CS4311 Design and Analysis of Algorithms

Homework 4 (Suggested Solution)

1. (a) **Ans.** Let r be the root of T. To prove the desired statement, it is equivalent if we show the following:

   (i) If r has at most one child, r is not an articulation point.

   (ii) If r has at least two children, r is an articulation point.

   For (i), if r has no child, the graph must have exactly one node, so that in this case, r is not an articulation point. On the other hand, if r has one child, say c, we know that if we remove r, only the edge (r, c) and the back edges to r will be removed. On the other hand, all tree edges, apart from (r, c), are still present, so that all other nodes will still be connected. Thus, r is not an articulation point.

   For (ii), r has at least two children. Let u be the child of r whose discovery time is the earliest, and v be another child of r. Then u must be discovered right after r, so that at time $d(u)$, all other nodes apart from r, are still undiscovered. Since v does not become u's descendant, by white-path theorem, there must not be a path between u and v without passing r. In other words, removing r must disconnect u and v, so that r is an articulation point.

   (b) **Ans.** To prove the desired statement, it is equivalent if we show the following:

   (i) If v has a child s such that there is no back edge from s or from any descendant of s to a proper descendant of v, v is an articulation point.

   (ii) If for each child s of v, there is some back edge from s or from some descendant of s to a proper descendant of v, v is not an articulation point.

   To ease our discussion, let p be the parent of v in the DFS tree.

   For (i), consider the subtree of the DFS tree rooted at s. For each edge with an endpoint w in this subtree, its other end-point, say x, must either be v or within the subtree. (The reason is that: If (w, x) is a tree edge, x must be in the subtree; otherwise, (w, x) is a back edge, and by the given condition, w cannot link to a proper ancestor of v, so that it must link to v or a node in the subtree). In this case, we see that if we remove v, w and p must be disconnected. This implies v must be an articulation point.

   For (ii), let $s_1, s_2, \ldots, s_r$ be the children of v in the DFS tree. Now, consider removing v from the graph. For the DFS tree, it will be partitioned into exactly $(r+1)$ connected components, where the vertices $p, s_1, s_2, \ldots, s_r$ will be in distinct components.

   However, from our condition, each s_i must be connected to p in G. Thus, the graph G after removal of v is still connected, so that v is not an articulation point.

   (c) **Ans.** We perform a post-order traversal on T. For each vertex v, we will set its $low[v]$ value when it is encountered.

   Suppose the children of v are $c_1, c_2, \ldots, c_k$. By our traversal order, at the time v is encountered, $low[c_1], low[c_2], \ldots, low[c_k]$ are already computed. It is easy to see that $low[v]$ is the minimum among `(i)` all $low[c_i]$'s, `(ii)` $d(v)$, and `(iii)` $d(w)$ for all (v, w) is a back edge from v. Thus, $low[v]$ can be found in $O(\deg(v))$ time, where $\deg(v)$ denotes the degree of v in the original graph G. The total time to find all low values is $O(\sum_v \deg(v)) = O(|E|)$.

(d) **Ans.** Let v be a non-root node, and s be a child of v. It is easy to check that $low[s] < d(v)$ if and only if s or some descendant of s has a back edge to a proper ancestor of v.

Thus, once $low[v]$ is computed for each non-root vertex v, we can decide if v is an articulation point by examining the *low* values of all its children. The time for this process is $O(|V|)$. For the root r, we can decide if it is an articulation point in $O(1)$ time by checking its degree in the DFS tree.

The time of the above process requires a traversal in the DFS tree, which is $O(|V|)$ time. By combining the time to compute all *low* values, the total time for finding all articulation points is $O(|V| + |E|)$.

2. (a) **Ans.** To prove the statement, it is equivalent if we prove the following:

(i) If there is no edge from v_i to v_{i+1}), G is not semi-connected.

(ii) If there is an edge (v_i, v_{i+1}) for all i, G is semi-connected.

For (i), we see that there is no path from v_i to v_{i+1}, and there is no path from v_{i+1} to v_i Thus, G is not semi-connected.

For (ii), there is a path $(v_1, v_2, \ldots, v_n)$ so that for each pair of vertices v_i and v_j, they are connected. Thus, G is semi-connected.

(b) **Ans.** We first find all SCCs and form the component graph S of G, which is a DAG. It is easy to check that if S is not semi-connected, G must not be semi-connected. On the other hand, if S is semi-connected, we can also show G is semi-connected; precisely, we shall show that for each u and v in G, either $u \leadsto v$ or $v \leadsto u$:

(Case 1:) If both u and v are in the same SCC, $u \leadsto v$;

(Case 2:) Otherwise u and v are in different SCCs. Let C_u and C_v denote the SCC containing u and v, respectively. WLOG, suppose that $C_u \leadsto C_v$ in the component graph. Since vertices in the same component is strongly connected, the above implies that there must be a path from u to v in the original graph G. Thus, $u \leadsto v$.

Thus, to decide if G is semi-connected, we can first construct the component graph S of G, and test if S is semi-connected. The time for the construction of component graph is $O(|V|+|E|)$ and the testing of S is done by a topological sort in $O(|V|+|E|)$ time. This gives a total of $O(|V| + |E|)$ time as desired.