Análise de Redes

Slide 05 - Walks, Paths, Distances

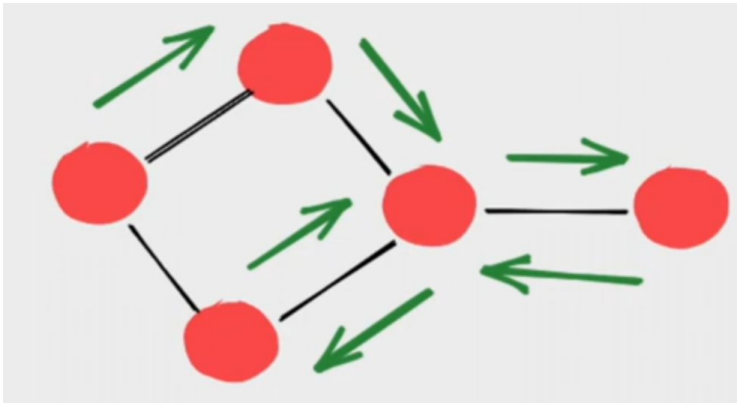
Prof. Patrick Terrematte

Análise de Redes

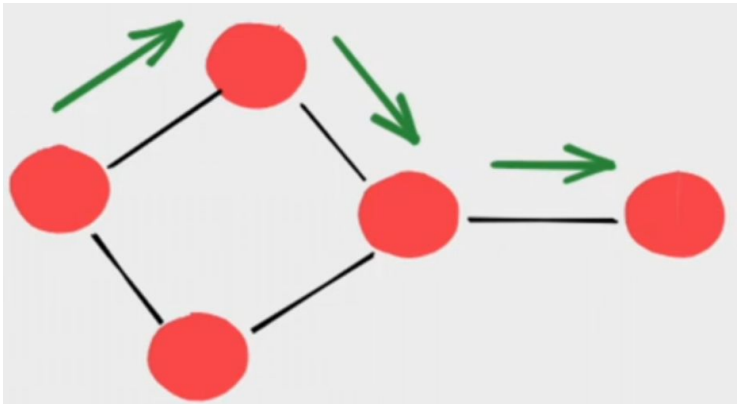
- Caminhada / Walk
    - Sequência de vértices consecutivos conectados por arestas.

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

Walk x Path

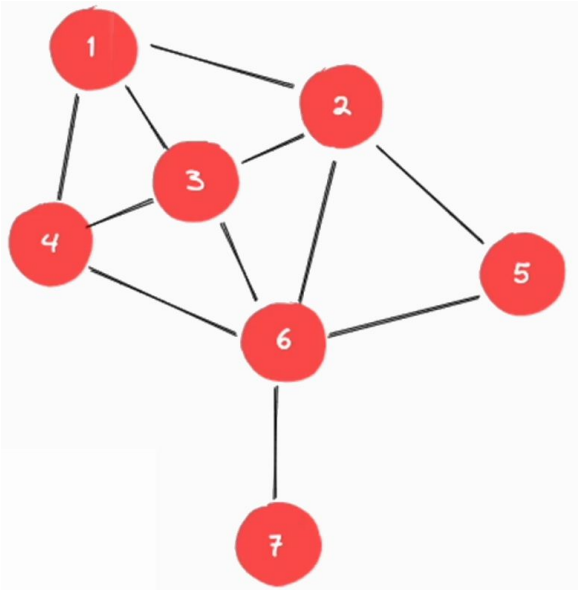


Caminhada de comprimento 6



Caminho de comprimento 3

Walk



Adjacent Matrix (A)

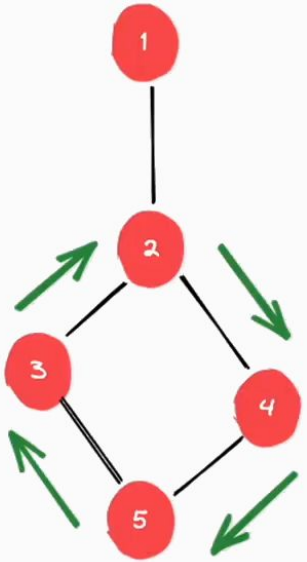
0	1	1	1	0	0	0
1	0	1	0	1	1	0
1	1	0	1	0	1	0
1	0	1	0	0	1	0
0	1	0	0	0	1	0
0	1	1	1	1	0	1
0	0	0	0	0	1	0

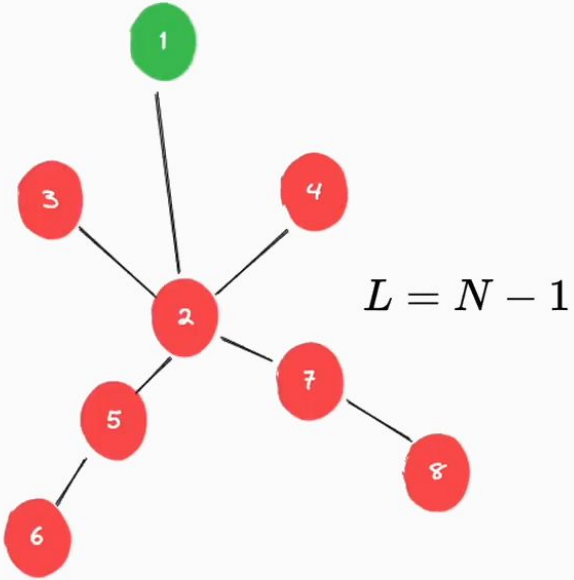
A^2

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





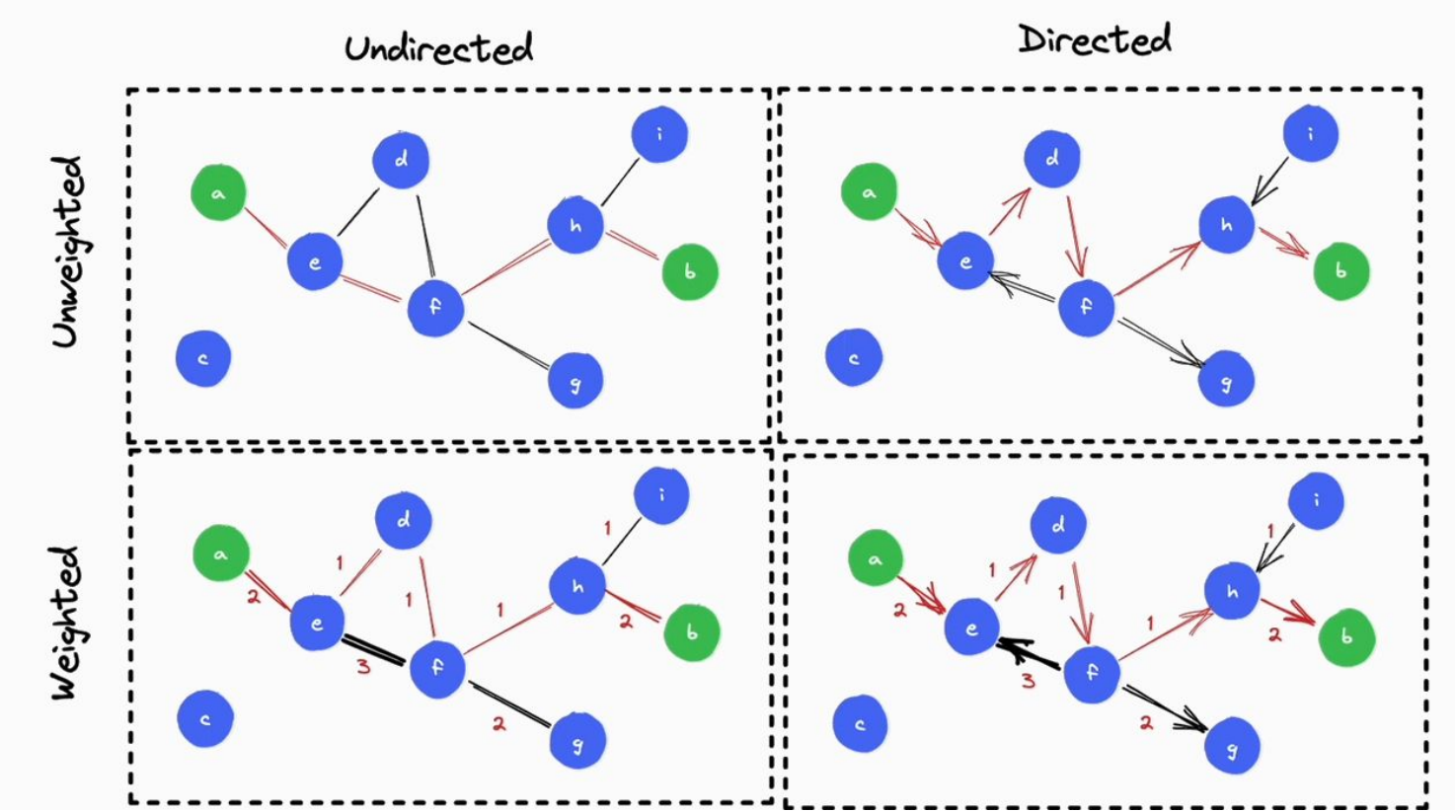
$$L = N - 1$$

```
# 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
```

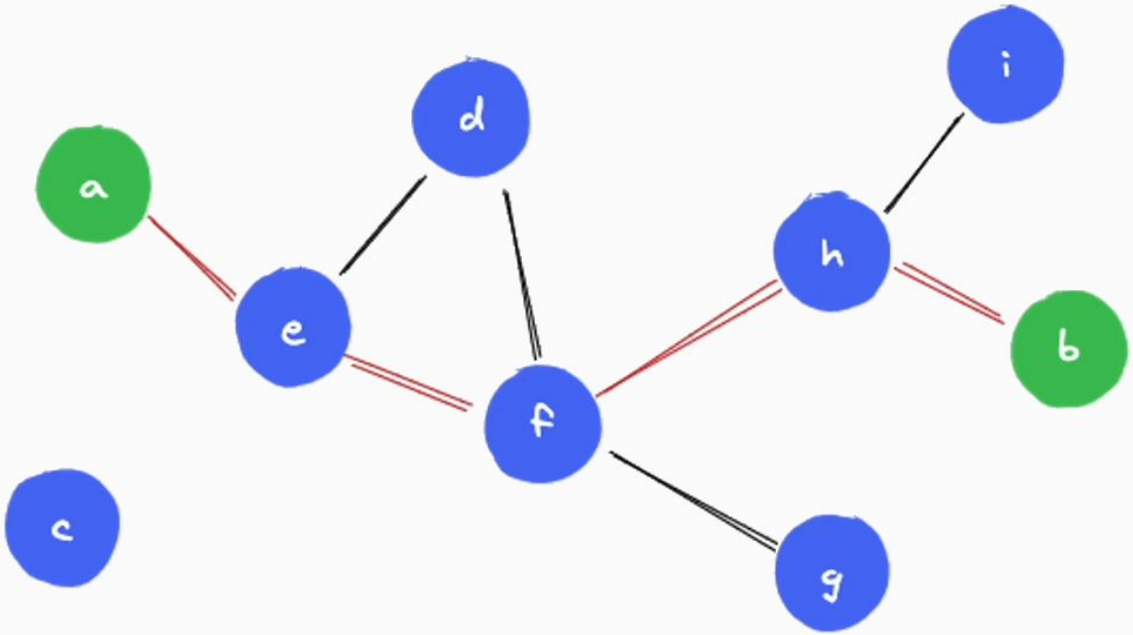
NetworkX
Network Analysis in Python

Caminho mais curto



Caminho mais curto médio

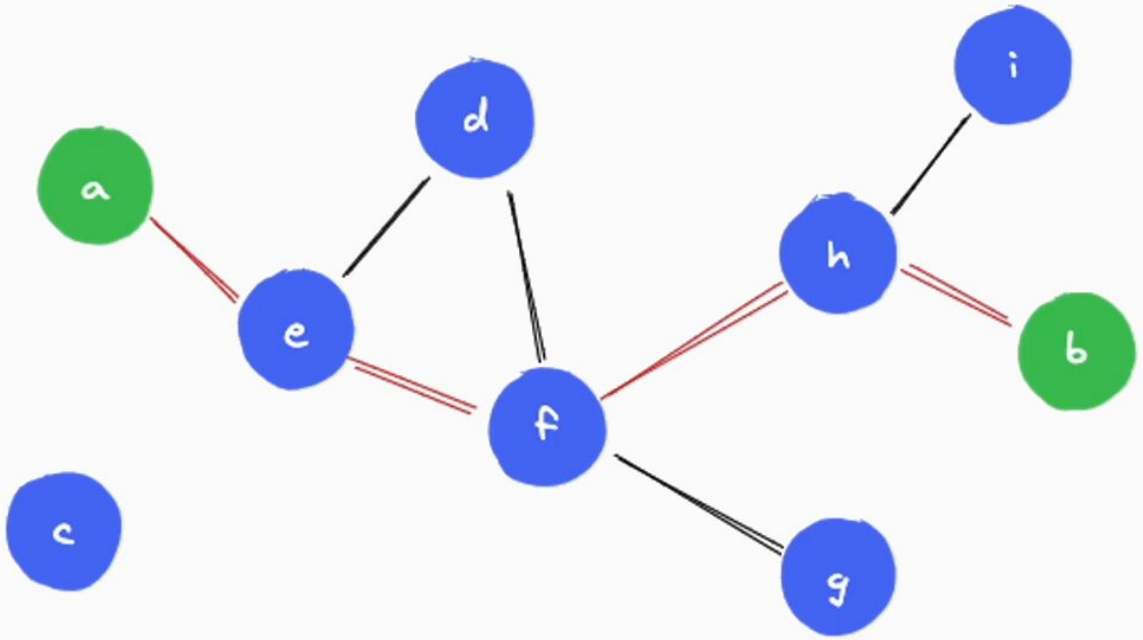
$$\langle l \rangle = \frac{\sum_{ij} l_{ij}}{\binom{N}{2}} = \frac{2\sum_{ij} l_{ij}}{N(N-1)}$$



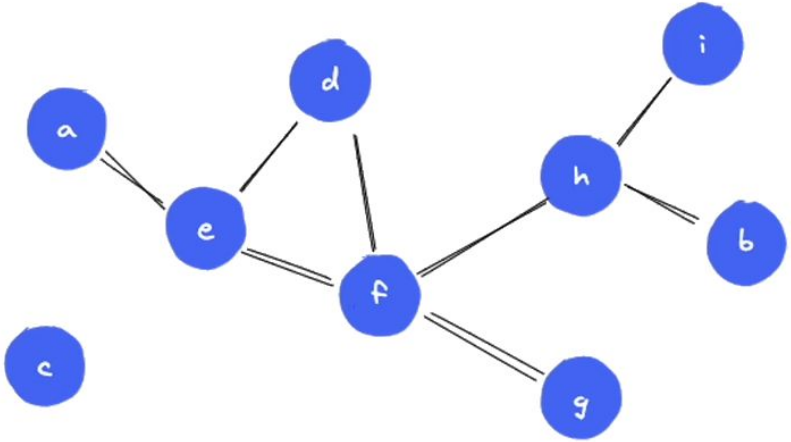
Diametro da rede

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

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

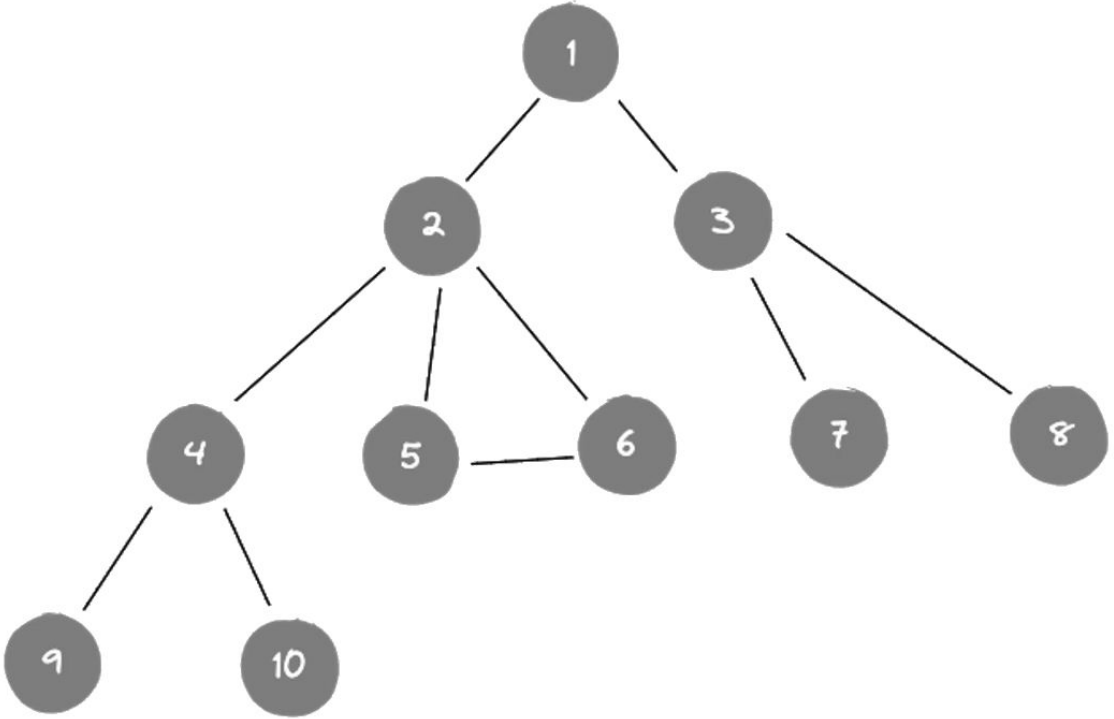


```
# G is connected or not?
nx.is_connected(G)
False

# 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

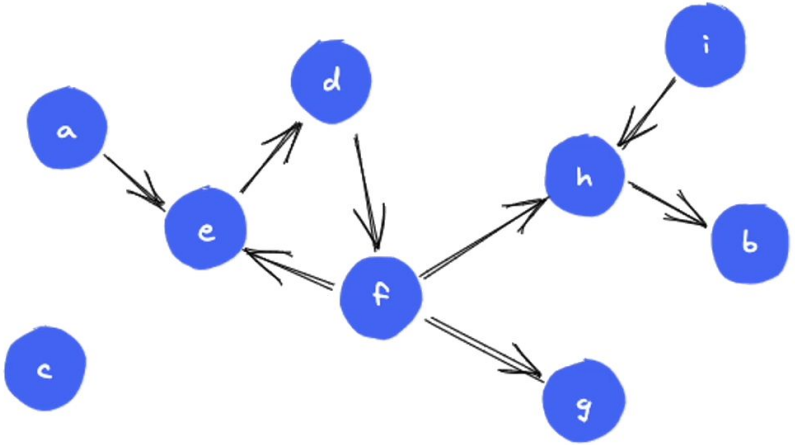
Breadth First Search (BFS) versus Depth First Search (DFS)



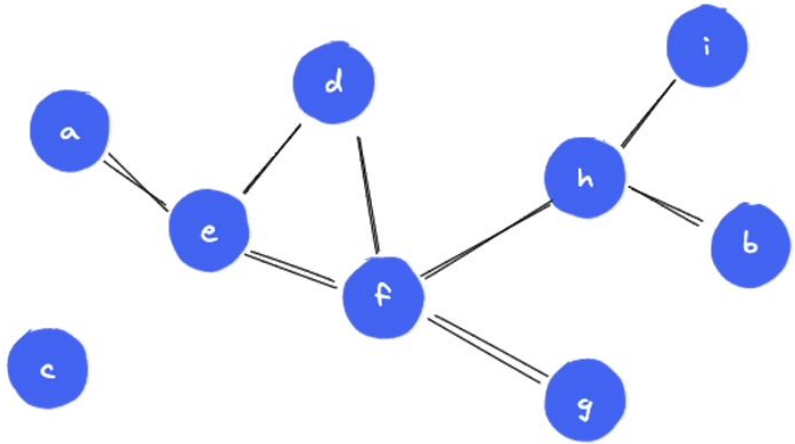
Key	Breadth First Search (BFS)	Depth First Search (DFS)
Definition	BFS stands for Breadth First Search.	DFS stands for Depth First Search.
Data structure	Uses a Queue to find shortest path. FIFO (First In First Out)	Uses a Stack to find the shortest path. LIFO (Last In First Out)
Source	BFS is better 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.
Tapping in loops	In BFS, there is no problem of trapping into finite loops.	In DFS, we may be trapped into infinite loops.

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'}
```

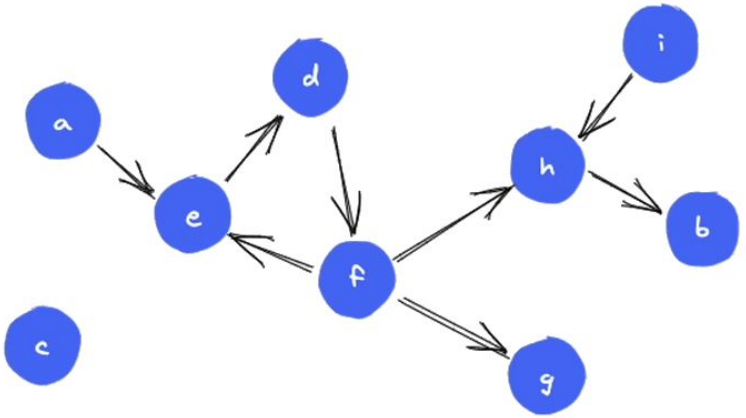


NetworkX
Network Analysis in Python

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

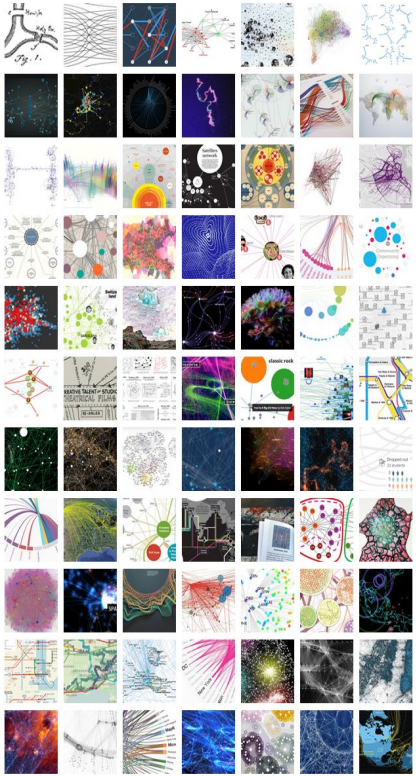
Um componente é fortemente conectado, quando em um subgrafo todos os nós se acessam.

```
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
```



NetworkX
Network Analysis in Python

Análise de Redes - Colab



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