November JDCC Solutions + Comments

Problem A (Lifting Weights):

Solution: A+B*(N-1) (She increases the amount she can lift by B, doing so N-1 times)

Time Complexity: O(1)
Space Complexity: O(1)

Problem B (Icy Spiral):

Solution: Every time you turn twice, you're left with a rink of size (H-1, W-1) to clean, and if either H or W is 1, then there are no more turns left to make, hence the answer is 2*(min(H, W) – 1).

Time Complexity: O(1)
Space Complexity: O(1)

Problem C (eLettery):

Solution: Two compare two names, find how many A's, B's, C's etc. are in both names, then the amount of shared A's in the two names is min(A's in first, A's in second). The same applies to all the other letters, and adding them together gives the total number of shared letters. To calculate the best match, as you read in input, see if it's better than your current best. If it is, keep it, otherwise discard it.

Time Complexity: O(*NL*) where *L* is the average length of a word.
Space Complexity: O(*L*)

Problem D (Lucky Tickets):

Solution: This is an ad-hoc search problem. You split the input string into the left and right halves, sum up the left half, and then call your recursive method on the sum and right half. The recursive method starts on the left side of the string and tries to find the next lucky number. First, it tries using the same digit as the one in the number. If that doesn't work, it increments the digit by at least one (you can use math to figure out the minimum it must increment by), then fills the rest of the string with the smallest number that sums to the target (usually a string of zeroes, then a digit, then a string of nines).

If the method can't find a solution (target is too large or too small), it returns false. If no next lucky number is found, increment the left side by one and find the smallest number that adds up to the new left sum to use as your right side.

Time Complexity: O(log *N)*
Space Complexity: O(log *N*)

Problem E (Randomize):

Solution: Essentially, you have to realize that if you find a cycle, then every number that was a part of that cycle will have the same cycle length. It is also possible to have non-circular cycles where the function repeats on a number that isn't the seed (e.g. 1->2->3->2), in which case the cycle length of the 1 will be slightly different.

To solve this problem, iterate through the possible SEEDs. If the cycle length was already found, move on, otherwise brute force one cycle, making sure to store what numbers where hit on the way. Then, iterate through all the numbers in the cycle and set their cycle lengths. If at any point your cycle hits a previously calculated cycle length, then it can stop and use that calculated value as the length of the rest of the cycle instead of continuing with the cycle. Once you iterate through the SEEDs, you average the cycle lengths and the answer pops out.

Note: This problem is a combination of graph theory (cycles in directed graphs) and dynamic programming (storing previously calculated cycle lengths).

Time Complexity: O(*P)*
Space Complexity: O(*P)*