Problem 1: Every instance of the stable matching problem allows at least one stable matching; however, in some cases more than one may exist. Design an algorithm to test whether a given instance has a single stable matching or more than one.


Problem 3: The strongly connected components algorithm presented in class seems to be more complicated than it should be. In particular, do we really need to use $G^{rev}$? Consider the following simpler algorithm:

i) call $DFS(G)$ to compute finishing times $f[u]$ for every vertex $u$.
ii) call $DFS(G)$ processing vertices in the main loop in increasing order of $f[u]$.
iii) output each tree in the second DFS forest as a separate component.

Prove or disprove the correctness of the above algorithm.

Problem 4: Show how to modify the DFS algorithm presented in class to compute $low[u]$ for every vertex $u$ in $O(n + m)$ time.

Problem 5: Let $G$ be a directed graph with vertices numbered $1 \ldots n$. For a vertex $i$ define $small(i) = \min\{j \mid j$ is reachable from $i\}$, that is, the smallest vertex reachable from $i$. Design an $O(n + m)$ time algorithm that computes $small(i)$ for every vertex in the graph.

Problem (extra credit): In a directed graph, a get-stuck vertex has in-degree $n - 1$ and out-degree 0. Assume the adjacency matrix representation is used. Design an $O(n)$ algorithm to test if a given graph has a get-stuck vertex. Yes, this problem can be solved without looking at the entire input matrix.