Assignment 2

Sample Solutions

CSCI 3110 — Summer 2018

Question 1

The algorithm. The following is a simple algorithm that takes $O(nm)$ time to decide whether a given connected graph $G = (V, E)$ is 2-edge-connected: Compute a spanning tree T of G. For every edge $e \in T$, test whether $G_e = (V, E \setminus \{e\})$ is connected. If G_e is connected for all $e \in T$, then report that G is 2-edge connected; otherwise, report that G is not 2-edge connected.

Running time. Computing a spanning tree T of G takes $O(n+m)$ time using BFS or DFS. For $n \geq 1$ and $m \geq 1$, this is in $O(nm)$. Constructing G_e from G takes constant time for each edge $e \in T$ because removing an edge from a graph in adjacency list representation takes constant time. (In order to get ready for constructing $G_{e'}$ for the next edge $e' \in T$, we need to restore G to its original state by adding e to G_e again, but this also takes constant time.) To test whether G_e is connected, we can compute the connected components of G_e and count them. This takes $O(n+m)$ time as discussed in class. Since G is connected, we have $m \geq n-1$, so $O(n+m) = O(m)$. Finally, observe that every tree T on n vertices has $n-1$ edges. Thus, testing whether all graphs G_e with $e \in T$ are connected takes $O(nm)$ time. In total, the running time of the algorithm is thus $O(nm)$.

Correctness. If the algorithm identifies an edge $e \in T$ such that G_e is not connected, then its answer is clearly correct: it just identified an edge whose removal disconnects G. So assume that G_e is connected for every edge $e \in T$. Since G is 2-edge-connected exactly if G_e is connected for every edge $e \in G$, we have to show that G_e being connected for every edge $e \in T$ implies that G_e is connected for every edge $e \in G$. For every edge $e \in T$, the algorithm verifies explicitly that G_e is connected. If $e \notin T$, then observe that T itself is connected (because it is a spanning tree of G). Thus, there exists a path P_{uv} in T between every pair of vertices $u, v \in V$. Since $T \subseteq G$, this path also exists in G. Since $e \notin T$ and $P_{uv} \subseteq T$, P_{uv} is also a path in G_e. Since this is true for every pair of vertices $u, v \in V$, G_e is thus connected. This finishes the proof.

Question 2

The key observation. For every vertex $v \in F$, let $\alpha(v)$ be its preorder number and let $\beta(v)$ be its postorder number. The key claim is

Lemma 1 *A vertex u is an ancestor of another vertex v if and only if $\alpha(u) \leq \alpha(v)$ and $\beta(u) \geq \beta(v)$.*

Proof "Only if." If u is an ancestor of v, then the definition of a preorder numbering implies that $\alpha(u) \leq \alpha(v)$ and the definition of a postorder numbering implies that $\beta(u) \geq \beta(v)$.

"If." Assume u is not an ancestor of v but $\alpha(u) \le \alpha(v)$ and $\beta(u) \ge \beta(v)$. Since $\alpha(u) \le \alpha(v)$, u cannot be a proper descendant of v because a preorder numbering numbers every vertex before all its descendants. Thus, neither u nor v is an ancestor of the other. Let a be the lowest common ancestor of u and v and let u' and v' be the children of a that are ancestors of u and v, respectively. Since $\alpha(u) \le \alpha(v)$, the definition of a preorder numbering implies that $\alpha(u') < \alpha(v')$ and thus u' is to the left of v' in the list of a's children. By the definition of a postorder numbering, this implies that $\beta(u') < \beta(v')$ and thus $\beta(u) < \beta(v)$, a contradiction. This shows that $\alpha(u) \le \alpha(v)$ and $\beta(u) \ge \beta(v)$ implies that u is an ancestor of v. $\square$

The data structure. The data structure consists of two arrays A and B where $A[v] = \alpha(v)$ and $B[v] = \beta(v)$.

Cost of constructing the data structure. We compute a preorder numbering α of F and store $\alpha(v)$ in $A[v]$ for each vertex $v \in F$. This takes $O(n)$ time. Similarly, constructing $B[v]$ takes $O(n)$ time. Thus, the data structure can be constructed in $O(n)$ time and clearly uses linear space because it consists of two arrays of size n.

The query procedure. Given a pair of vertices (u, v), we decide whether u is an ancestor of v by accessing $A[u]$, $A[v]$, $B[u]$, and $B[v]$ and testing whether $A[u] \le A[v]$ and $B[u] \ge B[v]$. If so, we answer yes; otherwise, we answer no. Since this procedure involves four memory accesses and two comparisons, it clearly takes constant time. Its correctness follows immediately from Lemma 1.