Problem Set 5

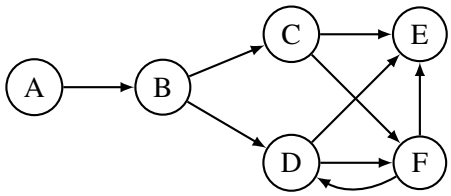
All parts are due on April 5, 2020 at 6PM. Please write your solutions in the LaTeX and Python templates provided. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct. Solutions should be submitted on the course website, and any code should be submitted for automated checking on `alg.mit.edu`.

Problem 5-1. [15 points] **Graph Practice**

(a) [3 points] Draw the undirected graph described on the right by **adjacency matrix** `Adj`: a direct access array Set mapping each vertex $u \in \{0, \ldots, 5\}$ to an adjacency list `Adj[u]`, where each adjacency list is also implemented using a direct access array Set such that `Adj[u][v]` $= 1$ if and only if vertices `u` and `v` are connected by an edge.

```
#        0  1  2  3  4  5
Adj = [[0, 0, 1, 0, 0, 0], # 0
       [0, 0, 0, 1, 1, 1], # 1
       [1, 0, 0, 1, 1, 0], # 2
       [0, 1, 1, 0, 0, 0], # 3
       [0, 1, 1, 0, 0, 1], # 4
       [0, 1, 0, 0, 1, 0]] # 5
```

(b) [3 points] Write down the adjacency list representation of the graph on the right: where a Python Dictionary maps each vertex `v` to its adjacency list `Adj[v]`, and each adjacency list is a Python List containing `v`'s outgoing neighbors **listed in alphabetical order**.



(c) [6 points] Run both BFS and DFS on the graph from part (b), starting from node A. While performing each search, visit the outgoing neighbors of a vertex in alphabetical order. For each search, draw the resulting tree and list vertices in the order in which they were first visited.

(d) [3 points] It is possible to remove a single edge from the graph in (b) to make it a DAG. State every edge with this property, and for each, state a topological sort order of the resulting DAG.

Problem 5-2. [10 points] **Power Plants**

The Boston electrical network contains n power plants and n^2 buildings, pairs of which may be directly connected to each other via bidirectional wires. Every building is powered by either: having a wire directly to a power plant, or having a wire to another building that is recursively powered. Note that the network has the property that no building could be powered by more than one power plant (or else accounting would be ambiguous). Boston has secured funds to install an emergency generator in one of the power plants, which would provide backup power to all buildings powered by it, were the power plant to fail. Given a list W of all the wires in the network, describe an $O(n^4)$-time algorithm to determine the power plant where Boston should install the generator that would provide backup power to the most buildings upon plant failure.

Problem 5-3. [10 points] **Short-Circuitry**

Star-crossed androids Romie-0 and Julie-8 are planning an engagement party and want to invite all their robotic friends. Unfortunately, many pairs of their friends can't be in the same room without short-circuiting. Romie-0 has an idea to throw *two* parties instead: inviting some friends to the first party, and the rest to the second. Ideally, everyone would get invited to a party, but nobody would go to the same party as someone that would make them short-circuit. Julie-8 points out to Romie-0 that this might not be possible, for example if they have three friends who all make each other short-circuit. Given the n short-circuiting pairs of friends, describe an $O(n)$-time algorithm to determine whether Romie-0 and Julie-8 can invite all of their friends to two peaceful parties, without anyone short-circuiting.

Problem 5-4. [10 points] **Ancient Avenue**

The ancient kingdom of Pesomotamia was a fertile land near the Euphris and Tigrates rivers. Newly-discovered clay tablets show a map of the kingdom on an $n \times n$ square grid, with each square either:

- part of a river, labeled as `'euphris'` or `'tigrates'`; or
- not part of a river, labeled with a string: the name of the farmer who owned the square plot of land.

The tablets accompanying the map state that ancient merchants built a **trade route** connecting the two rivers: a path along edges of the grid from some grid intersection adjacent to a `'euphris'` square to a grid intersection adjacent to a `'tigrates'` square. The tablets also state that the route:

- did not use any edge between two squares owned by the same farmer (so as not to trample crops); and
- was the shortest such path between the rivers (assume a shortest such path is unique).

Given the labeled map, describe an $O(n^2)$-time algorithm to determine path of the ancient trade route.

Problem 5-5. [15 points] **Statum Quest**

Liza Pover is a hero on a quest for free pizza in the Statum Center, a large labyrinth with some of the best free food in the entire Technological Institute of Mathachusetts. Many unsuspecting victims[1] have ventured into the labyrinth in search of free food, but few have escaped victorious. Luckily, Liza has downloaded a map from `⌊plans⌋.tim.edu` consisting of **locations** L (including rooms, hallways, staircases, and elevators[2]) and **doors** D which connect pairs of locations. One location $e \in L$ is marked as the **entrance/exit**: Liza's path must begin and end here. A subset $\Pi \subset L$ of locations each contain a free pizza. Liza's goal is to enter the maze, grab one pizza, and leave as quickly as possible.

Each door $d = (\ell_1, \ell_2)$ connects two locations ℓ_1 and ℓ_2 and may be **one-way** (can be traversed only from ℓ_1 to ℓ_2) or **two-way** (can be traversed in either direction). In addition, each door d may require **card access** to one of the four Statum Center labs — See-Sail, TOPS, S3C[3], and DSW[4]. A card-access door of type $t \in \{\text{SeeSail}, \text{TOPS}, \text{S3C}, \text{DPW}\}$ can only be traversed after Liza acquires the matching key card, where the key card of type t is in a known location ℓ_t.

Describe an $O(|L| + |D|)$-time algorithm to find a viable path from e to e that collects at least one pizza from Π (and possibly some of the key cards along the way to open some doors), and minimizes the number of times that Liza has to walk through a door.

[1] Formerly known as freshmen

[2] We model an entire elevator column as one location, with a door to each floor location it can access

[3] Super Secret Spider Consortium

[4] Department of Speechlore and Wisdomlove

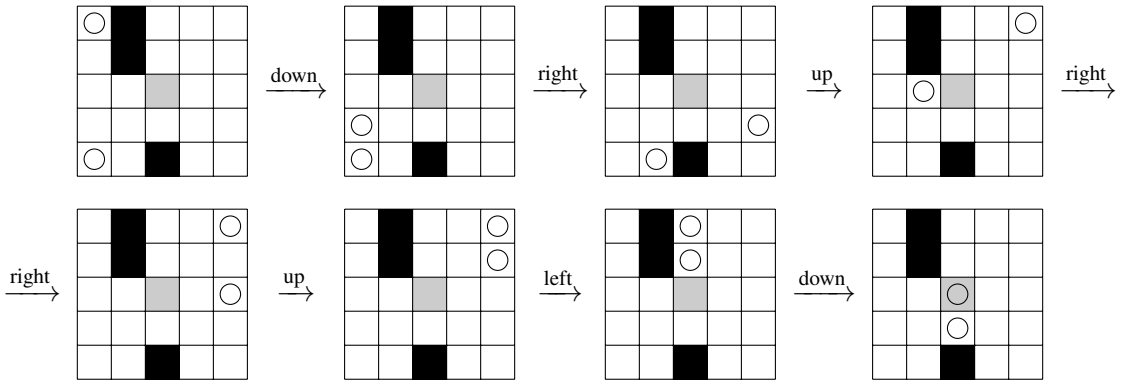
Problem 5-6. [40 points] **6.006 Tilt**

6.006 Tilt[5] is a puzzle game played on a Tilt **board**: an $n \times n$ square grid, where grid square contains either a fixed **obstacle**, a movable **slider**, or neither (the grid square is **empty**).

A Tilt **move** consists of tilting a board in any of the four cardinal directions (up, left, down or right), causing each slider to slide maximally in that direction until it hits either: the edge of the board, an obstacle, or another slider that cannot slide any further. A Tilt move results in a new Tilt board **configuration**.

Given a Tilt board B with one grid square t labeled as the **target**, a sequence of k Tilt moves **solves** 6.006 Tilt **puzzle** (B, t) if applying the moves in sequence to B results in a Tilt board configuration B' that contains a slider in square t.

The figure below shows a small 6.006 Tilt puzzle and a solution using the fewest possible moves $k = 8$. Obstacles are shown in black, movable sliders are circles, and the target square is shaded gray.



We represent an $n \times n$ board configuration `B` using a length-n tuple of length-n tuples, each representing a row of the configuration, where the grid square in row y (from the top) and column x (from the left) is `B[y][x]`, equal to either character `'#'` (an obstacle), `'o'` (a slider), or `'.'` (neither). We represent the target square by a tuple `t = (xt, yt)` where the puzzle is solved when `B[yt][xt] = 'o'`. Your code template has a function `move(B, d)` which computes a move from board `B` in direction `d` in $O(n^2)$ time.

(a) [2 points] Given a Tilt puzzle with starting $n \times n$ board B containing b fixed obstacles and s movable sliders, argue that the number of board configurations reachable from B is at most $C(n, b, s) = \frac{1}{s!} \prod_{i=0}^{s-1} (n^2 - b - i)$.

(b) [3 points] If Tilt board configuration B can reach another Tilt board configuration B' by a single move, we call B' a successor of B, and B a predecessor of B'. Argue that a board configuration B may have $\Omega(n^s)$ predecessors, but has at most $r = O(1)$ successors.

(c) [10 points] Given a 6.006 Tilt puzzle (B, t), describe an algorithm to return a move sequence that solves the puzzle in the fewest moves, or return that no such move sequence exists. If the puzzle's shortest solution uses k moves ($k = \infty$ if the puzzle is not solvable), your algorithm should run in $O(n^2 \min\{r^k, C(n, b, s)\})$ time.

(d) [25 points] Write a Python function `solve_tilt(B, t)` that implements your algorithm from part (d) using the template code provided. You can download the code template and some test cases from the website. Submit your code online at `alg.mit.edu`.

[5] This is a simplification of the actual game ***Tilt***, shown here: http://www.youtube.com/watch?v=mFGevho4OLY