CS 255 Midterm

Directions:

- Don't Panic.
- You have **70 minutes**.
- You are allowed to have one hand-written two-sided 8.5×11.5 cheat sheet.
- No phones, laptops, tablets, textbooks, lecture notes or other resources are allowed.
- I recommend you read every question first. If something is not clear, please ask for clarification as soon as possible.
- Answer all True/False questions and provide a brief justification of example (1/2 point for answer, 1/2 point for justification/example).
- The remaining 3 problems are worth 20 points each; **answer 2** of them. **Clearly indicate which questions you want graded.**
- Rubrics for flow reductions and linear programming are on the last pages.
- Write your name and SJSU ID below

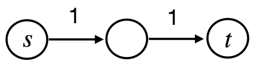
Name: ____

SJSU ID: ________________________

Determine if the following statements are True or False. Provide a brief justification or example. **(1 Point Each)**: 1/2 point for correct answer (True/False), 1/2 point for justification or example.

1. In *every* flow network with integer capacities, there exists an edge e where increasing the capacity of the edge will increase the value of the maximum flow.

False: Consider the following network with edge capacities written over each edge.



2. Consider a primal/dual pair of linear programs. If the dual problem is unbounded, then the primal problem is infeasible.

True: By weak duality, $c^T x \leq b^T y$ for any primal feasible x and dual feasible y. If $b^T y$ can be arbitrarily small then, the primal must be infeasible.

3. Let $G(L \cup R, E)$ be a bipartite graph and let $N(S)$ denote the neighbors of $S \subset L$. If $|L| = |R|$ and $|N(S)| \geq |S|$ for every subset $S \subseteq L$, then a perfect matching exists.

True: This is Hall's theorem.

4. Consider the primal/dual pair of linear programs below. The points $x = [1, 2]$ and $y = [1/4, 2]$ are optimal solutions to the primal and dual respectively.

$$\begin{array}{llcll} & \max x_1 + 2x_2 & \qquad & & \min 6y_1 + 3y_2 \\ \text{s.t.} & 4x_1 + x_2 \leq 6 & & \text{s.t.} & 4y_1 \geq 1 \\ & x_2 \leq 3 & & & y_1 + y_2 \geq 2 \\ & x_1, x_2 \geq 0 & & & y_1, y_2 \geq 0 \end{array}$$

False: By strong duality $c^T x^* = b^T y^*$ at optimal (x^*, y^*). However, for these points $1 + 2 \times 2 = 5 < 7.5 = 6/4 + 3 \times 2$.

5. Suppose you are in the lobby of Duncan Hall and you need to get to the 4th floor. Taking the stairs takes S seconds, while the elevator only takes $E \ll S$ seconds. However, the elevator comes at unknown time intervals; sometimes it comes quickly, sometimes it takes many minutes. Consider the following strategy. Wait for the elevator for up to $S - E$ seconds. If it arrives before $S - E$ seconds take it to the 4th floor, otherwise take the stairs. This strategy is 2 competitive.

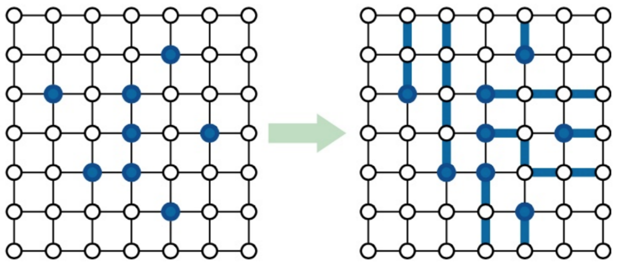
True: This strategy is optimal if the elevator arrives within $S - E$ seconds. In the worst case, you wait $S - E$ seconds then take the stairs. In total $S - E + S = 2S - E$. The hindsight optimal is just take the stairs with time S. Then

$$\frac{\text{ALG}}{\text{OPT}} = \frac{2S - E}{S} = 2 - E/S < 2.$$

6. **(20 Points)** An $n \times n$ grid is an undirected graph with n^2 vertices organized into n rows and n columns. Every vertex is connected to the nearest vertex (if any) above, below, to the right, and to the left.

Suppose m distinct vertices in the $n \times n$ grid are marked as *terminals*. The ***escape problem*** asks whether there are m vertex-disjoint paths in the grid that connect the terminals to any m distinct boundary vertices. Describe and analyze an efficient algorithm to solve the escape problem.

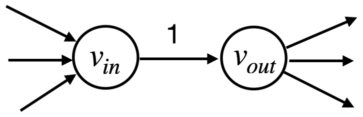
For example, given the input on the left below, your algorithm should return TRUE.



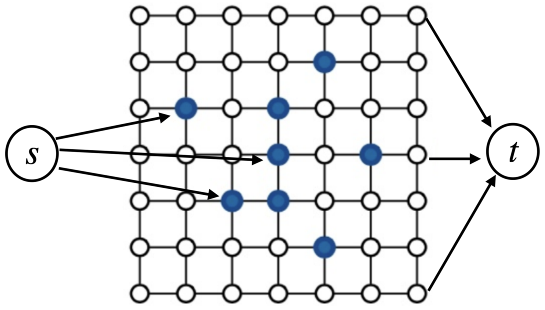
Solution: First, I will give a correct solution, then I will go over a few common approaches that are not quite right.

We convert this into a max-flow problem. First, we replace each undirected edge with two directed edges. These edges have capacity 1.

Next we give each vertex v a capacity by creating two copies v_{in} and v_{out}. All edges connected to v now connect to v_{in} and all out going edges connect to v_{out}. An edge with capacity 1 connects v_{in} to v_{out}. This ensures at most one path can use this vertex.



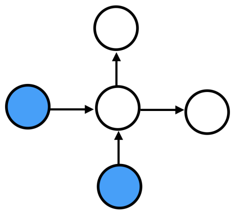
Next, we add a source vertex s and a sink vertex t. We add an edge from s to the v_{in} vertex for each of the m terminals with capacity 1. Finally, for each vertex v_{out} on the boundary, we create an edge with capacity 1 to t.



If there is a flow in this network with value m (the number of terminals), then there exist m vertex disjoint paths from the terminals to the boundary. We compute the max-flow and return TRUE if the value is m, FALSE otherwise.

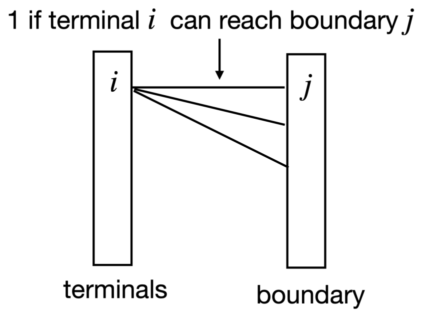
There are $|V| = n^2$ vertices, and $|E| = O(n^2)$ edges in the network. We can compute a max-flow in $O(VE)$ time. Therefore, the run time is $O(n^4)$.

Common Issue 1: Edge-disjoint Paths Max-Flow. This solution does almost the same construction as above, but does not give each vertex a capacity by creating v_{in} and v_{out} nodes. This will give edge-disjoint paths which, while closely related, is not quite correct. For example, see the figure below. Terminals are shown as blue circles. There are two paths that use different edges, but they share one vertex where they cross.



I set the baseline partial credit for this type of solution to 16/20.

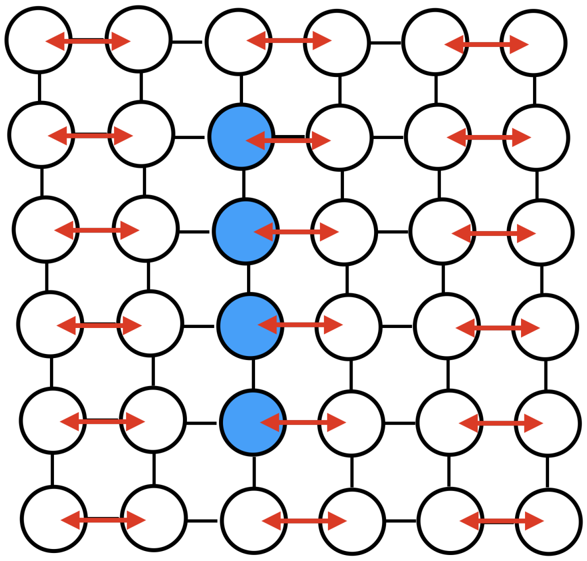
Common Issue 2: Matching Terminals to Paths. This type of solution proposed to match terminals to paths leading to boundary nodes. The exact approach varied, but generally the idea was to use BFS/DFS to check if terminal i and boundary node j are connected and then construct a bipartite graph similar to the one shown below.



This approach does ensure that paths end at distinct boundary points. However, it does not guarantee that the paths are vertex-disjoint. I set the partial credit baseline for this type of solution to 15/20. It is one point less than the approach above since the details of how we find the necessary paths were somewhat ambiguous.

Common Issue 3: Vertex-Disjoint Path Cover. The final type of solution tried a minimum vertex-disjoint path cover by creating a bipartite graph with one copy of the vertices on the left and another on the right. A matching assigns vertex i a successor j in a path. These paths will be vertex disjoint, but there is not guarantee that they reach the boundary. The grid graph G is not acyclic. This means that neighboring vertices i and j can match to each other. That is, i's successor is j and j's successor is i, creating a cycle. For example, in the figure below the double ended red arrows show neighbors that matched to each other. Notice that this is a maximum matching, indeed every vertex is assigned a successor. While it is possible for the four terminals (shown in blue) to reach the boundary, the matching does not find paths for *any* of them.

I set the baseline partial credit for this type of solution to 14/20.



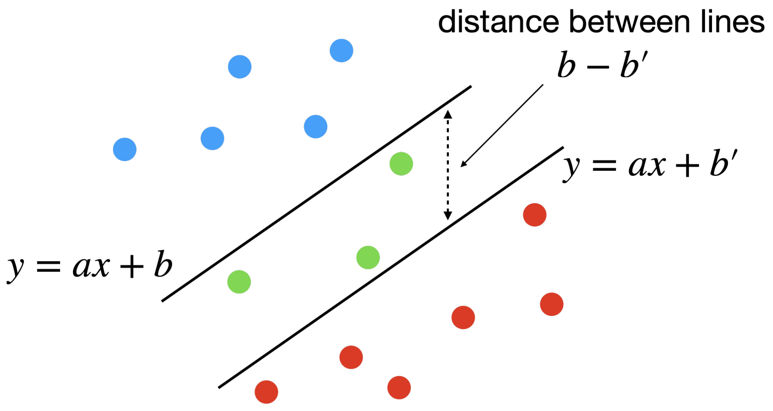
7. **(20 Points)** Suppose we are given a set R of n *red* points, a set G of n *green* points, and a set B of n *blue* points; each point is given as a pair (x, y) of real numbers. We call these sets separable if there is a pair of parallel lines $y = ax + b$ and $y = ax + b'$ such that (1) all red points are below both lines, (2) all blue points are above both lines, and (3) all green points are between the lines.

(a) Describe a linear program that is feasible if and only the sets G, B, R are separable.

(b) Give a linear program whose solution describes a pair of parallel lines that separates G, B, R whose vertical distance is as small as possible. (You can assume that G, B, R are separable.)

You do not need to solve the problems, just write down the linear programs.

Solution:
Our decision variables are the slope (of both lines) $a \in \mathbb{R}$ and the two intercept terms $b, b' \in \mathbb{R}$. The problem asks us to find parallel lines that separate the points as in the figure below.



To check feasibility, we only need the constraints. For part (b), the vertical distance between the lines is: $b - b'$ (assuming $b > b'$ as in the figure above and the constraints below). Therefore, we minimize this value.

$$\begin{aligned}
&\min\ b - b' \\
\text{s.t.}\quad & y_i \geq a x_i + b_i, \text{ for all blue points } (x_i, y_i) \in B \\
& y_i \geq a x_i + b'_i, \text{ for all blue points } (x_i, y_i) \in B \\
& y_i \leq a x_i + b_i, \text{ for all green points } (x_i, y_i) \in G \\
& y_i \geq a x_i + b'_i, \text{ for all green points } (x_i, y_i) \in G \\
& y_i \leq a x_i + b_i, \text{ for all red points } (x_i, y_i) \in R \\
& y_i \leq a x_i + b'_i, \text{ for all red points } (x_i, y_i) \in R \\
& a, b, b' \in \mathbb{R}
\end{aligned}$$

Note that the constraints

$$\begin{aligned}
y_i \leq a x_i + b_i, &\text{ for all green points } (x_i, y_i) \in G \\
y_i \geq a x_i + b'_i, &\text{ for all green points } (x_i, y_i) \in G,
\end{aligned}$$

imply that $b \geq b'$. We could force this condition by explicitly adding it as a constraint. There are two constraints per point (in B, R, or G); so $O(n)$ in total.

The only somewhat common issue in student solutions was leaving the objective as min $|b - b'|$, which is non-linear. I deducted two points for this. It would be possible to introduce a new variable $\delta > 0$, and make the objective min δ, with the additional constraints

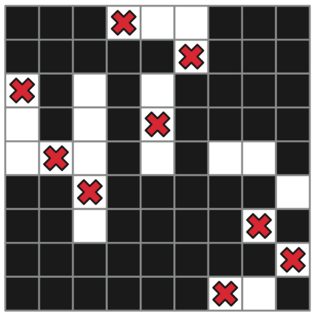
$$\begin{aligned}
\delta &\geq b - b' \\
\delta &\geq b' - b,
\end{aligned}$$

since this implies $\delta \geq \max(b - b', b' - b) = |b - b'|$.

8. **(20 Points)** Suppose we are given an $n \times n$ square grid, some of whose squares are colored black and the rest white. Describe and analyze an algorithm to determine whether tokens can be placed on the grid so that

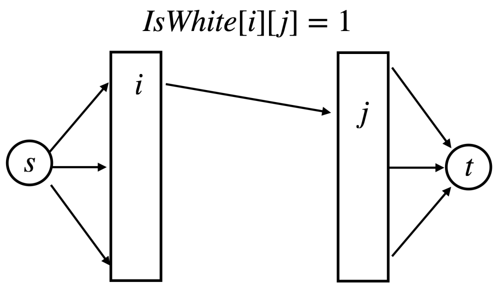
- every token is on a white square
- every row of the grid contains exactly one token
- every column of the grid contains exactly one token.

Your input is a two dimensional array is $IsWhite[1 \dots n][1 \dots n]$ of booleans, indicating which squares are white. Your output is a single boolean. For example, given the grid in the figure below, your algorithm should return TRUE.



Solution: We model the problem as a perfect bipartite matching in a graph $G(L \cup R, E)$. We create a left vertex for each of the n rows, and right vertex for each of the n columns. We make an edge with infinite capacity between the $i \in L$ and $j \in R$ if $IsWhite[i][j] =$ TRUE. If row $i \in L$ and col $j \in R$ are matched, this means place a token on the square (i, j). By construction, each row matches to at most one column and each column to at most one row. In a perfect matching, each row is assigned one column so exactly one token in each row and exactly one token in each column. There, we return TRUE if and only if there is a perfect matching.

You could just state run time at this point. Or, you can give a full reduction to max-flow in the standard way. We make a source s and add edges from s to each vertex $i \in L$ with capacity 1. We make sink t and add edges from each vertex on the right to t with capacity 1. Refer to the figure below.



We can compute a maximum bipartite matching (or max-flow) in $O(VE)$ time. In our graph there $|V| = 2n$ vertices and $|E| = O(n^2)$ edges in the worst case (all squares are white.) The time to construct the graph is $O(n^2)$. Then, the runtime is dominated by the matching, $O(n^3)$ overall.

Student solutions to this problem were mostly good. The only somewhat common issue was leaving the run time as $O(nE)$. We can use the simple bound $|E| = O(n^2)$ (worst case is when all squares are white.) I deducted 1 point for this.