1 Fundamental Concepts of Graph Theory

This module covers the basic concepts of graphs, including representations, connectivity, shortest paths, and basic graph properties. The exercises below are designed to help you understand and implement these concepts in Python.

1.1 Exercise 1: Graph Representation

Task: Implement both adjacency matrix and adjacency list representations of a graph.
Hint: Use a dictionary for adjacency lists and a 2D list for adjacency matrices.

```python
# Adjacency List Representation
class Graph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, u, v):
        if u not in self.graph:
            self.graph[u] = []
        self.graph[u].append(v)

G = Graph()
G.add_edge(1, 2)
G.add_edge(2, 3)
print(G.graph)
```

1.2 Exercise 2: Graph Matrices

Task: Implement functions to compute adjacency, incidence, and path matrices for a given graph.
Hint: Use NumPy for efficient matrix operations.

```python
import numpy as np

def adjacency_matrix(graph, n):
    matrix = np.zeros((n, n))
    for u in graph:
        for v in graph[u]:
            matrix[u-1][v-1] = 1
    return matrix
```

1.3 Exercise 3: Koenigsberg Bridges Problem

Task: Model the Koenigsberg Bridges problem as a graph and determine if an Eulerian path exists.
Hint: Check vertex degrees to determine Eulerian properties.

1.4 Exercise 4: Graph Isomorphism Check

Task: Implement an algorithm to check if two graphs are isomorphic.

Hint: Convert graphs into canonical representations and compare them.
Challenge: Implement an efficient version that works on large graphs.

1.5 Exercise 5: Degree and Connectivity Tests

Task: Write functions to compute degree sequences and test graph connectivity.

```python
def degree_sequence(graph):
    return sorted([len(graph[node]) for node in graph], reverse=True)

def is_connected(G):
    import networkx as nx
    return nx.is_connected(G)
```

1.6 Exercise 6: Shortest Path Algorithms

Task: Implement Dijkstra's and Floyd-Warshall shortest path algorithms.

```python
import heapq

def dijkstra(graph, start):
    pq = [(0, start)]
    distances = {node: float('inf') for node in graph}
    distances[start] = 0

    while pq:
        (dist, node) = heapq.heappop(pq)
        for neighbor, weight in graph[node]:
            new_dist = dist + weight
            if new_dist < distances[neighbor]:
                distances[neighbor] = new_dist
                heapq.heappush(pq, (new_dist, neighbor))

    return distances
```

1.7 Exercise 7: Eulerian and Hamiltonian Paths

Task: Implement algorithms to check if a graph has an Eulerian or Hamiltonian path.

1.8 Exercise 8: Graph Theorems Computational Tests

Task: Write Python programs to test the following theorems: - Handshaking Lemma: Sum of degrees $= 2 \times$ edges. - If all nodes have even degrees, the graph has an Eulerian cycle. - Euler's Formula: $V - E + F = 2$ for planar graphs.

1.9 Exercise 9: Graph Decomposition

Task: Implement algorithms to decompose a graph into subgraphs.
Hint: Use connected components and biconnected component algorithms.

1.10 Bonus Challenge 1: Graph Reconstruction Conjecture

Task: Given all vertex-deleted subgraphs of a graph, reconstruct the original graph.

1.11 Bonus Challenge 2: Planarity Testing

Task: Implement an algorithm to test whether a graph is planar using Kuratowski's theorem.

1.12 Bonus Challenge 3: Random Graph Generation and Analysis

Task: Generate random graphs and analyze their properties (degree distribution, connected components, average shortest path length).

Hint: Use NetworkX's random graph generators.