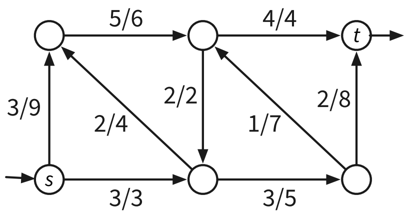
1. The figure below shows a flow network G, along with an (s,t)-flow f that is *not* a maximum flow. ***Clearly*** indicate the following structures in G:

   (a) An augmenting path for f.

   (b) The result of augmenting f along that path.

   (c) A maximum (s,t)-flow in G.

   (d) A minimum (s,t)-cut in G.



   (The answer booklet contains several drawings of G for you to annotate.)

2. A sequence of numbers $x_1, x_2, \ldots, x_\ell$ is ***restrained*** if each element after the first two is (loosely) between its two immediate predecessors; that is, for every index $i > 2$, we have $\min\{x_{i-1}, x_{i-2}\} \le x_i \le \max\{x_{i-1}, x_{i-2}\}$. Describe an efficient algorithm to compute the length of the longest restrained subsequence of a given array $A[1..n]$ of numbers.

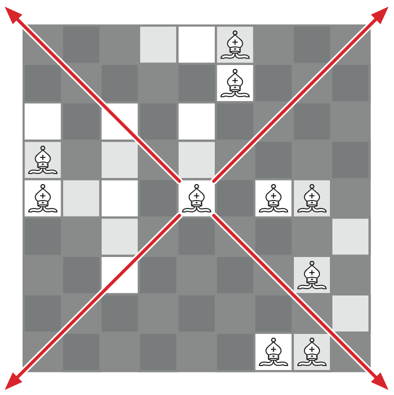
   For example, given the input array

$$[3,1,4,\underline{\mathbf{1}},5,\underline{\mathbf{9}},\underline{\mathbf{2}},6,5,3,5,\underline{\mathbf{8}},9,7,9,\underline{\mathbf{3}},2,3,\underline{\mathbf{8}},\underline{\mathbf{4}}],$$

   your algorithm should return the integer 7, which is the length of the restrained subsequence $\langle 1,9,2,8,3,8,4\rangle$.

3. Suppose you are given a chessboard with certain squares removed, represented as a two-dimensional boolean array $Legal[1..n, 1..n]$. A ***bishop*** is a chess piece that attacks every square on the same diagonal or back-diagonal; that is, a bishop on square (i,j) attacks every square of the form $(i+k, j+k)$ or $(i+k, j-k)$. Describe an algorithm to places as many bishops on the board as possible, each on a legal square, so that no two bishops attack each other.

   For example, given the 9×9 chess board shown below, where darker squares are illegal, your algorithm should return the integer 10.



4. Oh, no! Team Rocket has broken into your apartment and stolen your complete collection of Pokémon cards! You are too overwhelmed with grief to rebuild your *entire* collection, but you resolve to collect exactly 25% of the different card types. You can only buy one Pokémon card at a time. Each card comes in a sealed wrapper, which you can only open after you buy the card; each card you buy is equally likely to be any of the possible types. All purchases are fully independent.

   We can model your Pokémon-collection process using the following algorithm. We refer to each iteration of the main for-loop as a *phase*; for any index k, the kth phase ends just after you purchase the kth distinct card type. Here n denotes the number of different types; you can assume that n is divisible by 4.

```
GottaCatchAQuarterOfEm($n$):
  for $i \leftarrow 1$ to $n$
      $GotIt[i] \leftarrow 0$
  for $k \leftarrow 1$ to $n/4$
      $x \leftarrow$ Random($n$)          ⟨⟨buy a card⟩⟩
      while $GotIt[x] = 1$
          $x \leftarrow$ Random($n$)      ⟨⟨buy a card⟩⟩
      $GotIt[x] \leftarrow 1$
```

   (a) ***Prove*** that for all $1 \leq k \leq n/4$ and for all $m \geq 0$, the probability that you purchase more than m cards in the kth phase is at most 4^{-m}. *[Hint: Show that with probability at least* $3/4$*, each purchased card is a type you don't already own.]*

   (b) ***Prove*** that for all $1 \leq k \leq n/4$, the probability that the kth phase requires more than $2\log_2 n$ purchases is at most $1/n^2$. *[Hint: Use part (a).]*

   (c) ***Prove*** that with probability at least $1 - 1/n$, none of the $n/4$ phases requires more than $2\log_2 n$ purchases. *[Hint: Use part (b) and the union bound.]*

   (d) What is the *exact* expected *total* number of purchases to collect $n/4$ different card types? (A tight $O(\cdot)$ bound is worth significant partial credit.)

   [Hint: You may be able to solve each part by assuming earlier parts.]

5. Suppose you are given a bipartite graph $G = (L \sqcup R, E)$ and a maximum matching M in G. Describe and analyze efficient algorithms for the following operations:

   (a) Insert(u, v): Insert an edge between $u \in L$ and $v \in R$ and update the maximum matching. (You can assume that uv is not an edge before this function is called.)

   (b) Delete(uv): Delete edge uv and update the maximum matching. (You can assume that uv is actually an edge before this function is called.)

   Both of your algorithms should be significantly faster than recomputing the maximum matching from scratch. *[Hint: Think about the reduction from maximum matchings to maximum flows.]*

6. Let $G = (V, E)$ be an undirected graph. The ***neighborhood*** of a vertex v consists of v and every vertex adjacent to v. A ***double-dominating set*** in G is a set S of vertices such that for each vertex v, the neighborhood of v contains at least two vertices in S.

   Suppose you are given a graph G where every vertex has degree $d - 1$ (and thus the neighborhood of every vertex contains exactly d vertices), and each vertex v has a non-negative weight w_v. Your goal is to find a double-dominating set S in G whose total weight $\sum_{v \in S} w_v$ is as small as possible. Solving this problem *exactly* is NP-hard.

   (a) Write an integer linear program that ***exactly*** captures this problem. In particular, each solution of the integer linear program must describe a double-dominating set, and each double-dominating set must correspond to a solution of your integer linear program.

   You do not need to prove that your integer linear program is correct, but for partial credit, some justification is recommended.

   (b) Describe and analyze an efficient $(d/2)$-approximation algorithm for this problem. Remember to ***prove*** that your algorithm returns a valid solution, and ***prove*** that it achieves an approximation ratio of $d/2$. *[Hint: Use LP relaxation and rounding.]*

   Part (b) was broken; the correct approximation ratio from LP rounding is actually $d - 1$, not $d/2$. **Everyone received full credit for this subproblem.**