FALL 2019: **COT 6405** Analysis of Algorithms

[Homework 5; Due 11:59 PM, Nov 17, via Canvas]

General submission guidelines and policies: Add the following signed statement. Without this statement, your homework will not be graded.

> I have adhered to the collaboration policy for this class. In other words, everything written down in this submission is my own work. For problems where I received any help, I have cited the source, and/or named the collaborator.

Problems

21. (**Exercise**) Design an efficient algorithm to compute the in-degree and out-degree of every vertex in a given directed graph $G(V, E)$. The in-degree (out-degree, resp.) of a vertex is defined as the number of edges directed into (out of, resp.) that vertex. Assume that the input graph has n vertices and m edges. Analyze the time complexity and explain your analysis. I recommend that you modify the DFS algorithm we discussed in class.

22. (**Exercise**) Design an efficient algorithm that takes a directed graph $G(V, E)$ as input and outputs a directed graph $G'(V, E')$ where every edge is reversed. In other words, for every edge $e = (u, v) \in E$, there is an edge $e' = (v, u) \in E'$ and vice versa. Assume that the input graph has n vertices and m edges. Analyze the time complexity and explain your analysis.

23. (**Exercise**) Problem 22.2-7, page 602.

24. (**Exercise**) Modify DFS or BFS to design an algorithm called CheckForODDCycle for checking whether a given connected, undirected, unweighted, simple graph $G(V, E)$ has an odd length cycle. Assume that the input graph has n vertices and m edges. Note that the output is boolean. Provide an argument why you think the algorithm is correct and analyze its time complexity.

25. (**Regular**) *BFS numbering* of a vertex in an undirected, unweighted graph when BFS is performed from a start vertex s is the distance of that vertex from s. Given an undirected, unweighted graph $G(V, E)$ and a set $S \subset V$ of size k, design an efficient algorithm that simulates BFS from all k vertices of S. In other words, design an efficient algorithm that computes the distance to the **closest** vertex in S for each vertex.

26. (**Regular**) In the MST problem, we are required to find a spanning tree that has the minimum sum of weights of its edges. In the Miami version of the problem, MMST, we are required to minimize the largest edge weight. Design an algorithm to compute an MMST. Argue that it is correct. Analyze its time complexity.

27. (**Exercise**) Given a weighted undirected graph G with non-negative edge weights, if the edge weights are all increased by a positive additive constant, can the minimum spanning tree change? Can the output of Dijkstra's algorithm change for some (fixed) start vertex s? What if they are decreased by a positive constant? What if the edge weights are all multiplied by a positive constant? Give (very) simple examples, if you claim that they can change.