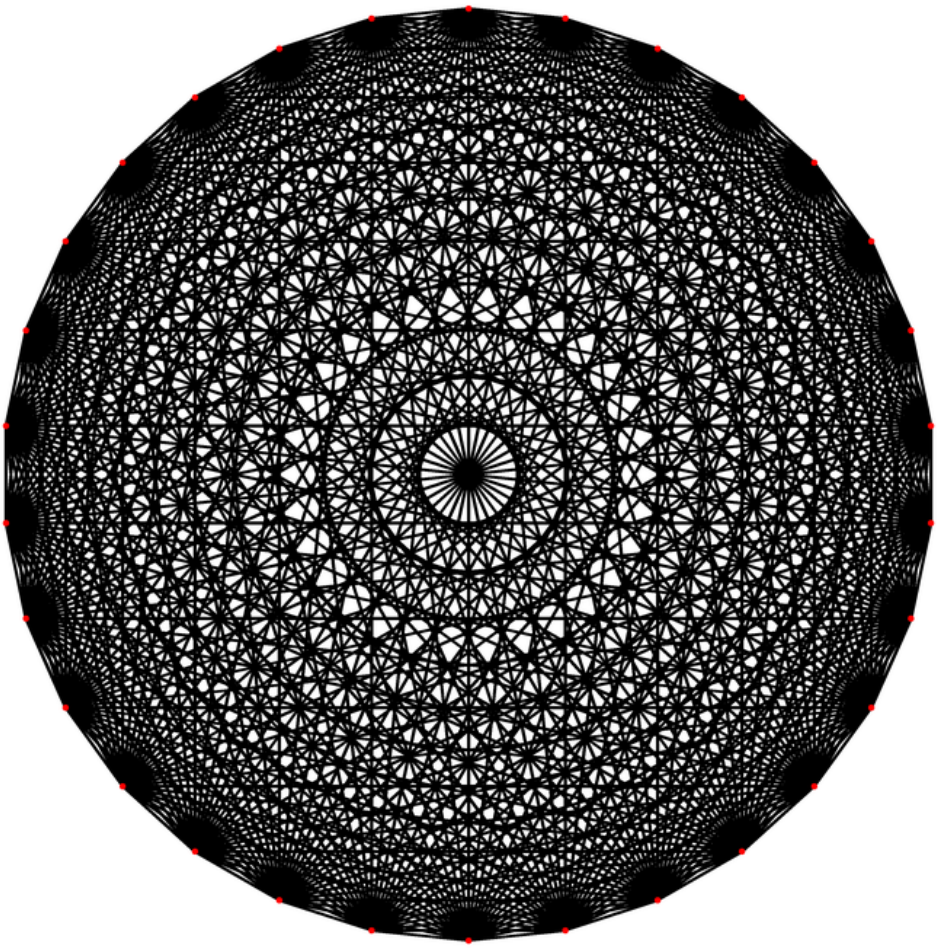
I coded this to have a unified way of calculating the distance between two nodes for when making shortcuts (getting the distance does ignore the addition of extra/random shortcuts). This method is used only in MastranGraph3.py.

Further, in the rest of this document I tried to make headers represent the main idea of the section, bold text to represent tasks to be done, and regular text to represent my commentary. Note that in some cases, text was bolded to explain a main idea or give an important answer.

In [288]:
```
import MastranGraph3 as g
```

In [289]:
```
sw = g.smallWorldNetwork(20,2,0)
sw = g.smallWorldNetwork(30,30,0)
```

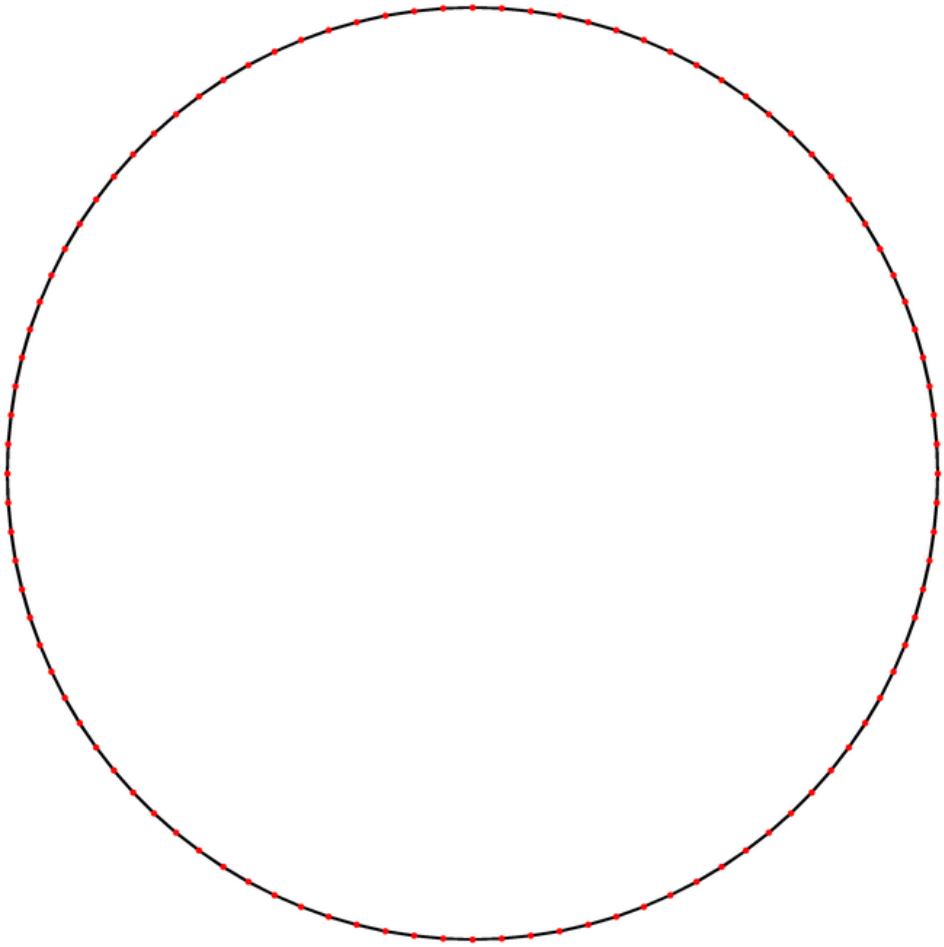




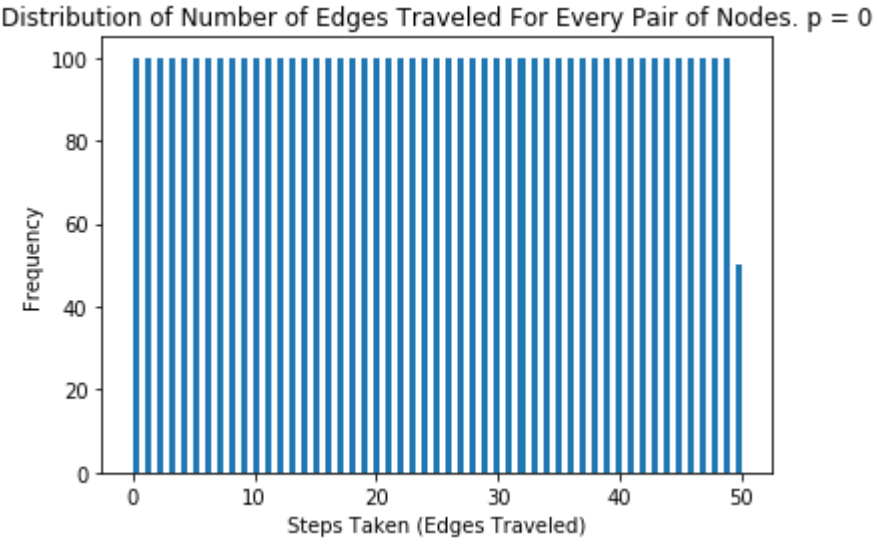
a.) Explore Six Degrees of Separation and FindAllPathLengths

Check my implementation of FindAllPathLengths(graph) by testing that the histogram of path lengths at p = 0 is constant for 0 < l < L/Z, as advertised.

In [49]:
```
sw1 = g.smallWorldNetwork(100,2,0)
```



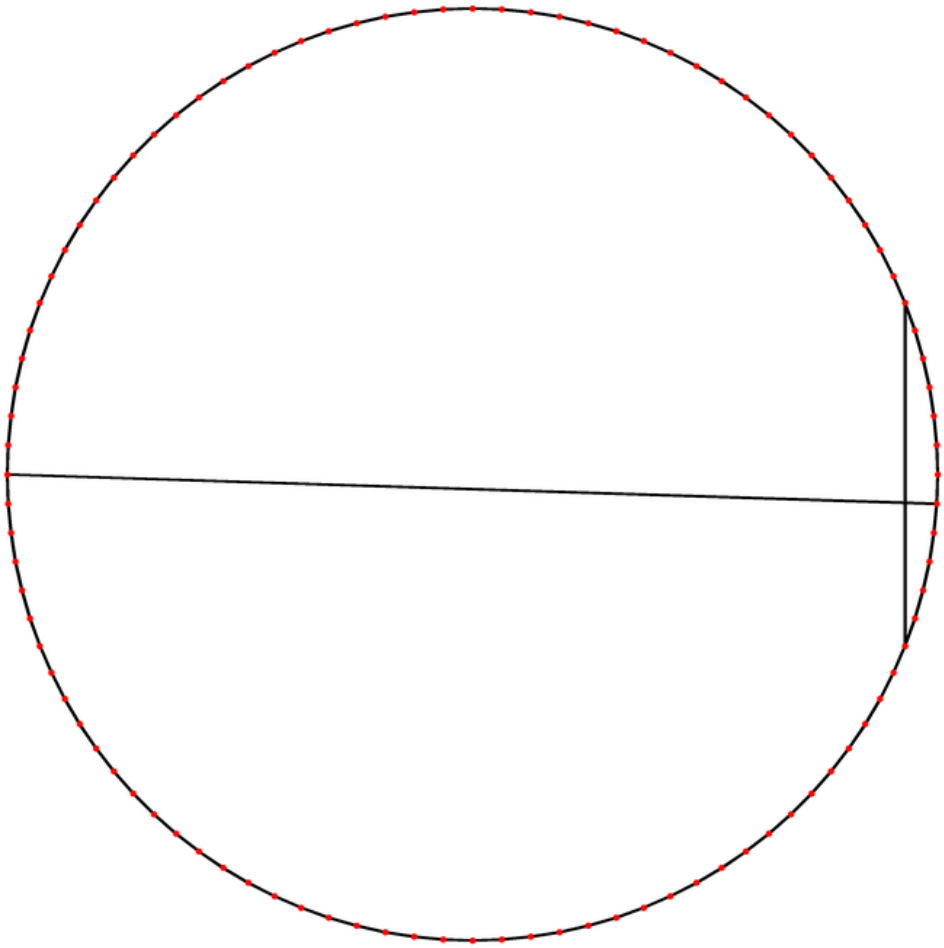
In [355]:
```
sw1res = g.FindAllPathLengths(sw1)
plt.title('Distribution of Number of Edges Traveled For Every Pair of
plt.xlabel('Steps Taken (Edges Traveled)')
plt.ylabel('Frequency')
plt.hist(sw1res[1], bins = 101)
plt.show()
```

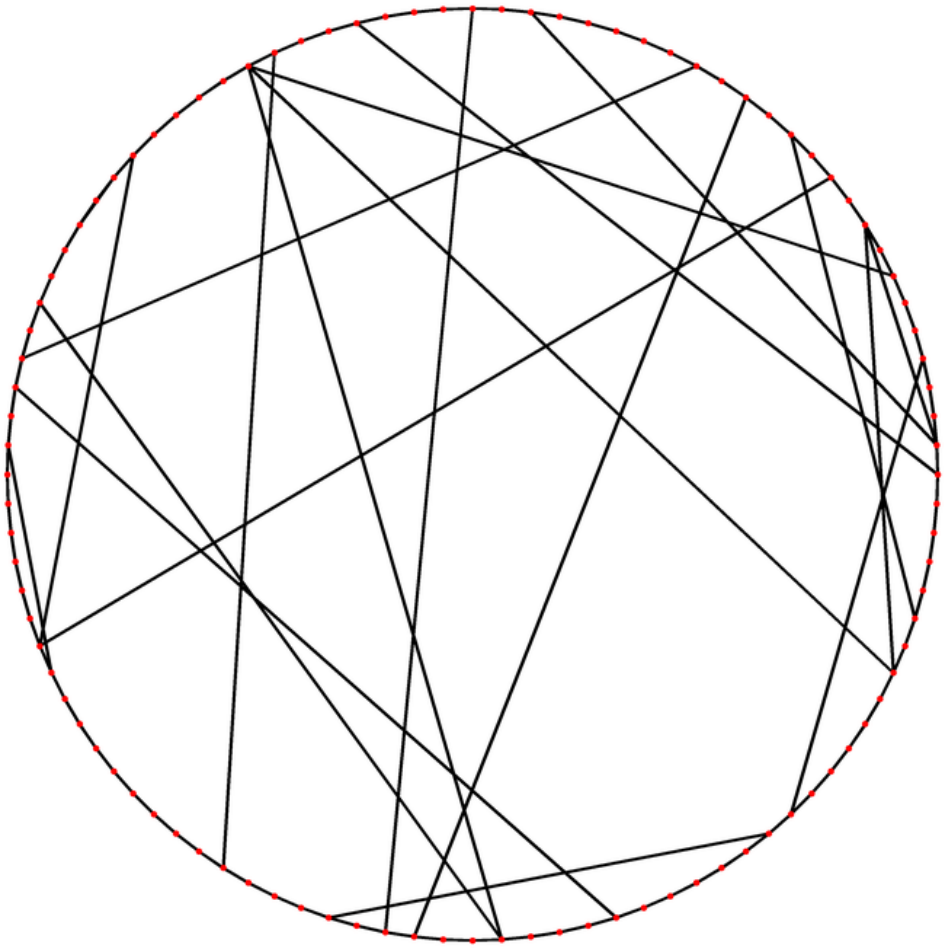


So this is what we would expect, an even distribution of lengths except for at the maximum. Why? Because every node has 2 nodes that are 1, 2, 3, 4, ..., L/2 - 1, steps/edges away. Therefore, 2 * 50 nodes = 100 nodes. However, Each node only has one partner that is L/2 away. Thus if there are 100 nodes, that means there are 100 nodes connected to another node a distance of 1 away, but only 50 nodes connected to another node that is a distance of 50 (the maximum) away. In addition, it makes sense that 50 is the maximum number of steps taken because L = 100, and L/2 is the maximum number of steps in any graph connected in a ring-shaped manner as we are investigating. When L = 100, then L/2 = 50.

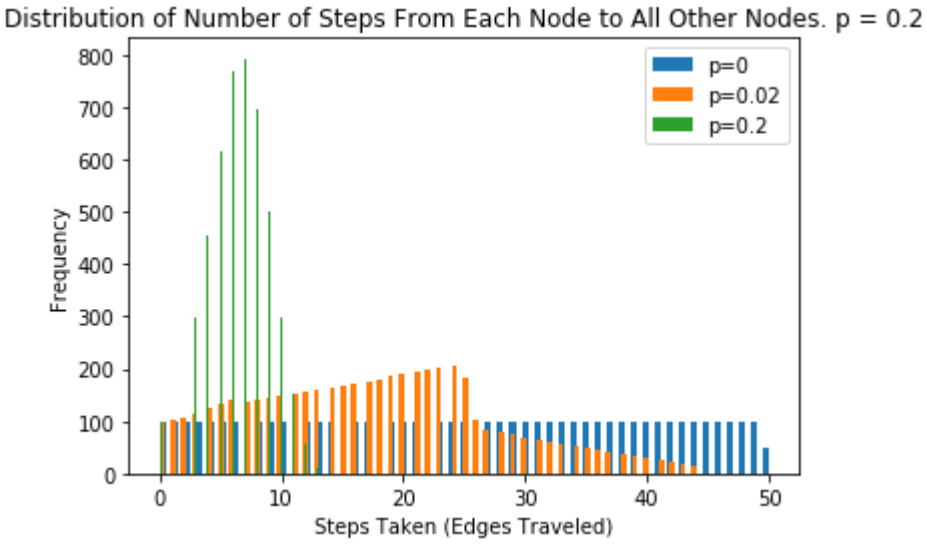
Generate graphs at L = 100 and Z = 2 for p = 0.02 and p = 0.2; display the circle graphs and plot the histogram of path lengths.

In [17]:
```
sw2 = g.smallWorldNetwork(100,2,.02)
sw3 = g.smallWorldNetwork(100,2,.2)
```





In [353]:
```
results2 = g.FindAllPathLengths(sw2)
results3 = g.FindAllPathLengths(sw3)
plt.title('Distribution of Number of Steps From Each Node to All Other
plt.xlabel('Steps Taken (Edges Traveled)')
plt.ylabel('Frequency')
plt.hist(sw1res[1], bins = 101, label='p=0')
plt.hist(results2[1], bins = 101, label='p=0.02')
plt.hist(results3[1], bins = 101, label='p=0.2')
plt.legend()
plt.show()
```

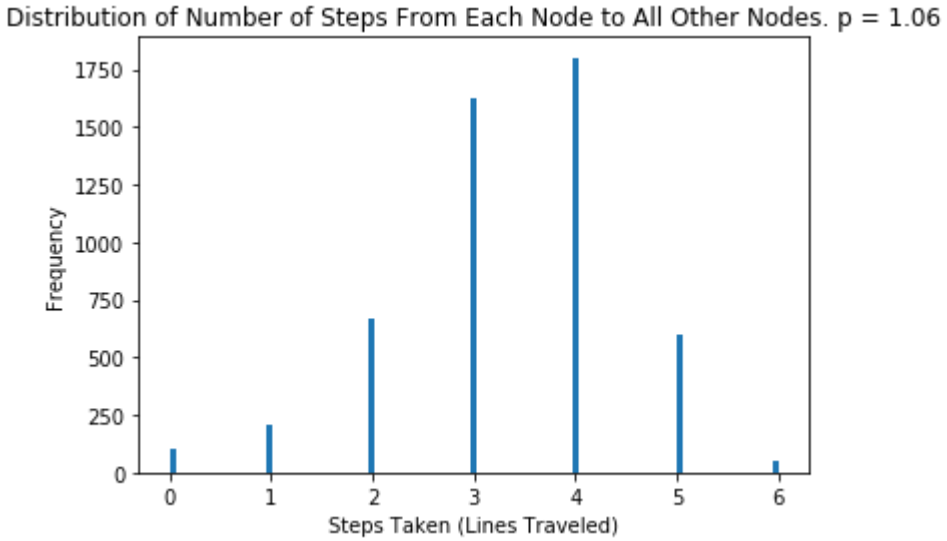


Ah! So as we increase number of shortcuts (by increasing p), the number of steps taken decreases between any 2 nodes, generally. This leads to the idea of Six degrees of separation: "various somewhat uncontrolled studies have shown that any random pair of people in the world can be connected to one another by a short chain of people (typically around six), each of whom knows the next fairly well."

Zoom in on the histogram; how much does it change with p? What value of p would you need to get 'six degrees of separation'?

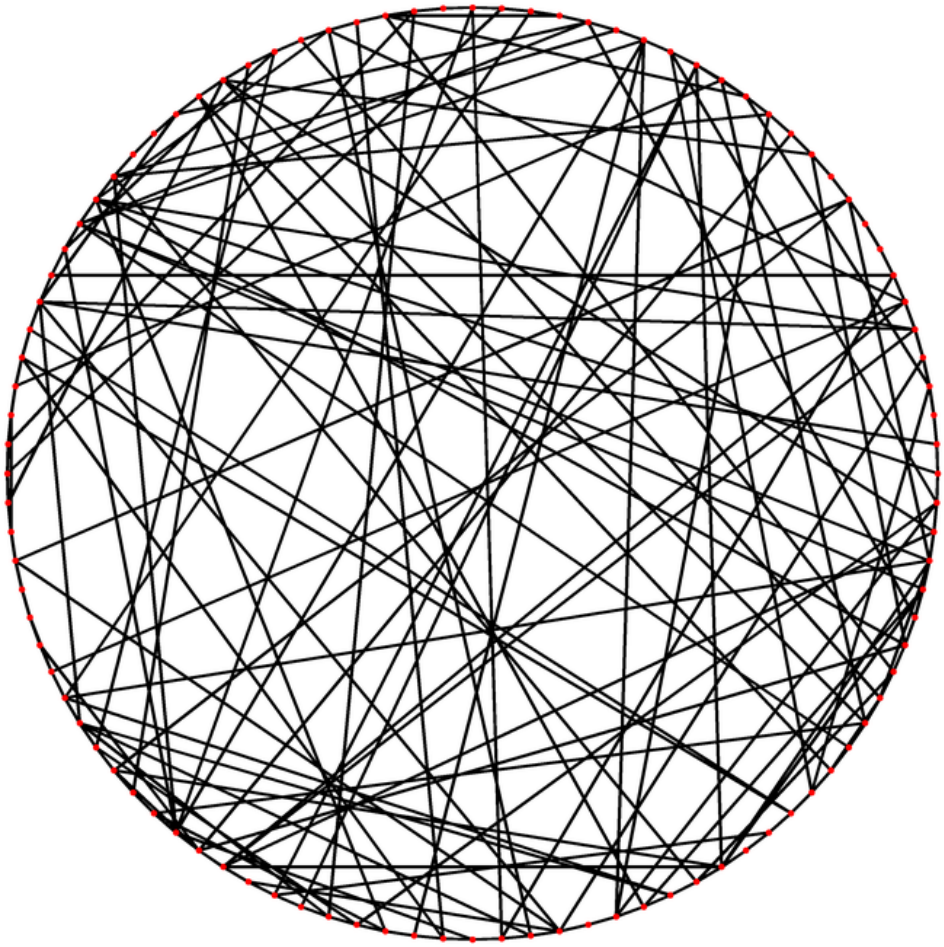
I find six degrees of separation, where the max number of 'jumps/moves' between nodes is 6, at p = 1.06 for when L=100 and Z=2:

In [33]:
```
swsixdeg = g.smallWorldNetwork(100,2,1.06,show=False)
resultssixdeg = g.FindAllPathLengths(swsixdeg)
plt.title('Distribution of Number of Steps From Each Node to All Other
plt.xlabel('Steps Taken (Lines Traveled)')
plt.ylabel('Frequency')
plt.hist(resultssixdeg[1], bins = 101)
plt.show()
```



Here is what a six degree small network looks like:

In [212]:
```
t = g.smallWorldNetwork(100,2,1.06)
```



I found at a p = 1.06 we get six degrees of separation which means there are 106 connections/shortcuts which represents that each node has a relationship with 2 people (from the start, the person to the node's left and to the node's right) as well as an additional 2 people. That means everyone knows at least 2 people and on average everyone knows 2 + 106*2/100 people = 4.12 people on average. (Note, we multiply 106 by 2 because each connection connects 2 people) This is not as important as...

This is not always going to give us six degrees of separation because my code randomly assigns the relationships. What does matter, in any case, is that the distances in relationships are randomly dispersed between short and long. In short, it is the randomness of the length of the shortcuts and the randomness of the nodes in contact that make this work.

Why?

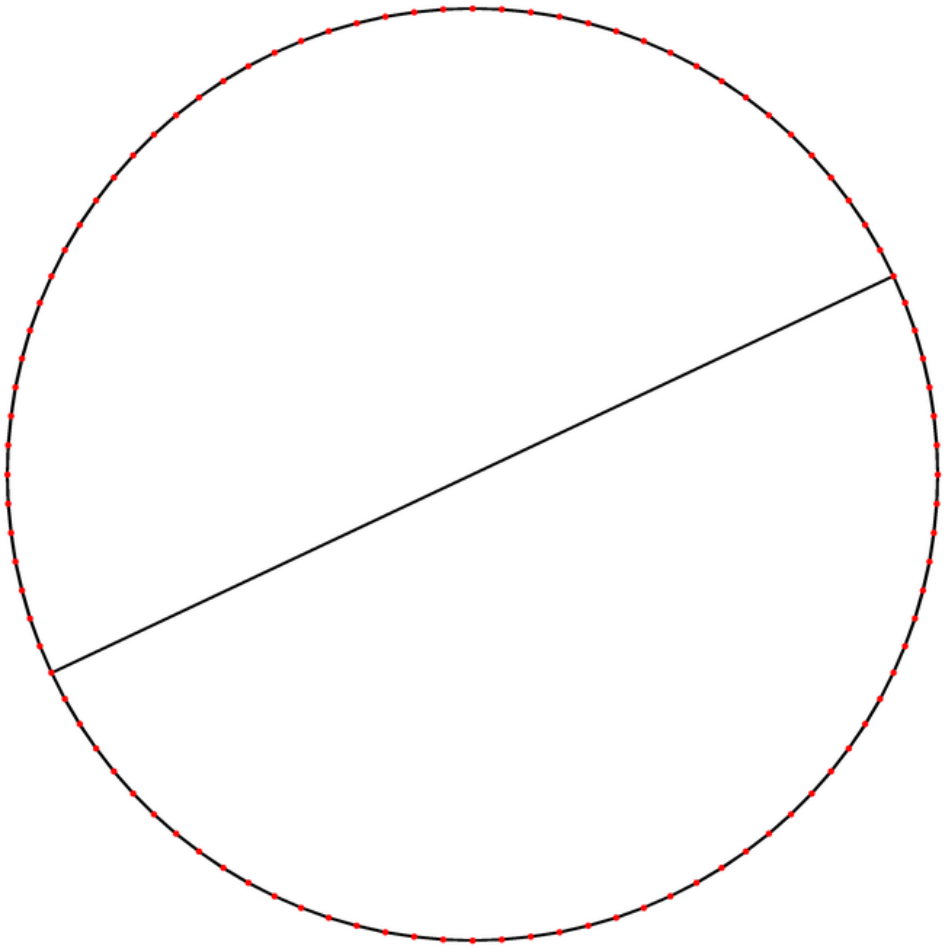
B.) Explore why six degrees of separation occurs

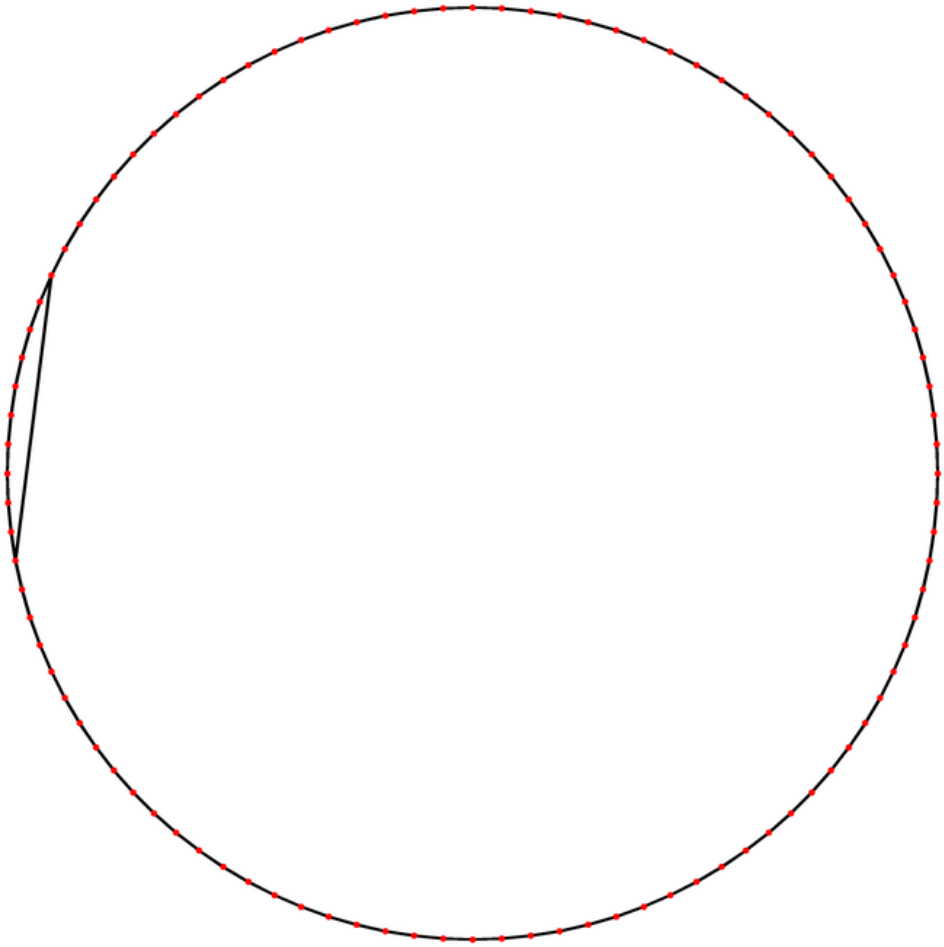
Can we get 6 degrees, on the same network of L = 100 and Z = 2, at a lesser value of p = 1.06 by changing the specified length of shortcuts?

First, I explored how effective one long shortcut is versus one short shortcut.

g.smallWorldNetworkSpecific makes a small world network with shortcuts of a specific length/depth. So in the case that depth = 50, that means there will only be shortcuts that are between nodes that are separated by 50 nodes. In the case that depth = 10, that means there will only be shortcuts added if there are exactly 10 nodes between the 2 nodes that are being connected by an edge/shortcut. This is depicted below:

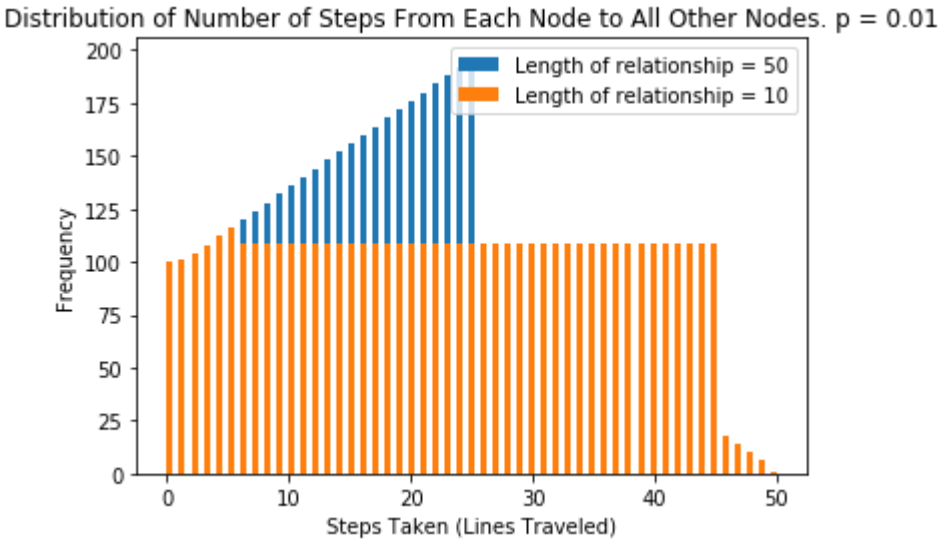
In [38]:
```
depth50 = g.smallWorldNetworkSpecific(100,2,.01,depth=50)
depth10 = g.smallWorldNetworkSpecific(100,2,.01,depth=10)
```

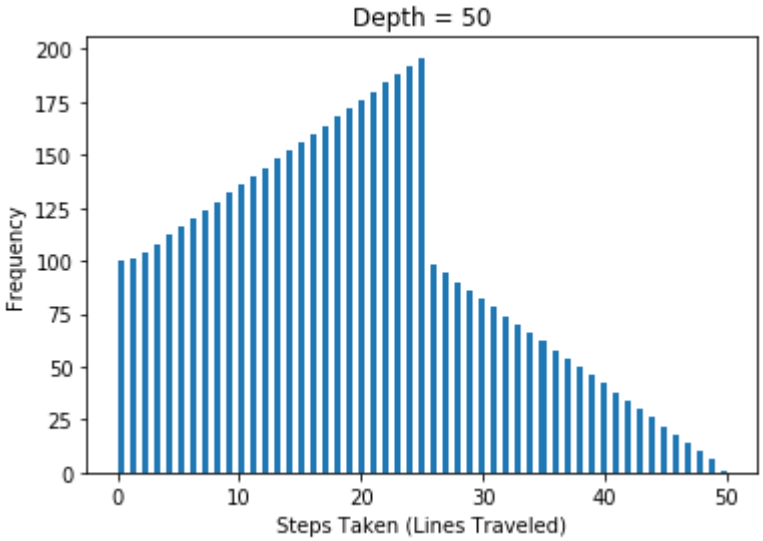


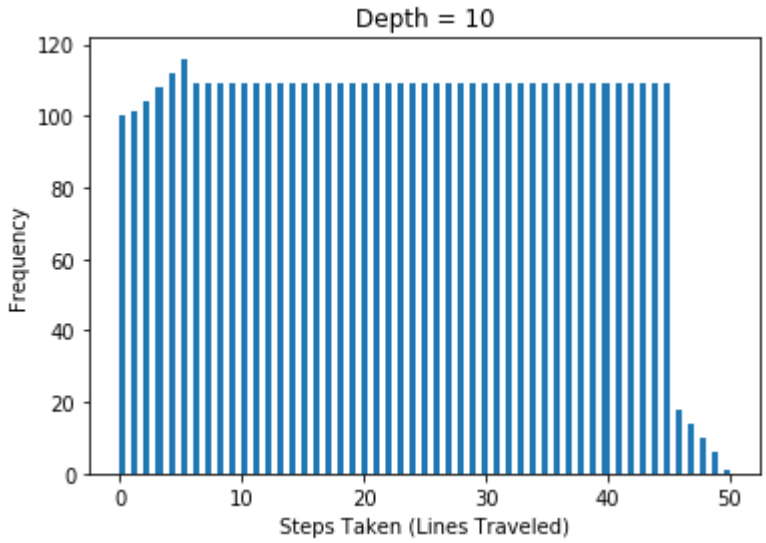


Make a histogram of each network above. One network has a single shortcut that connects 2 nodes a distance of 10 apart (thus, depth = 10) and the other has a single shortcut that connects 2 nodes 50 apart (depth = 50). Compare/Contrast the histograms and overall effectiveness of each shortcut.

In [40]:
```
results50 = g.FindAllPathLengths(depth50)
results10 = g.FindAllPathLengths(depth10)
plt.title('Distribution of Number of Steps From Each Node to All Other
plt.xlabel('Steps Taken (Lines Traveled)')
plt.ylabel('Frequency')
plt.hist(results50[1], bins = 101, label='Length of relationship = 50'
plt.hist(results10[1], bins = 101, label='Length of relationship = 10'
plt.legend()
plt.show()


plt.hist(results50[1], bins = 101, label='Length of relationship = 50'
plt.title('Depth = 50')
plt.xlabel('Steps Taken (Lines Traveled)')
plt.ylabel('Frequency')
plt.show()

plt.hist(results10[1], bins = 101, label='Length of relationship = 10'
plt.title('Depth = 10')
plt.xlabel('Steps Taken (Lines Traveled)')
plt.ylabel('Frequency')
plt.show()
```



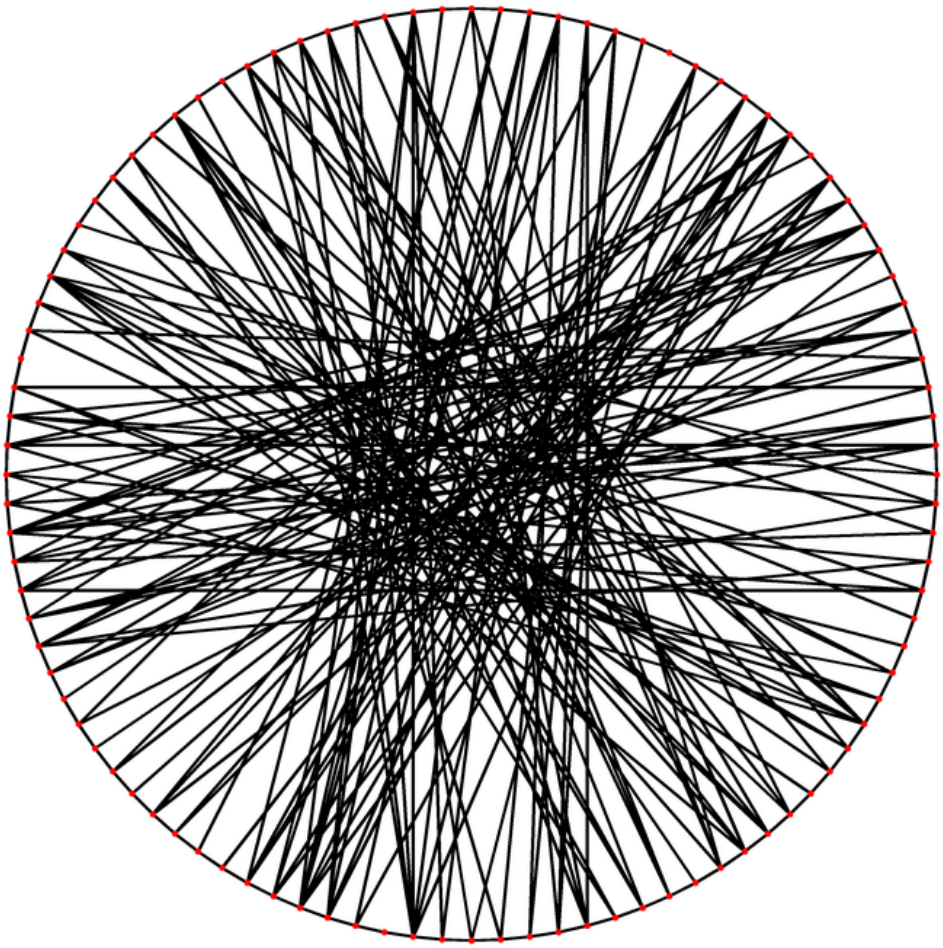


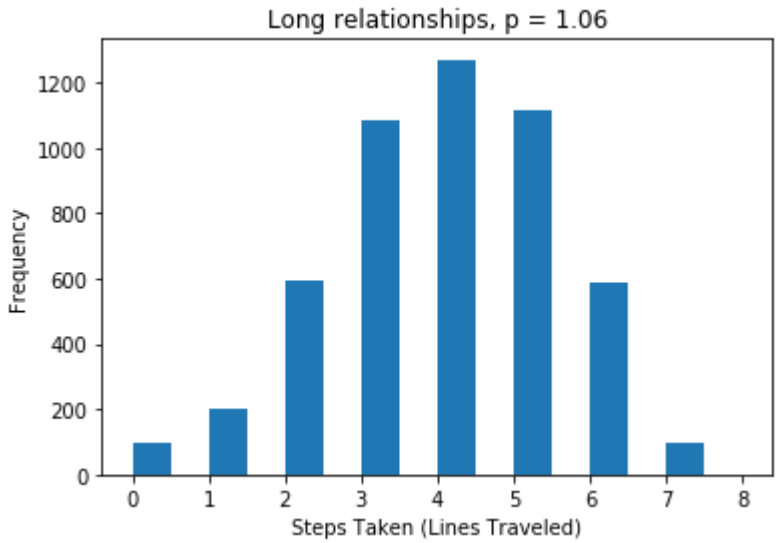


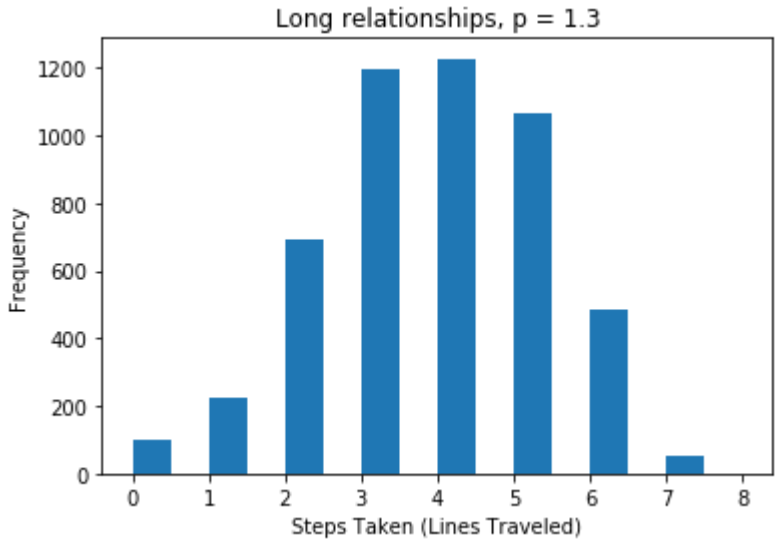
If we were just to go from the histograms above, it appears that longer relationships are more meaningful. That is, longer relationships, relationships that connect two nodes with many nodes that separate them, make it so there are more shorter paths than longer paths between nodes. While the short relationship really does not have an effect on the histogram; only in the slightest way. Therefore, with this logic, if we just stuck with long relationships we should get a smaller p to get six degrees of separation.

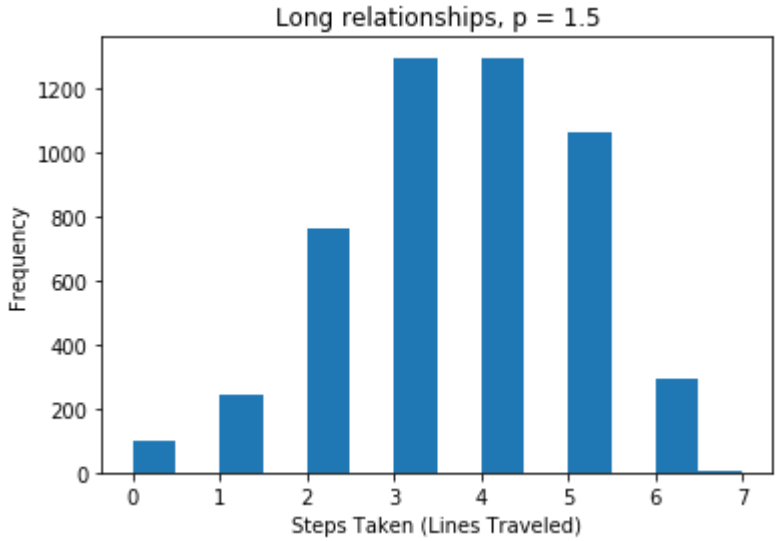
Is this true?

In [236]:
```
long1 = g.smallWorldNetworkLong(100,2,1.5,depth=41)
long2 = g.smallWorldNetworkLong(100,2,1.3,depth=41, show=False)
long3 = g.smallWorldNetworkLong(100,2,1.06,depth=41, show=False)
long4 = g.smallWorldNetworkLong(100,2,2,depth=41, show=False)
resultsLong3 = g.FindAllPathLengths(long3)
plt.hist(resultsLong3[1], bins = 16)
plt.title('Long relationships, p = 1.06')
plt.xlabel('Steps Taken (Lines Traveled)')
plt.ylabel('Frequency')
plt.show()

resultsLong2 = g.FindAllPathLengths(long2)
plt.hist(resultsLong2[1], bins = 16)
plt.title('Long relationships, p = 1.3')
plt.xlabel('Steps Taken (Lines Traveled)')
plt.ylabel('Frequency')
plt.show()

resultsLong1 = g.FindAllPathLengths(long1)
plt.hist(resultsLong1[1], bins = 14)
plt.title('Long relationships, p = 1.5')
plt.xlabel('Steps Taken (Lines Traveled)')
plt.ylabel('Frequency')
plt.show()
```




Long relationships, p = 1.06
Frequency
Steps Taken (Lines Traveled)
0
200
400
600
800
1000
1200
0
1
2
3
4
5
6
7
8





Conclusion: You need randomness, long relationships, and short relationships in order to achieve 6 degrees of separation. Think about it...

If I am trying to give a package to someone in China, and I have a friend who lives somewhere in Beijing (long relationship), so I give him the package. If we have a graph like implemented above, then he/she only knows someone far away, too, so he/she has to give the package to someone in Paris. If we had both short and long relationships then he could give it to someone else in China that is closer to the person I am trying to give the package to. Instead, since he/she only knows people far from him, which is also unrealistic, he/she has to ship it to somewhere like Paris and hope his/her friend in Paris knows the person I am looking for in China.

I know that this is a simplification because Z = 2 in the graph above and therefore each node is connected to its 2 nearest neighbors, but the logic still holds.

As a side note, the 3 histograms above display a nature of adding shortcuts. They follow the law of diminishing returns. As we add more and more shortcuts, they appear to become less and less effective. This is not something I investigate any further, but it appears to be true in all of our histograms: zero to one shortcut has pronounced effects while 100 to 101 does not have as much of an effect.

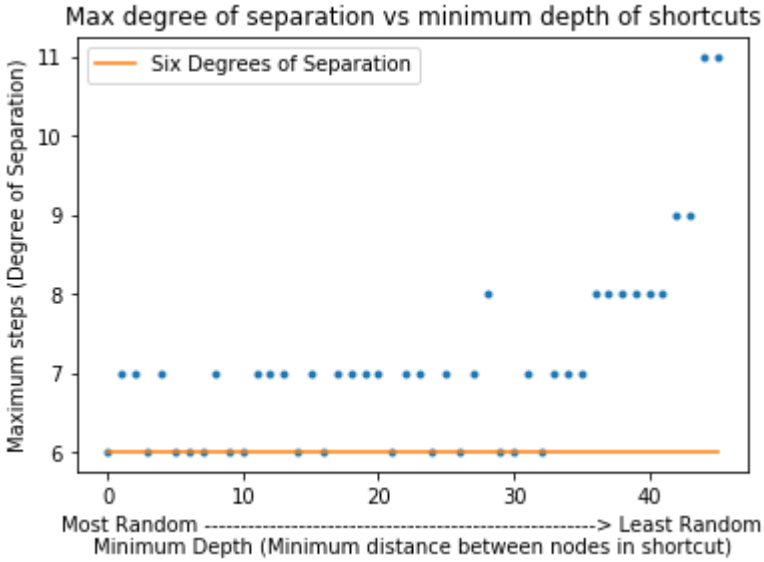
Plot as a function of depth, which is the length of shortcut where length is the amount of nodes between two nodes, the maximum steps of separation to see how the depth effects six degrees of separation at a constant p = 1.06 and. Also, plot the average length of steps in a network against depth to see how depth affects the overall averages of the chart. Use smallWorldNetworkLong, smallWorldNetworkSpecific, and smallWorldNetworkShort to see the overall effects on the plots.

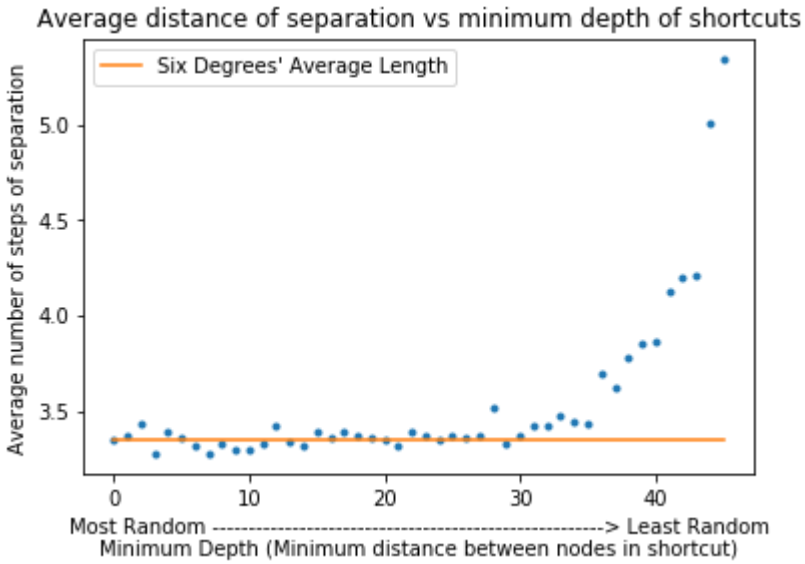
The graphs should be similar.

NOTE: g.smallWorldNetworkLong is different than smallWorldNetworkSpecific. Here, when we give a depth = 10, that means there must be **at least** 10 nodes between the two nodes being connected. This is different than smallWorldNetworkSpecific where there must be **exactly** 10 nodes between the two being connected. Lastly, g.smallWorldNetworkShort is **at most** the given depth.

In [282]:
```
depths = np.linspace(0,45,46)
#print(depths)
xs = []
xs2 = []
for i in depths:
    #print(i)
    sw = g.smallWorldNetworkLong(100,2,1.06,depth=i, show=False)
    xs.append(max(g.FindAllPathLengths(sw)[1]))
    xs2.append(g.FindAveragePathLength(sw))
```

In [297]:
```
plt.xlabel('Most Random ----------------------------------------------
plt.ylabel('Maximum steps (Degree of Separation)')
plt.title('Max degree of separation vs minimum depth of shortcuts')
plt.plot(depths, xs, '.')
plt.plot(depths, 6*np.ones(len(depths)), label='Six Degrees of Separat
plt.legend()
plt.show()
plt.xlabel('Most Random ----------------------------------------------
plt.ylabel('Average number of steps of separation')
plt.title('Average distance of separation vs minimum depth of shortcut
plt.plot(depths, xs2, '.')
plt.plot(depths, 3.35*np.ones(len(depths)), label="Six Degrees' Averag
plt.legend()
plt.show()
```





So as we increase the minimum depth (and thus decrease randomness), both the minimum distance between nodes and the average distance between nodes increases as a polynomial. Since depth, in this case, indicates the minimum number of nodes between two nodes

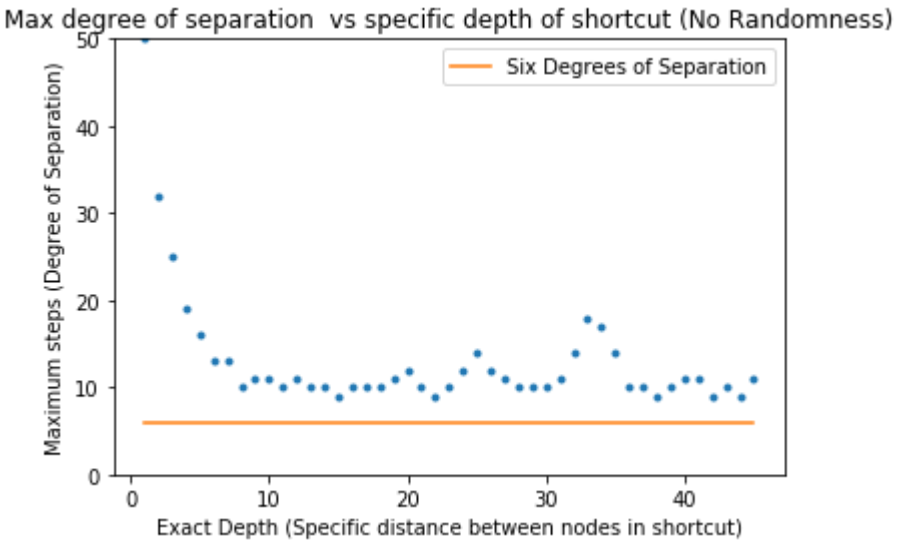
connected by a shortcut, these plots show that you can't have only long relationships to get six degrees of separation at our value of p = 1.06. You need a random arrangement to get six degrees of separation at a reasonable p.

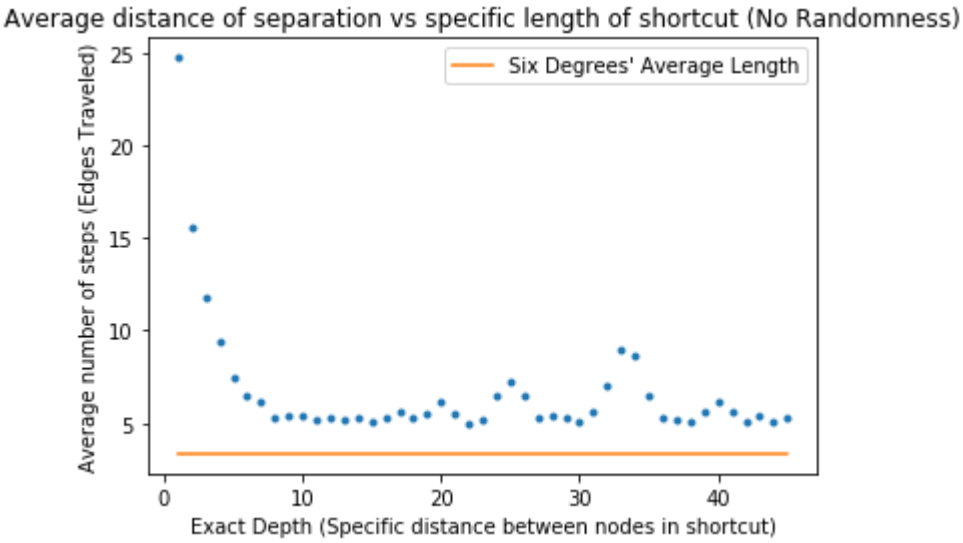
In short, these plots show that we need a random dispersion of shortcuts. We cannot only have shortcuts that connect nodes a distance of 40 away to get the best results. For best results, we need a random array of shortcuts with all different 'depths.'

But does this really prove that the shortcuts need to be random? Let's see what happens when we set all the 'length' of the shortcuts to be identical. In other words, if we set depth=10, then that means there only exists shortcuts that connect nodes a distance of 10 away. To do this, we use g.smallWorldNetworkSpecific, not g.smallWorldNetworkLong.

In [278]:
```
depths2 = np.linspace(1,45,45)
#print(depths)
xsp2 = []
xs2p2 = []
for i in depths2:
    #print(i)
    sw = g.smallWorldNetworkSpecific(100,2,1.06,depth=i, show=False)
    xsp2.append(max(g.FindAllPathLengths(sw)[1]))
    xs2p2.append(g.FindAveragePathLength(sw))
```

In [302]:
```
plt.xlabel('Exact Depth (Specific distance between nodes in shortcut)'
plt.ylabel('Maximum steps (Degree of Separation)')
plt.title('Max degree of separation  vs specific depth of shortcut (No
plt.plot(depths2, xsp2, '.')
plt.plot(depths2, 6*np.ones(len(depths2)), label='Six Degrees of Separ
plt.legend()
plt.ylim(0,50)
plt.show()
plt.xlabel('Exact Depth (Specific distance between nodes in shortcut)'
plt.ylabel('Average number of steps (Edges Traveled)')
plt.title('Average distance of separation vs specific length of shortc
plt.plot(depths2, xs2p2, '.')
plt.plot(depths2, 3.35*np.ones(len(depths2)), label="Six Degrees' Aver
plt.legend()
plt.show()
```





Yep randomness is needed... We do not see six degrees of separation anywhere (orange line in first plot) for p = 1.06 when we set a specific depth. We only ever achieve ~10 degrees of separation with no randomness around a depth = 10. The average number of steps hit a

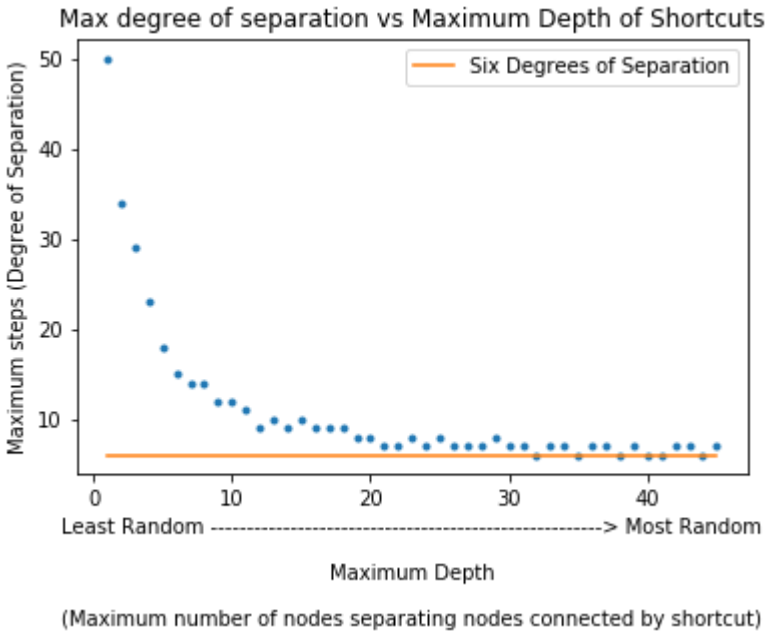
minimum around 5.0 steps (at a depth about 10). We will later find the average number of steps for our original six degrees of separation network is 3.35 (orange line in second plot), much less than 5.

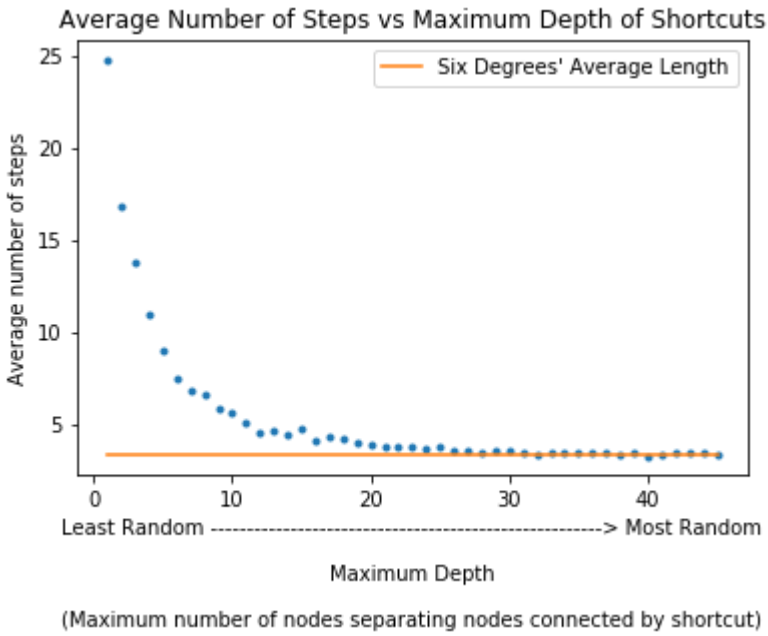
It appears that when the depth of the shortcuts are around 10 that they have the most pronounced effects. Depth of 10 is about 1/5th the nodes... However, this shows randomness is still even more effective than setting a specific depth of shortcuts.

To complete the analyzation, we will conclude by using g.smallWorldNetworkShort which sets the maximum depth, instead of g.smallWorldNetworkLong which sets the minimum depth.

In [252]:
```
depths3 = np.linspace(1,45,45)
#print(depths)
xsp3 = []
xs2p3 = []
for i in depths2:
    #print(i)
    sw = g.smallWorldNetworkShort(100,2,1.06,depth=i, show=False)
    xsp3.append(max(g.FindAllPathLengths(sw)[1]))
    xs2p3.append(g.FindAveragePathLength(sw))
```

In [300]:
```
plt.xlabel('Least Random ----------------------------------------------
plt.ylabel('Maximum steps (Degree of Separation)')
plt.title('Max degree of separation vs Maximum Depth of Shortcuts')
plt.plot(depths3, xsp3, '.')
plt.plot(depths2, 6*np.ones(len(depths2)), label='Six Degrees of Separ
plt.legend()
plt.show()
plt.xlabel('Least Random ----------------------------------------------
plt.ylabel('Average number of steps')
plt.title('Average Number of Steps vs Maximum Depth of Shortcuts')
plt.plot(depths3, xs2p3, '.')
plt.plot(depths2, 3.35*np.ones(len(depths2)), label="Six Degrees' Aver
plt.legend()
plt.show()
```





Once again, randomness is necessary to achieve 6 degrees of separation, as expected by this

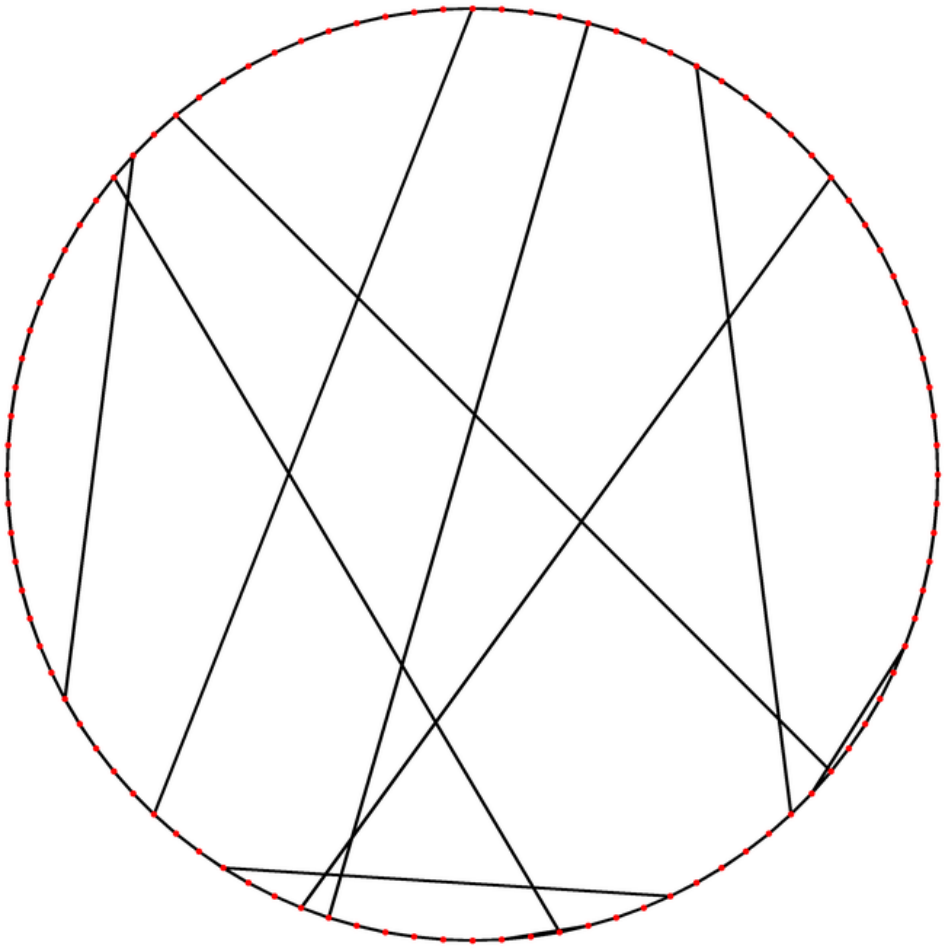
point.

What is intersting to note, and something that is discussed later, is that average number of steps ('Average Length') in a network seems to proportionally connected to degree of separation.

C.) Average length

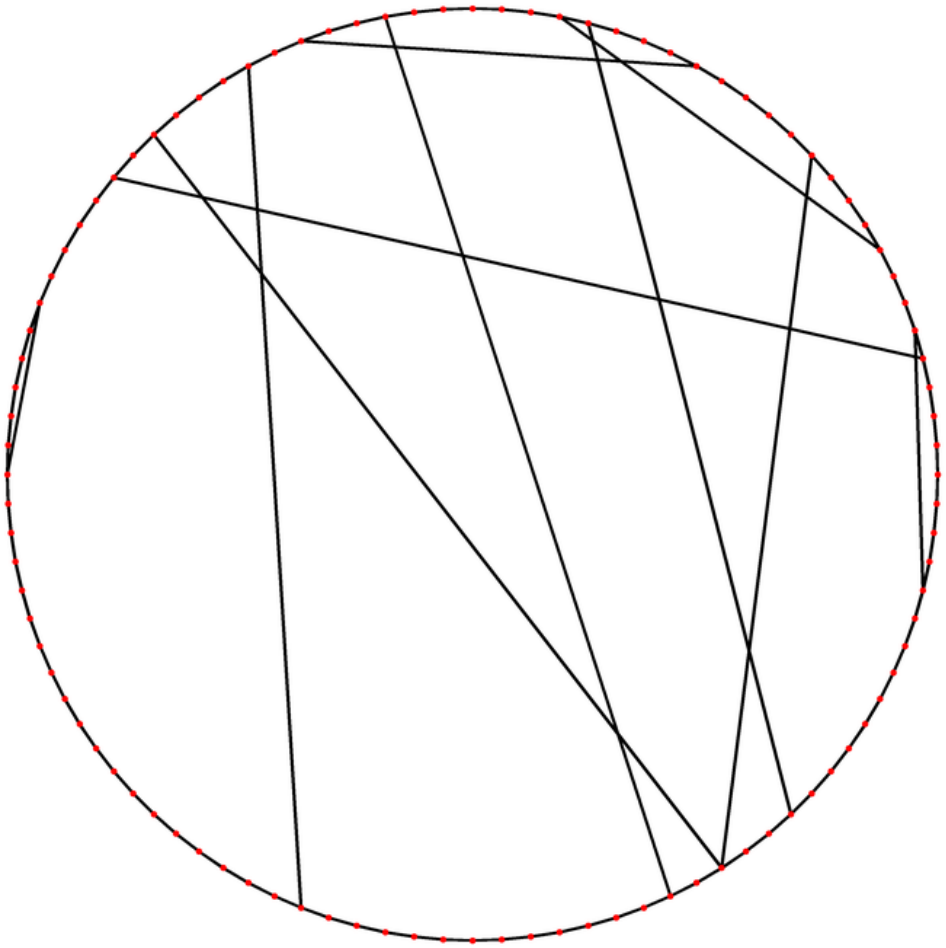
FindAveragePathLength(graph). Try L = 100, Z = 2, p = .1, and average should be around l = 10, why does it vary?

In [80]:
```
avg1sw = g.smallWorldNetwork(100,2,.1)
avg1 = g.FindAveragePathLength(avg1sw)
print('The average length between two nodes is: '+str(avg1))
```



```
The average length between two nodes is: 9.384950495049505
```

In [82]:
```
avg2sw = g.smallWorldNetwork(100,2,.1)
avg2 = g.FindAveragePathLength(avg2sw)
print('The average length between two nodes is: '+str(avg2))
```



```
The average length between two nodes is: 9.714059405940594
```

They are different because the lines and shortcuts are placed randomly, as we explored earlier, randomnness has a huge effect on the outputs of the networks.

What is average length for the p we found for six degrees of separation? 1.06 = p.

In [85]:
```
print(g.FindAveragePathLength(swsixdeg))
```

```
3.352277227722772
```

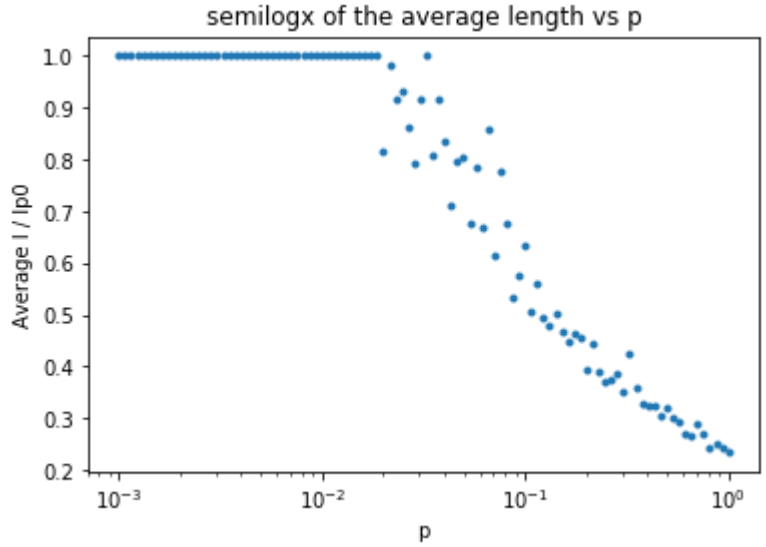
The average length is sub-6, which makes sense in order to achieve a maximum distance of 6 the average must be less.

As we add more lines, the average path length decreases, as we would expect.

Plot the average path length between nodes l(p) divided by l(p = 0) for Z = 2, L = 50, with p on a semi-log plot from p = 0.001 to p = 1. (Hint: Your curve should be similar to that of with Watts and Strogatz [142, fig. 2], with the values of p shifted by a factor of 100; see the discussion of the continuum limit below.) Why is the graph fixed at one for small p?

In [88]:
```
ps = 10**(-1 * np.linspace(0,3,100))
Z = 2
L = 50
lp0 = g.FindAveragePathLength(g.smallWorldNetwork(L,Z,0, show=False))
xs = []
for p in ps:
    x = g.FindAveragePathLength(g.smallWorldNetwork(L,Z,p, show=False)
    xs.append(x/lp0)
```

In [89]:
```
plt.semilogx(ps,xs, '.')
plt.ylabel('Average l / lp0')
plt.xlabel('p')
plt.title('semilogx of the average length vs p')
plt.show()
```



Why is the graph fixed at one for small p?

Because for small p, the amount of shortcuts added is 0 until around 10**-1.4

Large N and the emergence of a continuum limit This shows that no shortcuts are added for a really small p and they are identical to having a p=0. This is the idea that the number of shortcuts is proportional to the equation of shortcuts = p * L * Z / 2, so if p is really small, L * Z needs to be really big. As L increases, p can decrease proportionally to have the same number of shortcuts. This idea is explored in the next section, section D.

Use FindAverageAveragePathLength on several random small world networks with L=100,Z=2, and p=0.1. How wildly do these fluctuate?

In [178]:
```
smallWorlds = []
for i in np.arange(30):
    smallWorlds.append(g.smallWorldNetwork(100,2,.1, show=False))
resultAvgStdDev = g.FindAverageAveragePathLength(smallWorlds)
print(resultAvgStdDev)
```

```
(9.887438943894386, 0.9193688118606976)
```

In [179]:
```
print('Average length: '+str(resultAvgStdDev[0]))
print('Std. Deviation: '+str(resultAvgStdDev[1]))
```

```
Average length: 9.887438943894386
Std. Deviation: 0.9193688118606976
```

We used FindAverageAveragePathLength on several (30) random small world networks with L=100, Z=2, and p = 0.1 and found that the average length for all of them was 9.88 +- 0.92. They fluctuate a little bit, but not too much. Let's see if there is a lot of fluctuation with our six degree map's arguments of L=100, Z=2, and p=1.06:

Describe qunatitatively through standard deviation: how consistently will we achieve six degrees of separation with L = 100, Z = 2, and p = 1.06

In [315]:
```
smallWorldsSixDeg = []
for i in np.arange(30):
    smallWorldsSixDeg.append(g.smallWorldNetwork(100,2,1.06, show=Fals
resultAvgStdDevSixDeg = g.FindAverageAveragePathLength(smallWorldsSixD
print(resultAvgStdDevSixDeg)
```

```
(3.3917953795379545, 0.04704100227837375)
```

In [316]:
```
print('Average length: '+str(resultAvgStdDevSixDeg[0]))
print('Std. Deviation: '+str(resultAvgStdDevSixDeg[1]))
```

```
Average length: 3.3917953795379545
Std. Deviation: 0.04704100227837375
```

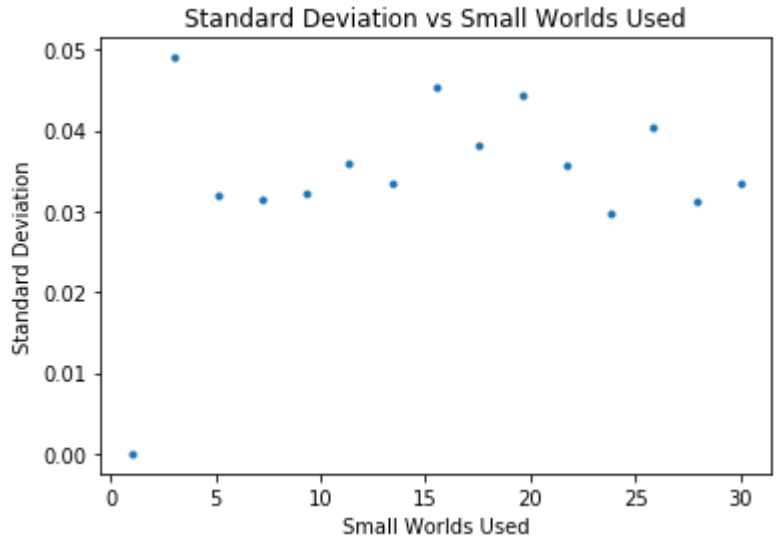
There is really little deviation in our six degree network of average path lengths, which is good and means we should consistently get a six degree network with L = 100, Z = 2, and p = 1.06. This indicates that at a p = 1.06, we will often get the same results in terms of average path length. Because the average number of steps ('Average Length') in a network seems to proportionally connect to degree of separation, we can say that little standard deviation here indicates how consistently we will achieve six degrees of separation.

There is less deviation in L = 100, Z = 2, and p = 1.06, than with L = 100, Z = 2, and p = 0.1 because there is more possible variation with fewer shortcuts.

How many smallworlds should we use to get an accurate standard deviation? Plot standard deviation vs number of small worlds used to see where it plateaus. Do it with constant L = 100, Z = 2, and p = 1.06. Try using 1-30 small worlds.

In [336]:
```
smallWorldsAcc = np.linspace(1,30,15)
smallWorldsAccResults = []
for j in smallWorldsAcc:
    smallW = []
    for i in np.arange(j):
        smallW.append(g.smallWorldNetwork(100,2,1.06, show=False))
    resultAvgStdDevAcc = g.FindAverageAveragePathLength(smallW)[1]
    smallWorldsAccResults.append(resultAvgStdDevAcc)
```

In [337]:
```
plt.plot(smallWorldsAcc, smallWorldsAccResults, '.')
plt.xlabel('Small Worlds Used')
plt.ylabel('Standard Deviation')
plt.title('Standard Deviation vs Small Worlds Used')
plt.show()
```



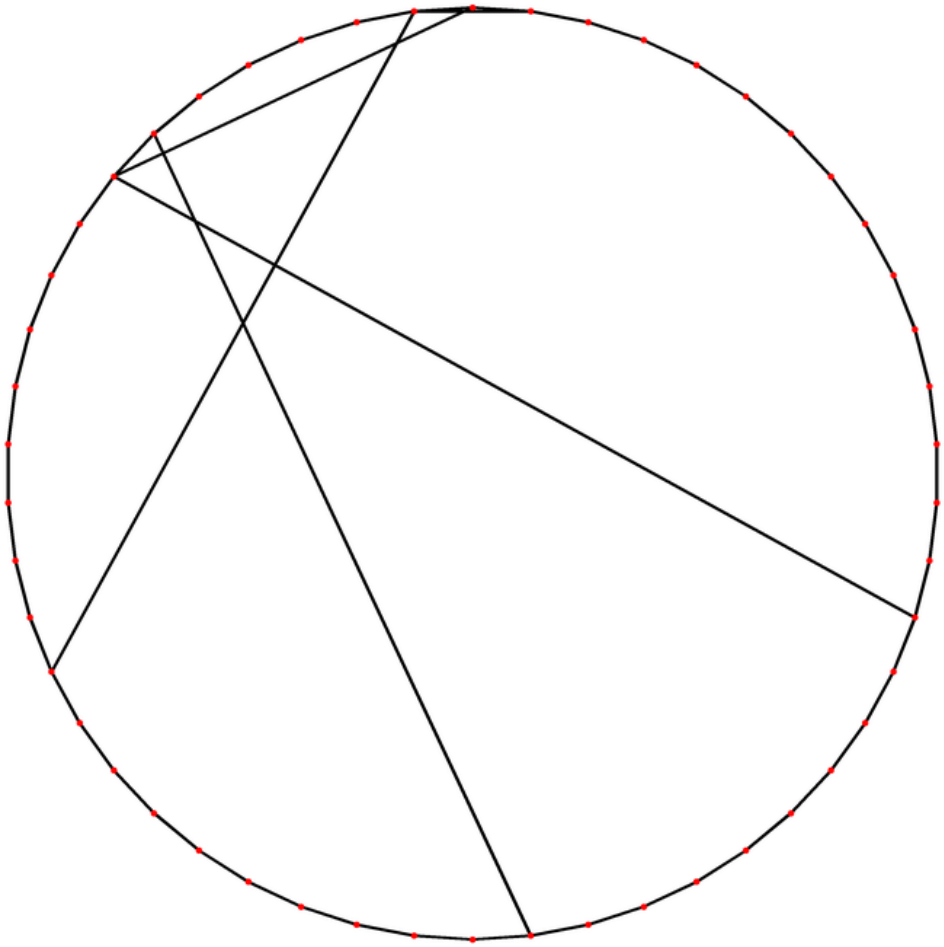
It appears that using 10 small worlds is good enough.

D.) Explore ratios of creating similar looking networks with different values

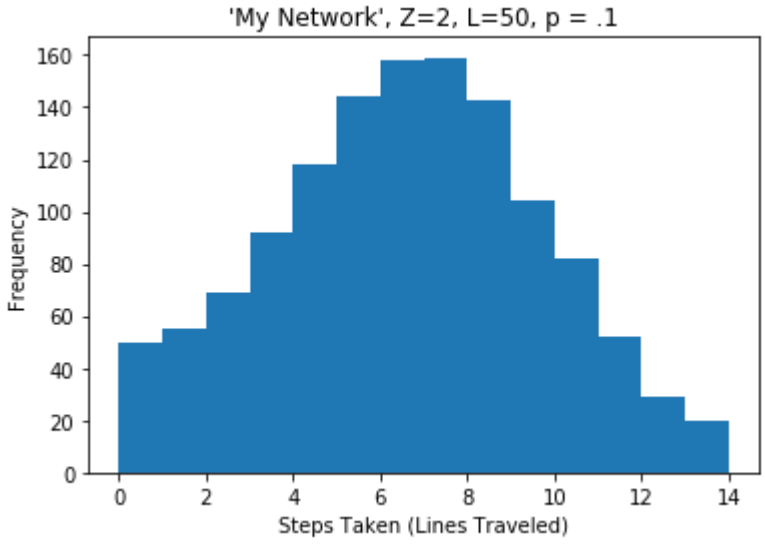
(Z = 2, L = 50) at p = 0.1; create and display circle graphs of Watts and Strogatz's geometry (Z = 10, L = 1000) at p = 0.1 and p = 0.001. Which of their systems looks statistically more similar to yours?

In [95]:
```
print("1st, 'my' graph")
Z = 2
L = 50
p = 0.1
swd1 = g.smallWorldNetwork(L,Z,p)
```

```
1st, 'my' graph
```

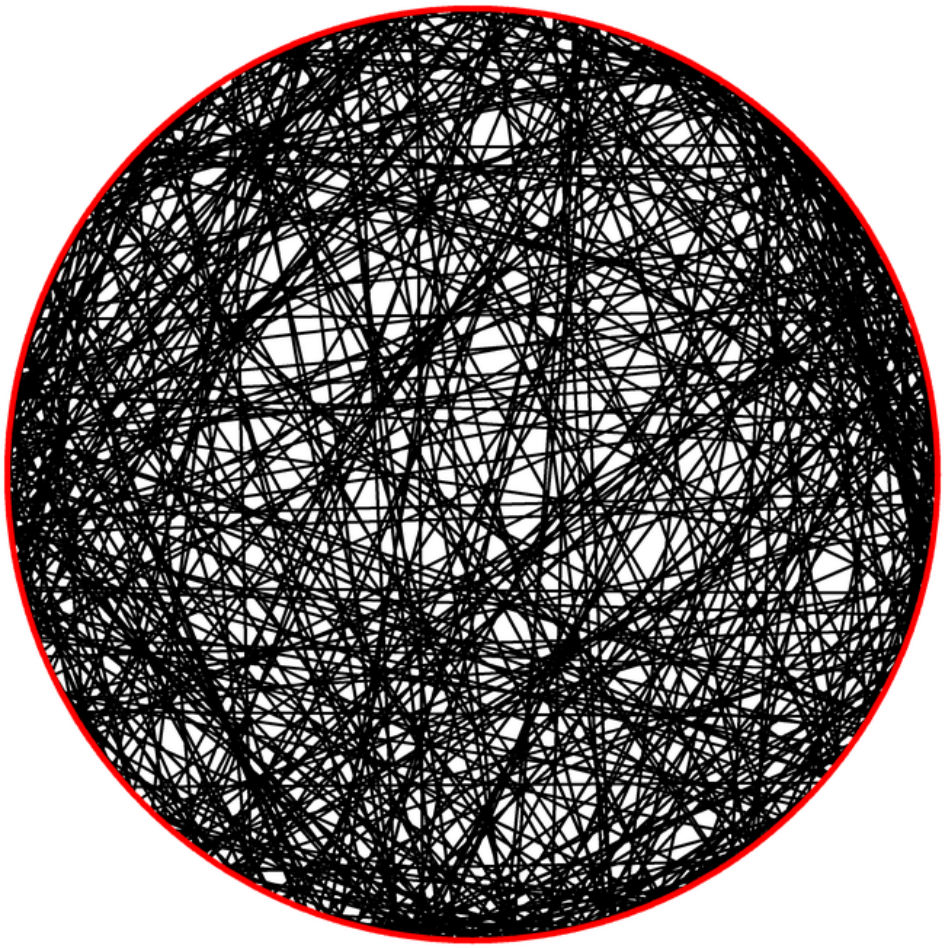


In [183]:
```
resultsmy = g.FindAllPathLengths(swd1)
plt.hist(resultsmy[1], bins = 14)
plt.title("'My Network', Z=2, L=50, p = .1")
plt.xlabel('Steps Taken (Lines Traveled)')
plt.ylabel('Frequency')
plt.show()
```

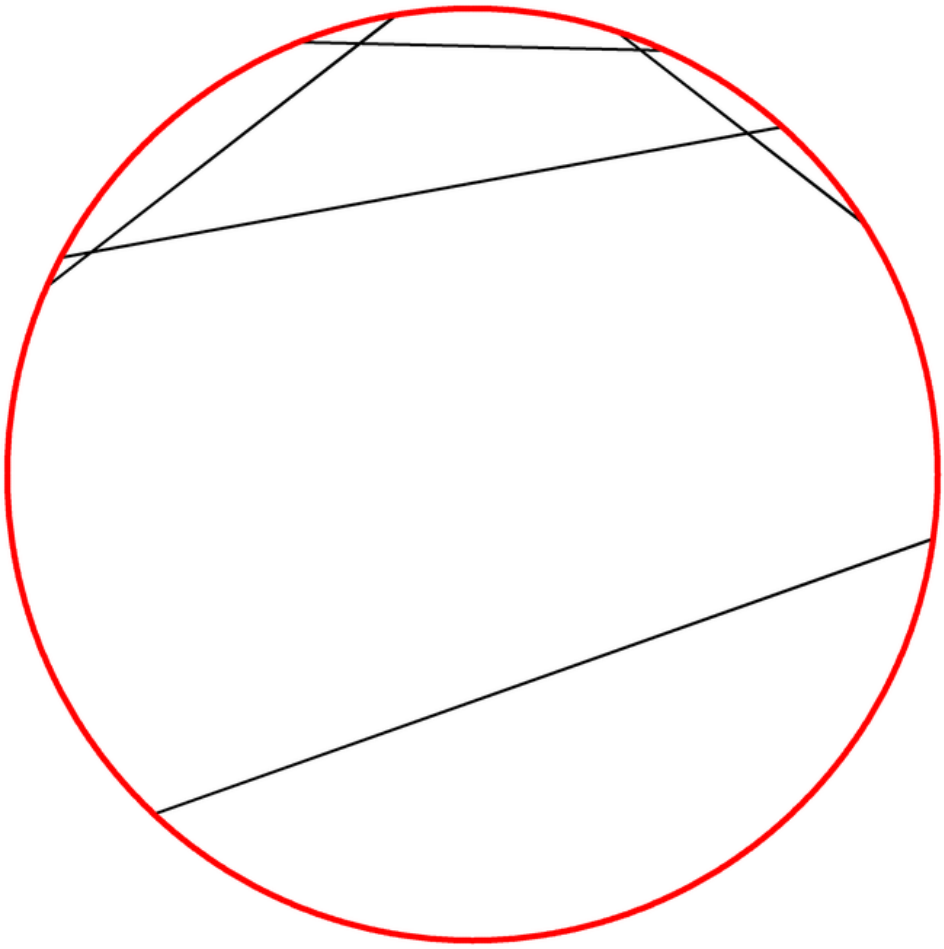


In [96]:
```
print("1st, Watts and Strogatz's geometry graph")
Z = 10
L = 1000
p = 0.1
swd2 = g.smallWorldNetwork(L,Z,p)
print("2nd, Watts and Strogatz's geometry graph")
Z = 10
L = 1000
p = 0.001
swd3 = g.smallWorldNetwork(L,Z,p)
```

```
1st, Watts and Strogatz's geometry graph
```



```
2nd, Watts and Strogatz's geometry graph
```



Which one looks more like my graph? The 2nd Watts and Strogatz's geometry of course! Even though p is 100 times smaller in the second graph.

Why?

This is because the number of shortcuts are based on the formula: p * L * Z / 2. Since we increase Z by 5 times and increase L by 20 times, that means we need to decrease p by 100 times to have the same number of shortcuts randomly drawn.

Note: I was going to try and compare their histograms outputs, but with 1000 nodes the computation seems to never finish: I am sure it would, it just would take a night to run, or two...

Lastly,

Relating to Real-World and Applications

In addition to relating to people and six degrees of separation, in the real-world, we can relate networks in the following ways (in would be hard to find data on the following and to implement, but it could be done):

Kevin Bacon. A similar investigation to six degrees of separation deals with actors/actresses that have acted with Kevin Bacon. It is hypothesized that there is something like 'six degrees of separation' with all actors and Kevin Bacon; all actors/actresses have acted with someone that has acted with someone that has... acted with Kevin Bacon. Where is the degree of separation here? This could be explored with networks.

The World Wide Web/Internet. The world wide web is not an undirected network/graph, instead it is a directed graph. When 'crawling' the world wide web, we can see that certain websites link to other websites, but that does not mean the relationship is in a two directional way. Just because a link on website A sends you to website B does not mean website B will have a link to website A. The world wide web and links between websites could be represented in a graph. In the graph, the nodes would represent websites and the edges would represent links. In addition, I have not investigated 'betweenness' of the graphs which describes the number of times an edge or a node is used to get to another node in a shortest path traversal. However, in the World Wide Web, Google or other search engines would have a high number of 'betweenness' because they have a link to a lot of websites. It is especially hard to implement code to traverse the links on the World Wide Web due to the idea that it is a directed network.

Airlines. Airline flight paths could be represented by graphs where nodes represent locations and the edges represent the flight path. Once again, this is an directed graph because flights are not always returned. In other words, if a plane flies from Pittsburgh to Chicago, that does not mean there is a flight from Chicago to Pittsburgh. The edges in this case could have a value to represent price, time on flight, or any other attribute to be more descriptive. Thus, if the edges had a price associated with them, we could explore things like the cheapest way to fly from Pittsburgh to Indonesia, etc. In this sense, shortest path is not using the fewest amounts of edges; rather, it is taking the path that is most cost effective.

Social Media Networks. Lastly, social media networks can be represented by graphs in regards to relationships between people on the social media. For simplicity, we can say that if person X is friends with person Y, then person Y is friends with person X. Thus, these graphs are undirected. These networks are used on real social media platforms to connect people. When you get 'suggested friends' that is because of the network than connects people on the social media form. If we were to implement breadth-first search we could investigate some cool things. BFS in this context could answer the question of is one person connected to another person, and if they are connected what is the shortest path between them and who are involved in the connection of people. 'Betweenness' here would represent how many people someone knows and thus perhaps their 'popularity.'

A lot of these examples I either learned about or heard about in CIS-320 with Dr. Holland-Minkley.

Note

I wish that I would've explored more about how increasing the value of Z affected the degrees of separation and how it compared to increasing the value of p. However, I believe that I have included enough in this document to not explore this at this time.

Thank you Dr. Hallenbeck,

James

In [ ]: