1. Prove that a tree has at most one perfect matching.

    **Solution:** Consider doing by induction on the number of nodes of the tree. We have for \( n = 1 \), no perfect matchings exist and for \( n = 2 \), exactly one perfect matching exists. Now assume for all trees with \( \leq k \) nodes, at most one perfect matching exists.

    Now consider any tree on \( k + 1 \) nodes. There exists some leaf node \( l \), which in any perfect matching must be matched with its parent node (because that is the only edge incident to \( l \)). Consider deleting \( l \) and its parent and all incident edges to those nodes from the tree. We are left with a forest, in which every tree has \( \leq k - 1 \) nodes. We know by inductive hypothesis that each of these trees has at most one perfect matching, so the original tree has at most one perfect matching (the unique perfect matchings of each tree in the forest and the edge connecting \( l \) to its parent). By induction, we are done.

2. Consider a flow network \( G = (V, E) \) with source \( s \), sink \( t \) and non-negative edge capacities, \( w : E \to \mathbb{R}_{\geq 0} \). Suppose \( (S, S^c) \) and \( (T, T^c) \) are both minimum \( s - t \) cuts in \( G \). Show that both \( (S \cup T, (S \cup T)^c) \) and \( (S \cap T, (S \cap T)^c) \) are also minimum \( s - t \) cuts in the network.

    **Solution:** We know if \( (S, S^c) \) and \( (T, T^c) \) are both minimum cuts, then the sum of edges crossing \( S \) to \( S^c \) and the sum of those crossing \( T \) to \( T^c \) must be the same. We wish to show that the sum crossing \( S \cup T \) to its complement and \( S \cap T \) to its complement are the same as well.

    We will use the fact that the minimum cuts are the same as max flows. In particular, a cut is minimum if we consider a maximum flow and have all edges crossing the cut fully saturated (and all reverse edges with no flow). No consider the maximum flow in \( G \), and look at the edges crossing the set \( ST \); we know all edges will either be crossing set \( S \) or \( T \) and thus must be saturated in the maximum flow. Similarly, any edge in the reverse direction corresponds to a reverse direction edge in the cut defined by either \( S \) or \( T \) so must have no flow. Therefore, \( (S \cup T, (S \cup T)^c) \) must also be a minimum cut.

    We use similar reasoning for the cut \( (S \cap T, (S \cap T)^c) \).

3. **Kleinberg and Tardos 7.9** Network flow issues come up in dealing with natural disasters and other crises, since major unexpected events often require the movement and evacuation of large numbers of people in a short amount of time.
Consider the following scenario. Due to large-scale flooding in a region, paramedics have identified a set of \( n \) injured people distributed across the region who need to be rushed to hospitals. There are \( k \) hospitals in the region, and each of the \( n \) people needs to be brought to a hospital that is within a half-hour’s driving time of their current location (so different people will have different options for hospitals, depending on where they are right now).

At the same time, one doesn’t want to overload any one of the hospitals by sending too many patients its way. The paramedics are in touch by cell phone, and they want to collectively work out whether they can choose a hospital for each of the injured people in such a way that the load on the hospitals is balanced: Each hospital receives at most \( \lceil n/k \rceil \) people.

Reduce this problem to a maximum-flow problem, for which you know how to solve.

**Solution:** We model the problem as a max-flow problem. Let \( N \) be the set of people and \( K \) be the set of hospitals. We define the network as follows:

**Nodes:** \( K \cup N \cup s, t \) where \( s \) is the source and \( t \) is the sink.

**Edges:**

- There is an edge with capacity one from \( s \) to every node in \( N \).
- There is an edge with capacity \( \lceil n/k \rceil \) from every node in \( K \) to \( t \).
- There is an edge with capacity one from a node in \( n \in N \) to a node \( k \in K \) if \( n \) is in half-hour distance from \( k \).

If the max-flow in this graph has values \( n \) then every person can be assigned to a hospital. This can be done in polynomial time.

4. **Kleinberg and 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 \( \{p_1, \ldots, p_k\} \). 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 \( S_i \subset S \) of the people go to work. Then the above definition of the total driving obligation \( \Delta_j \) for \( p_j \) can be written as

\[
\Delta_j = \sum_{i: p_j \in S_i} \frac{1}{|S_i|}.
\]

Ideally, we would 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.
Reduce this problem to a max-flow problem.

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

**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.

$$f_{p_i q_j} = \frac{1}{|S_j|}, \quad f_{sp_i} = \sum_{j : p_i \in S_j} \frac{1}{|S_j|} \leq \lceil \Delta_i \rceil, \quad f_{q_j t} = 1.$$  

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

5. **Kleinberg and Tardos 7.12** You are given a flow network with unit capacity edges: it consists of a directed graph $G = (V, E)$, a source $s \in V$ and a sink $t \in V$; and $c_e = 1$ for every $e \in E$. You are also given a parameter $k$.

The goal is to delete $k$ edges so as to reduce the maximum $s - t$ flow in $G$ by as much as possible. In other words, you should find a set of edges $F \subseteq E$ so that $|F| = k$ and the maximum $s - t$ flow in $G' = (V, E \setminus F)$ is as small as possible. Give a polynomial time algorithm to solve this problem.

**Solution:** Consider the following algorithm.

(a) Run Ford-Fulkerson to find max-flow/min $s - t$ cut.

(b) While an edge remains in the min cut, delete.

(c) If no other edges remain, delete random edge.

First notice that each edge we delete from the $s - t$ minimum cut reduces the minimum cut value by exactly 1 (this is the maximum possible we could reduce the cut by deleting an edge). If the minimum cut has value at least $k$, we reduce it by exactly $k$. If the cut value is already less than $k$, we reduce it to 0 (the minimum possible value) and it doesn’t matter which other edges we delete.