Solutions to Homework Practice Problems

Problem 1: Dijkstra's Algorithm

This is not covered in the lectures because it's the sort of thing many of you have seen before, possibly multiple times (GT undergrads see it at least 3 times in their algorithms, data structures, and combinatorics class). If you haven't seen it, or need a refresher, look at Chapter 4.4 of [DPV].

(a) What is the input and output for Dijkstra's algorithm?

The inputs for Dijkstra's algorithm are a graph $G = (V, E)$ with positive weights l_e for each edge $e \in E$, along with a source vertex s. (The weights must be positive in order for the algorithm to work.)
The outputs of Dijkstra's algorithm are the shortest paths from the source vertex to all other vertices of the graph. Specifically, Dijkstra's algorithm will output a `dist` array (containing the shortest distance from the source to each vertex) and a `prev` array (indicating the previous vertex that the shortest path uses to get to each vertex). The `prev` array can be used to construct the shortest paths.
(As a historical note, Dijkstra's original formulation of the algorithm found the shortest path between two input nodes, but the algorithm can easily find the shortest path to all vertices from a source by continuing the search until no more outgoing edges exist, instead of stopping after the target vertex is found. By now, Dijkstra's algorithm usually refers to the source-to-all variant, since it also has the same runtime.)

(b) What is the running time of Dijkstra's algorithm using min-heap (aka priority queue) data structure? (Don't worry about the Fibonacci heap implementation or running time using it.)

The running time of Dijkstra's algorithm using a min-heap is $O((|V| + |E|) \log |V|)$. This comes from the binary heap requiring $O(\log |V|)$ operations to either insert a key or remove the minimum key. Dijkstra's requires $O(|V|)$ key removals and $O(|V| + |E|)$ key insertions, yielding the $O((|V| + |E|) \log |V|)$ runtime.

(c) What is the main idea for Dijkstra's algorithm?

The main idea of Dijkstra's algorithm is to find the distances to the nearest nodes, then gradually expand the frontier of the search and determine the shortest paths to nodes that are further and further away. Since edge weights must be positive, there will never be a case where the shortest path must first go through a far away vertex (the path would already be longer than another one that had been found).

Problem 2: 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.

Problem 3: 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:

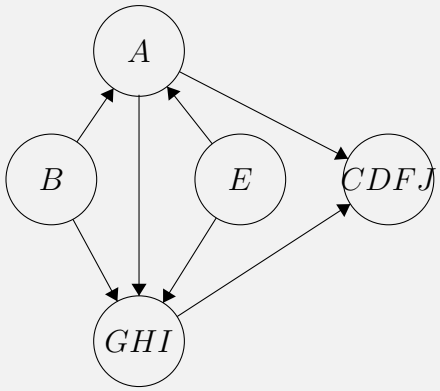
$$\{D, F, G, H, I\}, \{C\}, \{A, B, E\}$$

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

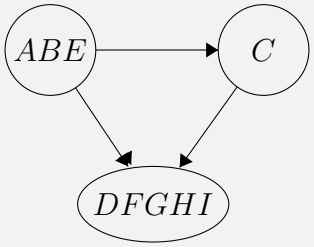
(i)
The source SCCs are $\{E\}$ and $\{B\}$. The sink SCC is $\{C, D, F, J\}$.
(ii)
The source SCC is $\{A, B, E\}$. The sink SCC is $\{D, F, G, H, I\}$.

(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 $\{A, B, E\}$.

Problem 4: 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.

Problem 5: 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:

$$0 \leq a_1 \leq S_1$$
$$0 \leq a_2 \leq S_2$$
$$0 \leq a_3 \leq S_3$$
$$a_1 + a_2 + a_3 = 11$$

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.

Problem 6: DPV 4.13 Cities and Highways

You are given a set of cities, along with the pattern of highways between them, in the form of an undirected graph $G = (V, E)$. Each stretch of highway $e \in E$ connects two of the cities, and you know its length in miles, l_e. You want to get from city s to city t. There's one problem: your car can only hold enough gas to cover L miles. There are gas stations in each city, but not between cities. Therefore, you can only take a route if every one of its edges has length $l_e \leq L$.

(a) Given the limitation on your car's fuel tank capacity, show how to determine in linear time whether there is a feasible route from s to t.

Consider a new graph G' obtained as follows: the set of vertices is V, the same as G. For edges in G' include only those edges of G such that $l_e \leq L$. It takes $O(n+m)$ to build this graph as we need to go through all edges in G. Now, run DFS on G' starting at s and using any prescribed order. If t is reachable from s in this new graph, then there is a feasible route between the two cities, since we deleted all edges bigger than L. Since DFS runs in $O(n+m)$ linear time, this algorithms has the desired running time.

(b) You are now planning to buy a new car, and you want to know the minimum fuel tank capacity that is needed to travel from s to t. Give an $O((|V|+|E|)\log|V|)$ algorithm to determine this.

To solve this problem we will modify the measure of distance used by Dijkstra's algorithm. For any pair of vertices u and v define

$$d(u,v) = \min_{P}\{\max_{e\in P} l_e\}$$

where P is the set of all paths from u to v; that is, for each possible path between u and v, find the maximum weight edge e on that path and then minimize (select) the path with the minimum of these maximum weight edges.

Notice that the last loop on Dijkstra's algorithm needs to change to capture this new distance. Instead of the sum $dist(u) + l(u,v)$ we do

$$\text{If } dist(v) > \max\{dist(u), l(u,v)\}$$

to check if the distance needs to be updated, if so we do

$$dist(v) = \max\{dist(u), l(u,v)\}$$

The lines $prev(v) = u$ and $Decreasekey(H, v)$ remain the same. Note that the $O((|V|+|E|)\log|V|)$ running time of Dijkstra's does not change with this new distance measurement. With this definition of distance, it follows that the minimum tank size is given by $dist(t)$.