1. Let $G = (V, E)$ be an arbitrary undirected graph. Suppose we color each vertex of G uniformly and independently at random from a set of three colors: red, green, or blue. An edge of G is *monochromatic* if both of its endpoints have the same color.

   (a) What is the *exact* expected number of monochromatic edges? (Your answer should be a simple function of V and E.)

   Solution: Each edge is monochromatic with probability $1/3$, so by linearity of expectation, the expected number of monochromatic edges is exactly $\boldsymbol{E/3}$. ■

   Rubric: 2 points: all or nothing.

   (b) For each edge $e \in E$, define an indicator variable X_e that equals 1 if e is monochromatic and 0 otherwise. ***Prove*** that $\Pr[(X_a = 1) \wedge (X_b = 1)] = \Pr[X_a = 1] \cdot \Pr[X_b = 1]$ for every pair of edges $a \neq b$. This claim implies that the random variables X_e are pairwise independent.

   Solution: As in part (a), we immediately observe that $\Pr[X_e = 1] = 1/3$ for every edge e. Thus, we need to prove that $\Pr[(X_a = 1) \wedge (X_b = 1)] = 1/9$ for every pair of edges $a \neq b$.

   Let $uv \neq xy$ be two arbitrary edges. There are two cases to consider:

   - Suppose uv and xy have no vertices in common. There are $3^4 = 81$ possible color assignments to the four vertices u, v, x, y, each of which is equally likely. There are exactly 9 color assignments in which both uv and xy are monochromatic: 3 possible colors for uv, times 3 possible colors for xy. Thus, the probability that both uv and xy are monochromatic is $9/81 = 1/9$.
   - Suppose uv and xy share one vertex; without loss of generality, we can assume that $u = x$. There are $3^3 = 27$ possible color assignments to the three vertices u, v, y, each of which is equally likely. There are exactly 3 color assignments in which both uv and vy are monochromatic: all red, all green, and all blue. Thus, the probability that both uv and xy are monochromatic is $3/27 = 1/9$.

   In both cases, we have $\Pr[(X_{uv} = 1) \wedge (X_{xy} = 1)] = 1/9$, as required. ■

   Rubric: 5 points: 1 for recognizing two cases + 2 for proof for four vertices + 2 for proof for three vertices.

(c) ***Prove*** that there is a graph G such that

$$\Pr[(X_a = 1) \wedge (X_b = 1) \wedge (X_c = 1)] \neq \Pr[X_a = 1] \cdot \Pr[X_b = 1] \cdot \Pr[X_c = 1]$$

for some triple of distinct edges a, b, c in G. This claim implies that the random variables X_e are *not necessarily* 3-wise independent.

Solution: Suppose G has three vertices x, y, z connected by three edges xy, xz, yz. There are 27 possible color assignments to the vertices x, y, z, each of which is equally likely. There are exactly 3 color assignments that make all three edges monochromatic: All red, all green, and all blue. Thus, $\Pr[(X_{xy} = 1) \wedge (X_{xz} = 1) \wedge (X_{yz} = 1)] = 1/9$. But $\Pr[X_{xy} = 1] \cdot \Pr[X_{xz} = 1] \cdot \Pr[X_{yz} = 1] = 1/27$.

More succinctly: If edges xy and yz are both monochromatic, then xz *must* be monochromatic, so the three variables X_{xy}, X_{yz}, and X_{xz} cannot be independent. ■

Rubric: 3 points = 1 for counterexample + 2 for analysis

2. The King of Hearts has installed an app on Rabbit's pocket watch to automatically remind Rabbit of any upcoming appointments. For each reminder Rabbit receives, Rabbit has a 50% chance of actually remembering his appointment (decided by an independent fair coin flip).

   First, suppose the King of Hearts sends Rabbit k separate reminders for a *single* appointment.

   (a) What is the *exact* probability that Rabbit will remember his appointment? Your answer should be a simple function of k.

   Solution: $\mathbf{1-2^{-k}}$. ■

   Rubric: 2 points.

   (b) What value of k should the King choose so that the probability that Rabbit will remember this appointment is at least $1-1/n^\alpha$? Your answer should be a simple function of n and α.

   Solution:
   $$\begin{aligned} 1-2^{-k} \geq 1-1/n^\alpha &\iff 1/2^k \leq 1/n^\alpha \\ &\iff 2^k \geq n^\alpha \\ &\iff k \geq \lg n^\alpha = \boldsymbol{\alpha \lg n} \end{aligned}$$
   ■

   Rubric: 3 points. Max 2 points for $k = \Omega(\log n)$ (but not $k = O(\log n)$). Max 1 point for correct but looser bound.

   Now suppose the King of Hearts sends Rabbit k separate reminders for each of n different appointments. (That's nk reminders altogether.)

   (c) What is the *exact* expected number of appointments that Rabbit will remember? Your answer should be a simple function of n and k.

   Solution: $\boldsymbol{n(1-2^{-k})}$, by linearity of expectation. ■

   Rubric: 2 points.

   (d) What value of k should the King choose so that the probability that Rabbit remembers *every* appointment is at least $1-1/n^\alpha$? Again, your answer should be a simple function of n and α.

   Solution (union bound): The union bound $\Pr[A \cup B] \leq \Pr[A] + \Pr[B]$ implies that the probability of forgetting at least one appointment is at most $n/2^k$. It follows that the probability of remembering every appointment is at least $1-n/2^k$. Now we proceed exactly as in part (b):

   $$\begin{aligned} 1-n/2^k \geq 1-1/n^\alpha &\iff n/2^k \leq 1/n^\alpha \\ &\iff 2^k/n \geq n^\alpha \\ &\iff 2^k \geq n^{\alpha+1} \\ &\iff k \geq \lg n^{\alpha+1} = \boldsymbol{(\alpha+1)\lg n} \end{aligned}$$
   ■

Solution (Markov's inequality): Let X denote the number of *forgotten* appointments. Part (c) and linearity of expectation imply that $\operatorname{E}[X] = n/2^k$. Thus, Markov's inequality implies $\Pr[X \geq 1] \leq \operatorname{E}[X] = n/2^k$. Thus, the probability of remembering every appointment is $\Pr[X = 0] \geq 1 - n/2^k$. Now we proceed exactly as in part (b):

$$\begin{aligned} 1 - n/2^k \geq 1 - 1/n^\alpha &\iff n/2^k \leq 1/n^\alpha \\ &\iff 2^k/n \geq n^\alpha \\ &\iff 2^k \geq n^{\alpha+1} \\ &\iff k \geq \lg n^{\alpha+1} = \boldsymbol{(\alpha+1)\lg n} \end{aligned}$$ ■

Solution (exact): The probability that Rabbit remembers every appointment is exactly $(1 - 2^{-k})^n$, and therefore at least $1 - n/2^k$.

- This inequality is trivial when $n \leq 0$, so assume otherwise. Then

$$\begin{aligned} (1-2^{-k})^n &= (1-2^{-k}) \cdot (1-2^{-k})^{n-1} \\ &\geq \left(1-2^{-k}\right) \cdot \left(1 - \frac{n-1}{2^k}\right) && \text{[by the induction hypothesis]} \\ &= 1 - \frac{n}{2^k} + \frac{n-1}{4^k} \\ &\geq 1 - \frac{n}{2^k}. \end{aligned}$$

- Alternatively, the binomial theorem implies $(1-2^{-k})^n = \sum_{i=0}^{n} \binom{n}{i}(-2^k)^i$. As long as $n \leq 2 \cdot 2^i$, the terms of this sum decrease geometrically in absolute value, so throwing out all but the first two terms gives us an upper bound.
- Alternatively, the World's Most Useful Approximation $1 + x \approx e^x$ immediately implies $(1-2^{-k})^n \approx e^{n/2^k} \approx 1 - n/2^k$.

Now we proceed exactly as in part (b):

$$\begin{aligned} 1 - n/2^k \geq 1 - 1/n^\alpha &\iff n/2^k \leq 1/n^\alpha \\ &\iff 2^k/n \geq n^\alpha \\ &\iff 2^k \geq n^{\alpha+1} \\ &\iff k \geq \lg n^{\alpha+1} = \boldsymbol{(\alpha+1)\lg n} \end{aligned}$$ ■

Rubric: 3 points = 2 points for $(1 - n/2^k)$ + 1 for remaining details. Max 2 points for $k = \Omega(\log n)$ (but not $k = O(\log n)$). Max 1 point for correct but looser bound.

3. Describe and analyze an algorithm to find the minimum number of stations that must be closed to block all rail travel from Ffarquhar to Tidmouth. The Sodor rail network is represented by an undirected graph, with a vertex for each station and an edge for each rail connection between two stations. Two special vertices f and t represent the stations in Ffarquhar and Tidmouth.

Solution (vertex-disjoint paths): Without loss of generality, assume there is no edge directly between f and t, since otherwise separating f and t is impossible.

We compute and return the maximum number of vertex-disjoint paths from f to t in $O(VE)$ time, using the algorithm described in class. This number is equal to the minimum number of vertices required to separate f from t, by Menger's theorem. ■

Solution (maximum flow): Let $G = (V, E)$ be the input graph. We construct a new directed flow network $G' = (V', E')$ as follows:

- V' contains two vertices v^+ and v^- for every vertex $v \in V$.
- E' contains two edges $u^+ \to v^-$ and $v^+ \to u^-$ for every edge $uv \in E$, and an edge $v^- \to v^+$ for every vertex $v \in V$.
- All edges $v^- \to v^+$ have 1 and all edges $u^+ \to v^-$ and $v^+ \to u^-$ have infinite capacity. (Giving *every* edge capacity 1 is also correct, but that makes the proof more complicated.)

We con construct G' in $O(V+E)$ time by brute force. Then we compute a maximum (f^+, t^-)-flow in G' and return its value. If we compute the maximum flow using Ford-Fulkerson, the resulting algorithm runs in $O(V'E') = \boldsymbol{O(VE)}$ ***time***.

The maxflow-mincut theorem implies that the value of the maximum (f^+, t^-)-flow is equal to the capacity of the minimum (f^+, t^-)-cut. Thus, to prove the algorithm correct, we need to prove the following claim: We can separate f and t by removing k vertices from G if and only if G' has an (f^+, t^-) cut with capacity k.

$\Rightarrow$ Suppose we can separate f and t by removing a set C of k vertices from G. Let $C' = \{v^- \to v^+ \in E' \mid v \in C\}$ be the corresponding set of k edges in G'. For every path $f^+ \to u_1^- \to u_1^+ \to u_2^- \to \cdots \to u_\ell^+ \to t^-$ in G', there is a corresponding path $f \to u_1 \to u_2 \to \cdots \to u_\ell \to t$ in G. For every path α' from f' to t' in G', the corresponding path α in G contains a vertex in C, and therefore α' contains an edge in C'. Thus, C' is the set of edges crossing an (f^+, t^-)-cut in G'. Because each edge in C' has capacity 1, the cut has capacity k.

$\Leftarrow$ Let (S, T) be an (f^+, t^-)-cut in G' with capacity k, and let $C' = \{u \to v \in E' \mid u \in S \text{ and } v \in T\}$ be the set of edges crossing this cut. C' must contain exactly k edges of the form $v^- \to v^+$, because all other edges have infinite capacity. For every path $f \to u_1 \to u_2 \to \cdots \to u_\ell \to t$ in G, there is a corresponding path $f^+ \to u_1^- \to u_1^+ \to u_2^- \to \cdots \to u_\ell^+ \to t^-$ in G'. For every path α from f to t in G, the corresponding path α' in G' contains an edge in C' (because S, T is a cut), and therefore α contains a vertex in $C = \{v \mid v^- \to v^+ \in C'\}$. Thus, the k vertices in C separate f from t.

■

Rubric: 10 points = 2 for vertices + 2 for edges + 2 for capacities + 2 for algorithm + 2 for running time. No proof of correctness is required, because we didn't ask for one. Yes, the first solution is worth full credit, even without the phrase "by Menger's Theorem". These are not the only correct solutions.

4. Describe and analyze an algorithm to determine whether a given student can graduate. The input to your algorithm is the list of m requirements (each specifying a subset of the n courses and the number of courses that must be taken from that subset) and the list of courses the student has taken.

Solution: For each index j, let S_j be the subset of courses listed in the jth requirement, and let k_j be the number of courses that must be taken from S_j. We construct a flow network $G = (V, E)$ as follows:

- V contains vertices $c_1, c_2, \ldots, c_n$ corresponding to classes, m vertices $r_1, r_2, \ldots, r_m$ corresponding to requirements, a source vertex s, and a target vertex t.
- E contains three types of edges:
  - An edge $s \rightarrow c_i$ with capacity 1, for each class i that the student has taken.
  - An edge $c_i \rightarrow r_j$ with capacity 1, for all indices i and j such that $i \in S_j$.
  - An edge $r_j \rightarrow t$ with capacity k_j, for each requirement j.

We can construct this graph in $O(V + E) = O(mn)$ time by brute force. The student can graduate if and only if the maximum (s,t)-flow in G saturates every edge entering t. Thus, we can solve the problem in $O(VE) = \boldsymbol{O(mn(m+n))}$ ***time*** using Orlin's algorithm.

In fact, we can assume without loss of generality that $k_j \geq 0$ for every j (otherwise we can discard all vacuous requirements in preprocessing) and therefore that $m \leq \sum_j k_j \leq n$ (otherwise nobody can ever graduate). Thus, our algorithm runs in $\boldsymbol{O(n^3)}$ ***time***. ■

Rubric: 10 points = 2 for vertices + 2 for edges + 2 for capacities + 2 for algorithm + 2 for running time ($O(n^3)$ is fine). No proof of correctness is required, because we didn't ask for one. This is not the only correct solution, or the only correct formulation of this solution.