Practice problems (don't turn in):

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. If you haven't seen it, or need a refresher, look at Chapter 4.4 of [DPV]. 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. [DPV] Problem 7.10 (max-flow = min-cut example)
7. [DPV] Problem 7.17 (bottleneck edges).
8. [DPV] Problem 7.19 (verifying max-flow)
9. For a bipartite graph $G = (V_1 \cup V_2, E)$ where $|V_1| = |V_2| = n$ a *perfect matching* is a subset S of edges where each vertex is incident exactly 1 edge in S. In other words, it's a matching of size n. Given a bipartite graph G show how to determine if G has a perfect matching by a reduction to the max-flow problem. So given G define an input to the max-flow problem, and given a max-flow for this input how do you determine if the original graph G has a perfect matching or not? What is the running time of your algorithm?
   (For hints see [DPV] Chapter 7.3 (Bipartite matching) and the beginning of Problem 7.24.)

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, 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. 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.

You can use Explore (subroutine of DFS) as one of the black-box algorithms.

Problem 1 [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 2 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.