COP 4516 Spring 2026 Week 8 Team Final Contest Solution Sketches

Problem A: We've got the Beat

This is meant to be one of the easiest questions in the contest. Read in the whole string, and start processing character by character. We can think of each note as the number of eighth notes it represents, so that we only need to use integers for our computations. Assign each note as follows: E = 1, Q = 2, H = 4 and W = 8. When reading through the string, just keep a counter of the sum of the notes processed. When that counter hits 8, output those letters on a line and continue until you get through the whole string. Substring methods make this code very easy to write.

Problem B: Bird Feeders

The structure of this problem allows us to use a Binary Index Tree (BIT) to perform updates to individual values and query ranges efficiently. For each 'A' operation, simply perform a point update to add the given value x at the given position p. For 'S' operations, however, we will need to be a bit more careful (pun intended). We will first find the amount v stored in the bird feeder at the given position p, which can be done with a range sum query over the range $[p, p]$. The value we will use for this point update is the maximum of $-x$ and $-v$, which ensures that the remaining amount in the bird feeder at position p is non-negative.

For 'Q' queries, we are given a target average value a and a range $[l, r]$ that needs to have this average or better. Recall that the average of a list of values is the sum of the values divided by the number of values in the list. We can extend this idea to see that the sum of values in the range $[l, r]$ needs to be at least a times the number of values in this range, which is $r-l+1$ since the endpoints are inclusive. We will perform a range query to find the total amount of birdseed stored in the bird feeders over the range $[l, r]$, which we will denote as s, and compare this value against $a*(r-l+1)$. Taking the difference $a*(r-l+1) - s$ will give us the amount that needs to be added to this range to achieve an average of a. Our answer will be 0 for this query if we already have a sum that is sufficient for the given average - that is, if $s \geq a*(r-l+1)$.

Problem C: Airport Charging Stations

We can observe that it's optimal to only charge at a single charging station. With this in mind, we can iterate through each station and find the maximum power level we can achieve using only that station. Since it takes x_i minutes to reach the i^{th} station and x_i minutes to return to the gate, we need to allot $2*x_i$ minutes for traveling between the gate and a charging station. Then, we can use the remaining amount of time to charge, which is $t - 2*x_i$, and we gain p_i power levels for each minute spent charging, which totals to $p_i*(t - 2*x_i)$ power levels. Taking the maximum of this value over all charging stations will give us the maximum power level that is attainable within t minutes. Note that if no charging stations are reachable within t minutes (i.e. $2*x_i > t$ for all charging stations), then our answer is 0.

Problem D: Freezing!

This problem is the easiest in the set! Just create a variable to store either 0 or 32 based on the temperature scale, then check to see if the given temperature is above, below or equal to this limit!

Problem E: Company Merger

We need quick access to the two smallest and two largest elements in our data structure, allowing for repeated items. We can handle this with a sorted set of objects, using the original index of the items to distinguish between companies worth the same value. Then we just repeatedly remove either the two smallest or two largest values and insert their sum into the sorted set.

Problem F: Perplexing Puzzle

This problem can be solved with a relatively straightforward recursive brute force. For each tile, try each value between 0 and 3 (inclusive), adding this value to a running sum and recursing to the next tile. Once the last tile's value has been selected, stop recursing and check that the sum of the chosen values is equal to the target sum. If so, add 1 to the answer, since this is a valid orientation of tiles. This approach finds the answer by testing all 4^10 combinations of tiles, which runs fast enough.

Problem G: Jumping Robot

This may be the most difficult problem on the set, but we can solve this problem using DP. Let dp[i][j] represent the probability that the robot is located at position i after making exactly j jumps. We can define our base case as follows: dp[0][0] = 1 and dp[i][0] = 0 for $i > 0$, since the robot always starts at position 0 before making any jumps.

For each number of jumps j from 0 to k-1, iterate through each position (a safe range would be to use positions 0 to x+p inclusive) and calculate the location probabilities after taking this jump.

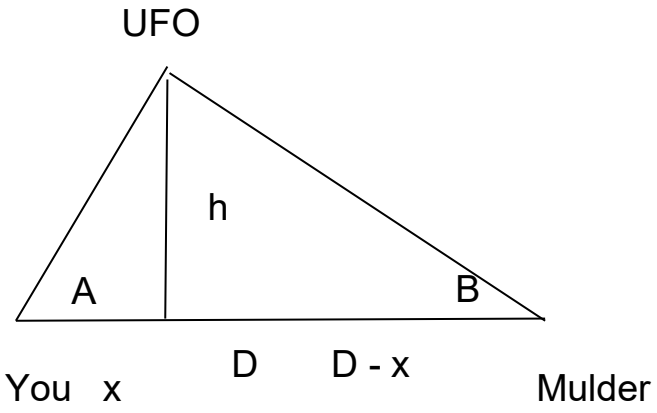
If the robot is already at position x, the robot no longer moves and we add dp[i][j] to dp[i][j+1]. Otherwise, we need to determine which direction the robot will jump in: if $i < x$, the robot travels right (adding to i), and if $i > x$, the robot travels left (subtracting from i). We will then iterate through all sizes of jumps m from 1 to p and add dp[i][j] / p (the probability that we are at position i and the next jump is of size m) to dp[i+m][j+1] if moving right, or to dp[i-m][j+1] if moving left.

Note that if the robot would move to a negative position (i.e. the robot jumps to the left from position i such that $m > i$), we will instead add the value of dp[i][j] / p to dp[0][j+1].

After filling in the DP table completely, the answer is stored in dp[x][k], which represents the probability that the robot is located at position x after exactly k jumps. The runtime of this approach is O(k*x*p), which is sufficiently fast for the input bounds.

Problem H: UFO Sighting

Let's take a look at our drawing:



This has a couple of added values compared to the problem specification picture. We draw in a perpendicular with length h (the height from the ground of the UFO). We label the distance from You to the point at the base of the triangle where the perpendicular with length h hits as x. This sets up the equations:

$$\tan A = \frac{h}{x}$$

$$\tan B = \frac{h}{D - x}$$

Solve both equations for h to yield:

$$xtanA = (D - x)tanB$$

Add xtanB to both sides and then continue to solve for x:

$$xtanA + xtanB = DtanB$$

$$x(tanA + tanB) = DtanB$$

$$x = \frac{DtanB}{tanA + tanB}$$

Now that we have x, we can then go and solve for h: $h = xtanA = \frac{DtanAtanB}{tanA+tanB}$.

Note: There are other ways to solve this as well. The judge solutions cover this method (in Python) and a different method in both the Java and C++ solutions.