Kick Start 2020 - Round G

Analysis: Combination Lock

We can see that if we decide a final value at which all wheels should be in the end, moves for each wheel to reach that value are independent of the moves performed on other wheels. Thus, we can calculate number of moves for each wheel separately. Consider a wheel i which is at value x currently. We want the wheel to finally reach the value y. There are 2 cases here:

- Case 1: $x \leq y$. Increasing the value of the wheel i would take y-x steps. Decreasing the value of wheel i would take **N** - y + x steps. Hence, the minimum number of moves in this case would be minimum of y-x and **N** - y + x.
- Case 2: $x > y$. Decreasing the value of the wheel i would take x-y steps. Increasing the value of wheel i would take **N** + y - x steps. Hence, the minimum number of moves in this case would be minimum of x-y and **N** + y - x.

Test Set 1

We can solve this test set by trying all possible **N** values that all the wheels should have in the end. For each value, we calculate the total number of moves to get all of the wheels to this value using Case 1 and Case 2. The answer is the minimum possible moves performed over all such X. There are **N** possible values and for each value, we perform O(**W**) operations. Thus, the time complexity of the solution would be O(**W** × **N**).

Test Set 2

A major observation is that we could always get the minimum possible moves by finally bringing all the wheels to one of the initial values of the given wheels. We can prove this by considering a value Val which is not among the initial values of wheels and then showing that we can get a same or better answer by moving all the wheels to one of the initial values. Consider the initial values of the wheels in the sorted order. Let the index of the wheel with smallest initial value greater than Val be j, if no such value exists, then we can consider j = 1 as it is first wheel next to value Val in cyclic order. Let the index of wheel with largest initial value smaller than Val be i, if no such value exists, then we can consider i = **W** as it is the first wheel before value Val in cyclic order. Now to reach the value Val, each value will either reach $\mathbf{X_i}$ or $\mathbf{X_j}$. Now say there are y wheels at value $\mathbf{X_i}$ currently which leaves us with **W** - y wheels at value $\mathbf{X_j}$. The number of moves for all wheels to reach value Val would be $y \times (Val - \mathbf{X_i}) + (\mathbf{W} - y) \times (\mathbf{X_j} - Val)$. We can get the same or better result than this. There are 2 possibilities:

- $y \leq \mathbf{W} - y$. If we choose to bring all the wheels to value $\mathbf{X_j}$, we will have number of moves as $y \times (Val - \mathbf{X_i}) + y \times (\mathbf{X_j} - Val)$, which is never large than the number of moves required for all wheels to reach value Val. Hence, we have a better answer if we bring all the wheels to value $\mathbf{X_j}$.
- $y \geq \mathbf{W} - y$. If we choose to bring all the wheels to value $\mathbf{X_i}$, we will have number of moves as $(\mathbf{W} - y) \times (Val - \mathbf{X_i}) + (\mathbf{W} - y) \times (\mathbf{X_j} - Val)$, which is never large than the number of moves required for all wheels to reach value Val. Hence, we have a better answer if we bring all the wheels to value $\mathbf{X_i}$.

Now instead of trying all possible values from 1 to **N**, we would try the initial values of the wheels and calculate moves required for all wheels to reach that value. For each value, we take O(**W**) time to calculate the moves required for all wheels to reach that value. There are **W** values. Hence the complexity is $O(\mathbf{W}^2)$.

Test Set 3

We have already proved that we only need to consider one of the initial values of the wheels. We need to calculate the number of moves required for all wheels to reach a particular value efficiently. We can do the following. Sort the initial values of the wheels. Maintain a prefix sum array Pre. Pre[i] denotes the sum of initial values of wheels from 1 to i. We define a method GetSum(i,j) which gives the sum of values between indexes i and j. This can be calulated in O(1) using Pre array.

Suppose that currently we are calulating the number of moves required for all wheels to reach $\mathbf{X_i}$. Consider any wheel j from index 1 to i. Minimum number of moves required for wheel j to reach value $\mathbf{X_i}$ would be minimum of $\mathbf{X_i} - \mathbf{X_j}$ and $\mathbf{N} - \mathbf{X_i} + \mathbf{X_j}$. Consider 2 indexes k and l such that $1 \le k < l \le i$. We can prove that it is not possible to have $\mathbf{X_i} - \mathbf{X_k} < \mathbf{N} - \mathbf{X_i} + \mathbf{X_k}$ and $\mathbf{N} - \mathbf{X_i} + \mathbf{X_l} < \mathbf{X_i} - \mathbf{X_l}$ simultaneously. This is because if we add the two inequalities, we get $\mathbf{N} - \mathbf{X_k} + \mathbf{X_l} < \mathbf{N} + \mathbf{X_k} - \mathbf{X_l}$ which implies $\mathbf{X_l} < \mathbf{X_k}$. But it is not possible to have such condition because we know that $\mathbf{X_k} \le \mathbf{X_l}$. Thus, we can say that there exists an index p such that $1 \le p \le i$ and for each wheel q such that $p \le q \le i$ will have minimum number of moves as $\mathbf{X_i} - \mathbf{X_q}$ and for each wheel r such that $1 \le r \le p-1$ will have minimum number of moves as $\mathbf{N} - \mathbf{X_i} + \mathbf{X_r}$. We can find this index p using binary search on wheels 1 to i. Now we need to find sum of $\mathbf{X_i} - \mathbf{X_q}$ for each q. This can be calculated in O(1) by $(i - p + 1) \times \mathbf{X_i}$ - GetSum(p,i). Now we need to find sum of $\mathbf{N} - \mathbf{X_i} + \mathbf{X_r}$ for each r. This can be calculated in O(1) by $(p-1) \times (\mathbf{N} - \mathbf{X_i})$ + GetSum(1,p-1). We have calculated number of moves required for wheels 1 to i to reach value $\mathbf{X_i}$.

Similarly, we can say that there exists an index b such that $i \le b \le \mathbf{W}$ and for each wheel c such that $i \le c \le b$ will have the minimum number of moves as $\mathbf{X_c} - \mathbf{X_i}$ and for each wheel d such that $b+1 \le d \le \mathbf{W}$ will have the minimum number of moves as $\mathbf{N} - \mathbf{X_d} + \mathbf{X_i}$. We can find this index b by using binary search on wheels 1 to **W**. Now we need to find sum of $\mathbf{X_c} - \mathbf{X_i}$ for each c. This can be calculated in O(1) by GetSum(i+1,c) - $(c-i) \times \mathbf{X_i}$. Now we need to find sum of $\mathbf{N} - \mathbf{X_d} + \mathbf{X_i}$ for each d. This can be calculated in O(1) by $(\mathbf{W} - b) \times (\mathbf{N} + \mathbf{X_i})$ - GetSum(b+1,**W**).

We have calculated the number of moves required for all wheels to reach a particular value in O(log **W**). Now we need to take the minimum over all such values. Thus we can calculate moves for all the initial values of all wheels in O($\mathbf{W} \times \log \mathbf{W}$) time complexity. Note that instead of using binary search to find indexes p and b, we can use two pointers approach to find them. We can prove that indexes p and b will keep on increasing as we increment i. The overall complexity of the solution remains same due to the sorting part.