ICS 311, Spring 2016, Problem Set 08, Topic 14

This is a 32 point homework. The extra 2 points are extra credit opportunities.
Due by midnight Tuesday 3/29

#1. Peer Credit Assignment

1 Point Extra Credit for replying

Please list the names of the other members of your peer group for class the week of March 14-17, and the number of points you think they deserve for their participation in group work on the two days combined.

- You have a total of 6 points to allocate across all of your peers.
- You can distribute the points equally, give them all to one person, or do something in between.
- You need not allocate all the points available to you.
- *You cannot allocate any points to yourself!* Points allocated to yourself will not be recorded.

#2 (8 pts) DFS and Cycles

DFS classifies edges as tree edges, back edges, forward edges, and cross edges (see p. 609).

(a) (2 pts) How can you modify DFS to detect back-edges? Say how you would modify the DFS and/or DFS-Visit procedures to print out all back edges found (reference CLRS line numbers to state your solution clearly), and then justify why your solution leads to the correct result.

(b) (2 pts) How can you modify DFS to determine whether a graph has a cycle? Say how you would modify the DFS and/or DFS-Visit procedures to print something and return when a cycle is found, and then justify why your solution leads to the correct result.
(c) **2 pts** What is the asymptotic run time of this modified cycle-detection algorithm on directed graphs? Assume that it exits the DFS as soon as a cycle is found. Justify your answer.

(d) **2 pts** What is the asymptotic run time of this modified cycle-detection algorithm on undirected graphs? Assume that it exits the DFS as soon as a cycle is found, and think carefully about how many edges you can have in an undirected graph without cycles. Justify your answer.

---

**#3 (8 pts) Bottom-Up Longest-Paths**

In Problem Set 7 you wrote `Longest-Path-Memoized`, a recursive (top-down) dynamic programming solution to the longest paths problem. Here you will solve the same problem in $\Theta(V+E)$ using a bottom-up dynamic programming approach. **Hint:** We need to arrange to solve smaller problems before larger ones: use topological sort (which you may assume has already been written).

(a) Write the pseudocode for `Longest-Path-Bottom-Up`.

(b) Explain why it works; in particular, why topological sort is useful.

(c) Analyze its asymptotic run time.

---

**#4 (8 pts) Counting Simple Paths in a DAG**
Design an 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 directed acyclic graph on the right contains exactly four simple paths from vertex $p$ to vertex $v$: $pov$, $poryv$, $posryv$, and $psryv$. Your algorithm should run in time $O(V + E)$. (Your algorithm needs only to count the simple paths, not list them.)

Hints: Solve the more general case of number of paths to all vertices. Use topological sort to solve smaller problems before larger ones. Be sure to consider the boundary cases where $s = t$ and where there are no simple paths between $s$ and $t$.

(a) Write the pseudocode.

(b) Explain why it works.

(c) Analyze its asymptotic run time.

#5 (8 pts) Bad Networks

An Internet network can be modeled using a directed graph $G = (V, E)$: The set of nodes $V$ consists of one node for every server in the network and there is an edge $(u, v) \in E$ if and only if there is a direct link from server $u$ to server $v$. Any server within a strongly connected component can send and receive packets to and from any server within the same strongly connected component. To ensure uninterrupted connectivity on the network, the networks are usually designed in such a way that there are more than one simple path between every pair of vertices. Then if any link goes down, the packets can still be routed using the remaining links.
In this problem we are interested in detecting **really bad networks**. Given a directed graph \( G = (V, E) \) design an algorithm that determines if there is at most **one** path between **every** pair of nodes in \( G \). For full credit, your algorithm should run in \( O(V(V+E)) \) time.

(a) Write the pseudocode.

(b) Explain why it works.

(c) Analyze its asymptotic run time.