Problem 1. An SCC graph G^{scc} of G is defined as follows:

- Each vertex in G^{scc} is a distinct SCC in G.
- For every two distinct SCCs S_i and S_j in G, G^{scc} has an edge from S_i to S_j if some vertex of S_i has an edge in G to some vertex of S_j.

Prove: G^{scc} is a DAG (directed acyclic graph).

Proof. Let $G = (V, E)$. Suppose that G^{scc} contains a cycle $C = S_0 - S_1 - \ldots - S_k$ where $S_k = S_0$. Hence, $(S_{i-1}, S_i) \in E^{\text{scc}}, \forall 1 \leq i \leq n$, meaning that $\exists s_{i-1} \in S_{i-1}, t_i \in S_i$ such that $(s_{i-1}, t_i) \in E$. Additionally, there is a path from t_i to s_i in G for all $1 \leq i \leq n$, by the definition of SCC that any two vertices in the same SCC are mutually reachable. Hence, any two vertices u, v satisfying $u \in S_i, v \in S_j$ where $1 \leq i, j \leq k$ are mutually reachable. This violates the definition that each vertex in G^{scc} is a distinct SCC. □

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

Solution. First, create an empty 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 $L(u)$. For each vertex $u \in V$, the adjacency list of G stores the out-neighbours of u in a linked list, which are all the edges in G. Scan the linked list, and for each outneighbour v of u, we add u to $L(v)$, indicating the edge from v to u.

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

Solution. Perform DFS on the input graph $G = (V, E)$ in $O(|V| + |E|)$ time. Create a time stamp variable t for the number of vertices already popped. Let A be a list recording the order of vertices that turned black. Then, when v is being popped, then increment t and set $A[t] = v$.

The reversed graph G^{rev} can be obtained in $O(|V| + |E|)$ time.

Scan through A from the last index, if $A[i]$ is not yet visited, then start the DFS from vertex $A[i]$, this discovers a connected component. Repeat this process until the scan through A is completed.

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 $W(u) = \min_{u \in R(u)} w_u$. Design an algorithm to compute the $W(u)$ values of all $u \in V$ in $O(|V| + |E|)$ time. (Hint: dynamic programming).

Solution. For every $v \in V$, let $S(v)$ be the set of outneighbours of v. Then,

$$R(v) = \left(\bigcup_{u \in S(v)} R(u) \right) \cup \{v\}, W(v) = \begin{cases} \min \left\{ \min\limits_{u \in S(v)} W(u), w_v \right\}, & \text{if } S(v) \neq \emptyset, \\ w_u, & \text{otherwise.} \end{cases}$$

Compute the topological order of G in $O(|V| + |E|)$ time. Following the reversed topological order, all $W(u)$'s for $u \in S(v)$ have been computed before computing $W(v)$. Only $O(|S(v)|)$ time is needed to obtain the minimum. Hence, $W(u)$ can be computed in $O(|V| + \sum_v |S(v)|) = O(V+E)$ time as required.

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. Consider two vertices u, v that belong to the same SCC and thus u and v are mutually reachable. This guarantees that any point reachable from u is reachable from v, and vice versa, implying that $R(u) = R(v)$. Obtain G^{scc} in $O(|V| + |E|)$ time. For each SCC vertex S_i, reassign a weight by taking the minimum over all weights that are originally assigned to any vertex in S_i, i.e. $\min_{v \in S_i} w_v$. Let $R(S_i)$ be the set of vertices in G^{scc} that S_i can reach and define $W(S) = \min_{T \in R(S_i)} w_T$. Then, we can compute $W(S)$ in $O(|V| + |E|)$ time by dynamic programming. Additionally, for any u, $W(u)$ is exactly $W(S)$ where S is the SCC containing u.

Problem 6. Prove: all the SCCs of a directed simple graph are mutually disjoint.

Proof. Suppose the otherwise that there exists a pair of non-disjoint SCCs. Let S_1 and S_2 the pair, where $S_1 \cap S_2 \neq \emptyset$. Suppose that $e \in S_1 \cap S_2$, then consider any $e_1 \in S_1 \backslash S_2, e_2 \in S_2 \backslash S_1$. Clearly, e and e_1 are mutually reachable, e and e_2 are also mutually reachable. Hence, e_1 can reach e_2 by stumbling onto e and follow the path where e can reach e_2. The same applies to e_2. Hence, e_1 and e_2 are mutually reachable, meaning that $e_1, e_2 \in S_1 \cap S_2$, which is a contradiction. $\square$

Problem 7. Let $G = (V, E)$ be a directed simple graph and G^{scc} be the SCC graph of G. Let S_1 and S_2 be two SCCs of G. Prove: if S_1 cannot reach S_2 in G^{scc}, then no vertex of S_1 can reach any vertex of S_2 in G.

Proof. Suppose the otherwise that some vertex of S_1, u, is able to reach a vertex of S_2, v, in G. Then, consider the path $u - u_1 - \ldots - u_k - v$. Clearly, if two nodes belongs to the different SCCs S_i and S_j, then there is an edge from S_i to S_j in G^{scc} by the definition of SCC graph. Hence, a walk is constructed, starting from S_1 and ends at S_2 in G^{scc}, which is a contradiction. $\square$

Problem 8. Prove: G and G^{rev} have the same SCCs.

Proof. Consider any two vertices u, v in the same SCC S on the original graph G by the paths P_1, P_2, one from u to v, the other from v to u. Hence, in the reversed graph G^{rev}, we can go from v to u through the edges of P_1 (since they are reversed in G^{rev}), and similarly, from u to v by P_2. Hence, u and v belong to the same SCC in G^{rev}. Let the SCC which u, v belong to in G^{rev} be S', then $S \subseteq S'$.

Interchanging G^{rev} and apply the above argument implies that $S' \subseteq S$ and hence $S = S'$, which is the desired. $\square$

Problem 9. Prove: if an SCC S_1 has a *path* to an SCC S_2 in G^{scc}, then $label(S_1) > label(S_2)$.

Proof. Let u be the first vertex in $S_i \cup S_j$ that turns gray in DFS.

If $u \in S_i$, u has a white path to every vertex in $S_i \cup S_j$. By the white path theorem, u turns black after all the vertices in S_j and is the last vertex in S_i turning black. This implies $label(S_i) > label(S_j)$.

If $u \in S_j$, u has a white path to every vertex in S_j but no white path to any vertex in S_i. By the white path theorem, u turns black after all the vertices in S_j and before every vertex in S_i. This again implies $label(S_i) > label(S_j)$.

$\square$

Problem 10. Prof. Goofy proposes his own SCC algorithm:

- Step 1: Perform DFS on the input graph G and compute a label for each vertex.
- Step 2: Perform another DFS on G subject to the following rules:
  - Start the first DFS from the vertex with the smallest label;
  - Whenever a restart is needed, do so from the white vertex with the smallest label.

Give a counterexample to this algorithm.

Solution.

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 4$$

For this input, vertex $1, 2, 3, 4$ has label $4, 3, 2, 1$, respectively. Hence, the algorithm marks $1, 2, 3, 4$ together as one SCC, while each vertex is a disjoint SCC in itself.