Homework 10

● Graded

Student

Nick Zhu

Total Points

100 / 100 pts

Question 1

Problem 1 — **25** / 25 pts

✓ **– 0 pts** Correct

Question 2

Problem 2 — **35** / 35 pts

✓ **– 0 pts** Correct

Question 3

Problem 3 — **40** / 40 pts

3.1 **a (i)** — **10** / 10 pts

✓ **– 0 pts** Correct

3.2 **a (ii)** — **10** / 10 pts

✓ **– 0 pts** Correct

3.3 **b** — **20** / 20 pts

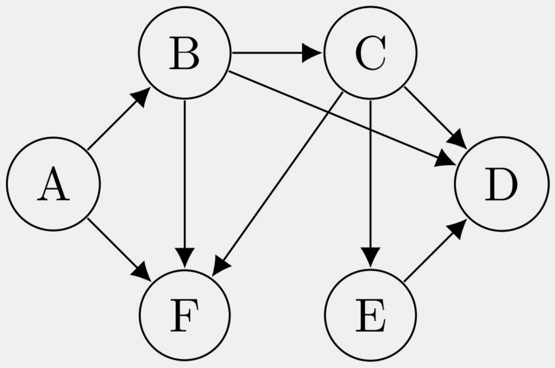
✓ **– 0 pts** Correct

Question assigned to the following page: 1

Problem 1 (Topological Sort, 25 pts)

How many valid topological sorts does the directed graph below have? List all the valid topological sorts in the following table. One of them has been listed as an example, where node A is output first and D is output last.

1.	A	B	C	F	E	D
2.						



Solution

Observe that the only precedence constraints are

$$A \to B \to C, \quad C \to E \to D, \quad \text{and} \quad A \to F,\ B \to F,\ C \to F,$$

so every valid ordering must begin A, B, C, then place F after C and respect $E \to D$. We must interleave the chain $E \to D$ with the single element F, which yields exactly three possibilities:

#	1	2	3	4	5	6
1.	A	B	C	F	E	D
2.	A	B	C	E	F	D
3.	A	B	C	E	D	F

Hence, there are $\boxed{3}$ valid topological sorts.

Question assigned to the following page: 2

Problem 2 (Reachability, 35 pts)

Suppose you are given a directed graph $G = (V, E)$ where each vertex, $u \in V$, is labeled with a unique value $L(u)$ from the set $\{1, \ldots, |V|\}$. For each vertex u, let $R(u)$ denote the set of vertices *reachable* from u in G. Define $\max(u)$ to be the *vertex* v_u^* in $R(u)$ with the maximum label, i.e. $L(v_u^*) \geq L(v)$ for all $v \in R(u)$.

Give an $O(|V| + |E|)$ time algorithm to compute $\max(u)$ for all vertices in $u \in V$. Briefly justify correctness and runtime of your algorithm.

Solution

We can compute $\max(u)$ for all $u \in V$ in $O(|V| + |E|)$ time by the following steps:

1. **Compute strongly-connected components (SCCs).** Run Kosaraju-Sharir algorithm on G to partition V into SCCs. Call the SCC containing u by $\text{comp}(u)$. This takes $O(|V| + |E|)$ time.

2. **Build the condensed DAG.** Form a DAG $G' = (V', E')$ whose vertices V' are the SCCs of G, and where there is an edge $(C \to C') \in E'$ iff there exists $(u \to v) \in E$ with $\text{comp}(u) = C$ and $\text{comp}(v) = C'$, $C \neq C'$. Since each original edge induces at most one edge in G', building G' is $O(|V| + |E|)$.

3. **Initialize local maxima.** For each SCC $C \in V'$, let

$$L_{\max}(C) = \max\{ L(u) : u \in C\},$$

and pick a representative vertex $r(C) \in C$ with $L(r(C)) = L_{\max}(C)$. Computing these takes $O(|V|)$.

4. **Topologically sort G'.** Since G' is a DAG, produce a topological order $C_1, \ldots, C_k$ of its SCCs in $O(|V'| + |E'|) = O(|V| + |E|)$.

5. **Dynamic-programming over the DAG.** Define an array $DP[\,\cdot\,]$ on components so that

$$DP[C] = \big(\text{max label reachable from } C \text{ in } G'\big) \quad \text{and} \quad \text{rep}[C] = \text{a vertex achieving that max.}$$

Process the C_i in *reverse* topological order $(C_k, \ldots, C_1)$. For each C_i:

$$DP[C_i] = \max\Big(L_{\max}(C_i), \max_{(C_i \to C_j) \in E'} DP[C_j]\Big),$$

and pick $\text{rep}[C_i]$ accordingly (either $r(C_i)$ or $\text{rep}[C_j]$ for which the maximum is attained). Each edge in E' is examined once, so this is $O(|V'| + |E'|)$.

6. **Assign answers back to original vertices.** Finally, for each $u \in V$ we have

$$\max(u) = \text{rep}\big(\text{comp}(u)\big),$$

in $O(|V|)$ total.

Correctness

1. SCC contraction preserves maximal labels. Vertices within the same SCC are mutually reachable, so the maximum label reachable from any $u \in C$ is at least the maximum label in C itself. By condensing each SCC into a single node and recording its local maximum $L_{\max}(C)$, we ensure no higher-labeled vertex inside C is ever forgotten.

2. Reachability respects the DAG structure. In the condensed graph G', there is an edge $C \to C'$ precisely when some $u \in C$ can reach some $v \in C'$ in one step. Any vertex reachable from u in the original graph either lies in $\text{comp}(u)$ or in some component C' reachable from $\text{comp}(u)$ along a directed path in G'. Thus, computing "max over successors" in G' exactly captures the global reachability in G.

3. Induction on reverse topological order. Let the topological order be $C_1, \ldots, C_k$. We prove by backward induction that after processing $C_i, \ldots, C_k$, $DP[C_j]$ equals the maximum label of any vertex reachable from component C_j for all $j \geq i$.

- *Base case:* For a sink C_k, there are no outgoing edges, so the set of reachable vertices is exactly C_k itself. We set $DP[C_k] = L_{\max}(C_k)$, which is correct.

- *Inductive step:* Suppose for all $j > i$, $DP[C_j]$ is correct. Consider C_i. Any vertex reachable from C_i either lies in C_i (so its label $\leq L_{\max}(C_i)$) or in some successor C_j with $j > i$. By induction, $DP[C_j]$ is the maximum label reachable from C_j. Taking the maximum over $L_{\max}(C_i)$ and all $DP[C_j]$ therefore yields the true maximum label reachable from C_i. Our update sets $DP[C_i]$ exactly to that value, and stores the corresponding representative.

4. Correctness of the final mapping. Every vertex u belongs to exactly one SCC, $\text{comp}(u)$. By construction, $\max(u)$ must be the highest-labeled vertex reachable from $\text{comp}(u)$. Since $\text{rep}(\text{comp}(u))$ records precisely that vertex, the final assignment $\max(u) = \text{rep}(\text{comp}(u))$ is exact.

Together, these invariants guarantee that for every original u, the algorithm outputs the unique vertex v_u^* of maximum label in $R(u)$.

Runtime.

As is already stated above, each step—SCC computation, DAG construction, topological sort, DP propagation, and final assignment—runs in $O(|V|+|E|)$. Hence the overall algorithm is $O(|V|+|E|)$.

Question assigned to the following page: 3.1

Problem 3 (Helping Botanical Garden, 40 pts)

Suppose the New York Botanical Gardens are trying to drum up interest for their tree and shrub collections by offering tours through the arboretum, and you are tasked with reviewing their proposed routes. There are a set V of trees they want some tour to stop at, and a set E of routes from tree to tree their proposed tours would make. In order to make sure all the trees are visited, they want the following property:

For all pairs $u, v \in V$ such that $u \neq v$, we must have a path from u to v ***or*** *a path from v to u (or both).*

Notice how this property is different from strongly connectedness. They provide you and your classmates with many proposed plans and a strict deadline for your feedback, and so you want to develop an efficient algorithm to automate this task.

(a) Suppose $G = (V, E)$ is a directed acyclic graph (DAG). Your classmates propose two possible algorithms to determine whether or not the given graph G satisfies the desired property. For each of the proposed algorithms below, **analyze its runtime and whether it is correct. If it is correct, give a justification. If it is incorrect, provide a counterexample.**

(i) Student A proposes the following algorithm: First, find a source vertex s in the graph that has no incoming edges. If there are multiple sources, pick an arbitrary one. Then, run a **DFS-Visit** (notice this is only the recursive portion of DFS, not the entire DFS) on G starting from s. G satisfies the property if and only if the DFS-Visit visits all the vertices in the graph.

(ii) Student B proposes the following algorithm: First, run DFS on the graph to obtain a topological sort of the vertices. Let the topological sort be $v_1, v_2, \ldots, v_n$. Then, we check for the following edges $(v_1, v_2), (v_2, v_3), (v_3, v_4), \ldots, (v_{n-1}, v_n)$. If all of these edges exist, then we say G satisfies the property. If any of the edges are missing, we say G does not satisfy the property.

(b) Now you want to generalize the algorithm to work for **any** directed graph. Design a $O(|V| + |E|)$ time algorithm to determine whether or not the given graph G satisfies the desired property. You may use the algorithms from part (a) as a subroutine. Make sure to justify the correctness and runtime of your proposed algorithm.

Solution

(a) We analyze the two proposed algorithms.

(i) **Student A's algorithm.**

- *Runtime:* Finding all in-degrees takes $O(|V| + |E|)$; the single `DFS_Visit` is $O(|V| + |E|)$. Total $O(|V| + |E|)$.
- *Correctness: Incorrect.*
  Counterexample:

$$V = \{1, 2, 3\}, \quad E = \{1 \to 2,\ 1 \to 3\}.$$

This graph is a DAG but does *not* satisfy the "for every pair $u \neq v$, either $u \to^* v$ or $v \to^* u$" property (there is neither a path $2 \to 3$ nor $3 \to 2$). Yet Student A's DFS from the unique source 1 *does* visit all three vertices, so the algorithm would (falsely) accept.

(ii) **Student B's algorithm.**

- *Runtime:* DFS with topo-order is $O(|V| + |E|)$, plus $O(n)$ checks of candidate edges, so its $O(|V| + |E|)$.
- *Correctness: Correct.*
  Justification: Student B's check is correct because in a DAG any topological order

$$v_1, \dots, v_n$$

ensures all edges go from earlier to later vertices. If, in addition, each consecutive pair has a direct edge

$$(v_i, v_{i+1}) \quad \text{for } i = 1, \dots, n-1,$$

then for any $i < j$ there is a path

$$v_i \to v_{i+1} \to v_{i+2} \to \cdots \to v_j,$$

so v_i reaches v_j. Conversely, if some (v_k, v_{k+1}) is missing, neither v_k reaches v_{k+1} nor v_{k+1} reaches v_k, violating the property. Therefore, checking exactly those $n-1$ edges is both necessary and sufficient.

(b) **General directed-graph algorithm.** We reduce to the DAG case in (a).

1. *Compute SCCs.* Run Kosaraju-Sharir algorithm on G in $O(|V| + |E|)$. Let comp(u) be the SCC containing u.
2. *Condense to a DAG.* Build $G' = (V', E')$ whose vertices are the SCCs of G, with an edge $C \to C'$ iff $\exists\, u \to v \in E$ with $\text{comp}(u) = C \neq C' = \text{comp}(v)$. This is $O(|V| + |E|)$.
3. *Run Student B Algo on G'.* Perform a DFS on G' to get its unique topological order $C_1, \dots, C_k$, then check that each direct edge (C_i, C_{i+1}) is present in E'. By part (a)(ii), this correctly decides the "comparability" property on the DAG of SCCs, in $O(|V'| + |E'|) = O(|V| + |E|)$.
4. *Answer for original G.* Since within each SCC all vertices are mutually reachable, the original graph G satisfies "for every $u \neq v$, $u \to^* v$ or $v \to^* u$" if and only if the condensed DAG G' is a total order. We already checked that in step 3.

Overall runtime. Each step—SCC computation, condensation, DFS with toposort, and $k-1$ edge-checks—runs in $O(|V| + |E|)$. Thus the algorithm is $O(|V| + |E|)$.

Overall correctness. By contracting each strongly-connected component into a single node, we guarantee that all vertices within a component are mutually reachable. The resulting condensed graph is a DAG whose vertices represent these components and whose edges represent inter-component reachability in the original graph. The original property ("for every two vertices u, v, either $u \to^* v$ or $v \to^* u$") holds exactly when, in this DAG, every pair of components is comparable by reachability. Checking that the DAG's components form a single chain—by verifying that each consecutive pair in its topological order has a direct edge—ensures

that for any two components C_i, C_j, either $C_i \to^* C_j$ or $C_j \to^* C_i$. Therefore, either two original vertices lie in the same component (so reach each other) or their components are ordered so one reaches the other. If any consecutive edge is missing, that pair of components (and thus some vertices) fails the reachability requirement. Hence the algorithm is correct.