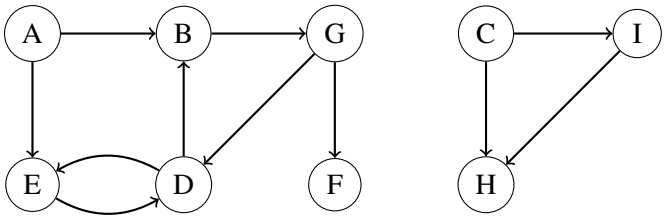
CS 170 Algorithms
Spring 2015 Christos Papadimitriou and Prasad Raghavendra **HW 4**

Due Feb. 20, 5:00pm

Instructions. This homework is due Friday, February 20, at 5:00pm electronically. Same rules as for prior homeworks.

1. (10 pts.) Depth-First Search Warm-up



(a) Run DFS at node A, trying to visit nodes alphabetically (i.e. given a choice between nodes D and F, visit D first).

- List the nodes in the order you visit them (so each node should appear in the ordering exactly once).
- List each node with its pre- and post-number. The numbering starts from 1 and ends at 18.
- Label each edge as **T**ree, **B**ack, **F**orward or **C**ross.

(b) How many strongly connected components are there?

2. (20 pts.) Minimal Graphs

(a) A city has n intersections, and m undirected roads. However, road maintenence is getting really expensive, so the city would like to reduce the number of roads. They want to do this without effecting the overall connectivity of the intersections.

More formally, you are given as input a graph $G = (V,E)$, with edges undirected. You would like to create a new graph $G' = (V,E')$, with edges undirected and $|E'|$ minimized with the following constraint that if there exists a path from u to v in G, then there exists a path from u to v in G'.

Give an efficient algorithm to determine the minimum number of undirected edges we need in the modified graph. Note that if there isn't a path from u to v in G, then there may or may not be a path from u to v in G'.

(b) Repeat the same excercise, but now for directed graphs. More specifically, you are given as input a graph $G = (V,E)$ with edges **directed**. You would like to create a new graph $G' = (V,E')$ with edges **directed** and $|E'|$ minimized with the following constraint that if there is a path from u to v in G, then there must be a path from u to v in G'. (Note that we don't necessarily need a path from v to u).

Give an efficient algorithm to determine the minimum number of directed edges we need in the modified graph. Note again that if there isn't a path from u to v in G, then there may or may not be a path from u to v in G'.

3. (10 pts.) DAG Revisited

Design an algorithm that takes a directed acyclic graph, $G = (V, E)$, and determines whether G contains a directed path that touches every vertex exactly once. The algorithm should run in $O(|V| + |E|)$ time.

4. (20 pts.) Unique Shortest Path

Shortest paths are not always unique: sometimes there are two or more different paths with the minimum possible length. Show how to solve the following problem in $O((|V| + |E|) \log |V|)$ time.

Input: An undirected graph $G = (V, E)$; edge lengths $l_e > 0$; starting vertex $s \in V$.

Output: A Boolean array `usp[·]`: for each node u, the entry `usp[u]` should be `true` if and only if there is a *unique* shortest path from s to u. (Note: `usp[s] = true`.)

5. (30 pts.) Biconnected Components

Let $G = (V, E)$ be an undirected graph. For any two edges $e, e' \in E$, we say that $e \sim e'$ if either $e = e'$ or there is a (simple) cycle containing both e and e'. We say a set of edges C is a *biconnected component* if there is some edge $e \in E$ such that $C = \{e' \in E | e \sim e'\}$.

(a) Show that two distinct biconnected components cannot have any edges in common.

(b) Associate with each biconnected component all the vertices that are endpoints of its edges. Show that the vertex sets corresponding to two different biconnected components are either disjoint or intersect in a single vertex. Such a vertex is a *separating vertex*.

We will now walk you through how you can use DFS to identify the biconnected components and separating vertices of a graph in linear time. Consider a DFS tree of G.

(c) Show that the root of the DFS tree is a separating vertex if and only if it has more than one child in the tree.

(d) Show that a non-root vertex v of the DFS tree is a separating vertex if and only if it has a child v' none of whose descendants (including itself) has a backedge to a proper ancestor of v.

For each vertex u define $\texttt{pre}(u)$ to be the pre-visit time of u (as described in section 3.2.4 of the textbook). Define $\texttt{low}(u)$ to be the minimum possible value of $\texttt{pre}(w)$, where either $w = u$ or there exists a backedge (v, w) for $v = u$ or some descendant v of u.

Another way of stating the result of (d) is that a non-root vertex u is a separating vertex if and only if $\texttt{pre}(u) \leq \texttt{low}(v)$ for any child v of u.

(e) Give an algorithm that computes all separating vertices and biconnected components of a graph in linear time. (Hint: Think of how to compute `low` in linear time with DFS. Use `low` to identify separating vertices and run an additional DFS with an extra stack of edges to remove biconnected components one at a time.)

6. (1 pts.) Road Network Design (Extra credit)

There is a directed network of roads $G = (V, E)$ connecting a set of cities V. Each road in E has an associated length $l_e > 0$. There is a proposal to add k new roads to this network, and there is a list E' of pairs of cities between which the new roads can be built. Each such potential road $e' \in E'$ has an associated length. As a designer for the public works department you are asked to select k roads of E' whose addition to the existing network G would result in the maximum decrease in the driving distance between two fixed cities s and t in the network. Give an efficient algorithm for solving this problem. Your time complexity should depend on $|V|$, $|E|$, and k.