SOLID understanding of material! :)

88/100 mostly due to not writing out explicit graph representation steps & analysis

Ravi Dayabhai

Introduction to Algorithms: 6.006
Massachusetts Institute of Technology
Instructors: Erik Demaine, Jason Ku, and Justin Solomon

Problem Set 5

Problem Set 5

Please write your solutions in the LaTeX and Python tem-plates provided. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct.

12/15

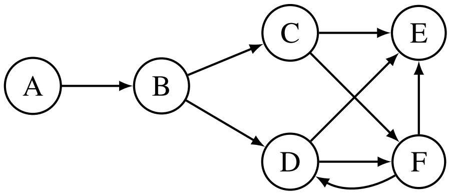
silly mistakes!

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.

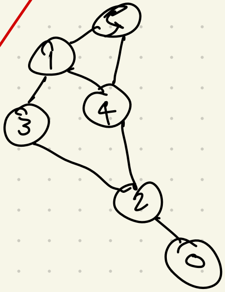
8/10

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.

(SEE NEXT 2 PAGES)

① [A] ✓



[B]

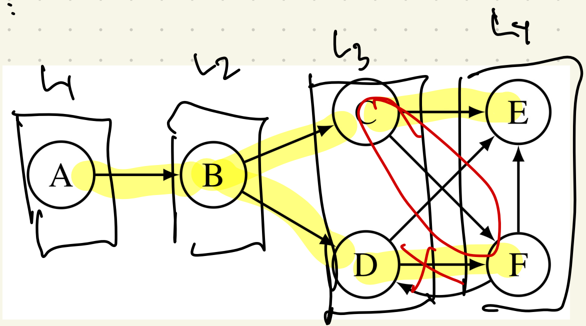
```
{"A": ["B"],
 "B": ["C", "D"],
 "C": ["E", "F"],
 "D": ["E", "F"],
 "E": [],
 "F": ["D"]}
```

"E"

-1

[C] BFS:

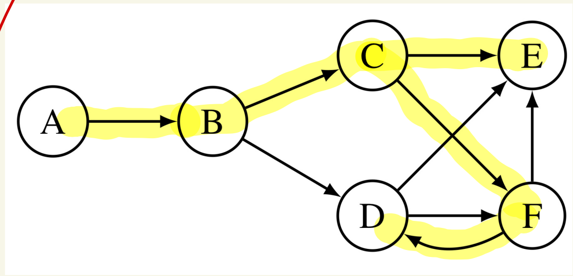
-1



A, B, C, D, E, F is ordering of this BFS (i.e., order in which they're added and processed by queue.). ✓

(HIGHLIGTED is the search tree.)

DFS: ✓

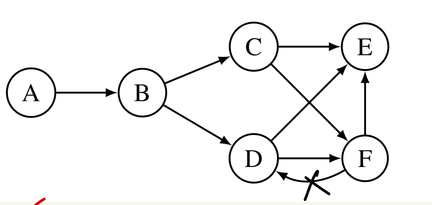


If ordering on first visited (i.e., order in which they were added to `visited` set):

A, B, C, E, F, D ✓

(① CONT.)

$\boxed{b}$



Removing (F, D) edge makes this graph a DAG. A top-sort with this edge removed:

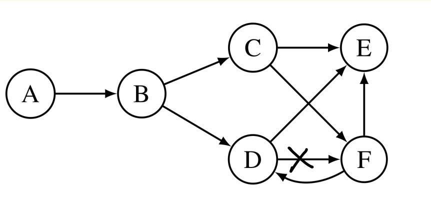
(A, B, C, D, E, F).

-1

(D, F) can also be the single edge that's removed.

⇒ Top sort:

(A, B, C, F, D, E)



② $|W|$ is upper-bounded by $\binom{n+n^2}{2} = \binom{n(n+1)}{2} =$

$$\frac{(n(n+1))((n(n+1))-1)}{2} = O(n^4) = |E|.$$

Running DFS from each of n power-plants can give the size of the connected component associatid with that powerplant. Since $|E| = O(n^4)$ and $|V| = O(n^2)$ ⇒ $O(|V|+|E|) = O(n^4)$, satisfying the runtime constraint.

-2 for not considering the graph construction in $O(n^4)$

(SEE NEXT 3 PAGES)

8/10

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.

7/10

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.

10/15

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 $\lfloor\texttt{plans}\rfloor\texttt{.tim.edu}$ 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

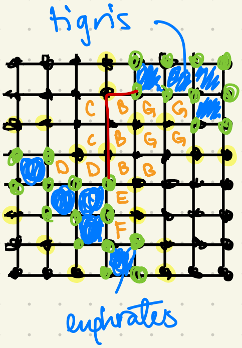
(3) This problem reduces to determining if a graph is two colorable. The graph here is determined by the n edges / pairs of robots that short-circuit. Let V be the set of unique robots that short-circuit with at least one other robot. Start with arbitrary vertex in V and color it red. Follow any edge incident to this vertex and color it green. From this newly colored green vertex, repeat the process: pick an incident edge that's yet to be processed; color vertex on the other end red if this vertex is uncolored, otherwise pick another unprocessed incident edge and proceed when the other vertex is green/opposite color; if other vertex is red stop — an odd-length cycle exists so two parties is not a feasible solution. (If no more edges in a connected component (and two-coloring invariant is maintained), continue with an unprocessed $v \in V$.) Continue until graph is two-colored OR early termination due to invariant violation.

This runs in $O(n)$ time because there are $O(n)$ vertices processed and $O(n)$ edges traversed.

(-2) didn't mention graph construction & its time complexity

-3

again, graph construc-tion



④ We can use BFS to find a single-source shortest path. WLOG, create a super node T that connects to all vertices that are adjacent to the Tigris. In $O(n^2)$ time we can run BFS from T since checking valid edges for traversel is $O(1)$ operations (given grid graph). Search terminates as soon as a Euphrates vertex is reached. Now to recover path (assuming we've been keeping track of parent pointers), we just need to connect this shortest path through T to a Tigris vertex, which can be done by checking the neighbors of the vertex before T in this shortest path, one of which is guaranteed to be a Tigris vertex.

⑤ A "sequence" of key locations:

- starts and ends with e, any ordering in between
- includes at least one location with a pizza π
- can include some permutation of any combination of $\ell_{t_1}, \ell_{t_2}, \ell_{t_3}, \ell_{t_4}$

-5 no discussion of graph construction

e.g., $e \rightsquigarrow \pi \rightsquigarrow e$

$e \rightsquigarrow \ell_{t_2} \rightsquigarrow \pi \rightsquigarrow \ell_{t_1} \rightsquigarrow e$

$e \rightsquigarrow \ell_{t_4} \rightsquigarrow \ell_{t_3} \rightarrow \ell_{t_1} \rightsquigarrow \pi \rightsquigarrow e$

There are $\leq (5!)(2^4)$ such sequences with the longest sequence consisting of at most 7 elements $\Rightarrow$ $O(|L|+|D|)$ time if each consecutive pair in a sequence represents a BFS from the former to the latter in the pair.

(⑤ cont.)

Removing this ensures exhaustive search on all sequences.

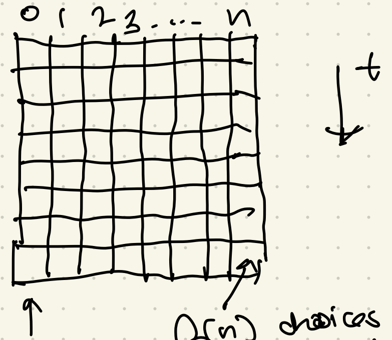
So, we can run BFS for each $\overset{BFS}{\rightsquigarrow}$ for each sequence. The digraph on which a $\overset{BFS}{\rightsquigarrow}$ proceeds depends on which ℓ_t precede it (we effectively add door directed edges according to what key cards have been already acquired in the sequence). ~~For $\overset{BFS}{\rightsquigarrow} \pi$, we can stop as soon as any $\pi \in \Pi$ is reached and then search $\pi^* \rightsquigarrow e$ where π^* is the first pizza found.~~

⑥ $\boxed{A}$ We are trying to place s sliders on an n×n board that has b obstacles. ⟹ $\binom{n^2-b}{s}$ counts the ways we can place s sliders on the board ⟺

$$\frac{\prod_{i=0}^{n^2-b-1} n^2-b-i}{s!\,(n^2-b-s)!} = \frac{1}{s!}\prod_{i=0}^{s-1} n^2-b-i .$$

$\boxed{B}$ A board B has O(1) successors because at most there are 4 unique B' (one corresponding to each tilt direction) since all sliders move maximally in the tilt direction. To see why B may have $\Omega(n^s)$ predecessors consider, WLOG, a tilt direction t. When board B is tilted in direction t, all s sliders will move maximally toward the bottom row (given by choice of t). However, we can place slider s_0 in rank_0 in any of $\Omega(n)$ squares, slider s_1 in rank_1 in any of $\Omega(n)$ squares, etc.

(⑧ CONT.)



$\Omega(n)$ choices to place $s_0, s_n, s_{2n}, \ldots$ to construct predecessor to B where B has sliders in these ranks.

$\Omega(n)$ choices to place slider here as well to construct predecessor to B

If $s > n$, we still have $n-1 = \Omega(n)$ choices for slider s_n in rank_0, $n-1$ choices for s_{n+1} in rank_1, etc. The same holds for any predecessor of B where B has multiple sliders on a given rank when $s \leq n$. (This is most easily seen for a board with no obstacles and B having sliders stacked along bottom row.)

⑧ $\boxed{D}$. We can model the game state using the representation described in the problem. A transition from $B \rightarrow B'$ takes $O(n^2)$ to update the n^2 elements. Since each B has r successors, we can get all possible B' in $O(1) \times O(n^2)$ time. Performing BFS until a state is found with a slider in target position will take r^k time. (If it's impossible to find such a state, the entire state space will be exhausted: $C(n,b,s)$ vertices will have been visited.) Thus, the overall runtime is $O(n^2)$ for each edge/transition times $\min(r^k, C(n,b,s))$.

40/40

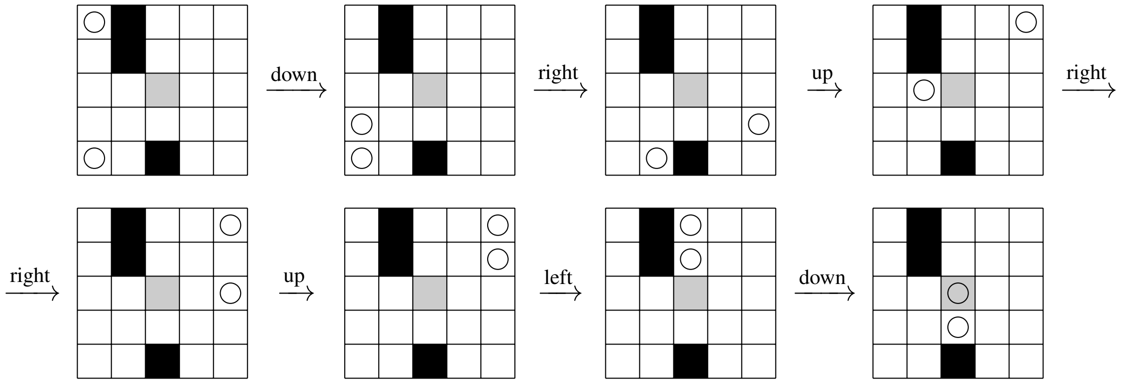
Problem 5-6. [40 points] **6.006 Tilt**

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 $C(n, b, s) = \frac{1}{s!}\prod_{i=0}^{s-1}(n^2 - b - i)$. (SEE PREV. PAGES)

(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.

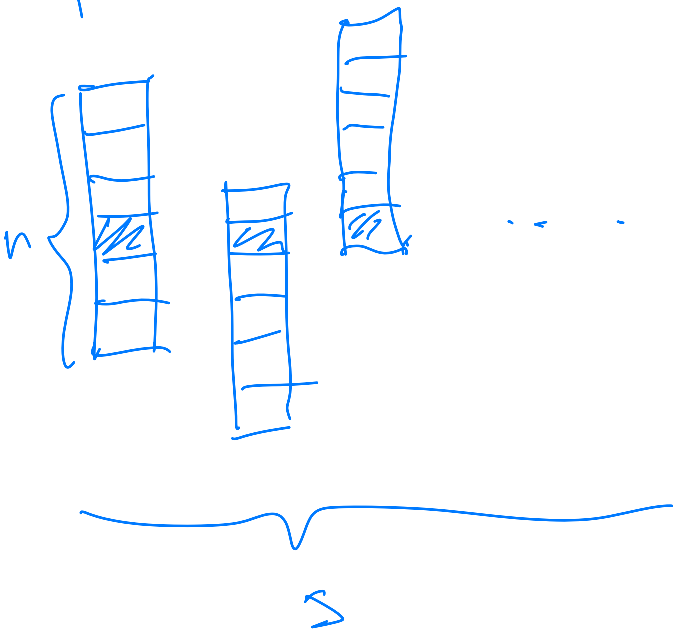
(c) [10 points] Given a 6.006 Tilt puzzle (B, t), describe an algorithm to return a move sequence that solves the puzzle in the fewest moves, or return that no such move sequence exists. If the puzzle's shortest solution uses k moves ($k = \infty$ if the puzzle is not solvable), your algorithm should 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. (SEE CODE)

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

B may have $\Omega(n^s)$ predecessors.

$$\underbrace{n \cdot n \cdot n \cdots n}_{s}$$



We know in state B, all sliders will be at their furthest