Practice problems (don't turn in):

1. Dijkstra's algorithm: We will assume you know the answer to the following questions:
   (a) What is the input and output for Dijkstra's algorithm?
   (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.)
   (c) What is the main idea for Dijkstra's algorithm?
2. [DPV] Problem 3.3 (Topological ordering example)
3. [DPV] Problem 3.4 (SCC algorithm example)
4. [DPV] Problem 3.5 (Reverse of graph)
5. [DPV] Problem 3.8 (Pouring water), if your book has a part (c) you can skip it... or not... this is just practice!
6. [the interval $(prev, post)$ in DFS] After running DFS on a directed graph $\overrightarrow{G}$ you notice that the edge $\overrightarrow{uv}$ is a *cross edge*. Show that the intervals $(prev(u), post(u))$ and $(prev(v), post(v))$ are disjoint.
7. [more on $(prev, post)$] Consider two vertices u and v with the following property: after running DFS you got the inclusion: $(prev(u), post(u)) \subset (prev(v), post(v))$. Are u and v strongly connected? What can you say about the connectivity of these two vertices?
1. DPV Problem 4.1 (running Dijkstra's alg.)
2. DPV Problem 4.2 (running Bellman-Ford)
3. DPV Problem 4.8 (Professor Lake)
4. DPV Problem 4.11 (shortest cycle)
5. DPV Problems 5.1-5.3
6. DPV Problem 5.21 (heaviest edge on a cycle)
7. DPV Problem 5.25 (set of *equality and disequality* constraints)

Instructions:

In this class, assume we're always using min-heap data structure for Dijkstra's. Report runtime accordingly.

In the algorithm design problems: use the algorithms from class, such as DFS, BFS, Dijkstra's, SCC (connected components), etc., as a black-box subroutine for your algorithm. So say what you are giving as input, then what algorithm you are running, and what's the output you're taking from it. You can use Explore (subroutine of DFS) as one of the black-box algorithms.

Here's an example:
`I take the input graph` G`, I first find the vertex with largest degree, call it` v^*`. I take the complement of the graph` G`, call it` $\overline{G}$`. Run Dijkstra's algorithm on` $\overline{G}$ `with` $s = v^*$ `and then I get the array` $dist[v]$ `of the shortest path lengths from` s `to every other vertex in the graph` $\overline{G}$`. I square each of these distances and return this new array.`

We don't want you to go into the details of these algorithms and tinker with it, just use it as a black-box as showed with Dijkstra's algorithm above.

Make sure to explain your algorithm in words, no pseudocode. Explain why it works and stay and justify its running time.

Problem 1 [DPV] Problem 3.11 (edge on a cycle)

Design a linear time algorithm which takes as input an undirected graph G and a particular edge e in it, and determines whether G has a cycle containing e.

Problem 2 [DPV] Problem 3.15 (Computopia)

- Assume that the Town Hall sits at 1 single intersection.
- The mayor's claim in (b) is the following: If you can get to another intersection, call it u, from the Town Hall intersection, then you can get back to the Town Hall intersection from u.
- Note, linear time means $O(n + m)$ where $n = |V|$ and $m = |E|$.

Part (a):

Part (b):

Problem 3 [DPV] Problem 4.20 (building a new road)

There is a network of roads $G = (V, E)$ connecting a set of cities V. Each road has an associated length ℓ_e...

Problem 4 Edge on MST

You are given a weighted graph $G = (V, E)$ with positive weights, c_i for all $i \in E$. Give a linear time ($O(|E| + |V|)$) algorithm to decide if an input edge $e = (u, v) \in E$ with weight c_e is part of some MST of G or not.

You should describe your algorithm in words (a list is okay); no pseudocode.