😎 Graficar redes con NetworkX

Recordemos:

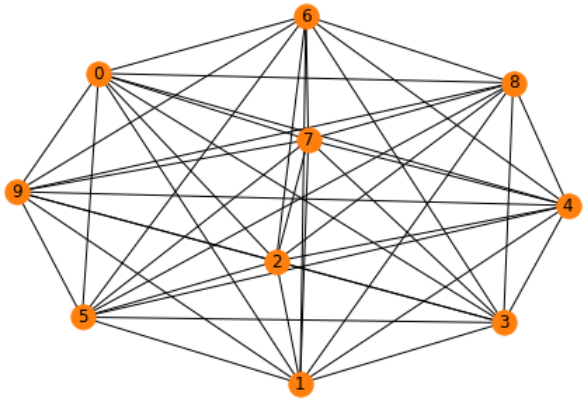
- **Bipartita: Una red es bipartita cuando contiene dos tipos de nodos distintos y todos los bordes conectan un nodo del primer tipo con un nodo del segundo tipo.**
- **Dirigida: Una red está dirigida cuando cada borde tiene una orientación, es decir, cada borde va explícitamente de un nodo a otro.**
- **Marcas de tiempo: Cuando una red tiene marcas de tiempo, se conoce el tiempo de creación de cada borde.**
- **No dirigida: Una red no está dirigida cuando sus bordes no tienen una orientación.**
- **Unipartita: Una red es unipartita cuando contiene un solo tipo de nodo.**
- **Ponderada: Una red se pondera si sus bordes están etiquetados con pesos de borde, por ejemplo, valores de clasificación.**

In [1]:
```
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from random import sample
```

Generar gráficos aleatorios

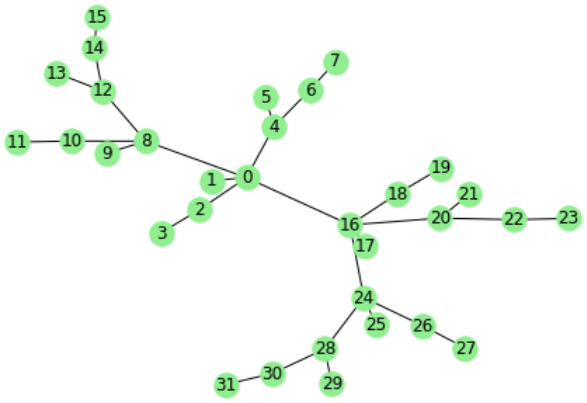
In [2]:
```
graph = nx.complete_graph(10)
print(nx.info(graph))
nx.draw(graph, node_color='C1',with_labels=True)
plt.show()
```

```
Name:
Type: Graph
Number of nodes: 10
Number of edges: 45
Average degree:   9.0000
```



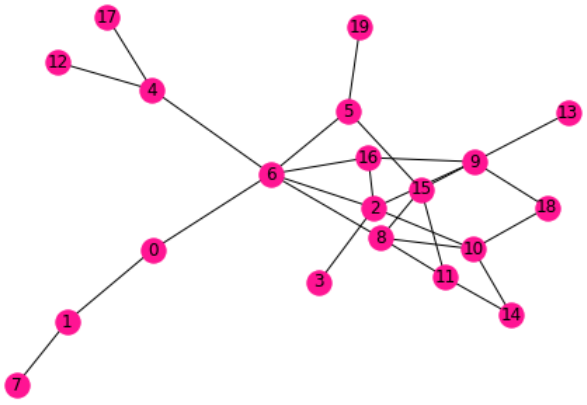
El siguiente ejemplo devuelve un árbol binomial de orden n

In [3]:
```
graph = nx.complete_graph(10)
graph = nx.binomial_tree(5)
nx.draw(graph, node_color='lightgreen', with_labels=True)
plt.show()
```



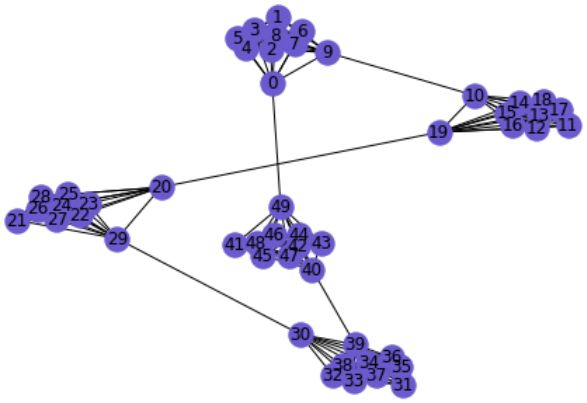
El siguiente ejemplo devuelve un gráfico aleatorio, también conocido como gráfico Erdős-Rényi o gráfico binom
ial.

In [4]:
```
graph = nx.binomial_graph(20,0.15)
nx.draw(graph, node_color='#FF1493', with_labels=True)
plt.show()
```



El siguiente ejemplo devuelve un gráfico conectado de grupos de tamaño k.

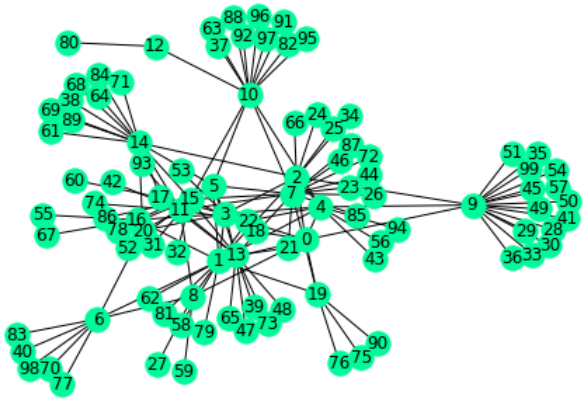
In [5]:
```
graph = nx.connected_caveman_graph(5,10)
nx.draw(graph, node_color='#6A5ACD', with_labels=True)
plt.show()
```



El siguiente ejemplo genera un gráfico aleatorio no dirigido que se asemeja a la red de Internet.

In [6]:

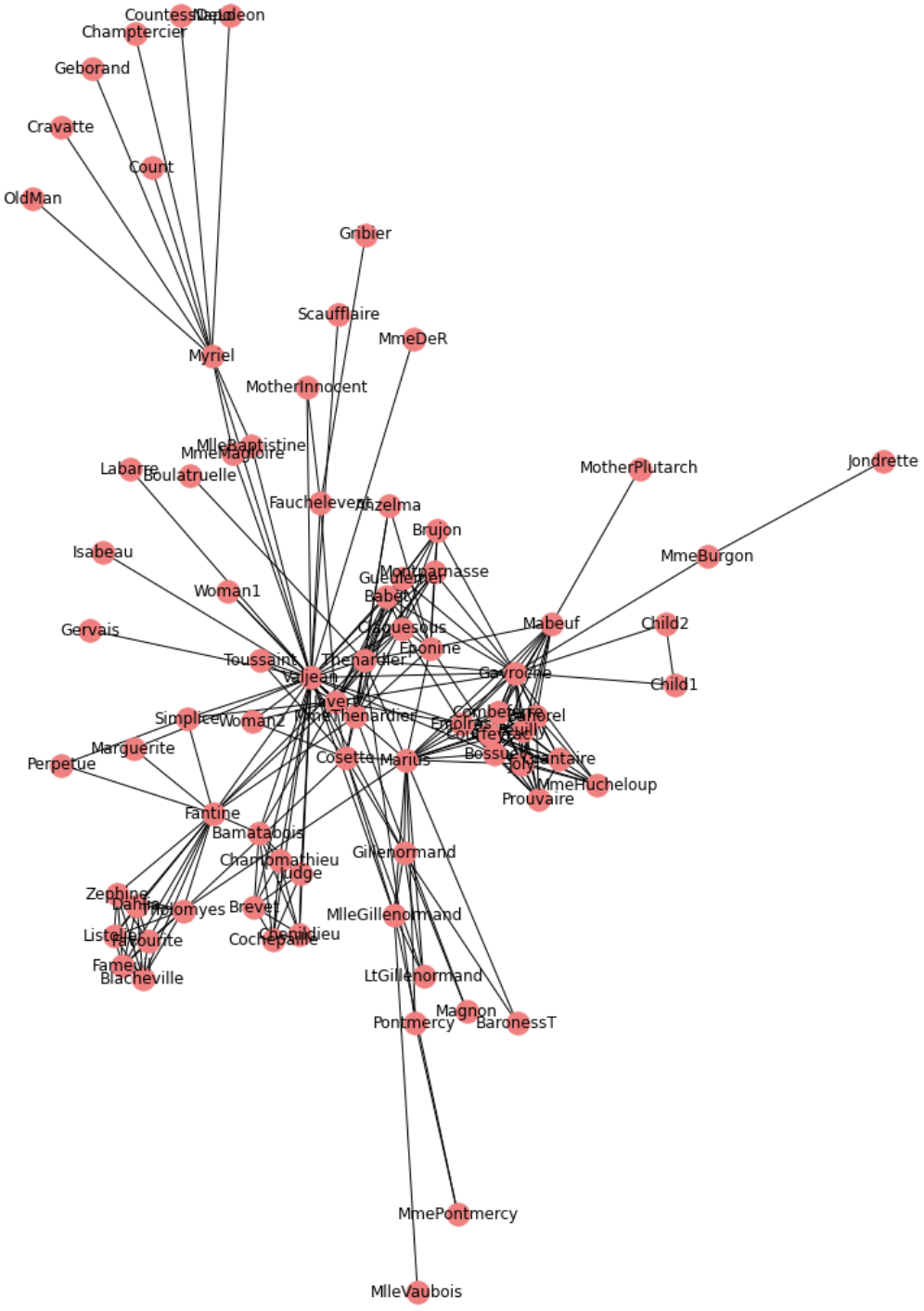
```
graph = nx.random_internet_as_graph(100)
nx.draw(graph, node_color='#00FA9A', with_labels=True)
plt.show()
```



El siguiente ejemplo devuelve la co-aparición de la red de personajes en la novela Los Miserables.

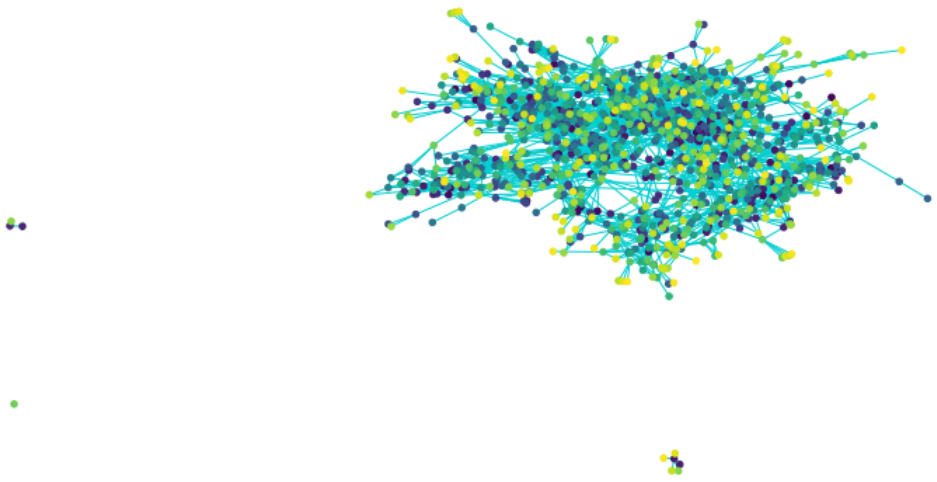
In [7]:

```
plt.figure(figsize=(10,15))
graph = nx.les_miserables_graph()
nx.draw(graph, node_color='#F08080', with_labels=True)
plt.show()
```



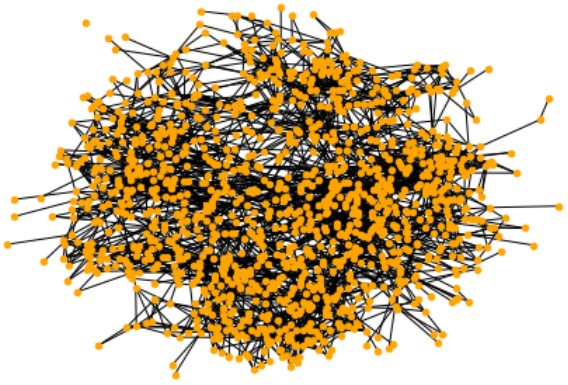
Graficando una red desde datos de un archivo utilizando Pandas

In [8]:
```
xls = pd.ExcelFile('archs/15.Social Network Dataset.xlsx')
network_data = pd.read_excel(xls, sheet_name=['Elements', 'Connections'])
elements_data = network_data['Elements']
connections_data = network_data['Connections']
edge_cols = ['Type', 'Weight', 'When']
graph = nx.convert_matrix.from_pandas_edgelist(connections_data, source='From', target='To',edge_attr=edge_cols)
node_dict = elements_data.set_index('Label').to_dict(orient='index')
nx.set_node_attributes(graph, node_dict)
fig = plt.figure(figsize=(10,5))
colors = np.linspace(0,1,len(graph.nodes))
nx.draw(graph, node_size=20, node_color=colors, edge_color='#00CED1')
fig.set_facecolor('white')
```



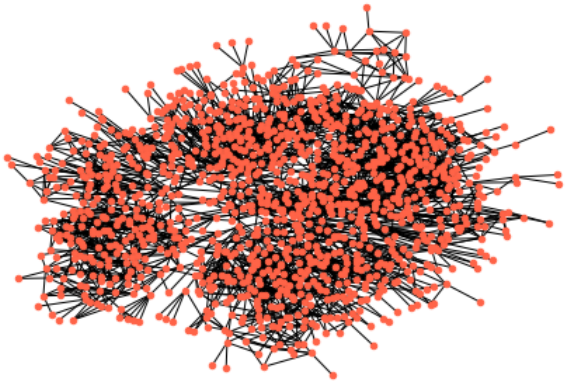
spring_layout: ubica los nodos utilizando el algoritmo dirigido por la fuerza de Fruchterman-Reingold

In [9]:
```
layout = nx.spring_layout(graph,k=0.1)
nx.draw(graph, node_size=20, node_color='#FFA500', pos=layout)
```



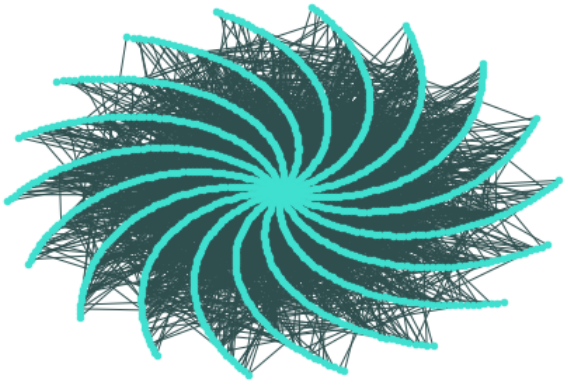
kamada_kawai_layout: ubica los nodos utilizando la función de costo de longitud de ruta Kamada-Kawai

In [10]:
```
layout = nx.kamada_kawai_layout(graph)
nx.draw(graph, node_size=20, node_color='#FF6347', pos=layout)
```



spiral_layout: ubica los nodos en un diseño en espiral.

In [11]:
```
layout = nx.spiral_layout(graph)
nx.draw(graph, node_size=20, node_color='#40E0D0', edge_color='#2F4F4F', pos=layout)
```



degree_centrality calcula la centralidad de grados para los nodos en una red bipartita.

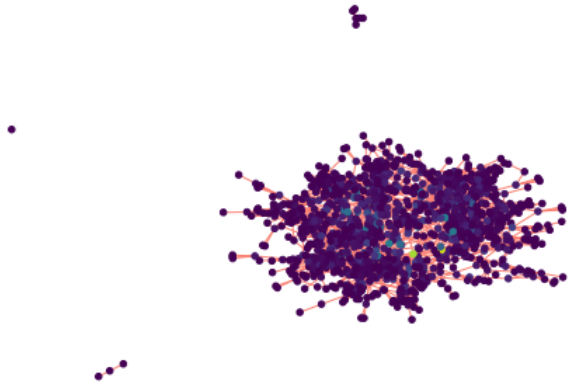
In [12]:
```
centrality = nx.degree_centrality(graph)
colors = list(centrality.values())
nx.draw(graph, node_size=20, node_color=colors, edge_color='#F5DEB3')
```

closeness_centrality calcula la centralidad de proximidad para los nodos en una red bipartita.

In [13]:
```
centrality = nx.closeness_centrality(graph)
colors = list(centrality.values())
nx.draw(graph, node_size=20, node_color=colors, edge_color='#48D1CC')
```

betweenness_centrality calcula la centralidad de intermediación para los nodos en una red bipartita.

In [14]:
```
centrality = nx.betweenness_centrality(graph)
colors = list(centrality.values())
nx.draw(graph, node_size=20, node_color=colors, edge_color='salmon')
```



katz_centrality calcula la centralidad de Katz para los nodos del gráfico.

In [15]:
```
centrality = nx.katz_centrality(graph)
colors = list(centrality.values())
nx.draw(graph, node_size=20, node_color=colors, edge_color='salmon')
```

Crear y visualizar subgrafos

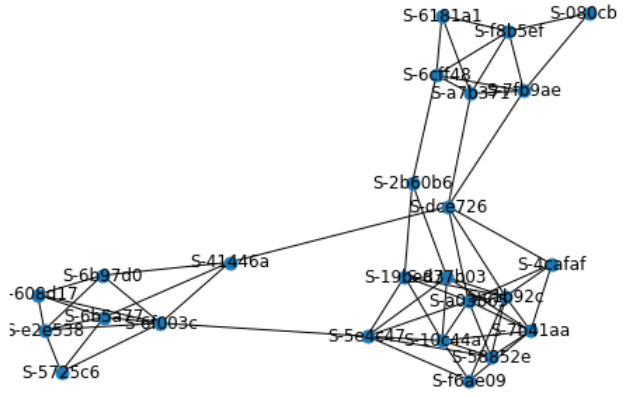
subgrafo de Social Network

In [16]:
```
len(graph.nodes)
len(graph.edges)
node = sample(graph.nodes,1)[0]
print(node)
graph.nodes[node]
sampled_nodes =sample(graph.nodes, 100)
print(sampled_nodes)
subgraph = graph.subgraph(sampled_nodes)
nx.draw(subgraph, node_size=10, with_labels=False, node_color='#FF69B4', edge_color='#0DC6F4')
```

```
S-29a11a
['S-05c79f', 'S-4bf0d6', 'S-0f9a49', 'S-dcdf4c', 'S-5e9fe0', 'S-604fa0', 'S-231baf', 'S-89d761', 'S-f88274', 'S-571e0e', 'S-8513a4', 'S-e8c3ba', 'S-51d629', 'S-7e13cc', 'S-088fc7', 'S-2c3a5b', 'S-3f54e7', 'S-f88332', 'S-f6671e', 'S-a8a1ca', 'S-d98160', 'S-4e3f28', 'S-4f5a03', 'S-c5a9f1', 'S-11a9e4', 'S-8fb145', 'S-f588c9', 'S-1ddd10', 'S-0b9e56', 'S-2b2261', 'S-f8c108', 'S-dfdcf9', 'S-e4978f', 'S-7333d0', 'S-44ada3', 'S-aa39ce', 'S-3586c8', 'S-d89b94', 'S-31ddaf', 'S-cdccc4', 'S-c92441', 'S-e93094', 'S-5e5aa0', 'S-b1d301', 'S-654088', 'S-f82119', 'S-1b0fab', 'S-d11a5e', 'S-adb7e5', 'S-16e928', 'S-5df058', 'S-cd07c0', 'S-ece748', 'S-178b40', 'S-34d7a8', 'S-90b463', 'S-1e878c', 'S-d6b191', 'S-3132c1', 'S-c1bdf7', 'S-2bbba0', 'S-6bbdd3', 'S-359c63', 'S-613bfc', 'S-df2ac4', 'S-308c4e', 'S-2732bb', 'S-8a6ffb', 'S-4ab39a', 'S-4a8988', 'S-dfc912', 'S-34f712', 'S-e25b3a', 'S-a55782', 'S-166930', 'S-7eb72a', 'S-120d4a', 'S-a47614', 'S-aacac0', 'S-e6fd5e', 'S-8acd96', 'S-a4b713', 'S-a2e84d', 'S-2d2310', 'S-a695ad', 'S-631ad2', 'S-2eb417', 'S-3168b8', 'S-2895f5', 'S-70ec7f', 'S-3ef8bb', 'S-80b447', 'S-1b7dba', 'S-d77b03', 'S-389483', 'S-dc649c', 'S-d5f474', 'S-00ecfd', 'S-a8555b', 'S-756549']
```

In [17]:

```
from collections import defaultdict

nodes_school_id = nx.get_node_attributes(graph,'School (ID)')
school_nodes = defaultdict(list)
for node, school_id in nodes_school_id.items():
    school_nodes[school_id].append(node)

school_nodes[5]
graph.nodes['S-087f53']
subgraphs = {}

for school_id, nodes in school_nodes.items():
    subgraph = graph.subgraph(nodes)
    subgraphs[school_id] = subgraph

subgraphs[5].nodes

nx.draw(subgraphs[3],
        node_size=80,
        with_labels=True)
plt.figure(figsize=(15,10))
plt.show()
```

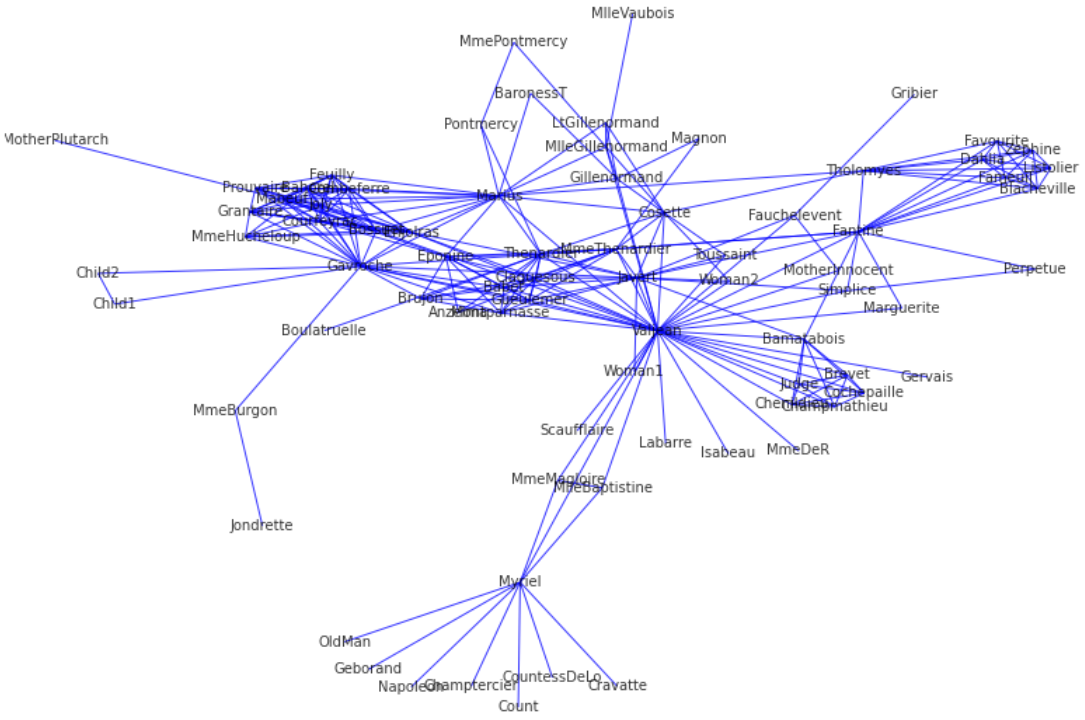


<Figure size 1080x720 with 0 Axes>

Subgrafo de Los miserables

In [18]:

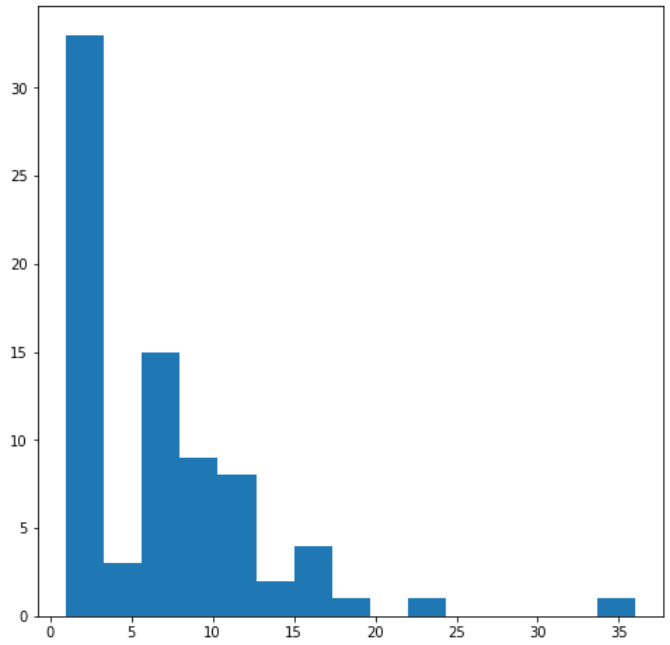
```
G = nx.read_gml('archs/lesmiserables.gml')
plt.figure(figsize=(14,10))
nx.draw_networkx(G,node_size=0,edge_color='b', alpha=.8, font_size=10)
plt.axis('off')
plt.show()
```



El grafo en sí no muestra información útil, entonces para entender más la información del grafo se creará un histograma del número de conexiones por nodo.

In [22]:

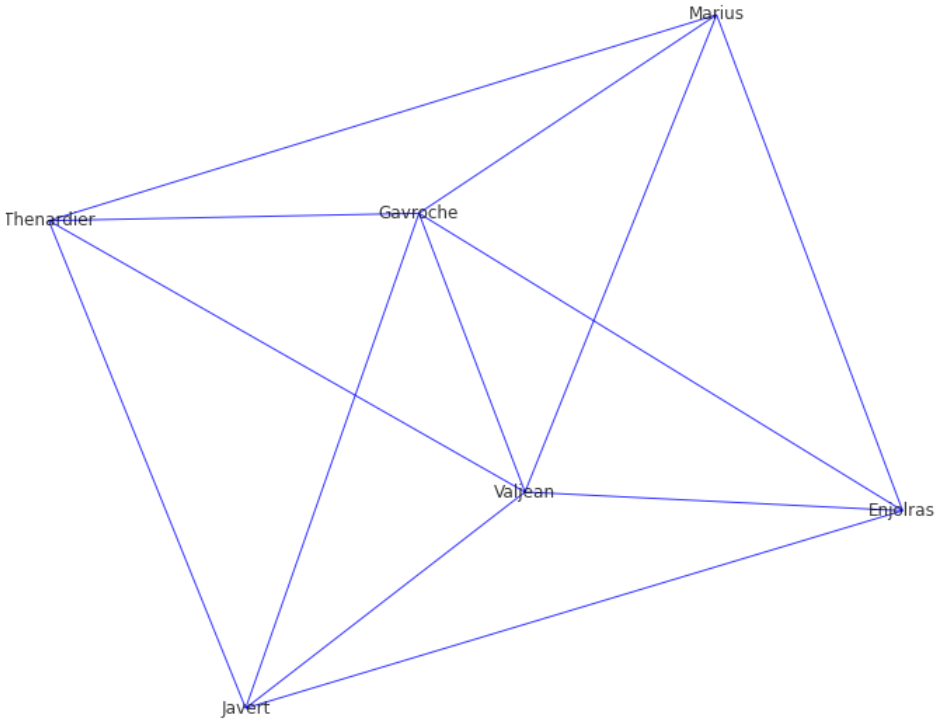
```
plt.figure(figsize=(8,8))
my_degrees = G.degree()
degree_values = [v for k, v in my_degrees]
plt.hist(degree_values,bins=15)
plt.show()
```



Parece que sólo pocos personajes tienen más de 10 conexiones. A continuación se muestra el código de un grafo de sólo
esos personajes con más de 10 conexiones:

In [23]:

```
def trim_nodes(G,d):
    """ retorna una copia de G sin los nodos de menos de d conexiones"""
    Gt = G.copy()
    #Se define la instancia de degree con la copia de G
    dn = nx.degree(Gt)
    #Se recorre los nodos y se remueve los que tengan menos de d conexiones
    for n in Gt.copy():
        if dn[n] <= d:
            Gt.remove_node(n)
    #Se retorna el nuevo G
    return Gt
Gt = trim_nodes(G,10)
plt.figure(figsize=(12,10))
nx.draw_networkx(Gt,node_size=0,edge_color='b', alpha=.8, font_size=12)
plt.axis('off')
plt.show()
```



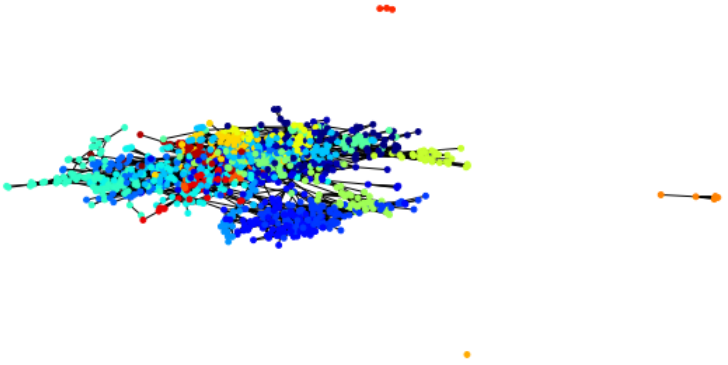
Detección de comunidades en networkx

Es importante que tenga networkx actualizado:

- pip install --upgrade networkx
 y además se debe instalar:
- pip install community
- pip install python-louvain

In [24]:
```
xls = pd.ExcelFile('archs/15.Social Network Dataset.xlsx')
network_data = pd.read_excel(xls, sheet_name=['Elements', 'Connections'])
elements_data = network_data['Elements']
connections_data = network_data['Connections']
edge_cols = ['Type', 'Weight', 'When']
graph = nx.convert_matrix.from_pandas_edgelist(connections_data, source='From', target='To',edge_attr=edge_cols)
node_dict = elements_data.set_index('Label').to_dict(orient='index')
nx.set_node_attributes(graph, node_dict)
```

In [26]:
```
from community import community_louvain

spring_pos = nx.spring_layout(graph)
parts = community_louvain.best_partition(graph)
values = [parts.get(node) for node in graph.nodes()]
fig = plt.figure(figsize=(10,5))
plt.axis("off")
nx.draw_networkx(graph,pos = spring_pos,
                 cmap = plt.get_cmap("jet"),
                 node_color = values,
                 node_size = 15,
                 with_labels = False)
plt.figure(figsize=(15,10))
plt.show()
```



<Figure size 1080x720 with 0 Axes>