3 Graph Coloring

This module covers vertex coloring, edge coloring, chromatic numbers, matching, and key theorems. The exercises will help you implement these concepts in Python.

3.1 Exercise 1: Vertex Coloring with Greedy Algorithm

Task: Implement the greedy coloring algorithm.

Hint: Order vertices arbitrarily and assign the smallest available color.

```
def greedy_coloring(graph):
    color = {}
    for node in graph:
        used_colors = {color[neighbor] for neighbor in graph[node] if neighbor
            in color}
        color[node] = next(c for c in range(len(graph)) if c not in used_colors
            )
    return color
```

3.2 Exercise 2: Chromatic Number Calculation

Task: Implement a function to compute the chromatic number of a graph.

Hint: Try different vertex orderings and compare results.

3.3 Exercise 3: Edge Coloring (Vizing's Theorem)

Task: Implement an algorithm for edge coloring using at most $\Delta + 1$ colors.

3.4 Exercise 4: Bipartite Graph Coloring

Task: Write a function to check if a graph is bipartite and 2-colorable.

Hint: Use BFS to assign alternating colors.

3.5 Exercise 5: Brook's Theorem Implementation

Task: Verify that any graph (except complete graphs and odd cycles) satisfies $\chi(G) \leq \Delta(G)$.

3.6 Exercise 6: Matching in Bipartite Graphs (Hall's Theorem)

Task: Implement Hopcroft-Karp algorithm to find maximum matching.

3.7 Bonus Challenge 1: Graph Coloring for Scheduling

Task: Model an exam scheduling problem as a graph coloring problem and solve it.

3.8 Bonus Challenge 2: Mycielski's Construction

Task: Implement Mycielski's theorem to construct triangle-free graphs with higher chromatic numbers.