CS41 Lab 3

The lab and homework this week focus on graph algorithms for undirected graphs. The following definitions might be helpful/relevant.

- A **path** \( P \) on a graph \( G = (V, E) \) is a sequence of vertices \( P = (v_1, v_2, \ldots, v_k) \) such that \((v_i, v_{i+1}) \in E\) for all \( 1 \leq i < k \).
- A path is **simple** if all vertices are distinct.
- The **length** of a path \( P = (v_1, \ldots, v_k) \) equals \( k - 1 \). (Think of the path length as the number of edges needed to get from \( v_1 \) to \( v_k \) on this path).
- A **cycle** is a sequence of vertices \((v_1, \ldots, v_k)\) such that \( v_1, \ldots, v_{k-1} \) are all distinct and \( v_k = v_1 \). A cycle is odd (even) if it contains an odd (even) number of edges.

This week, **work on problem 1 first**, and check your answers with me before moving on to problems 2 or 3 (which you’re encouraged to investigate in any order you wish).

1. **Testing Bipartiteness** A graph \( G = (V, E) \) is **bipartite** if \( V \) can be partitioned into disjoint sets \( A, B \) such that any edge \( e \in E \) has one endpoint in \( A \) and one endpoint in \( B \). Alternatively, a graph is bipartite if it is possible to color each vertex in the graph one of two colors (say, green or blue) so that each edge is **bichromatic**—the endpoints of each edge have different colors.

   In this problem, you will develop and analyze an efficient algorithm to decide if a graph is **bipartite**.

   (a) Create a bipartite graph \( G_1 \) and a non-bipartite graph \( G_2 \).
   (b) Which of the graphs on the whiteboard in front of class are bipartite?
   (c) Design an efficient algorithm that takes a graph \( G = (V, E) \) as input and outputs **yes** iff \( G \) is bipartite.
   (d) Rigorously prove that your algorithm works. You should prove that your algorithm outputs **yes** given any bipartite graph, and outputs **no** given any graph that is non-bipartite.
   (e) What is the runtime of your algorithm? You may use any data structure from CS35 by name, without needing to justify their runtimes.

2. **Cycle Detection.** Design and analyze an efficient algorithm for finding a cycle in a graph. Your algorithm should take as input a graph \( G = (V, E) \) and report a cycle (or output NO if no cycle exists). If there are multiple cycles in the graph, your algorithm should just output one.

3. **Testing Tripartiteness.** Call a graph \( G = (V, E) \) **tripartite** if \( V \) can be partitioned into disjoint sets \( A, B, C \) such that for any edge \((u, v) \in E\), the vertices \( u, v \) lie in different sets. Design and analyze an algorithm to test a graph for tripartiteness.