Problem 1. Consider a rooted DAG (directed, acyclic graph with a vertex – the root – that has a path to all other vertices). Assume the graph is stored as an adjacency list. Give a linear time ($O(m+n)$) algorithm to find the length of the longest simple path from the root to each of the other vertices. (The length of a path is the number of edges on it.)

Problem 2. Consider the strongly connected components algorithm. Assume that the depth-first-search algorithm, given the choice, will visit smaller numbered vertices first.

(a) Give an example of a directed graph with around 10 to 15 vertices, numbered 1 to n, satisfying the following conditions: (1) The depth-first-search forest has exactly two trees. (2) At least one tree in the depth-first-search forest contains more than one strongly connected component, where at least two of those strongly connected components have at least three vertices. (3) There is at least one forward edge between two vertices in the same strongly connected component. (4) There is at least one forward edge between two vertices in different strongly connected components. (5) There is at least one cross edge between two vertices in the same tree. (6) There is at least one cross edge between two vertices in different trees.

Draw the graph. Label the node numbers. Circle the strongly connected components.

(b) Draw the graph. Label the nodes with their finish numbers (not their node numbers). Label the back edges, forward edges, and cross edges. Circle the strongly connected components.

(c) Reverse the edges. Draw the graph. Label the nodes with their finish numbers. Circle the strongly connected components.

Problem 3. Consider the incorrect algorithm that tries to find strongly connected components by processing the nodes in the second pass in increasing order of finish time (without reversing the edges).

(a) Briefly give our intuition of why this is a good idea.

(b) Show that the algorithm is incorrect (by giving a counterexample).

(c) Briefly explain why our intuition was incorrect.

Problem 4. Do Exercise 8 on pages 109-110 of Kleinberg and Tardos:

A number of stories in the press about the structure of the Internet and the Web have focused on some version of the following question: How far apart are typical nodes in these networks? If you read these stories carefully, you find that many of them are confused about the difference between the *diameter* of a network and the *average distance* in a network; they often jump back and forth between these concepts as though they're the same thing.

As in the text, we say that the distance between two nodes u and v in a graph $G = (V, E)$ is the minimum number of edges in a path joining them; we'll denote this by *dist(u,v)*. We say that the *diameter* of G is the maximum distance between any pair of nodes; and we denote this quantity by *diam(G)*.

Let's define a related quantity, which we'll call the *average pairwise distance* in G (denoted *apd(G)*). We define *apd(G)* to be the average, over all $\binom{n}{2}$ sets of two distinct nodes u and v, of the distance between u and v. That is,

$$apd(G) = \left[\sum_{(u,v)\subseteq V} dist(u,v)\right] / \binom{n}{2}.$$

Here's a simple example to convince yourself that there are graphs G for which $diam(G) \neq apd(G)$. Let G be a graph with three nodes u, v, w, and with the two edges $\{u, v\}$ and $\{v, w\}$. Then

$$diam(G) = dist(u, w) = 2,$$

while

$$apd(G) = [dist(u, v) + dist(u, w) + dist(v, w)]/3 = 4/3.$$

Of course, these two numbers aren't all *that* far apart in the case of this three-node graph, and so it's natural to ask whether there's alway a close relation between them. Here's a claim that tries to make this precise.

> *Claim: There exists a positive natural number c so that for all connected graphs G, it is the case that*
>
> $$\frac{diam(G)}{apd(G)} \leq c.$$

Decide whether you think the claim is true or false, and give a proof of either the claim or its negation.

Problem 5. An *strong articulation point* in an undirected graph is a vertex whose removal separates the graph into at least three connected components.

(a) Give an efficient algorithm to find all of the strong articulation points of a graph. Both describe your algorithm briefly in English, and write the pseudo-code.

(b) How fast is your algorithm?

Problem 6. A *Hamiltonian cycle* in a graph is a cycle that visits every vertex exactly once.

(a) Show that if a graph has a Hamiltonian cycle then it does not have an articulation point.

(b) What is the converse?

(c) Decide whether you think the converse is true or false, and give a proof of either the converse or its negation.

Problem 7. A *Hamiltonian path* in a graph is a path that visits every vertex exactly once.

(a) Show that if a graph has a Hamiltonian path then it does not have a strong articulation point.

(b) What is the converse?

(c) Decide whether you think the converse is true or false, and give a proof of either the converse or its negation.