Donovan Guelde
CSCI 5352
PS 2

1.a. $< k_m >= (n_m - 1)p_m$

$= (n_m - 1)(\frac{A}{(n_m-1)^\beta})$

$= \frac{A}{(n_m-1)^{\beta-1}}$

b. $C = \frac{c}{n-1}$ (Eq. (12.11)
$= p = A(n_m - 1)^\beta$

c. If$< C_m >$ and $< k >^{\frac{-\beta}{1-\beta}}$ are proportional, then:

$\frac{<C_m>}{<k>^{\frac{-\beta}{1-\beta}}} =$ some constant k
$\frac{<C_m>}{<k>^{\frac{-\beta}{1-\beta}}} = \frac{A(n_m-1)^-\beta}{A(n_m-1)^{1-\beta}} = (n_m - 1)^{-1}$, which approaches zero as n grows larger

a value of $\frac{3}{7}$ for β would result in $< k >^{-.75}$

2.a. Number Possible Triangles = $\binom{n}{3}$

prob(3 vertexes are connected) = $(\frac{c}{n-1})^3$; for large $n, = (\frac{c}{n})^3$

therefore, # triangles = $\binom{n}{3}(\frac{c}{n})^3$

$= (\frac{n!}{6(n-3)!})(\frac{c^3}{n^3})$

$= (\frac{(n)(n-1)(n-2)(n-3)!}{6(n-3)!})(\frac{c^3}{n^3})$, which approaches $\frac{c^3}{6}$ for large n

b. # connected triples = $\binom{n}{3}(\frac{c}{n})^2(3)$

$= 3(\frac{n!}{6(n-3)!})(\frac{c^2}{n^2}) = \frac{3(n)(n-1)(n-2)(n-3)!(c^2)}{6(n-3)!n^2}$
which approaches $\frac{nc^2}{2}$ as n grows large

c. $C = \frac{3(number\ of\ triangles)}{(number\ of\ connected\ triples)}$ (Eq (7.41))
$= \frac{\frac{3c^3}{6}}{\frac{nc^2}{2}} = \frac{c}{n}$
$C = \frac{c}{n-1}$ (Eq. (12.11)); approaches $\frac{c}{n}$ for large n

3. Eq. (7.29) states that $C_i = \frac{1}{l_i} = \frac{n}{\sum_j d_{ij}}$
Distance from node A to any node j in component B is given by $d_{Bj} + 1$
Likewise, distance from node B to any node i in component A is given by $d_{Ai} + 1$
For all nodes in component B, $\sum d_A = \sum d_B + n_b$
and for all nodes in component A, $\sum d_B = \sum d_A + n_a$
And, by Eq. (7.29), $\sum d_B = \frac{n}{C_B}$ and $\sum d_A + \frac{n}{C_A}$

So, $\frac{n}{C_A} + n_A = \frac{n}{C_B} + n_B$

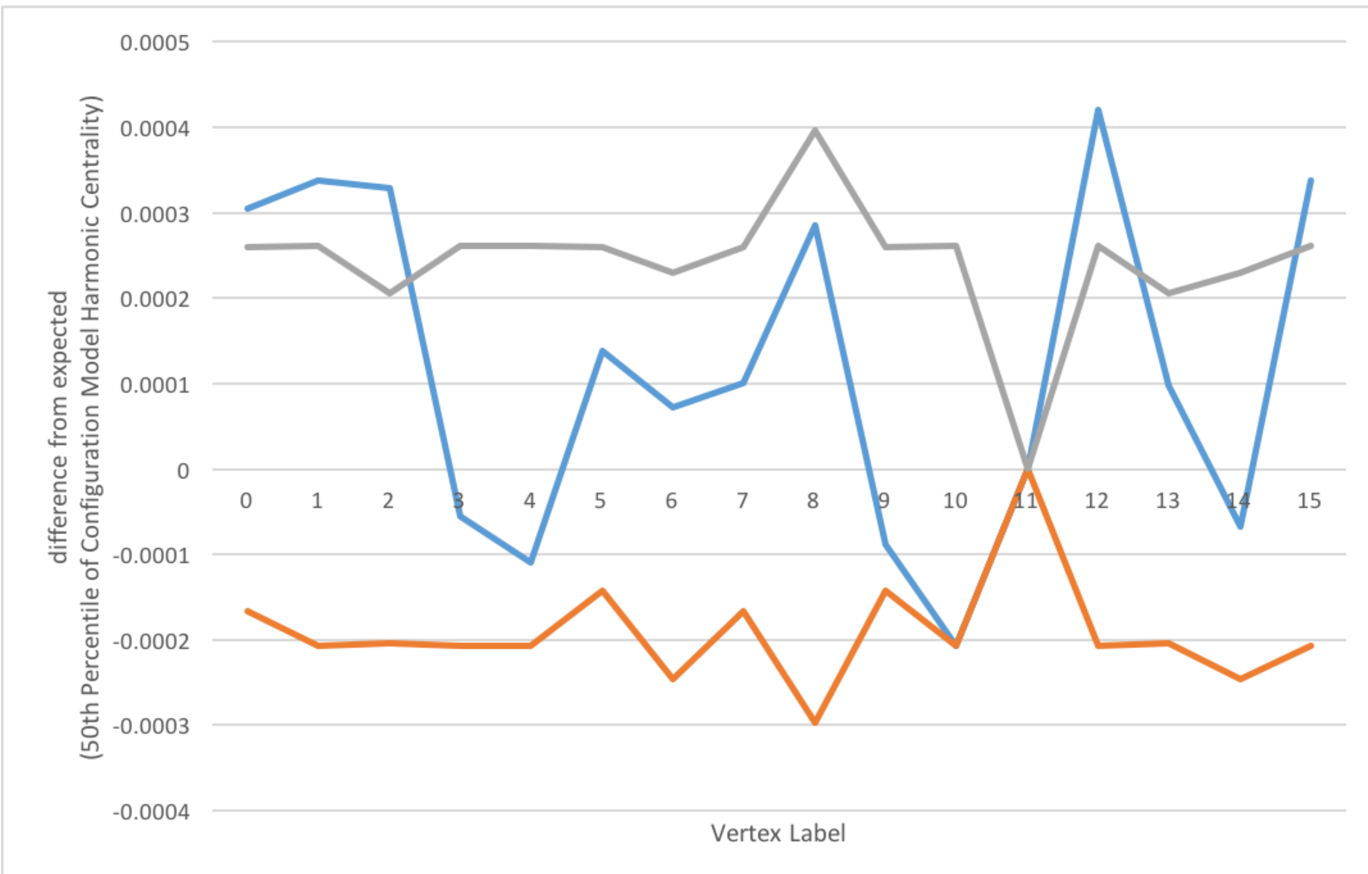
$\frac{1}{C_A} + \frac{n_A}{n} = \frac{1}{C_B} + \frac{n_B}{n}$

4.Total number of paths in the original tree = $n(n-1)$; approaches n^2 as n grows larger
number paths passing through any vertex v = n^2 - (# paths internal to v's branches)
paths internal to v's branches = $\sum_{i=1}^{k} n_i^2$ where k = degree of tree
so $b_v = n^2 - \sum_{i=1}^{k} n_i^2$

5.

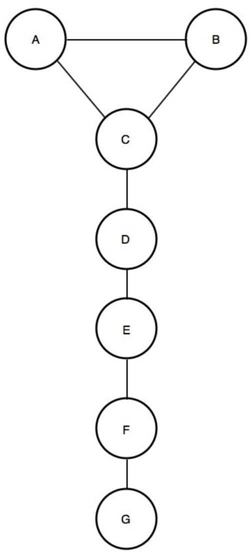
Degree Centrality		Harmonic Centrality		Eigenvector Centrality		Betweenness Centrality	
Medici	6	Medici	0.00267	Medici	0.43031	Medici	0.00233
Guadagni	4	Ridolfi	0.00238	Strozzi	0.35598	Guadagni	0.00155
Strozzi	4	Albizzi	0.00230	Ridolfi	0.34155	Albizzi	0.00123
Albizzi	3	Tornabuoni	0.00230	Tornabuoni	0.32584	Bischeri	0.00101
Bischeri	3	Guadagni	0.00222	Guadagni	0.28912	Salviati	0.00101
Castellani	3	Barbadori	0.00208	Bischeri	0.28280	Tornabuoni	0.00093
Peruzzi	3	Strozzi	0.00208	Peruzzi	0.27573	Strozzi	0.00090
Ridolfi	3	Bischeri	0.00190	Castellani	0.25903	Ridolfi	0.00085
Tornabuoni	3	Castellani	0.00185	Albizzi	0.24396	Barbadori	0.00077
Barbadori	2	Salviati	0.00185	Barbadori	0.21170	Castellani	0.00071
Salviati	2	Acciaiuoli	0.00175	Salviati	0.14592	Peruzzi	0.00063
Acciaiuoli	1	Peruzzi	0.00175	Acciaiuoli	0.13215	Lambertesch	0.00055
Ginori	1	Ginori	0.00159	Lambertesch	0.08879	Acciaiuoli	0.00053
Lambertesch	1	Lambertesch	0.00155	Ginori	0.07492	Pazzi	0.00053
Pazzi	1	Pazzi	0.00136	Pazzi	0.04481	Ginori	0.00044
Pucci	0	Pucci	0.00000	Pucci	0.00000	Pucci	0.00001

In all measures of centrality used, the Medici family emerges as the most central family(vertex) in the network of prominent 14th century Florentine families.This is in full agreement with the explanation for the Medici family's success, as given by Padgett and Andel in 1993. What was unexpected, at least to me, was the variation in rank of the other families. The Guadagni family was in the number two position in Degree and Betweenness centralities, but dropped to number five in both Harmonic and Eigenvector centralities, for example. Many of the other families experienced radical changes in their positions, depending on the index used. One possible explanation for this is the number of secondary hubs in the network. The Ridolfi, Tornabuoni, and Strozzi families, among others, are all well-connected, with exact rankings varying by method used.



While the Medici family's Harmonic Centrality score is approaching the 75th percentile, it is still within the 'norm' of 25th to 75th percentile, when compared to a run of 100,000 configuration models with identical degree sequence. This would indicate that the Medici family, while having a high degree centrality score, is no more central to the network than should be expected. The outliers in the harmonic centrality scores were the families connected to the Medici family. Five of six of the Medici family's neighbors had harmonic centrality scores above the 75th percentile, as a result of the high centrality of the Medici family. This may have been the key to the Medici family's success, not by being extremely central themselves, but by influencing the network in such a way that all of their neighbors were unusually well-connected.

6.a.



Vertex	Degree	Betweenness
A	2	0.005414411
B	2	0.005414411
C	3	0.012078301
D	2	0.012911287
E	2	0.012078301
F	2	0.009579342
G	1	0.005414411

Vertex C has the highest degree, at 3, while vertex D has the highest betweenness, at 0.012911287.

```
# Donovan Guelde
# CSCI 5352, Fall '16
# Problem 5
# References: networkx documentation, numpy docs

import networkx
import matplotlib.pyplot as plt
import sys
import numpy as np

```

```

class Node:
        def __init__(self,number,name):
                self.name = name
                self.number = number
                self.neighbors = []
        def assignNeighbors(self,neighbors):
                for item in neighbors:
                        self.neighbors.append(item)

class Network:
        def __init__(self,n):
                self.nodes = []
                self.n = n

if __name__ == "__main__":
        network = Network(16)
        names = ["Acciaiuoli","Albizzi","Barbadori","Bischeri","Castellani","Ginori","Guadagni","Lamberteschi","Medici",
                                "Pazzi","Peruzzi","Pucci","Ridolfi","Salviati","Strozzi","Tornabuoni"]
        neighbors = {0:[8],1:[5,6,8],2:[4,8],3:[6,10,14],4:[2,10,14],5:[1],6:[1,3,7,15],7:[6],8:[0,1,2,12,13,15],
                                9:[13],10:[3,4,14],11:[],12:[8,14,15],13:[8,9],14:[3,4,10,12],15:[6,8,12]}
        graph = networkx.from_dict_of_lists(neighbors)

        shortestPaths = [] #an array to hold ALL shortest paths, to avoid the networkx habit of using only 1 shortest path,
                                #even if more exist
        for index in range(0,16):
                for index2 in range(0,16):

                        try:
                                shortestPaths.append([p for p in networkx.all_shortest_paths(graph,index,index2)])
                                                                #vertex 11 will cause error
                        except(networkx.exception.NetworkXNoPath):
                                print"" #do nothing for vertex 11, it has no shortest paths except self-loop



        print "degree centrality"
        degreeCentrality = []
        centrality = networkx.degree_centrality(graph)
        for index in range(0,16):
                indexCentrality = int(centrality[index]*15)
                print str(index)+":",indexCentrality
                degreeCentrality.append(indexCentrality)
        print degreeCentrality

        print "harmonic centrality"
        for index in range(0,16):
                if(index!=11):
                        sum=0
                        for index2 in range(0,16):
                                if(index!=index2 and index !=11 and index2 != 11): #again, don't try for vertex 11
                                        sum+=networkx.shortest_path_length(graph,index,index2)
                        print (1/float(sum))/15
                else:
                        print "0"

        print "eigenvector centrality"
        eigenvectorCentrality = networkx.eigenvector_centrality(graph)
```

```
        for index in range(0,16):
                print eigenvectorCentrality[index]

#betweenness, didn't use networkx command to allow for multiple shortest paths
        print "betweenness centrality"

        for index in range(0,16):
                counter = 0
                counter2=0
                for item in shortestPaths:
                        for item2 in item:
                                counter2 += 1
                                if (index in item2):

                                        counter+=1

                print (float(counter)/float(counter2))/pow(16,2)


        print "configuration model:"


        configurationResults = np.zeros(( 16, 100000 ))

        for repetition in range (0,100000):
                tempGraph = networkx.configuration_model(degreeCentrality)
                #perform violence
                tempGraph = networkx.Graph(tempGraph) #collapse multi-edges
                tempGraph.remove_edges_from(tempGraph.selfloop_edges()) #eliminate self-loops
                for index in range(0,16):
                        sum=0
                        for index2 in range(0,16):
                                if(index!=index2):
                                        try:
                                                sum+=networkx.shortest_path_length(tempGraph,index,index2)

                                        except (networkx.exception.NetworkXNoPath):
                                                sum+=0

                        try:
                                configurationResults[index][repetition]=(1/float(sum))/15

                        except (ZeroDivisionError):
                                configurationResults[index][repetition]=0


        percentilesArray = np.zeros((16,3))
        for index in range (0,16):
                percentilesArray[index][0] = np.percentile(configurationResults[index],25)
                percentilesArray[index][1] = np.percentile(configurationResults[index],50)
                percentilesArray[index][2] = np.percentile(configurationResults[index],75)
        print "25"
        for index in range (0,16):
                print percentilesArray[index][0]
        print "50"
        for index in range (0,16):
                print percentilesArray[index][1]
        print "75"
```

```
        for index in range (0,16):
                print percentilesArray[index][2]
```

```
# Donovan Guelde
# CSCI 5352, Fall '16
# Problem 6
# References: networkx documentation, numpy docs


import networkx
import matplotlib.pyplot as plt
import sys
import numpy as np


class Node:
        def __init__(self,number,name):
                self.name = name
                self.number = number
                self.neighbors = []
        def assignNeighbors(self,neighbors):
                for item in neighbors:
                        self.neighbors.append(item)

class Network:
        def __init__(self,n):
                self.nodes = []
                self.n = n

if __name__ == "__main__":
        network = Network(16)
        names = ["A","B","C","D","E","F","G"]
        neighbors = {0:[1,2],1:[0,2],2:[0,1,3],3:[2,4],4:[3,5],5:[4,6],6:[5]}
        graph = networkx.from_dict_of_lists(neighbors)
        shortestPaths = []
        for index in range(0,7):
                for index2 in range(0,7):
                        #if (index != index2 and index != 11 and index2 != 11):
                        shortestPaths.append([p for p in networkx.all_shortest_paths(graph,index,index2)])

        print "degree centrality"
        degreeCentrality = []
        centrality = networkx.degree_centrality(graph)
        for index in range(0,7):
                indexCentrality = int(centrality[index]*6)
                print str(index)+":",indexCentrality
                degreeCentrality.append(indexCentrality)
        print degreeCentrality
        print "betweenness centrality"

        for index in range(0,7):
                counter = 0
                counter2=0
                for item in shortestPaths:
                        for item2 in item:
                                counter2 += 1
                                if (index in item2):

```

```
                    counter+=1

    print (float(counter)/float(counter2))/pow(7,2)
```