Solutions to Homework 1: Basics and DFS

Solution 1: We have omitted the drawing of G^R. For parts (b)–(d), see Fig. 1. The final strong components are $\{f, g, h, i, j\}$, $\{a, b\}$, and $\{c, d, e\}$.

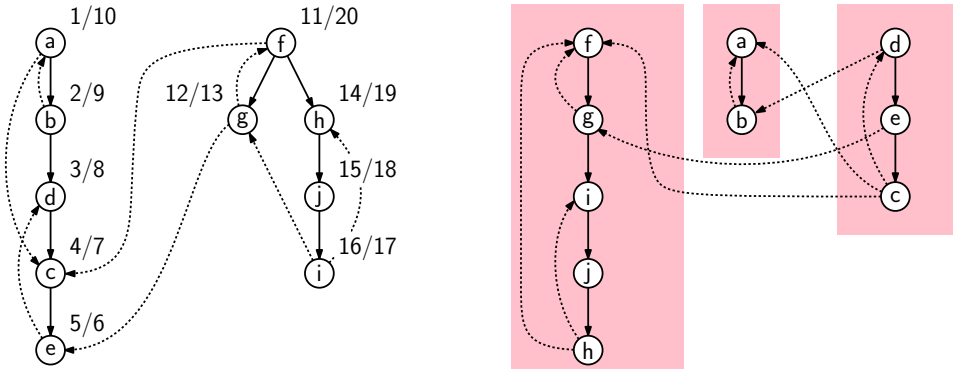

Figure 1: Solution to Problem 1.

Solution 2: See Fig. 2. The bridges are (f, h), (h, g), and (i, j). The final 2-edge connected components are $\{a, b, e, f, i\}$, $\{h\}$, $\{c, d, g\}$, and $\{j, k, \ell\}$.

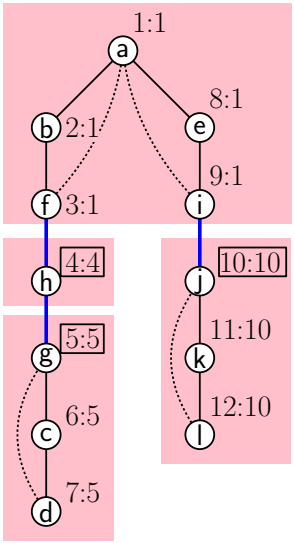

Figure 2: Solution to Problem 2.

Solution 3:

(a) **The root of the DFS tree is a cut vertex if and only if it has two or more children:**

(⇒) Recall that in the DFS of an undirected graph, there can be no cross edges. If the root has two or more children, then there can be no edges between the subtrees rooted at these two children. Thus, the removal of the root will cause the graph to be disconnected (see Fig. 3(a)).

(⇐) Conversely, if the root has one child, then if it were to be removed, the rest of the DFS tree would still be connected, thus the entire graph would be connected. If it had zero children then this is the only node and this is trivially true.

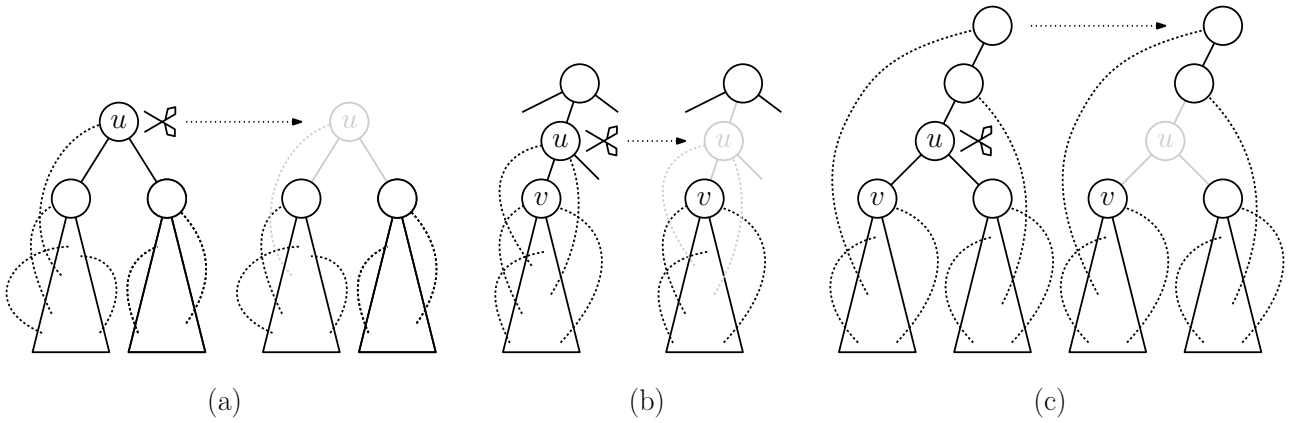

Figure 3: Solution to Problem 3.

(b) **No leaf of the DFS tree can be a cut vertex.**

If a leaf is removed from a tree, the tree remains connected. Since the DFS tree is a subgraph of G, it follows that the removal of a leaf cannot disconnect G, and hence no leaf can be a cut vertex.

(c) **A non-root, internal vertex u of the DFS tree is a cut vertex if and only if it has a child v such that** $\text{Low}[v] \geq d[u]$**:**

(⇒) If a non-root, internal vertex u has a child v such that $\text{Low}[v] \geq d[u]$, then (by definition of $\text{Low}[v]$) no back edge in v's subtree goes to a proper ancestor of u. It follows that if u were deleted, then this entire subtree would become disconnected from the rest of the graph (see Fig. 3(b)). (Note that since u is not the root, the rest of the graph is not empty.)

(⇐) Conversely, if every child of u has a back edge that leads to a proper ancestor of u, then the removal of u would not disconnect the graph, since these back edges would serve as "detours" connecting each of u's descendants to the rest of the graph (see Fig. 3(c)).

Solution 4: The solution is remarkably easy. It involves simply running Bellman-Ford (as given in class) using u as the source vertex. (If you like, you could force it to stop after k iterations of the while loop, but we will show below that it will converge after at most k iterations, so the algorithm will terminate on its own after the $(k+1)$st iteration.) On termination, output $d[v]$ as the distance

from u to v, and the shortest path arises by tracing the predecessor links from v back to u and reversing the resulting sequence.

The only issue that remains to be shown is that the algorithm will converge after k iterations of the repeat-loop. This is a consequence of the induction hypothesis in our correctness proof given in the lecture notes. We showed there that if the shortest path from the source u to any vertex v consists of i edges, then the after the ith iteration of the repeat-loop, we will have $d[v] = \delta(u, v)$. (That is, we have converged to the true distance from u to v.) By the hypothesis that all shortest paths in G consist of at most k edges, it follows directly that the algorithm will converge after k iterations, and hence it will terminate after $k+1$ iterations (where the last iteration is just to verify that we have really converged). For the sake of completeness, let us prove this from first principles.

Lemma: If the shortest path from u to v has at most k edges, then after the k-th iteration of the repeat-loop in the Bellman-Ford algorithm, $d[v] = \delta(u, v)$.

Proof: Let $u = u_0, u_1, \ldots, u_j = v$ denote the sequence of vertices on the shortest path from u to v, where $j \leq k$. We will show by induction on i that after the ith iteration of the repeat loop, $d[u_i] = \delta(u, u_i)$. (That is, the distances up to vertex i in the sequence have converged to their correct values.)

The basis case is trivial, since by the initialization in Bellman-Ford, $d[u] = 0$, and $\delta(u, u)$ is clearly zero. So, assume that $i \geq 1$. Because this is the shortest path, we know that for $1 \leq i \leq j$, $\delta(u, u_i) = \delta(u, u_{i-1}) + w(u_{i-1}, u_i)$. Assume inductively that after the $(i-1)$st iteration of the while loop, we have $d[u_{i-1}] = \delta(u, u_{i-1})$. During the ith iteration of the repeat loop, when we apply relax(u_{i-1}, u_i), the Bellman-Ford algorithm will enforce the condition that $d[u_i] \leq d[u_{i-1}] + w(u_{i-1}, u_i)$. Thus, by the induction hypothesis, we have

$$d[u_i] \;\leq\; d[u_{i-1}] + w(u_{i-1}, u_i) \;=\; \delta(u, u_{i-1}) + w(u_{i-1}, u_i) \;=\; \delta(u, u_i).$$

As argued in class, $d[u_i]$ is based on the cost of an actual path in G, therefore $d[u_i] \geq \delta(u, u_i)$. In conclusion, we have $d[u_i] = \delta(u, u_i)$. By induction, after $j \leq k$ iterations, we have $d[v] = d[u_j] = \delta(u, u_j) = \delta(u, v)$. Since we have converged to the optimal value, further iterations will not alter this.

Solution to the Challenge Problem: There are a number of different solutions. Here is one.

First off, let us assume that G is connected. If not, we can run this procedure on each of its connected components. Run DFS on G. If there are no back edges, then G is a tree. Because a tree on n vertices has only $n-1$ edges, by the pigeonhole principal, in any direction of the edges, at least one vertex does not have an incoming edge. Therefore, we can report that such an orientation is impossible.

Otherwise, the DFS of G must have a back edge. Let (u, v) be this back edge. We know that both of these vertices, v in particular, lies on a cycle. Run DFS again starting at v. Because v is on a cycle, the root v of the resulting DFS tree has a back edge directed into it. Direct this back edge into v, and direct all the tree edges from parent to child. (The remaining back edges can be oriented in any manner.) As a result, every non-root node has an edge entering from its parent, and the root has an incoming edge from the back edge. Therefore, we have achieved the desired source-free orientation.

Correctness follows directly from the discussion. Clearly, the running time of the algorithm is dominated by the time to perform the two DFSs, which is $O(n + m)$.