Introduction-to-NetworkX

January 3, 2020

1 Handling Graphs with NetworkX

Python offers the library NetworkX for manipulating graphs. You can learn more here:

https://networkx.github.io/

https://networkx.github.io/documentation/stable/tutorial.html

[97]:
```
import networkx as nx
%matplotlib inline
```

1.0.1 Creating a graph

[98]:
```
G = nx.Graph()
```

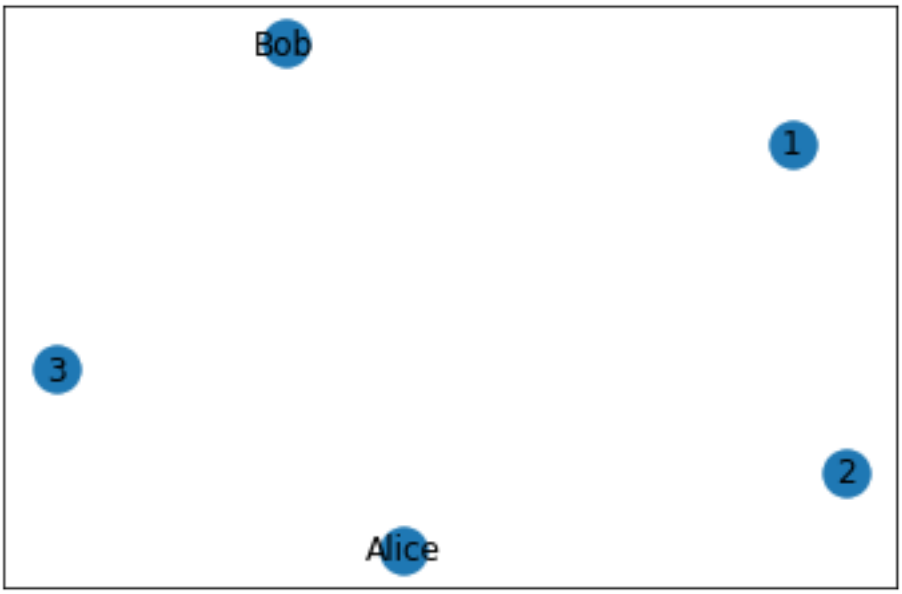
Add nodes to the graph

[99]:
```
G.add_node(1)
G.add_nodes_from([2,3])
G.add_node('Alice')
G.add_node('Bob')
print(G.nodes())
```

```
[1, 2, 3, 'Alice', 'Bob']
```

Draw the graph

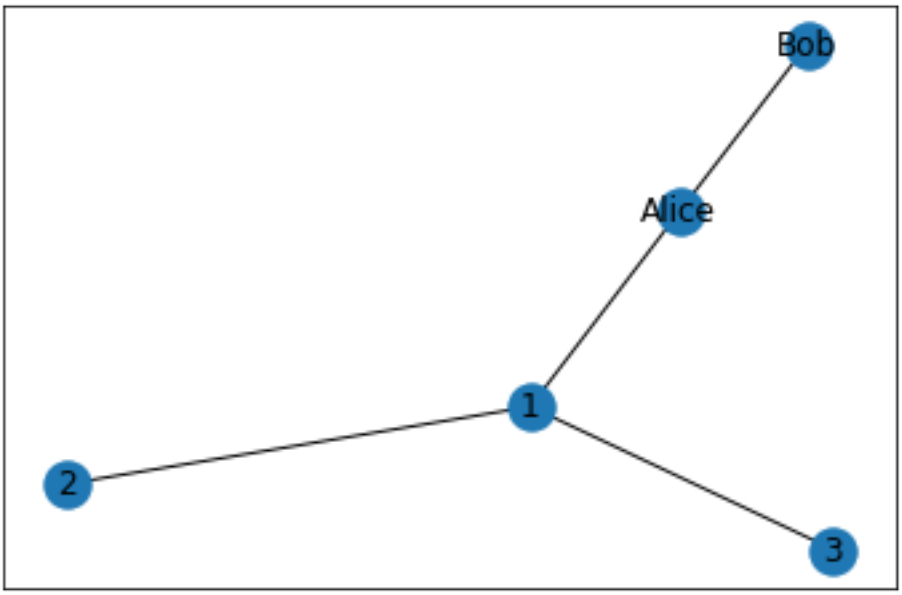
[100]:
```
nx.draw_networkx(G)
```



Add edges to the graph

[101]:
```
G.add_edge(1,2)
G.add_edges_from([(1,3),('Alice','Bob')])
e = (1,'Alice')
G.add_edge(*e)
```

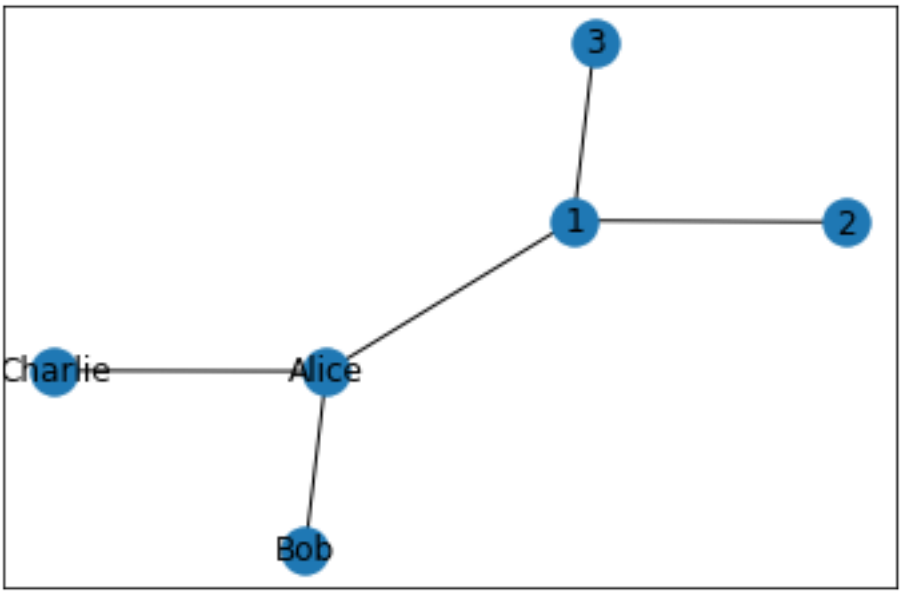
[102]:
```
nx.draw_networkx(G)
```



Adding an edge with a new node will create the node Charlie in the graph

[103]:
```
G.add_edge('Alice','Charlie')
print(G.edges())
print(G.nodes())
nx.draw_networkx(G)
```

```
[(1, 2), (1, 3), (1, 'Alice'), ('Alice', 'Bob'), ('Alice', 'Charlie')]
[1, 2, 3, 'Alice', 'Bob', 'Charlie']
```



A graph is a dictionary with nodes as the keys

Each node is a dictionary with the neighbors as the keys, and the edge properties as values

[104]:
```
G[1]
```

[104]:
```
AtlasView({2: {}, 3: {}, 'Alice': {}})
```

[105]:
```
print(G.nodes)
print(G.edges)
```

```
[1, 2, 3, 'Alice', 'Bob', 'Charlie']
[(1, 2), (1, 3), (1, 'Alice'), ('Alice', 'Bob'), ('Alice', 'Charlie')]
```

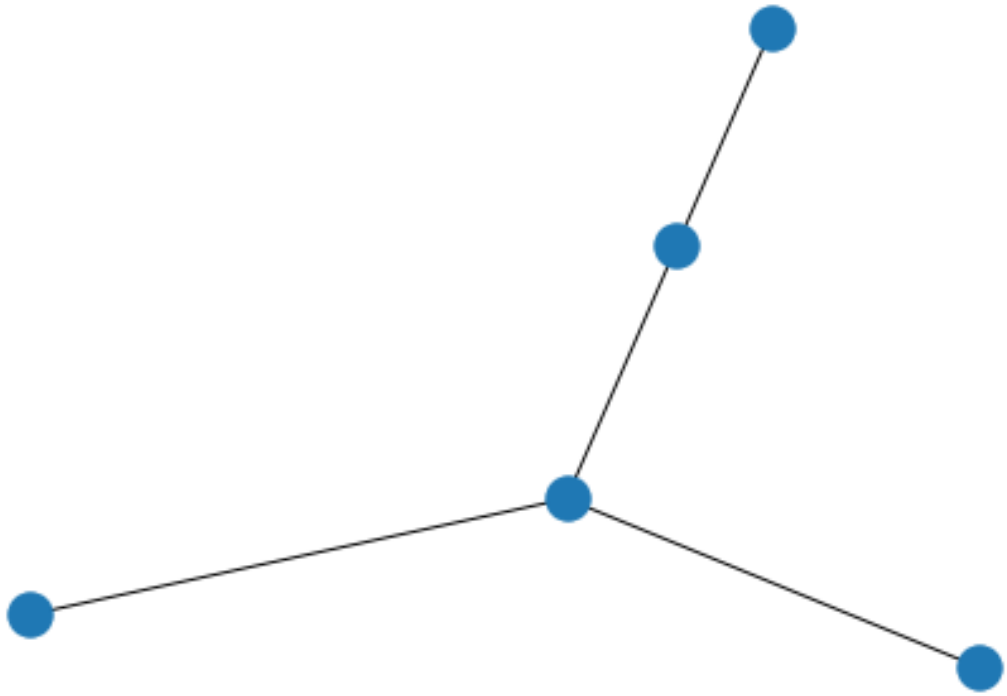
[106]:
```
print(G.nodes[1])
```

```
{}
```

Creating a graph from edges

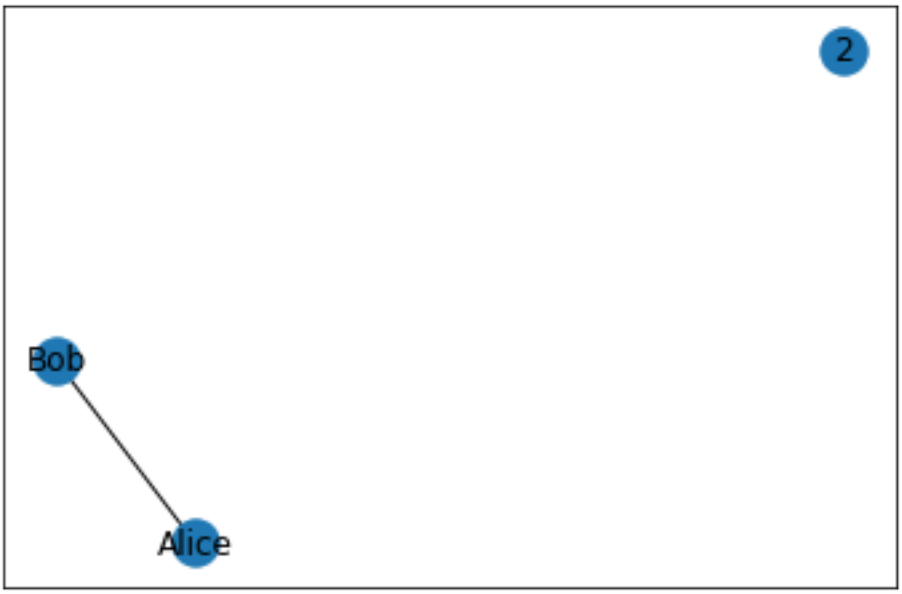
[107]:
```
G2 = nx.Graph()
G2.add_edges_from([(1,2),(1,3),('Alice','Bob'),(1,'Alice')])
print(G2.nodes())
print(G2.edges())
nx.draw(G2)
```

```
[1, 2, 3, 'Alice', 'Bob']
[(1, 2), (1, 3), (1, 'Alice'), ('Alice', 'Bob')]
```



[108]:
```
G2.remove_edge(1,3)
G2.remove_node(3)
G2.remove_node(1)
print(G2.nodes())
print(G2.edges())
nx.draw_networkx(G2)
```

```
[2, 'Alice', 'Bob']
[('Alice', 'Bob')]
```

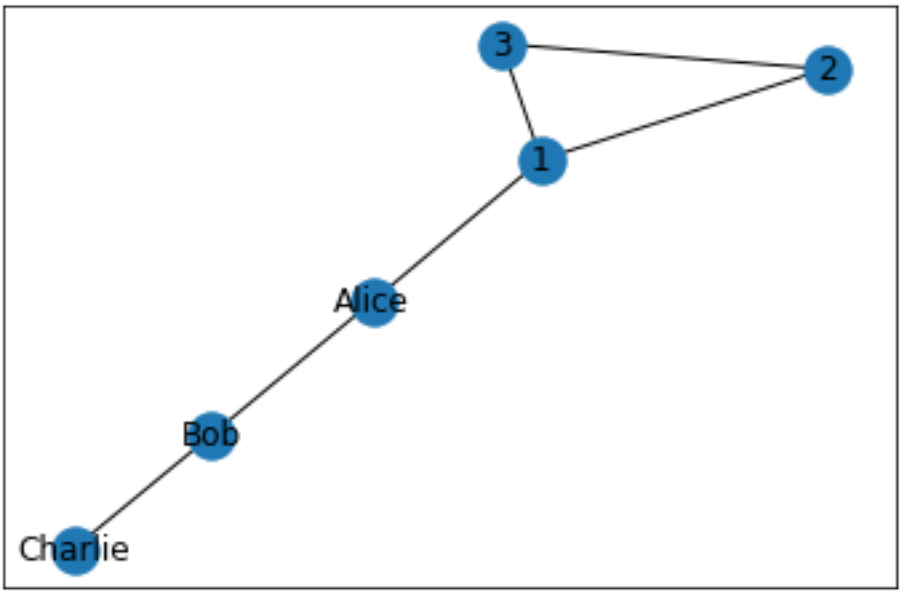


Reading a graph from a file with list of edges

http://networkx.readthedocs.io/en/networkx-1.11/reference/readwrite.html

[109]:
```
#Read a graph from a list of edges
G3 = nx.read_edgelist('graph_edges.txt')
print(G3.nodes())
print(G3.edges())
nx.draw_networkx(G3)
```

```
['1', '2', '3', 'Alice', 'Bob', 'Charlie']
[('1', '2'), ('1', '3'), ('1', 'Alice'), ('2', '3'), ('Alice', 'Bob'), ('Bob',
'Charlie')]
```



1.0.2 Graph attributes

You can assign attributes and values to the nodes and edges of the graph

[110]:
```
G3.nodes['Alice']['gender'] = 'female'
G3.nodes['Bob']['gender'] = 'male'
G3.nodes['Charlie']['gender'] = 'male'
G3.nodes['1']['value'] = 1
G3.nodes['2']['value'] = -1
G3.nodes['3']['value'] = 0
for n in G3.nodes():
    print(G3.nodes[n])
```

```
{'value': 1}
{'value': -1}
{'value': 0}
{'gender': 'female'}
{'gender': 'male'}
{'gender': 'male'}
```

[111]:
```
for n in G3.nodes():
    print(G3[n])
```

```
{'2': {}, '3': {}, 'Alice': {}}
{'1': {}, '3': {}}
{'1': {}, '2': {}}
{'Bob': {}, '1': {}}
```

```
{'Alice': {}, 'Charlie': {}}
{'Bob': {}}
```

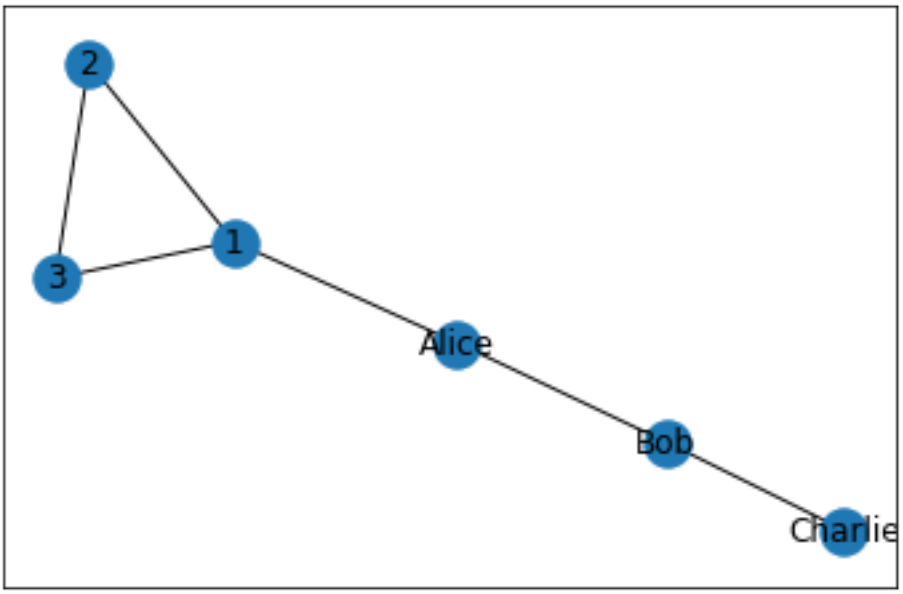
[112]:
```
G3.nodes['Alice']['value'] = 1
G3.nodes['Bob']['value'] = -1
G3.nodes['Charlie']['value'] = 1
for n in G3.nodes():
    print(n+ ":" + str(G3.nodes[n]['value']))

for n in G3.nodes():
    print(n,G3.nodes[n])
```

```
1:1
2:-1
3:0
Alice:1
Bob:-1
Charlie:1
1 {'value': 1}
2 {'value': -1}
3 {'value': 0}
Alice {'gender': 'female', 'value': 1}
Bob {'gender': 'male', 'value': -1}
Charlie {'gender': 'male', 'value': 1}
```

[113]:
```
G3['Alice']['Bob']['label'] = 'strong'
print(G3['Bob']['Alice'])
print(G3['Alice'])
print(G3['Bob'])
```

```
{'label': 'strong'}
{'Bob': {'label': 'strong'}, '1': {}}
{'Alice': {'label': 'strong'}, 'Charlie': {}}
```

[116]:
```
nx.draw_networkx(G3, with_labels=True)
```



1.0.3 Weighted graphs

A special attribute of a an edge is the "weight". When adding weighted edges, you enter triples consisting of the two edge endpoints and the weight of the edge. This weight is stored in an attribute "weight" by default.

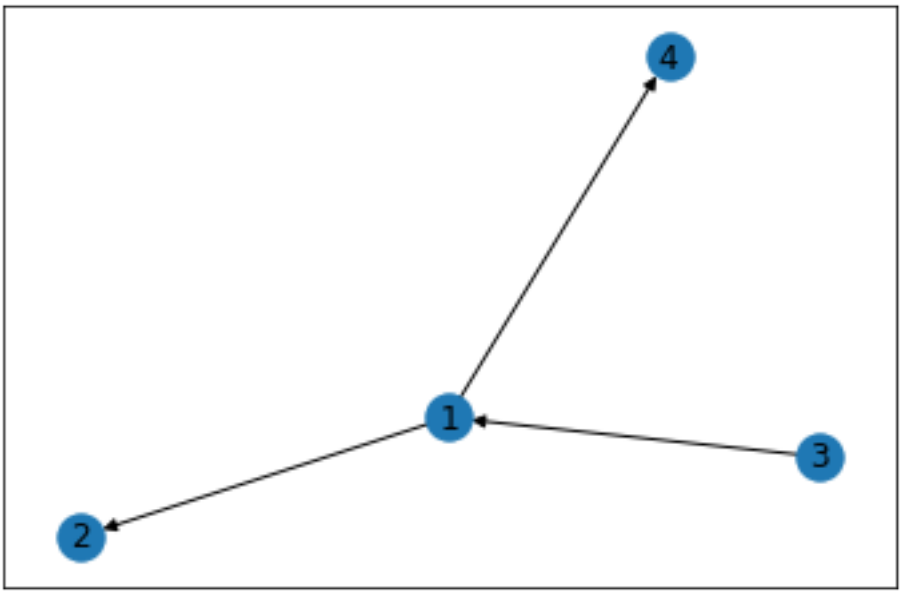
[114]:
```
G4 = nx.Graph()
G4.add_weighted_edges_from([(1,2,0.5),(2,3,0.1),(3,4,0.7)])
for (a,b) in G4.edges():
    print (G4[a][b])
for (a,b,w) in G4.edges(data =True): #data=True returns weight as well
    print (str(a)+" "+ str(b) + " " + str(w['weight']))
for n in G4:
    print(G4[n])
```

```
{'weight': 0.5}
{'weight': 0.1}
{'weight': 0.7}
1 2 0.5
2 3 0.1
3 4 0.7
{2: {'weight': 0.5}}
{1: {'weight': 0.5}, 3: {'weight': 0.1}}
{2: {'weight': 0.1}, 4: {'weight': 0.7}}
{3: {'weight': 0.7}}
```

1.0.4 Directed Graphs

[117]:
```
DG=nx.DiGraph()
DG.add_weighted_edges_from([(1,2,0.5), (3,1,0.75), (1,4,0.1)])
print(DG.edges())
for n in DG:
    print(DG[n])
nx.draw_networkx(DG)
```

```
[(1, 2), (1, 4), (3, 1)]
{2: {'weight': 0.5}, 4: {'weight': 0.1}}
{}
{1: {'weight': 0.75}}
{}
```



1.0.5 Graph Operations

Some common graph operations and algorithms are implemented in networkx library.

http://networkx.readthedocs.io/en/networkx-1.11/reference/algorithms.html

Neighbors, degrees and adjancency matrix

[118]:
```
print(G.neighbors(1)) # returns the neighbors of a node
print(G.degree(1)) # returns the degree of a node
print(G4.degree(3, weight='weight'))
print(G4.degree(3))
A = nx.adjacency_matrix(G)
print(A)
```

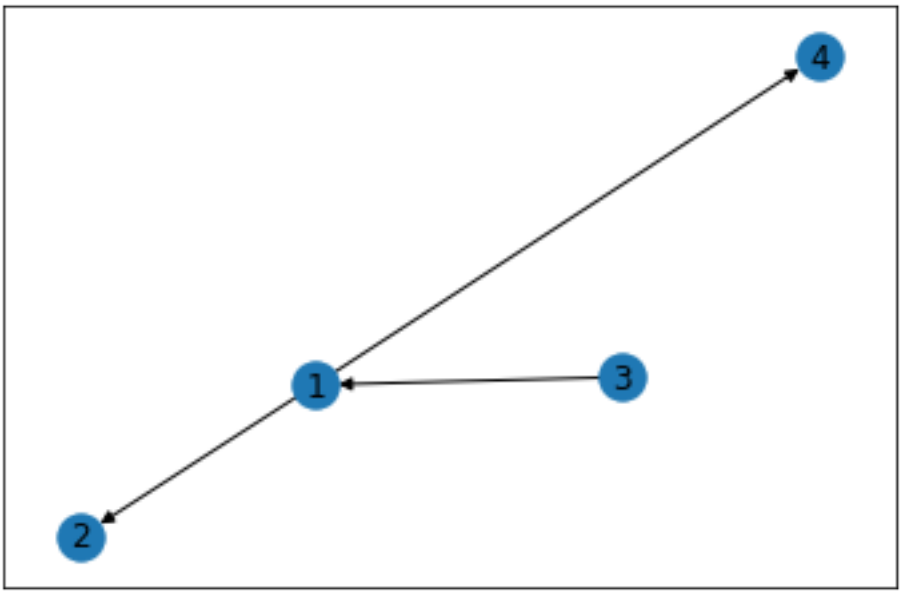
```
#the adjacency matrix is stored as a sparse matrix
print(type(A))
```

```
<dict_keyiterator object at 0x00000152B8005EA8>
3
0.7999999999999999
2
  (0, 1)        1
  (0, 2)        1
  (0, 3)        1
  (1, 0)        1
  (2, 0)        1
  (3, 0)        1
  (3, 4)        1
  (3, 5)        1
  (4, 3)        1
  (5, 3)        1
<class 'scipy.sparse.csr.csr_matrix'>
```

Neighbors and degrees for directed or weighted graphs

[119]:
```
nx.draw_networkx(DG)
print(list(DG.successors(1)))
print(list(DG.neighbors(1)))
print(list(DG.predecessors(1)))
print(DG.out_degree(1,weight='weight'))
print(DG.out_degree(1))
print(DG.in_degree(1))
```

```
[2, 4]
[2, 4]
[3]
0.6
2
1
```



1.0.6 Connected components

[120]:
```
G3.add_edge('1','Alice')
G3.remove_edge('1','Alice') #you can also remove_node, or a list of nodes or␣
 ↪edges (remove_nodes_from, remove_edges_from)
G3.add_edge('Alice','Charlie')
G3.add_edge('1','4')
nx.draw_networkx(G3)
print(nx.number_connected_components(G3))
```

2

[121]:
```
C = nx.connected_components(G3)
print(type(C))
for c in C:
    print(c)
```

```
<class 'generator'>
{'4', '2', '1', '3'}
{'Charlie', 'Alice', 'Bob'}
```

Get the connected component subgraphs

[122]:
```
connected_subgraphs = [nx.subgraph(G3,c) for c in nx.connected_components(G3)]
for GC in connected_subgraphs:
    print(GC.nodes())
    print(GC.edges())
    print(len(GC))
```

```
['1', '2', '3', '4']
[('1', '2'), ('1', '3'), ('1', '4'), ('2', '3')]
4
['Charlie', 'Alice', 'Bob']
[('Charlie', 'Bob'), ('Charlie', 'Alice'), ('Alice', 'Bob')]
3
```

Get the largest connected component

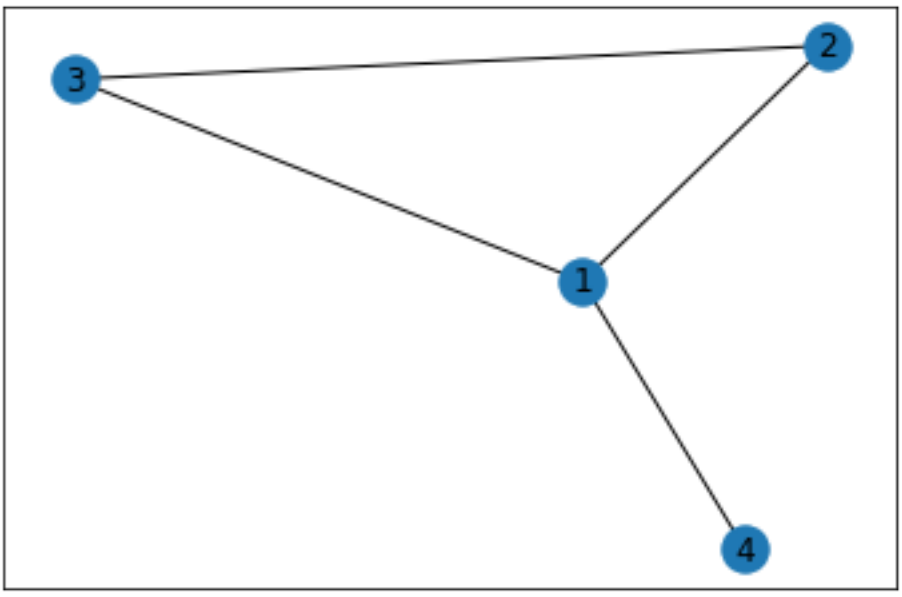
[123]:
```
# Get the nodes

largest_cc = max(nx.connected_components(G3), key=len)
```

```
print(largest_cc)
```

```
{'4', '2', '1', '3'}
```

[124]:
```
#Get the subgraph

largest_cc = max(nx.connected_components(G3), key=len)
print(largest_cc)
CC_max = nx.subgraph(G3,largest_cc)
nx.draw_networkx(CC_max)
```

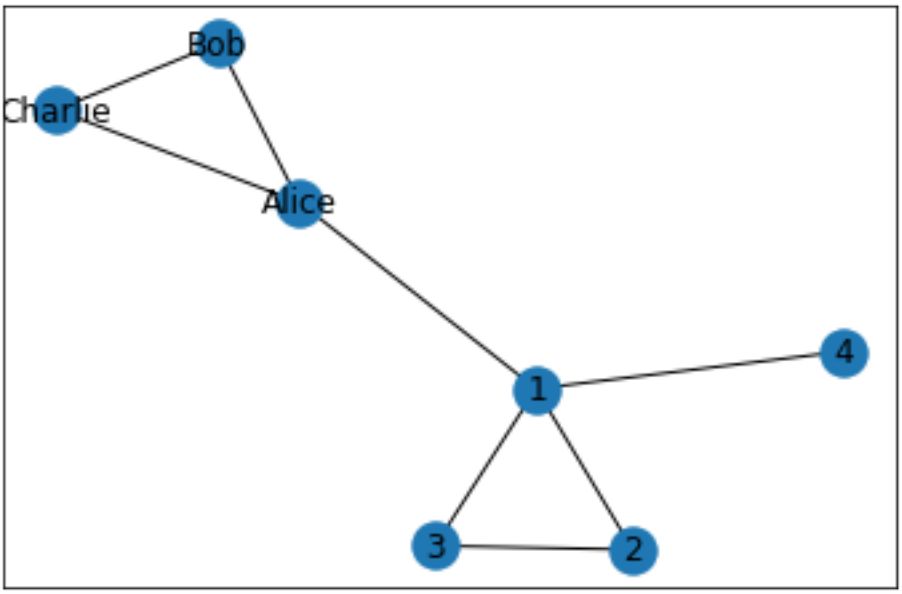
```
{'4', '2', '1', '3'}
```



1.0.7 Shortest paths

[125]:
```
G3.add_edge('1','Alice')
nx.draw_networkx(G3)
sp = nx.shortest_path(G3,'3','Bob')
print(sp)
print(len(sp)-1)
print(nx.shortest_path_length(G3,'3','Bob'))
```

```
['3', '1', 'Alice', 'Bob']
3
3
```



[126]:
```
SP1 = nx.single_source_shortest_path(G3,'1')
print(SP1)
#print(nx.single_source_shortest_path_length(G3,'1'))
```

```
{'1': ['1'], '2': ['1', '2'], '3': ['1', '3'], '4': ['1', '4'], 'Alice': ['1',
'Alice'], 'Bob': ['1', 'Alice', 'Bob'], 'Charlie': ['1', 'Alice', 'Charlie']}
```

[127]:
```
SP = dict(nx.all_pairs_shortest_path(G3))
print(SP)
```

```
{'1': {'1': ['1'], '2': ['1', '2'], '3': ['1', '3'], '4': ['1', '4'], 'Alice':
['1', 'Alice'], 'Bob': ['1', 'Alice', 'Bob'], 'Charlie': ['1', 'Alice',
'Charlie']}, '2': {'2': ['2'], '1': ['2', '1'], '3': ['2', '3'], '4': ['2', '1',
'4'], 'Alice': ['2', '1', 'Alice'], 'Bob': ['2', '1', 'Alice', 'Bob'],
'Charlie': ['2', '1', 'Alice', 'Charlie']}, '3': {'3': ['3'], '1': ['3', '1'],
'2': ['3', '2'], '4': ['3', '1', '4'], 'Alice': ['3', '1', 'Alice'], 'Bob':
['3', '1', 'Alice', 'Bob'], 'Charlie': ['3', '1', 'Alice', 'Charlie']}, 'Alice':
{'Alice': ['Alice'], 'Bob': ['Alice', 'Bob'], 'Charlie': ['Alice', 'Charlie'],
'1': ['Alice', '1'], '2': ['Alice', '1', '2'], '3': ['Alice', '1', '3'], '4':
['Alice', '1', '4']}, 'Bob': {'Bob': ['Bob'], 'Alice': ['Bob', 'Alice'],
'Charlie': ['Bob', 'Charlie'], '1': ['Bob', 'Alice', '1'], '2': ['Bob', 'Alice',
'1', '2'], '3': ['Bob', 'Alice', '1', '3'], '4': ['Bob', 'Alice', '1', '4']},
'Charlie': {'Charlie': ['Charlie'], 'Bob': ['Charlie', 'Bob'], 'Alice':
['Charlie', 'Alice'], '1': ['Charlie', 'Alice', '1'], '2': ['Charlie', 'Alice',
'1', '2'], '3': ['Charlie', 'Alice', '1', '3'], '4': ['Charlie', 'Alice', '1',
'4']}, '4': {'4': ['4'], '1': ['4', '1'], '2': ['4', '1', '2'], '3': ['4', '1',
'3'], 'Alice': ['4', '1', 'Alice'], 'Bob': ['4', '1', 'Alice', 'Bob'],
```

```
'Charlie': ['4', '1', 'Alice', 'Charlie']}}
```

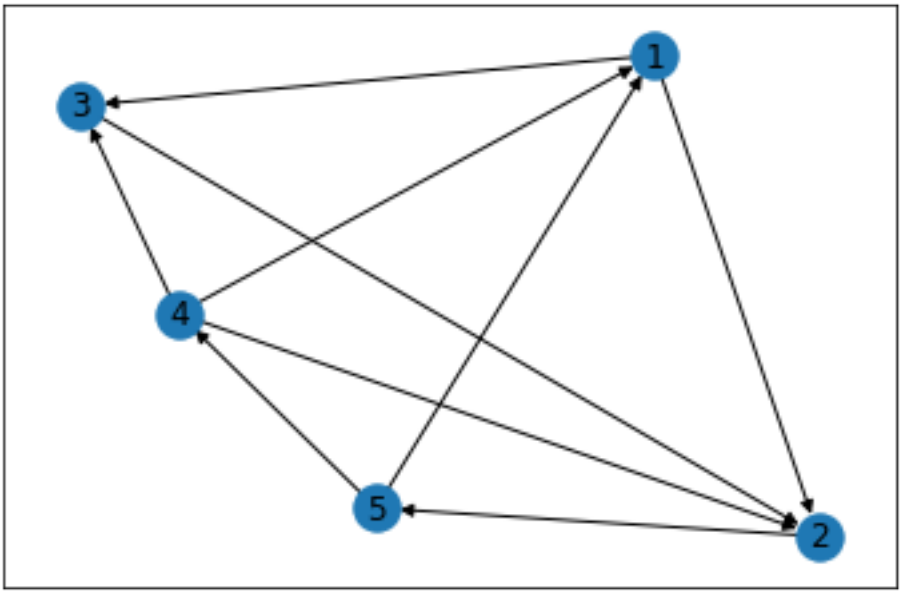
[128]:
```
print(SP['1']['Bob'])
```

```
['1', 'Alice', 'Bob']
```

1.0.8 Link Analysis

https://networkx.github.io/documentation/stable/reference/algorithms/link_analysis.html

[129]:
```
DG2 = nx.DiGraph()
DG2.add_edges_from([(1,2),(1,3),(3,2),(2,5),(4,1),(4,2),(4,3),(5,1),(5,4)])
nx.draw_networkx(DG2)
```



1.0.9 Pagerank

[130]:
```
pr = nx.pagerank(DG2)
print(pr)

pr = nx.pagerank(G3)
print(pr)
```

```
{1: 0.18064505060873787, 2: 0.2713164308772404, 3: 0.14665711544131715, 5:
0.26061906832422166, 4: 0.14076233474848301}
{'1': 0.24203081965436962, '2': 0.12671382905463274, '3': 0.12671382905463274,
'Alice': 0.17987731223897474, 'Bob': 0.12590180503903042, 'Charlie':
0.12590180503903042, '4': 0.0728605999193293}
```

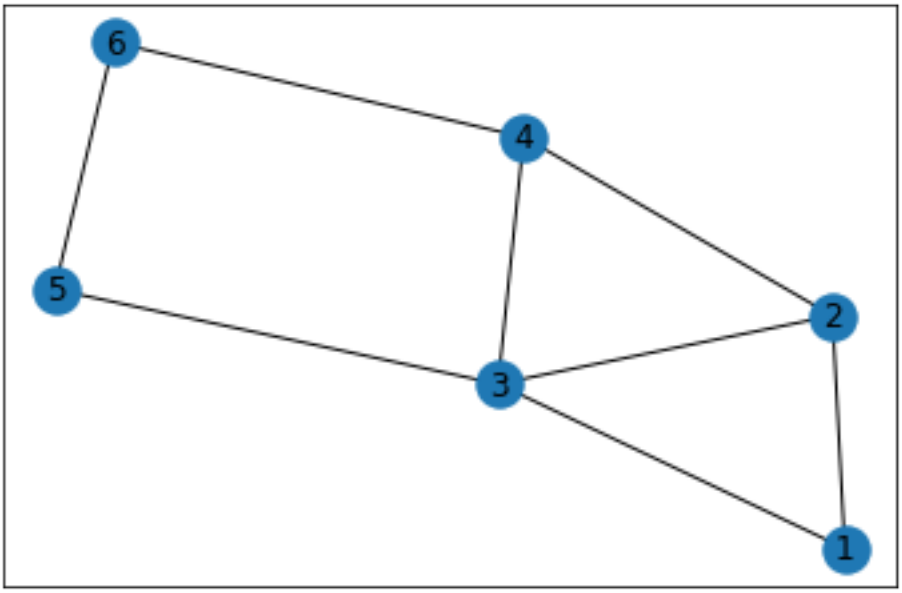
1.0.10 HITS

[131]:
```
[h,a] = nx.hits(DG2)
print(h)
print(a)
print(a[2])
```

```
{1: 0.3028419086392418, 2: 1.3109311069706554e-15, 3: 0.1674519922094525, 5:
0.1254412274444104, 4: 0.40426487170689396}
{1: 0.23681288036482923, 2: 0.3909843234563998, 3: 0.31612245503718484, 5:
3.06089826220615e-15, 4: 0.05608034114158294}
0.3909843234563998
```

1.0.11 Class Example

[132]:
```
G4 = nx.read_edgelist('graph-example.txt')
nx.draw_networkx(G4)
```



[133]:
```
print(nx.pagerank(G4))
print(nx.pagerank(G4, alpha = 0.5))
print(nx.pagerank(G4, alpha = 0.5, personalization = {'1':1}))
print(nx.pagerank(G4, alpha = 0.5, personalization = {'6':1}))
```

```
{'1': 0.1275509986653759, '2': 0.18232887836414124, '3': 0.2394858218322733,
'4': 0.18433868256410274, '5': 0.13267870340998367, '6': 0.13361691516412283}
{'1': 0.1390335398088414, '2': 0.17416862981664863, '3': 0.21337523839947284,
'4': 0.17643097727783, '5': 0.14740281022602297, '6': 0.14958880447118414}
{'1': 0.5510064790190851, '2': 0.16964493982349704, '3': 0.18185949066208873,
```

```
'4': 0.054964764048023335, '5': 0.026690643658381363, '6': 0.015833682788924434}
{'1': 0.015833407896886878, '2': 0.039357701756408944, '3': 0.07419162031220411,
'4': 0.15675161523961573, '5': 0.150192049257395, '6': 0.5636736055374894}
```

1.0.12 Betweeness

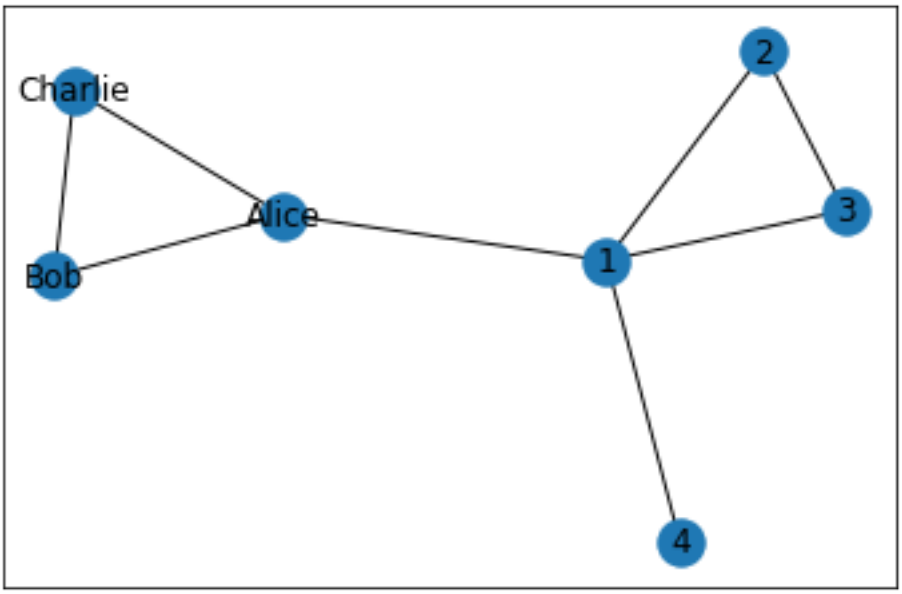
[134]:
```
BC = nx.edge_betweenness_centrality(G3)
print(BC)
```

```
{('1', '2'): 0.23809523809523808, ('1', '3'): 0.23809523809523808, ('1', '4'):
0.2857142857142857, ('1', 'Alice'): 0.5714285714285714, ('2', '3'):
0.047619047619047616, ('Alice', 'Bob'): 0.23809523809523808, ('Alice',
'Charlie'): 0.23809523809523808, ('Bob', 'Charlie'): 0.047619047619047616}
```

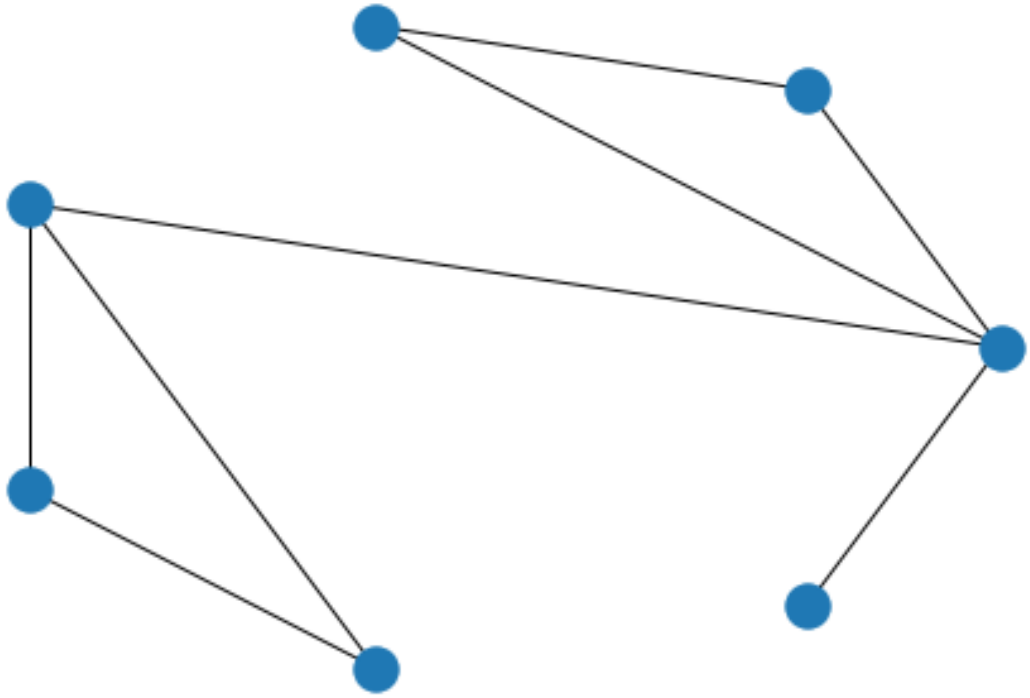
1.0.13 Drawing Graphs

http://networkx.readthedocs.io/en/networkx-1.11/reference/drawing.html

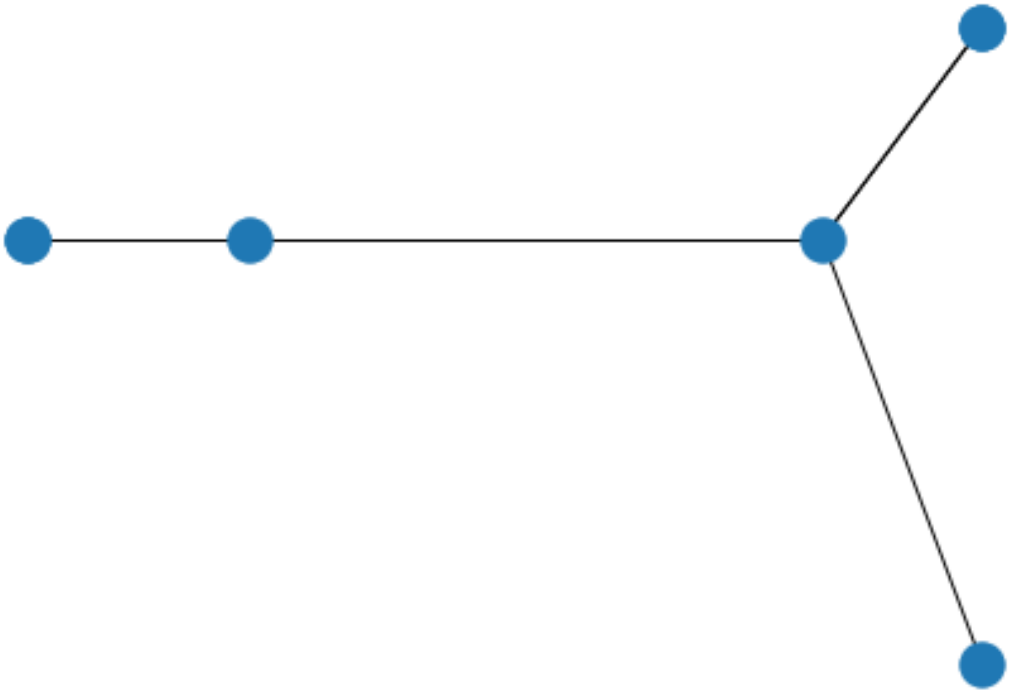
[135]:
```
nx.draw_networkx(G3)
```



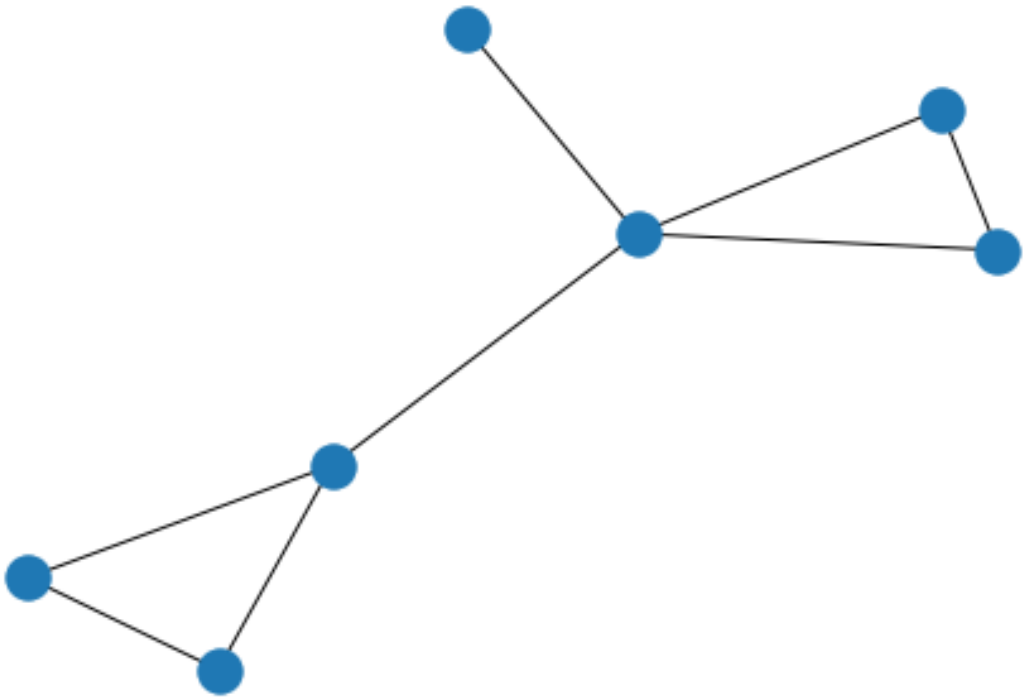
[136]:
```
nx.draw_circular(G3)
```



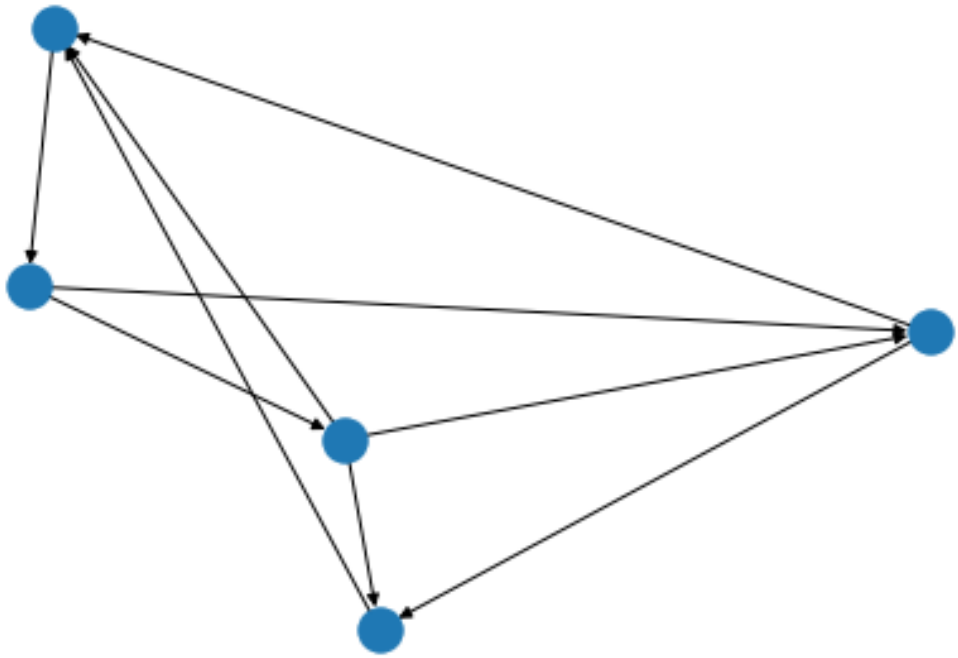
[137]:
```
nx.draw_spectral(G3)
```



[138]:
```
nx.draw_spring(G3)
```

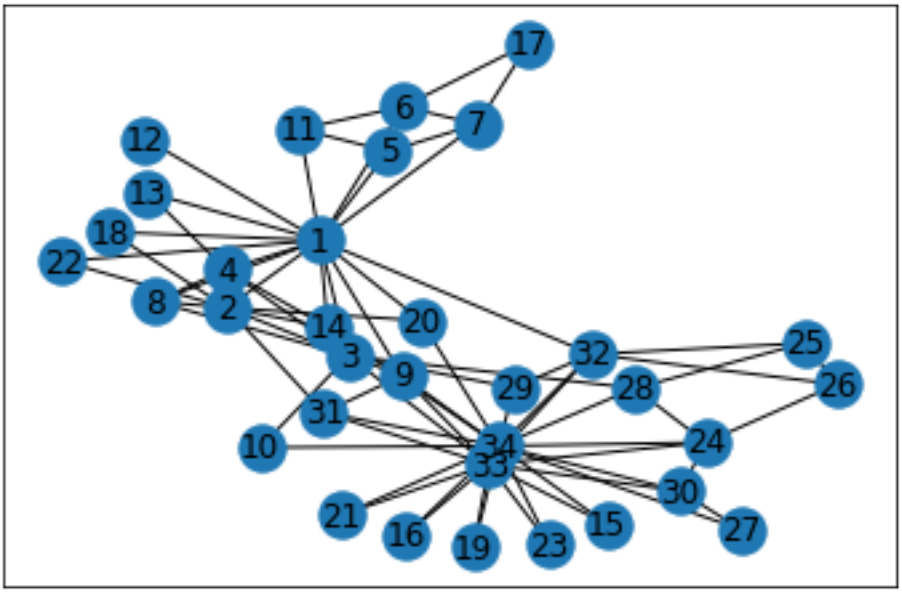


[139]:
```
nx.draw_spring(DG2)
```

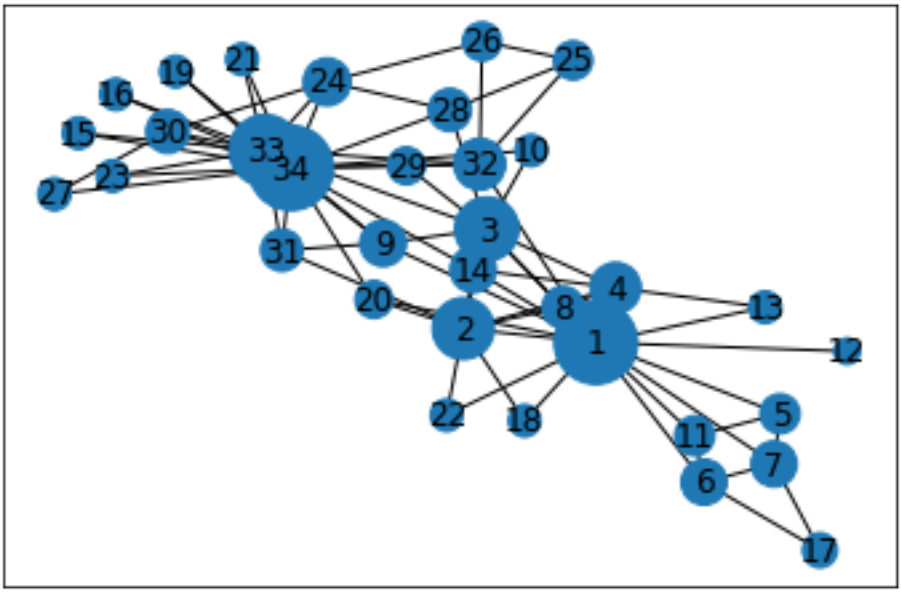


1.0.14 An example

[140]:
```
karate=nx.read_gml("karate.gml",label='id')
nx.draw_networkx(karate)
```



[141]:
```
pr = nx.pagerank(karate)
nx.draw_networkx(karate,node_size=[10000*v for v in pr.values()])
```



[ ]: