CS 457, Data Structures and Algorithms I
Fifth Problem Set

November 20, 2019

Due on December 2. Collaboration is not allowed. Contact Daniel and me for questions.

For any algorithms that you propose, also provide a (high level) justification regarding their worst-case running time bounds, and a brief explanation regarding their correctness.

1. (25 pts) Consider a variation of the Strongly-Connected-Components(G) (see Page 617 of your textbook) which, rather than calling DFS(G^T) in Step 3, calls DFS(G) instead (everything else remains the same). Provide a graph $G = (V, E)$ for which this variation is incorrect, i.e., it does not output the correct set of strongly connected components of G. For full credit, explain what the variation of the algorithm would do for this graph G:

   i) provide $v.d$ and $v.f$ for all $v \in V$ for Step 1 (you can choose the initial ordering of the vertices),
   ii) provide $v.d$ and $v.f$ for all $v \in V$ for Step 3,
   iii) provide the set of connected components that it would output, and
   iv) explain why this output is incorrect.

2. (25 pts) Provide an algorithm that, given a directed acyclic graph $G = (V, E)$ and two vertices $s, t \in V$, returns the number of simple paths from s to t in G. Your algorithm's worst-case running time should be $O(m + n)$.

3. (25 pts) An Eulerian tour of a strongly connected directed graph $G = (V, E)$ is a directed cycle that contains *every* edge in E exactly once (at least once and no more than once). First, prove that G has an Eulerian tour *if and only if* every vertex in V has its in-degree equal to its out-degree. Then, using this fact, provide a $O(m)$ algorithm that returns an Eulerian tour of G, if one exists, or reports that no Eulerian tour exists. Hint: your algorithm can merge edge-disjoint cycles.

4. (25 pts) Provide a DFS-based algorithm that takes as input a graph $G = (V, E)$ and determines if there exists an edge $e \in E$ such that removing e from E increases the number of connected components of G. If such an edge exists, your algorithm should return all of the edges in E with this property. For instance, if $V = \{v_1, v_2, v_3\}$ and $E = \{(v_1, v_2), (v_2, v_3)\}$, then $G = (V, E)$ has a single connected component. Removing either one of the two edges in E would increase the connected components from one to two, so the algorithm should return both of these edges in this example. The worst-case running time of the algorithm should be $O(m + n)$.