(1) (36 pts. total)

Prove that the following decision problem is NP-complete.

TWO-COLORING WITH FAULTS
INSTANCE: An undirected graph G, an integer $k \geq 1$.
QUESTION: Can G be colored with 2 colors so that at most k edges have both nodes of the same color?

Answer: The problem belongs to NP because a coloring is smaller than the size of the input graph and whether the number of faults is $\leq k$ can be checked by iterating through the edges. To show completeness we define a mapping reduction f from 3SAT to it. Let any 3CNF formula

$$\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$$

be given. We define $f(\phi) = (G_\phi, k)$ where G_ϕ is in fact exactly the same graph used for the reduction to INDEPENDENT SET in lecture, but k is different: $k = m$ (however, see below).

That is to say, G_ϕ has $2n$ "rung vertices" $x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n$, with an edge connecting each pair $(x_i, \bar{x}_i)$. And for each clause C_j containing 3 literals, G_ϕ has 3 nodes connected in a triangle labeled with those literals. Finally, a literal x_i in a triangle has an edge going to its complement $\bar{x}_i$ in a rung, and $\bar{x}_i$ similarly has an edge to x_i in the rung. (There is nothing wrong with connecting to the same-sign vertex in the rung instead.) A single pass through ϕ sufficies to build G_ϕ in $O(m)$ time, so this is a polynomial-time reduction.

For *correctness*, first observe that every triangle must have at least one fault in any 2-coloring, so k is the minimum target value that can possibly be achieved. When is it achievable? For the rungs and corssing edges not to add to the faults, each rung $(x_i, \bar{x}_i)$ must have x_i green and $\bar{x}_i$ red or vice-versa. If x_i is green then consider the assignment to have $x_i = 1$, else $x_i = 0$. If such a 2-coloring exists, then at least one node in each clause must be green, since 3 reds means 3 faults, so the assignment satisfies each clause, so ϕ is satisfiable.

Going the other way, if an assignment satisfies ϕ, then it cannot color all theee literals red in any clause. But oops, it *could* color all three green, which would also cause 3 faults not 1 fault. How to cope with this?

The simplest way is to play "bait-and-switch" and say we are reducing from the problem NOT ALL EQUAL 3SAT instead. Then the correctness analysis is perfect. Another is to use the Cook-Levin proof as it was given: ϕ is assumed to be in the range of this proof, meaning its clauses come in triads $(u \vee w) \wedge (v \vee w) \wedge (\bar{u} \vee \bar{v} \vee \bar{w})$. Like in the proof for GRAPH 3-COLORING in the ALR notes and lecture, we add to G_ϕ a "Green Node" whose (arbitrary) color determines which color stands for "true." It is connected to the $\bar{x}_i$ part of each rung and to a third node r_j added to C_j of the form $(u \vee w)$ or $(v \vee w)$ to make it a triangle. That node must be "red," which prevents the three-green case from coming up. And it can't come up in a triangle for $(\bar{u} \vee \bar{v} \vee \bar{w})$ because having all of u, v, w be false makes the original ϕ unsatisfied. Finally, clauses of 1 literal, including (w_0) for the output wire, just "hang there" with one node and its crossing edge, nothing else, and cause no faults provided they are satisfied. So with this addendum, the reduction remains correct, except that "$k = m$" has to be re-interpreted as "$k =$" the number of clauses apart from those having just 1 literal.

(2) (36 pts. total)

Arora-Barak, chapter 2, exercise 2.17, both EXACTLY ONE 3SAT and SUBSET SUM. Needless to say, it is forbidden to look these up on the Internet.

Answer: Clearly both problems are in NP. To reduce 3SAT to EXACTLY ONE 3SAT, consider a general clause

$$C = (\ell_1 \vee \ell_2 \vee \ell_3)$$

Here ℓ_i stands for a *literal* which could be a negated variable; it is convenient to write this way rather than refer to variables directly. The simplest answer I know translates every C to the clause group

$$C' = (\bar{\ell}_1 \vee a \vee b) \wedge (\bar{\ell}_2 \vee c \vee d) \wedge (\bar{\ell}_3 \vee e \vee f) \wedge (a \vee c \vee e).$$

If all three literals are false, then their negations are true in the first three clauses, so we have to make a, b, c, d, e, f all false, which prevents $(a \vee c \vee e)$ from being satisfied. But otherwise, some literal is true, say ℓ_1 without loss of generality. Since $\bar{\ell}_1$ is false, we take a from the first clause true and set c and e both false. Regardless of the truth values of the other two literals, we can freely set d and f so as to make each clause exactly-one-satisfied.

Gives a 3CNF formula ϕ, it is easy to create an equivalent instance ϕ' of EXACTLY ONE 3SAT by replacing each clause C_j of ϕ by the corresponding group C'_j. Hence EXACTLY ONE 3SAT is NP-complete. (There is a footnote below about variations on this reduction.)

For the second part, we need to reduce EXACTLY ONE 3SAT to SUBSET SUM. The idea is that we give every clause C_j a numerical value $t_j > 0$ and set the target $T = \sum_{j=1}^{m} t_j$ (except see modification below). Every time C_j is satisfied by a literal ℓ_i (which could be x_i or $\bar{x}_i$ depending on which one appears in C_j), we add t_j to our total. So we will get T *if* every clause is satisfied exactly once.

If C_j is satisfied twice-over then we get "*ka-ching!*" $+t_j$ added twice—which we actually wish to avoid. So we want to space out the values t_j so that $2t_j$ or $3t_j$ can never work as part of a sum adding to T. Taking $t_j = 2^{2(j-1)}$ suffices, since $t_1 = 1$ and $t_{j+1} = 4t_j$ for $j > 1$, so an increment of $2t_j$ or $3t_j$ (which causes a '1' in an even-numbered bit place) can never be wiped out by other terms. The value T has 1s in every odd-numbered place in binary notation, e.g. for $m = 4$ clauses we get $1010101 = 85$. Thus we can get T *only if* every clause is satisfied exactly once.

We have proved much about the reduction before even giving the construction of the set U of numbers involved. U has one number a_i for each x_i and b_i for $\bar{x}_i$, giving $2n$ numbers in all. The value a_i equals the sum of t_j over all j such that C_j has x_i, and similarly b_i over all clauses that have $\bar{x}_i$. An assignment that satisfies each clause exactly once then gives rise to a subset A of size n that picks exactly one of a_i, b_i for each i and sums to T. The last detail is how to enforce the "exactly one of a_i, b_i" part. The answer is to use t_j also for $j = m + 1$ to $j = m + n$. To both a_i and b_i we add t_{m+i}, and we re-define T to be $\sum_{j=1}^{m+n} t_j$. Now by the same logic as why the subset-sum fails if any clause is satisfied twice or thrice over, we must use exactly one of a_i and b_i. Calculating the numbers a_i and b_i requires just one pass through the formula, so this is a polynomial-time reduction and correctly reduces EXACTLY ONE 3SAT to SUBSET SUM. Hence SUBSET SUM is NP-complete.

Footnote on exactly-once satisfiability: There are some interesting variations of the above answer, which we illustrate first in the case of a clause $C = (\ell_1 \vee \ell_2)$ of two not three literals. Each literal could be either a plus or negated variable—it won't matter and it's convenient to write them this way. First consider replacing C by the single 3-clause

$$C' = (\bar{\ell}_1 \vee \bar{\ell}_2 \vee z),$$

where z is a new variable. If both ℓ_1 and ℓ_2 are true then we can exactly-one-satisfy this with $z = 1$; if one of them is true then we use $z = 0$. If both ℓ_1 and ℓ_2 are false then C' has two true literals and so cannot be exactly-one-satisfied, but that is exactly what we want since that's the case were C is false.

There's an alternate way to do this with no additional negations:

$$C' = (\ell_1 \vee x) \wedge (\ell_2 \vee y) \wedge (x \vee y \vee z).$$

An assignment that makes both ℓ_1 and ℓ_2 true can be handled by making $x = 0$, $y = 0$, and $z = 1$. An assignment that makes ℓ_1 true but not ℓ_2 is handled by $y = 1$ and $x = z = 0$, and one that makes ℓ_2 true but ℓ_1 false allows $x = 1$ and $y = z = 0$. Finally, an assignment that makes both ℓ_1 and ℓ_2 false forces $x = y = 1$, but then the $(x \vee y \vee z)$ clause in C' cannot be uniquely satisfied. We can *pad* the first two clauses in C' to length exactly 3 by inserting a where we separately have clauses $(a \vee b \vee c)$ and $(a \vee \bar{b} \vee \bar{c})$ which force $a = 0$ for exact-one satisfiability.

Here is a third way that involves negation and leads into the logic of how you could come up with the above-given answer for a 3-literal clause:

$$C' = (\bar{\ell}_1 \vee a \vee b) \wedge (\bar{\ell}_2 \vee c \vee d) \wedge (a \vee c).$$

Incidentally, if we wished b and d to be the same variable, we could enforce this by adding the clause $(b \vee \bar{d})$, which is exactly-one satisfied if and only if b and d have the same value. So there is no loss of generality in beginning with all different new variables in the first two clauses. Now if both ℓ_1 and ℓ_2 are false then all of a, b, c, d have to be false, which prevents $(a \vee c)$ from being satisfied. If both are true then we can pick $a, d = 1$ and $b, c = 0$. If ℓ_1 is true and ℓ_2 is false then pick only $a = 1$, and if vice-versa then make $c = 1$. So this works. The 3-literal answer followed a similar pattern with the extra variables e and f.

Here are two further questions:

- Can a 3-clause C be handled entirely without negation, as we did for a 2-clause? Perhaps if we allow a 4-clause in the resulting group C'?
- Can we simplify matters if we consider ϕ to come from the range of the Cook-Levin proof for NAND (or for NOR)?

Recall that in the Cook-Levin proof the only 3-literal clauses are ones with all-negated variables, and those three literals *cannot all be satisfied*. So we can pretend that the assignment $\ell_1 = \ell_2 = \ell_3$ cannot happen—or rather, that it is correct to cause a situation in the C' we want to build where it *can't* be "exactly-one" satisfied. Put another way, is it easier to reduce "Not-All-Equal 3SAT" to Exactly-One 3SAT? Well, this is more an esthetic question than a problem.