Chapter 2: Working in NetworkX

In [ ]:

```
# Configure plotting in Jupyter
from matplotlib import pyplot as plt
%matplotlib inline
plt.rcParams.update({
    'figure.figsize': (7.5, 7.5),
    'axes.spines.right': False,
    'axes.spines.left': False,
    'axes.spines.top': False,
    'axes.spines.bottom': False})
# Seed random number generator
import random
from numpy import random as nprand
seed = hash("Network Science in Python") % 2**32
nprand.seed(seed)
random.seed(seed)
```

In [ ]:

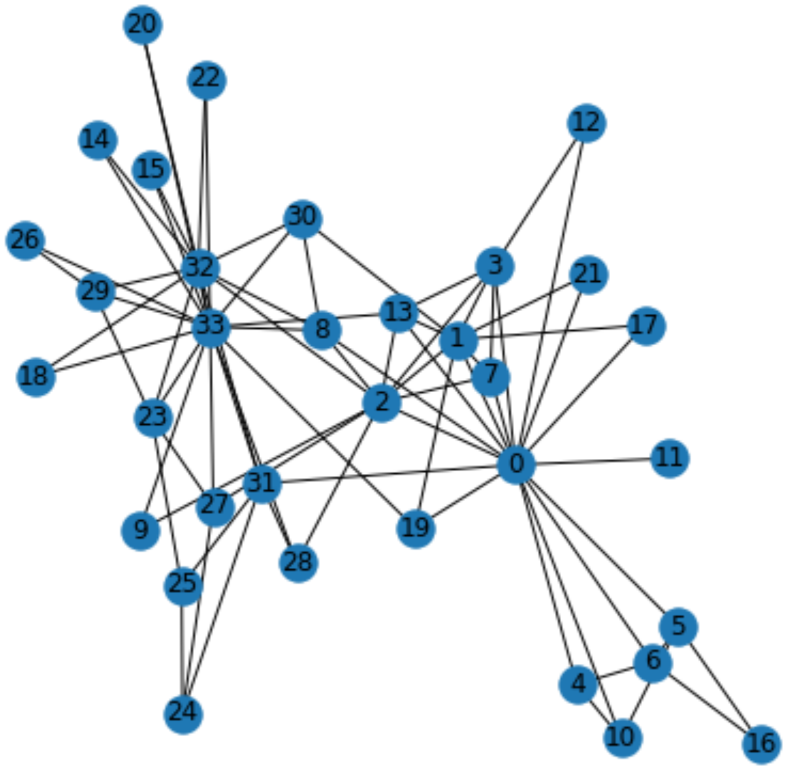
```
# Import networkx
import networkx as nx
```

The Graph Class: Working with undirected networks

For this homework, we are supposed to modify this notebook. This has primarily been done in the section relating to the bridges of Konigsburg.

In [ ]:

```
G = nx.karate_club_graph()
karate_pos = nx.spring_layout(G, k=0.3)
nx.draw_networkx(G, karate_pos)
```



In [ ]:

```
list(G.nodes)
```

Out[ ]:
```
[0,
 1,
 2,
 3,
 4,
 5,
 6,
 7,
 8,
 9,
 10,
 11,
 12,
 13,
 14,
 15,
 16,
 17,
 18,
 19,
 20,
 21,
 22,
 23,
 24,
 25,
 26,
 27,
 28,
 29,
 30,
 31,
 32,
 33]
```

In [ ]:
```
list(G.edges)
```

Out[ ]:
```
[(0, 1),
 (0, 2),
 (0, 3),
 (0, 4),
 (0, 5),
 (0, 6),
 (0, 7),
 (0, 8),
 (0, 10),
 (0, 11),
 (0, 12),
 (0, 13),
 (0, 17),
 (0, 19),
 (0, 21),
 (0, 31),
 (1, 2),
 (1, 3),
 (1, 7),
 (1, 13),
    (1, 17),
    (1, 19),
    (1, 21),
    (1, 30),
    (2, 3),
    (2, 7),
    (2, 8),
    (2, 9),
    (2, 13),
    (2, 27),
    (2, 28),
    (2, 32),
    (3, 7),
    (3, 12),
    (3, 13),
    (4, 6),
    (4, 10),
    (5, 6),
    (5, 10),
    (5, 16),
    (6, 16),
    (8, 30),
    (8, 32),
    (8, 33),
    (9, 33),
    (13, 33),
    (14, 32),
    (14, 33),
    (15, 32),
    (15, 33),
    (18, 32),
    (18, 33),
    (19, 33),
    (20, 32),
    (20, 33),
    (22, 32),
    (22, 33),
    (23, 25),
    (23, 27),
    (23, 29),
    (23, 32),
    (23, 33),
    (24, 25),
    (24, 27),
    (24, 31),
    (25, 31),
    (26, 29),
    (26, 33),
    (27, 33),
    (28, 31),
    (28, 33),
    (29, 32),
    (29, 33),
    (30, 32),
    (30, 33),
    (31, 32),
    (31, 33),
 (32, 33)]
```

Checking for nodes

In [ ]:

```
mr_hi = 0
mr_hi in G
```

Out[ ]:

```
True
```

In [ ]:

```
G.has_node(mr_hi)
```

Out[ ]:

```
True
```

In [ ]:

```
wild_goose = 1337
wild_goose in G
```

Out[ ]:

```
False
```

In [ ]:

```
G.has_node(wild_goose)
```

Out[ ]:

```
False
```

Finding node neighbors

In [ ]:

```
list(G.neighbors(mr_hi))
```

Out[ ]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 17, 19, 21, 31]
```

In [ ]:

```
member_id = 1
(mr_hi, member_id) in G.edges
```

Out[ ]:

```
True
```

In [ ]:

```
G.has_edge(mr_hi, member_id)
```

Out[ ]:

```
True
```

In [ ]:

```
john_a = 33
(mr_hi, john_a) in G.edges
```

Out[ ]:

```
False
```

In [ ]:

```
G.has_edge(mr_hi, john_a)
```

Out[ ]:

```
False
```

Adding attributes to nodes and edges

In [ ]:

```
member_club = [
    0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
    0, 0, 0, 0, 1, 1, 0, 0, 1, 0,
    1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1]
```

In [ ]:

```
for node_id in G.nodes:
    G.nodes[node_id]["club"] = member_club[node_id]
```

In [ ]:

```
G.add_node(11, club=0)
```

In [ ]:

```
G.nodes[mr_hi]
```

Out[ ]:

```
{'club': 0}
```

In [ ]:

```
G.nodes[john_a]
```

Out[ ]:

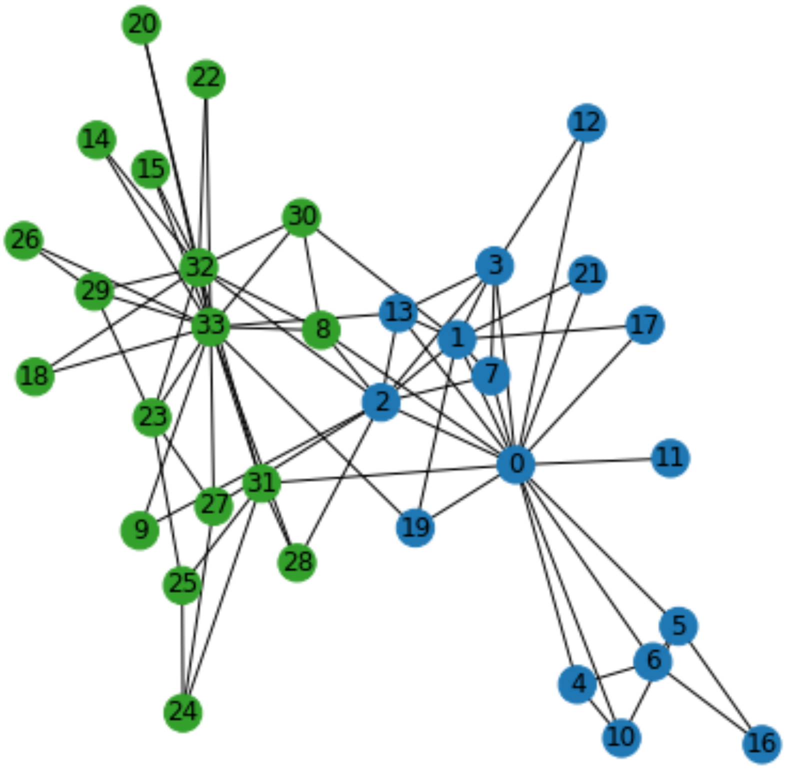
```
{'club': 1}
```

In [ ]:

```
node_color = [
    '#1f78b4' if G.nodes[v]["club"] == 0
    else '#33a02c' for v in G]
```

In [ ]:

```
nx.draw_networkx(G, karate_pos, label=True, node_color=node_color)
```



In [ ]:

```
# Iterate through all edges
for v, w in G.edges:
    # Compare `club` property of edge endpoints
    # Set edge `internal` property to True if they match
    if G.nodes[v]["club"] == G.nodes[w]["club"]:
        G.edges[v, w]["internal"] = True
    else:
        G.edges[v, w]["internal"] = False
```

In [ ]:

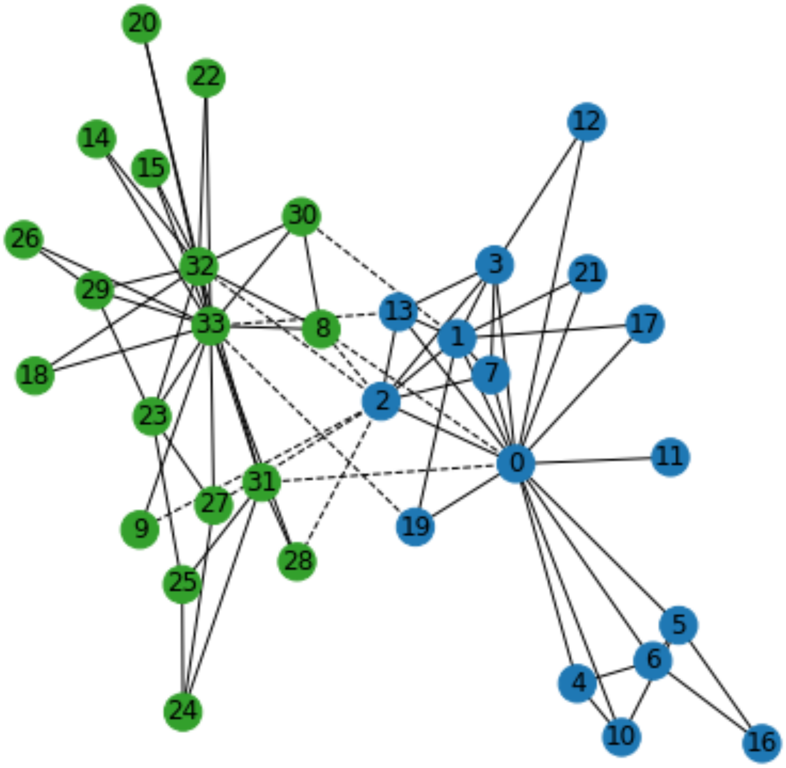
```
internal = [e for e in G.edges if G.edges[e]["internal"]]
external = [e for e in G.edges if not G.edges[e]["internal"]]
```

In [ ]:

```
# Draw nodes and node labels
nx.draw_networkx_nodes(G, karate_pos, node_color=node_color)
nx.draw_networkx_labels(G, karate_pos)
# Draw internal edges as solid lines
nx.draw_networkx_edges(G, karate_pos, edgelist=internal)
# Draw external edges as dashed lines
nx.draw_networkx_edges(G, karate_pos, edgelist=external, style="dashed")
```

Out[ ]:

```
<matplotlib.collections.LineCollection at 0x7f9cb8a43af0>
```



Adding Edge Weights

In [ ]:

```
def tie_strength(G, v, w):
    # Get neighbors of nodes v and w in G
    v_neighbors = set(G.neighbors(v))
    w_neighbors = set(G.neighbors(w))
    # Return size of the set intersection
    return 1 + len(v_neighbors & w_neighbors)
```

In [ ]:

```
# Calculate weight for each edge
for v, w in G.edges:
    G.edges[v, w]["weight"] = tie_strength(G, v, w)
# Store weights in a list
edge_weights = [G.edges[v, w]["weight"] for v, w in G.edges]
```

In [ ]:

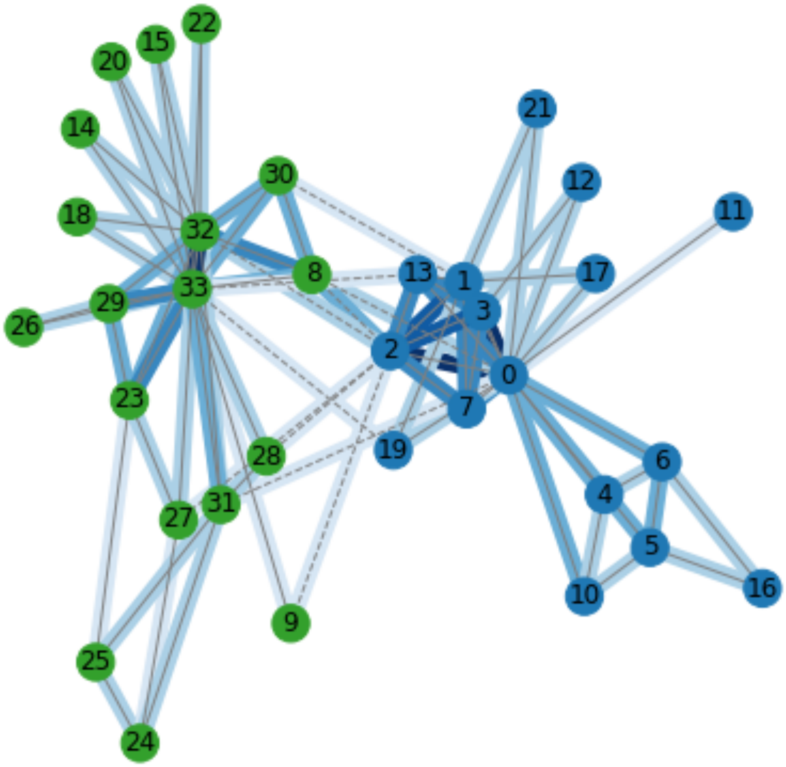
```
weighted_pos = nx.spring_layout(G, pos=karate_pos, k=0.3, weight="weight")
```

In [ ]:

```
# Draw network with edge color determined by weight
nx.draw_networkx(
    G, weighted_pos, width=8, node_color=node_color,
    edge_color=edge_weights, edge_vmin=0, edge_vmax=6, edge_cmap=plt.cm.Blue
# Draw solid/dashed lines on top of internal/external edges
nx.draw_networkx_edges(G, weighted_pos, edgelist=internal, edge_color="gray"
nx.draw_networkx_edges(G, weighted_pos, edgelist=external, edge_color="gray"
```

Out[ ]:

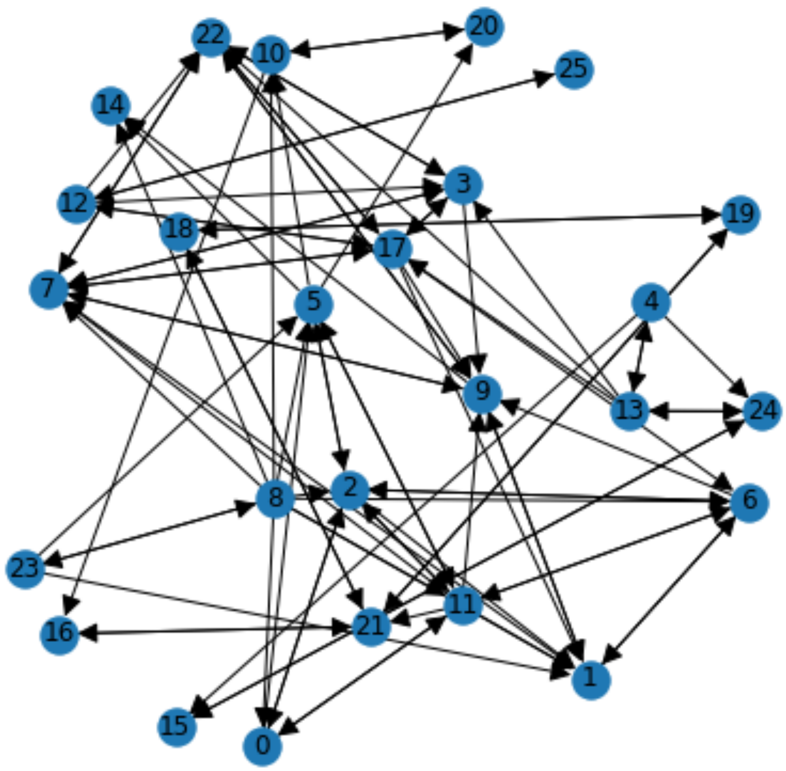
```
<matplotlib.collections.LineCollection at 0x7f9cc92ae8c0>
```



The DiGraph Class: When direction matters

In [ ]:

```
# modified the path such that the code will run
G = nx.read_gexf("/Users/NathanBick/Documents/Graduate School/MATH517 - Soci
student_pos = nx.spring_layout(G, k=1.5)
nx.draw_networkx(G, student_pos, arrowsize=20)
```



In [ ]:

```
list(G.neighbors(0))
```

Out[ ]:

```
[2, 5, 11]
```

In [ ]:

```
list(G.successors(0))
```

Out[ ]:

```
[2, 5, 11]
```

In [ ]:

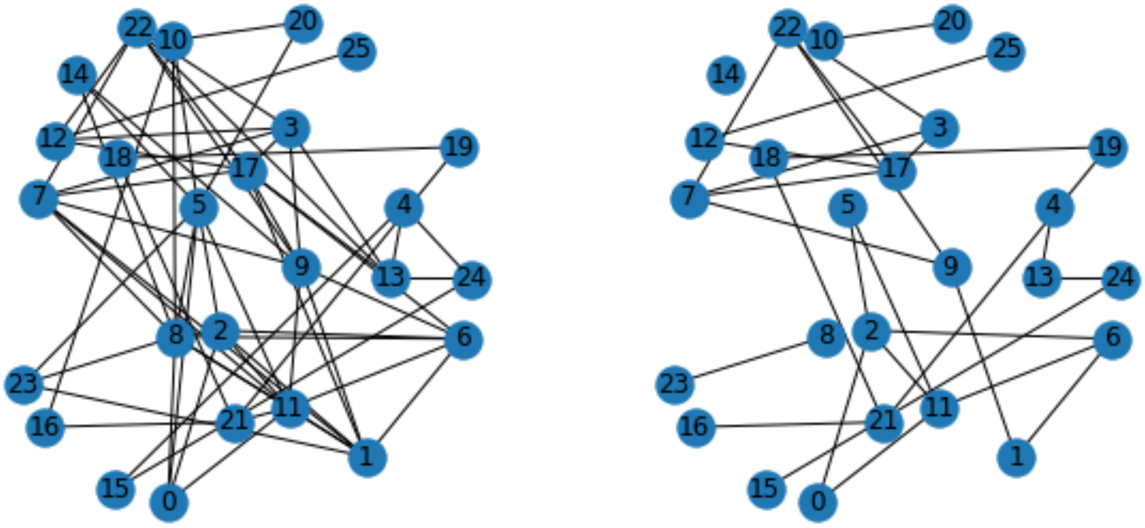
```
list(G.predecessors(0))
```

Out[ ]:

```
[2, 11, 8]
```

In [ ]:

```
# Create undirected copies of G
G_either = G.to_undirected()
G_both = G.to_undirected(reciprocal=True)
# Set up a figure
plt.figure(figsize=(10,5))
# Draw G_either on left
plt.subplot(1, 2, 1)
nx.draw_networkx(G_either, student_pos)
# Draw G_both on right
plt.subplot(1, 2, 2)
nx.draw_networkx(G_both, student_pos)
```



MultiGraph and MultiDiGraph: Parallel edges

In [ ]:

```
# The seven bridges of Königsberg
G = nx.MultiGraph()
G.add_edges_from([
    ("North Bank", "Kneiphof", {"bridge": "Krämerbrücke"}),
    ("North Bank", "Kneiphof", {"bridge": "Schmiedebrücke"}),
    ("North Bank", "Lomse",    {"bridge": "Holzbrücke"}),
    ("Lomse",      "Kneiphof", {"bridge": "Dombrücke"}),
    ("South Bank", "Kneiphof", {"bridge": "Grüne Brücke"}),
    ("South Bank", "Kneiphof", {"bridge": "Köttelbrücke"}),
    ("South Bank", "Lomse",    {"bridge": "Hohe Brücke"})
])
```

Out[ ]:

```
[0, 1, 0, 0, 0, 1, 0]
```

In [ ]:

```
print(G)
```

```
MultiGraph with 4 nodes and 7 edges
```

In [ ]:

```
list(G.edges)[0]
```

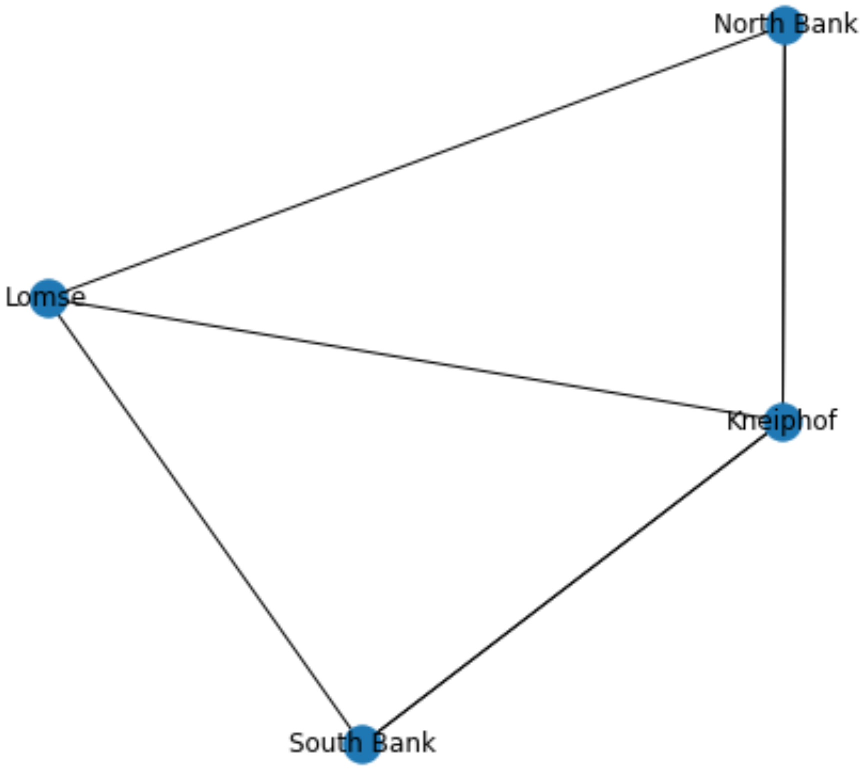
Out[ ]:

```
('North Bank', 'Kneiphof', 0)
```

```
In [ ]: print(G.edges['North Bank', 'Kneiphof',0])
        print(G.edges['North Bank', 'Kneiphof',1])
```

```
{'bridge': 'Krämerbrücke'}
{'bridge': 'Schmiedebrücke'}
```

```
In [ ]: nx.draw_networkx(G, with_labels=True, connectionstyle='arc3, rad = 0.1')
```


North Bank
Lomse
Kneiphof
South Bank