1. Bipartite Matching, Chess and Dominoes

A matching in a graph \( G = (V, E) \) is a node-disjoint subset of edges \( M \subseteq E \). That is, each vertex \( v \) has at most one incident edge \((v, u)\) in \( M \).

(a) Define the maximum matching problem (i.e., find a matching of largest size) as an integer linear program with variables \( x_e \) for each \( e \in E \) indicating whether \( e \in M \). Your integer LP should have \( m + n = |E| + |V| \) constraints, including \( x_e \in \{0, 1\} \) for all \( e \in E \).

(b) Now, we relax the constraint \( x_e \in \{0, 1\} \) and just require the \( x_e \) to be positive (and not necessarily integral). This is called an LP relaxation. Write the obtained LP (again, \( m + n \) constraints) and derive its dual. What classic problem is this a relaxation of?

(c) For general graphs, the relaxation is strict, and can attain strictly higher objective value than the maximum matching size. Provide such an example solution in a triangle graph.

(d) For bipartite graphs, the relaxation is tight. Using maximum flow and integrality of the maximum flow in a network with integer capacities, prove that the maximum matching size in a bipartite graph is precisely equal to the optimal value of the LP of part (b). Along the way, provide a polynomial-time algorithm to compute a maximum matching in a bipartite graph.
A classic puzzle asks to prove that after “cutting out” opposite corner squares of an 8 × 8 chessboard, as in Figure 1, one cannot cover the remaining squares of board with non-overlapping 2 × 1 domino tiles, as in Figure 2. Now, you are provided with an n × n chess board with some given subset of squares cut out, give a polynomial-time algorithm to compute a covering with 2 × 1 domino tiles of the remaining

![Chessboard with opposite corners cut out](image1)

**Figure 1**: An 8 × 8 chessboard with opposite corners cut out (indicated by scissors).

![Domino tile](image2)

**Figure 2**: A 2 × 1 domino tile.

### 2. Min-Cut Edges

Each edge in a flow network \( N = (V, E, s, t, c) \) either appears in no minimum \( s-t \) cuts, some of the \( s-t \) cuts, or all of the \( s-t \) cuts. For this question you may assume for simplicity that the capacities are all integral.

(a) Give a linear-time (i.e. \( O(m+n) \)) algorithm to find all edges that appear in all \( s-t \) min cuts of a flow network.

(b) Give a polynomial-time algorithm to find all edges that appear in some \( s-t \) cuts. Extra credit will be given for a linear-time algorithm.

**Important**: prove correctness of your algorithms.

### 3. Placing Stones

Given an \( n \times m \) chessboard. You are asked to place stones onto the chessboard, such that the \( i \)-th row contains at least \( L_i \) stones, and the \( j \)-th column contains at least \( R_j \) stones. You are also given a set of positions on the chessboard, on which you cannot place stones. Determine the minimum number of stones you need to place to satisfy all constraints. Your algorithm should run in polynomial time.

\(^1\)Can you give an elementary proof of this?
4. **Counting Solutions**

You are given \( n \) sets of integers \( S_1, S_2, \ldots, S_n \subseteq \{0,1,2,\ldots,p-1\} \). For each \( r \in \{0,1,2,\ldots,p-1\} \), count the number of solutions to the following set of constraints.

\[
x_i \in S_i, \text{ for } 1 \leq i \leq n,
\]

and

\[
\left( \sum_{i=1}^{n} x_i \right) = r \mod p.
\]

Notice that the algorithm should output \( p \) numbers (for each \( r \)). Your algorithm should run in \( O(np \log p) \) time.

5. **Preflow Push Example**

Consider the following undirected flow graph

\[
s-v_1-\cdots-v_n-t
\]

where all edges have capacity 2 except the edge \((v_n, t)\) which has capacity 1.

Show that the Preflow-Push algorithm from class requires \( \Theta(n^2) \) relabels and pushes on this graph.