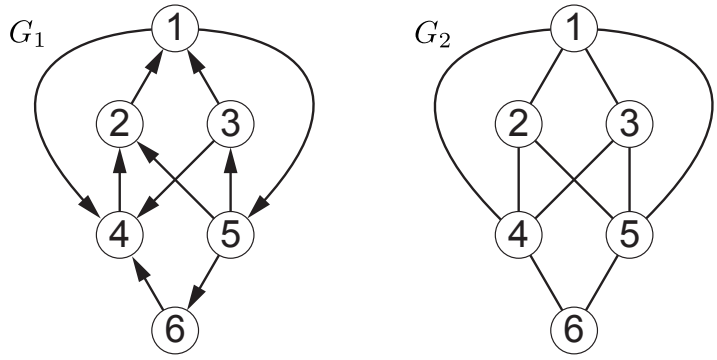
Problem Set 7

All parts are due on April 12, 2018 at 11PM. Please write your solutions in the LaTeX and Python templates provided. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct. Solutions should be submitted on the course website, and any code should be submitted for automated checking on `py.mit.edu/6.006`.

Problem 7-1. [30 points] **Graph Practice II**

(a) [20 points] For graphs drawn and embedded in the plane, it is common to fix an order to search outgoing edges: for example, counterclockwise order from the edge from which you entered a vertex. We will refer to a depth-first search using such an ordering as a CCW DFS. For direct graph G_1 and undirected graph G_2 **as drawn** below, perform a depth-first search from vertex 1 using CCW DFS, starting the search along the edge from vertex 1 to vertex 4. Then for each graph, (i) label each edge as either a **tree**, **back**, **forward**, or **cross** edge, and (ii) list the graph vertices in a topological sort based on the finishing times of your depth-first search, or argue that a topological sort does not exist.



(b) [5 points] Prove that for any connected undirected graph G on n vertices and any positive integer $k < n$, either G has some path of length at least k, or G has fewer than $k \cdot n$ edges.

(c) [5 points] An *articulation point* of a connected undirected graph is any vertex that, if removed from the graph would make the graph no longer connected. For example, if only one path of vertices connects vertices u and v in a graph, then every vertex in the path between u and v would be an articulation node. Given an undirected graph, describe a $O(|V||E|)$ time algorithm to output a list of articulation points in a graph. **Challenge:** listing all articulation points of an undirected graph can be accomplished in $O(|V| + |E|)$ time.

Problem 7-2. [20 points] **Buzzword Bearing**

As an MIT student you naturally want to take 19.006, Introduction to Quantum Machine Learning on the Blockchain with Space Applications. Assume you adhere strictly to prerequisites, taking no classes before having satisfied the relevant prerequisite. You've scraped from the registrar, a list of all classes taught at MIT and all their prerequisite classes. For this problem, you may assume that every class is taught every semester at MIT.

(a) [10 points] Given a list of classes you've already taken, describe a linear time algorithm to determine the earliest semester you can take 19.006, assuming you can take as many classes as you want in any semester.

(b) [10 points] Unfortunately, being able to take as many classes as you want isn't realistic, and your adviser is upset at you for proposing such a schedule. You've been given a credit limit of only 1 class per semester. Since you don't want to pay tuition indefinitely, design an algorithm, that returns a valid schedule for every semester that minimizes the number of semesters you're here.

Problem 7-3. [20 points] **Louis' Mansion**

One day, a vacuum salesman named Louis inherited his grandparent's mansion, which was said to be haunted. When he arrives to the mansion, trickster ghosts swooped through the hallways and opened up every single door! Louis needs to lock all the doors of the mansion as soon as possible to ensure that Browser, a large spider monster who lurks within the mansion cannot escape.

Every hallway connects exactly two rooms. Each room has an initially-open door to all of its hallways, and each door can either be open, closed, or locked. Doors can be locked from either side, but once a door is locked, even Louis can't open it. Louis has a very poor memory, and has no map of the mansion, but is very good a following instructions. Write a set of instructions for Louis that ensures that he locks every door in the mansion, starting and ending at the front door (and as always, be sure to analyze correctness and running time).

Problem 7-4. [40 points] **Solving Sudoku**

A **Sudoku** puzzle consists of a 9 x 9 grid, with each grid square possibly a numerical digit from 1 to 9 inclusive. In an **unsolved** puzzle, some cells of the grid are pre-filled with digits, but many do not, as shown in the example below. The goal of the puzzle is to fill in each un-filled cell with a digit between 1 and 9 inclusive, such each row, each column, and each 3×3 sub-grid (with highlighted boundaries), contains the digits 1 through 9 exactly once. In this problem, we will develop an algorithm to return a fully filled, **solved** Sudoku given an unsolved Sudoku as an input.

	2		5		1		9	
8			2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

Input (Unsolved Sudoku)

4	2	**6**	5	**7**	1	**3**	9	**8**
8	**5**	**7**	2	**9**	3	**1**	**4**	6
1	3	**9**	**4**	6	**8**	**2**	7	**5**
9	**7**	1	**3**	**8**	**5**	6	**2**	**4**
5	4	**3**	**7**	**2**	**6**	**8**	1	9
6	**8**	2	**1**	**4**	**9**	7	**5**	**3**
7	9	**4**	**6**	3	**2**	**5**	8	**1**
2	**6**	**5**	8	**1**	4	**9**	**3**	7
3	1	**8**	9	**5**	7	**4**	6	**2**

Output (Solved Sudoku)

(a) [5 points] **Implicit Graph:** Given a Sudoku configuration C with some filled cells, let the neighbors of C be the configurations which fill the first unfilled cell with a valid digit. A valid digit is one that does not cause any of the conditions to be violated. We will define the *first unfilled cell* to be the first empty cell when scanning each row left to right starting from the top. What is the minimum and maximum number of neighbors of any Sudoku puzzle configuration? Describe how to compute all neighbors of a given configuration.

(b) [10 points] **Sudoku Solve:** Given an input unsolved Sudoku puzzle, describe an algorithm to output a solved Sudoku, using the implicit graph defined in part (a). If there are multiple solutions, your algorithm should return the solution which leads to the smallest number when concatenating all the digits in the initially empty cells, in order from the top row to the bottom row. Your algorithm should use only $O(d)$ space, where d is the number of unfilled cells in the initial Sudoku. (**Hint:** can the graph from (a) contain cycles?)

(c) [25 points] **Implement:** Write the Python function `solve_sudoku(config)` that implements your puzzle solving algorithm. An input configuration will be a list of lists, each of length 9, corresponding to each row of the Sudoku board. Each element in a row will be an integer from 0 to 9 inclusive, where 0 corresponds to an empty cell, e.g., the first row of the unsolved Sudoku shown above would be represented as `[0,2,0,5,0,1,0,9,0]`. Your function should return a solved Sudoku using the algorithm described in part (b). The output should be in the same format as the input, but containing no 0 entries. If no valid solution exits, your function should return `None`. Submit your code online at `py.mit.edu/6.006`.