Análise de Redes

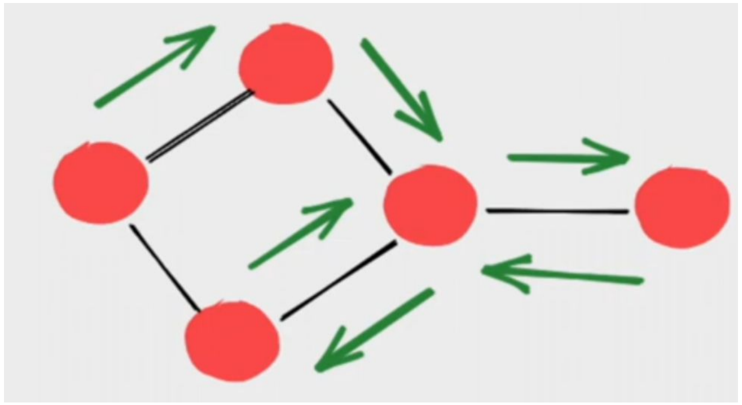
Slide 05 - Walks, Paths, Distances

Prof. Patrick Terrematte

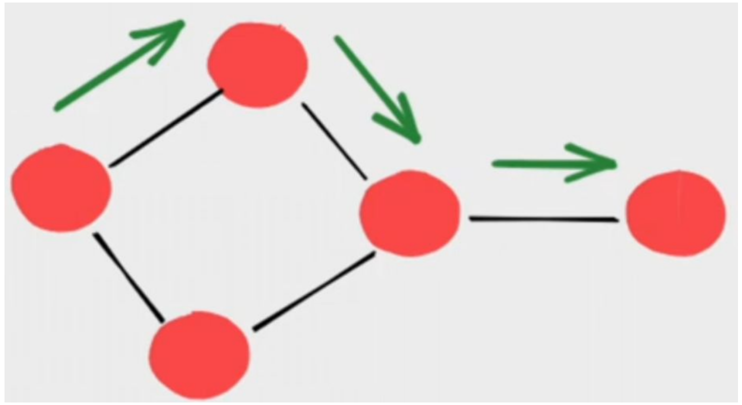
Análise de Redes

- Caminhada / Walk
  - Sequência de vértices consecutivos conectados por arestas.
- Caminho / Path
  - Sequência de vértices consecutivos conectados por arestas, sem ciclos.
- Distância
  - Menor caminho entre dois vértices.

Walk x Path

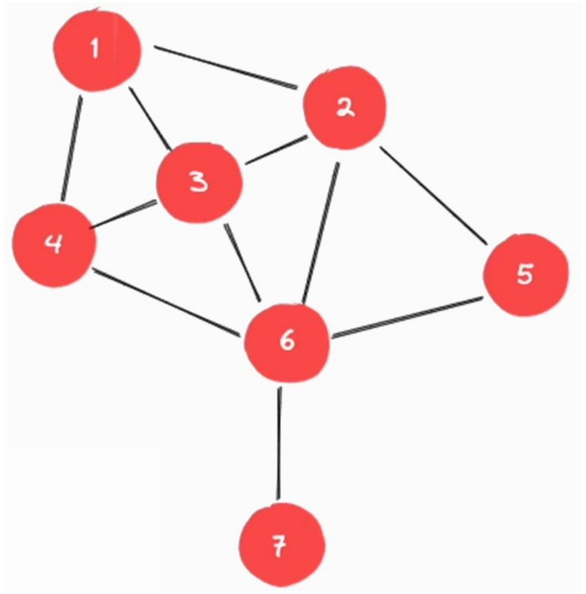


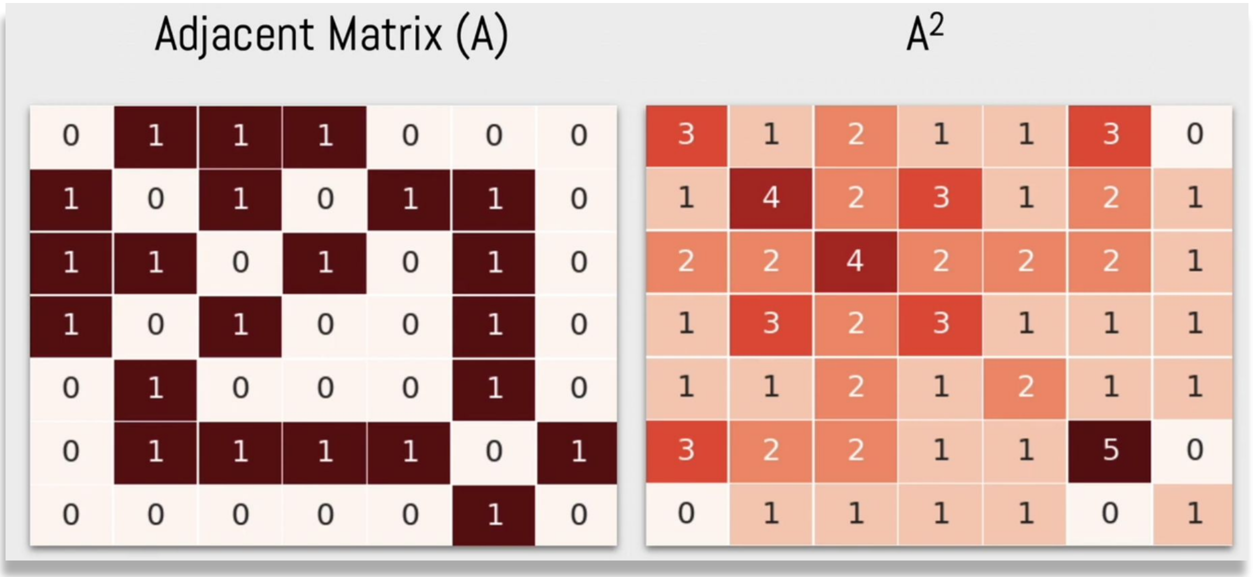
Caminhada de comprimento 6



Caminho de comprimento 3

Walk





Adjacent Matrix (A)

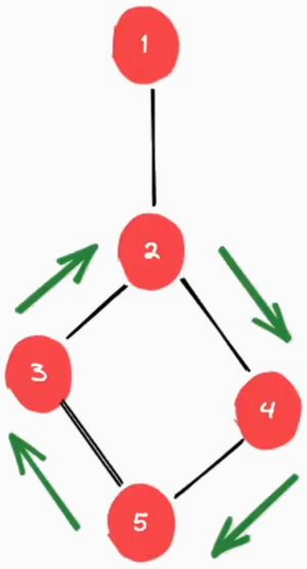
0	1	1	1	0	0	0
1	0	1	0	1	1	0
1	1	0	1	0	1	0
1	0	1	0	0	1	0
0	1	0	0	0	1	0
0	1	1	1	1	0	1
0	0	0	0	0	1	0

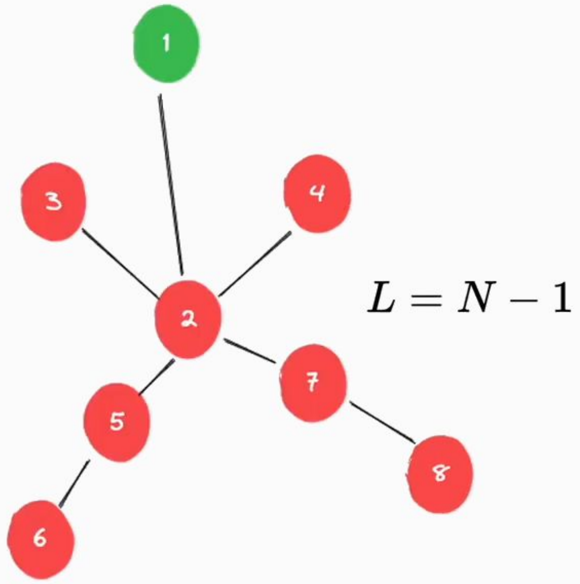
A^2

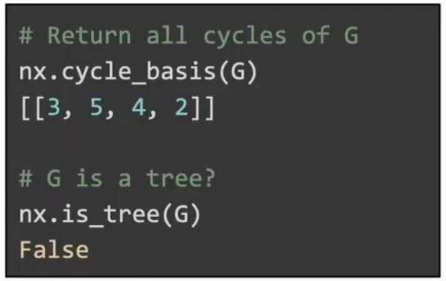
3	1	2	1	1	3	0
1	4	2	3	1	2	1
2	2	4	2	2	2	1
1	3	2	3	1	1	1
1	1	2	1	2	1	1
3	2	2	1	1	5	0
0	1	1	1	1	0	1

Os caminhos de comprimento 2 da rede.
A diagonal representa os graus de cada nó.

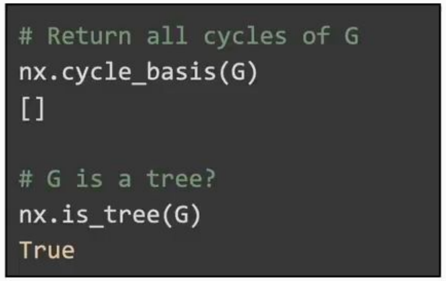
Ciclos







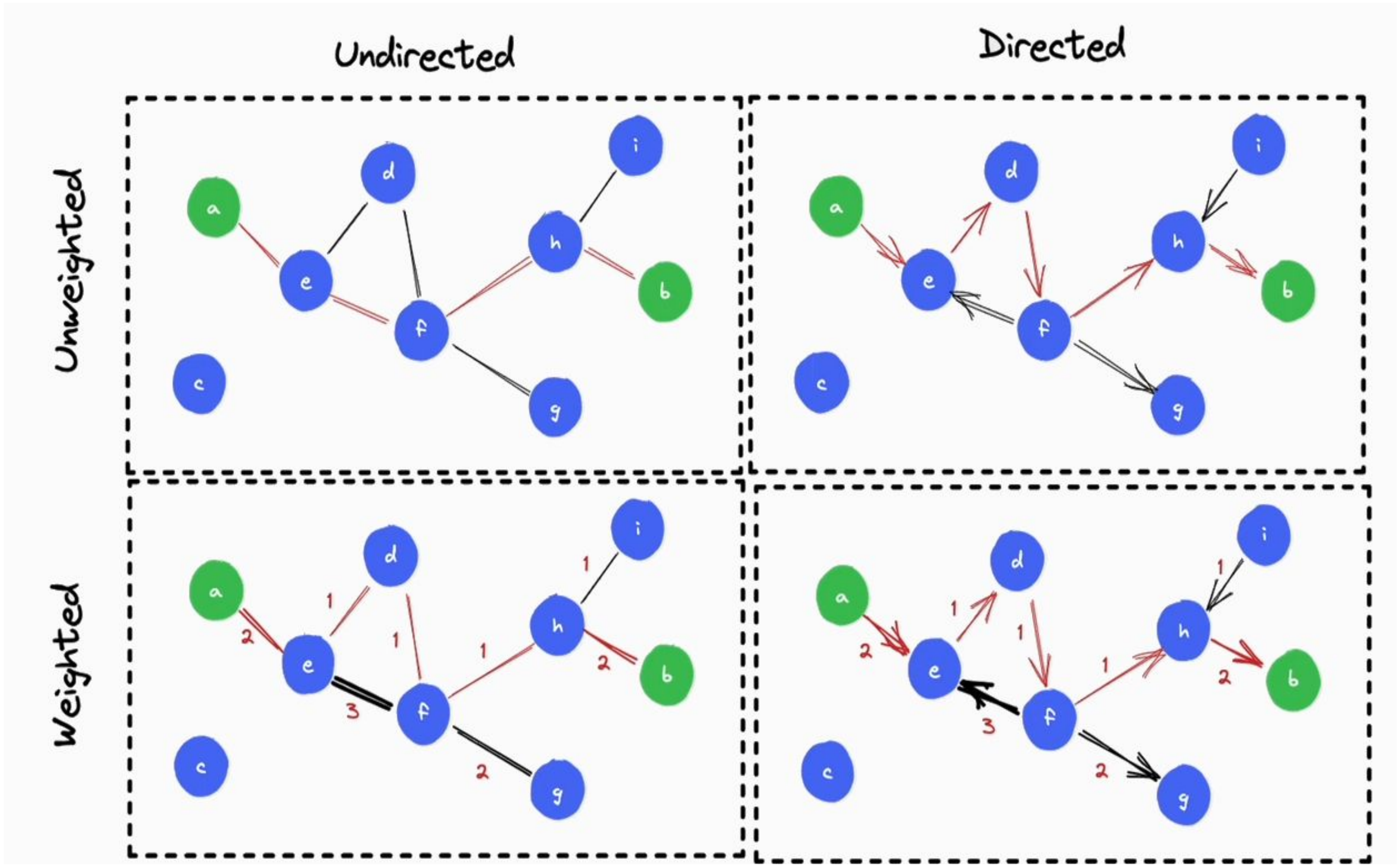
```
# Return all cycles of G
nx.cycle_basis(G)
[[3, 5, 4, 2]]

# G is a tree?
nx.is_tree(G)
False
```



```
# Return all cycles of G
nx.cycle_basis(G)
[]

# G is a tree?
nx.is_tree(G)
True
```

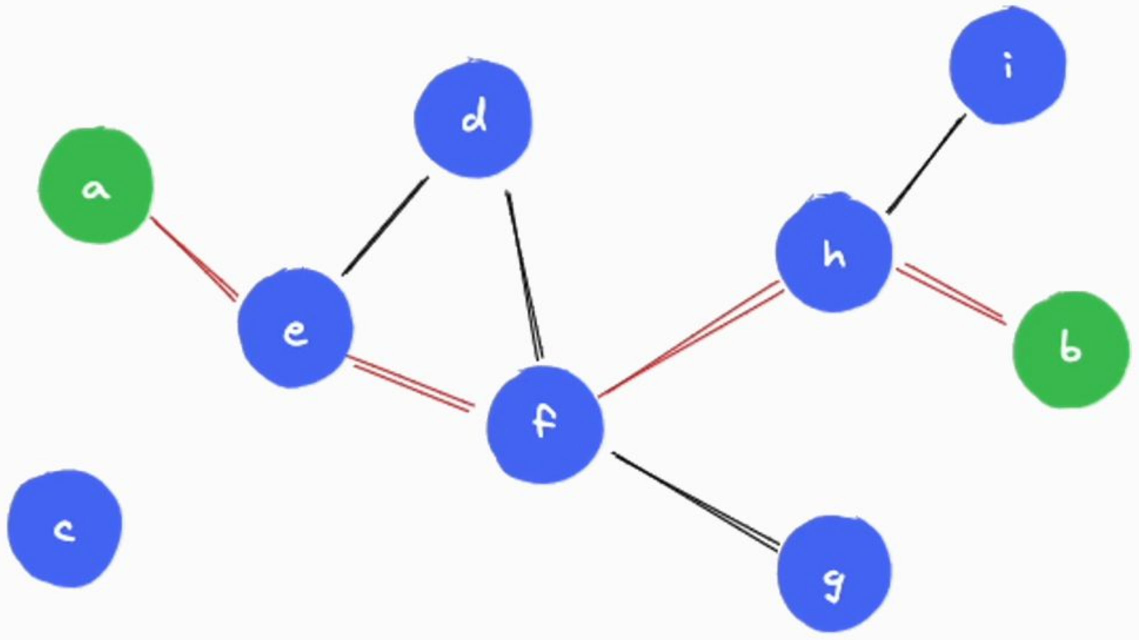
NetworkX
Network Analysis in Python

Caminho mais curto



Caminho mais curto médio

$$\langle l \rangle = \frac{\sum_{ij} l_{ij}}{\binom{N}{2}} = \frac{2\sum_{ij} l_{ij}}{N(N-1)}$$



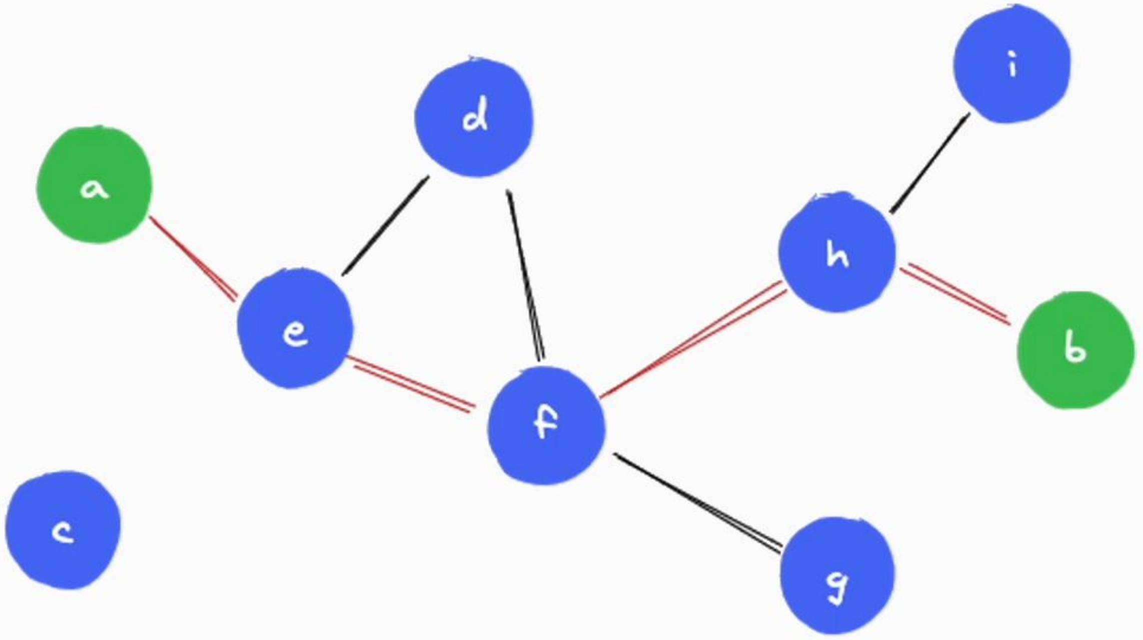
Diametro da rede

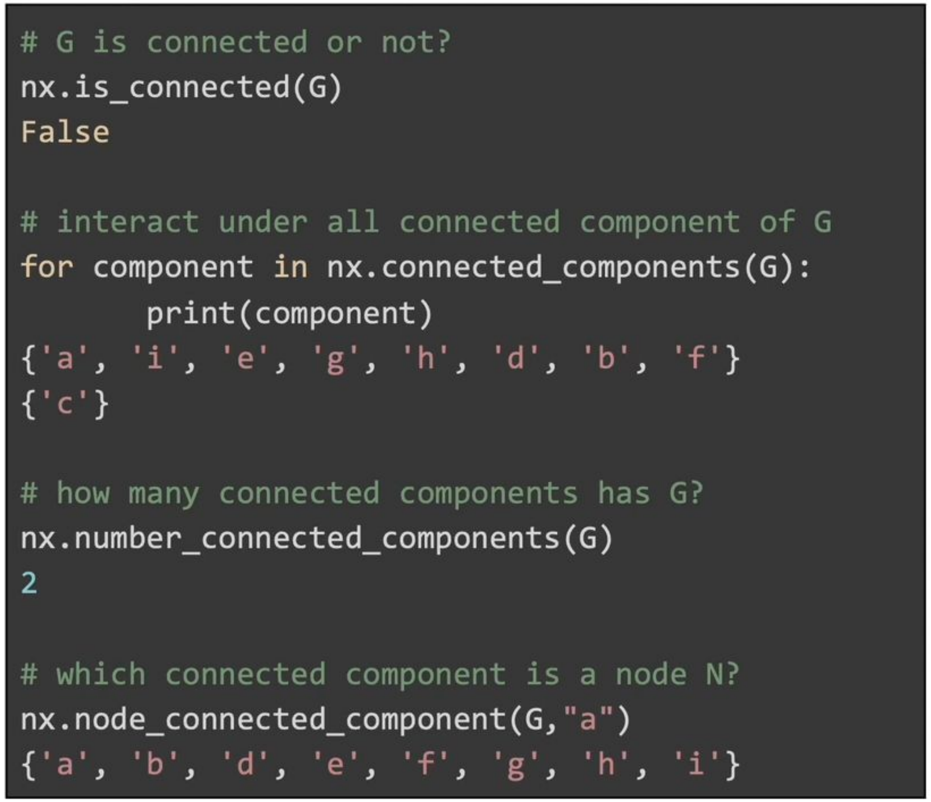
$$l_{max} = \max_{ij}\ l_{ij}$$

(src, dest)
(a,b)
a - e - f - h - b
(a,c)
-
(a,d)
a - e - d
(a,e)
a - e
....

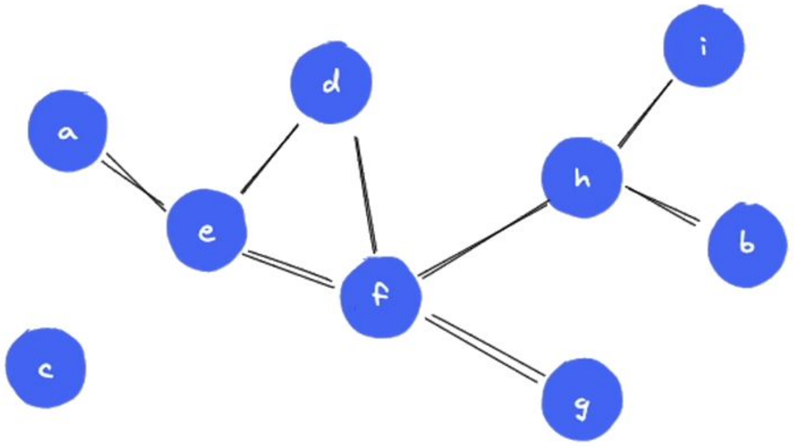
(b,c)
-
(b,d)
b - h - f - d
(b,e)
b - h - f - e
(b,f)
b - h - f
....

....





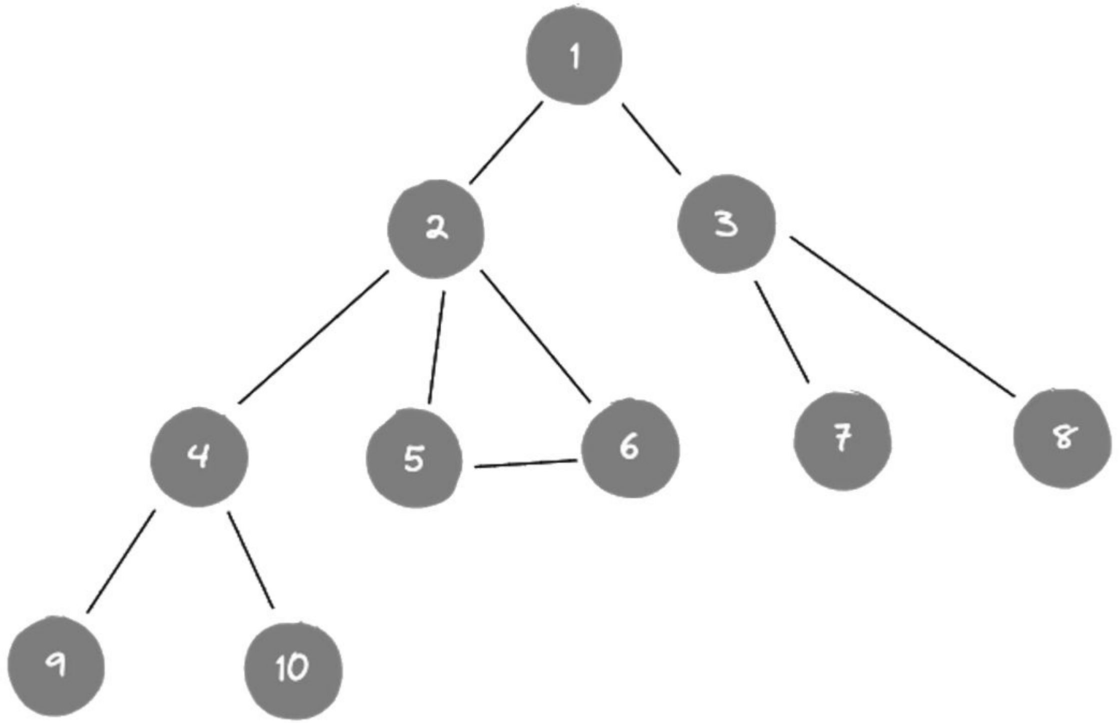
```
# G is connected or not?
nx.is_connected(G)
False

# interact under all connected component of G
for component in nx.connected_components(G):
       print(component)
{'a', 'i', 'e', 'g', 'h', 'd', 'b', 'f'}
{'c'}

# how many connected components has G?
nx.number_connected_components(G)
2

# which connected component is a node N?
nx.node_connected_component(G,"a")
{'a', 'b', 'd', 'e', 'f', 'g', 'h', 'i'}
```



NetworkX
Network Analysis in Python

metropole DIGITAL

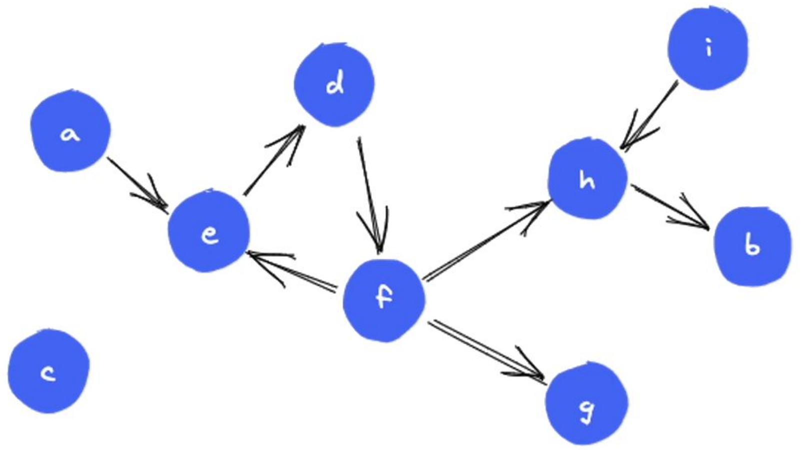
Breadth First Search (BFS) versus Depth First Search (DFS)

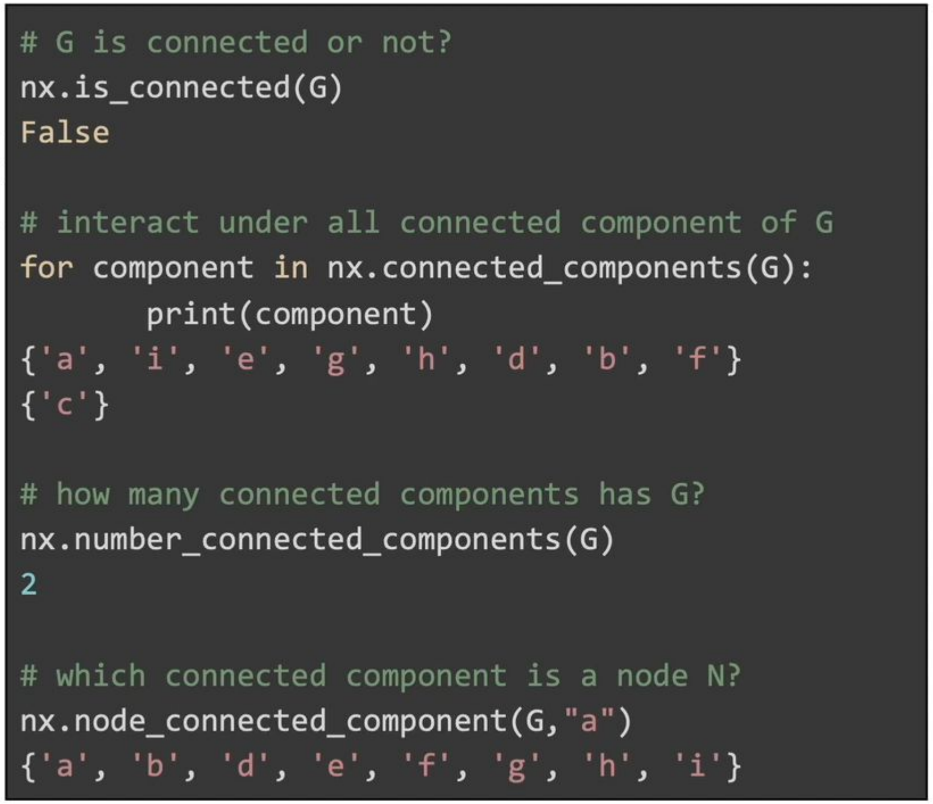


Key	Breadth First Search (BFS)	Depth First Search (DFS)
Definition	BFS stands for Breadth First Search.	DFS stands for Depth First Search.
Data structure	Uses a Queue to find shortest path. FIFO (First In First Out)	Uses a Stack to find the shortest path. LIFO (Last In First Out)
Source	BFS is better when target is closer to Source.	DFS is better when target is far from source.
Suitability for decision tree	As BFS considers all neighbor so it is not suitable for decision tree used in puzzle games.	DFS is more suitable for decision tree. As with one decision, we need to traverse further to augment the decision. If we reach the conclusion, we won.
Speed	BFS is slower than DFS.	DFS is faster than BFS.
Time Complexity	Time Complexity of BFS = O(V+E) where V is vertices and E is edges.	Time Complexity of DFS is also O(V+E) where V is vertices and E is edges.
Memory	BFS requires more memory space.	DFS requires less memory space.

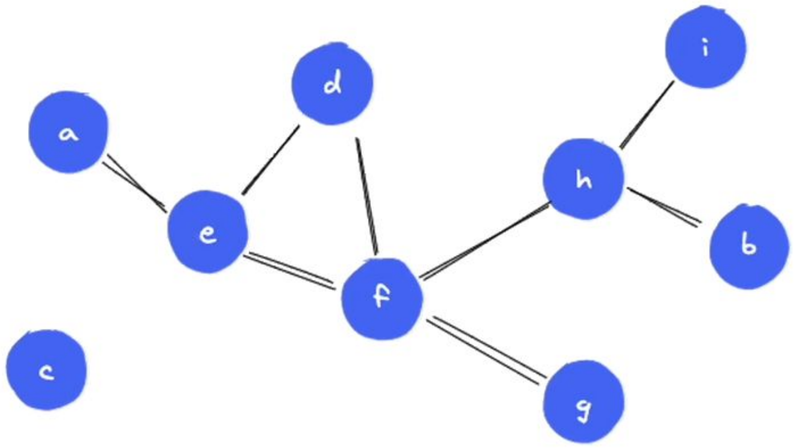
Componentes Conectados

- Se dois nós não podem ser conectados por uma caminhada, então estão em diferentes componentes conectados (subgrafos).
- Componentes conectados são subgrafos com nós que podem ser atingidos pelas arestas/links da rede.
- Giant Connected Components (GCC) é um subgrafo que se destaca dos outros subgrafos pelo número de nós.





```
# G is connected or not?
nx.is_connected(G)
False

# interact under all connected component of G
for component in nx.connected_components(G):
        print(component)
{'a', 'i', 'e', 'g', 'h', 'd', 'b', 'f'}
{'c'}

# how many connected components has G?
nx.number_connected_components(G)
2

# which connected component is a node N?
nx.node_connected_component(G,"a")
{'a', 'b', 'd', 'e', 'f', 'g', 'h', 'i'}
```

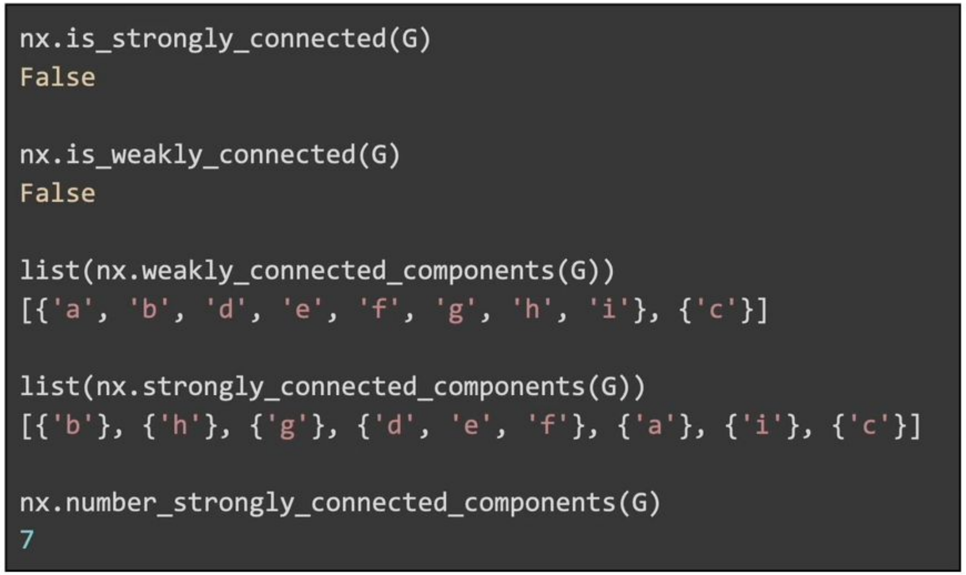


NetworkX
Network Analysis in Python

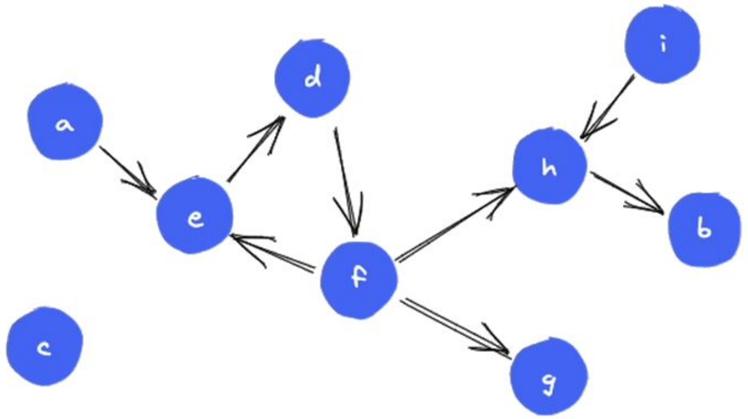
metropole DIGITAL

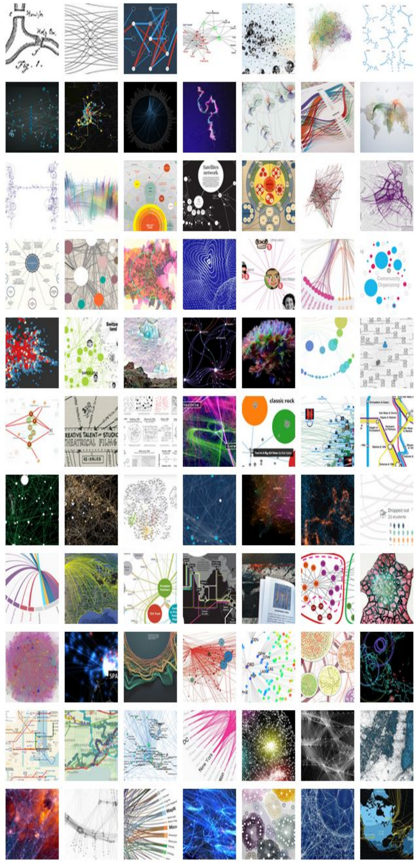
Strongly Connected Components (SCC) Weakly Connected Components (WCC)

Um componente é fortemente conectado, quando em um subgrafo todos os nós se acessam.



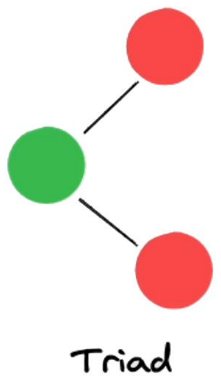
```
nx.is_strongly_connected(G)
False

nx.is_weakly_connected(G)
False

list(nx.weakly_connected_components(G))
[{'a', 'b', 'd', 'e', 'f', 'g', 'h', 'i'}, {'c'}]

list(nx.strongly_connected_components(G))
[{'b'}, {'h'}, {'g'}, {'d', 'e', 'f'}, {'a'}, {'i'}, {'c'}]

nx.number_strongly_connected_components(G)
7
```

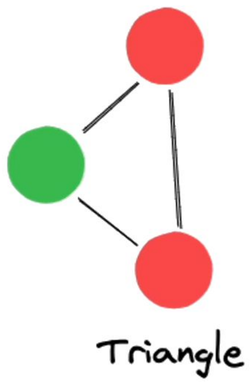




Análise de Redes - Colab

http://colab.research.google.com/github/terrematte/network_analysis/blob/main/notebooks/02_hubs_path_and_structures.ipynb


Triad

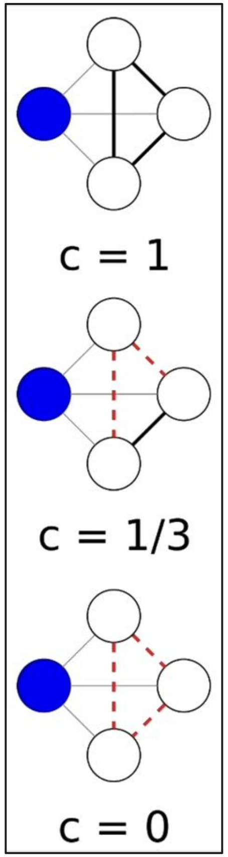

Triangle

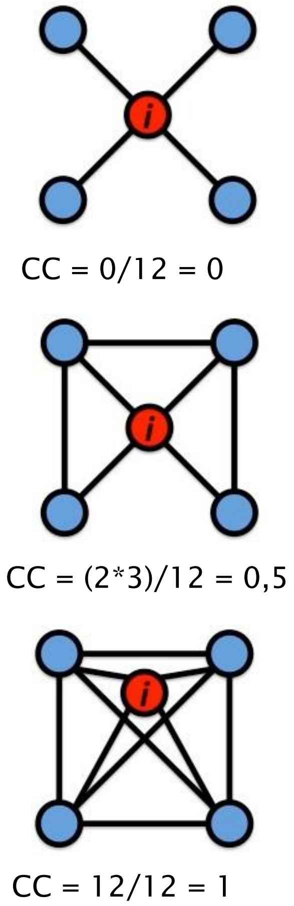
Coeficiente de Clusterização

- O coeficiente de clusterização é a fração de todos possíveis triângulos que contém aquele nó.
- Mede o quanto os nós se relacionam.
- CC não expressa uma propriedade do vértice e sim dos seus vizinhos!

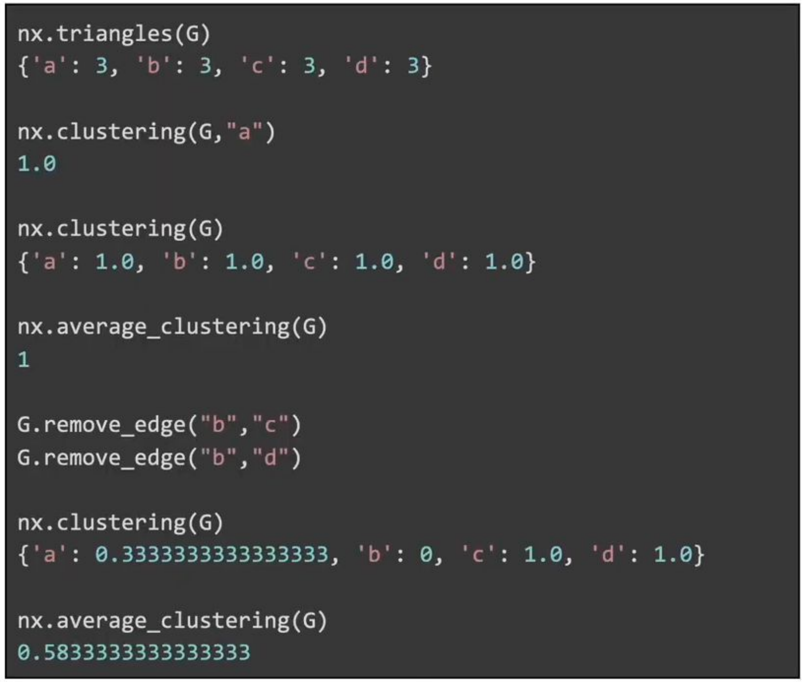
$$C(i) = \frac{\tau(i)}{\tau_{max}(i)} = \frac{\tau(i)}{\binom{k_i}{2}} = \frac{2\tau(i)}{k_i(k_i-1)}$$

$$C = \frac{\sum_{i;k_i>1} C(i)}{N_{k>1}}$$

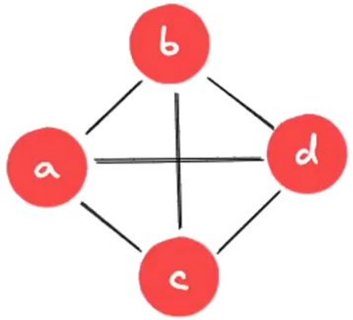


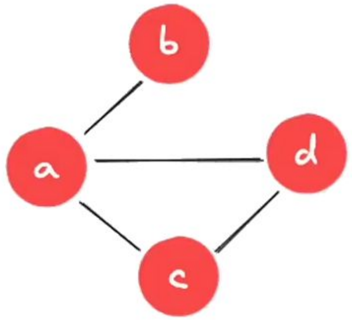


Coeficiente de Clusterização



```
nx.triangles(G)
{'a': 3, 'b': 3, 'c': 3, 'd': 3}

nx.clustering(G,"a")
1.0

nx.clustering(G)
{'a': 1.0, 'b': 1.0, 'c': 1.0, 'd': 1.0}

nx.average_clustering(G)
1

G.remove_edge("b","c")
G.remove_edge("b","d")

nx.clustering(G)
{'a': 0.3333333333333333, 'b': 0, 'c': 1.0, 'd': 1.0}

nx.average_clustering(G)
0.5833333333333333
```

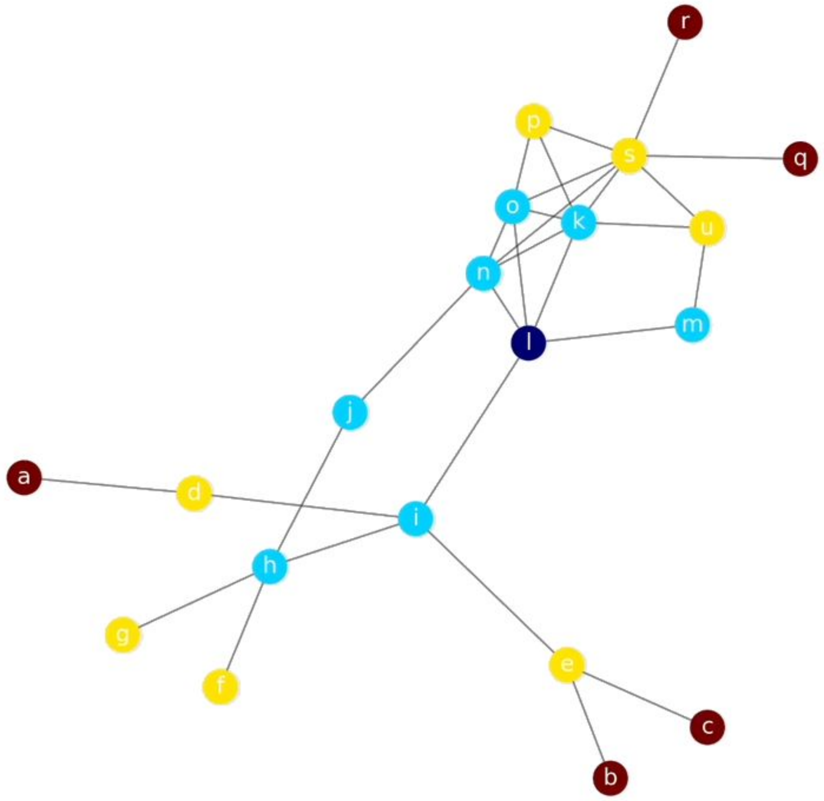




Eccentricity

It is the maximum distance from a node to all other node in the network.

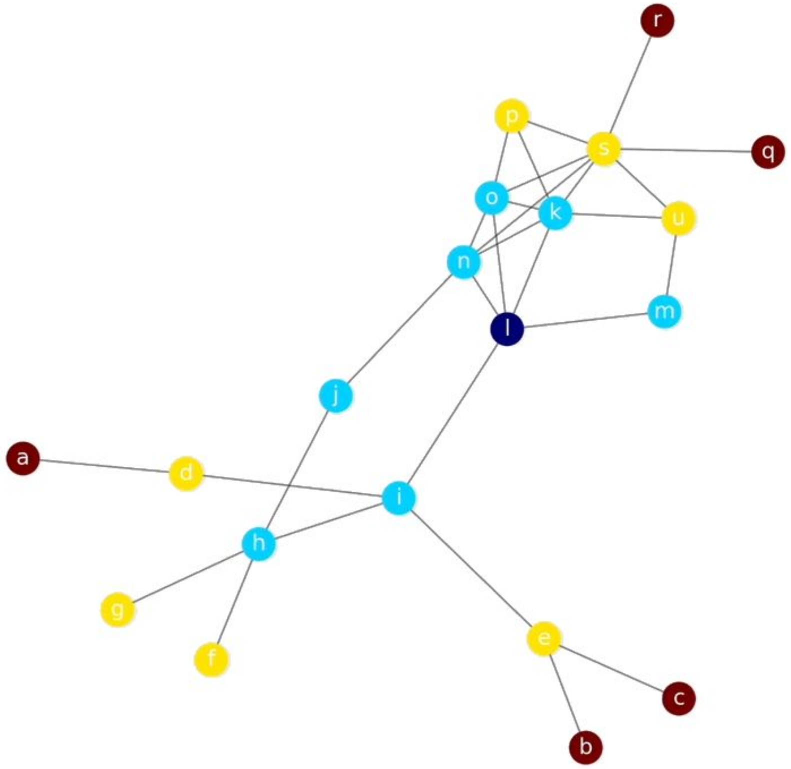
```
nx.eccentricity(g)
{'a': 6, 'b': 6, 'c': 6, 'd': 5,
 'e': 5, 'f': 5, 'g': 5, 'h': 4,
 'i': 4, 'j': 4, 'k': 4, 'l': 3,
 'm': 4, 'n': 4, 'o': 4, 'p': 5,
 'q': 6, 'r': 6, 's': 5, 'u': 5}
```



Diameter

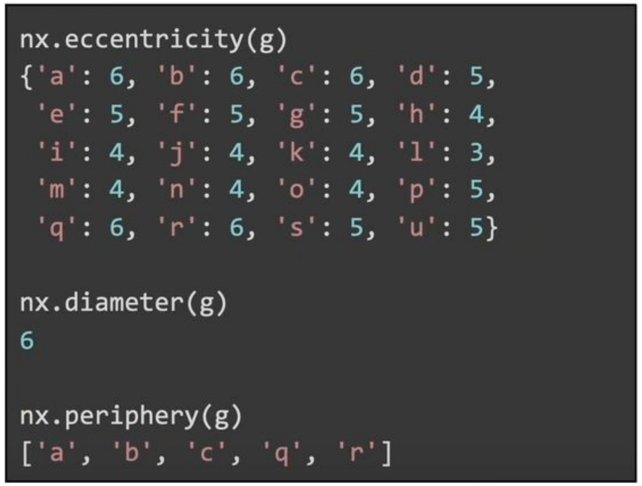
The diameter of a network is the maximum eccentricity.

```
nx.eccentricity(g)
{'a': 6, 'b': 6, 'c': 6, 'd': 5,
 'e': 5, 'f': 5, 'g': 5, 'h': 4,
 'i': 4, 'j': 4, 'k': 4, 'l': 3,
 'm': 4, 'n': 4, 'o': 4, 'p': 5,
 'q': 6, 'r': 6, 's': 5, 'u': 5}

nx.diameter(g)
6

[k for k,v in nx.eccentricity(g).items()
if v == nx.diameter(g)]
['a', 'b', 'c', 'q', 'r']
```

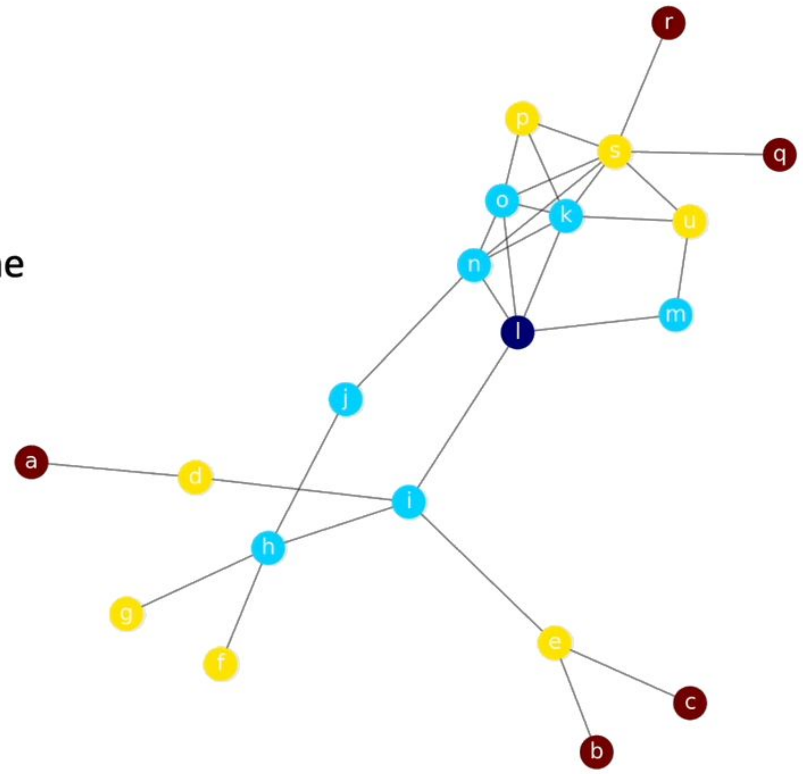


Periphery

The periphery of a network is a set of all nodes whose eccentricity equals the diameter.



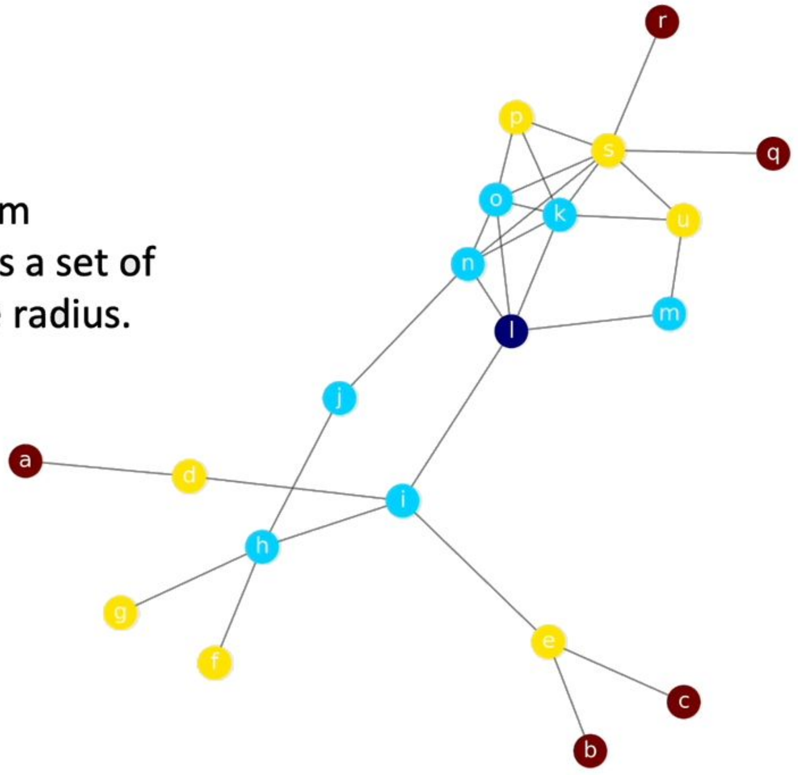
```
nx.eccentricity(g)
{'a': 6, 'b': 6, 'c': 6, 'd': 5,
 'e': 5, 'f': 5, 'g': 5, 'h': 4,
 'i': 4, 'j': 4, 'k': 4, 'l': 3,
 'm': 4, 'n': 4, 'o': 4, 'p': 5,
 'q': 6, 'r': 6, 's': 5, 'u': 5}

nx.diameter(g)
6

nx.periphery(g)
['a', 'b', 'c', 'q', 'r']
```



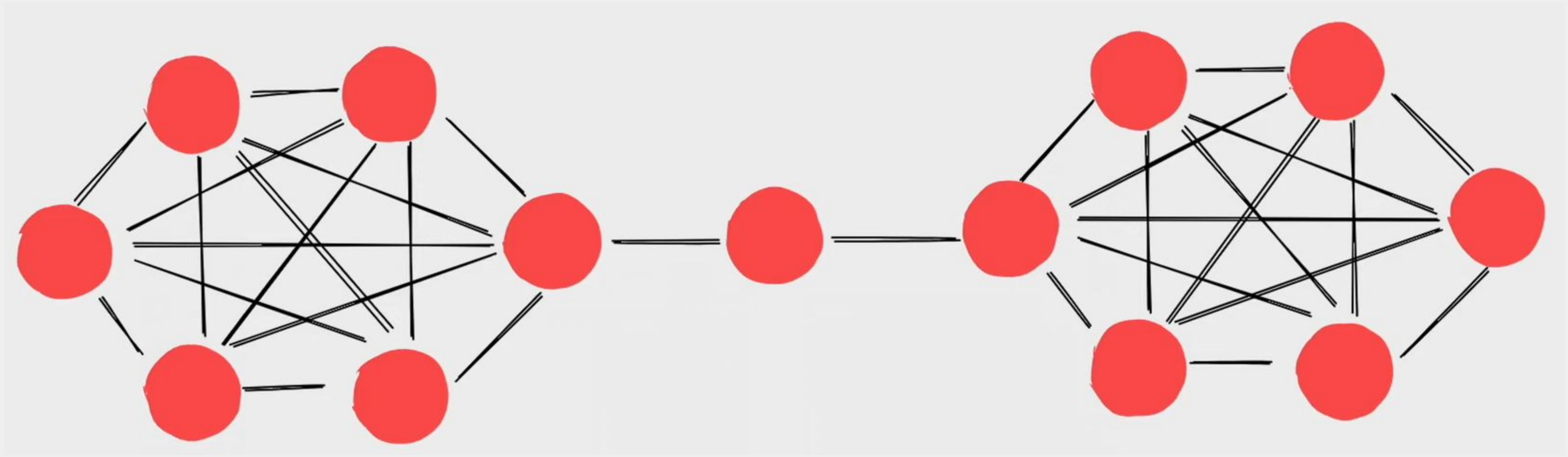
Radius & Center

The radius of a network is the minimum eccentricity. The center of a network is a set of all nodes whose eccentricity equal the radius.

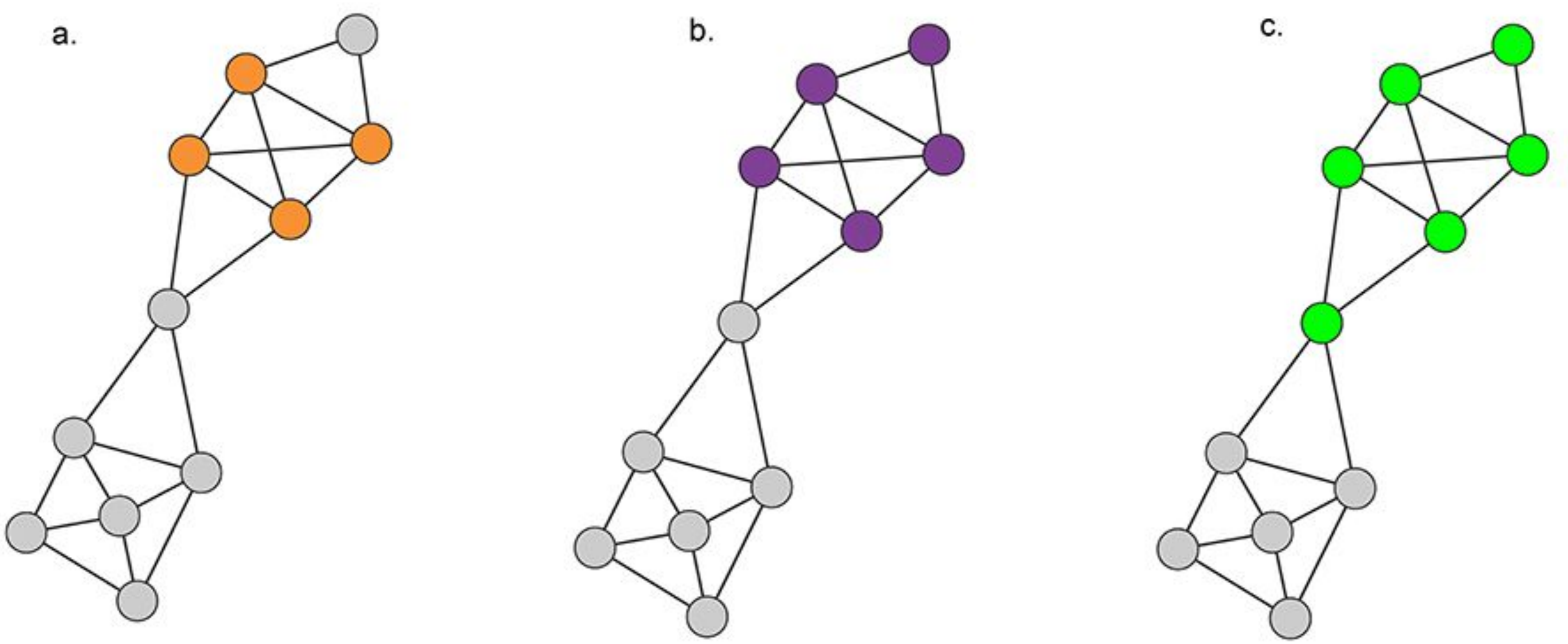
```
nx.eccentricity(g)
{'a': 6, 'b': 6, 'c': 6, 'd': 5,
 'e': 5, 'f': 5, 'g': 5, 'h': 4,
 'i': 4, 'j': 4, 'k': 4, 'l': 3,
 'm': 4, 'n': 4, 'o': 4, 'p': 5,
 'q': 6, 'r': 6, 's': 5, 'u': 5}

nx.radius(g)
3
[k for k,v in nx.eccentricity(g).items()
if v == nx.radius(g)]
['l']
nx.center(g)
['l']
```



Como medir a importância de um nó?



Como medir a importância de um nó?



Degree Centrality

Number of connections

Closeness centrality

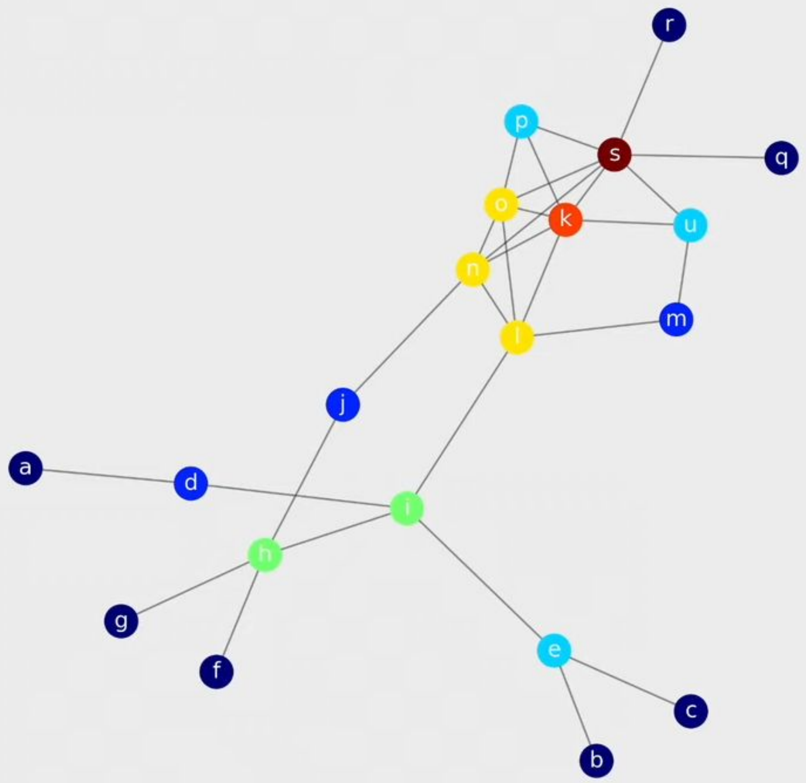
Average distance to all other vertices

Betweenness Centrality

Position on the shortest path

Eigenvector Centrality

Authority score based on the score of the neighbors



Degree Centrality

Number of connections

Closeness centrality

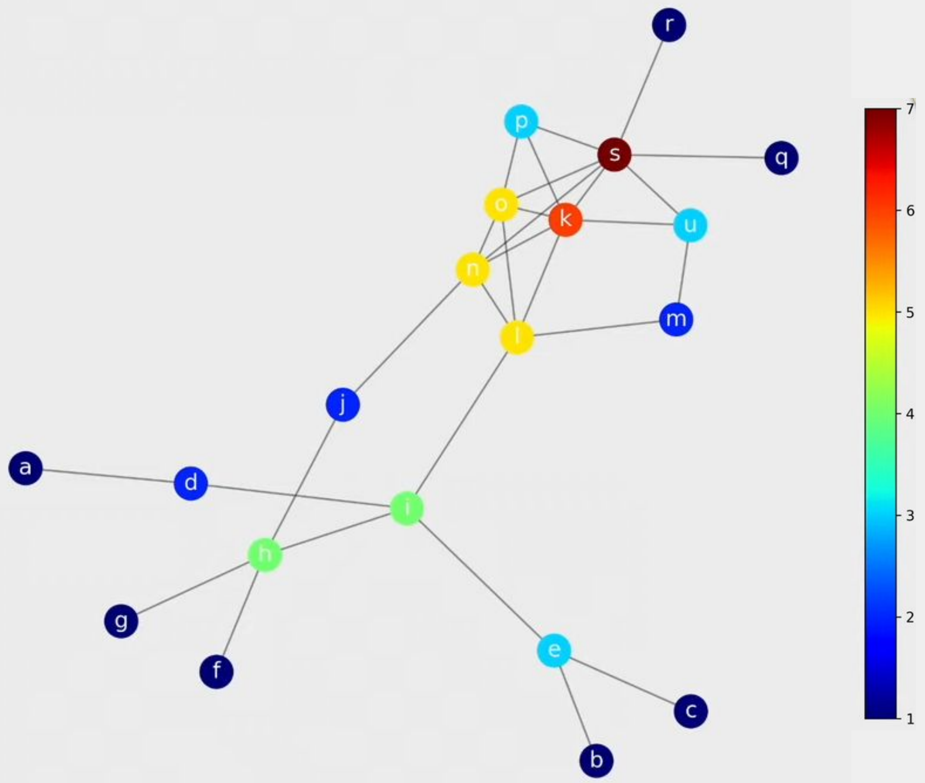
Average distance to all other vertices

Betweenness Centrality

Position on the shortest path

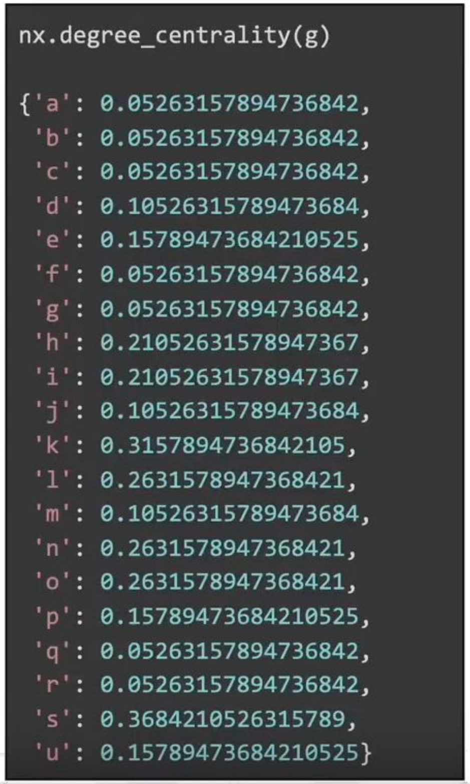
Eigenvector Centrality

Authority score based on the score of the neighbors

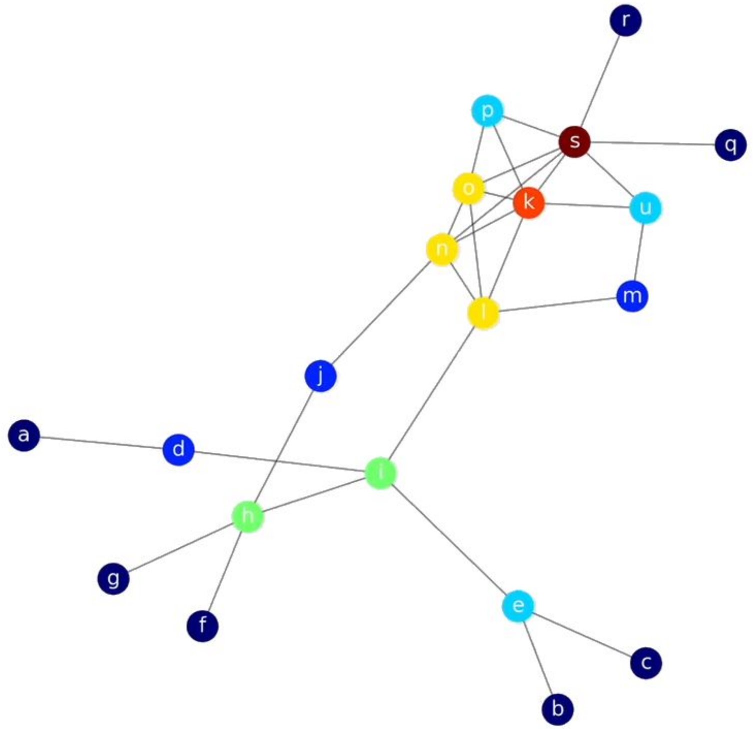


Degree Centrality

Number of connections



```
nx.degree_centrality(g)

{'a': 0.05263157894736842,
 'b': 0.05263157894736842,
 'c': 0.05263157894736842,
 'd': 0.10526315789473684,
 'e': 0.15789473684210525,
 'f': 0.05263157894736842,
 'g': 0.05263157894736842,
 'h': 0.21052631578947367,
 'i': 0.21052631578947367,
 'j': 0.10526315789473684,
 'k': 0.3157894736842105,
 'l': 0.2631578947368421,
 'm': 0.10526315789473684,
 'n': 0.2631578947368421,
 'o': 0.2631578947368421,
 'p': 0.15789473684210525,
 'q': 0.05263157894736842,
 'r': 0.05263157894736842,
 's': 0.3684210526315789,
 'u': 0.15789473684210525}
```



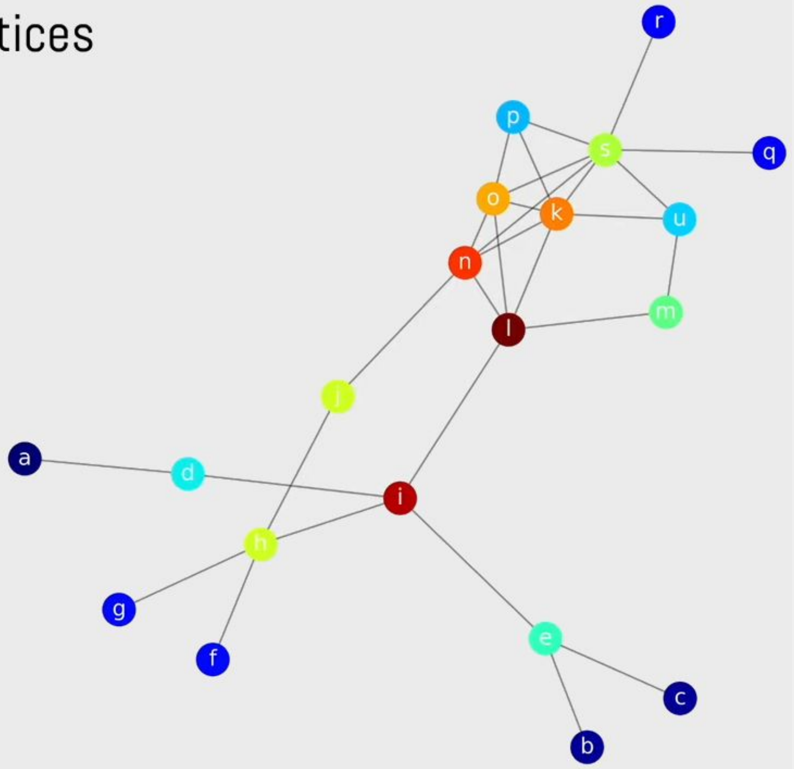
$$degree\_centrality(g) = \frac{\langle k \rangle}{N-1}$$

Degree of the node /
Number of all other nodes.

Closeness Centrality

Average distance to all other vertices

Another way to measure the centrality of a node is by determining how "close" it is to the other nodes. This can be done by summing the distances from the node to all others.

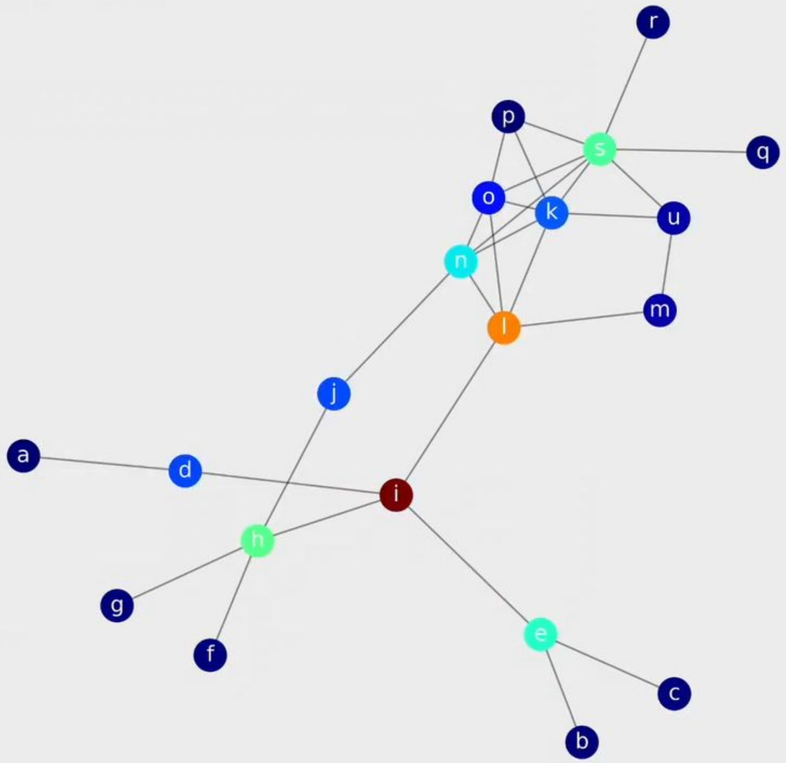


$$closeness\_centrality(g, i) = \frac{N-1}{\sum_{j \neq i} l_{ij}}$$

```
nx.closeness_centrality(g)
```

Betweenness Centrality

Position on the shortest path



Many phenomena taking place in networks are based on diffusion processes. Examples include the transmission of information across a social network, the traffic of goods through a port, and the spread of epidemics in the network of physical contacts between the individuals of a population. This has suggested a third notion of centrality, called **betweenness**: a node is the more central, the more often it is involved in these processes.

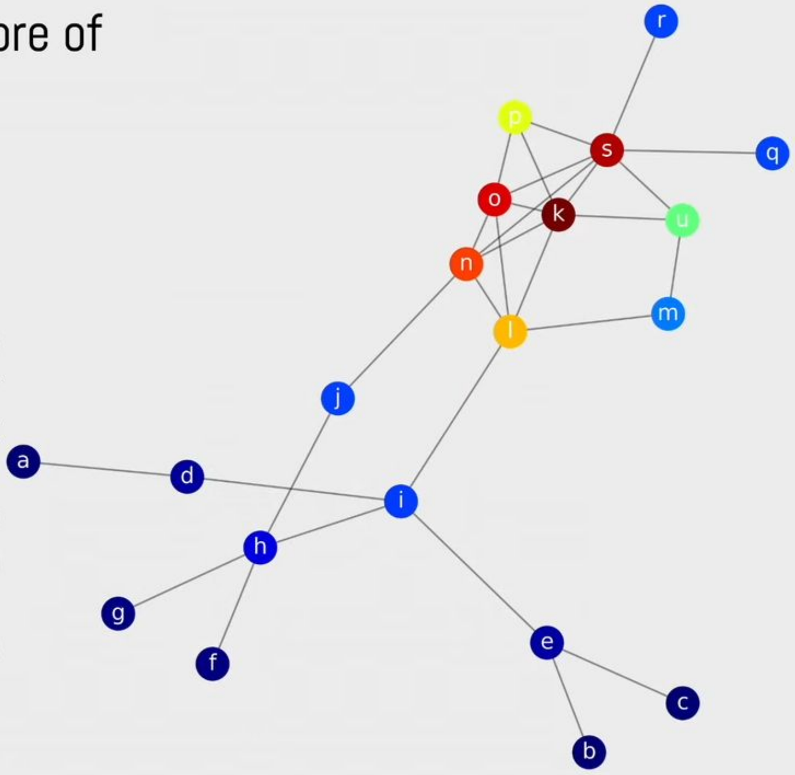
$$betweenness\_centrality(g,i) = \sum_{h \neq j \neq i} \frac{\sigma_{hj}(i)}{\sigma_{hj}}$$

```
nx.betweenness_centrality(g)
```

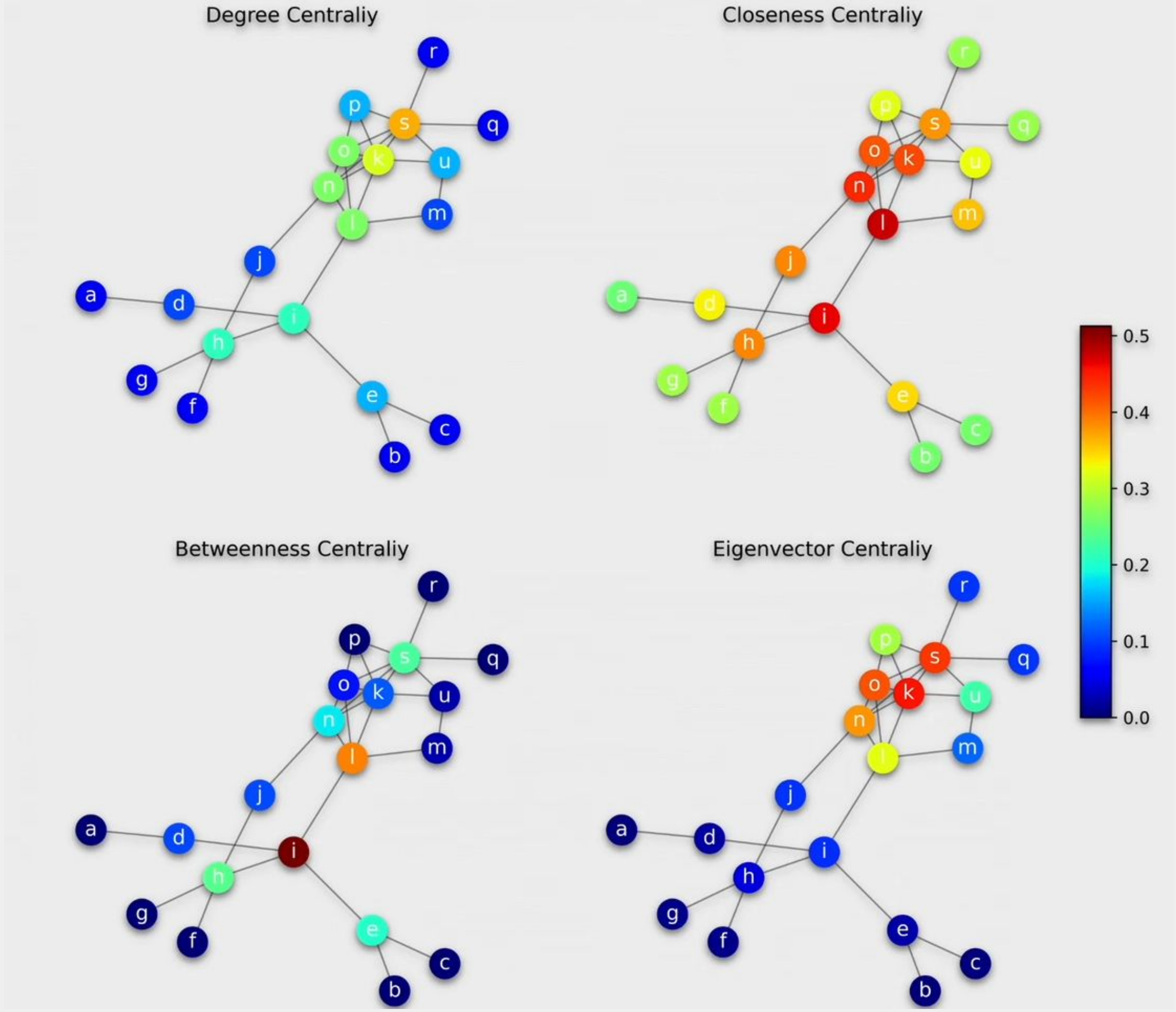
Eigenvector Centrality

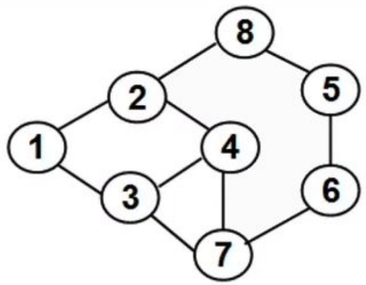
Authority score based on the score of the neighbors

In many circumstances a node's importance in a network is increased by having connections to other nodes that are themselves important. For instance, you might have only one friend in the world, but if that friend is the CR7 then you yourself may be
an important person. Thus centrality is not only about how many people you know but also who you know.



```
nx.eigenvector_centrality(g)
```


Degree Centraliy
Closeness Centraliy
Betweenness Centraliy
Eigenvector Centraliy
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
u
0.5
0.4
0.3
0.2
0.1
0.0



$$\begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 3 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 7 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 8 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{array} \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 3 \\ 3 \\ 2 \\ 2 \\ 3 \\ 2 \end{bmatrix} \equiv \begin{bmatrix} 0.277 \\ 0.416 \\ 0.416 \\ 0.416 \\ 0.277 \\ 0.277 \\ 0.416 \\ 0.277 \end{bmatrix} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}$$

Iteration 1 Normalized Value = 7.21

$$\begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 3 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 7 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 8 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{array} \times \begin{bmatrix} 0.277 \\ 0.416 \\ 0.416 \\ 0.416 \\ 0.277 \\ 0.277 \\ 0.416 \\ 0.277 \end{bmatrix} = \begin{bmatrix} 0.832 \\ 0.971 \\ 1.109 \\ 1.248 \\ 0.555 \\ 0.693 \\ 1.109 \\ 0.693 \end{bmatrix} \equiv \begin{bmatrix} 0.316 \\ 0.369 \\ 0.422 \\ 0.474 \\ 0.211 \\ 0.264 \\ 0.422 \\ 0.264 \end{bmatrix} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}$$

Iteration 2 Normalized Value = 2.63

$$\begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 3 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 7 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 8 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{array} \times \begin{bmatrix} 0.316 \\ 0.369 \\ 0.422 \\ 0.474 \\ 0.211 \\ 0.264 \\ 0.422 \\ 0.264 \end{bmatrix} = \begin{bmatrix} 0.791 \\ 1.054 \\ 1.212 \\ 1.212 \\ 0.527 \\ 0.632 \\ 1.159 \\ 0.579 \end{bmatrix} \equiv \begin{bmatrix} 0.298 \\ 0.397 \\ 0.457 \\ 0.457 \\ 0.198 \\ 0.238 \\ 0.437 \\ 0.219 \end{bmatrix} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}$$

Iteration 3 Normalized Value = 2.65

$$\begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 3 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 7 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 8 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{array} \times \begin{bmatrix} 0.298 \\ 0.397 \\ 0.457 \\ 0.457 \\ 0.198 \\ 0.238 \\ 0.437 \\ 0.219 \end{bmatrix} = \begin{bmatrix} 0.855 \\ 0.974 \\ 1.192 \\ 1.292 \\ 0.457 \\ 0.636 \\ 1.153 \\ 0.596 \end{bmatrix} \equiv \begin{bmatrix} 0.321 \\ 0.366 \\ 0.449 \\ 0.486 \\ 0.172 \\ 0.239 \\ 0.434 \\ 0.224 \end{bmatrix} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}$$

Iteration 4 Normalized Value = 2.66

$$\begin{array}{c|cccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 3 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 7 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 8 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{array} \times \begin{bmatrix} 0.321 \\ 0.366 \\ 0.449 \\ 0.486 \\ 0.172 \\ 0.239 \\ 0.434 \\ 0.224 \end{bmatrix} = \begin{bmatrix} 0.815 \\ 1.032 \\ 1.241 \\ 1.248 \\ 0.434 \\ 0.606 \\ 1.174 \\ 0.538 \end{bmatrix} \equiv \begin{bmatrix} 0.306 \\ 0.388 \\ 0.467 \\ 0.469 \\ 0.174 \\ 0.228 \\ 0.441 \\ 0.202 \end{bmatrix} \begin{array}{c} \text{Vertex ID} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}$$

Iteration 5 Normalized Value = 2.66 **Eigenvector Centrality**

Referências

- Network Science by Albert-László Barabási
  http://networksciencebook.com/

- Network Analysis Course - Prof. Ivanovitch (DCA/UFRN) YouTube Playlist

Exercícios

- https://rosalind.info/problems/topics/graph-algorithms/
- https://rosalind.info/problems/topics/graphs/