# NetworkFlow

**1.**
- Don't know, we know that vertex cover without the word 'minimum' is in NP, but there's no clear way to show that a particular $k$ is minimum, how would you even give a proof for that which isn't brute forcing everything.
- Yes, this is in NP. We can actually just solve this problem in Polynomial time, as done in Q4 in Network Flow. So to verify that the answer is $k$, we just solve it using our network flow algorithm and check if we also get $k$.
- Don't know, there's no clear way to prove that 2 programs are the same itself.
- Don't know, even though a proof could be just 2 inputs where the programs give different outputs, verifying this will take exponential time, either because the inputs could be exponentially large, or the program could take exponential time itself.
- Don't know. If we have some $n$ 'for all"s, we could have $2^n$ different choices of variables for which we have to verify the formula as true, so a brute force proof will take exponential time. There is also no clear way to provide a shorter proof in general, for any boolean formula.

**2.** We claim that a set is a vertex cover if and only if its complement (all vertices outside our set) is an independent set. If $S$ is a vertex cover, for every edge $uv$, either $u \in S$ or $v \in S$. So there's no edge $uv$ where both $u \in S^c$ and $v \in S^c$. But this is just saying $S^c$ is an independent set. Now for the converse, if $S$ is independent, there is no edge $uv$ such that both $u \in S$ and $v \in S$. But this means for every edge, either $u \in S^c$ or $v \in S^c$, which is just saying $S^c$ is a vertex cover. So if we want to check if we have an independent set of size $k$, we just need to check if we have a vertex cover of size $n - k$. This is because if there is an independent set of size $k$, the complement set will have size $n - k$ which is a vertex cover. And if we don't have an independent set of size $k$, we can't have a vertex cover of size $n - k$, else by our equivalence we can find an independent set of size $n - (n - k) = k$.

**3.** The idea is like this, we keep the elements of our sets as edges, since if we want to cover all edges it becomes equivalent to covering all elements of our sets. So our universe is going to be the set of all edges. And selecting a vertex should be equivalent to selecting some set right, so each set will just be set of all edges incident on our vertex. Now we claim that we have a vertex cover of size $k$ if and only if we have a set cover of size $k$. But this is obvious right, we just select the corresponding sets we made for the vertex cover. Since every edge is incident with some vertex, every element is in some set. Similarly if we have a set cover of size $k$, we have a vertex cover of size $k$. Since any element is in some set, the corresponding edge is incident with some vertex.

**4.** For the load balancing problem, one machine is going to get some subset of $\{t_1, \dots, t_n\}$, and the other machine will get the rest of the elements. Let the subset of loads be $S$. The loads in each machine are $\sum_{i|t_i \in S} t_i$ and $\sum_{i|t_i \notin S} t_i = \sum_i t_i - \sum_{i|t_i \in S} t_i$. Both of these should be at most $T$, from which we get the following inequalitiies, $\sum_{i|t_i \in S} t_i \leq T$, and $\sum_{i|t_i \in S} t_i \geq \sum_i t_i - T$. The terms on the RHS of each inequality are constants no matter what $S$ is. So we can see that this is now very similar to the knapsack problem. So our motivation will be like this, we are going to make sure $W_1, W_2$ add to sum of all the weights, so that we can reduce to load balancing. So let's do that.

Let $w'$ be such that $\sum_i w_i + w' = W_1 + W_2$ i.e. $w' = W_1 + W_2 - \sum_i w_i$, we are adding this new weight to our list of weights, and are now going to do load balancing with $T = w_2$. If we have a solution for this, choose the set of loads which don't have $w'$, and call this set $S$. We know that $\sum_{i|w_i \in S} w_i \leq T = w_2$, now let's write the inequality for the other set.

$$\sum_{i|w_i \notin S} w_i + w' \leq T = W_2$$
$$\sum_i w_i - \sum_{i|w_i \in S} w_i + W_1 + W_2 - \sum_i w_i \leq W_2$$
$$W_1 \leq \sum_{i|w_i \in S}$$

So this subset satisfies the knapsack constraints. The converse is also easy to show i.e. if a subset satisfies the knapsack constraints, it's going to satisfy the load balancing constraints, by just going backwards. We already know the subset without $w'$ has a sum $\leq W_2$, now for the subset with $w'$.

$$\sum_{i|w_i \in S} w_i \geq W_1$$
$$\sum_{i|w_i \in S} w_i + \sum_{i|w_i \notin S} w_i + W_2 \geq W_1 + W_2 + \sum_{i|w_i \notin S}$$
$$\sum_i w_i + W_2 \geq W_1 + W_2 + \sum_{i|w_i \notin S}$$
$$W_2 \geq W_1 + W_2 - \sum_i w_i + \sum_{i|w_i \notin S} = w' + \sum_{i|w_i \notin S}$$

**5.** Each variable in our formula $\phi$ can be a variable in our linear equations. If $x_1, \ldots, x_n$ are the variables, we put the constraints $0 \leq x_i \leq 1$ to ensure each variable is 0 or 1. Now how do we write clauses? Take the clause $x_1 \wedge \neg x_2 \wedge x_4$. To satisfy this clause we need at least one of $x_1, \neg x_2, x_4$ to be 1. This can be actually represented as $x_1 + (1 - x_2) + x_4 \geq 1$. So we'll have $2n + k$ different linear inequalities.