Homework 3

Due Wednesday, February 9th at 11:59 pm ET on Gradescope

Solution guidelines For problems that require you to provide an algorithm, you must give the following:

1. a precise description of the algorithm in English and pseudocode (*),
2. a proof of correctness,
3. an analysis of the asymptotic running time and space.

You may use algorithms from class as subroutines. You may also use any facts that we proved in class, e.g. correctness of subroutines, running time of subroutines.

You should be as clear and concise as possible in your write-up of solutions.

A simple, direct analysis is worth more points than a convoluted one, both because it is simpler and less prone to error and because it is easier to read and understand.

(*) It is fine if the English description concentrates on the high level ideas and doesn't include all the details. But the reader should not have to figure out your solution solely based on the pseudocode. You can also add comments to your pseudocode, in fact that is best practice. FYI we will share a document on good pseudocode style with you and it will also be discussed in labs and lecture.

Problem 1.

Consider an undirected graph $G(V, E)$ of a social media network. Each person in the network is represented by a node in V. Two people are connected by an edge in E if they are friends in the network. We define the **friendship graph** of G to be a new graph $F(G) = (V', E')$, where the friendships represented by edges in G become the new vertices of $F(G)$. Two nodes in $F(G)$ have an edge between them if and only if the corresponding edges in G share a common vertex (i.e. the two friendships share a mutual friend).

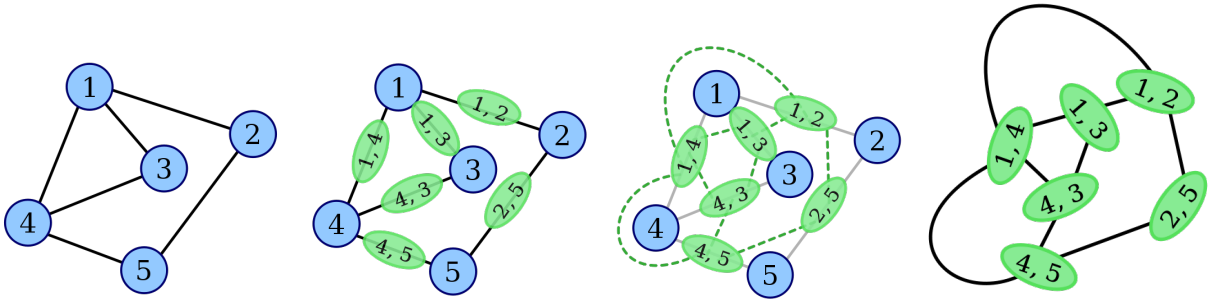

Figure 1: Producing $F(G)$ from G

Find an algorithm `FriendshipGraph( )` that produces the friendship graph $F(G)$ from a given graph G (input as an adjacency list). Your algorithm should run in time $O(|V'| + |E'|)$. Write down a formula for the sizes of $|V'|$ and $|E'|$ in terms of V and E.

Problem 2.

Let $G(V, E, s, c)$ be an undirected graph with s as a source node. Each of its edges $(u, v) \in E$ is either blue or red, and $c(u, v)$ denotes the color of edge (u, v). Let all edges adjacent to s be blue.

A path between nodes u and v in graph G, called $P_{u \to v}$, is a *shortest* path if there is no other path between u and v with fewer edges. A simple path P in G is called *alternating* if the edge colors in the path alternate between red and blue. An *alternating shortest path* is a path that is both shortest and alternates.

Find an algorithm `Alternate( )` that decides whether there is an alternating shortest path from s to every other node.

The **input** to your algorithm is $G(V, E, s, c)$ and a length-n array `dist`. G is represented as an adjacency list (implemented as a hash table). The colors c will be given to you as a hash table indexed with the node pairs of the edges. Thus $c[(u, v)]$ returns the color of edge (u,v). (Since this is an undirected graph $c[(u, v)]$ and $c[(v, u)]$ return the same color. The array entry `dist[i]` contains the distance (length of the shortest path) from s to node i in the non-edge-colored sense. The algorithm needs to find whether there is an alternating shortest path between s and every other node v in V.

The **output** of your algorithm is "yes" if there exists an alternating shortest path from s to every v. (Note that it is possible that there are both alternating and non-alternating path of the same length.) The output is "no" if there is at least one node v that doesn't have an alternating shortest path to s.