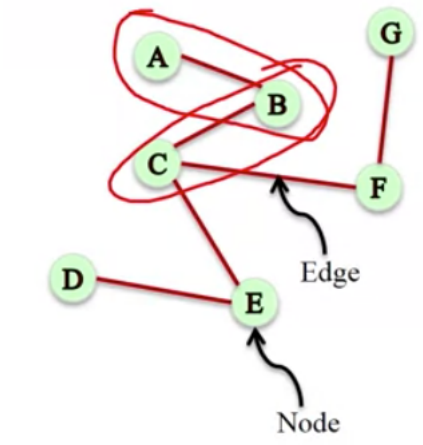
Michigan Applied Social Network Analysis in Python

Lecture 1. Network and Basics (networkx)

1. Definition

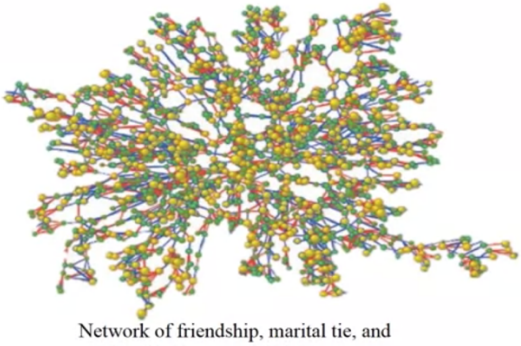
- Network (or Graph): A representation of connections among a set of items
- Items are called nodes (or vertices)
- Connections are called edges (or links /ties)
- Adding edges in Python:

```
Import networkx as nx

G = nx.Graph()

G.add_edge('A','B')

G.add_edge('B','C')
```

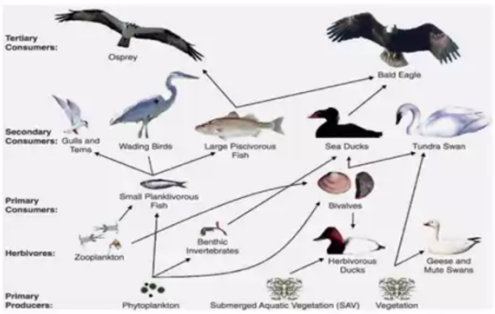


2. Cases of network:

- Types
- Symmetric and Asymmetric: what eats what

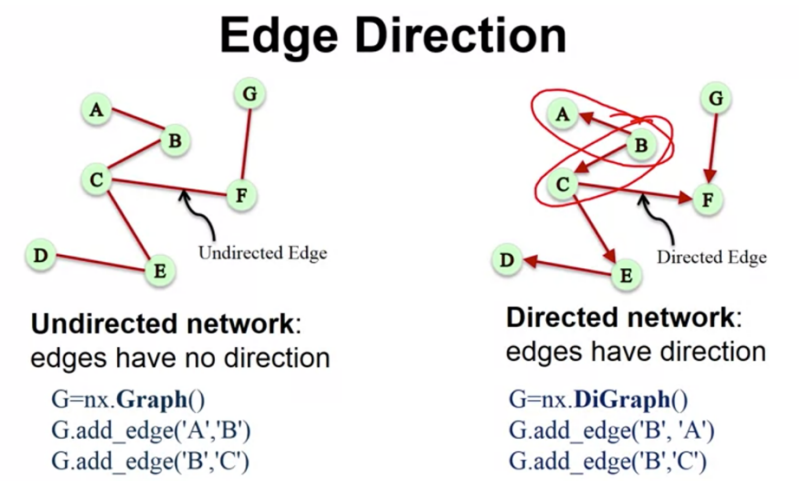


Network of friendship, marital tie, and family tie among 2200 people

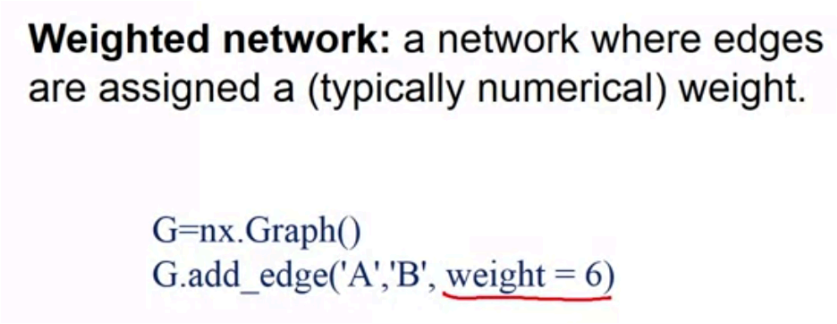


Chesapeake Bay Water bird Food Web

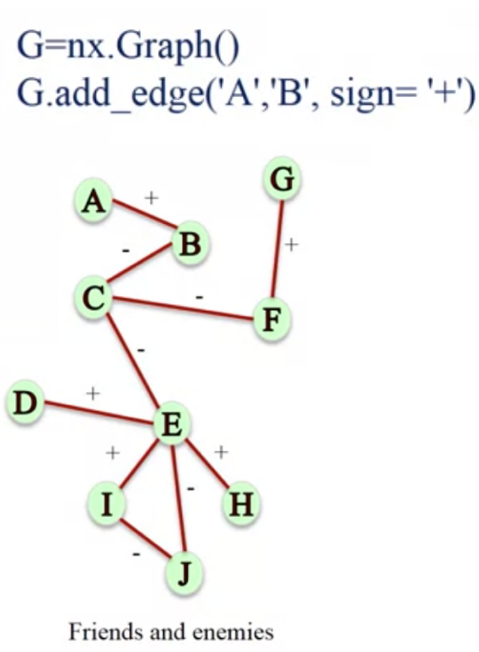
- Edge Direction



- Weighted Networks: some edges carry higher weight than others

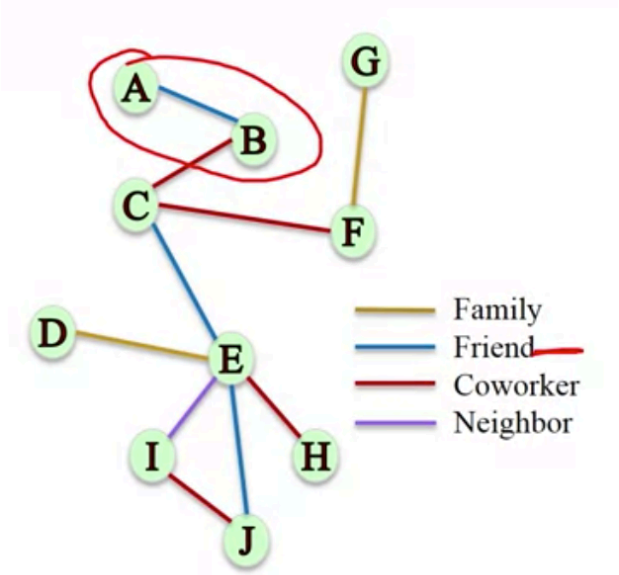


- Signed Networks: a network where edges are assigned positive or negative sign



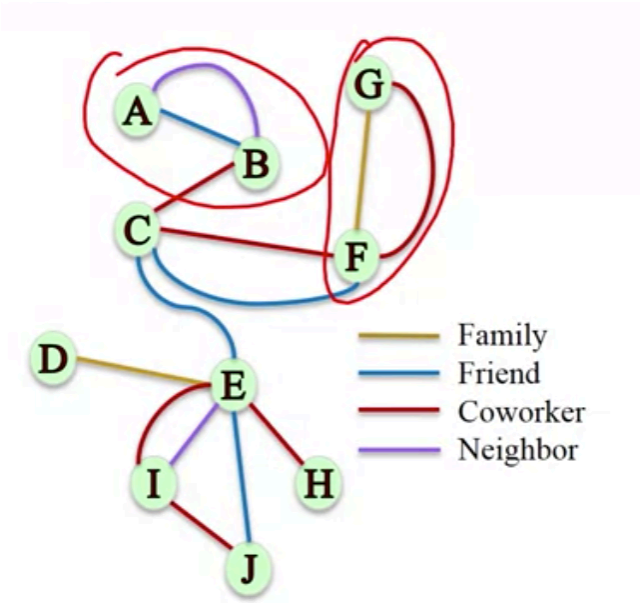
- Other edge attributes

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



- Multigraphs: a network where multiple edges can connect the same nodes (parallel edges)

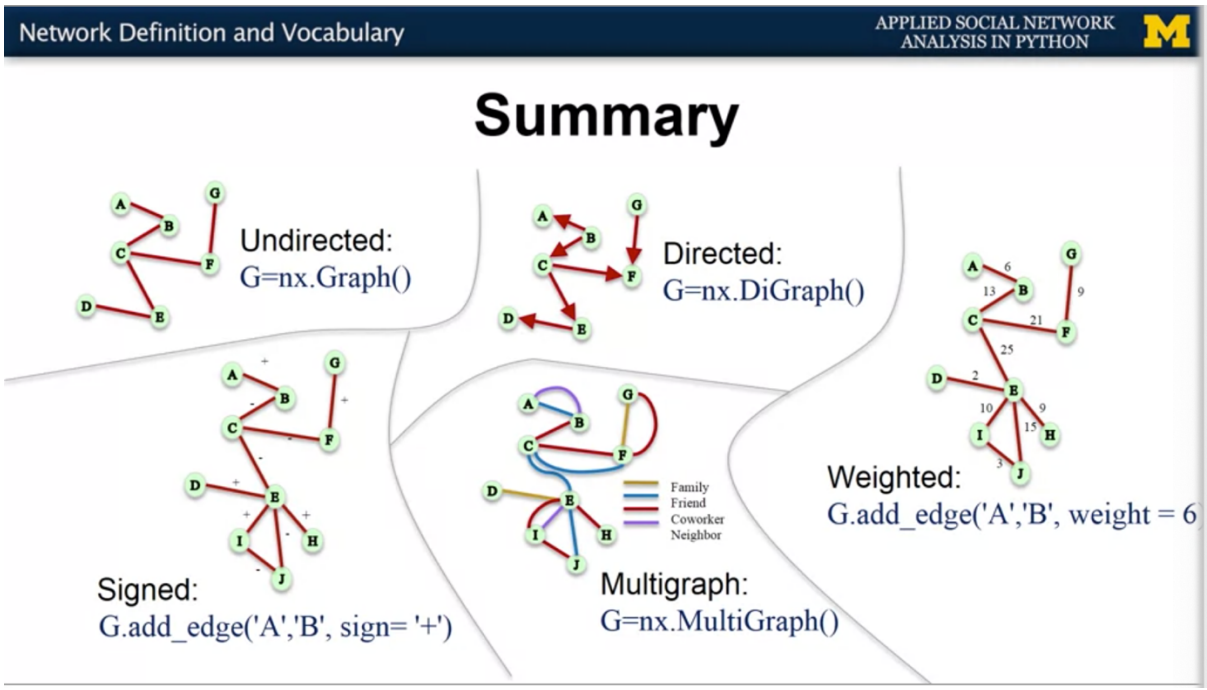
```
G=nx.MultiGraph()
G.add_edge('A','B', relation= 'friend')
G.add_edge('A','B', relation= neighbor')
```



- Exercise: We would like to construct a graph on NetworkX, where the nodes represent employees of a company and the edges represent the number of times an employee sent an email to another employee. What would be the best way to represent this network?

  Solution: Weighted, directed graph since we want to capture who sent the email and who received it, also we want to capture the number of times an employee emailed another, thus we want the edges to have weights.

3. SUMMARY



Lecture 2. Node and Edge Attributes

1. Access edge attributes

- Simple access to edge attributes in a nx.graph()

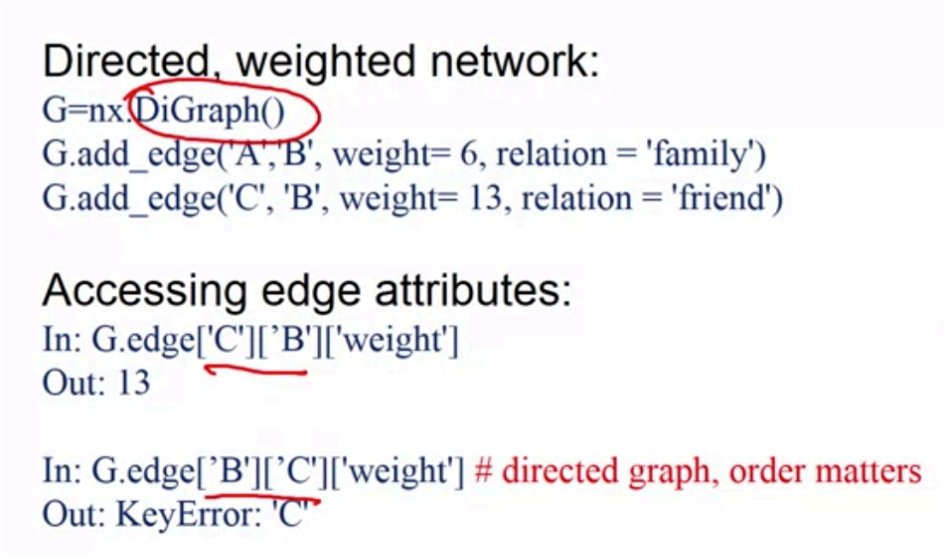
```
G=nx.Graph()
G.add_edge('A','B', weight= 6, relation = 'family')
G.add_edge('B','C', weight= 13, relation = 'friend')

In: G.edges() #list of all edges
Out: [('A', 'B'), ('C', 'B')]

In: G.edges(data= True)  #list of all edges with attributes
Out: [('A', 'B', {'relation': 'family', 'weight': 6}),
('C', 'B', {'relation': 'friend', 'weight': 13})]

In: G.edges(data= 'relation') #list of all edges with attribute 'relation'
Out: [('A', 'B', 'family'), ('C', 'B', 'friend')]
```

- Attributes to a specific edge

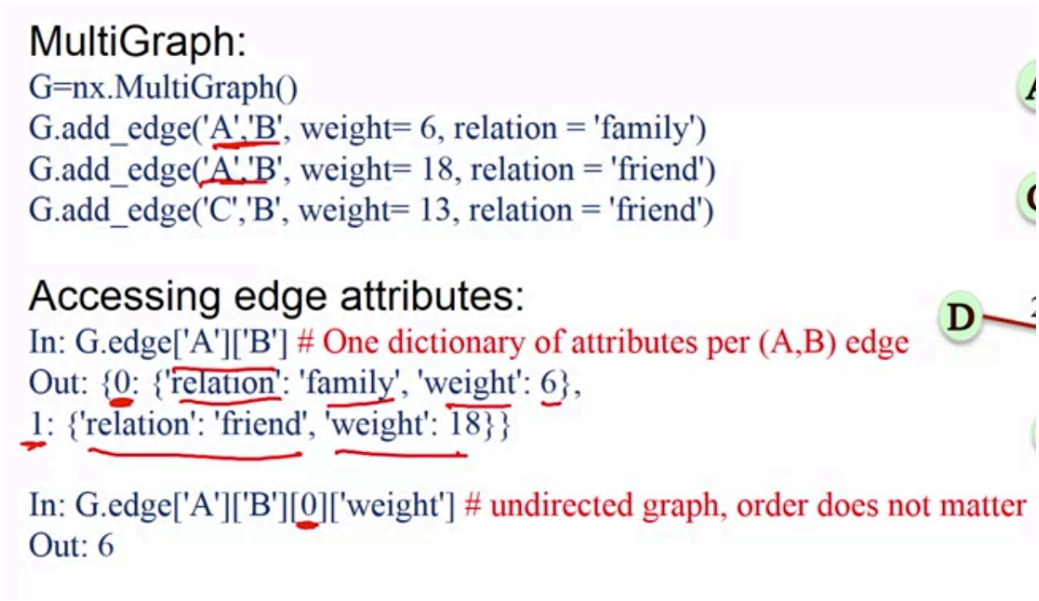
- Undirected network

```
In: G.edge['A']['B'] # dictionary of attributes of edge (A, B)
Out: {'relation': 'family', 'weight': 6}

In: G.edge['B']['C']['weight']
Out: 13

In: G.edge['C']['B']['weight'] # undirected graph, order does not matter
```

- Directed weighted network



- MultiGraph:



Directed MultiGraph:

```
G=nx.MultiDiGraph()
G.add_edge('A','B', weight= 6, relation = 'family')
G.add_edge('A','B', weight= 18, relation = 'friend')
G.add_edge('C','B', weight= 13, relation = 'friend')
```

Accessing edge attributes:

```
In: G.edge['A']['B'][0]['weight']
Out: 6

In: G.edge['B']['A'][0]['weight'] # directed graph, order matters
Out: KeyError: 'A'
```

Exercise:

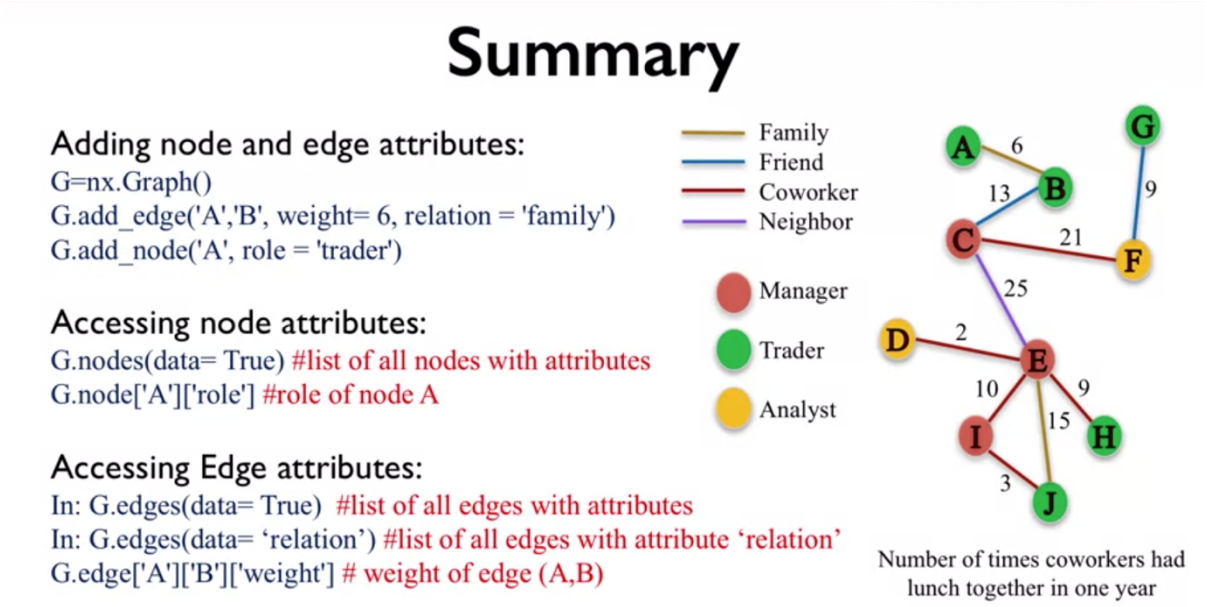
```
1  import networkx as nx
2
3  G=nx.MultiDiGraph()
4
5  G.add_edge('John', 'Ana', weight= 3, relation =
       'siblings')
6  G.add_edge('Ana', 'David', weight= 4, relation =
       'cousins')
7  G.add_edge('Ana', 'Bob', weight= 1, relation =
       'friends')
8  G.add_edge('Ana', 'Bob', weight= 1, relation =
       'neighbors')
9
10 print( G.edge['Bob']['Ana'][1]['relation'] )
11
```

Solution: KeyError:'Ana' (Since G is a directed graph)

2. Node Attribute

```
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'
```

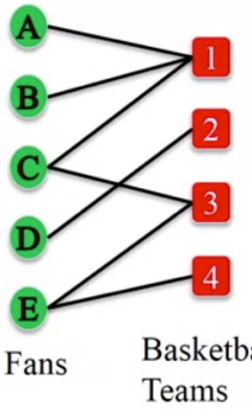
3. SUMMARY



Lecture 3. Bipartite Graphs

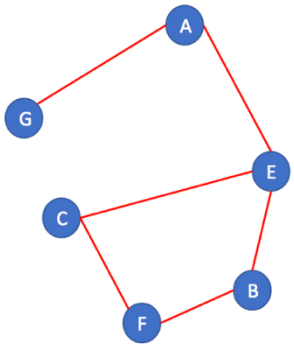
1. Bipartite Graph

- Definition: a graph whose nodes can be split into two sets L and R and very edge connects an node in L with a node in R

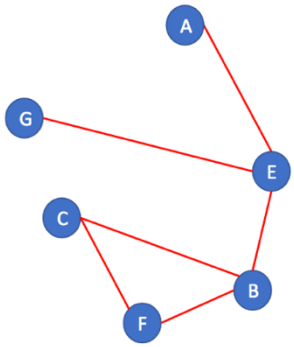



```
from networkx.algorithms import bipartite
B = nx.Graph() #No separate class for bipartite graphs
B.add_nodes_from(['A','B','C','D', 'E'], bipartite=0) #label one set of nodes 0
B.add_nodes_from([1,2,3,4], bipartite=1) #label other set of nodes 1
B.add_edges_from([('A',1), ('B',1), ('C',1), ('C',3), ('D',2), ('E',3), ('E', 4)])
```

- A bipartite graph cannot contain a cycle of an odd number of nodes.
- Exercise: B is not a bipartite graph

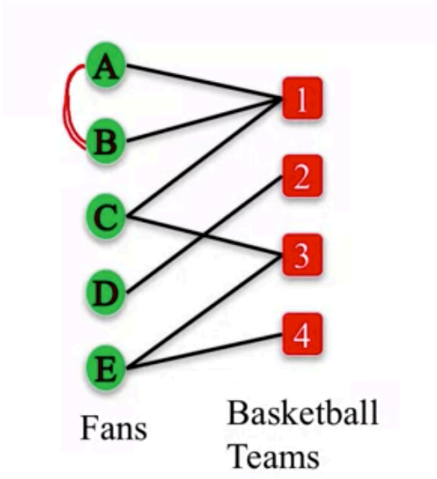


Graph A



Graph B

- Check if a graph is bipartite




```
In: bipartite.is_bipartite(B) # Check if B is bipartite
Out: True

In: B.add_edge('A', 'B')
In: bipartite.is_bipartite(B)
Out: False

B.remove_edge('A', 'B')
```

- Check if a set of nodes is a bipartition of a graph

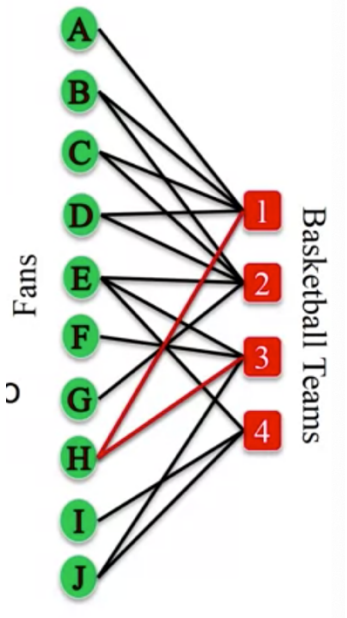
```
In: X = set([1,2,3,4])
In: bipartite.is_bipartite_node_set(B,X)
Out: True

X = set(['A', 'B', 'C', 'D', 'E'])
In: bipartite.is_bipartite_node_set(B,X)
Out: True

X = set([1,2,3,4, 'A'])
In: bipartite.is_bipartite_node_set(B,X)
Out: False
```

- Getting each set of nodes of a bipartite graph

```
In: bipartite.sets(B)
Out: ({'A', 'B', 'C', 'D', 'E'}, {1, 2, 3, 4})

In: B.add_edge('A', 'B')
In: bipartite.sets(B)
Out: NetworkXError: Graph is not bipartite.
```

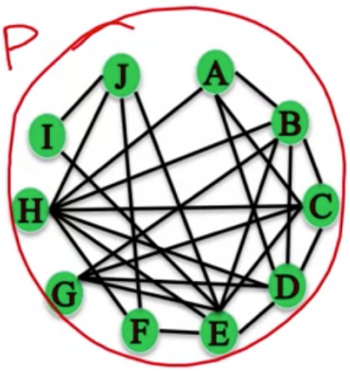
2. Projected Graph

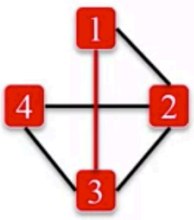
- L-Bipartite graph projection: Network of nodes in group L, where a pair of nodes is connected if they have a common neighbor in R in the bipartite graph.
- Similar definition applies for R-Bipartite graph projection



-
```
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)
```

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

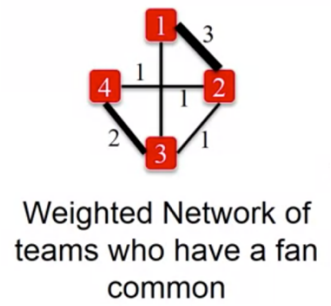




Network of teams who have a fan common

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

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



Weighted Network of teams who have a fan common

- Exercise:

What is the weight of the edge (A,C)?

```
1  import networkx as nx
2  from networkx.algorithms import bipartite
3
4  B = nx.Graph()
5  B.add_edges_from([('A', 'G'),('A','I'), ('B','H'),
       ('C', 'G'), ('C', 'I'),('D', 'H'), ('E', 'I'),
       ('F', 'G'), ('F', 'J')])
6  X1 = set(['A', 'B', 'C', 'D', 'E', 'F'])
7
```

Run

Reset

Solution: 2 , A and C share {I, G}

3. SUMMARY