4 Graph Matching

This module covers graph matching concepts, including maximum matching, bipartite matching, and key theorems. The exercises below will help you understand and implement these concepts in Python.

4.1 Exercise 1: Maximum Matching in General Graphs

Task: Implement the Edmonds' Blossom Algorithm to find maximum matching in general graphs.
Hint: Use augmenting paths and shrinking blossoms.

4.2 Exercise 2: Maximum Bipartite Matching (Hopcroft-Karp Algorithm)

Task: Implement the Hopcroft-Karp algorithm to find the maximum matching in a bipartite graph.

```python
from collections import deque

def bfs(graph, pair_u, pair_v, dist):
    queue = deque()
    for u in graph:
        if pair_u[u] == 0:
            dist[u] = 0
            queue.append(u)
        else:
            dist[u] = float('inf')
    dist[0] = float('inf')
    while queue:
        u = queue.popleft()
        if dist[u] < dist[0]:
            for v in graph[u]:
                if dist[pair_v[v]] == float('inf'):
                    dist[pair_v[v]] = dist[u] + 1
                    queue.append(pair_v[v])
    return dist[0] != float('inf')
```

4.3 Exercise 3: Hall's Theorem Verification

Task: Implement a function to check if a bipartite graph satisfies Hall's marriage condition.

4.4 Exercise 4: Stable Matching (Gale-Shapley Algorithm)

Task: Implement the Gale-Shapley algorithm for stable matching.

4.5 Exercise 5: Weighted Matching (Hungarian Algorithm)

Task: Implement the Hungarian Algorithm for finding the maximum weight matching in a weighted bipartite graph.

4.6 Bonus Challenge 1: Random Matching Generation

Task: Generate a random graph and find its maximum matching.

4.7 Bonus Challenge 2: Matching for Job Assignment

Task: Model a job assignment problem as a matching problem and solve it using the Hungarian Algorithm.