DPV Problem 3.8 (a) and (b).

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.

A theoretical idea for a graph is to model a graph with nodes where each node stores the value for the distribution of water in each node. A single node could then be represented by a format such as (a, b, c) where a is the volume in the 10 pints container, b is the volume in the 7 pints container and c is the volume in the 4 pints container. following this, an edge would be created between two nodes if it contains a possible transfer of volume.

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

A possible algorithm involves implementing a DFS search until a state is found which results in either $(a, b, 2)$ or $(a, 2, c)$.

DPV Problem 3.15.

The police department in the city of Computopia has made all streets one-way. The mayor con- tends 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.

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

The problem can be formulated in the form of a directed graph of $G = (V, E)$. Each intersection will be a vertex v with another vertex u where $v, u \in V$. There will be an edge (u, v) with $(u, v) \in E$ if there is a road going from u to v.
It can be considered legal to drive from one intersection to another if and only if the graph is strongly connected between the two. This can therefore be solved in linear time using DFS or BFS to determine if the whole graph is strongly connected.

(b) Suppose it now turns out that the mayors 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 same graph G can be used from (a) with the addition of a label t to represent the position of the town hall. The property specified in the question will be true if and only if there is never a situation where there is a path from t to a vertex v where $v \in V$ while there is no path from v to t.
There must be no path from the town hall to an vertex outside the connected components of t. This can be determined by numbering each of its connected components and then conducting either a BFS or a DFS starting from t. If the search reaches a component that has not been numbered, the claim will be false.

DPV Problem 3.22. The running time of your algorithm should be linear. *Hint* : **For all problems in the homework, but this one especially, try to use one of the algorithms we talked about in class as a subroutine instead of inventing your own.**

Give an efficient algorithm which takes as input a directed graph $G = (V, E)$, and determines whether or not there is a vertex $s \in V$ from which all other vertices are reachable.

A possible algorithm involves starting at a random node and conducting DFS in reverse to the specified directions to identify the node with the highest post number from the initial node (which will be a source node). Following this, conduct a second DFS (this one is a normal DFS) from this highest post node. Identify if this DFS reaches every node in the graph. It is true if it does reach reach every node and false otherwise.

DPV Problem 3.24. *Hint* : **For all problems in this homework, but this one especially, try to use one of the algorithms we talked about in class as a subroutine instead of inventing your own.**

Give a linear-time algorithm for the following task.
Input: A directed acyclic graph G
Question: Does G contain a directed path that touches every vertex exactly once?

This algorithm consists of two steps.
The first is to topologically sort the whole graph starting from any node.
The second step is to then go through this topologically sorted series and check if there is an edge from one node to the next in every pair within this series. If there is, the statement is true.

DPV Problem 3.25. The two-tiered structure in the problem refers to the fact that all directed graphs are DAGs of their strongly connected components.

You are given a directed graph in which each node $u \in V$ has an associated price p_u which is a positive integer. Define the array cost as follows: for each $u \in V$,

cost[u] = price of the cheapest node reachable from u (including u itself).

For instance, in the graph below (with prices shown for each vertex), the cost values of the nodes A, B, C, D, E, F are $2, 1, 4, 1, 4, 5$, respectively.

Your goal is to design an algorithm that fills in the *entire* cost array (i.e., for all vertices).

(a) Give a linear-time algorithm that works for directed acyclic graphs. ($Hint$: Handle the vertices in a particular *order*.)

A possible algorithm involves using two steps, first, topologically sort the graph. Following this, find the $cost[u]$ of each node in reverse topological order.

(b) Extend this to a linear-time algorithm that works for all directed graphs. ($Hint$: Recall the two-tiered structure of directed graphs.)

For this algorithm, first take the DAGs of the strongly connected components of the graph as the remaining part of the algorithm will be run on it. For a random strongly connected node, find the $price_C = min(price_u)$ for all $u \in C$. $price_C$ will represent the costs for all u in C. after this the nodes will be topologically sorted from highest, to lowest post numbers.