6 Graph Connectivity

This module covers connectivity concepts, including vertex and edge connectivity, cut vertices, bridges, and network flow. The exercises will help you implement these concepts in Python.

6.1 Exercise 1: Connected Components in an Undirected Graph

Task: Implement a function to find all connected components of an undirected graph.
Hint: Use BFS or DFS to traverse the graph.

```python
def find_connected_components(graph):
    visited = set()
    components = []

    def dfs(node, component):
        visited.add(node)
        component.append(node)
        for neighbor in graph[node]:
            if neighbor not in visited:
                dfs(neighbor, component)

    for node in graph:
        if node not in visited:
            component = []
            dfs(node, component)
            components.append(component)

    return components
```

6.2 Exercise 2: Vertex and Edge Connectivity

Task: Implement functions to compute the vertex and edge connectivity of a graph.
Hint: Use min-cut properties to determine connectivity.

6.3 Exercise 3: Cut Vertices (Articulation Points)

Task: Implement an algorithm to find all articulation points in a graph.
Hint: Use DFS with low-link values.

6.4 Exercise 4: Bridges (Cut Edges)

Task: Implement an algorithm to find all bridges in a graph.

6.5 Exercise 5: Network Flow (Max-Flow Min-Cut Theorem)

Task: Implement the Ford-Fulkerson algorithm to compute maximum flow in a flow network.

6.6 Exercise 6: Strong Connectivity in Directed Graphs

Task: Implement Kosaraju's or Tarjan's algorithm to find strongly connected components (SCCs).

6.7 Bonus Challenge 1: Random Graph Connectivity Testing

Task: Generate random graphs and compute their connectivity properties.

6.8 Bonus Challenge 2: Critical Edge and Vertex Removal

Task: Given a graph, identify the most critical edge or vertex whose removal minimizes connectivity.