Solutions to Homework Practice Problems

[DPV] 3.3 Topological Ordering Example
Run the DFS-based topological ordering algorithm on the following graph. 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.

Running DFS gives the following pre- and post-numbers:

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?

Th 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 $2 \cdot 2 \cdot 2 = 8$ different topological orderings for this graph.

[DPV] 3.4 SCC Algorithm Example

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)
The SCCs are found in the following order:

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

(ii)
The SCCs are found in the following order:

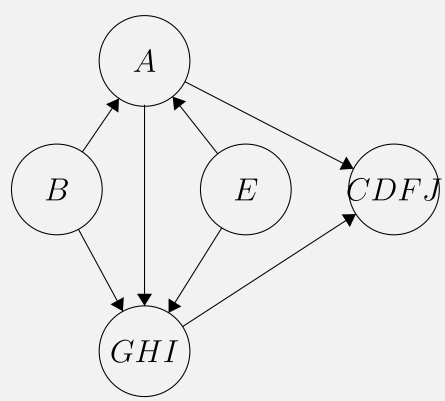
$$\{D, F, G, H, I\}, \{C\}, \{A, B, E\}$$

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

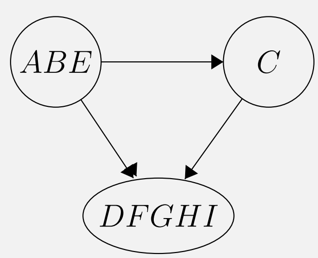
(i)
The source SCCs are $\{E\}$ and $\{B\}$. The sink SCC is $\{C, D, F, J\}$.
(ii)
The source SCC is $\{A, B, E\}$. The sink SCC is $\{D, F, G, H, I\}$.

(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] 3.5 Reverse of Graph
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.

First, initialize an empty adjacency list formatted graph on V (takes $O(|V|)$ time). Then, for each $u \in V$, go through the list of neighbors of u. For each neighbor v of u in G, add u as a neighbor of v in the new adjacency list for G^R. This will take $O(|V| + |E|)$ time to check each vertex and add each edge to the G^R.

[DPV] 3.8 Pouring Water

We have three containers whose sizes are 10 pints, 7 pints, and 4 pints, respectively. The 7-pint and 4-pint containers start out full of water, but the 10-pint container is initially empty. We are allowed one type of operation: pouring the contents of one container into another, stopping only when the source container is empty or the destination container is full. We want to know if there is a sequence of pourings that leaves exactly 2 pints in the 7- or 4-pint container.

(a) Model this as a graph problem: give a precise definition of the graph involved and state the specific question about this graph that needs to be answered.

We can model this problem as a graph where each vertex depicts a distribution of the water over the three containers. The vertices of the graph are triples (a_1, a_2, a_3) where a_i is the amount of liquid in the container with volume S_i, with $S_1 = 10, S_2 = 7$, and $S_3 = 4$. Then, we start at $(0, 7, 4)$, since the 10-pint container starts empty and the 7- and 4-pint containers start full. In order for vertices to be valid, they must represent a possible distribution of water over the containers. More specifically, each vertex (a_1, a_2, a_3) must satisfy:

$$\begin{gathered}0 \leq a_1 \leq S_1 \\ 0 \leq a_2 \leq S_2 \\ 0 \leq a_3 \leq S_3 \\ a_1 + a_2 + a_3 = 11\end{gathered}$$

The (directed) edges of the graph indicate possible state transitions (pouring water between containers). An edge from vertex (a_1, a_2, a_3) to vertex (b_1, b_2, b_3) exists if and only if:
1. The two vertices differ in exactly two coordinates, with the third coordinate the same in both.
2. Call the two different coordinates i and j. Either $b_i = 0$ or $b_j = 0$, or $b_i = S_i$ or $b_j = S_j$ (either one of the containers is now empty, or one of the containers is now full).
Then, the specific question we need to answer is whether there exists a path from vertex $(0, 7, 4)$ to a vertex of the form $(*, 2, *)$ or $(*, *, 2)$ (either the 7- or 4-pint container has exactly 2 pints).

(b) What algorithm should be applied to solve this problem?

In order to answer our question, run the `Explore` algorithm from vertex $(0, 7, 4)$, and check if we ever visit a vertex of the form $(*, 2, *)$ or $(*, *, 2)$. If `Explore` finishes without finding such a vertex, then there would be no sequence of pourings that leaves exactly 2 pints in the 7- or 4-pint container.

[DPV] Problem 3.15 Computopia

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.

We will represent the city in this problem as a directed graph $G = (V; E)$. The 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. Then, the problem is to determine whether a path from u to v exists for all $u, v \in V$, and to do so in linear time.

We can solve this problem using the SCC algorithm. If the entire graph G is itself a single strongly connected component, then the mayor's claim is true. Why? In a SCC, there is a path from every vertex to every other vertex in the same SCC. If the graph has a single SCC then every intersection has a route to every other intersection. The SCC algorithm takes linear time $O(n + m)$ for its two runs of DFS, 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.

The weaker claim requires that the town hall resides in a sink SCC. Why? If it lies in a sink SCC S then 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.

The algorithm requires computing the SCCs (again, in $O(n + m)$ time) and then checking if there are any edges out of the SCC containing the town hall. We can do this by examining either the DAG of the meta-graph of the SCC or each vertex which lies in the same SCC S as the town hall to see if they have any outgoing edges to vertices not in SCC S. This examination also takes $O(n + m)$ time, so the total running time of this algorithm is $O(n + m)$, which is linear time.