What is NetworkX?

- A Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex graphs and networks.
- https://networkx.org/

Software for Complex Networks

Release: 3.1  
Date: Apr 04, 2023

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. It provides:

- tools for the study of the structure and dynamics of social, biological, and infrastructure networks;
- a standard programming interface and graph implementation that is suitable for many applications;
- a rapid development environment for collaborative, multidisciplinary projects;
- an interface to existing numerical algorithms and code written in C, C++, and FORTRAN; and
- the ability to painlessly work with large nonstandard data sets.

With NetworkX you can load and store networks in standard and nonstandard data formats, generate many types of random and classic networks, analyze network structure, build network models, design new network algorithms, draw networks, and much more.

Citing

To cite NetworkX please use the following publication:
Installation

• Version 3.1
• Install manually from
  https://pypi.python.org/pypi/networkx
• Using `pip`
  
  ```shell
  $ pip install networkx
  ```
• Usage

  ```python
  import networkx as nx
  ```
Basic Example

In [ ]:

```python
G = nx.Graph()  # creates a graph
G.add_node('donut')  # creates a new node in the graph called 'donut'
G.add_edge(3, 4)  # creates an edge in the graph (3,4) while also creating new nodes
print(G.nodes())  # prints list of nodes in G
print(G.edges())  # prints list of edges in G
```

`['donut', 3, 4]
[(3, 4)]`
Graph Types

- **Graph**: undirected simple (allows self-loops)
- **DiGraph**: directed simple (allows self-loops)
- **MultiGraph**: undirected with parallel edges
- **MultiDiGraph**: directed with parallel edges

In [ ]:
```
g = nx.Graph()
d = nx.DiGraph()
m = nx.MultiGraph()
h = nx.MultiDiGraph()
```
Adding Nodes to a Graph

In [ ]:

```python
G = nx.Graph()
G.add_node('m')
G.add_nodes_from(['n','p'])
G.add_nodes_from('abc')
print(G.nodes())

G2 = nx.path_graph(5)
print(G2.nodes())
G.add_nodes_from(G2)
print(G.nodes())
```

```
['m', 'n', 'p', 'a', 'b', 'c']
[0, 1, 2, 3, 4]
['m', 'n', 'p', 'a', 'b', 'c', 0, 1, 2, 3, 4]
```
Adding Edges to a Graph

In [ ]:

```python
G = nx.Graph([('w', 'x'), ('x', 'y'), ('y', 'w')])
print(G.edges())

G.add_edge('w', 'z')
G.add_edges_from([('z', 'y'), ('z', 'x')])
print(G.edges())
```

```
[('w', 'x'), ('w', 'y'), ('x', 'y')]
[('w', 'x'), ('w', 'y'), ('w', 'z'), ('x', 'y'), ('x', 'z'), ('y', 'z')]
```
Node Attributes - 1

- Each node can have an attribute that you can assign. Then you are also able to easily access them through `G.nodes[]`

```python
In [ ]:
G = nx.Graph()
G.add_node(1, color='green')
G.add_nodes_from([2,3], color='red')
print(G.nodes[1])
print(G.nodes[1]['color'])
print(G.nodes[2]['color'])
print(G.nodes[3]['color'])

{'color': 'green'}
green
red
red
```
Node Attributes - 2

- Can also use `nx.set_node_attributes()`

```python
In [ ]:
G.nodes[3]['color'] = 'blue'
nx.set_node_attributes(G, 0.5, 'weight')
print(G.nodes[1])
print(G.nodes[2])
print(G.nodes[3])
```

```python
In [ ]: {'color': 'green', 'weight': 0.5}
{'color': 'red', 'weight': 0.5}
{'color': 'blue', 'weight': 0.5}
```

```python
In [ ]:
# you can also use a dict to specify the attributes
label_dict = dict(zip(range(1,4), range(1,4)))
nx.set_node_attributes(G, label_dict, 'label')
print(G.nodes[1])
```

```python
In [ ]: {'color': 'green', 'weight': 0.5, 'label': 1}
```
Edge Attributes - 1

• Each edge can have an attribute that you can assign. Then you are also able to easily access them through `G.edge[]`

In [ ]:
```python
G.add_edge(1, 2, thickness=1.5)
G.add_edges_from(([(2,3), (3,4)], thickness=2.0))
print(G.edges[1,2])
print(G.edges[2,3])
print(G.edges[3,4])
print(G.edges[2,1])
```

```
{'thickness': 1.5}
{'thickness': 2.0}
{'thickness': 2.0}
{'thickness': 1.5}
```
Edge Attributes - 2

- One can also add a third argument (a dict) to specify an edge attribute and its value

In [ ]:
```python
G.add_edges_from([(1, 2, {'size': 15})])
```

- The 'weight' attribute

In [ ]:
```python
G.add_weighted_edges_from([(10, 20, 4.9)])
```

In [ ]:
```python
# print each edge of G and their attributes
for e1, e2 in G.edges():
    print('edge (' + str(e1) + ', ' + str(e2) + '): ' + str(G.edges[e1, e2]))
```

edge (1, 2): {'thickness': 1.5, 'size': 15}
edge (2, 3): {'thickness': 2.0}
edge (3, 4): {'thickness': 2.0}
edge (10, 20): {'weight': 4.9}
Graph Attributes

In [ ]:

```python
# number of nodes
print('The number of nodes in G:"
print(len(G))
print(G.number_of_nodes())
print(G.order())

# number of edges
print('The number of edges in G:"
print(G.number_of_edges())

# node membership
print('Check node membership in G:"
print(G.has_node(1))

# edge presence
print('Check edge presence in G:"
print(G.has_edge(1, 2))
```
Node Neighbors

In [ ]:

```python
# new Graph (path graph)
G = nx.path_graph(4)

# use list comprehension to list all edges
print([e for e in G.edges()])

# adjacency list
print([((n, n nbrs) for n,nbrs in G.adjacency())])

[(0, 1), (1, 2), (2, 3)]
[(0, {1: {}}), (1, {0: {}, 2: {}}), (2, {1: {}, 3: {}}), (3, {2: {}})]
```
Node Degrees

- A node $v$'s degree describes the number of nodes in the graph connected to $v$. We use the `degree()` method.

In [ ]:

```
# return the degree of node 0
print(G.degree(0))

# return a list of (node, degree) pairs
print(G.degree())

# using list comprehension to get degree distribution
print([x[1] for x in G.degree()])
```

```
1
[(0, 1), (1, 2), (2, 2), (3, 1)]
[1, 2, 2, 1]
```
Graph Generators

- Complete Graph  
  `nx.complete_graph()`

- Path (or Chain) Graph  
  `nx.path_graph()`

- Bipartite Graph  
  `nx.complete_bipartite_graph()`

- Random Graph  
  `nx.erdos_renyi_graph()`

- Watts Strogatz Graph  
  `nx.watts_strogatz_graph()`

- Barabasi Albert Graph  
  `nx.barabasi_albert_graph()`

Consult the API (https://networkx.org/documentation/stable/reference/generators.html) for more info about these generators.
Other Useful Functions

- Includes shortest path, betweenness centrality, average clustering, and diameter

```python
In [ ]:
g = nx.Graph([(0,1), (3, 4), (4,5), (5,3), (5,0)])

print('Shortest path:')
print(nx.shortest_path(g, 0, 4))

print('Betweenness Centrality:')
print(nx.betweenness_centrality(g))

print('Average Clustering:')
print(nx.average_clustering(g))

print('Diameter:')
print(nx.diameter(g))
```
Visual Representation

• Use `matplotlib` module to draw (plot) graphs
  • Note that visual representation is not unique!
• Use `plt.savefig()` to save the figure onto a file.

In [ ]:
```python
import matplotlib.pyplot as plt

G = nx.path_graph(10)
nx.draw(G)
# plt.savefig('chain_graph.pdf')
plt.show()
```
Resources

• NetworkX Docs

• NetworkX Tutorial
  • https://networkx.org/documentation/stable/tutorial.html
Thank you!

Questions?