Algorithms Lab 9

The in-lab problems are to be solved during the lab time. Work with your team, but write your solutions individually. Your answers will not be graded, however you need to work through these problems; please staple the answers to the in-class exercises to the assignment that you hand in.

The homework problem set is due in one week. Work with your team, but write your solutions individually. List the people with whom you discussed the problems.

In lab

1. Finish "Applications of BFS and DFS" handout.

2. The two-colorability problem from handout: Is it possible that the vertices of a given graph be assigned one of two colors, such that no edge connects vertices of the same color? (Note: this is equivalent to the question: is G bipartite?))

3. (CLRS 22.4-3) Give an algorithm that determines whether or not a given undirected graph $G = (V, E)$ contains a cycle. Your algorithm should run in $O(|V|)$ time, independent of $|E|$.

4. (CLRS 22.5-5) Give a linear time algorithm to compute the component graph of a directed graph $G = (V, E)$. Make sure there is at most one edges between two vertices in the comopnent graph your algorithm produces.

Homework

1. (CLRS 22.4-2) Give a linear-time algorithm that takes as input a directed acyclic graph $G = (V, E)$ and two vertices s and t, and returns the number of simple paths from s to t in G. For example, the DAG in Fig. 22.8 CLRS contains exactly four simple paths from p to v: pov, $poryv$, $posryv$ abd $psryv$. Your algorithms needs to only count the simple paths, not list then.

   Hint: dynamic programming on DAGs.

2. Assume that the length of a path is the number of edges on the path. (a) Describe how to compute the longest path in a DAG starting from a specified vertex. (b) Describe how to compute the longest path in a DAG.

Can you extend your algorithm to compute shortest paths, instead of longest?

Hint: dynamic programming on DAGs.

3. (4.2.27 Sedgewick Wayne) Explain why the following algorithm does not necessarily produce a topological order: Run BFS, and label the vertices by increasing distance to their respective sources. Note: To prove that a certain algorithm does not work, it's sufficient to show a counter-example.

4. (4.2.31 Sedgewick Wayne) Describe a linear time algorithm for computing the strong component containing a given vertex v. On the basis of that algorithm, describe a simple quadratic time algorithm for computing the strong components of a digraph.

5. (4.2.32 Sedgewick Wayne) (Hamiltonian paths in DAGs) Given a DAG, design a linear time algorithm to determine whether there is a directed path that visits each vertex exactly one.