COP 4516 Spring 2022 Week 15 Final Team Contest Solution Sketches

Problem A: How Many Pieces of Candy?

The key observation here is that you are ***forced*** to buy candy at each store. So, at a minimum, you must get $\mathbf{a_i}$ pieces from store i. Let X equal the sum of the $\mathbf{a_i}$'s. The only choice you have each store is whether or not to upgrade to the large pack and get $\mathbf{b_i}$ candies instead. This upgrade can be seen as adding $\mathbf{b_i - a_i}$ candies to the base starting point of X candies. In some sense, for each store i, we can choose to either add $\mathbf{b_i - a_i}$ candies, or not to. This is exactly the subset sum problem. Thus, this problem maps to the following: given the set of values $\mathbf{b_i - a_i}$, for each store, calculate the number of unique sums of subsets that can be obtained.

To solve the problem, run the subset sum or knapsack DP algorithm (subset sum is the knapsack problem where the weights are equal to the values) and look at the number of unique sums for which there exist subsets.

Problem B: The King's Escape

The method of solving this problem is clear: breadth first search. The difficulty is in the implementation. With 5 pieces with 5 different rules of movement, many may try writing five separate functions/pieces of code to mark illegal squares that are blocked by the black pieces. Though each individual piece is straight-forward, it's likely some bug will be introduced into the code if separate code is written. Complicating the matter is blocked pieces. If one black piece, B, gets in the way of another, A, then "future" squares A would block are not blocked.

To ease implementation, DX/DY arrays are recommended. In marking blocked squares, start at a black piece in question and loop through each of its directions of movement. For each direction of movement, step out 0 steps, 1 step, 2 steps, etc. Keep going until either (a) Your piece can only move once so you stop after 1 step, (b) You hit another piece, (c) You go out of bounds.

Once the blocked squares are marked, then just run a straight BFS on the adjusted grid to the top row.

Problem C: Flowers

This is the easiest problem in the set. For each team, calculate the number of packs of flowers needed (ceiling(problems/packsize)), taking care to use integer division appropriately. Then multiply the number of packs by the price and this is the cost for each team. Add up all the team costs and voila!

Problem D: Fractorial

The small version can be solved easily by calculating all the factorials to 19 and then trying to divide each of them in order by a^b. As long as longs are used, no overflow will occur using this straight-forward process since each answer is guaranteed to be less than 20.

For the large version, we have to recall that the number of times that a prime p divides evenly into n! is $\sum_{i=1}^{\infty} \left\lfloor \frac{n}{p^i} \right\rfloor$. Thus, the answer is greater than or equal to $\lfloor n/p \rfloor$. Unfortunately, we don't know n in this problem. Notice that the structure of the problem is as follows: it's easy if we know n, but unclear how to solve for n given p and its exponent. One final note is that as n increases, the exponent of p increases. These are all the hallmarks of binary search (a function that moves in one direction and is easy to calculate one way, but the inverse is hard). Thus, the idea is to binary search the value of n for each prime p in the prime factorization of a^b.

Step 1: prime factorize a. The exponents of the prime factorization of a^b are just b times the exponents in the prime factorization of a.

Step 2: For each prime p in the prime factorization of a^b, let k be its exponent. Binary search the minimum value of n such that n! is divisible by p^k. Then, just take the maximum of each of the bounds over all unique primes p in the prime factorization of a^b.

Problem E: Frogger

This problem is easier than it could be because all of the cars move horizontally. Thus, it's fairly easy to calculate the time when Kermit ***might*** intersect a car. If Kermit's velocity is v and his angle of movement is θ, then he moves $v\sin\theta$ in one unit of time. It follows that if he's going to intersect a car with y value of y, it will happen at time $t = y/(v\sin\theta)$. His x-coordinate at this time is $x = t(v\cos\theta)$. Finally, we just want to know ***at what time*** the front of the car passes this point and the back of the car passes this point. We can calculate this by looking at the ***signed*** distance the car must travel to get to the point from its starting point and dividing by its velocity. If the time Kermit gets there is in between, there is a collision between him at this car. If not, there isn't a collision. Kermit makes it if he doesn't collide with ANY car.

Problem F: Trilots

For the small data, running the permutation algorithm on n = 9 points suffices. Try each permutation. For each permutation group together the first three points, the next three points and the last three points and add up twice the triangle areas of each (via cross product magnitude calculation).

For the large data, we must realize that the permutation solution redundantly tries the same set of triangles many times. For n = 15, it suffices to try each set of triangles once. We can do this by picking all 3 points of our triangle at once and forcing ourselves to always choose the smallest unchosen point and pairing it with any two other unchosen points. This amounts to a double for loop for selecting the last 2 points for each triangle. For each unique set, calculate the desired sum of twice the area and output the minimum value.