COSC 3320: Data Structure and Algorithms **(Due: 06/28/20, 11.59pm)**

Programming Assignment #2

Submission Policy: Read all the instructions below carefully before you start working on the assignment, and before you make a submission.

- Create an account on Leetcode (https://leetcode.com/)
- Submit your code on your Leetcode account.
- Mention your Leetcode profile in your submission report.
- Your assignment submission should contain a code file, a pdf (report), and a link to your leetcode profile.
- Please don't copy your code/solution from anywhere or anybody.
- Please read the academic dishonest policy posted in BB and Piazza. By submitting this assignment you affirm that the work is entirely your own.
- If you violate the academic dishonesty policy, then your score for this assignment will be -MAX = -100.

Problem : Bipartite Graph Checking (100 points)

Leetcode Link: https://leetcode.com/problems/is-graph-bipartite/

Problem Description

Given a graph, the goal is to check whether the graph is bipartite or not.

A graph $G = (V, E)$ is bipartite if its vertex set V can be partitioned into two disjoint subsets (S, T) such that every edge $(u, v) \in E$ has one endpoint in S and the other endpoint in T, i.e., either u belongs to S and v belongs to T or vice versa. See Leetcode and Figures 1 and 2 for examples of bipartite and non-bipartite graphs.

There are other equivalent definitions of a bipartite graph. A graph is bipartite if and only if it has no *odd* length cycle, i.e., no cycle in the graph has an odd number of edges. Note that if a graph has no cycles it is bipartite (hence trees are bipartite graphs).

It is an interesting exercise for you to show that the above two definitions are equivalent (highly recommended if you want to get a taste of graph theory). But that is not the goal of this exercise.

Using the odd cycle definition one can use DFS or BFS to elegantly check whether a graph is bipartite in linear time, i.e., $O(m + n)$ time. Such solutions are discussed in leetcode (note that some of these may not be fully correct). This is also not the goal of this exercise. If you give such solutions, these are not acceptable and will *not get any points*.

The goal of this exercise, is to explore yet another approach to check bipartiteness and is based on the following idea. You have to argue that the idea is correct (as mentioned below) and implement the idea as an algorithm that checks for bipartiteness of a graph.

Let $G = (V, E)$ be the input graph that you want to check for bipartiteness. We first transform G into a new graph $G' = (V', E')$ as follows. For each each node $v \in V$, construct two nodes $v_1, v_2 \in V'$ and for each edge $(u, v) \in E$, create two edges (u_1, v_2) and (u_2, v_1) in E'. This completes the description of constructing G' from G. See Figure 1 for an illustration.

The idea of this construction is that it *reduces* the problem of *checking bipartiteness of* G to *checking the number of connected components of* G' as stated in the following theorem.

Theorem 1. *Let C be the number of connected components in G. Then the number of connected components in G' is $2C$ if and only if the graph G is bipartite.*

Your first goal is to give a proof of the above theorem (see the submissions instructions below).

To get an idea of why the above theorem might be true, see the two examples given in Figures 1 and 2. The graph in Example 1 is bipartite (no cycle) and the number of connected components is 1; it is 2 in G'. The graph in Example 2 is not bipartite (odd cycle of length 3) and the number of connected components is 1; it is 1 also in G'.

Your second goal is to use the theorem above to give an efficient algorithm for checking bipartiteness.

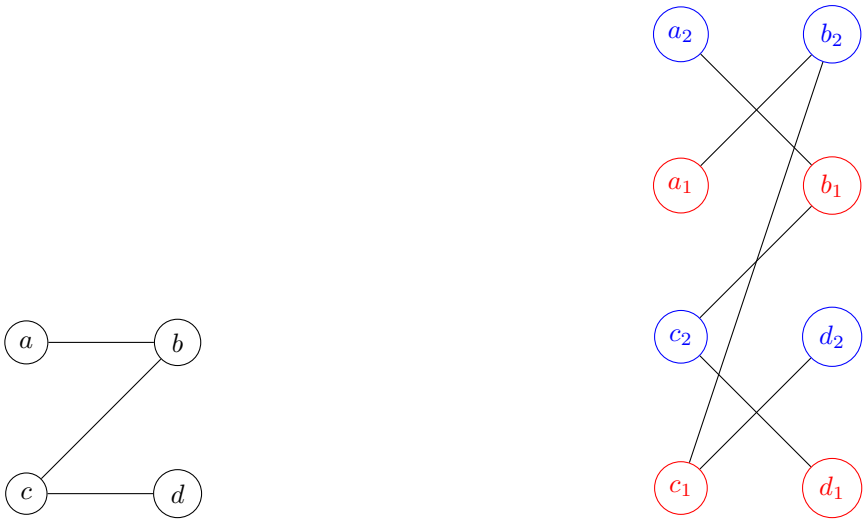

Figure 1: Example 1

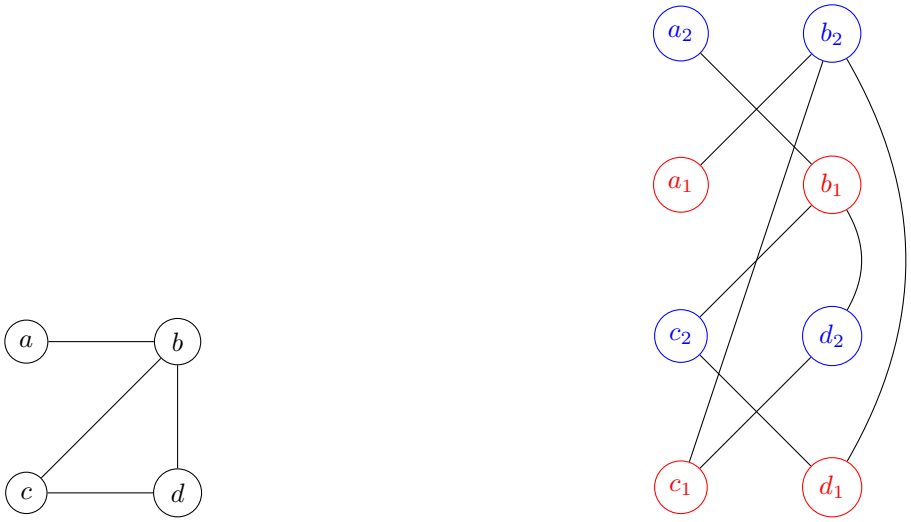

Figure 2: Example 2

The following has to be submitted.

(a) Proof of Theorem 1. (30 points)

(b) Describe an efficient algorithm for checking bipartiteness based on the above approach. What is the running time of your algorithm and why? (20 points)

(c) Code up your solution in C/C++/Python/Java. Your code should be well commented. Your code should compile, otherwise no points. (50 points)

(d) Submit your solution in Leetcode. Leetcode will automatically check your code and let you know if it is working correctly.

(e) Submit your code to the folder (even if it is not accepted by Leetcode). Say whether your code was accepted or not by leetcode. We will also check whether your code was accepted by leetcode, in case if you say so.

Hints:

- Get started early.
- You can try the idea on other examples of graphs. Does the theorem check out? This should give an idea for the Theorem proof. Using the odd cycle definition of bipartite graph can be helpful.

- Note that the statement of the theorem says "if and only if." That means what?
- Note that you have to count the number of connected components. This can be done quite easily as we have learnt in the course.
- Your running time should be linear in the size of the graph.
- (This is not a hint, but a remark). Algorithms and theoretical computer science uses elegant mathematical ideas (such as Theorem 1) to design correct and efficient algorithms (and also show impossibility of certain tasks). If you are interested in exploring more fascinating ideas, then the graduate course COSC 6320 is worth a look.

  If you are interested in solving and coding algorithmic problems, then coding competition is a natural arena to explore. UHCS is looking for undergrads for their ACM programming contest team. If you are interested, contact me **after the grades of this course are posted.**