1. **Algorithm:** Construct a bipartite graph $G = (V_1, V_2, E)$ as follows.
   (a) Create one vertex $u_i \in V_1$ for each row; create one vertex $v_j \in V_2$ for each column.
   (b) For each pair i, j where $1 \geqslant i \geqslant n$ and $1 \leqslant j \leqslant m$, create an edge (u_i, v_j) with capacity $c(u_i, v_j) = b_{i,j}$.
   (c) Create a source s and an edge from s to each $u_i \in V_1$, with $c(s, u_i) = r_i$.
   (d) Create a sink t and an edge from each $v_j \in V_2$ to t, with $c(v_j, t) = c_j$.

   Then, run a max-flow algorithm to find a maximum flow on the constructed network. If the resulting flow f does not saturate the edges connecting to s and t, then there is no possible matrix solution. Otherwise $a_{i,j} = f(u_i, v_j)$ gives a solution.

   Pseudo-code:

```
$V_1 \leftarrow \{u_1, u_2, \dots, u_n\}$
$V_2 \leftarrow \{v_1, v_2, \dots, v_m\}$
$E \leftarrow \{(s, u_i) \text{ with } c(s, u_i) = r_i : 1 \leqslant i \leqslant n\} \cup$
      $\{(v_j, t) \text{ with } c(v_j, t) = c_j : 1 \leqslant j \leqslant m\} \cup$
      $\{(u_i, v_j) \text{ with } c(u_i, v_j) = b_{i,j} : 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m\}$
Find max-flow $f$ in the network $N = (\{s,t\} \cup V_1 \cup V_2, E)$
if $\exists u_i \in V_1, f(s, u_i) < c(s, u_i)$ or $\exists v_i \in V_2, f(v_i, t) < c(v_i, t)$:
    return NIL
else:
    Build a matrix $A$ such that $a_{i,j} = f(u_i, v_j)$
    return $A$
```

   Correctness: Suppose the max-flow f saturate all edges from s and all edges to t. By the conservation constraint on $u_i \in V_1$, we have:

$$r_i = c(s, u_i) = f(s, u_i) = \sum_{k=1}^{m} f(u_i, v_k) = \sum_{k=1}^{m} a_{i,k}$$

   Similarly by the conservation constraint on $v_i \in V_2$, we have:

$$c_j = c(v_j, t) = f(v_j, t) = \sum_{k=1}^{m} f(u_k, v_j) = \sum_{k=1}^{m} a_{k,j}$$

   Therefore the returned matrix A is a valid solution.

   On the other hand, for a valid matrix solution A, consider the following flow f by:

   - $\forall u_i \in V_1, f(s, u_i) = c(s, u_i)$
   - $\forall v_j \in V_2, f(v_j, t) = c(v_j, t)$
   - $\forall u_i \in V_1, v_j \in V_2, f(u_i, v_j) = a_{i,j}$

   f is a flow that satisfies the conservation constraint on all vertices and $|f| = \sum_{i=1}^{n} r_i = \sum_{j=1}^{m} c_j$. Therefore if the max-flow algorithm outputs a flow that does not saturate the edges from s and edges to t, it means that no such matrix solution A exists.

   Runtime: Constructing the network takes time $\mathcal{O}(nm)$, and so does constructing a solution matrix A from maximum flow values. Finding a maximum flow is the most time-consuming part of the algorithm, requiring time $\mathcal{O}(nm^2)$ if we use the Edmonds-Karp implementation of the Ford-Fulkerson algorithm.

2. **Algorithm:** For every non-blocked chessboard position (x,y), create a vertex $v_{x,y}$. For every pair of non-blocked positions (x_1,y_1) and (x_2,y_2) on which knights can attack each other, create an edge between v_{x_1,y_1} and v_{x_2,y_2}. Note that the resulting graph is a bipartite graph because each odd position (i.e., $x+y$ is odd) can only attack even positions (i.e., $x+y$ is even) and vice versa. Therefore run the maximum independent set algorithm on the bipartite graph to find a maximum independent set, and place a knight on a chessboard position iff the corresponding vertex is in the resulting independent set.

Pseudo-code:

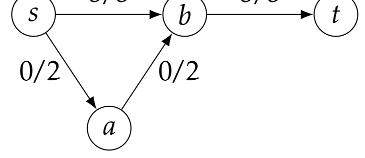
```
$V \leftarrow \{v_{0,0}, v_{0,1}, \dots, v_{n-1,n-1}\}$
$E \leftarrow \varnothing$
for $i$ in $0,1,\dots,n-1$:
    for $j$ in $0,1,\dots,n-1$:
        if $(i,j)$ and $(i-1,j-2)$ are non-blocking valid positions:
            $E \leftarrow E \cup \{(v_{i,j}, v_{i-1,j-2})\}$
        if $(i,j)$ and $(i-1,j+2)$ are non-blocking valid positions:
            $E \leftarrow E \cup \{(v_{i,j}, v_{i-1,j+2})\}$
        if $(i,j)$ and $(i+1,j-2)$ are non-blocking valid positions:
            $E \leftarrow E \cup \{(v_{i,j}, v_{i+1,j-2})\}$
        if $(i,j)$ and $(i+1,j+2)$ are non-blocking valid positions:
            $E \leftarrow E \cup \{(v_{i,j}, v_{i+1,j+2})\}$
        if $(i,j)$ and $(i-2,j-1)$ are non-blocking valid positions:
            $E \leftarrow E \cup \{(v_{i,j}, v_{i-2,j-1})\}$
        if $(i,j)$ and $(i-2,j+1)$ are non-blocking valid positions:
            $E \leftarrow E \cup \{(v_{i,j}, v_{i-2,j+1})\}$
        if $(i,j)$ and $(i+2,j-1)$ are non-blocking valid positions:
            $E \leftarrow E \cup \{(v_{i,j}, v_{i+2,j-1})\}$
        if $(i,j)$ and $(i+2,j+1)$ are non-blocking valid positions:
            $E \leftarrow E \cup \{(v_{i,j}, v_{i+2,j+1})\}$
$I \leftarrow$ find maximum independent set for $G = (V,E)$
place knights on all positions $(i,j)$ where $v_{i,j} \in I$
```

Correctness: Because any two vertices in an independent set do not share an edge, the placed knights therefore will not be able to attack each other. So every independent set yields a valid placement of knights. On the other hand, any knight coexistence placement maps back to some independent set on the constructed bipartite graph. Hence, the maximum number of placed knights is equal to the maximum size of any independent set.

Runtime: Constructing the network takes time $\mathcal{O}(n^2)$, and so does constructing a solution from a maximum independent set. Finding a minimum cut (to find a maximum independent set) is the most time-consuming part of the algorithm, requiring time $\mathcal{O}(nm^2)$ if we use the Edmonds-Karp implementation of the Ford-Fulkerson algorithm.

3. **(a)** We *disprove* the statement with the counter-example on the right, where $e_0 = (s,b)$ and the network N and max flow f^* are as depicted.

If we reduce $c(s,b)$ from 3 to 2, it is possible to adjust the flow in the resulting network N' by setting $f'(s,b) = 2$ and $f'(s,a) = f'(a,b) = 1$ so that $|f'| = 3 = |f^*|$.



(b) Algorithm: On input N, f^*, e_0 as described in the problem statement, let $e_0 = (u, v)$:

i. Start with $f'(e) = f^*(e)$ for all edges $e \in N'$.

ii. Run BFS in N' to find *reducing* paths starting from v—where every forward edge has non-zero flow ($f'(e) > 0$) and every backward edge has unused capacity ($f'(e) < c(e)$). This yields one of two possibilities: a path from v to t, or a path from v to u that does *not* go through t.

iii. If this finds a path from v to u that does not go through t, then N' contains a *reducing cycle* C starting and ending at u.
   - Reduce flow f' along cycle C, by subtracting 1 from the flow on every forward edge (including e_0) and adding 1 to the flow on every backward edge.
   - Return f'.

iv. Else, this finds some reducing path P_1 from v to t.
   - Run BFS again in N' to find a reducing path P_2 from s to u.
   - Reduce flow f' along path $P_2 + e_0 + P_1$, by subtracting 1 from the flow on every forward edge (including e_0) and adding 1 to the flow of every backward edge.
   - Run BFS in N' to find an augmenting path; if one exists, augment flow f' along that path.
   - Return f'.

Correctness: Finding a reducing path allows us to adjust the initial flow $f' = f^*$ so that $f'(e_0) = c'(e_0)$ while preserving flow conservation (for every node on the reducing path, the total flow in is still equal to the total flow out after the adjustment). After this adjustment, f' is a valid flow for N'. Moreover, if the flow is adjusted along a reducing cycle, then this does not reduce the total flow in the network so the resulting f' is still maximum. Else, the only difference between N and N' is that $c'(e_0) = c(e_0) - 1$, and we have $|f'| = |f^*| - 1$ after the reduction, so either f' is a maximum flow for N' (in which case we will not find an augmenting path and the algorithm will return f'), or it is possible to augment f' by 1 unit (in which case we will find an augmenting path and the algorithm will return the updated f').

Runtime: Each execution of BFS takes time $\Theta(|V|+|E|)$; each flow reduction and augmentation takes time $\Theta(|V|)$ (because each path has length at most $|V|$). The total time is therefore linear: $\Theta(|V| + |E|)$.

(c) On input $N = (V, E)$:

```
S = $\varnothing$
Run an implementation of Ford-Fulkerson to find a maximum flow $f$.
For each edge $e \in E$ such that $f(e) = c(e)$:
    Run the algorithm from the previous part with $e_0 = e$,
        to update the maximum flow to $f'$.
    If $|f'| < |f|$, add $e$ to $S$.
Return $S$.
```

The algorithm runs in time $\mathcal{O}(|E| \times (|V| + |E|))$, and it is correct because it directly verifies whether or not reducing $c(e)$ reduces the value of the max flow, for each edge e with $f(e) = c(e)$.