1974. Minimum Time to Type Word Using Special Typewriter

Easy Greedy String

Leetcode Link

Problem Description

The problem describes a special typewriter with the lowercase English alphabet arranged in a circle, starting at 'a' and ending with 'z'. The pointer of the typewriter starts at 'a' and can only type the character it is pointing at. To type a given string, you can either move the pointer clockwise or counterclockwise around the circle to reach the desired character or you can type the character the pointer is currently on. Each of these operations takes one second. The task is to find the minimum amount of time required to type a given word using this typewriter.

Intuition

To minimize the time taken to type a word, we should take the shortest path to reach each character from the current position of the pointer. When we want to move from one character to another, we have two possible paths:

1. Moving clockwise
2. Moving counterclockwise

The shortest path is the one which has the least number of characters between the current and the target characters, considering the circular arrangement of the typewriter.

The intuition behind the solution is:

- Calculate the difference in the positions of the current and the desired characters.
- Determine whether it is quicker to reach the next character by moving clockwise or counterclockwise.
- Add this minimum number of moves to an accumulator that keeps track of the total seconds needed.
- Since typing the character also takes one second, we add one second for every character typed.

Thus the total time taken for typing the word will be the sum of the shortest number of moves to each character plus the number of characters in the word.

Solution Approach

To implement the solution, we employ a straightforward approach without the need for complex data structures or algorithms. The solution makes use of the Python `ord()` function, which returns the ASCII value of a character, to easily calculate the position of letters on the typewriter.

Here's a step-by-step breakdown of the Solution code:

1. Initialize `ans` as 0 to keep track of the total number of seconds taken, and `prev` as 0 to store the position of the previously typed character ('a', at the beginning, represented by 0).
2. Iterate over each character `c` in the given word `word`.
   - Calculate `curr` which is the position of the character `c` by subtracting the ASCII value of 'a' from the ASCII value of `c`. This effectively translates the character to a numerical position in the range 0-25 on the circle.
3. Calculate the absolute difference `t` between `curr` (the target character's position) and `prev` (the current character's position). This represents the distance if we move in one direction without considering the circular nature.
4. Since the typewriter is circular, we also calculate the distance if we were to go the other way around the circle, which is `26 - t`. Pick the minimum of these two values, which is the shortest path to the next character.
5. Update `ans` by adding the time taken to move to the next character (`t`) plus one second to account for the time taken to type the character (since every operation takes one second).
6. Set `prev` to `curr` as we have now moved to the next character.
7. After processing all characters in `word`, return `ans`, which now contains the minimum time to type out the given word.

This approach ensures that we always take the shortest path to the next character, thereby minimizing the overall time taken to type the entire word. It leverages the circular nature of the alphabet arrangement by always considering the direct and reverse paths and choosing the quicker one.

Example Walkthrough

Let's take the word "bza" as an example to illustrate the solution approach.

1. We start with `ans = 0` and `prev = 0` since the typewriter pointer starts at the character 'a'.
2. The first character we need to type is 'b'.
   - We calculate the position of 'b' by subtracting the ASCII value of 'a' from that of 'b' (`ord('b') - ord('a') = 1 - 0 = 1`). So `curr = 1`.
   - The absolute difference `t` between `curr` and `prev` is `1 - 0 = 1`.
   - The other way round the circle would be `26 - 1 = 25`. We take the minimum of 1 and 25 which is `1`.
   - We add this to `ans` along with 1 second for typing: `ans = 0 + 1 + 1 = 2`.
   - We update `prev` to `curr`. So, `prev = 1`.
3. Now, the next character is 'z'.
   - Its position is `ord('z') - ord('a') = 25`.
   - The difference `t` is `abs(25 - 1) = 24`.
   - Going the other way around, `26 - 24 = 2`. We pick the minimum which is `2`.
   - Update `ans` to `2 + 2 + 1 = 5`.
   - Update `prev` to `curr` (`prev = 25`).
4. Finally, we need to type 'a', going back to the start.
   - The position of 'a' is `ord('a') - ord('a') = 0`.
   - The difference `t` is `abs(0 - 25) = 25`.
   - Going the other way around, `26 - 25 = 1`, which is the shorter path.
   - Update `ans` to `5 + 1 + 1 = 7`.
   - Since it's the last character, we don't need to update `prev`.

After walking through the steps, the minimum time required for typing "bza" with our special typewriter is 7 seconds. We moved from 'a' to 'b' in 2 seconds, 'b' to 'z' in 3 seconds, and 'z' to 'a' in 2 seconds. Each move includes the time taken to type the character.

Python Solution

```python
class Solution:
    def minTimeToType(self, word: str) -> int:
        # Initialize the time to type the word to 0 and the initial pointer position to 'a' (which is 0 in 0-25 index)
        total_time = pointer_position = 0

        # Iterate over each character in the word
        for char in word:
            # Find the position of the current character (0-25 index)
            current_position = ord(char) - ord('a')

            # Calculate the distance between the current and previous character
            distance = abs(pointer_position - current_position)

            # The minimum rotations needed is either the direct distance or the wrap around distance
            # Wrap around distance can be calculated by subtracting direct distance from total characters (26)
            distance = min(distance, 26 - distance)

            # Add the distance to type the character, plus 1 second for the typing action
            total_time += distance + 1

            # Update pointer position to the current character's position
            pointer_position = current_position

        # Return the total time to type the word
        return total_time
```

Java Solution

```java
class Solution {
    // Method to calculate the minimum time to type a word given the initial pointer position is at 'a'
    public int minTimeToType(String word) {
        int totalTime = 0; // Total time to type the word
        int prevPosition = 0; // Starting position is 'a', so the initial index is 0

        // Iterate through each character in the word
        for (char ch : word.toCharArray()) {
            int currentPosition = ch - 'a'; // Convert the character to an index (0-25)

            // Calculate the clockwise and counterclockwise distance between the current and previous character
            int clockwiseDistance = Math.abs(currentPosition - prevPosition);
            int counterClockwiseDistance = 26 - clockwiseDistance;

            // Take the minimum distance of the two possible ways to type the character
            int minDistanceToType = Math.min(clockwiseDistance, counterClockwiseDistance);

            // Add the distance to type the character + 1 second for typing the character itself
            totalTime += minDistanceToType + 1;

            // Update the previous position to the current character's position for the next iteration
            prevPosition = currentPosition;
        }

        // Return the total time to type the word
        return totalTime;
    }
}
```

C++ Solution

```cpp
class Solution {
public:
    // Function to calculate the minimum time to type a word on a circular keyboard
    int minTimeToType(string word) {
        // Initialize the answer with 0 time
        int totalTime = 0;
        // Starting at 'a', represented as 0
        int previousPosition = 0;
        // Iterate over each character in the word
        for (char& currentChar : word) {
            // Find the numeric position of the current character, 'a' being 0
            int currentPosition = currentChar - 'a';
            // Calculate the direct distance between previous and current position
            int directDistance = abs(previousPosition - currentPosition);
            // Calculate the circular distance (26 letters in the alphabet)
            int circularDistance = 26 - directDistance;
            // Take the minimum of the direct and circular distances
            int travelTime = min(directDistance, circularDistance);
            // Add travel time for this character plus 1 second for typing the character
            totalTime += travelTime + 1;
            // Set the previous position to the current character's position for the next iteration
            previousPosition = currentPosition;
        }
        // Return the total time taken to type the word
        return totalTime;
    }
};
```

Typescript Solution

```typescript
// Function to calculate the minimum time to type a word on a circular keyboard
function minTimeToType(word: string): number {
    // Initialize the total time with 0 time
    let totalTime: number = 0;
    // Starting at 'a', represented as 0
    let previousPosition: number = 0;

    // Iterate over each character in the word
    for (const currentChar of word) {
        // Find the numeric position of the current character, 'a' being 0
        let currentPosition: number = currentChar.charCodeAt(0) - 'a'.charCodeAt(0);
        // Calculate the direct distance between previous and current position
        let directDistance: number = Math.abs(previousPosition - currentPosition);
        // Calculate the circular distance (26 letters in the alphabet)
        let circularDistance: number = 26 - directDistance;
        // Take the minimum of the direct and circular distances
        let travelTime: number = Math.min(directDistance, circularDistance);
        // Add travel time for this character plus 1 second for typing the character
        totalTime += travelTime + 1;
        // Set the previous position to the current character's position for the next iteration
        previousPosition = currentPosition;
    }

    // Return the total time taken to type the word
    return totalTime;
}
```

Time and Space Complexity

The time complexity of the given code is `O(n)`, where `n` is the length of the input string `word`. This is because the code iterates through each character in the string exactly once regardless of its content.

The space complexity of the code is `O(1)`. This is due to the fact that the amount of extra space used by the algorithm does not depend on the length of the input string and is limited to a few variables that store integers, which include `ans`, `prev`, `curr`, and `t`.