

Basic graph manipulation		Bipartite graphs (cont)	
<code>import networkx as nx</code>		<code>bipartite.sets(B)</code>	Get each set of nodes of bipartite graph
<code>G=nx.Graph()</code>			
<code>G=nx.MultiGraph()</code>	Create a graph allowing parallel edges	<code>bipartite.projected_graph(B, X)</code>	Bipartite projected graph - nodes with bipartite friends in common
<code>G.add_edges_from([(0, 1), (0, 2), (1, 3), (2, 4)])</code>	Create graph from edges		
<code>nx.draw_networkx(G)</code>	Draw the graph	<code>P=bipartite.weighted_projected_graph(B, X)</code>	projected graph with weights (number of friends in common)
<code>G.add_node('A', role='manager')</code>	Add a node		
<code>G.add_edge('A', 'B', relation='friend')</code>	Add an edge		
<code>G.nodes['A']['role'] = 'team member'</code>	Set attribute of a node		
<code>G.nodes['A'], G.edges [('A', 'B')]</code>	View attributes of node, edge		
<code>G.edges(), G.nodes()</code>	Show edges, nodes		
<code>list(G.edges())</code>	Return as list instead of EdgeView class		
<code>G.nodes(data=True), G.edges(data=True)</code>	Include node/edge attributes		
<code>G.nodes(data='relation')</code>	Return specific attribute		

### Network Connectivity

### Creating graphs from data

<code>G=nx.read_adjlist('G_adjlist.txt', nodetype=int)</code>	<code>nx.clusters</code>	<b>Create</b> from adjacency list	Local clustering coefficient
<code>G=nx.Graph(G_mat)</code>	<code>nx.transitivity(G)</code>	<b>Create</b> from matrix ( <code>np.array</code> )	Global clustering coefficient
<code>G=nx.read_edgelist('G_edgelist.txt', data=[('weight', int)])</code>	<code>nx.shortest_path(G, n1, n2)</code>	<b>Create</b> from matrix ( <code>np.array</code> )	Transitivity (% of open triads)
<code>G=nx.from_pandas_dataframe(G_df, 'n1', 'n2', edge_attr='weight')</code>	<code>nx.shortest_path_length(G, n1, n2)</code>	<b>Create</b> from edgelist	Outputs the path itself
	<code>T=nx.bfs_tree(G, n1)</code>	<b>Create</b> from df	Create breadth-first search tree from node n1
<b>Adjacency list format</b> 0 1 2 3 5 1 3 6 ...	<code>nx.average_shortest_path_length(G)</code>		Average distance between all pairs of nodes
<b>Edgelist format:</b> 0 1 14 0 2 17	<code>nx.diameter(G)</code>		Maximum distance between any pair of nodes
<b>Bipartite graphs</b>	<code>nx.eccentricity(G)</code>		Returns each node's distance to furthest node
<code>from networkx.algorithms import bipartite</code>	<code>nx.radius(G)</code>		Minimum eccentricity in the graph
<code>bipartite.is_bipartite(B)</code>	<code>nx.periphery(G)</code>		Set of nodes where eccentricity=diameter
<code>bipartite.is_bipartitenode_set(B, set)</code>	<code>nx.center(G)</code>		Set of nodes where eccentricity=radius

Connectivity: Network Robustness	
<code>nx.node_connectivity(G)</code>	Min nodes removed to disconnect a network
<code>nx.minimum_node_cut()</code>	Which nodes?
<code>nx.edge_connectivity(G)</code>	Min edges removed to disconnect a network
<code>nx.minimum_edge_cut(G)</code>	Which edges?
<code>nx.all_simple_paths(G, n1, n2)</code>	Show all paths between two nodes



By **RJ Murray** (murenei)  
[cheatography.com/murenei/tutify.com.au](http://cheatography.com/murenei/tutify.com.au)

Published 4th June, 2018.  
 Last updated 4th June, 2018.  
 Page 1 of 5.

Sponsored by **ApolloPad.com**  
 Everyone has a novel in them. Finish Yours!  
<https://apollopad.com>

### Network Connectivity: Connected Components

<code>nx.is_connected(G)</code>	Is there a path between every pair of nodes?
<code>nx.number_connected_components(G)</code>	# separate components
<code>nx.node_connected_component(G, N)</code>	Which connected component does <i>N</i> belong to?
<code>nx.is_strongly_connected(G)</code>	Is the network connected directionally?
<code>nx.is_weakly_connected(G)</code>	Is the directed network connected if assumed undirected?

### Common Graphs

<code>G=nx.karate_club_graph()</code>	Karate club graph (social network)
<code>G=nx.path_graph(n)</code>	Path graph with <i>n</i> nodes
<code>G=nx.complete_graph(n)</code>	Complete graph on <i>n</i> nodes
<code>G=random_regular_graph(d,n)</code>	Random <i>d</i> -regular graph on <i>n</i> -nodes

See NetworkX Graph Generators reference for more.  
Also see "An Atlas of Graphs" by Read and Wilson (1998).

### Influence Measures and Network Centralization

### Influence Measures and Network Centralization (cont)

<code>nx.edge_betweenness_centrality(G)</code>	Edge betweenness centrality
<code>nx.edge_betweenness_centrality_subset(G, {subset})</code>	Edge betweenness centrality for a subset of nodes

Normalization: Divide by number of pairs of nodes.

### PageRank and Hubs & Authorities Algorithms

<code>nx.pagerank(G, alpha=0.8)</code>	Scaled PageRank of <i>G</i> with dampening parameter
<code>h,a=nx.hits(G)</code>	HITS algorithm - outputs 2 dictionaries (hubs, authorities)

<code>h,a=nx.hits(G,max_iter=10,norm=True)</code>	Constrained HITS and normalized by sum at each stage
---	--

Centrality measures make different assumptions about what it means to be a "central" node. Thus, they produce different rankings.

### Network Evolution - Real-world Applications

<code>G.degree(), G.in_degree(), G.out_degree()</code>	Distribution of node degrees
--	------------------------------

**Preferential Attachment Model**  
Results in power law -> many nodes with low degrees; few with high degrees

<code>dc=nx.degree_centrality(G)</code>	Degree centrality for network	$G = \text{barabasi\_albert\_graph}(n, m)$	Preferential Attachment Model with
<code>dc[node]</code>	Degree centrality for a node		$n$ nodes and each new node attaching to $m$ existing nodes
<code>nx.in_degree_centrality(G), nx.out_degree_centrality(G)</code>	DC for directed networks		
<code>cc=nx.closeness_centrality(G, normalized=True)</code>	Closeness centrality (normalised) for the network	Small World model	High average degree (global clustering) and low average shortest path
<code>cc[node]</code>	Closeness centrality for an individual node		
<code>bC=nx.betweenness_centrality(G)</code>	Betweenness centrality	$G = \text{watts\_strogatz\_graph}(n, k, p)$	Small World network of $n$ nodes, connected to its $k$ nearest neighbours, with chance $p$ of rewiring
<code>..., normalized=True, ...)</code>	Normalized betweenness centrality		
<code>..., endpoints=False, ...)</code>	BC excluding endpoints		
<code>..., K=10, ...)</code>	BC approximated using random sample of $K$ nodes	$G = \text{connected\_watts\_strogatz\_graph}(n, k, p, t)$	$t = \text{max iterations to try to ensure connected graph}$
<code>nx.betweenness_centrality_subset(G, {subset})</code>	BC calculated on subset	$G = \text{newman\_watts\_strogatz\_graph}(n, k, p)$	$p = \text{probability of adding (no rewiring)}$
		Link Prediction measures	How likely are 2 nodes to connect, given an existing network



### Network Evolution - Real-world Applications (cont)

<code>nx.common_neighbors(G, n1, n2)</code>	Calc common neighbors of nodes <i>n1</i> , <i>n2</i>
<code>nx.jaccard_coefficient(G)</code>	Normalised common neighbors measure
<code>nx.resource_allocation_index(G)</code>	Calc RAI of all nodes not already connected by an edge
<code>nx.adamic_adar_index(G)</code>	As per RAI but with log of degree of common neighbor
<code>nx.preferential_attachment(G)</code>	Product of two nodes' degrees
Community Common Neighbors	Common neighbors but with bonus if they belong in same 'community'
<code>nx.cn_soundarajan_hopcroft(n1, n2)</code>	CCN score for <i>n1</i> , <i>n2</i>
<code>G.node['A']['community']=1</code>	Add community attribute to node
<code>nx.ra_index_soundarajan_hopcroft(G)</code>	Community Resource Allocation score

These scores give only an indication of whether 2 nodes are likely to connect.

To make a link prediction, you would use these scores as features in a classification ML model.



By **RJ Murray** (murenei)  
[cheatography.com/murenei/tutify.com.au](https://cheatography.com/murenei/tutify.com.au)

Published 4th June, 2018.  
Last updated 4th June, 2018.  
Page 3 of 5.

Sponsored by **ApolloPad.com**  
Everyone has a novel in them. Finish Yours!  
<https://apollopad.com>