Solutions to Homework 5 Practice Problems

CS6515 Summer 2026

[DPV] Problem 3.3 – Topological Ordering Example

Problem Statement

Run the DFS-based topological ordering algorithm on the following graph (see DPV). Whenever you have a choice of vertices to explore, always pick the one that is alphabetically first.

(a) Indicate the pre- and post-numbers of the nodes.

Node	A	B	C	D	E	F	G	H
`pre`	1	15	2	3	11	4	5	7
`post`	14	16	13	10	12	9	6	8

(b) What are the sources and sinks of the graph?

The graph has two sources (A and B) and two sinks (G and H)

(c) What topological ordering is found by the algorithm?

The topological ordering of the graph is found by reading the post-numbers in decreasing order: B, A, C, E, D, F, H, G.

(d) How many topological orderings does this graph have?

Any topological ordering of the graph will be of the form $[AB]C[DE]F[GH]$, with the ordering of the pairs in brackets arbitrary (for example, $ABCEDFHG$ is valid). Each bracketed pair can be organized in 2 different ways, so there are $2 * 2 * 2 = 8$ different topological orderings for this graph.

[DPV] Problem 3.4 – SCC Algorithm Example

Problem Statement

Run the strongly connected components algorithm on the following directed graphs G. When doing DFS on G^R: whenever there is a choice of vertices to explore, always pick the one that is alphabetically first.

(a) In what order are the strongly connected components (SCCs) found?

i. $\{C, D, F, J\}, \{G, H, I\}, \{A\}, \{E\}, \{B\}$

ii. $\{D, F, G, H, I\}, \{C\}, \{A, B, E\}$

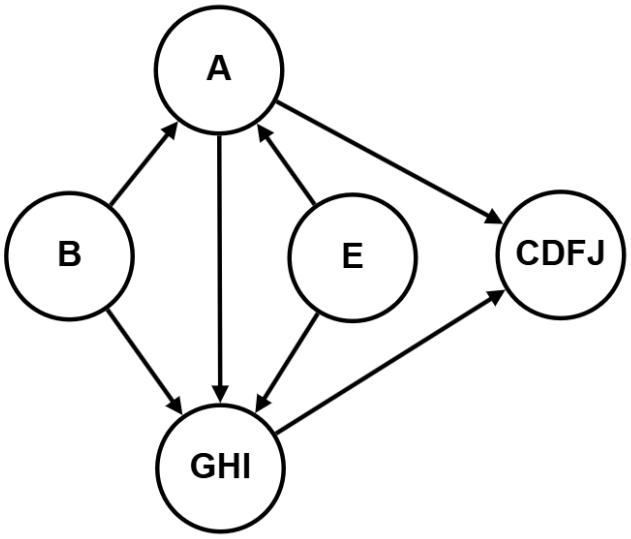
(b) Which are source SCCs and which are sink SCCs?

i. The source SCCs are $\{E\}$ and $\{B\}$. The sink SCC is $\{C, D, F, J\}$.

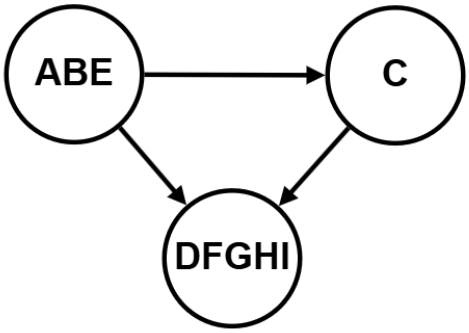
ii. The source SCC is $\{A, B, E\}$. The sink SCC is $\{D, F, G, H, I\}$.

(c) Draw the "metagraph" (each meta-node is an SCC of G).

i



ii



(d) What is the minimum number of edges you must add to this graph to make it strongly connected?

i. Two edges must be added to make the entire graph strongly connected: one from any vertex in $\{C, D, F, J\}$ to B, and one from any vertex in $\{C, D, F, J\}$ to E.

ii. One edge must be added to make the entire graph strongly connected: from any vertex in $\{D, F, G, H, I\}$ to any vertex in $\{A, B, E\}$.

[DPV] Problem 3.5 – Reverse of Graph

Problem Statement

The reverse of a directed graph $G = (V, E)$ is another directed graph $G^R = (V, E^R)$ on the same vertex set, but with all edges reversed; that is, $E^R = \{(v, u) : (u, v) \in E\}$.
Give a linear-time algorithm for computing the reverse of a graph in adjacency list format.

(a) Algorithm:

First, initialize an empty graph G^R (in adjacency list form) for n vertices where $n = |V|$. Then, for each $u \in V$, iterate through the neighbors of u. For each neighbor v of u in G, add u as a neighbor of v in G^R.

(b) Justification of Correctness:

The reverse of a graph is one where the vertices are the same, but the direction of the edges are flipped. By adding edge $\overrightarrow{(v, u)}$ for each edge $\overrightarrow{(u, v)}$, we accomplish the objective.

(c) Runtime Analysis:

Creating the new adjacency list takes $O(n)$ time. Traversing the original graph to reverse all m edges takes $O(n+m)$ time, with $O(1)$ to add each edge. Our overall run time is $O(n)+O(n+m) = O(n+m)$ time, which is linear in the input.

[DPV] Problem 3.15 – Computopia

Problem Statement

The police department in the city of Computopia has made all streets one-way. The mayor contends that there is still a way to drive legally from any intersection in the city to any other intersection, but the opposition is not convinced. A computer program is needed to determine whether the mayor is right. However, the city elections are coming up soon, and there is just enough time to run a *linear-time* algorithm.

Part (a):

Formulate this problem graph-theoretically, and explain why it can indeed be solved in linear time.

(a) Algorithm:

We will represent the city in this problem as a directed graph $G = (V, E)$, where vertices in V represent the intersections in the city and the directed edges in E represent the one-way streets of the city between intersections. The problem is then represented by determining whether a path from u to v exists for all $(u, v) \in V$ in linear time.

We can solve this problem by running the SCC algorithm and checking if there is a single SCC. We can do this by checking that the returned SCC metagraph has a single vertex (meaning the original graph is a single SCC). If that is the case, then we report the mayor's claim is TRUE. Otherwise, there is more than one SCC so we report that the mayor's clam is FALSE.

(b) Justification of Correctness:

This algorithm works because by modeling the original problem as a graph, we are able to efficiently solve the problem using the properties of an SCC, which is the presence of a path from every vertex to every other vertex in the same SCC. If the graph has a single SCC, then we know every intersection has a route to every other intersection. If there is more than one SCC, then there is at least one sink SCC and at least one source SCC, and the vertices (intersections) of the sink SCC(s) would not have a path to the vertices (intersections) of the source SCC(s).

(c) Runtime Analysis:

Creating the graph representation of the city takes $O(n + m)$ time to model the n intersections and m roads. Running the SCC algorithm takes $O(n + m)$. Checking for a single vertex in the SCC metagraph takes $O(1)$ constant time. The overall run time is $O(n + m)$, linear as required.

Part (b):

> Suppose it now turns out that the mayor's original claim is false. She next claims something weaker: if you start driving from town hall, navigating one-way streets, then no matter where you reach, there is always a way to drive legally back to the town hall. Formulate this weaker property as a graph-theoretic problem, and carefully show how it too can be checked in linear time.

(a) Algorithm:

The algorithm requires modeling the problem as a graph and computing its SCCs as in Part A (again, in $O(n+m)$ time). We instead check if the town hall is in a *sink SCC* - that is, ensuring there are no edges directed out of the SCC containing the town hall. We can do this by examining the SCC metagraph DAG to ensure the SCC containing town hall has no outgoing edges. The mayor's claim is TRUE if there are no outgoing edges from the town hall's SCC, else it is FALSE.

(b) Justification of Correctness:

The weaker claim requires that the town hall resides in a sink SCC. If it lies in a sink SCC S, then we know from the town hall we can reach every other intersection in S and from every other intersection in S we can get to the town hall. And, if S is not a sink SCC, then there are edges out of it, and therefore there are intersections that can be reached from the town hall but cannot get back to the town hall.

(c) Runtime Analysis:

As in part A, it takes linear $O(n+m)$ time to model the problem as a graph and run the SCC algorithm. Checking the SCC metagraph for zero outgoing edges from the SCC labeled with `ccnum[town hall]` takes $O(1)$. The total running time of this algorithm is $O(n+m)$, which is linear time.