Triadic Closure Python Script
Intro to Web Science
10 points

Write a Python script called tradic_closure.py that determines which nodes satisfy the Strong Triadic Closure (STC) property and which edges are local bridges. The script should use the networkx library and use the starter code at the end of this document. The script uses the nx.read_edgelist() function to read the edgelist from the filename supplied on the command line. Example:

```
$ tradic_closure.py myedgelist.txt
```

The edges are then assigned a Boolean 'strong' property, indicating if an edge is strong or weak. Strong edges have a weight of at least 5.

The strong_triadic_closure() and is_local_bridge() bridge functions are left for you to implement.

- strong_triadic_closure() should return True if the given node satisfies the Strong Triadic Closure property. In other words, if the node has a strong edge to two other nodes, the two other nodes should be connected with an edge. If any single violation occurs, strong_triadic_closure() should return False. You will need to loop through all combinations of two neighbors and examine the edges between the nodes to write this function.

- is_local_bridge() should return True if the two given nodes are connected by an edge that serves as a local bridge. In other words, the nodes must share an edge but no common neighbors, and if the edge is removed from the graph, no new components are created. You'll find the following networkx functions helpful:
    - nx.common_neighbors()
    - nx.number_connected_components()
    - G.copy()
    - G.remove_edge()

Submit your working program to Canvas. If you pair-program, put both people's names in comments at the top of the script, and only one person should submit the solution.

Example edgelist text file:

```
A B 10
A C 4
A D 2
A E 5
B E 1
B C 7
C D 9
D E 6
```

```
D F 2
F G 3
G E 1
```

Output from example file:

```
Strong triadic closure:
A = True
B = True
C = False
D = False
E = True
F = True
G = True

Local bridges:
A-B = False
A-C = False
A-D = False
A-E = False
B-E = False
B-C = False
C-D = False
D-E = False
D-F = True
E-G = True
F-G = True
```

Starter code:

```
import sys
import networkx as nx
import matplotlib.pyplot as plt

# Return true if all possible triangle relationship combinations for the node that
# have two strong edges from the node are triangles.
def strong_triadic_closure(G, node):
    return True

# Returns true if nodes and connected but share no mutual friends and removing edge
# does NOT create two separate components
def is_local_bridge(G, node1, node2):
    return True


if len(sys.argv) < 2:
    print('Provide edgelist filename.')
else:
    filename = sys.argv[1]

    try:
        # Read edgelist and assign weight to each edge
        G = nx.read_edgelist(filename, data=(('weight',int),))

        #nx.draw(G, with_labels=True)
        #plt.show()
        # Assign new edge property to all edges called 'strong' where the weight >= 5
        for (node1, node2, weight) in G.edges.data('weight'):
            print(f'{node1} - {node2} : {weight}')
            G[node1][node2]['strong'] = weight >= 5

        #print(G.edges(data=True))

        print('Strong triadic closure:')
        for node in G.nodes:
            print(f'{node} = {strong_triadic_closure(G, node)}')

        print('\nLocal bridges:')
        for (node1, node2) in G.edges:
            print(f'{node1}-{node2} = {is_local_bridge(G, node1, node2)}')

    except FileNotFoundError:
        print(f'File not found: {filename}')
```