Problem B: Get a Clue!

Shortest judge solution: 1871 bytes. Shortest team solution (during contest): 1841 bytes.
Python solutions by the judges: both Pypy and CPython

This is a problem that can be solved by a brute-force search, but the implementation can be a bit messy, and depending on your exact approach, it may be important to have good constant factors.

The very simplest approach is just to generate all disjoint sets of cards for the four hands (well, one hand is already given so there's only one option for that) and then go through all the played rounds and check that they are consistent with assignment of cards to hands. However, this will probably not run in time unless it is fairly carefully implemented, so something a bit better is needed.

The next approach could be to first guess the murderer, weapon, and room, and then do the above-mentioned brute-force search for a partition of cards into hands, and break as soon as a valid solution is found. This turns out to run a bit faster and is definitely possible to make fast enough (because the worst case for the above is when pretty much all partitions of cards into hands yields a valid solution, and many of those partitions into hands will result in the same murderer/weapon/room).

A different algorithmic approach which essentially makes constant factor worries go away is to generate the possible hands separately: for player 2 we compute all possible subsets of 5 cards $S_1, S_2, \ldots$ that are compatible with all the played rounds, and similarly for player 3 all sets of 4 cards $T_1, T_2, \ldots$ and for player 4 all subsets of 5 cards $U_1, U_2, \ldots$. Now for a given guess of murderer/weapon/room, let X be the set of remaining cards after removing the three answer cards and the hand of player 1. We are then trying to find i, j, k such that $S_i \cup T_j \cup U_k = X$. This can easily be done in time $O(\#S \cdot \#T)$ – simply try all S_i's and T_j's and then check if $X \Delta S_i \Delta T_j$ is one of the U_k's (by keeping a dictionary of all U_k's). The subsets are most conveniently represented by integers, which makes lookups quick. This solution is fast enough that it can even be done in CPython.

Problem C: Mission Improbable

Shortest judge solution: 818 bytes. Shortest team solution (during contest): 1178 bytes.
Python solutions by the judges: both Pypy and CPython

If not for the top-down view, this problem would be really straightforward. Since solving easy problems is easier than solving harder problems, let's go over that first.

Consider the highest stack of crates in the input, and assume it has height H_1. It has to appear both in the front view and the side view at least once. Assume that it appears m_1 times in the front view, and n_1 times in the side view, and without loss of generality, assume $n_1 \leq m_1$. In that case, we will need at least m_1 stacks of height H_1, and we can achieve the correct side and front view by arranging m_1 stacks so that there is at least one in each of the m_1 columns and n_1 rows.

Then, proceed to the next height, H_2. Again, we have m_2 columns and n_2 rows that have to contain a stack with height H_2. The one thing that is different here is that it is possible for,