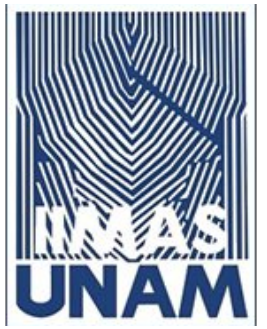


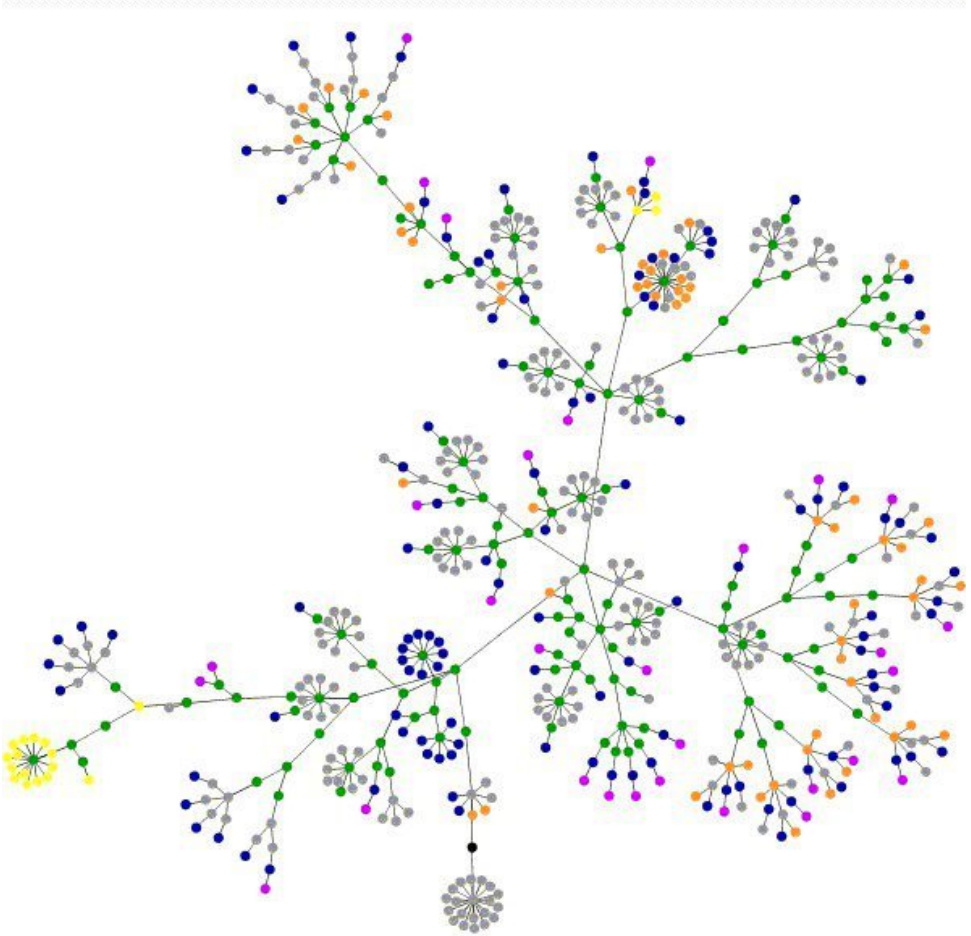
Taller de influencia y centralidad en redes

M.C. Alejandro Esteban Pimentel Alarcón
Posgrado en Ciencia e Ingeniería de la Computación

*Taller basado en:
Curso Applied Machine Learning in Python (Universidad de Michigan)
Graph Analysis and visualization: Discovering Business Opportunity in Linked Data. Richard Brath and David Jonker

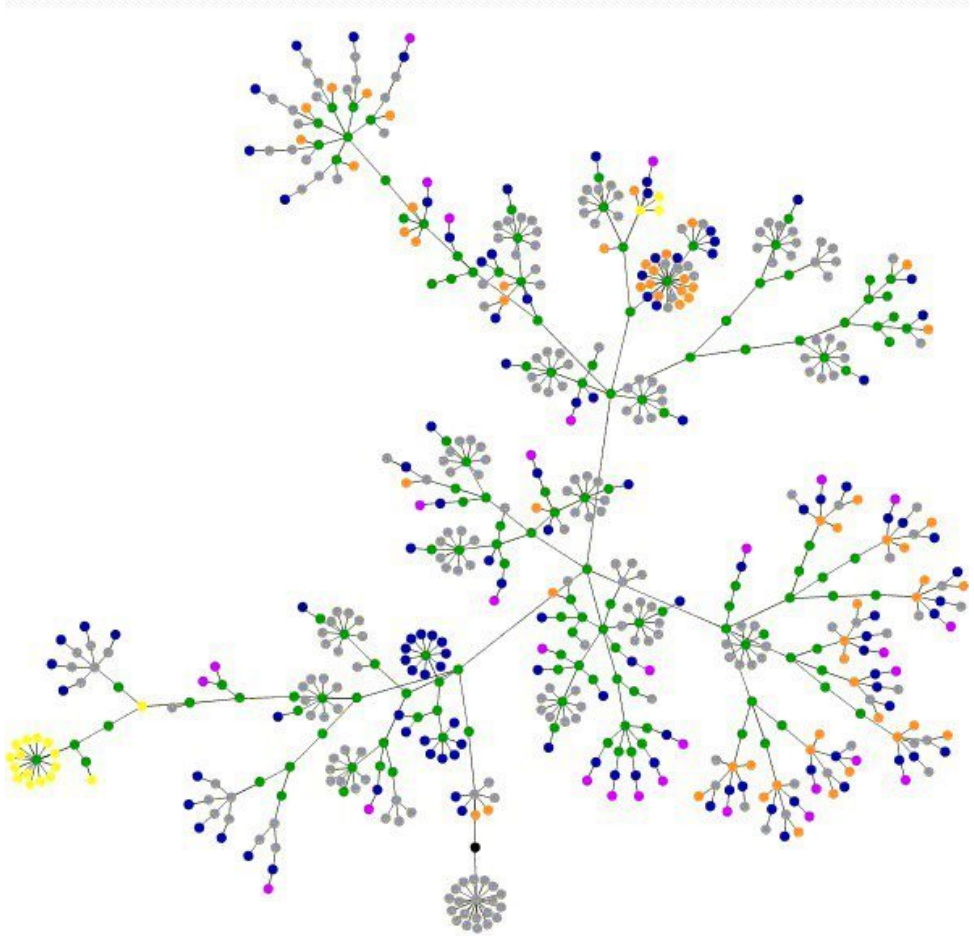
Centralidad de una red

- Identificar los nodos más importantes de un grafo
  - "Influencers"
  - Puntos de falla
  - Páginas relevantes



Medidas de centralidad

- Centralidad de grado
- Centralidad de cercanía
- Centralidad intermedia
- Page Rank
- Hubs y Autoridades



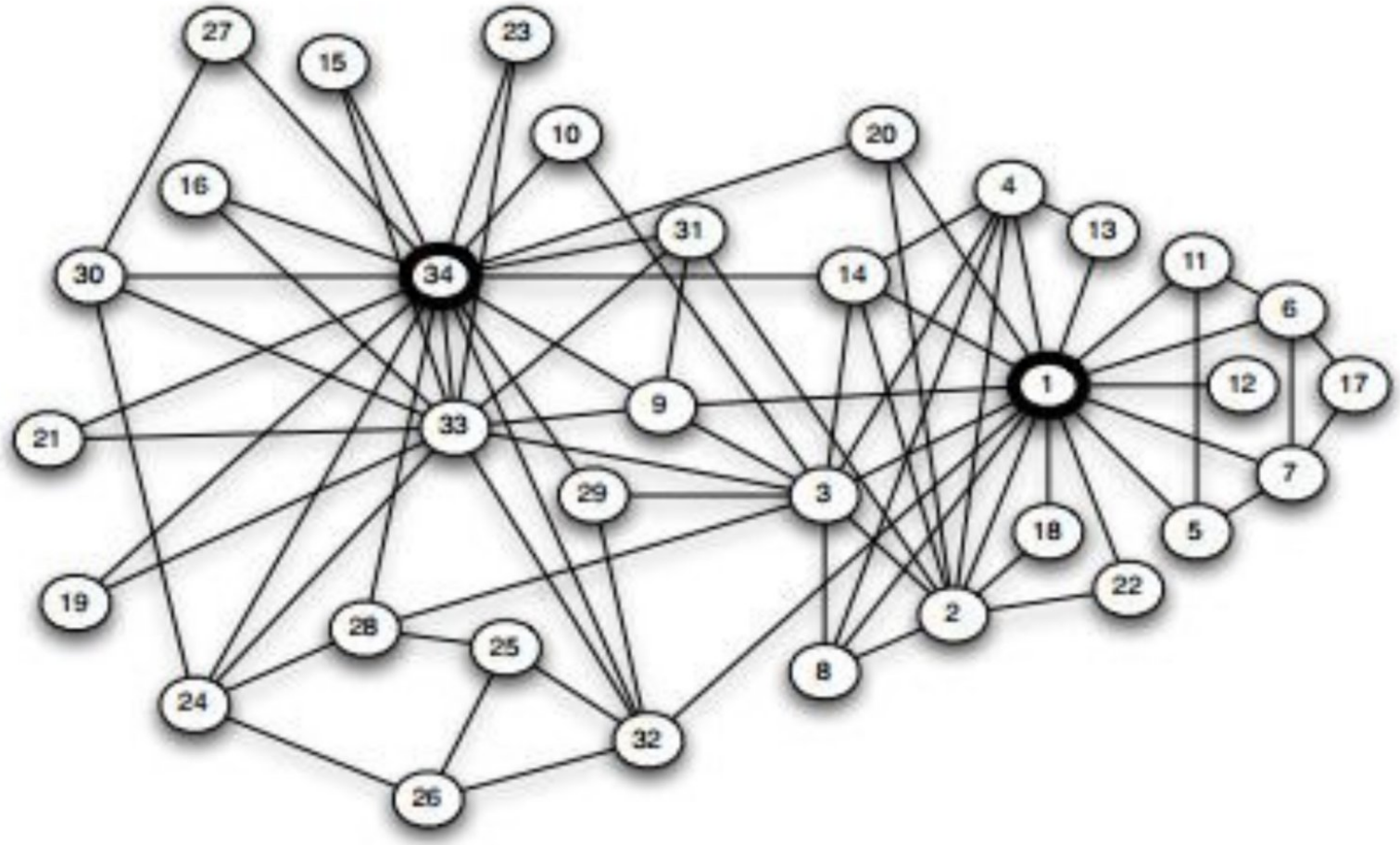
```python
from collections import defaultdict
import networkx as nx

class Grafo_Centralidad:

    def __init__(self,dirigido=False):
        self.dirigido = dirigido
        if dirigido:
            self.G = nx.DiGraph()
        else:
            self.G = nx.Graph()

    def cargar_aristas(self,nombre_aristas):
        with open(nombre_aristas,"r") as archivo_aristas:
            for arista in archivo_aristas:
                arista = arista.strip()
                par = arista.split()
                self.G.add_edge(par[0],par[1])
```

CENTRALIDAD DE GRADO


27
15
23
10
20
16
4
13
31
11
30
34
14
6
1
12
17
9
21
33
7
29
3
18
5
19
22
2
28
25
8
24
32
26

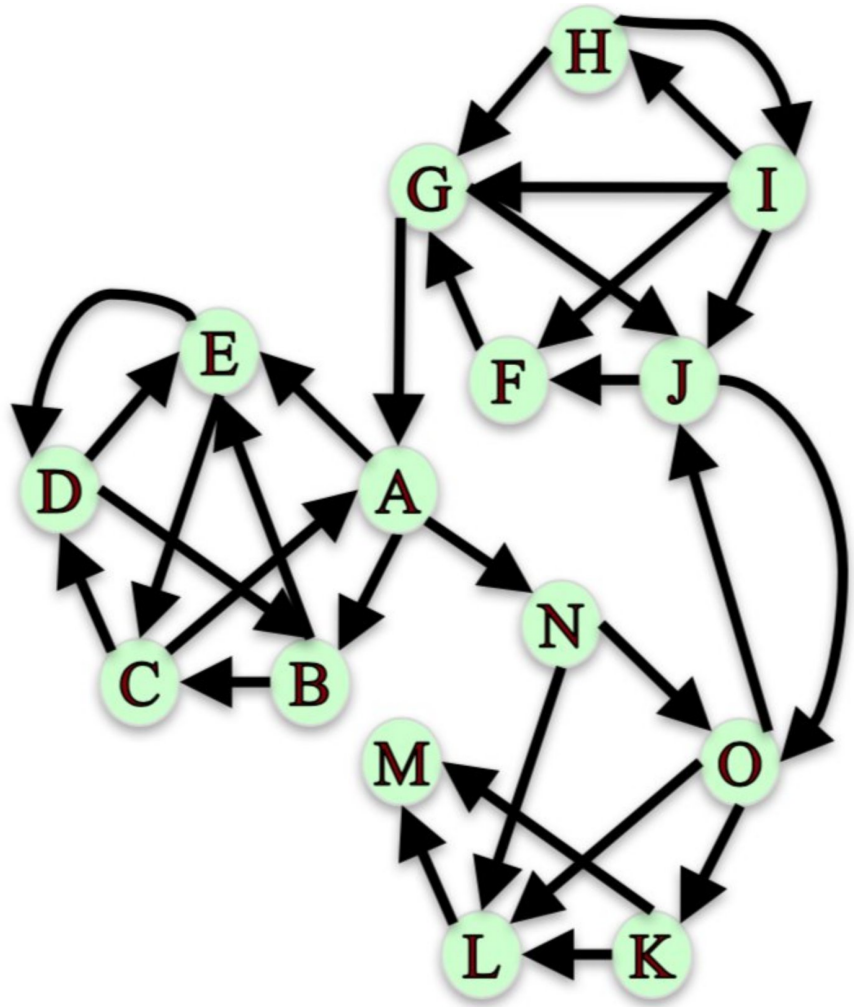
Para grafos no dirigidos

$$Cg(v) = \frac{d_v}{N-1}$$

Para grafos no dirigidos

```
def centralidad_grado(self):
    N = self.G.number_of_nodes()

    centralidad = {}
    for nodo in self.G.nodes:
        centralidad[nodo] = self.G.degree[nodo]/(N-1)
    return centralidad
```


H
G
I
E
F
J
D
A
N
C
B
M
O
L
K

Para grafos dirigidos

$$Cg_{\text{in}}(v) = \frac{d_v^{\text{in}}}{N-1}$$

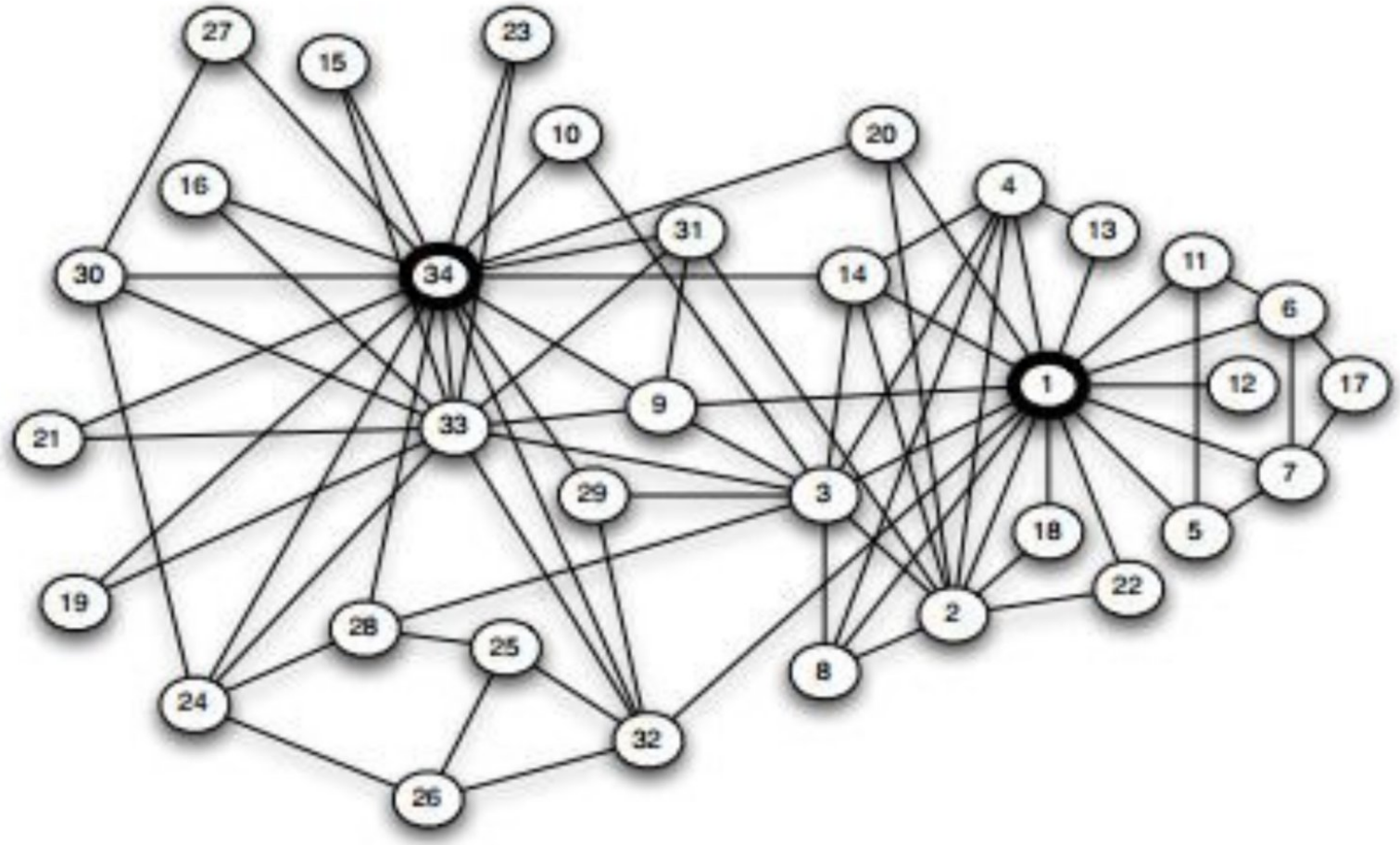
$$Cg_{out}(v) = \frac{d_v^{out}}{N-1}$$

Para grafos dirigidos

```
def centralidad_grado(self):
    N = self.G.number_of_nodes()

    if self.dirigido:
        centralidad_in = {}
        centralidad_out = {}
        for nodo in self.G.nodes:
            centralidad_in[nodo] = self.G.in_degree[nodo]/(N-1)
            centralidad_out[nodo] = self.G.out_degree[nodo]/(N-1)
        return centralidad_in , centralidad_out
```

CENTRALIDAD DE CERCANÍA


27
15
23
10
20
16
4
13
31
11
30
34
14
6
1
12
17
9
21
33
7
29
3
18
5
19
22
2
28
25
8
24
32
26

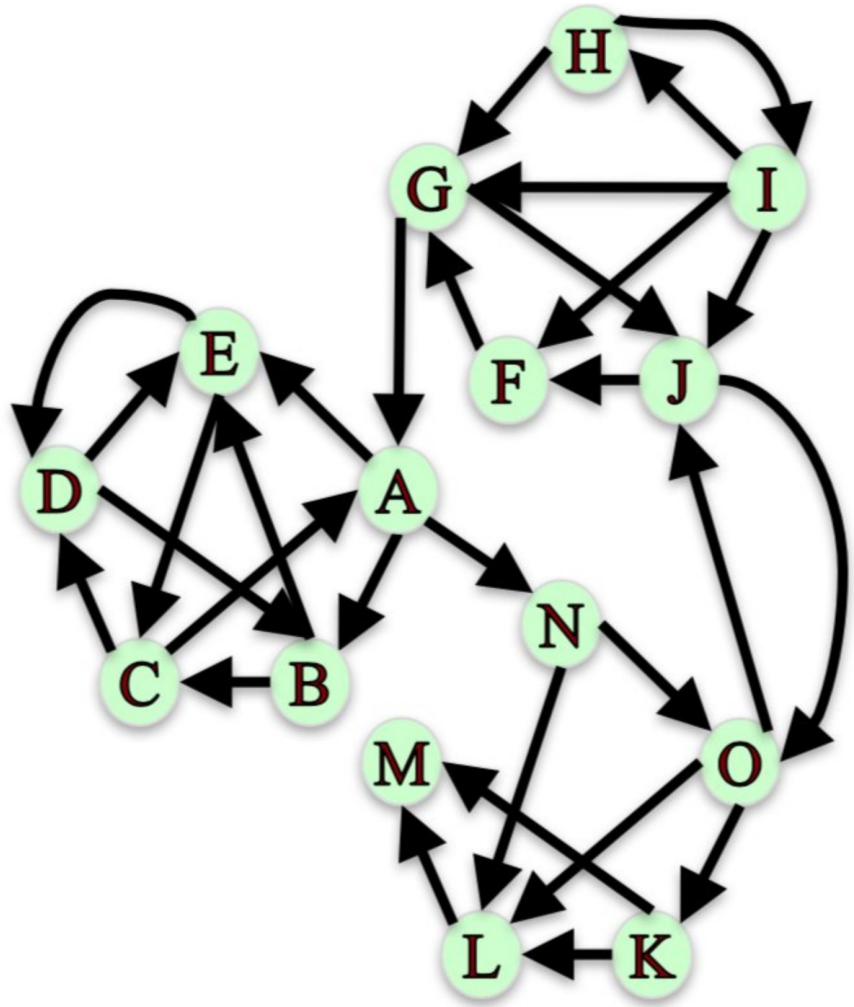
Para grafos no dirigidos

$$Cc(v) = \frac{N-1}{\sum_{u \in N} d(v,u)}$$

Para grafos no dirigidos

```
def centralidad_cercania(self):
    N = self.G.number_of_nodes()
    centralidad = {}

    for nodoOrigen in self.G.nodes:
        suma = 0
        for nodoDestino in self.G.nodes:
            min_dist = nx.shortest_path_length(self.G,nodoOrigen,nodoDestino)
            suma += min_dist
        centralidad[nodoOrigen] = (N-1)/suma
    return centralidad
```


H
G
I
E
F
J
D
A
N
C
B
M
O
L
K

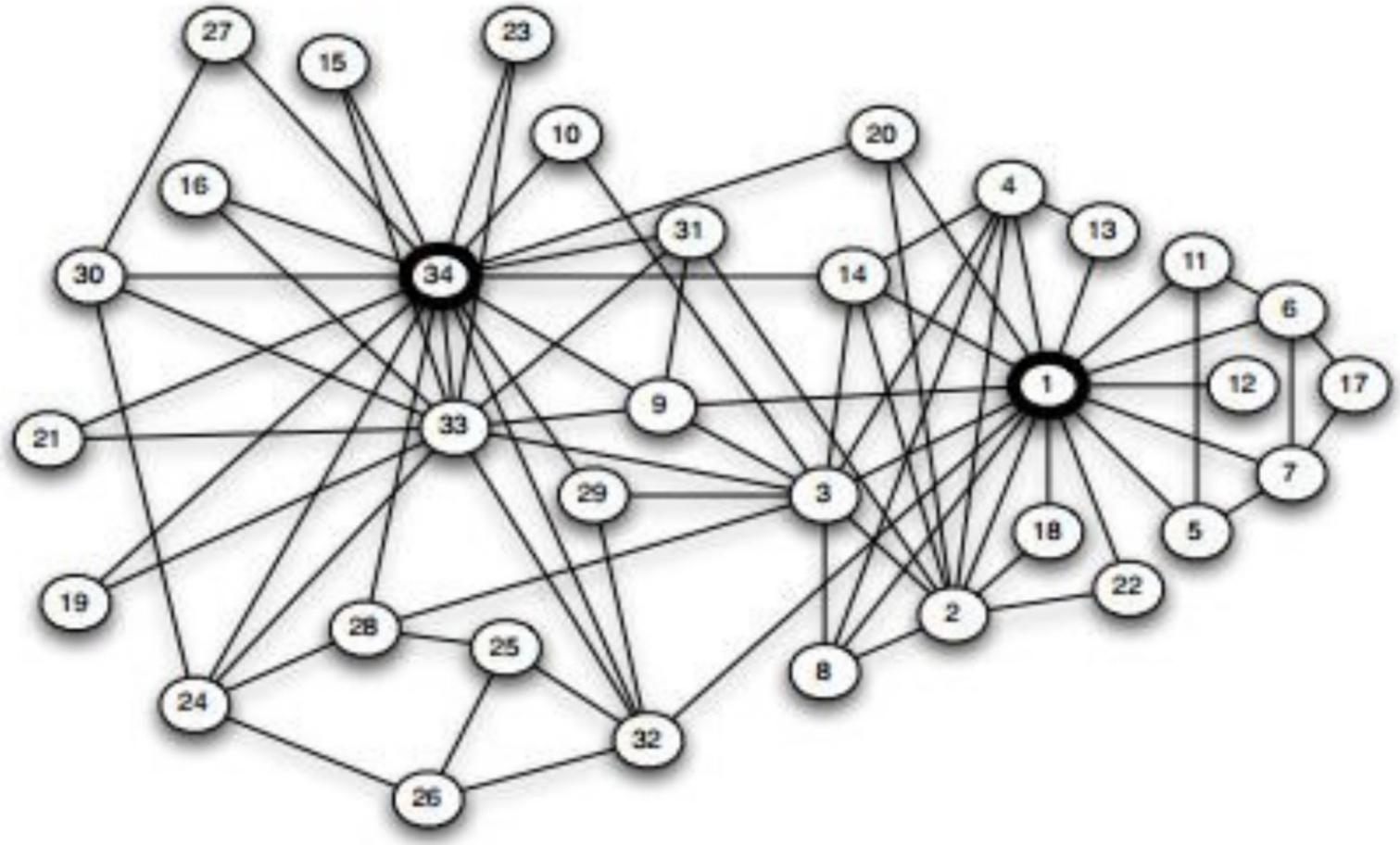
Para grafos dirigidos

$$Cc_d(v) = \frac{R(v)}{N-1} * \frac{R(v)}{\sum_{u \in R(v)} d(v,u)}$$

Para grafos dirigidos

```
def centralidad_cercania(self):
    N = self.G.number_of_nodes()
    centralidad = {}
    if self.dirigido:
        for nodoOrigen in self.G.nodes:
            R = 0
            suma = 0
            for nodoDestino in self.G.nodes:
                try:
                    min_dist = nx.shortest_path_length(self.G,nodoOrigen,nodoDestino)
                    R += 1
                    suma += min_dist
                except:
                    pass
            if suma == 0:
                centralidad[nodoOrigen] = 0
            else:
                centralidad[nodoOrigen] = R/(N-1) * R/suma
        return centralidad
```

CENTRALIDAD INTERMEDIA


27
15
23
10
20
16
4
13
31
11
30
34
14
6
1
12
17
9
21
33
7
29
3
18
5
19
22
2
28
25
8
24
32
26

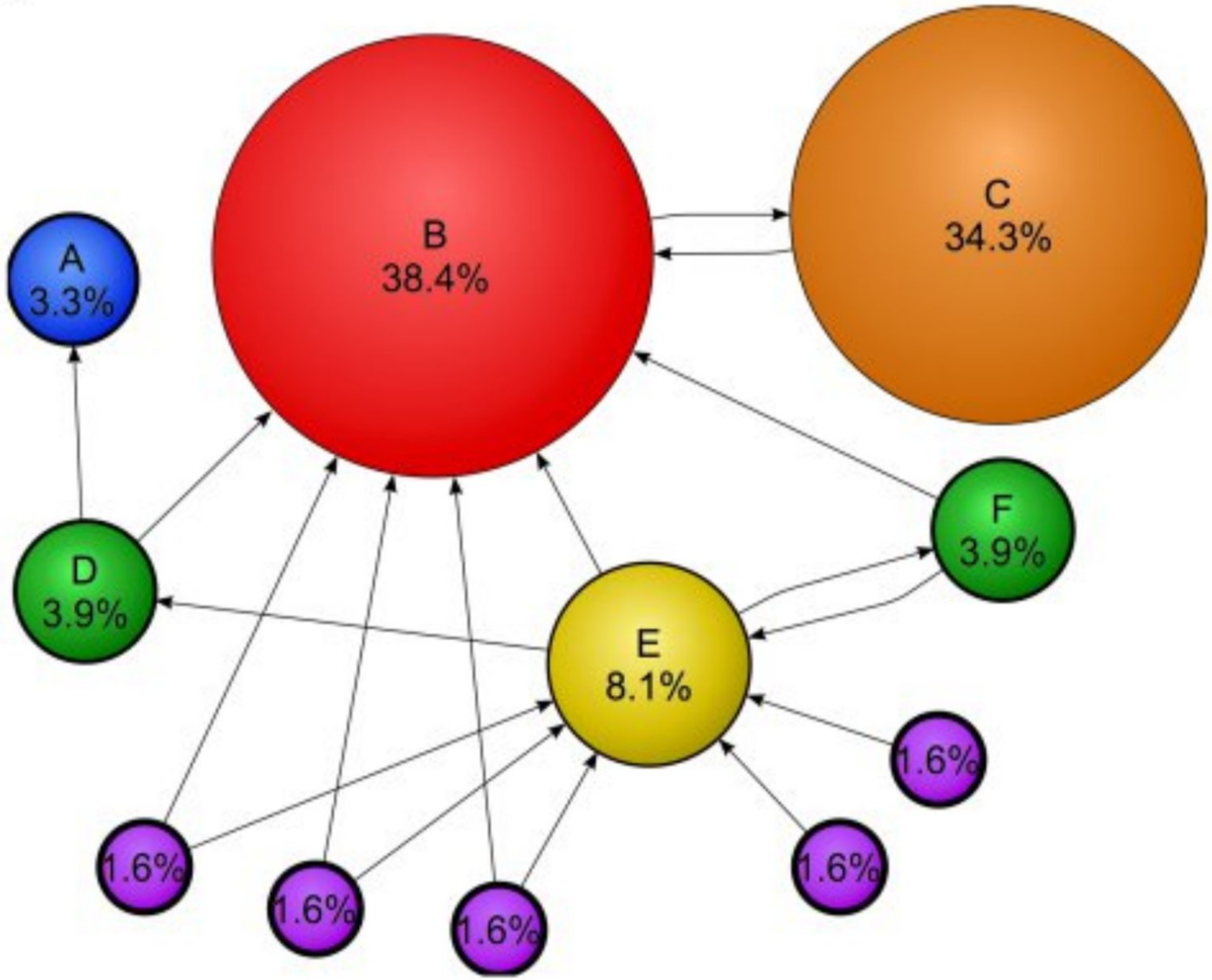
$$Ci(v) = \sum_{s,t \in N} \frac{\sigma_{s,t}(v)}{\sigma_{s,t}}$$

Normalización

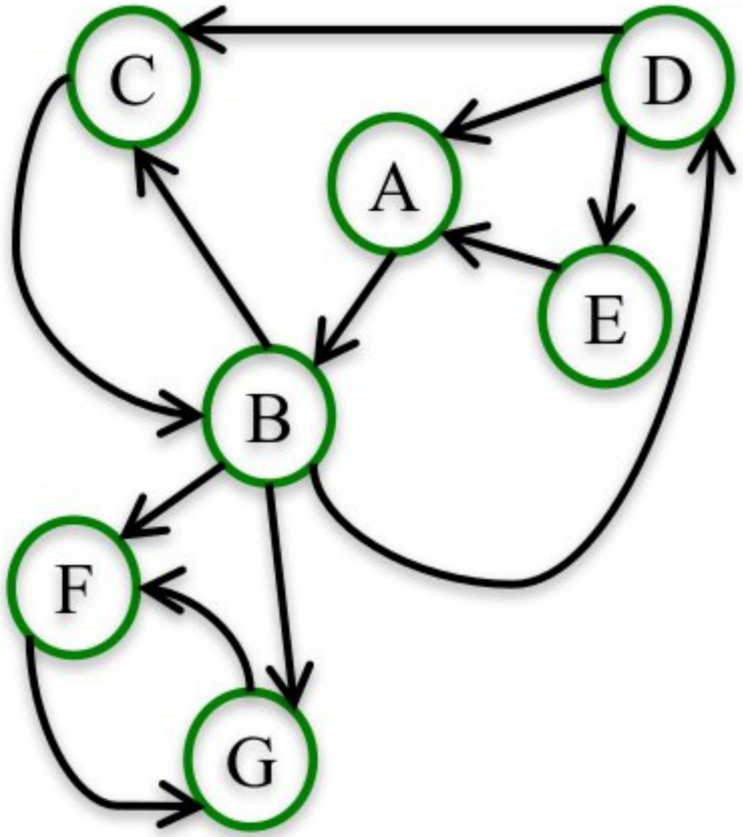
$$\frac{Ci_{nd}(v)}{\frac{1}{2}(N-1)(N-2)} \mid \frac{Ci_{d}(v)}{(N-1)(N-2)}$$

```python
def centralidad_intermedia(self):
    N = self.G.number_of_nodes()
    centralidad = defaultdict(lambda:0)
    for nodoOrigen in self.G.nodes:
        for nodoDestino in self.G.nodes:
            num_rutas_transversales = defaultdict(lambda:0)
            num_rutas = 0
            rutas_minimas = nx.all_shortest_paths(self.G,nodoOrigen,nodoDestino)
            try:
                for ruta in rutas_minimas:
                    num_rutas += 1
                    for nodo in ruta:
                        num_rutas_transversales[nodo] += 1
                for nodo, rutas in num_rutas_transversales.items():
                    centralidad[nodo] += rutas/num_rutas
            except:
                pass
    for nodo, C in centralidad.items():
        pares_totales = (N-1)*(N-2)
        if self.dirigido:
            centralidad[nodo] = C/pares_totales
        else:
            centralidad[nodo] = 2*C/pares_totales
    return centralidad
```

PAGE RANK

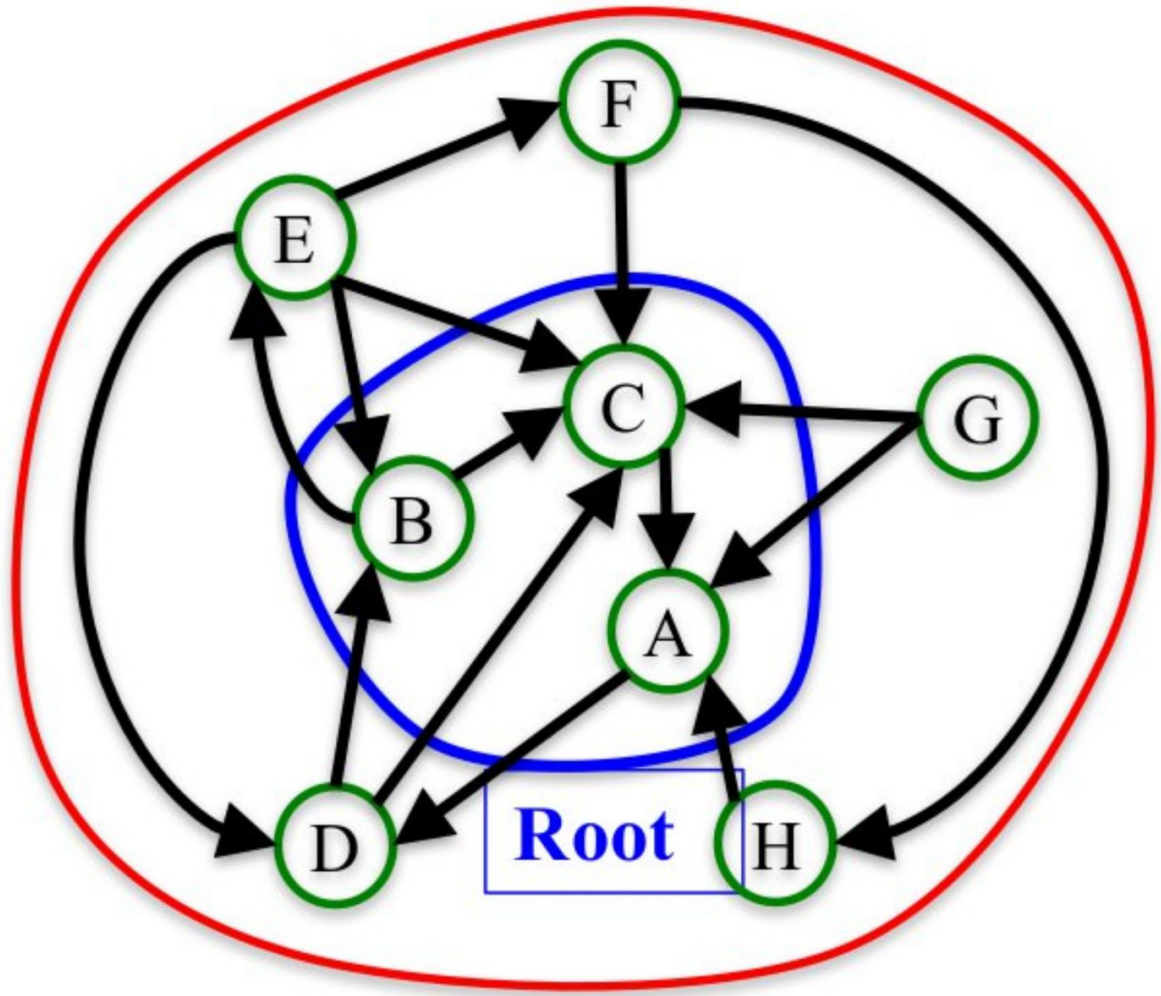

A
3.3%
B
38.4%
C
34.3%
D
3.9%
E
8.1%
F
3.9%
1.6%
1.6%
1.6%
1.6%
1.6%

Amortiguamiento



```
def pagerank(self,k,alpha=0.85):
    N = self.G.number_of_nodes()
    G = self.G.to_directed()
    scores = {}
    for nodo in self.G.nodes:
        scores[nodo] = 1/N
    for i in range(k):
        scores_pre = scores.copy()
        scores = defaultdict(lambda:0)
        for nodoOut in self.G.nodes:
            amortiguamiento = (1-alpha)*scores_pre[nodoOut]/N
            for nodoIn in self.G.nodes:
                scores[nodoIn] += amortiguamiento
            out_degree = G.out_degree[nodo]
            salida = alpha*scores_pre[nodoOut]/out_degree
            for vecino in self.G[nodoOut]:
                scores[vecino] += salida
    return scores
```

HUBS y AUTORIDADES


F
E
C
G
B
A
D
Root
H
Base

```
def hits(self,k):
    N = self.G.number_of_nodes()
    hubs = {}
    autoridades = {}
    for nodo in self.G.nodes:
        hubs[nodo] = 1/N
        autoridades[nodo] = 1/N
    for i in range(k):
        hubs_pre = hubs.copy()
        autoridades_pre = autoridades.copy()
        hubs = defaultdict(lambda:0)
        autoridades = defaultdict(lambda:0)
        total_hubs = 0
        total_autoridades = 0
        for nodoOut in self.G.nodes:
            for vecino in self.G[nodoOut]:
                hubs[nodoOut] += autoridades_pre[vecino]
                autoridades[vecino] += hubs_pre[nodoOut]
                total_hubs += autoridades_pre[vecino]
                total_autoridades += hubs_pre[nodoOut]
        for nodo in self.G.nodes:
            hubs[nodo] = hubs[nodo]/total_hubs
            autoridades[nodo] = autoridades[nodo]/total_autoridades
    return hubs , autoridades
```

LIBRERÍAS

- **Centralidad de grado:**

```
C = nx.degree_centrality(G)
Cin = nx.in_degree_centrality(G)
Cout = nx.out_degree_centrality(G)
```

- **Centralidad de Cercanía:**

```
C = nx.clossness_centrality(G)
```

LIBRERÍAS

- Centralidad Intermedia:

```
C = nx.betweenness_centrality(G)
```

- Page Rank:

```
PR = nx.pagerank(G,alpha=0.85)
```

- Hubs y Autoridades:

```
H,A = nx.hits(G)
```