1. **(Kleinberg Tardos 7.27)** Some of your friends with jobs out West decide they really need some extra time each day to sit in front of their laptops, and the morning commute from Woodside to Palo Alto seems like the only option. So they decide to carpool to work. Unfortunately, they all hate to drive, so they want to make sure that any carpool arrangement they agree upon is fair and doesn’t overload any individual with too much driving. Some sort of simple round-robin scheme is out, because none of them goes to work every day, and so the subset of them in the car varies from day to day.

Here’s one way to define fairness. Let the people be labeled $S = \{p_1, \ldots, p_k\}$. We say that the total driving obligation of $p_j$ over a set of days is the expected number of times that $p_j$ would have driven, had a driver been chosen uniformly at random from among the people going to work each day. More concretely, suppose the carpool plan lasts for $d$ days, and on the $i^{th}$ day a subset $S_i \subseteq S$ of the people go to work. Then the above definition of the total driving obligation $\Delta_j$ for $p_j$ can be written as $\Delta_j = \sum_{i:S_i \ni p_j} \frac{1}{|S_i|}$.

Ideally, we’d like to require that $p_j$ drives at most $\Delta_j$ times; unfortunately, $\Delta_j$ may not be an integer.

So let’s say that a driving schedule is a choice of a driver for each day — that is, a sequence $p_{i_1}, p_{i_2}, \ldots, p_{i_d}$ with $p_{i_t} \in S_t$ — and that a fair driving schedule is one in which each $p_j$ is chosen as the driver on at most $\lceil \Delta_j \rceil$ days.

(a) Prove that for any sequence of sets $S_1, \ldots, S_d$, there exists a fair driving schedule.

(b) Give an algorithm to compute a fair driving schedule with running time polynomial in $k$ and $d$.

**Solution:** We convert the problem into a network flow problem. First we construct a graph as follows. We denote the vertex $p_i$ as the $i$-th driver. Moreover we denote the vertex $q_j$ as the $j$-th day. If $p_i$ can drive on the $j$-th day, we simply draw a directed edge from $p_i$ to $q_j$ of capacity 1. Finally we draw a source $s$ which connects each $p_i$ with capacity $\lceil \Delta_i \rceil$ and a sink which connects each $q_j$ with capacity 1. It is easy to find that computing a fair driving schedule is equivalent to computing the maximum flow problem. The only thing we need to do is to prove that the value of the maximum flow is $d$.

First of all, it is obvious that for any flow $f$, $|f| \leq d$. Thus if we are able to find a flow $f$ with $|f| = d$, we are done. This is easy to achieve. Consider the following flow.

$$f_{pq} = \frac{1}{|S_j|}, \quad f_{sp} = \sum_{i:S_i \ni p_j} \frac{1}{|S_j|} \leq \lceil \Delta_i \rceil, \quad f_{qj} = 1.$$  

This flow satisfies all the constraints and have value $n$. Thus there exists a fair schedule. For computing it, we simply adopt the Ford algorithm.

2. Recall Karger’s algorithm for the global min-cut problem. In this problem we modify the algorithm to improve its running time.
(a) Prove that if we stop the original Karger’s algorithm when the remaining number of vertices is 
\[ \max \left\{ \left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil, 2 \right\}, \]
the probability that we have contracted an edge in the min-cut is less than 1/2. Let’s call this procedure \textit{Partial Karger}.

\textbf{Solution:} Denote \( A_k \) as the event that we do not contract an edge in the min cut. Suppose we stop at the \( k \)-th step, then:

\[ P(\bigcap_{i=1}^{k} A_i) \geq \frac{(n - k)(n - k - 1)}{n(n - 1)} \]

If we set \( k = n - \left\lceil \frac{n}{\sqrt{2}} \right\rceil - 1 \), we have:

\[ P(\bigcap_{i=1}^{n-\left\lceil \frac{n}{\sqrt{2}} \right\rceil-1} A_i) \geq \frac{\left\lceil \frac{n}{\sqrt{2}} \right\rceil \left( \left\lceil \frac{n}{\sqrt{2}} \right\rceil + 1 \right)}{n(n - 1)} \geq \frac{n/\sqrt{2}(n/\sqrt{2} + 1)}{n(n - 1)} \geq \frac{1}{2} \]

(b) Now suppose we apply \textit{Partial Karger} to two copies of \( G \) to produce graphs \( G_1 \) and \( G_2 \). We then recursively apply these steps to \( G_1 \) and \( G_2 \) and so on until each recursive call returns a graph on two vertices. If \( r(n) \) is the running time of this process as a function of the number of vertices \( n \) of \( G \), derive a recursive equation for \( r(n) \) and solve it to obtain an explicit expression for the running time (you may use \( O(\cdot) \) notation to simplify your recursive equation).

\textbf{Solution:} The operation cost for contracting a single edge is \( O(n) \). Thus the operation cost for partial Karger is \( O(n^2) \). By recursion we have:

\[ r(n) = 2r(n/\sqrt{2}) + O(n^2) \]

By Master’s Theorem we obtain \( r(n) = O(n^2 \log n) \).

(c) Show that the algorithm in part (b) produces \( O(n^2) \) contracted graphs on two vertices each. Prove that the probability that at least one of them contains a global min-cut is at least \( 1/\log(n) \) up to a multiplicative constant.

\textbf{Hint:} Think of the recursion as a binary tree with paths leading to the \( O(n^2) \) leaves representing the two-vertex contracted graphs.

\textbf{Solution:} By using partial Karger’s algorithm, we obtain graphs \( G_1, G_2 \) from the original graph \( G \). Here \( G_1, G_2 \) have \( \left\lceil n/\sqrt{2} \right\rceil \) vertices. We continue using partial Karger’s algorithm, so that \( G_1, G_2 \) keep branching. In the end we get a binary tree. The height of the tree is \( \log_{\sqrt{2}} n \). Thus the total number of leaves is \( O(2^{\log_{\sqrt{2}} n} = O(n^2) \). Now we proceed to prove that the probability that at least one of the leaves contains a global min cut is greater than \( c/\log n \). We denote
such probability as $f(n)$. Moreover we denote $p$ as the probability in part (a). We know from part (a) that $p < 1/2$.

We consider $1 - f(n)$, which is the probability that none of the leaves contains a global min cut. Since the algorithm $G$ branches to $G_1$ and $G_2$. By independence we only consider the probability that none of $G_1$’s leaves contains a global min cut. There are two cases that can make this happen. (1) We contracted an edge in the min cut when we derive $G_1$ from $G$ using Karger’s algorithm, which has probability $p$. (2) We did not contract an edge in the min cut when we derive $G_1$ from $G$, but we unfortunately contracted a min-cut-edge in the following steps, which has probability $(1 - p)(1 - f(n/\sqrt{2}))$. Thus we have:

$$1 - f(n) = (p + (1 - p)(1 - f(n/\sqrt{2})))^2 = (1 - (1 - p)f(n/\sqrt{2}))^2$$

$$\leq (1 - \frac{1}{2}f(n/\sqrt{2}))^2 = 1 - f(n/\sqrt{2}) + \frac{1}{4}f^2(n/\sqrt{2})$$

Thus, $f(n) \geq f(n/\sqrt{2}) - \frac{1}{2}f^2(n/\sqrt{2})$. We prove by induction that $f(n) \geq c/\log n$ for some small $c$. Here we take the logarithm under base $\sqrt{2}$. Suppose this holds for $k \geq n$, then for $k = n$, we have:

$$f(n) \geq f(n/\sqrt{2}) - \frac{1}{4}f^2(n/\sqrt{2}) \geq \frac{c}{\log n - 1/2} - \frac{c^2}{4(\log n - 1/2)^2}$$

We want to prove that the right hand side is less than $c/\log n$, that is:

$$\frac{c}{\log n - 1/2} - \frac{c^2}{4(\log n - 1/2)^2} \leq \frac{c}{\log n}$$

This is equivalent to $\frac{2}{\log n} \leq c$. If we choose $c$ to be less than 2, then this indeed holds. Thus we completed the induction process.

(d) Compare the running time of the above algorithm to Karger’s original given the same probability of failure.

Solution: For the partial Karger’s algorithm, the success probability is $c/\log n$. Thus we need to run it $(\log n)$ times to achieve constant success rate. The total run time is $O(n^2\log n)$ times $O(\log n)$, which is $O(n^2\log^2 n)$ time. For traditional Karger’s algorithm, the total run time is $O(n^2m\log n)$. Obviously, partial Karger’s algorithms is significantly smaller.

3. An independent set in a graph is a set of vertices with no edges connecting them. Let $G$ be a graph with $nd/2$ edges ($d > 1$), and consider the following probabilistic experiment for finding an independent set in $G$: delete each vertex of $G$ (and all its incident edges) independently with probability $1 - 1/d$.

(a) Compute the expected number of vertices and edges that remain after the deletion process. Now imagine deleting one endpoints of each remaining edge.
Solution. Each node survives with probability $1/d$. Thus the expected number of nodes is $n/d$. Each edge survives with probability $1/d^2$ (both its ends must survive independently). Thus the expected number of edges is $nd/2 	imes 1/d^2 = n/2d$.

(b) From this, infer that there is an independent set with at least $n/2d$ vertices in any graph with on $n$ vertices with $nd/2$ edges.

Solution. After applying this sampling, we create an independent set as follows: for each edge in the resulting graph, delete one of the endpoints. After doing this for each edge, none of the remaining vertices are connected by any edges, i.e. form an independent set. If $G' = (V', E')$ is the graph we obtain after sampling, we expect the size of the independent set to be

$$E[\text{size of independent set}] = E[|V'| - |E'|] = n/d - n/2d = n/2d$$

Since there will be at least one outcome with a value equal to (or greater than) the expectation, by the probabilistic method there must be an independent set of size $\geq n/2d$.

4. Let $G = (V, E)$ be a bipartite graph on $n$ vertices with a list $S(v)$ of at least $1 + \log_2 n$ colors associated with each vertex $v \in V$. A graph coloring is an assignment of colors to nodes such that no two adjacent nodes have the same color. Prove that there is a proper coloring of $G$ assigning to each vertex of $v$ a color from its list $S(v)$.

Solution. Consider the union of all colors $S = \cup_v S(v)$. Then for each color $c \in S$, flip a fair coin, if heads, use $c$ to color all nodes $u$ on the left side which have $c \in S(u)$, else use it on the right. Never use color $c$ again, and never consider the colored nodes again.

If the above algorithm colors every node, it will have done so correctly since nodes on opposing sides of the partition never get the same color. It remains to show that the algorithm will with nonzero probability color all the nodes.

Let $X$ be the number of nodes that don’t receive a color. Let $X_i$ be the indicator variable that node $i$ does not get assigned a color. That means all the colors in $S(i)$ must have been assigned to the other side of the partition. Thus $E[X_i] = P[X_i] \leq (1/2)^{\log n + 1} = 1/2n$. So we have

$$E[X] = \sum_{i=1}^{n} E[X_i] \leq n(1/2n) = 1/2 < 1$$

Thus we expect less than 1 node to be left uncolored. Since $X$ only takes integer values, by definition of expectation this means $Pr[X = 0] > 0$. Thus the algorithm has some nonzero probability of outputting a valid coloring and thus one exists.

5. Prove that a graph can only have at most $\binom{n}{2}$ different cuts that realize the global minimum cut value.

Solution: Suppose we run Karger’s min cut algorithm we saw in class. Let $x_i$ be the event that the algorithm returns the $i^{th}$ global min cut. Suppose there are $s$ different
min cuts, then the probabilities of realizing each in the algorithm will be disjoint (all end with different sets of nodes at the conclusion of the algorithm). We saw in class that for a given global min cut, the probability of returning that cut is \( \geq \frac{2}{n(n-1)} \). So we have:

\[
P[\text{returning a global min cut}] = P[\bigcup_{i=1}^{s} x_i] = \sum_{i=1}^{s} P[x_i] \geq s \frac{2}{n(n-1)}
\]

We also have that the probability of returning a global min cut is \( \leq 1 \), so we need the above to be upper bounded by 1, which means \( s \leq \frac{n(n-1)}{2} \).

6. Exhibit a graph \( G = (V, E) \) where there are an exponential (in \( |V| = n \), the number of nodes) number of minimum cuts between a particular pair of vertices. Do this by constructing a family of graphs parameterized on \( n \) and give a pair of vertices \( s, t \) such that there are exponentially many minimum cuts between \( s \) and \( t \).

Solution. For \( n = 3 \), we simply have a path of length 2 between the two ends \( s \) and \( t \). For \( n > 3 \), each new vertex will be connected to \( s \) and \( t \) (and nothing else) creating an additional path of length 2 between \( s \) and \( t \).

For general \( n \), to separate \( s \) and \( t \), we must cut one edge along every edge-disjoint path between them. There are \( n - 2 \) paths between \( s \) and \( t \), each of length 2. So we have \( n - 2 \) binary choices, giving \( 2^{n-2} \) different minimum cuts.

7. A cycle cover \( C \) of a directed graph \( G(V, E) \) is a collection of vertex-disjoint directed cycles so that each vertex in \( V \) belongs to some cycle in \( C \). Give a polynomial time algorithm to compute a cycle cover of a given directed graph, or correctly report that one does not exist.

Solution. Let \( A = V \) and \( B = V \) and connect with directed edges from \( A \) to \( B \) if edge is on original graph. If there exists a perfect matching, in which the vertex \( v_i \) in \( A \) is matched to \( v_{\pi(i)} \) in \( B \). Here \( \{\pi(1), \ldots, \pi(n)\} \) is a permutation of \( \{1, 2, \ldots, n\} \). Then we see that the edges \( \{(v_i, v_{\pi(i)}) : i = 1, 2, \ldots, n\} \) forms a circle cover. Thus determining whether there exists a circle cover is equivalent to deciding whether there exists a perfect matching. We can use the network flow to solve this problem effectively.

8. Exhibit a directed graph that has cover time exponentially large in the number of nodes. Contrast this with the cover time of undirected graphs discussed in class.

Solution: Consider the digraph consisting of a directed cycle \( (1, 2, \ldots, n, 1) \), and edges \( (j, 1) \), from vertices \( j = 2, \ldots, n - 1 \). Starting from vertex 1, the expected time for a random walk to reach vertex \( n \) is \( 3 \times 2^{n-2} - 2 \); in fact, let \( T \) be the hitting time at state \( n \). \( E[T|x_0 = n] = 0, E[T|x_0 = 1] = E[T|x_0 = 2] + 1, E[T|x_0 = i] = (1/2)E[T|x_0 = 1] + (1/2)E[T|x_0 = i + 1] + 1 \) for \( i = 2, \ldots, n - 1 \). Solving this linear system, we get \( E[T|x_0 = 1] = 3 \times 2^{n-2} - 2, E[T|x_0 = i] = 3(2^{n-2} - 2^{i-2}) \) for \( i = 2, \ldots, n \).

9. A connected \( d \)-regular graph \( G \) is one in which all vertices have degree \( d \). Prove the cover time of a \( d \)-regular graph with \( n \) nodes is \( O(n^2 \log(n)) \).
**Solution:** For a $d$-regular graph, we know $m = \frac{nd}{2}$. From class, we know $C(G) \leq 2e^3 m R(G) \log(n) + n$. We know $R(G) = \max_{u,v} R_{u,v} \leq \text{diam}(G)$ because for any two nodes $u, v$, the max resistance will be less than or equal to the shortest path between these nodes in series. Then the diameter of $G$ is the longest shortest path between any two nodes, so it will upper bound the max resistance of the graph.

Now we show that the diameter of a $d$-regular graph is $O(n/d)$. To see this, consider every third node on a path realizing the diameter. Each such node cannot share any neighbors, because if they did then the diameter could be made shorter, contradicting the claim that the path realizes the diameter. The total number of nodes in the graph is $n$, and this provides an upper bound on the sum of all neighbors of every third node on the path that realizes the diameter. Let the diameter be $D$, then $(D/3)d \leq n$, giving $D = O(n/d)$. Now we have the following:

$$C(G) \leq 2e^3 m R(G) \log(n) + n$$
$$\leq 2e^3 \frac{nd}{2} \text{diam}(G) \log(n) + n$$
$$= e^3 ndO(n/d) \log(n) + n = O(n^2 \log(n))$$