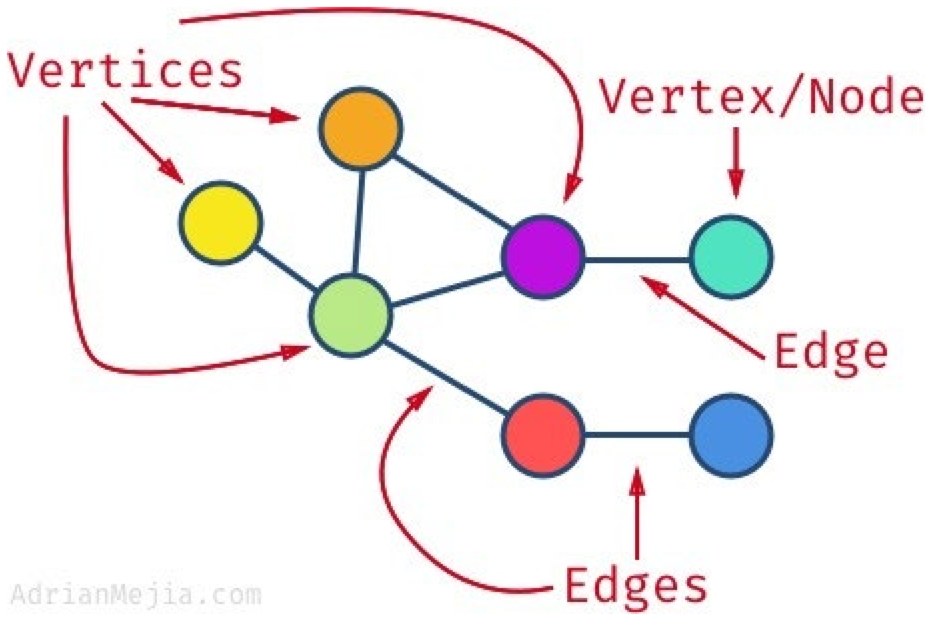
NetworkX
Network Analysis in Python

Working with Network Data

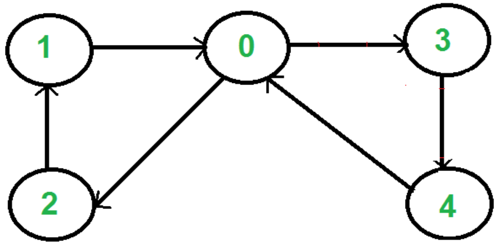
LECTURE 7

Organization of Lecture 7

- Dissecting Graphs
- Network Analysis Sequence
- Harnessing Networkx

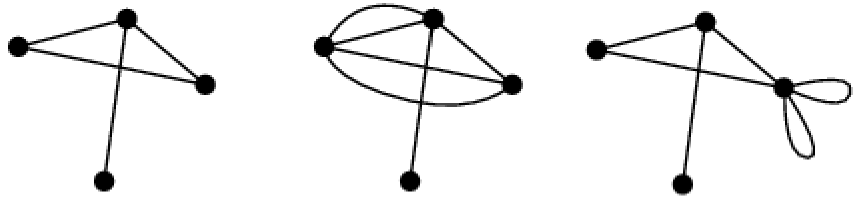


Graph Terminology

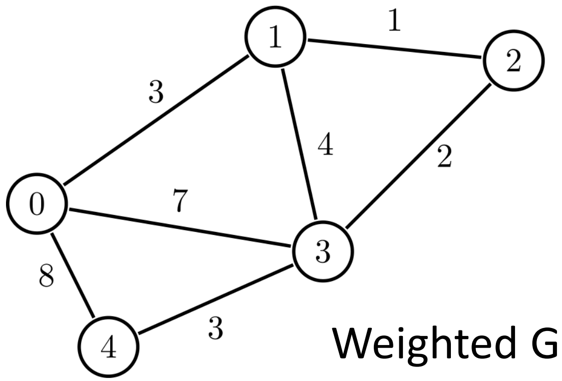


Directed Graph

- Graph – set of nodes connected with edges.
- Directed Graph – edges have arrows
- Multigraph – node(s) can be connected by more than one edge
- Simple Graph – every pair of nodes connected by 0 or one edge, no loops
- Weighted Graph – Edges have a weight assigned to them



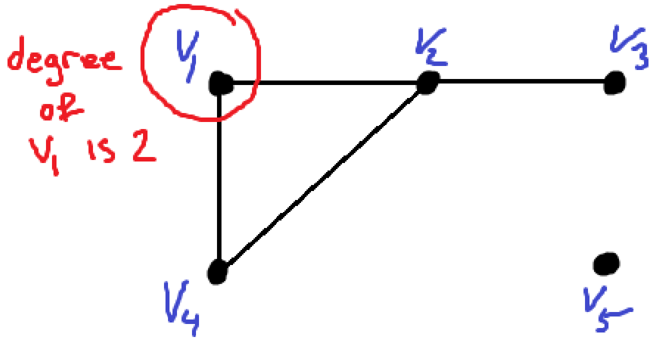
simple graph *multigraph* *pseudograph*



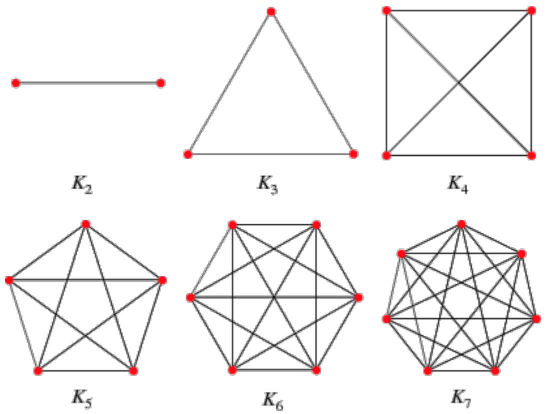
Weighted Graph

More Terminology

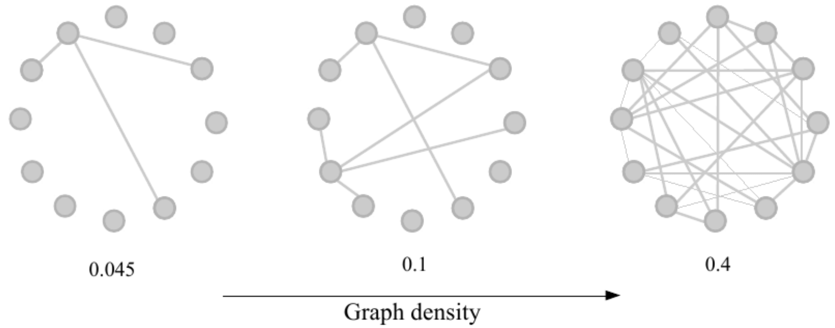
- Degree(node) - # edges connected to it
  - Directed graph nodes have in and out degrees



- Complete Graph - each pair of graph vertices is connected by an edge

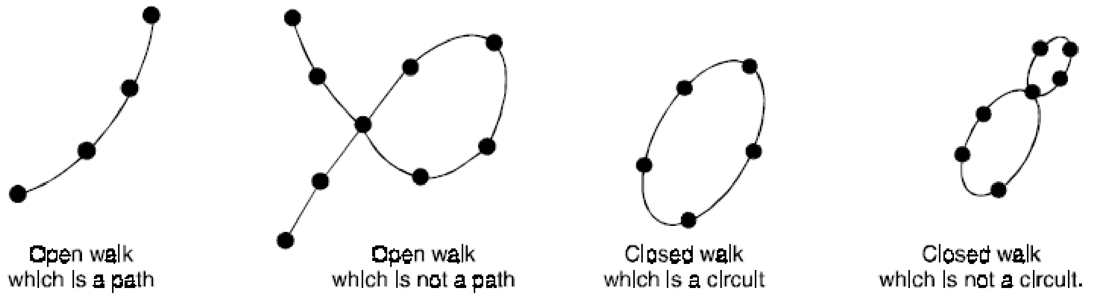


- Graph Density – how close graph is to a complete graph
  - d = number of edges / total possible edges
  - Directed graph   d =   e / n(n-1)
  - Simple graph   d = 2e/n(n-1)



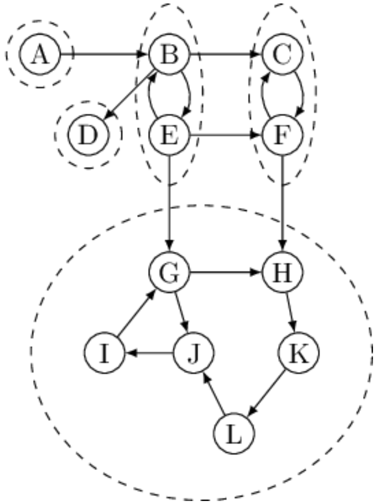
Graph Structure - Paths

- Walk – sequence of edges (where end of one edge is beginning of next)
- Path – walk that doesn't include same node twice (except for initial/last)
- Distance – number of edges in a path
  - Largest distance between two nodes is called diameter

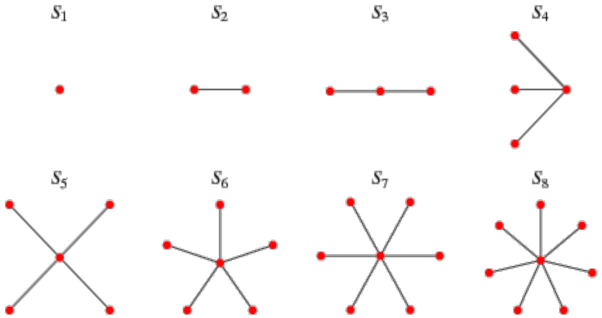


Graph Structure

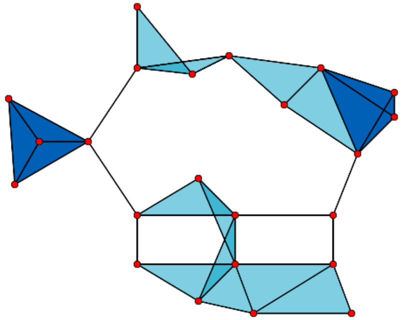
- Connected component – set of nodes where there is a path from each node to all the other nodes
  - *giant connected component (GCC)* - Largest connected component
  - *Bridge* – edge where if it is removed, the graph becomes disconnected



- Star Graph – one node connects all other nodes in the graph

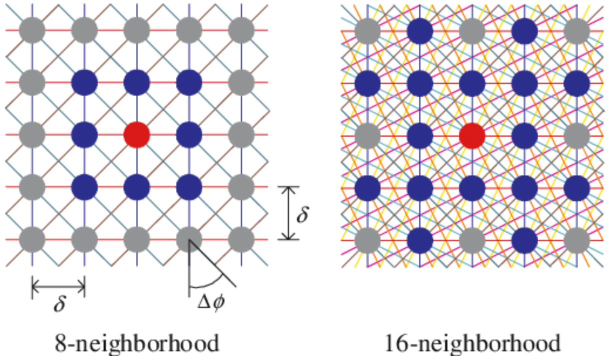


- *Clique – set of nodes,* such that each is directly connected to each other node in the set.
  - *Maximum clique* – largest clique in the graph

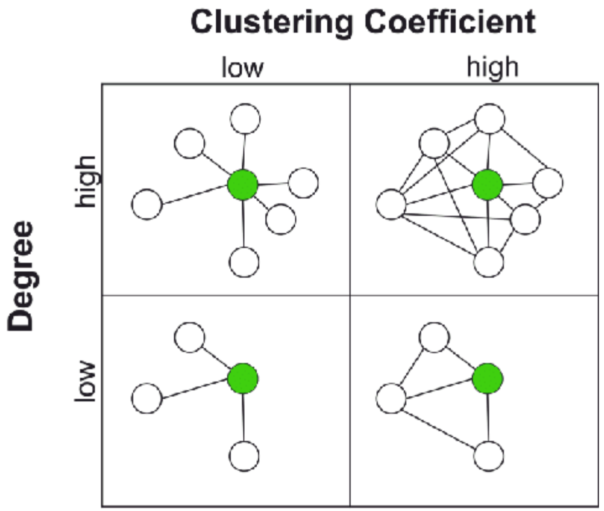


Graph Structure - Neighborhood

- Neighborhood – set of nodes directly connected to a particular node
  - Key part of *snowballing* – start with seed node and then propagate to neighbors

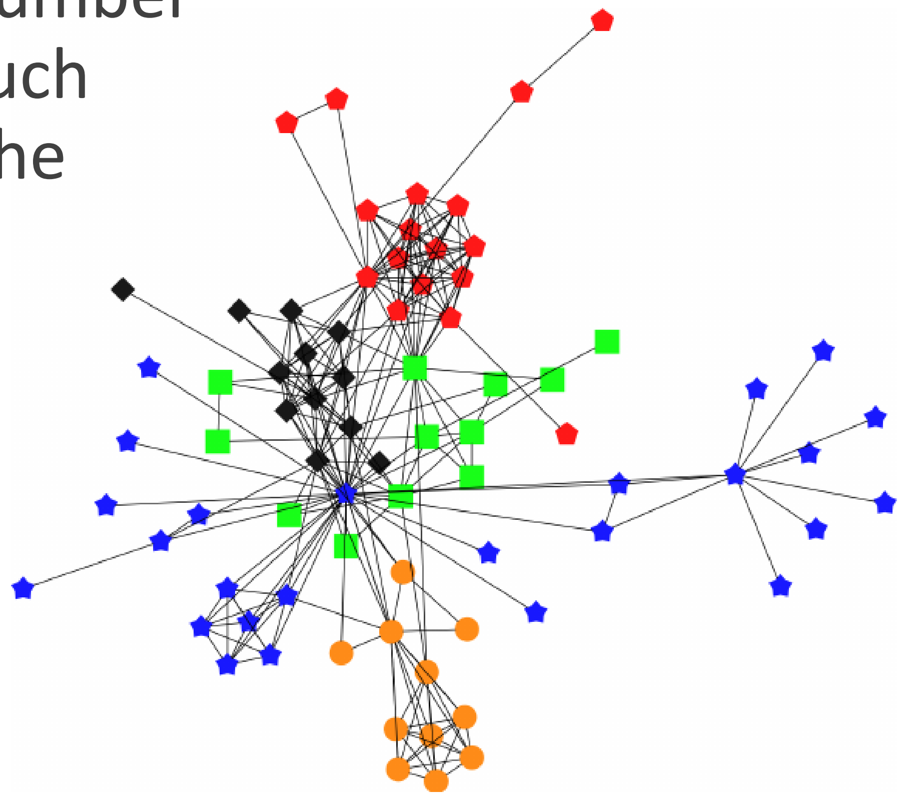


- Local clustering coefficient of a node *a* –
  - CC(node *a*) = #edges in a's neighborhood (excluding the edges directly connected to *a) /* max possible #edges
  - CC(node *a*) = density of neighborhood of A without node A itself
  - CC(node in a star) = 0
  - CC (Kn) = 1



Graph Structure - Community

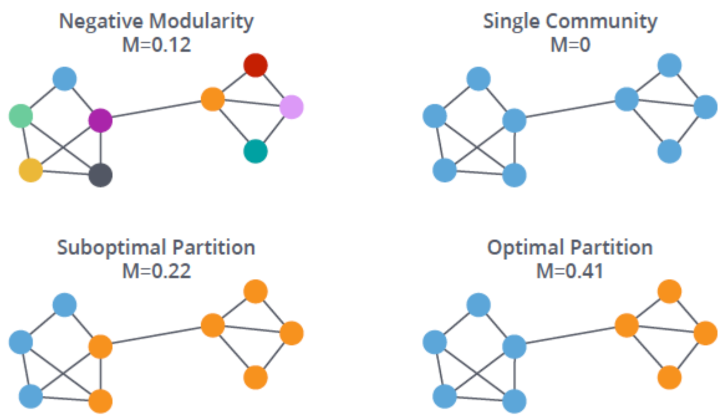
- Community – set of nodes, such that the number of edges interconnecting these nodes is much larger than the number of edges crossing the community boundary.



Les Miserables" graph

Graph Structure - Modularity

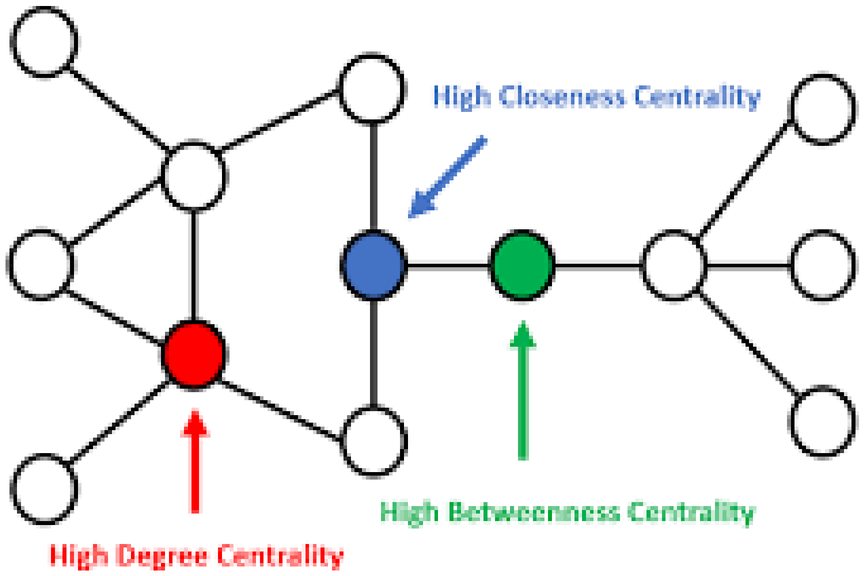
- Modularity [-1/2, 1] – measures quality of community structure
  - Modularity – fraction of edges that fall within the community – expected such fraction, if edges were distributed at random
  - High modularity (m approx. 1) – network with dense and clearly visible communities.
  - VERY IMPORTANT in NETWORK ANALYSIS SEQUENCE



Modularity

Graph Structure - Centralities

- *Centrality* – several different measures of the importance of node in a network
- Can be based on:
  - *Degree of A* – number of neighbors of A/total possible (scaled between 0 and 1)
  - *Closeness of A* – reciprocal of average shortest path from all other nodes to A
  - *Betweenness of A* – fraction of all shortest paths between all pairs of all nodes in the network, excluding A, that contain A



Network Analysis Sequence

- Consists of Steps
  1. Identify discrete entities (nodes) and relations between them (edges).
     - Binary relations (1-edge, 0-no edge); continuous or discrete but not binary (weights)
     - May need to be above a certain threshold: too high - graph may get disconnected, too low – tangled
  2. Calculate network measures (density, number of components, GCC size, diameter, centralities, clustering coefficients, etc.
  3. Identify network communities
     - Modular – high modularity - assign labels to communities and replace with super nodes
  4. Interpret results and produce report with lots of pictures
     - Networkx gives pathetic pictures, doesn’t work in replit.
     - Gephi.org for visualization

Research Application

Component fault-prone relationships -

In Release 2, there are 7 fault-prone relationships involving 10 fault prone components as determined by multi-file defect coupling measure and a cumulative defect coupling measure.

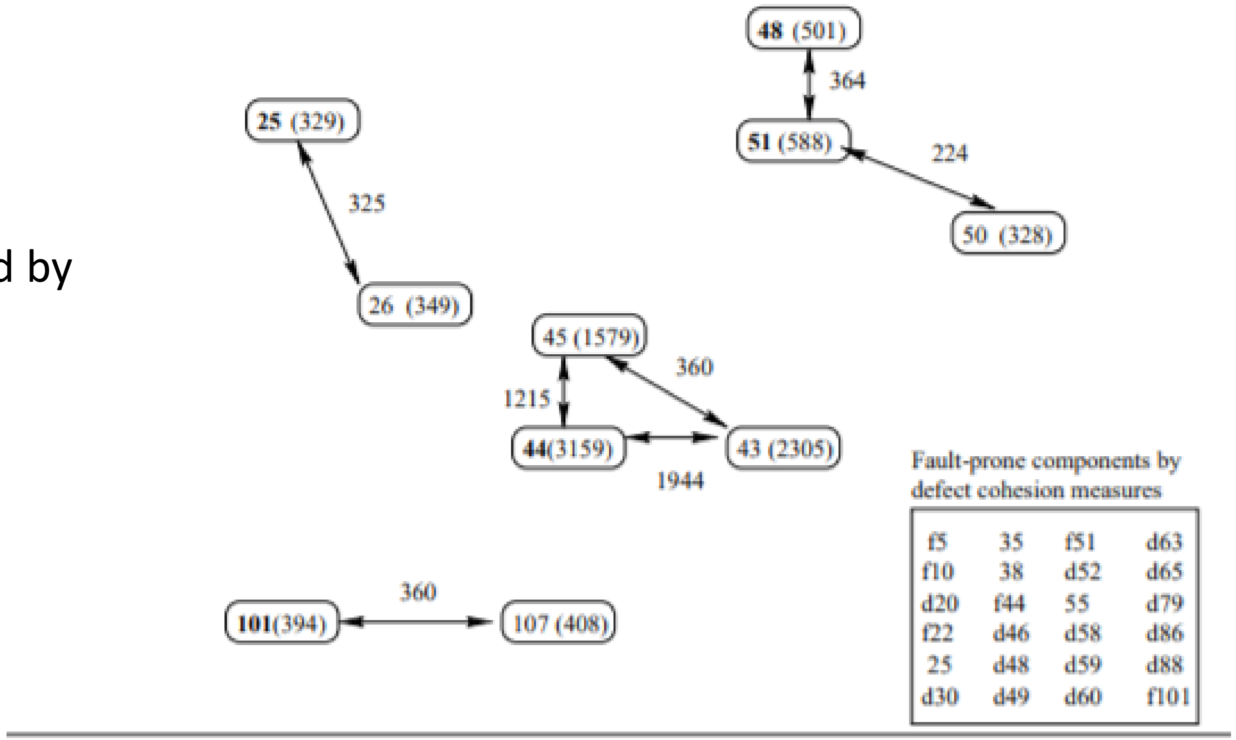


Figure 4: Release 2 Component Level Fault Architecture.

https://cs.msutexas.edu/~stringfellow/papers/faultarch.pdf

Harnessing Networkx

- Networkx modules contains tools for creating, modifying, exploring, plotting, exportin, and importin networks.
- Supports simple, directed and multi – graphs.
- You can nodes, edges and attributes
- You can calculate network measures
- You can explore the structure of a graph

Building and Fixing a Network Example

- Use Wikipedia data to construct and explore network of nation states (based on land borders).
- Graph is undirected –if country A borders B, then B borders A.
- Graph has no loops – country cannot border itself
- Graph has no multi-edges – A borders B only once.

- Importing a table in Wiki to Python

Borders Example Result of Visualization

Coding Basics

- `import network as nx`
- `graph_obj = nx.Graph()   #creates an empty graph`
- graph_obj.add_node(“*node_label*”)
- graph_obj.add_nodes_from *([some list])*
- graph_obj.remove_node(“*node_label*”)
- graph_obj.add_edge(“*initial node*”, “*final node*”)
- graph_obj.add_edges_from([*list of pairs])*

Analyzing Graphs

- graph_obj.degree(“*node_label*”)
- graph_obj.indegree (“*node_label*”)
- graph_obj.outdegree (“*node_label*”)
- #returns dictionary of degrees indexed by node labels
  graph_obj.degree()

Analyzing Graphs

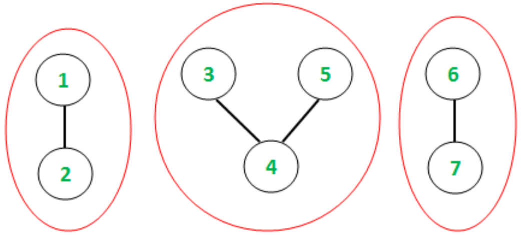
- Get nodes with highest degree / incorrect syntax in text

```
degrees = pandas.DataFrame(list(graph_obj.degree()),            \
                  columns=(“index_name", "degree")).set_index(“index_name")
print(“Nodes with highest degrees are ",   degrees.sort_values("degree").tail(5))
```

Clustering

- NetworKit – efficient parallizable network analysis toolkit
  - (e.g. for 2 billion nodes for Facebook's social graph)
  - Now integrated with matplotlib, scipy, numpy, pandas, and networkx
  - Just saying…
- Function `clustering`() returns a dictionary of all clustering coefficients
  - `nx.clustering(graph_obj))`
- Clustering coefficient is undefined for *directed* graphs, but you can turn a directed graph into an undirected graph if needed.
  - `nx.clustering(nx.Graph(directed_graph_obj))`

Connected Components



The **counts** of connected components are - **2, 3 and 2**

- `Connected_components()` returns a *list generator* of respective connected components (as node label lists) from the graph.
  - You can use the generator in an iterator expression (loop or list comprehension) or convert it to a list

```
list(nx.connected_components(image_graph))
           #returns  [{1, 2}  {3, 4, 5}, {6, 7}]

 [len(x) for x in nx.connected_component_subgraphs(image_graph)]
           #returns [2, 3, 2]
```

Centrality

- Centrality functions return either a dictionary of centralities, indexed by node labels, or individual node centralities.
  - Good building blocks for data frames and indexed series.

  - nx.degree_centrality(*graph_obj)* #might return China if looking at borders
  - nx.in_degree_centrality (*graph_obj)*
  - nx.out_degree_centrality (*graph_obj)*
  - nx.closeness_centrality (*graph_obj)* #might return France if looking at borders
  - nx.betweenness_centrality (*graph_obj)* #might return France if looking at borders

Managing Attributes

- A networkx graph, *as well as node and edge attributes*, are implemented as dictionaries.
- A graph has a dictionary interface to the nodes
- A node has a dictionary interface to its edges
- An edge has a dictionary interface to its attributes
- Can pass attribute names and values as optional parameters to functions that manage them

Managing Attibutes cont.

- Examples:

```
borders[“Germany”][“Poland”] [“weight”] = 456.0
borders.node[“Germany”][“area”] = 357168
borders.add_node(“Penguinia”, area=14000000]
```

- When you add nodes() and edges() functions with `data=True` parameter

```
borders.node(data=True)
⇒ [<<..>>  (‘Germany’, {‘area’: 357168}), <<..>>]
borders.edges(data=True)
⇒ [<<..>>  (‘Uganda’, ‘Rwanda’, {‘weight’: 169.0}), <<..>>]
```

Cliques

- Functions `find_cliques()` and `isolates()` detect maximal cliques and isolates zero-degree nodes, respectively.
  - Find_cliques not implemented for directed graph (must convert to simple)
  - Returns a list generator

```
list( nx.find_cliques(simple_graph_obj))
⇒[<<..>>, [‘Iran’, ‘Afghanistan’, ‘Pakistan’], <<..>>]

nx.isolates(graph_obj)
⇒ [‘Antartica’, <<..>>]
```

Communities

- Module for community detection not in Anaconda, must be installed separately. Does not support digraphs.
- Function `best_partition()` function returns a dictionary of numeric community labels indexed by the node labels.
- Function `modularity()` reports the modularity of the partition

```
import community
partition = community.best_partition(graph_obj)
community.modularity(partition, graph_obj)
```

If modularity is too low (< 0.5), does not have a clear community to rely on.

Input and Output

- Many read/write functions read network data from files and write data to files.

```
read_adjlist(f)   /        write_adjlist(f)
read_edgelist(f)    /      write_edgelist(f)
read_gml(f)         /      write_gml(f)            #.gml file
read_graphml(f)     /      write_graphml (f)       #.graphml
read_pajek(f)       /      write_pajak(G,f)        #.net
```

- Examples

```
with open(“borders.graphml”, “wb”) as netfile:
    nx.write_pajek(borders, netfile)
with open(“file.net”, “rb”) as netfile:
     borders = nx.read_pajek(netfile)
```