CSCI3160: Regular Exercise Set 8

Prepared by Yufei Tao

Problem 1. Consider the SCC graph G^{scc} discussed in our lecture. Prove: G^{scc} is a DAG (directed acyclic graph).

Solution. Suppose that G^{scc} contains a cycle. Let S_1 and S_2 be two arbitrary SCCs inside the circle. By how G^{scc} is constructed, we can infer:

- in G, each vertex of S_1 can reach all the vertices of S_2;
- in G, each vertex of S_2 can reach all the vertices of S_1.

Thus, S_1 violates the maximality condition of SCC, yielding a contradiction.

Problem 2. Let $G = (V, E)$ be a directed simple graph stored in the adjacency-list format. Define $G^{rev} = (V, E^{rev})$ be the reverse graph of G, namely, $E^{rev} = \{(v, u) \mid (u, v) \in E\}$. Design an algorithm to produce the adjacency list of G^{rev} in $O(|V| + |E|)$ time. You can assume that $V = \{1, 2, ..., n\}$.

Solution. First, create an empty linked list $L(u)$ for each vertex $u \in V$, and initialize an array A of size $|V|$ where $A[u]$ stores the head pointer to $L(u)$ (note: u is an integer). For each vertex $u \in V$, the adjacency list of G stores the out-neighbors of u in a linked list; we scan this linked list and, for each out-neighbor v of u, add u to $L(v)$. After completing the procedure for all $u \in V$, the set $\{L(u) \mid u \in V\}$ constitutes the adjacency list of G^{rev}.

Problem 3. Implement the SCC algorithm discussed in our lecture in $O(|V| + |E|)$ time. You can assume that $V = \{1, 2, ..., n\}$.

Solution. To implement Step 1, simply perform DFS on the input graph $G = (V, E)$ in $O(|V|+|E|)$ time. Store the turn-black order in an array A, namely, $A[i] = u$ (for $i \in [1, n]$) if vertex $u \in V$ has label i. It is easy to generate A during the aforementioned DFS without increasing the time complexity.

Step 2 can be completed using the solution to Problem 2.

To implement Step 3, start DFS from vertex $A[n]$ (i.e., the vertex having the largest label). When a restart is needed, examine $A[n-1]$, $A[n-2]$, ... until reaching the first vertex $A[i]$ whose color is still white. Start the second DFS with $A[i]$. When another restart is needed, choose the starting vertex in the same manner. Repeat the above until all vertices have been visited by DFS.

Problem 4. Let $G = (V, E)$ be a DAG, where each vertex $u \in V$ carries an integer *weight* denoted as w_u. Let $R(u)$ be the set of vertices in G that u can reach (i.e., for each vertex $v \in R(u)$, G has a path from u to v); note that $u \in R(u)$ (i.e., a node can reach itself). Define $W(u) = \min_{v \in R(u)} w_v$. Design an algorithm to compute the $W(u)$ values of all $u \in V$ in $O(|V| + |E|)$ time.
(Hint: dynamic programming).

Solution. For each $u \in V$, let $\mathrm{Out}(u)$ be the set of out-neighbors of u. We have:

$$W(u) = \begin{cases} w_u & \text{if } \mathrm{Out}(u) = \emptyset \\ \min\{w_u, \min_{v \in \mathrm{Out}(u)} W(v)\} & \text{otherwise} \end{cases}$$

We can therefore calculate the $W(u)$ values of all $u \in V$ by dynamic programming (go over the vertices by reversing a topological order).

Problem 5*. Let $G = (V, E)$ be an arbitrary directed simple graph, where each vertex $u \in V$ carries an integer *weight* denoted as w_u. Let $R(u)$ be the set of vertices in G that u can reach; note that $u \in R(u)$. Define $W(u) = \min_{u \in R(u)} w_u$. Design an algorithm to compute the $W(u)$ values of all $u \in V$ in $O(|V| + |E|)$ time.

Solution. Observe that if u and v belong to the same SCC of G, then $R(u)$ is exactly the same as $R(v)$.

First, obtain the SCCs of G in $O(|V| + |E|)$ time and then generate the SCC graph G^{scc} in $O(|V| + |E|)$ time (this is a special exercise of this week). For each SCC S, define the weight of its vertex in G^{scc} as $w_S = \min_{u \in S} w_u$. Define $R^{scc}(S)$ as the set of vertices in G^{scc} that S can reach, and define $W(S) = \min_{T \in R^{scc}(S)} w_T$. Use the solution to Problem 4 to find the $W(S)$ values for all the vertices S in G^{scc}.

For every vertex u in G, its $W(u)$ value equals exactly $W(S)$ where S is the SCC containing u.