CS 473 ✧ Fall 2022

Homework 8

Due **Wednesday**, November 9, 2022 at 9pm

---

1. The Autocratic Party is gearing up their fund-raising campaign for the 2024 election. Party leaders have already chosen their slate of candidates for president and vice-president, as well as various governors, senators, representatives, city council members, school board members, judges, and dog-catchers. For each candidate, the party leaders have determined how much money they must spend on that candidate's campaign to guarantee their election.

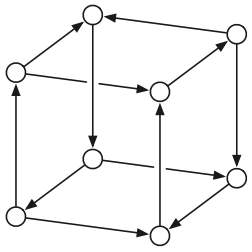
   The party is soliciting donations from each of its members. Each voter has declared the total amount of money they are willing to give each candidate between now and the election. (Each voter pledges different amounts to different candidates. For example, everyone is happy to donate to the presidential candidate,[1] but most voters in New York will not donate anything to the candidate for Trash Commissioner of Los Angeles.) Federal election law limits each person's total political contributions to $100 per day.

   Describe and analyze an algorithm to compute a donation schedule, describing how much money each voter should send to each candidate on each day, that guarantees that every candidate gets enough money to win their election. (Party members will of course follow their given schedule perfectly.[2]) The schedule must obey both Federal laws and individual voters' budget constraints. If no such schedule exists, your algorithm should report that fact.

   Assume there are n candidates, p party members, and d days until the election. The input to your algorithm is a pair of arrays $Win[1..n]$ and $Limit[1..p, 1..n]$, where $Win[i]$ is the amount of money candidate i needs to win, and $Limit[i, j]$ is the total amount party member i is willing to donate to candidate j.

   Your algorithm should return an array $Donate[1..p, 1..n, 1..d]$, where $Donate[i, j, k]$ is the amount of money party member i should donate to candidate j on day k.

2. A ***k-orientation*** of an undirected graph G is an assignment of directions to the edges of G so that every vertex of G has at most k incoming edges. For example, the figure below shows a 2-orientation of the graph of the cube.



[1] or some nice men in suits will be visiting their home.

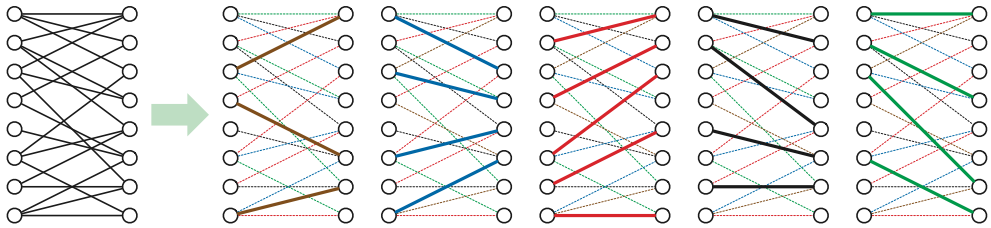
[2] See previous footnote.

Describe and analyze an algorithm that determines the smallest value of k such that G has a k-orientation, given the undirected graph G as input. Equivalently, your algorithm should find an orientation of the edges of G such that the maximum in-degree is as small as possible. For example, given the cube graph as input, your algorithm should return the integer 2.

3. Let $G = (L \sqcup R, E)$ be a bipartite graph, whose left vertices L are indexed $\ell_1, \ell_2, \ldots, \ell_n$ and whose right vertices are indexed $r_1, r_2, \ldots, r_n$. A matching M in G is ***non-crossing*** if, for every pair of edges $\ell_i r_j$ and $\ell_{i'} r_{j'}$ in M, we have $i < i'$ if and only if $j < j'$. If we place the vertices of G in index order along two vertical lines and draw the edges of G as straight line segments, a matching is non-crossing if its edges do not intersect.

   Describe and analyze an algorithm to find the smallest number of disjoint non-crossing matchings $M_1, M_2, \ldots, M_k$ such that each edge in G lies in exactly one matching M_i.

   [Hint: How would you compute the largest non-crossing matching in G?]



Decomposing a bipartite graph into non-crossing matchings.

4. **[just for practice, not for submission]** Suppose we are given a chessboard with certain squares removed, represented as a two-dimensional boolean array $A[1..n, 1..n]$. Describe and analyze efficient algorithms to place as many chess pieces of a given type onto the board as possible, so that no two pieces attack each other. A piece can be placed on the square in row i and column j if and only if $A[i, j] = \text{TRUE}$. Specifically:

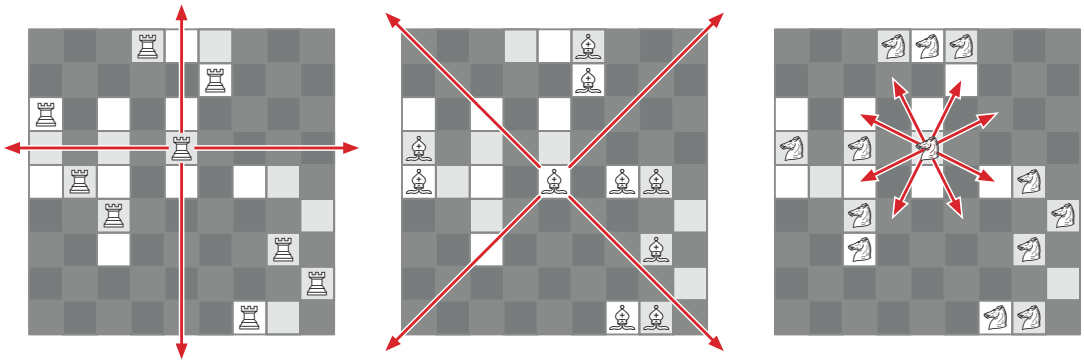
   (a) Describe an algorithm to places as many *rooks* on the board as possible. A rook on square (i, j) attacks every square in the same row or column; that is, every square of the form (i, k) or (k, j).

   (b) Describe an algorithm to places as many *bishops* on the board as possible. A bishop on square (i, j) attacks every square on the same diagonal or back-diagonal; that is, every square of the form $(i + k, j + k)$ or $(i + k, j - k)$.

   *(c) Describe an algorithm to places as many *knights* on the board as possible. A knight on square (i, j) attacks the eight squares $(i \pm 1, j \pm 2)$ and $(i \pm 2, j \pm 1)$.

   ★(d) Prove that placing as many *queens* on the board as possible is NP-hard. A queen attacks like either a rook or a bishop; that is, it attacks every square on the same row, column, diagonal, or back-diagonal.

Examples of (a), (b), and (c) are shown on the next page.



From left to right: Rooks, bishops, and knights.