Kick Start 2019 - Round C

Analysis: Wiggle Walk

Test set 1 (Visible)

Approaching the problem naively, one might try to simply simulate what has been mentioned in the problem statement i.e. keep on moving the robot in the specified direction till it reaches an un-visited square. This approach is going to have a time complexity of $O(\mathbf{N}^2)$, which although good enough for test set 1.

Test set 2 (Hidden)

The problem with the above approach is that it visits a lot of already visited squares which in worst case will be all the previously visited squares (Consider the input where you are given alternating `E` and `W` throughout). If we somehow get to the destination square for each instruction faster, we might be able to reduce the complexity.

Let's say the robot is in some row r and received an instruction `W`. Now, all the already visited squares (if any) it will pass before reaching an unvisited square have to form a contiguous interval in row r. This suggests that we may use intervals to represent all the visited squares in the same row.

With that in mind, consider we have a set of intervals for each row and each column of the grid to represent which cells have been visited in that particular row or column, let's call them interval-sets. Initially, all these sets are empty except for the set corresponding to row $\mathbf{S_R}$, which has a single interval ($\mathbf{S_C}$, $\mathbf{S_C}$), and the set corresponding to the column $\mathbf{S_C}$, which has a single interval ($\mathbf{S_R}$, $\mathbf{S_R}$).

Now, using this data-structure, let's try to find the destination square for the robot. Let's say it's in square (r, c) and got an instruction `W`. For this, first we search in the interval-set corresponding to row r. We will try to find which interval in this set contains c (there must definitelty be one!). Once we find it, we immediately know what's going to be the new position for the robot! It's apparent that the same method works for all other directions as well.

All that remains now is to find a way to update our data-structure suitably to also include the newly visited square. This can be done in a very standard manner by simply finding the adjacent intervals for that square in both, the corresponding column interval-set and the corresponding row interval-set and then updating them either by extending one of the intervals or merging them or adding a new 1 length interval.

Since we add at most one interval in each case, the number of intervals is $O(\mathbf{N})$. Since all operations are about finding/inserting/removing a single interval, all of those can be handled easily in $O(\log(\mathbf{N}))$ time. So the over all run time of this approach is $O(\mathbf{N}\log(\mathbf{N}))$. There is also a $O(\mathbf{N})$ solution to this problem using hash tables. It is left as an exercise to the reader.