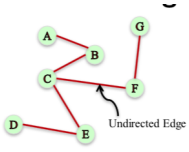
Week1

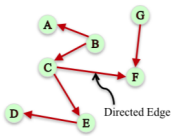
Network Definition and Vocabulary

- Edges & Nodes (or vertices)

- **Undirected** network (symmetrical) and **Directed** network (asymmetrical)

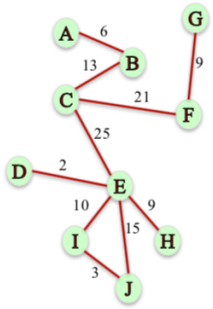
```
#Undirected
G=nx.Graph() G.add_edge('A','B')
G.add_edge('B','C')
#Directed
G=nx.DiGraph()
G.add_edge('B', 'A')
G.add_edge('B','C')
```





- **Weighted network**: a network where edges are assigned a (typically numerical) weight.

```
G=nx.Graph()
G.add_edge('A','B', weight = 6)
G.add_edge('B','C', weight = 13)
```

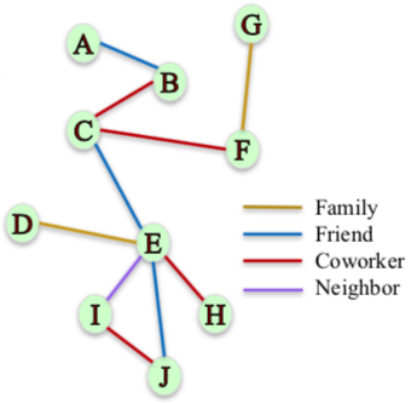


- **Signed Networks:** a network where edges are assigned positive or negative sign. Some networks can carry information about friendship and antagonism based on conflict or disagreement

```
G=nx.Graph()
G.add_edge('A','B', sign= '+')
G.add_edge('B','C', sign= '-')
```

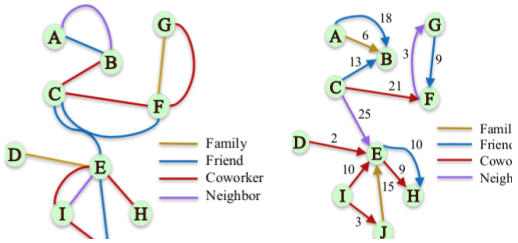
- **Other Edge Attributes**: Edges can carry many other labels or attributes

```
G=nx.Graph()
G.add_edge('A','B', relation= 'friend')
G.add_edge('B','C', relation= 'coworker')
G.add_edge('D','E', relation= 'family')
G.add_edge('E','I', relation= 'neighbor')
```



- **Mutigraph w/without direction**: A pair of nodes can have different types of relationships simultaneously

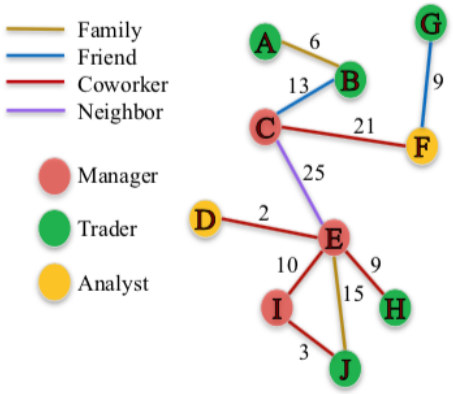
```
G=nx.MultiGraph()
G.add_edge('A','B', relation= 'friend')
G.add_edge('A','B', relation= neighbor')
G.add_edge('G','F', relation= 'family')
G.add_edge('G','F', relation= 'coworker')
##
G=nx.MultiDiGraph()
G.add_edge('A','B', weight=6,relation= 'friend
G.add_edge('A','B', weight=8, relation= 'neigh
G.add_edge('G','F', weight=3, relation= 'famil
```



Node attribute

Nodes can have another attribute which describes them

```
G=nx.Graph()
G.add_edge('A','B', weight= 6, relation = 'family')
G.add_edge('B','C', weight= 13, relation = 'friend')
G.add_node('A', role = 'trader')
G.add_node('B', role = 'trader')
G.add_node('C', role = 'manager')
In: G.nodes() # list of all nodes
Out: ['A', 'C', 'B']
In: G.nodes(data= True) #list of all nodes with attributes
Out: [('A', {'role': 'trader'}), ('C', {'role': 'manager'}), ('B', {'role': 'trader'})]
In: G.node['A']['role']
Out: 'manager'
```

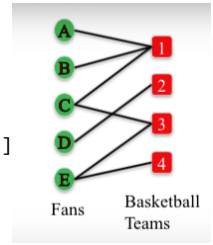


Number of times coworkers had lunch together in one year

Bipartite graphs

A graph whose nodes can be split into two sets L and R and every edge connects an node in L with a node in R.

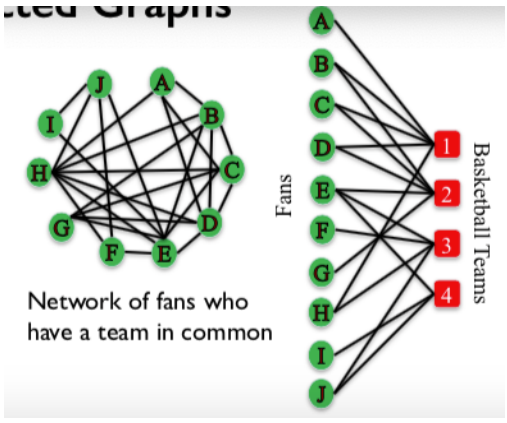
```
from networkx.algorithms import bipartite
B = nx.Graph() #No separate class for bipartite graphs
B.add_nodes_from(['A','B','C','D', 'E'], bipartite=0) #label one set of nodes 0
B.add_nodes_from([1,2,3,4], bipartite=1) #label other set of nodes 1
B.add_edges_from([('A',1), ('B',1), ('C',1), ('C',3), ('D',2), ('E',3), ('E', 4)]
bipartite.is_bipartite(B) # Check if B is bipartite
#Checking if a set of nodes is a bipartition of a graph
X = set(['A', 'B', 'C', 'D', 'E'])
bipartite.is_bipartite_node_set(B,X) #True
#Getting each set of nodes of a bipartite graph
bipartite.sets(B) #Out: ({'A', 'B', 'C', 'D', 'E'}, {1, 2, 3, 4})
```



Projected Graphs

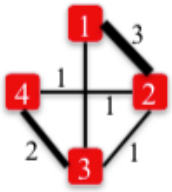
L-Bipartite graph projection: Network of nodes in group L, where a pair of nodes is connected if they have a common neighbor in R in the bipartite graph.

```
B = nx.Graph()
B.add_edges_from([('A',1), ('B',1),
('C',1),('D',1),('H',1), ('B', 2),
('C', 2), ('D', 2),('E', 2), ('G', 2),
('E', 3), ('F', 3), ('H', 3), ('J', 3),
('E', 4), ('I', 4), ('J', 4) ])
X = set(['A','B','C','D', 'E', 'F','G', 'H', 'I','J'])
P = bipartite.projected_graph(B, X)
```



- ***L*-Bipartite weighted graph projection:** An *L*-Bipartite graph projection with weights on the edges that are proportional to the number of common neighbours between the nodes.

```
X = set([1,2,3,4])
P = bipartite.weighted_projected_graph(B, X)
```



Weighted Network of teams who have a fan common

Loading graphs

Adjacency list

each row has the following format: source-node node_x node_y node_n, where node_x, node_y, etc are all nodes connected to the source node. Note that adjacencies are only accounted for once (e.g. node 2 is adjacent to node 0, but node 0 is not listed in node 2's row, because that edge has already been accounted for in node 0's row).

```
0 1 2 3 5 # node 0 is adjacent to nodes 1, 2, 3, 5
1 3 6 # node 1 is (also) adjacent to nodes 3, 6
2 # node 2 is (also) adjacent to no new nodes
3 4 # node 3 is (also) adjacent to node 4
G2 = nx.read_adjlist('G_adjlist.txt', nodetype=int)
```

Adjacency matrix

The elements indicate whether pairs of vertices are adjacent or not in the graph. Each node has a corresponding row and column. For example, row `0`, column `1` corresponds to the edge between node `0` and node `1`. Reading across row `0`, there is a '`1`' in columns `1`, `2`, `3`, and `5`, which indicates that node `0` is adjacent to nodes 1, 2, 3, and 5.

```
G_mat = np.array([[0, 1, 1, 1, 0, 1, 0, 0, 0, 0],
                  [1, 0, 0, 1, 0, 0, 1, 0, 0, 0],
                  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [1, 1, 0, 0, 1, 0, 0, 0, 0, 0],
                  [0, 0, 0, 1, 0, 1, 0, 1, 0, 0],
                  [1, 0, 0, 0, 1, 0, 0, 0, 1, 0],
                  [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
                  [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]])
G3 = nx.Graph(G_mat)
```

Edgelist

Represents edge pairings in the first two columns. Additional edge attributes can be added in subsequent columns. Looking at `G_edgelist.txt` this is the same as the original graph `G1`, but now each edge has a weight. Not good for isolated nodes

For example, from the first row, we can see the edge between nodes `0` and `1`, has a weight of `4`.

```
0 1 4
0 2 3
0 3 2
0 5 6
1 3 2
1 6 5
3 4 3
4 5 1
4 7 2
5 8 6
8 9 1
G4 = nx.read_edgelist('G_edgelist.txt', data=[('Weight', int)])
G4.edges(data=True)
[('0', '1', {'Weight': 4}),
 ('0', '2', {'Weight': 3}),
 ('0', '3', {'Weight': 2}),
 ('0', '5', {'Weight': 6}),
 ('1', '3', {'Weight': 2}),
 ('1', '6', {'Weight': 5}),
 ('3', '4', {'Weight': 3}),
 ('5', '4', {'Weight': 1}),
 ('5', '8', {'Weight': 6}),
 ('4', '7', {'Weight': 2}),
 ('8', '9', {'Weight': 1})]
```

Pandas DataFrame

Graphs can also be created from pandas dataframes if they are in edge list format.

```
G5 = nx.from_pandas_dataframe(G_df, 'n1', 'n2', edge_attr='weight')
```