Kick Start 2022 - Round D

Analysis: Touchbar Typing

Test Set 1

When there is no duplicate key in the keyboard, there is only one way to type any given string. It would suffice to simulate typing the given string and count the cost, for test set 1.

We can build a mapping from each character in the keyboard to its position in the keyboard. Let us denote the position of i-th character of $\mathbf{S}$ on the keyboard as p_i.

Let us denote the current position of the finger on the keyboard with y. We should start with the finger on the keyboard on p_1 position.

Then in each step, we need to add $|y - p_i|$ to the answer, and update the current keyboard position to p_i.

The time complexity of creating the mapping from each character to a position on the keyboard is $O(\mathbf{M})$. With help of the mapping previously mentioned, the time complexity of calculating the answer is $O(\mathbf{N})$. The total time complexity of this approach is $O(\mathbf{N} + \mathbf{M})$, which is sufficient for test set 1.

Test Set 2

In test set 2, keys on the keyboard are not unique. At each step of typing, there might be multiple possible positions on the keyboards to move to next, and making optimal choice at every step will lead to the minimum number of moves.

Let us denote the set of positions of i-th character of $\mathbf{S}$ on the keyboard as P_i.

We can consider a dynamic programming approach, where for each position x on the given string, and each position y on the keyboard, we need to find the optimal solution based on possible options for that instance. Then we have the recurrent function F,
$F(x, y) = min(F(x - 1, j) + |(y - j)|), j \in P_{x-1}$.

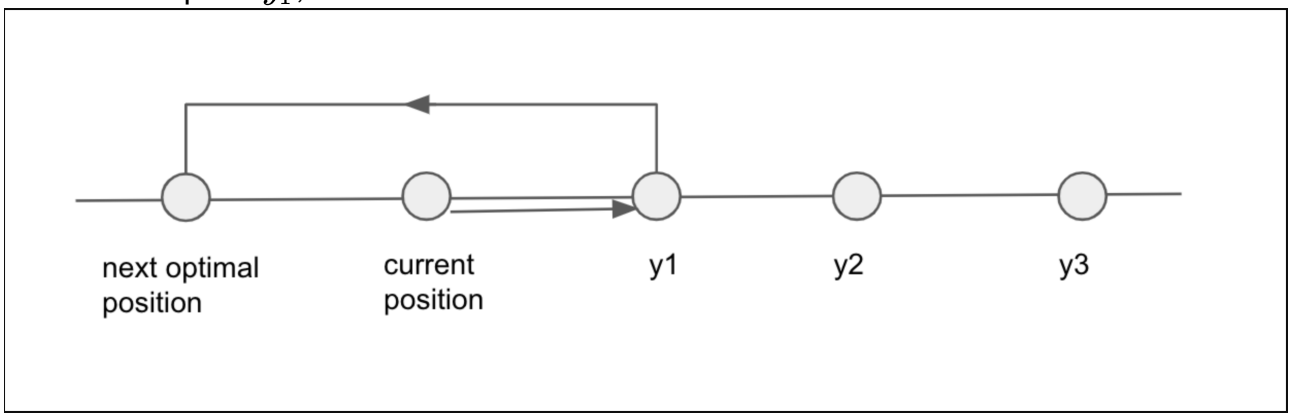
This approach gives us a time complexity of $O(\mathbf{N} \times \mathbf{M} \times \mathbf{M})$, as there are total $\mathbf{N} \times \mathbf{M}$ possible states, and on each state, there are at most $\mathbf{M}$ options in the worst case. This is sufficient for test set 2.

Test Set 3

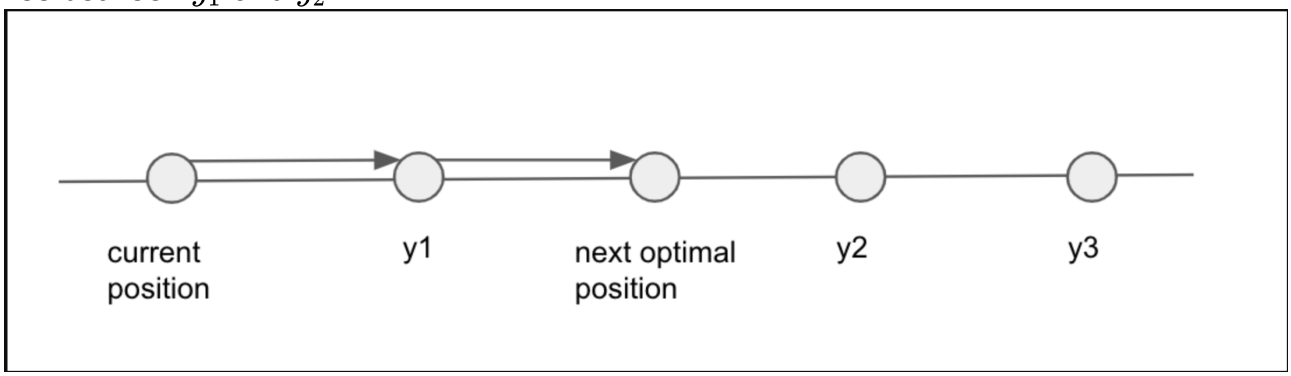
For test set 3, $O(\mathbf{N} \times \mathbf{M} \times \mathbf{M})$ is not sufficient. We need to make the choices more carefully to reduce the complexity. Try to think a little on how to optimise this step before looking into the solution :)

One interesting observation is that every time when there are multiple possible keyboard positions to jump to type the next character, it is always optimal to pick either the one with the minimal distance on the left direction, or the one with the minimal distance on the right direction. Let us consider the current position on the keyboard is x, and there are multiple possibilities on the right, denoted by y_1, y_2, y_3 and so on, sorted by in the order of increasing distance from x. Now, jumping to y_2 or y_3 cannot be a better choice than jumping to y_1. Let us consider two cases,

1. The next optimal move lies in the opposite direction, aka left. Then it is better to jump to the closest option y_1, to minimise the total distance.



2. The next optimal move lies in the same direction, aka right. Then jumping to y_1 is at least not worse than jumping to y_2, y_3. It can even be a better option, if the next optimal position lies between y_1 and y_2.



The same reasoning is applicable in case of making a move to the left. Then it basically boils down to making a choice of whether to move right, or left on the keyboard from any current position. The recurrent function then changes to F,
$F(x, y) = min((F(x-1, j_{left}) + |y - j_{left}|), (F(x-1, j_{right}) + |y - j_{right}|))$ where $j_{left}, j_{right} \in P_{x-1}$ and j_{left} and j_{right} are the closest two positions from x on the left and the right respectively.

This makes the number of choices $O(1)$, giving us a total time complexity of $O(\mathbf{N} \times \mathbf{M})$, which is sufficient for test set 3.

If you would like to explore more tasks like Glide Type on the Crowdsource app and directly train Googleâ€™s artificial intelligence systems to make Google products work equally well for *everyone, everywhere*, you can download it here.