SYSM 6302 - Lab 3

Jonas Wagner, jrw200000

In [1]:

```
import networkx as nx
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import Image
```

Snub Code

This function prints the top five (or `num`) nodes according to the centrality vector `v` , where `v` takes the form: `v[nidx]` is the centrality of the node that is the `nidx` -th element of `G.nodes()`

In [2]:

```
def print_top_5(G,v, num=5):
    thenodes = list(G.nodes())
    idx_list = [(i,v[i]) for i in range(len(v))]
    idx_list = sorted(idx_list, key = lambda x: x[1], reverse=True)

    for i in range(min(num,len(idx_list))):
        nidx, score = idx_list[i]
        print('  %i. %s (%1.4f)' % (i+1,thenodes[nidx],score))
        #print '  %i. %s' % (i+1,G.node_object(idx))
    return idx_list
```

This function returns the index of the maximum of the array. If two or more indices have the same max value, the first index is returned.

In [3]:

```
def index_of_max(v):
    return np.where(v == max(v))[0]
```

This function accepts a dictionary of nodes with centrality values and returns a centrality vector

In [4]:

```
def centrality_vector(G,d):
    thenodes = list(G.nodes())
    v = np.zeros((G.number_of_nodes(),))
    for i,u in enumerate(thenodes):
        v[i] = d[u]
    return v
```

This function provides the index of a node based on its order in `G.nodes()`

In [5]:

```
def node_index(G,n):
    thenodes = list(G.nodes())
    return thenodes.index(n)
```

Now we read in the edgelist file that contains the coappearance network we will analyze. We will look at two different networks, corresponding to only the *Lord of the Rings* series and the *Lord of the*

Rings series plus the prequel, The *Hobbit*. The `unweighted` boolean, if set to `True` will set all the edge weights to one. Recall that setting all weights to 1 is different (in NetworkX) from having no weights assigned, which could be accomplished instead by: `G = nx.read_edgelist('LoTR_characters.edgelist',data=False)` .

In [6]:
```
unweighted = True
G = nx.read_weighted_edgelist('LotR_characters.edgelist') # just Lord of the Rings
if unweighted:
    for u,v in G.edges():
        G[u][v]['weight'] = 1
A = nx.adjacency_matrix(G).todense().T
N = G.number_of_nodes()
```

Section 7.1: Degree Centrality

In [7]:
```
# Degree Centrality Calculation
v = np.zeros(G.number_of_nodes())
for i,node in enumerate(G.nodes()):
    for edge in list(G[node]):
        v[i] += G[node][edge]['weight']
print('v = ',v)
print('\n')
print('Top 5 Nodes (Degree Centrality)')
print_top_5(G,v)
v_degree_unweighted = v
```

```
v =  [ 78. 140.  88.  80.  47. 124. 144.  65. 102.  63. 125.  90.  71.  71.
  58.  38.  81.  27.  99. 153.  69.  67.  77. 111.  27. 108.  94.  58.
  40.  44.  86.  48.  40.  40.  40.  40.  40.  40.  74. 113.  40.  40.
  91. 137. 110. 104.  40.  53.  50.  51. 105.  66.  40.  31.  64.  45.
  11.  26.  24.  24.  26.  17.  39.  14.  32.  55.  16.  16.  84.   7.
   7.   7.  17.  17.  57.  17.  76.  19.  64.   9.   7.  23.  21.  64.
  79.  52.  68.  61.  21.  24.  62.  24.  24.  24.  24.  24.  62.  54.
  58.  58.  96.  49.  92.  53.  43.  43.  47.  43.  49.  49.  49.  45.
  20.  31.  11.  25.  13.  16.  16.  21.  18.  11.  23.  18.  28.  24.
  22.  22.  13.  38.  20.  18.  24.  12.  24.  20.  20.   6.  23.  11.
  18.  18.  28.  22.  22.  28.  37.  22.  22.  20.  30.  31.  22.   8.
  26.   8.  21.  21.  21.  17.  12.  15.  15.]


Top 5 Nodes (Degree Centrality)
  1. gandalf (153.0000)
  2. frodo (144.0000)
  3. aragorn (140.0000)
  4. pippin (137.0000)
  5. elrond (125.0000)
```

Section 7.2: Eigenvector Centrality

In [8]:
```
print('Eigenvector Centrality (by NetworkX):')
v_nx = centrality_vector(G,nx.eigenvector_centrality(G))
eig_un = print_top_5(G,v_nx)
v_eig_unweighted = v_nx
```

```
Eigenvector Centrality (by NetworkX):
  1. gandalf (0.1682)
  2. aragorn (0.1641)
  3. frodo (0.1618)
  4. elrond (0.1541)
  5. pippin (0.1533)
```

In [9]:

```
print('Eigenvector Centrality (by linear algebra):')
k, V = la.eig(A)
k1_idx = index_of_max(k) # find the index of the largest eigenvalue
v = np.abs(V[:,k1_idx])
deg_un = print_top_5(G,v)
```

```
Eigenvector Centrality (by linear algebra):
  1. gandalf (0.1682)
  2. aragorn (0.1641)
  3. frodo (0.1618)
  4. elrond (0.1541)
  5. pippin (0.1533)
```

Calc Confirmation

In [10]:

```
noi = 'arwen'
noi_idx = node_index(G,noi)

# print('\n Neighbors:')
cent_sum = 0
for node in G.neighbors(noi):
    idx = node_index(G,node)
#     print(node, '%f' % v[idx])
    cent_sum += v[idx] / np.real(k[k1_idx])

print('Confirming that eigenvector centrality is a steady-state of sorts for node %s:'
print('Eig %f' % v[noi_idx])
print('Sum %f' % cent_sum)
# compare the eigenvector centrality of arwen to the sum of the centralities of its nei
```

```
Confirming that eigenvector centrality is a steady-state of sorts for node arwen:
Eig 0.112654
Sum 0.112654
```

Convergence of Eigenvector Centrality

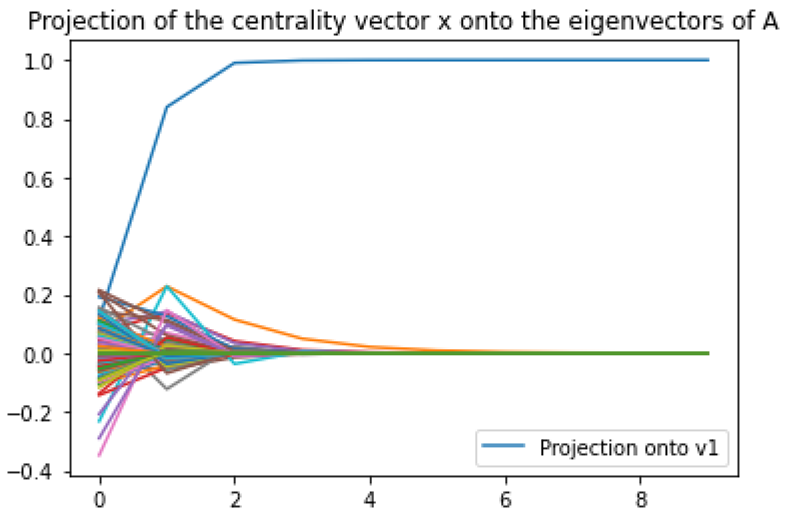
In [11]:

```
print('Showing the convergece of eigenvector centrality...')
num_steps = 10
x = np.zeros((N,1)) # initial centrality vector
x[76] = 1
cs = np.zeros((N,num_steps))
v = V
for i in range(num_steps):
    x = x/la.norm(x) # at each step we need to normalize the centrality vector
    for j in range(G.number_of_nodes()):
        cs[j,i] = np.real(np.dot( x.T , v[:,j] ))[0] # project x onto each of the eigen
    x = np.dot(A,x) # "pass" the centrality one step forward

plt.figure() # this creates a figure to plot in
for i in range(G.number_of_nodes()): # for each eigenvector plot the projection of x on
    if i == k1_idx:
        plt.plot(range(num_steps),cs[i,:],label='Projection onto v1') # only label the
    else:
        plt.plot(range(num_steps),cs[i,:])
#plt.ylim([-0.2,1.1]) # this sets the limits for the y axis
```

```
plt.legend(loc='best') # this attaches a legend
plt.title('Projection of the centrality vector x onto the eigenvectors of A') # this ad
plt.show() # this makes the figure appear
```

```
Showing the convergece of eigenvector centrality...
```



In this plot the dot product between the propogated state x (guessed centrality) and the calculated (and correctly known) eigenvalue centrality. As each iteration pases the guessed centrality aligns with v1 (aproches abs(x) * abs(v1)) and becomes orthogonal with all other potential eigenvectors (aproches zero).

The iterative process itself is propogating the guessed centrality through the network (with A) and ultimently is approaching the steady-state centrality vector.

Section 7.3: Katz Centrality

In [12]:
```
def katz(A, alpha = 0.85, beta = 1):
    n = np.size(A,1)
    # Asumming beta are scalers (same for all nodes... at least for defualt)
    old = np.array(np.linalg.inv(np.eye(np.size(A,1)) - np.dot(alpha,A)).dot(beta * np.
    new = beta * np.linalg.inv(np.eye(n) - alpha * A) @ (np.ones((n,1)))
    v_nx = centrality_vector(G,nx.katz_centrality_numpy(G,alpha,beta))
    return v_nx
```

In [13]:
```
Alpha = [0.1,0.25,0.5,0.75,0.9]
for alpha in Alpha:
    v_katz = katz(A,alpha)
    print('Katz Centrality (alpha = %f):' % alpha)
    print_top_5(G,v_katz)
    print('\n')
v_katz_unweighted = katz(A)
# katz_un = print_top_5(G,v_katz_unweighted)
```

```
Katz Centrality (alpha = 0.100000):
  1. galadriel (0.1523)
  2. gimli (0.1385)
  3. arathorn (0.1211)
  4. legolas (0.1113)
  5. halbarad (0.1036)
```

```
Katz Centrality (alpha = 0.250000):
  1. bilbo (0.2356)
  2. dunhere (0.2067)
  3. frodo (0.2025)
  4. fundin (0.1754)
  5. gandalf (0.1736)


Katz Centrality (alpha = 0.500000):
  1. hammerhand (0.1958)
  2. angbor (0.1950)
  3. aragorn (0.1594)
  4. theoden (0.1431)
  5. varda (0.1329)


Katz Centrality (alpha = 0.750000):
  1. eothain (0.3417)
  2. theoden (0.2415)
  3. anborn (0.1854)
  4. rumil (0.1808)
  5. celeborn (0.1694)


Katz Centrality (alpha = 0.900000):
  1. frar (0.3656)
  2. smeagol (0.1873)
  3. eowyn (0.1734)
  4. arvedui (0.1722)
  5. oin (0.1653)
```

PageRank

In [14]:

```
def pageRank(A, alpha = 0.85, beta = 1):
    # Asumming beta are scalers (same for all nodes... at least for defualt)
    D = np.diag(np.array(np.sum(A,0)).flatten())
    return np.array(np.dot(np.linalg.inv(np.eye(np.size(A,1))
                                         - np.dot(np.dot(alpha,A),
                                                  np.linalg.inv(D))),
                           beta * np.ones(np.size(A,1)))).flatten()
```

In [15]:

```
alpha = 0.85
v_PR = pageRank(A,alpha)
print('Page Rank (alpha = %f):' % alpha)
pr_un = print_top_5(G,v_PR)
v_PR_unweighted = v_PR
```

```
Page Rank (alpha = 0.850000):
  1. gandalf (22.1478)
  2. frodo (20.9287)
  3. pippin (19.7588)
  4. aragorn (19.4926)
  5. bilbo (17.7511)
```

PageRank proof

A is given as the adjacency matrix of a directed network, Let

$$v_1 = [k_1, k_2, \dots, k_n]^T, \quad k_i = k_i^{in}$$

be the vector of the degrees of the nodes.\ Similarily, let

$$D = \backslash\text{diag}[k_1, k_2, \dots, k_n]$$

, be representative of the degrees along each node (but also ensure that any zero k_i terms are set to 1).\ From degree centrality we know that AD^{-1} itself charectorizes the degree centrality and thus we know that by

$$AD^{-1}x = \lambda_i x$$

the primary (1st SVD) eigenvector is v.\ Looking at the definition of Page Rank,

$$x = (I - \alpha A D^{-1})^{-1},$$

it is know posible to say that when $\alpha = 1$ the quantity $I - \alpha A D^{-1}$ becomes singular, causing the inverse to be undefined. Therefore we can conclude that $\alpha \in [0, 1)$.

Section 7.5: Hubs & Authorities

The authority centrality of a node is proportional to the sum of the hub centralities one nodes that point to it:

$$x_i = \alpha \sum_j A_{ij} y_j$$

Similarily the hub centrality of a node is a sum of the authority centralities of nodes it points to:

$$y_j = \beta \sum_i A_{ij} x_i$$

This can be written in matrix form as:

$$x = \alpha A y$$

and

$$y = \beta A x$$

once they are combined and restructured to form the following:

$$AA^T x = \lambda x$$

and

$$A^T A y = \lambda y$$

with $\lambda = (\alpha\beta)^{-1}$. Since both share the same eigenvalues A and A^T are commutable, so:

$$AA(A^T x) = \lambda (A^T x)$$

which with another substitution becomes:

$$y = A^T x$$

Section 7.7: Betweenness Centrality

In [16]:
```
G_unweighted = G
```

In [17]:
```
print('Betweeness Centrality (by NetworkX):')
v = centrality_vector(G,nx.betweenness_centrality(G))
btw_un = print_top_5(G,v)
v_between_unweighted = v
```

```
Betweeness Centrality (by NetworkX):
  1. gandalf (0.0797)
  2. frodo (0.0675)
  3. pippin (0.0613)
  4. aragorn (0.0486)
  5. bilbo (0.0393)
```

Weighted Analysis

In [18]:
```
G = nx.read_weighted_edgelist('LotR_characters.edgelist') # just Lord of the Rings
A = nx.adjacency_matrix(G).todense().T
N = G.number_of_nodes()
```

Degree Centrality Comparrision

In [19]:
```
print('Top 5 Nodes (Degree Centrality) - Unweighted')
print_top_5(G_unweighted,v_degree_unweighted);
```

```
Top 5 Nodes (Degree Centrality) - Unweighted
  1. gandalf (153.0000)
  2. frodo (144.0000)
  3. aragorn (140.0000)
  4. pippin (137.0000)
  5. elrond (125.0000)
```

In [20]:
```
# Degree Centrality Calculation
v = np.zeros(G.number_of_nodes())
for i,node in enumerate(G.nodes()):
    for edge in list(G[node]):
        v[i] += G[node][edge]['weight']
print('Top 5 Nodes (Degree Centrality) - Weighted')
deg_w = print_top_5(G,v)
v_degree_weighted = v
```

```
Top 5 Nodes (Degree Centrality) - Weighted
  1. gandalf (762.0000)
  2. frodo (661.0000)
  3. aragorn (632.0000)
  4. pippin (606.0000)
  5. elrond (484.0000)
```

Degree centrality remained pretty much the same as order is concerned. Obviously the values increased but that is to be expected.

Eigenvector Centrality

In [21]:
```
print('Eigenvector Centrality - Unweighted')
print_top_5(G_unweighted,v_eig_unweighted);
```

```
Eigenvector Centrality - Unweighted
  1. gandalf (0.1682)
  2. aragorn (0.1641)
  3. frodo (0.1618)
  4. elrond (0.1541)
  5. pippin (0.1533)
```

In [22]:
```
print('Eignevector Centrality - Weighted')
v = centrality_vector(G,nx.eigenvector_centrality(G))
eig_w = print_top_5(G,v)
v_eig_weighted = v
```

```
Eignevector Centrality - Weighted
  1. gandalf (0.1682)
  2. aragorn (0.1641)
  3. frodo (0.1618)
  4. elrond (0.1541)
  5. pippin (0.1533)
```

Eigenvector centrality remained basically the same.

Katz Centrality

In [23]:
```
print('Katz Centrality - Unweighted')
print_top_5(G_unweighted,v_katz_unweighted);
```

```
Katz Centrality - Unweighted
  1. hammerhand (0.2842)
  2. finrod (0.2516)
  3. celeborn (0.2416)
  4. frar (0.1839)
  5. goldberry (0.1711)
```

In [24]:
```
print('Katz Centrality - Weighted')
v = katz(A)
katz_w = print_top_5(G,v)
v_katz_weighted = v
```

```
Katz Centrality - Weighted
  1. hammerhand (0.2842)
  2. finrod (0.2516)
  3. celeborn (0.2416)
  4. frar (0.1839)
  5. goldberry (0.1711)
```

Honestly the Katz centrality was just all over the place for me... didn't work well at all... but looking at the results it appears to greatly change.

Page Rank

In [25]:
```
print('Page Rank Centrality - Unweighted')
print_top_5(G_unweighted,v_PR_unweighted);
```

```
Page Rank Centrality - Unweighted
  1. gandalf (22.1478)
  2. frodo (20.9287)
  3. pippin (19.7588)
  4. aragorn (19.4926)
  5. bilbo (17.7511)
```

In [26]:
```
alpha = 0.85
v_PR = pageRank(A,alpha)
print('Page Rank (alpha = %f):' % alpha)
pr_w = print_top_5(G,v_PR)
v_PR_unweighted = v_PR
```

```
Page Rank (alpha = 0.850000):
  1. gandalf (44.3470)
  2. frodo (39.4107)
  3. aragorn (36.0736)
  4. pippin (35.6541)
  5. elrond (28.1555)
```

Betweeness Centrality

In [27]:
```
print('Betweeness Centrality - Unweighted')
print_top_5(G_unweighted,v_between_unweighted);
```

```
Betweeness Centrality - Unweighted
  1. gandalf (0.0797)
  2. frodo (0.0675)
  3. pippin (0.0613)
  4. aragorn (0.0486)
  5. bilbo (0.0393)
```

In [28]:
```
print('Betweeness Centrality - Weighted')
v = centrality_vector(G,nx.betweenness_centrality(G))
btw_w = print_top_5(G,v)
v_between_weighted = v
```

```
Betweeness Centrality - Weighted
  1. gandalf (0.0797)
  2. frodo (0.0675)
  3. pippin (0.0613)
  4. aragorn (0.0486)
  5. bilbo (0.0393)
```

This appear to change a lot more then most of the others.

A Sequel (Prequel - The Hobbit)

In [29]:
```
G_weighted = G
```

In [30]:
```
G = nx.read_weighted_edgelist('hobbit_LotR_characters.edgelist') # with the Hobbit
A = nx.adjacency_matrix(G).todense().T
N = G.number_of_nodes()
```

```
In [31]:
# Degree Centrality Calculation
v = np.zeros(G.number_of_nodes())
for i,node in enumerate(G.nodes()):
    for edge in list(G[node]):
        v[i] += G[node][edge]['weight']
print('Top 5 Nodes (Degree Centrality) - Hobbit')
deg_h = print_top_5(G,v)
v_deg_hobbit = v
```

```
Top 5 Nodes (Degree Centrality) - Hobbit
  1. gandalf (901.0000)
  2. frodo (661.0000)
  3. aragorn (632.0000)
  4. pippin (606.0000)
  5. bilbo (602.0000)
```

Eigenvector Centrality

```
In [32]:
print('Eignevector Centrality - Hobbit')
v = centrality_vector(G,nx.eigenvector_centrality(G))
eig_h = print_top_5(G,v)
v_eig_hobbit = v
```

```
Eignevector Centrality - Hobbit
  1. gandalf (0.1692)
  2. aragorn (0.1618)
  3. frodo (0.1598)
  4. elrond (0.1534)
  5. bilbo (0.1531)
```

Katz Centrality

```
In [33]:
print('Katz Centrality - Hobbit')
v = katz(A)
katz_h = print_top_5(G,v)
v_katz_hobbit = v
```

```
Katz Centrality - Hobbit
  1. theoden (0.2725)
  2. hammerhand (0.2584)
  3. eothain (0.2499)
  4. brand (0.2494)
  5. hamfast (0.1465)
```

Honestly the Katz centrality was just all over the place for me... didn't work well at all... but looking at the results it appears to greatly change.

Page Rank

```
In [34]:
alpha = 0.85
v_PR = pageRank(A,alpha)
print('Page Rank (alpha = %f):' % alpha)
pr_h = print_top_5(G,v_PR)
v_PR_unweighted = v_PR
```

```
Page Rank (alpha = 0.850000):
  1. gandalf (48.7526)
  2. frodo (36.8473)
  3. aragorn (33.6945)
  4. pippin (33.3878)
  5. bilbo (33.2073)
```

Betweeness Centrality

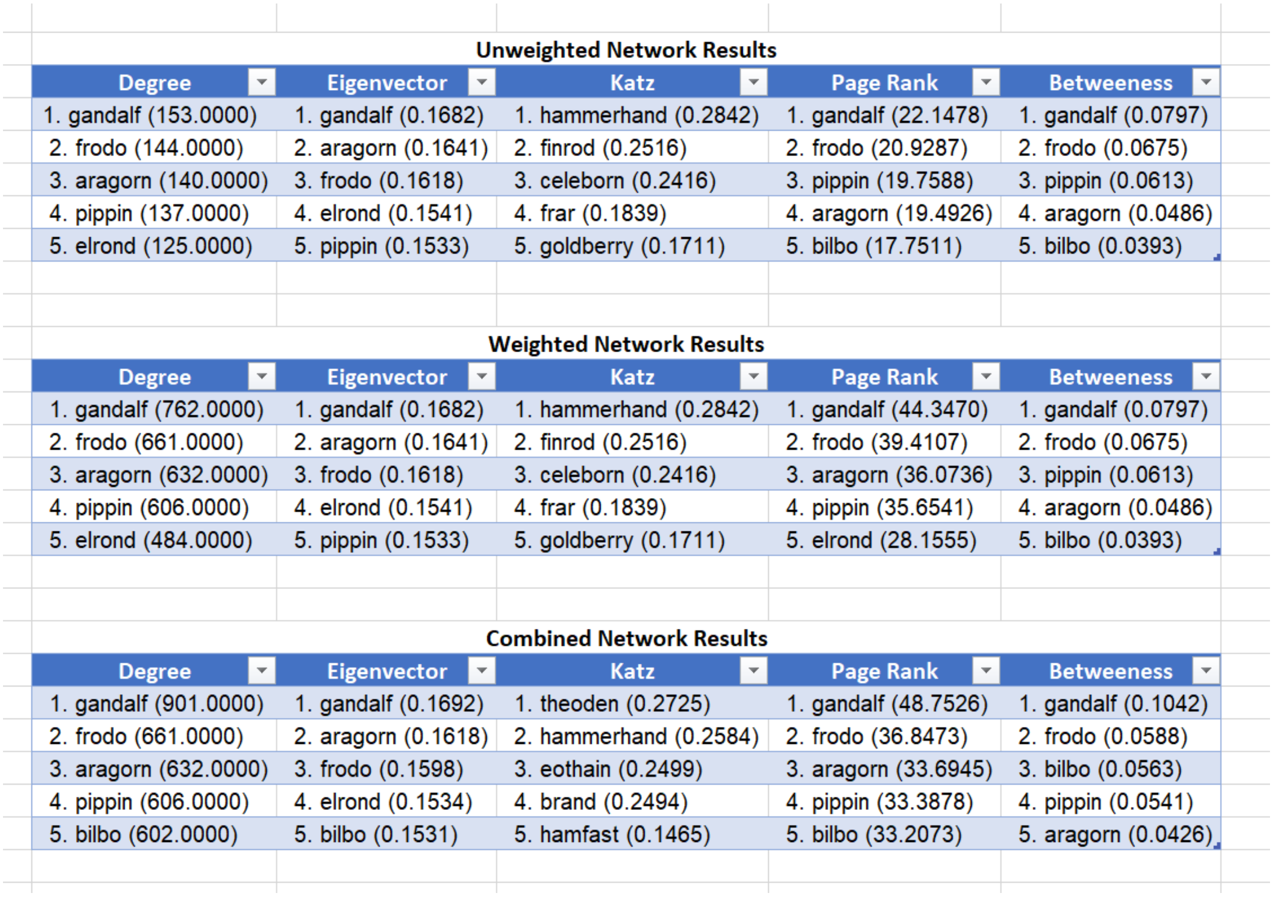
In [35]:
```python
print('Betweeness Centrality - Hobbit')
v = centrality_vector(G,nx.betweenness_centrality(G))
btw_h = print_top_5(G,v)
v_between_hobbit = v
```

```
Betweeness Centrality - Hobbit
  1. gandalf (0.1042)
  2. frodo (0.0588)
  3. bilbo (0.0563)
  4. pippin (0.0541)
  5. aragorn (0.0426)
```

Results

In [36]:
```python
Image(filename='fig/finaldata.png')
```

Out[36]:



Unweighted Network Results

Degree	Eigenvector	Katz	Page Rank	Betweeness
1. gandalf (153.0000)	1. gandalf (0.1682)	1. hammerhand (0.2842)	1. gandalf (22.1478)	1. gandalf (0.0797)
2. frodo (144.0000)	2. aragorn (0.1641)	2. finrod (0.2516)	2. frodo (20.9287)	2. frodo (0.0675)
3. aragorn (140.0000)	3. frodo (0.1618)	3. celeborn (0.2416)	3. pippin (19.7588)	3. pippin (0.0613)
4. pippin (137.0000)	4. elrond (0.1541)	4. frar (0.1839)	4. aragorn (19.4926)	4. aragorn (0.0486)
5. elrond (125.0000)	5. pippin (0.1533)	5. goldberry (0.1711)	5. bilbo (17.7511)	5. bilbo (0.0393)

Weighted Network Results

Degree	Eigenvector	Katz	Page Rank	Betweeness
1. gandalf (762.0000)	1. gandalf (0.1682)	1. hammerhand (0.2842)	1. gandalf (44.3470)	1. gandalf (0.0797)
2. frodo (661.0000)	2. aragorn (0.1641)	2. finrod (0.2516)	2. frodo (39.4107)	2. frodo (0.0675)
3. aragorn (632.0000)	3. frodo (0.1618)	3. celeborn (0.2416)	3. aragorn (36.0736)	3. pippin (0.0613)
4. pippin (606.0000)	4. elrond (0.1541)	4. frar (0.1839)	4. pippin (35.6541)	4. aragorn (0.0486)
5. elrond (484.0000)	5. pippin (0.1533)	5. goldberry (0.1711)	5. elrond (28.1555)	5. bilbo (0.0393)

Combined Network Results

Degree	Eigenvector	Katz	Page Rank	Betweeness
1. gandalf (901.0000)	1. gandalf (0.1692)	1. theoden (0.2725)	1. gandalf (48.7526)	1. gandalf (0.1042)
2. frodo (661.0000)	2. aragorn (0.1618)	2. hammerhand (0.2584)	2. frodo (36.8473)	2. frodo (0.0588)
3. aragorn (632.0000)	3. frodo (0.1598)	3. eothain (0.2499)	3. aragorn (33.6945)	3. bilbo (0.0563)
4. pippin (606.0000)	4. elrond (0.1534)	4. brand (0.2494)	4. pippin (33.3878)	4. pippin (0.0541)
5. bilbo (602.0000)	5. bilbo (0.1531)	5. hamfast (0.1465)	5. bilbo (33.2073)	5. aragorn (0.0426)