Untitled2

jupyter.org/try-jupyter/retro/notebooks

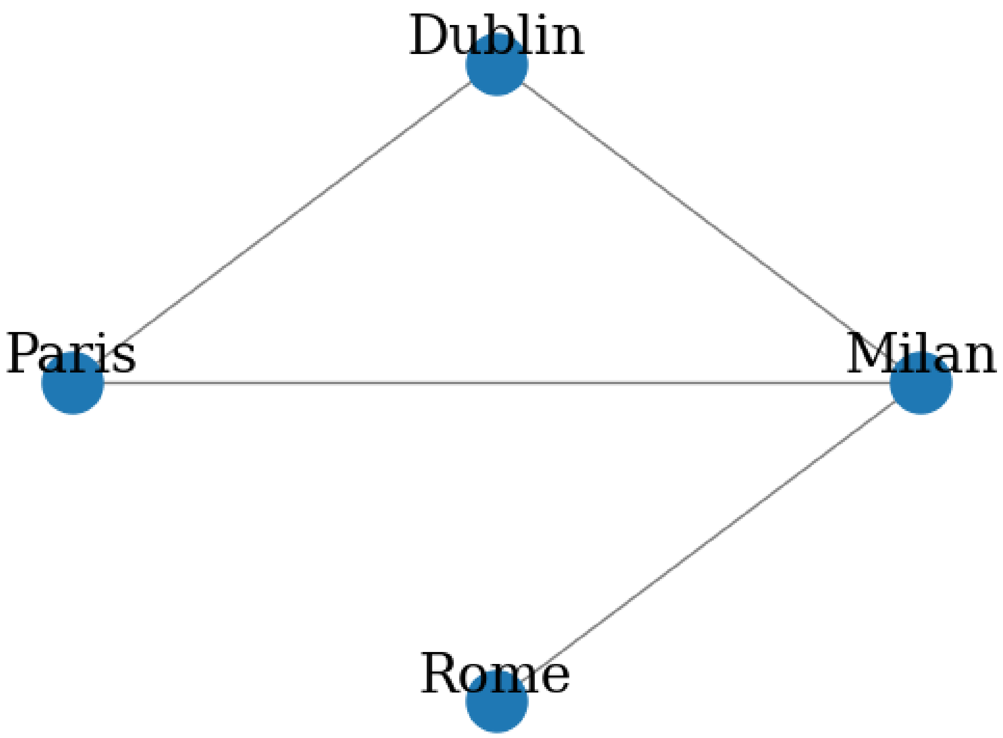
[1]:

```
def draw_graph(G, pos_nodes, node_names={},
node_size=50, plot_weight=False):

    nx.draw(G, pos_nodes, with_labels=False,
node_size=node_size, edge_color='gray', arrowsize=30)



    pos_attrs = {}

    for node, coords in pos_nodes.items():

        pos_attrs[node] = (coords[0], coords[1] + 0.08)



    nx.draw_networkx_labels(G, pos_attrs, font_family='serif', font_size=20)





    if plot_weight:

        pos_attrs = {}

        for node, coords in pos_nodes.items():

            pos_attrs[node] = (coords[0], coords[1] + 0.08)



        nx.draw_networkx_labels(G, pos_attrs, font_family='serif', font_size=20)

        edge_labels=dict([((a,b,),d["weight"]) for a,b,d in G.edges(data=True)])

        nx.draw_networkx_edge_labels(G, pos_nodes, edge_labels=edge_labels)



    plt.axis('off')

    axis = plt.gca()

    axis.set_xlim([1.2*x for x in axis.get_xlim()])

    axis.set_ylim([1.2*y for y in axis.get_ylim()])
```

[2]:

```
import numpy as np

import pandas as pd

import networkx as nx

import matplotlib.pyplot as plt



G = nx.Graph()

V = {'Dublin', 'Paris', 'Milan', 'Rome'}

E = [('Milan','Dublin'), ('Milan','Paris'), ('Paris','Dublin'), ('Milan','Rome')]

G.add_nodes_from(V)

G.add_edges_from(E)

draw_graph(G, pos_nodes=nx.shell_layout(G), node_size=500)
```

```
/lib/python3.11/site-packages/networkx/drawing/nx_pylab.py:304: UserWarning:

The arrowsize keyword argument is not applicable when drawing edges
with LineCollection.

To make this warning go away, either specify `arrows=True` to
force FancyArrowPatches or use the default value for arrowsize.
Note that using FancyArrowPatches may be slow for large graphs.

  draw_networkx_edges(G, pos, arrows=arrows, **edge_kwds)
```



[3]:

```
{G.degree(v): v for v in G.nodes}
```

[3]:

```
{2: 'Dublin', 1: 'Rome', 3: 'Milan'}
```

[4]:

```
print(f"Graph Order: {G.number_of_nodes()}")

print(f"Graph Size: {G.number_of_edges()}")

print(f"Degree for nodes: { {v: G.degree(v) for v in G.nodes} }")

print(f"Neighbors for nodes: { {v: list(G.neighbors(v)) for v in G.nodes} }")
```

```
Graph Order: 4
Graph Size: 4
Degree for nodes: {'Paris': 2, 'Rome': 1, 'Milan': 3, 'Dublin': 2}
Neighbors for nodes: {'Paris': ['Milan', 'Dublin'], 'Rome': ['Milan'], 'Milan':
['Dublin', 'Paris', 'Rome'], 'Dublin': ['Milan', 'Paris']}
```

[5]:

```
ego_graph_milan = nx.ego_graph(G, "Milan")
```

```
print(f"Nodes: {ego_graph_milan.nodes}")

print(f"Edges: {ego_graph_milan.edges}")
```

```
Nodes: ['Paris', 'Rome', 'Milan', 'Dublin']
Edges: [('Paris', 'Milan'), ('Paris', 'Dublin'), ('Rome', 'Milan'), ('Milan',
'Dublin')]
```

[6]:

```
new_nodes = {'London', 'Madrid'}

new_edges = [('London','Rome'), ('Madrid','Paris')]

G.add_nodes_from(new_nodes)

G.add_edges_from(new_edges)

print(f"V = {G.nodes}")

print(f"E = {G.edges}")
```

```
V = ['Paris', 'Rome', 'Milan', 'Dublin', 'Madrid', 'London']
E = [('Paris', 'Milan'), ('Paris', 'Dublin'), ('Paris', 'Madrid'), ('Rome',
'Milan'), ('Rome', 'London'), ('Milan', 'Dublin')]
```

[7]:

```
node_remove = {'London', 'Madrid'}

G.remove_nodes_from(node_remove)

print(f"V = {G.nodes}")

print(f"E = {G.edges}")
```

```
V = ['Paris', 'Rome', 'Milan', 'Dublin']
E = [('Paris', 'Milan'), ('Paris', 'Dublin'), ('Rome', 'Milan'), ('Milan',
'Dublin')]
```

[8]:

```
node_edges = [('Milan','Dublin'), ('Milan','Paris')]

G.remove_edges_from(node_edges)

print(f"V = {G.nodes}")

print(f"E = {G.edges}")
```

```
V = ['Paris', 'Rome', 'Milan', 'Dublin']
E = [('Paris', 'Dublin'), ('Rome', 'Milan')]
```

[9]:

```
print(nx.to_edgelist(G))
```

```
[('Paris', 'Dublin', {}), ('Rome', 'Milan', {})]
```

[10]:

```
print(nx.to_pandas_adjacency(G))
```

```
        Paris  Rome  Milan  Dublin
Paris     0.0   0.0    0.0     1.0
Rome      0.0   0.0    1.0     0.0
Milan     0.0   1.0    0.0     0.0
Dublin    1.0   0.0    0.0     0.0
```

[11]:

```
import networkx as nx

G = nx.DiGraph()

V = {'Dublin', 'Paris', 'Milan', 'Rome'}

E = [('Milan','Dublin'), ('Paris','Milan'), ('Paris','Dublin'), ('Milan','Rome')]

G.add_nodes_from(V)

G.add_edges_from(E)

print(nx.to_pandas_edgelist(G))

print(nx.to_pandas_adjacency(G))
```

```
  source  target
0  Paris   Milan
1  Paris  Dublin
2  Milan  Dublin
3  Milan    Rome
        Paris  Rome  Milan  Dublin
Paris     0.0   0.0    1.0     1.0
Rome      0.0   0.0    0.0     0.0
Milan     0.0   1.0    0.0     1.0
Dublin    0.0   0.0    0.0     0.0
```

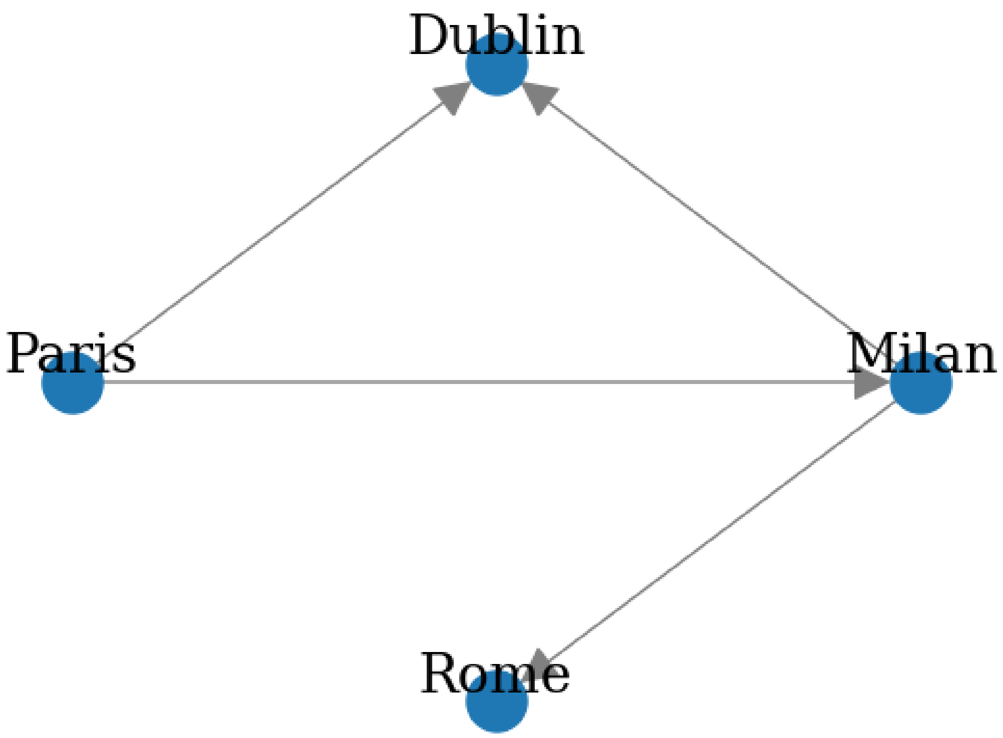
[12]:

```
print(f"Indegree for nodes: { {v: G.in_degree(v) for v in G.nodes} }")

print(f"Outegree for nodes: { {v: G.out_degree(v) for v in G.nodes} }")
```

```
Indegree for nodes: {'Paris': 0, 'Rome': 1, 'Milan': 1, 'Dublin': 2}
Outegree for nodes: {'Paris': 2, 'Rome': 0, 'Milan': 2, 'Dublin': 0}
```

[13]:

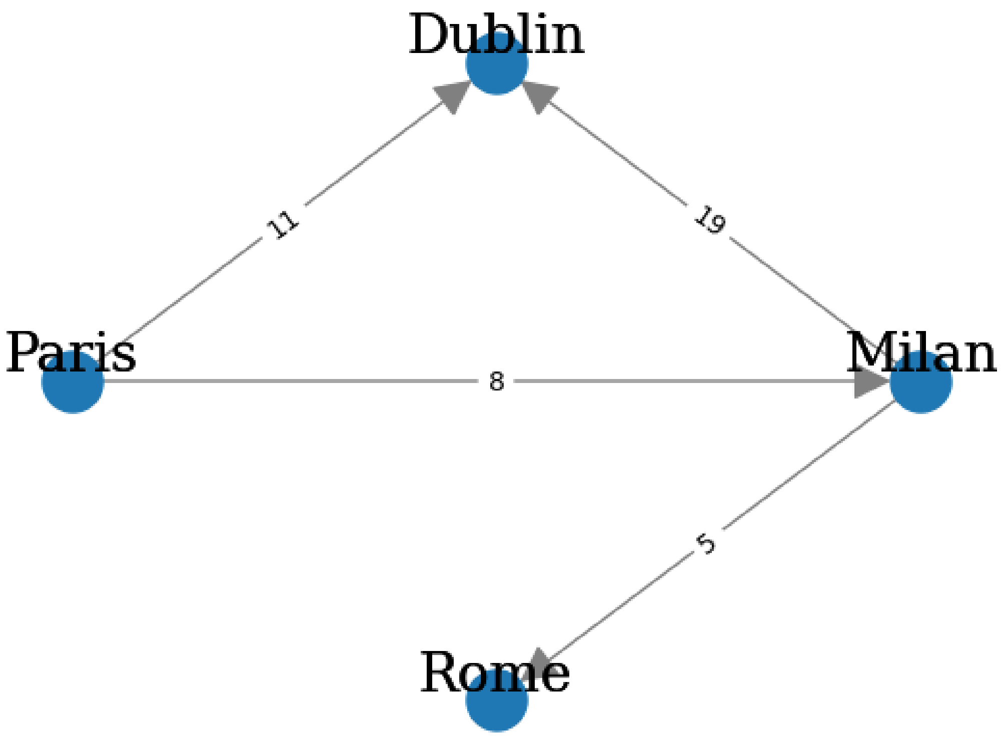
```
draw_graph(G, pos_nodes=nx.shell_layout(G), node_size=500)
```



[14]:

```
import networkx as nx

G = nx.MultiDiGraph()

V = {'Paris', 'Dublin','Milan', 'Rome'}

E = [ ('Paris','Dublin', 11), ('Paris','Milan', 8),

     ('Milan','Rome', 5),('Milan','Dublin', 19)]

G.add_nodes_from(V)

G.add_weighted_edges_from(E)

draw_graph(G, pos_nodes=nx.shell_layout(G), node_size=500, plot_weight=True)

print(nx.to_pandas_edgelist(G))

print(nx.to_pandas_adjacency(G))
```

```
  source  target  weight
0  Paris  Dublin      11
1  Paris   Milan       8
2  Milan    Rome       5
3  Milan  Dublin      19
        Paris  Rome  Milan  Dublin
Paris     0.0   0.0    8.0    11.0
Rome      0.0   0.0    0.0     0.0
Milan     0.0   5.0    0.0    19.0
Dublin    0.0   0.0    0.0     0.0
```

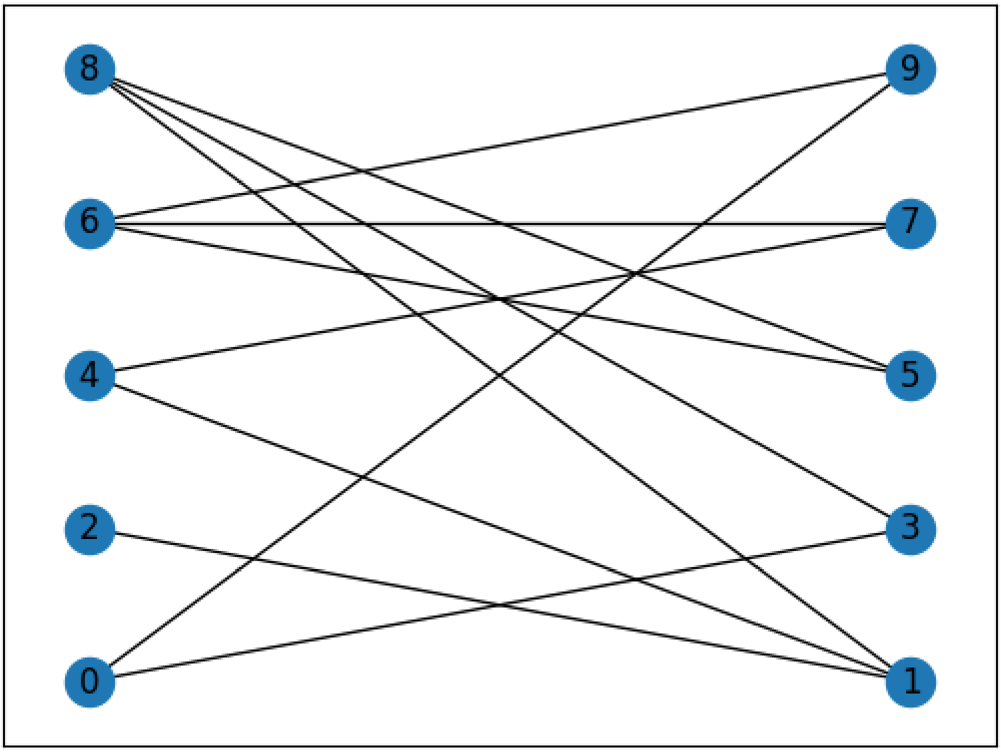


[15]:

```
n_nodes = 10

n_edges = 12

bottom_nodes = [ith for ith in range(n_nodes) if ith % 2 ==0]

top_nodes = [ith for ith in range(n_nodes) if ith % 2 ==1]

iter_edges = zip(

    np.random.choice(bottom_nodes, n_edges),

    np.random.choice(top_nodes, n_edges))

edges = pd.DataFrame([
    {"source": a, "target": b} for a, b in iter_edges])

B = nx.Graph()

B.add_nodes_from(bottom_nodes, bipartite=0)

B.add_nodes_from(top_nodes, bipartite=1)

B.add_edges_from([tuple(x) for x in edges.values])
```

[16]:

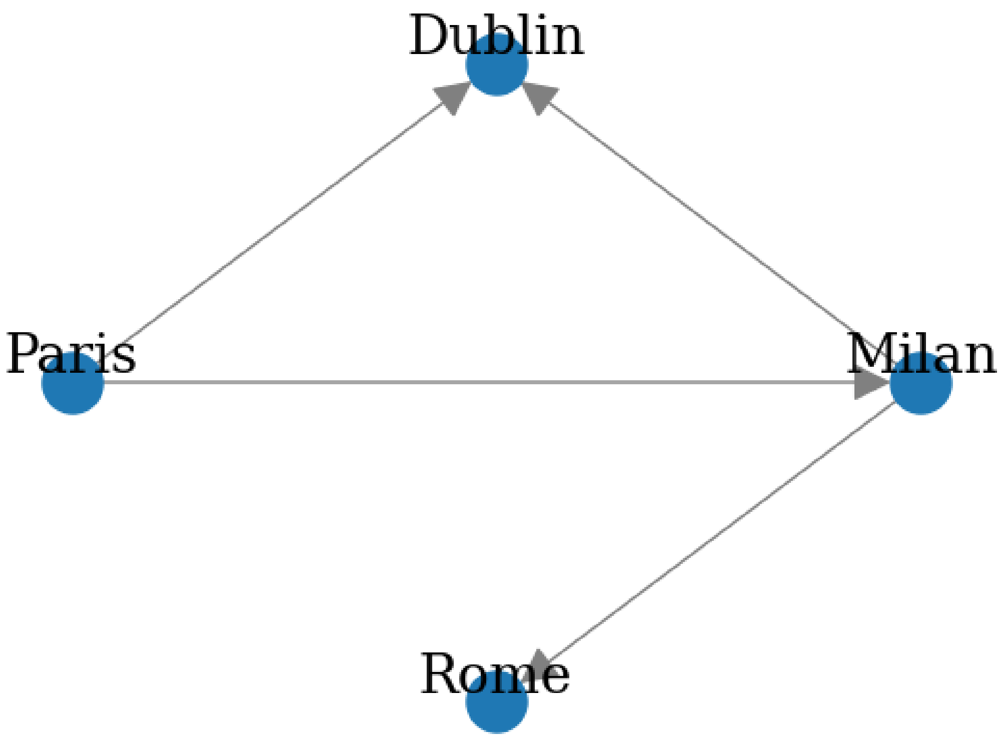
```
from networkx.drawing.layout import bipartite_layout

pos = bipartite_layout(B, bottom_nodes)

nx.draw_networkx(B, pos=pos)
```



[17]:

```
import networkx as nx

directed_multi_graph = nx.MultiDiGraph()

V = {'Dublin', 'Paris', 'Milan', 'Rome'}

E = [('Milan','Dublin'), ('Milan','Dublin'), ('Paris','Milan'),
('Paris','Dublin'), ('Milan','Rome'), ('Milan','Rome')]

directed_multi_graph.add_nodes_from(V)

directed_multi_graph.add_edges_from(E)
```

```
draw_graph(G, pos_nodes=nx.shell_layout(G), node_size=500)
```



[ ]:

Try the Notebook Tour.

×