Zachary's karate club

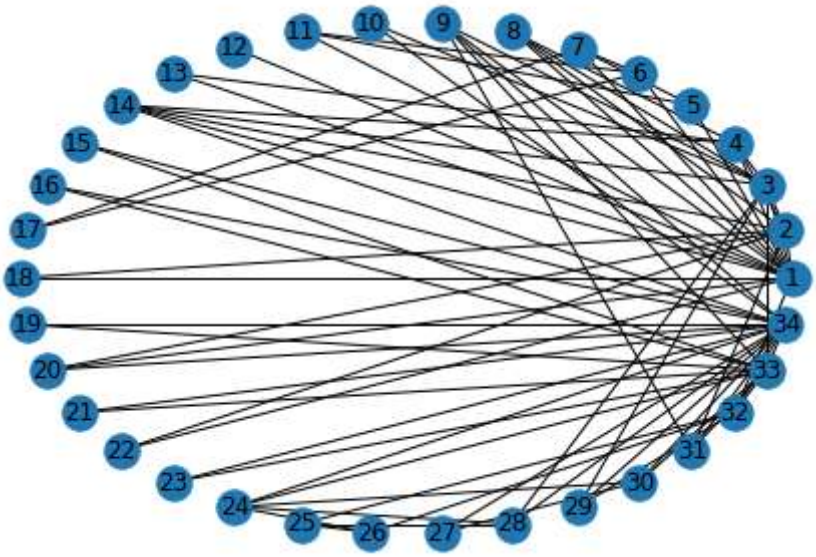
In [793]:

```
import numpy as np
import networkx as nx
G = nx.read_gml('networks/karate/karate.gml', label = 'id')
```

Visualização

In [794]:

```
nx.draw_circular(G, with_labels=True)
plt.show()
```



Caracterização

In [795]:

```
n = G.number_of_nodes()
m = G.number_of_edges()
print('Número de vértices:', n)
print('Número de arestas:', m)
print('Grafo conexo?',nx.is_connected(G))
```

```
Número de vértices: 34
Número de arestas: 78
Grafo conexo? True
```

1) Grau

In [796]:

```
degrees = np.array([val for (node, val) in G.degree()])
```

In [797]:

```
from statistics import median
print('Máximo:', degrees.max())
print('Mínimo:', degrees.min())
print('Média:',degrees.mean())
print('Mediana:', median(degrees))
print('Desvio padrão:', degrees.std())
```

```
Máximo: 17
Mínimo: 1
Média: 4.588235294117647
Mediana: 3.0
Desvio padrão: 3.820360677912828
```

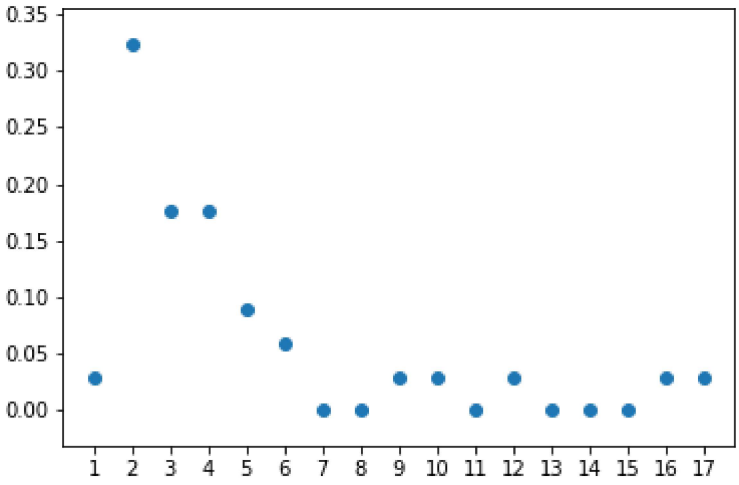
Distribuição empírica

PMF:

In [798]:

```
f = np.array(nx.degree_histogram(G))/n
f = f[1:] # elimina o grau 0

plt.scatter(range(1,len(f)+1),f)
plt.xticks(range(1,len(f)+1))
plt.show()
```



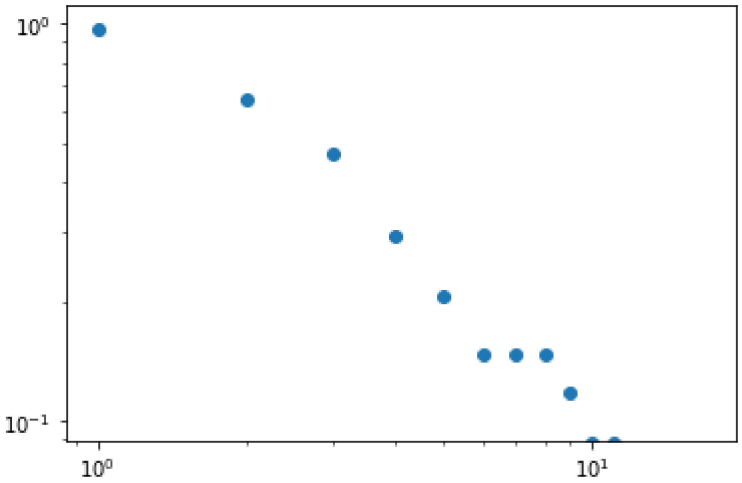
CCDF:

In [799]:

```
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf

# Outro médodo:
#ccdf2 = np.zeros(len(f))
#for k,_ in enumerate(f):
#    ccdf2[k]  = 1-f[0:k+1].sum()

plt.scatter(range(1,len(f)+1),ccdf)
#plt.plot(range(1,len(f)+1),ccdf2)
plt.xscale('log')
plt.yscale('log')
plt.show()
```



2) Distância

Calculando a distância entre cada par de vértices:

In [800]:

```
dist = np.array([])
for v in G.nodes():
    spl = nx.single_source_shortest_path_length(G, v) # distância short path lenght
    spl2 = dict((v2,d) for v2,d in spl.items() if v2 != v) # distância exluindo d(v,v)
    dist = np.append(dist, list(spl2.values()))
```

Estatísticas básicas:

In [801]:

```
print('Máximo:', dist.max())
print('Mínimo:', dist.min())
print('Média:',dist.mean())
print('Mediana:', median(dist))
print('Desvio padrão:', dist.std())
```

```
Máximo: 5.0
Mínimo: 1.0
Média: 2.408199643493761
Mediana: 2.0
Desvio padrão: 0.9303002808496257
```

Verificando o cálculo da distância média através do método disponibilizado pela biblioteca:

In [802]:

```
d_mean = nx.average_shortest_path_length(G)
print('Distância média:',d_mean)
```

```
Distância média: 2.408199643493761
```

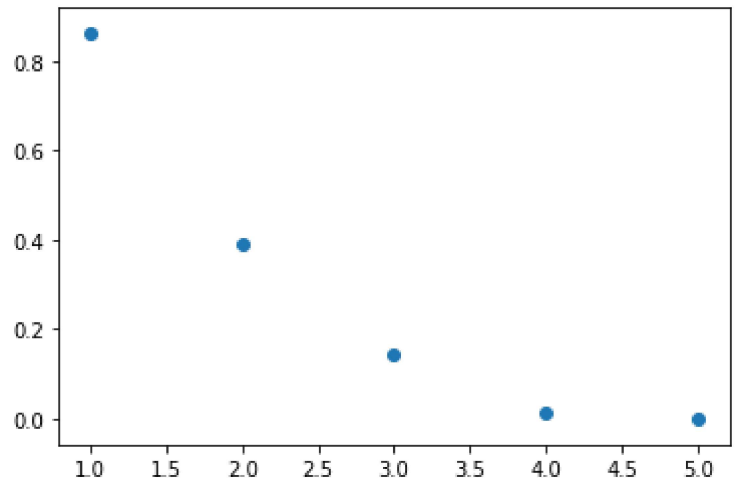
Frequência relativa e CCDF:

In [803]:

```
from scipy.special import comb
Lmax = comb(n,2) # número máximo de arestas

counts, f_d = np.unique(dist, return_counts=True)
cdf = f_d.cumsum()/f_d.sum()
ccdf = 1-cdf

plt.scatter(counts,ccdf)
plt.show()
```



3) Tamanho das componentes conexas

Número de componentes conexas:

In [804]:

```
nx.number_connected_components(G)
```

Out[804]:

```
1
```

Tamanho das componentes:

In [805]:

```
[len(c) for c in sorted(nx.connected_components(G), key=len, reverse=True)]
```

Out[805]:

```
[34]
```

4) Clusterização

4.1) Clusterização Local

In [806]:

```
cluster = np.array(list(nx.clustering(G).values()))
print('Máximo:', cluster.max())
print('Mínimo:', cluster.min())
print('Média:', cluster.mean())
print('Mediana:', median(cluster))
print('Desvio padrão:', cluster.std())
```

```
Máximo: 1.0
Mínimo: 0.0
Média: 0.5706384782076823
Mediana: 0.5
Desvio padrão: 0.342266011779565
```

Verificando a clusterização média:

In [807]:

```
nx.average_clustering(G)
```

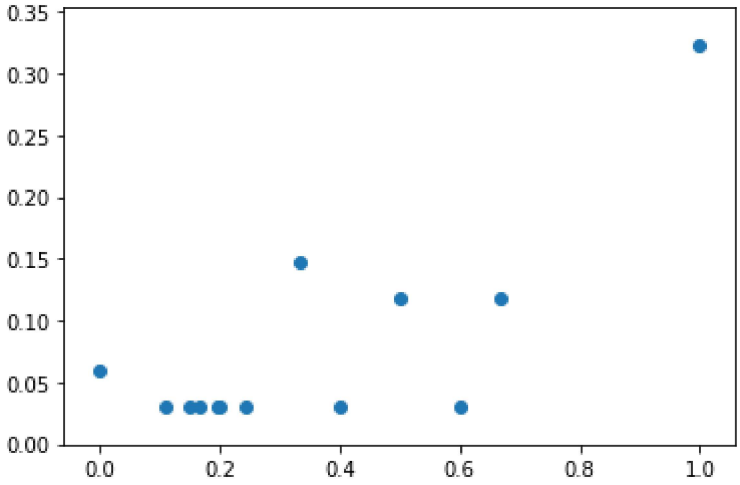
Out[807]:

```
0.5706384782076823
```

PMF:

In [808]:

```
x, f = np.unique(cluster, return_counts=True)
pmf = f/f.sum()
plt.scatter(x,pmf)
plt.show()
```

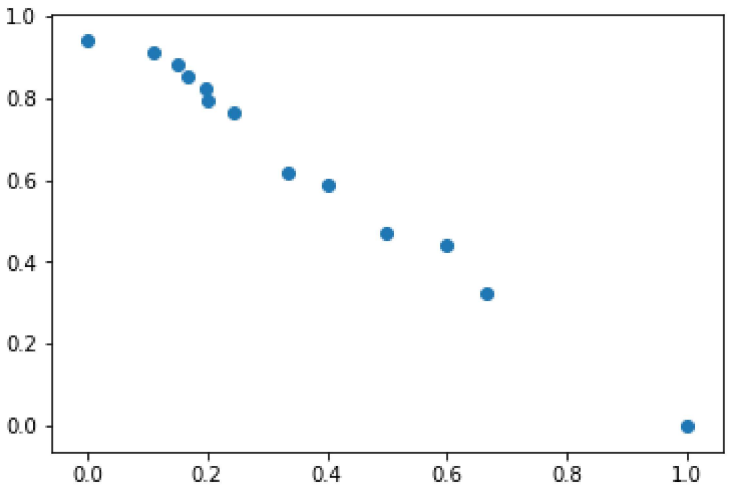


CCDF:

In [809]:

```
x, f = np.unique(cluster, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf

plt.scatter(x,ccdf)
plt.show()
```



4.2) Clusterização global

In [810]:

```
n_triang = np.array(list(nx.triangles(G).values())).sum()/3 # método conta 3 vezes o tr
iângula (1x para cada vértice)
print('Número de triângulos:', n_triang)
```

```
Número de triângulos: 45.0
```

Clusterização global:

In [811]:

```
nx.transitivity(G)
```

Out[811]:

```
0.2556818181818182
```

5) Centralidade

5.1) Centralidade de Grau

In [812]:

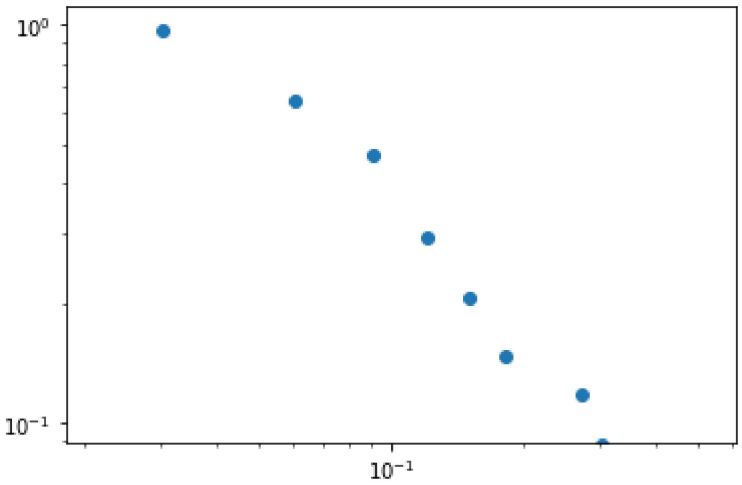
```
cent = np.array(list(nx.degree_centrality(G).values()))
print('Máximo:', cent.max())
print('Mínimo:', cent.min())
print('Média:', cent.mean())
print('Mediana:', median(cent))
print('Desvio padrão:', cent.std())
```

```
Máximo: 0.5151515151515151
Mínimo: 0.030303030303030304
Média: 0.1390374331550802
Mediana: 0.09090909090909091
Desvio padrão: 0.11576850539129781
```

CCDF:

In [813]:

```
x, f = np.unique(cent, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf
plt.scatter(x,ccdf)
plt.xscale('log')
plt.yscale('log')
plt.show()
```



5.2) Betweeness

In [814]:

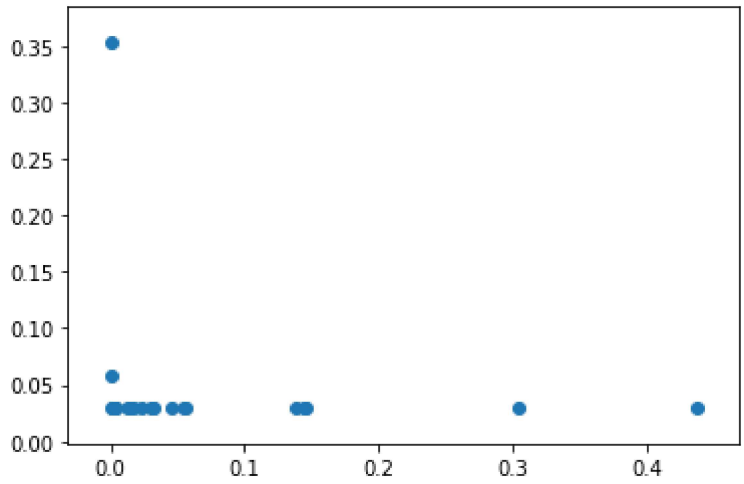
```
btw = np.array(list(nx.betweenness_centrality(G).values()))
print('Máximo:', btw.max())
print('Mínimo:', btw.min())
print('Média:', btw.mean())
print('Mediana:', median(btw))
print('Desvio padrão:', btw.std())
```

```
Máximo: 0.43763528138528146
Mínimo: 0.0
Média: 0.044006238859180036
Mediana: 0.0025658369408369406
Desvio padrão: 0.09254294988248436
```

PMF:

In [815]:

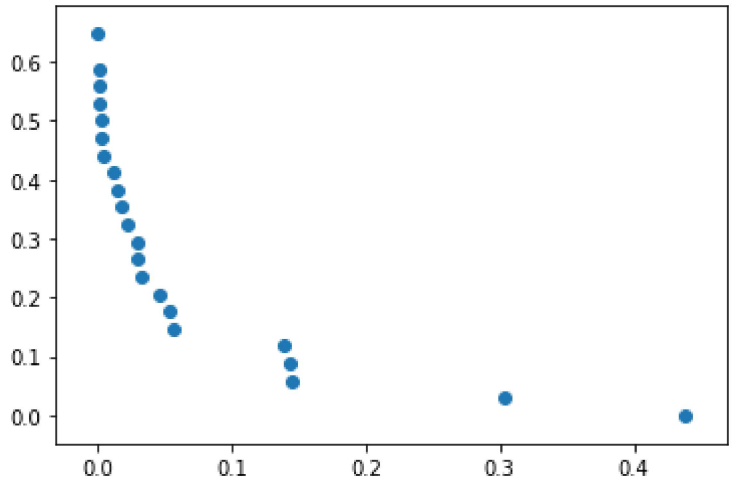
```
x, f = np.unique(btw, return_counts=True)
pmf = f/f.sum()
plt.scatter(x,pmf)
plt.show()
```



CCDF:

In [816]:

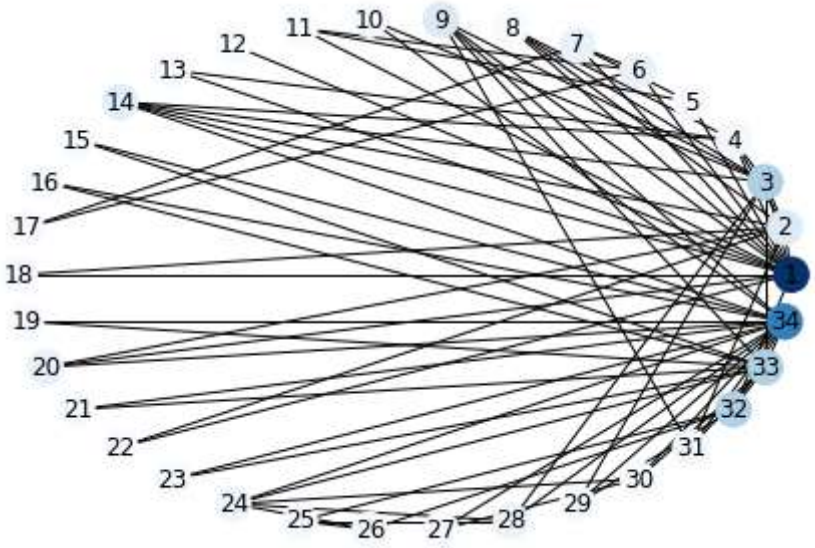
```
x, f = np.unique(btw, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf
plt.scatter(x,ccdf)
plt.show()
```



Visualziação do Betweenes:

In [817]:

```
nx.draw_circular(G, node_color=btw, cmap=plt.cm.Blues, with_labels=True)
```



5.3) Closeness

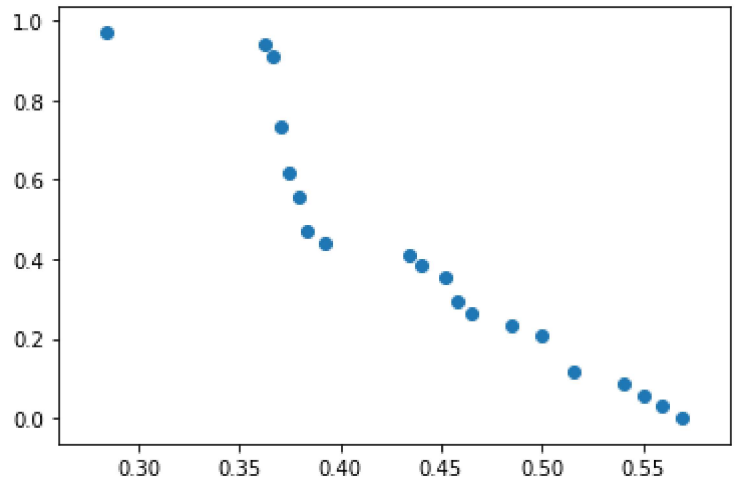
In [818]:

```
close = np.array(list(nx.closeness_centrality(G).values()))
print('Máximo:', close.max())
print('Mínimo:', close.min())
print('Média:', close.mean())
print('Mediana:', median(close))
print('Desvio padrão:', close.std())
```

```
Máximo: 0.5689655172413793
Mínimo: 0.28448275862068967
Média: 0.4264796325735234
Mediana: 0.38372093023255816
Desvio padrão: 0.07102385104221558
```

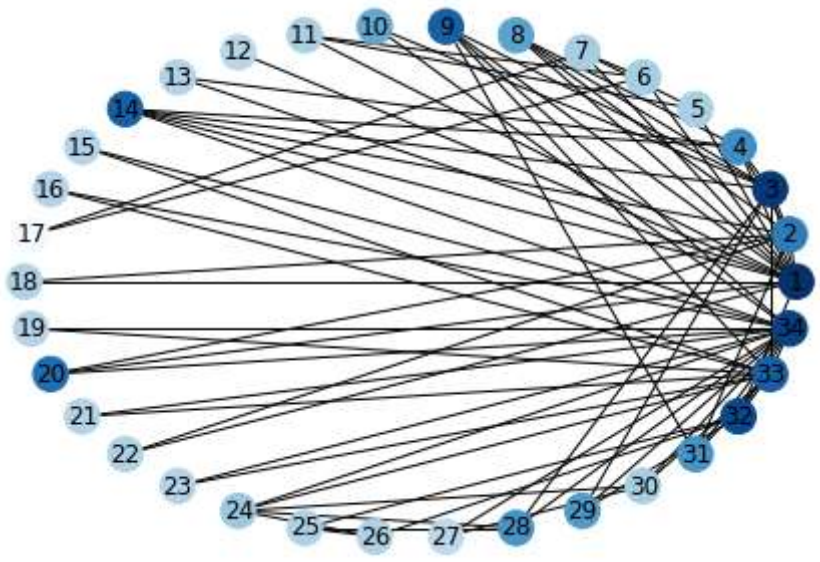
In [819]:

```
x, f = np.unique(close, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf
plt.scatter(x,ccdf)
plt.show()
```



In [820]:

```
nx.draw_circular(G, node_color=close, cmap=plt.cm.Blues, with_labels=True)
```



5.4) Auto-Vetor

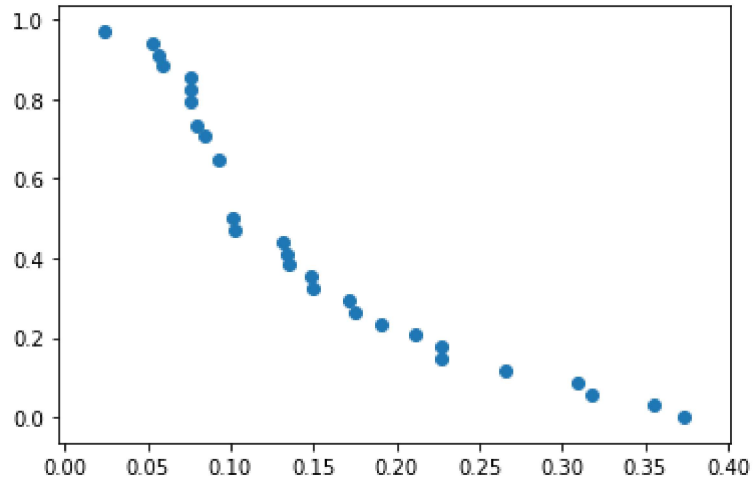
In [821]:

```
ev = np.array(list(nx.eigenvector_centrality(G).values()))
print('Máximo:', ev.max())
print('Mínimo:', ev.min())
print('Média:', ev.mean())
print('Mediana:', median(ev))
print('Desvio padrão:', ev.std())
```

```
Máximo: 0.373371213013235
Mínimo: 0.023634794260596875
Média: 0.14641138111110436
Mediana: 0.10204073438454295
Desvio padrão: 0.089305499198097
```

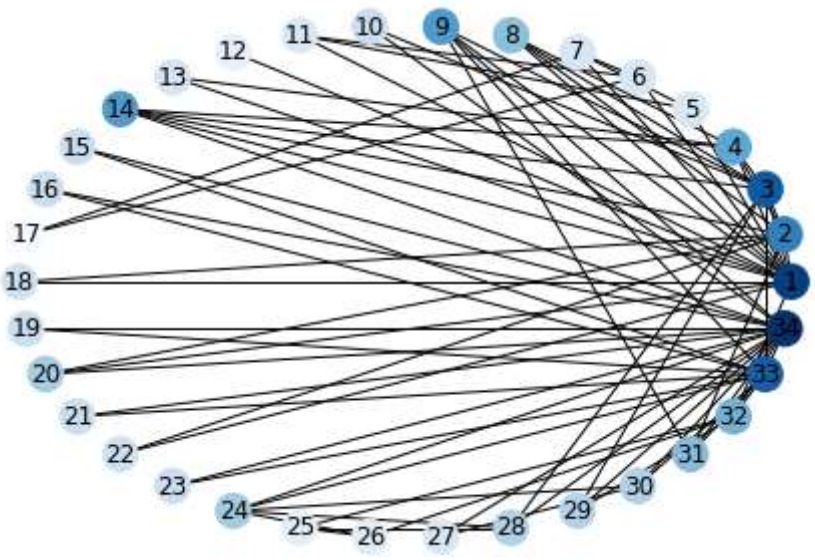
In [822]:

```
x, f = np.unique(ev, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf
plt.scatter(x,ccdf)
plt.show()
```



In [823]:

```
nx.draw_circular(G, node_color=ev, cmap=plt.cm.Blues, with_labels=True)
```



5.6) PageRank

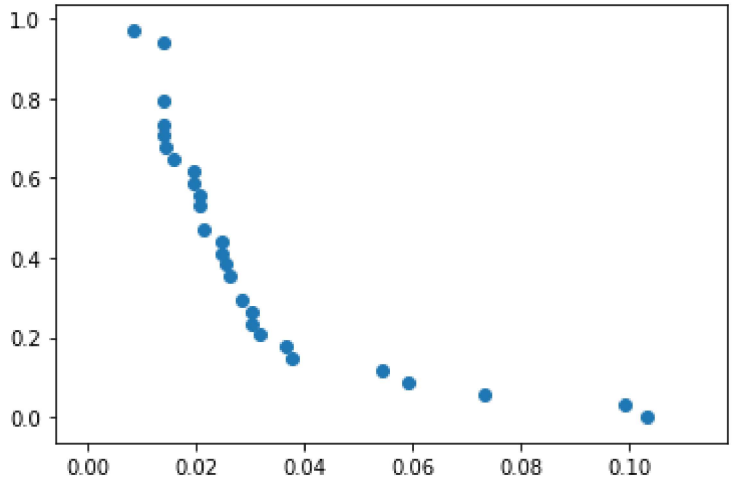
In [824]:

```
pr = np.array(list(nx.pagerank(G, alpha=0.9).values()))
print('Máximo:', pr.max())
print('Mínimo:', pr.min())
print('Média:', pr.mean())
print('Mediana:', median(pr))
print('Desvio padrão:', pr.std())
```

```
Máximo: 0.10345460652842152
Mínimo: 0.008523220243546811
Média: 0.029411764705882353
Mediana: 0.02129205163493875
Desvio padrão: 0.02268437677519535
```

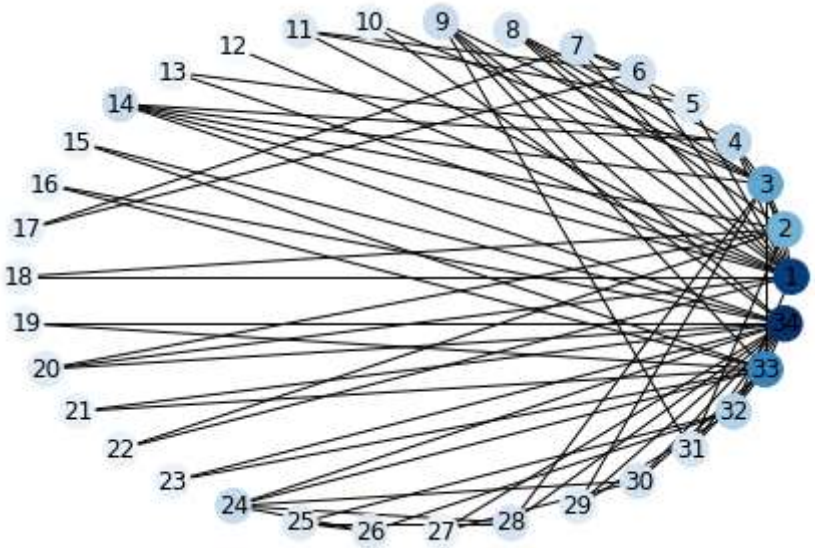
In [825]:

```
x, f = np.unique(pr, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf
plt.scatter(x,ccdf)
#plt.xscale('log')
#plt.yscale('log')
plt.show()
```



In [826]:

```
nx.draw_circular(G, node_color=pr, cmap=plt.cm.Blues, with_labels=True)
```



5) Similaridade

5.1) Jaccard

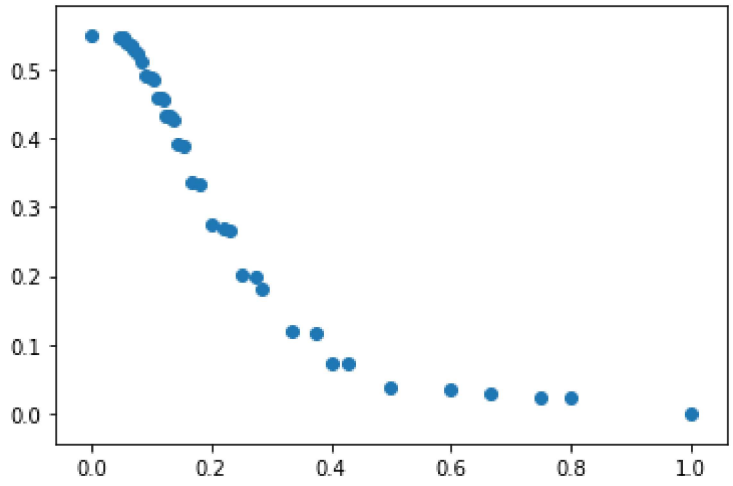
In [827]:

```
jaccard = np.array([p for (u, v, p) in nx.jaccard_coefficient(G)])
print('Máximo:', jaccard.max())
print('Mínimo:', jaccard.min())
print('Média:', jaccard.mean())
print('Mediana:', median(jaccard))
print('Desvio padrão:', jaccard.std())
```

```
Máximo: 1.0
Mínimo: 0.0
Média: 0.15214237132461966
Mediana: 0.09090909090909091
Desvio padrão: 0.20488274065459014
```

In [828]:

```
x, f = np.unique(jaccard, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf
plt.scatter(x,ccdf)
#plt.xscale('log')
#plt.yscale('log')
plt.show()
```

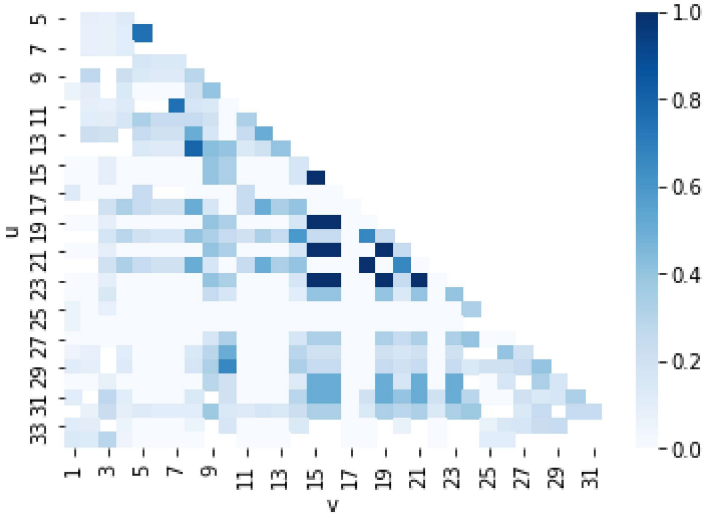


In [829]:

```
mapa = [[u,v, p] for (u, v, p) in nx.jaccard_coefficient(G)]

import pandas as pd
df = pd.DataFrame(mapa)
df.set_index([0,1],inplace=True)
df_m = df.unstack(level=0)

import seaborn as sns
ax = sns.heatmap(df_m, cmap=plt.cm.Blues)
locs, labels = plt.xticks()
plt.xticks(locs,range(1,33,2))
plt.xlabel('v')
plt.ylabel('u')
plt.show()
```



5.2) Adamic/Adar

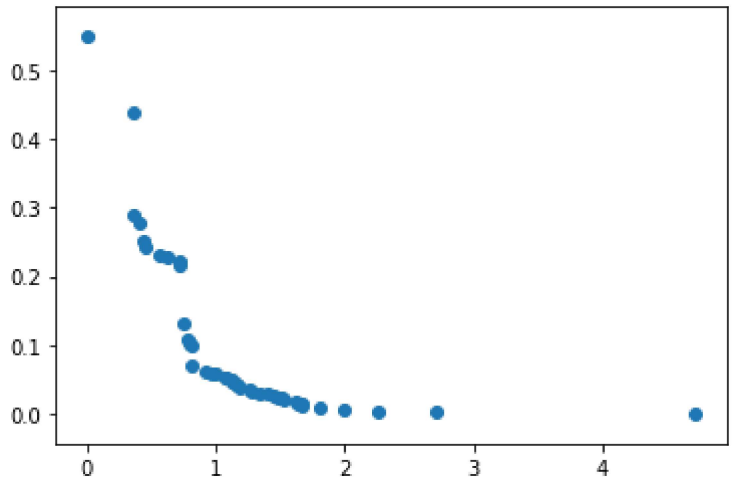
In [830]:

```
adamic = np.array([p for (u, v, p) in nx.adamic_adar_index(G)])
print('Máximo:', adamic.max())
print('Mínimo:', adamic.min())
print('Média:', adamic.mean())
print('Mediana:', median(adamic))
print('Desvio padrão:', adamic.std())
```

```
Máximo: 4.719381261461351
Mínimo: 0.0
Média: 0.34660169559173803
Mediana: 0.35295612386476116
Desvio padrão: 0.45611399967946203
```

In [831]:

```
x, f = np.unique(adamic, return_counts=True)
cdf = f.cumsum()/f.sum()
ccdf = 1-cdf
plt.scatter(x,ccdf)
plt.show()
```



In [832]:

```
#pairs = dict([((u,v), p) for (u, v, p) in nx.adamic_adar_index(G)])
mapa = [[u,v, p] for (u, v, p) in nx.adamic_adar_index(G)]

import pandas as pd
df = pd.DataFrame(mapa)
df.set_index([0,1],inplace=True)
df_m = df.unstack(level=0)

import seaborn as sns
ax = sns.heatmap(df_m, cmap=plt.cm.Blues)
locs, labels = plt.xticks()
plt.xticks(locs,range(1,34,2))
plt.xlabel('v')
plt.ylabel('u')
plt.show()
```