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 the Zamboni turns twice, it's left with a rink of size (H-1, W-1) to clean. Note that if either H or W is 1, then there are no more turns left to make, hence the total number of turns it makes is 2*(min(H, W) – 1).

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

Problem C (eLettery):

Solution: To compare two names, first 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 the value for every letter together gives the total number of shared letters. To calculate the best match, as the input is read in see if the current name is better than the current best. If it is, make it the new best, otherwise discard it. Once all the names are processed, output the best.

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

Problem D (Lucky Tickets):

Solution: This is an ad-hoc search problem. First, split the input string into the left and right halves and sum up the left half, and then call the recursive method on the sum and right half to find the next lucky number. The recursive method first tries using the same digit as the one in the original right half, recursing on the rest of the string. If that doesn't work, it increments the current digit by at least one (math can be used to figure out the minimum it must increment by), then calls another method that 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 a solution isn't found using this method, then there's an overflow. In that case, add one to the left half, and then call that method used for the recursion that fills the right half with the smallest number that sums to the target.

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

Problem E (Randomize):

Solution: The key to solving this problem is realizing that if one cycle is found, 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 initial part (in this case 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 were visited on the way. Once a number is visited for the second time, iterate through all the numbers in the cycle and set their cycle lengths. If at any point the search hits a previously calculated cycle length, then it can stop and use that calculated value as the length of the rest of the cycle. After iterating through the SEEDs, average the cycle lengths.

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)*