Kick Start 2022 - Coding Practice with Kick Start Session #3

Analysis: Wiggle Walk

View problem and solution walkthrough video

Test set 1

A naive approach to this problem is to simply simulate what has been mentioned in the problem statement. In other words, for each instruction, we will keep moving the robot in the specified direction until it reaches an unvisited square. This approach is going to have a time complexity of $O(\mathbf{N}^2)$, which is good enough for test set 1.

Test set 2

For this test set, the naive approach will not be good enough. The problem with the naive approach is that it visits a lot of already visited squares. In cases such as when the input alternates continuously between `E` and `W`, each instruction will visit all the previously visited squares. We need a way to compute the destination square for each instruction faster to reduce the complexity.

Let us use intervals to represent all the contiguous segments of visited squares in the same row. For example, if in a row r the robot has already visited squares $(r, 3)$, $(r, 5)$, $(r, 6)$, and $(r, 7)$, the visited squares in this row can be represented with the intervals $(3, 3)$ and $(5, 7)$.

With that in mind, consider we have a set of intervals for each row and each column of the grid to represent all the squares that have been visited in that particular row or column. Let us call them interval-sets. Initially, the robot has only visited the starting squares ($\mathbf{S_R}$, $\mathbf{S_C}$) it started in, so 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 ($\mathbf{S_R}$, $\mathbf{S_R}$).

Using this data structure, we can quickly find the destination square for the robot for each instruction. Let the robot be located at (r, c). For each instruction, we will look at the interval-set corresponding to the row r (if the instruction is `W` or `E`) or column c (if the instruction is `N` or `S`). Then, we will find the interval in this set that contains the robot's current position (there is guaranteed be one, because the square the robot is currently in is already considered to be visited). Once we find that interval, we will immediately know the location of the next unvisited squares in either direction, as they will be first row or column that lies outside of that interval.

For example, let the robot be located in square $(2, 6)$ and the next instruction is `W`. Let the interval set for row 2 be $(3, 3)$ and $(5, 7)$. The robot is currently located at column 6, which is inside the interval $(5, 7)$. From this, we know that the next unvisited square in the west is $(2, 4)$, and the next unvisited square in the east is $(2, 8)$. As the instruction is `W`, the robot will move to square $(2, 4)$.

All that remains now is to find a way to update our interval-sets suitably whenever the robot visits a new square. Let (r, c) be the newly visited square. This can be done by updating the interval-set for both row r and column c. For both interval-sets, we will check if any existing intervals are next to the newly visited square:

- If there are two intervals adjacent to the newly visited square on both sides, we will merge these two intervals into one.
- If there is one interval adjacent to the newly visited square, we will extend this interval by 1 to include the newly visited square.
- Otherwise, we will add a new interval of length 1 containing just the newly visited square.

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 overall 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.