Github @angel870326

Programming Assignment 1: Star War Social Network Analysis

1. Adjacent Matrix

(1) Use *pandas* package to create an empty *DataFrame* with index and column both range from 0 to [number of 'nodes' - 1], which means that both index and columns have fixed length of the number of 'nodes' and each of them represents node index.

```
[7] nodes = data['nodes']
[9] adj_matrix = pd.DataFrame(index=range(len(nodes)), columns=range(len(nodes)))
```

(2) For each link in the 'links', we get the values of 'source' and 'target' which represent node indexes. Then, we set the corresponding cells in the DataFrame to binary weight 1. After the loop, we fill in all NaN values as binary weight 0 with .fillna().

```
[8] links = data['links']
[10] # Get source node index and target node index
     for link in links:
         i = link['source']
         j = link['target']
         adj_matrix[i][j] = adj_matrix[j][i] = 1
     adj_matrix = adj_matrix.fillna(0)
```

2. Betweenness Centrality

(1) Create vertices and edges

For the vertices, we collect every sources and targets in the 'links', remove the duplicates, and sort the values. We can find that node 79 is not in the links, so there are only 111 nodes in the vertices. For the edges, we collect each of the link in the form of tuple (source, target). Since the links are undirected, we simply make the tuple sorted that the source node index is smaller than the target node index in a tuple.

(2) Create graph

We define a function called **create_graph()** to add all the nodes and edges into *Graph()* of package *networkx* by giving two parameters, vertices and edges.

(3) Calculate betweenness centrality of the given node with betweenness_centrality() that we defined

Before calculating the betweenness value of the given node, we have to make pairs of all the vertices except the given node. So we first remove the node from vertices, then use nested for loop to pair every other nodes as tuples. Moreover, source and target should not be the same in each tuple, and we still hold that the nodes in each tuple should be sorted, which means source index is smaller than the target index in a tuple.

```
        # List of all vertices except the given node
        other_nodes = vertices.copy()
        other_nodes.remove(node)
        # Combination of all vertices except the given node
        comb = []
        for i in range(0, len(other_nodes)):
            for j in range(i+1, len(other_nodes)):
                if i != j:
                    comb.append((other_nodes[i], other_nodes[j]))
```

For each pairs we just created, we calculate the shortest paths by *all_shortest_paths()* in package *networkx*, to get ALL the shortest paths for the given source and target, then store each of them in a list. Next, we check if the given node is in each of the paths and count the number of paths passing through the given node. Finally, we can calculate the betweenness value by adding (number of the shortest paths passing through the given node / number of the shortest paths).

```
        # Calculate betweenness of the given node
        betweenness = 0
        for c in comb:
            # Find all the shortest paths
            shortest_paths = [p for p in all_shortest_paths(G, c[0], c[1], weight=None, method='dijkstra')]   # generator to list
            # Count the number of paths containing the given node
            count = 0
            for p in shortest_paths:
                if node in p:
                    count += 1
            betweenness += count/len(shortest_paths)
        return round(betweenness, 2)
```

(4) Calculate betweenness centrality for all the vertices and find the top 10 values with top_k_betweenness_centrality() that we defined

For each node in the vertices, we calculate betweenness centrality by calling **betweenness_centrality()** that we defined earlier and store the node index and corresponding betweenness value in a list. Then, we sort the list by the betweenness values and return the top k.

Finally, we can simply call top_k_betweenness_centrality(vertices, edges, 10) to get the top 10 betweenness values and node indexes.