925. Long Pressed Name

Two Pointers String Easy

Problem Description

The problem presents a scenario where your friend is typing his name on a keyboard, but some characters might get typed more than once due to the key getting long-pressed. Your task is to determine if the typed string of characters could represent the actual name of your friend, even with some characters potentially being repeated due to long presses. In other words, you need to verify if the typed string is a valid representation of the actual name with the allowance for extra characters that are the result of

long presses. For example, if your friend's name is "alex" and the typed string is "aaleex", the function should return True because the extra "a"

and "e" could result from long pressing those keys. However, if the typed string is "aaleexa", the function should return False because the 'a' at the end of the typed string cannot be accounted for by a long press when typing "alex".

The intuition behind the solution is to traverse both the name and the typed strings and compare them character by character. We

return False.

Intuition

start by initializing two pointers, one for each string. As we progress through both strings: • If the current characters don't match, it's clear that typed does not match name, and we return False. • If the characters match, we then need to count the subsequent occurrences of that character in both strings to ensure that the typed string

- does not contain fewer repetitions of the character than the name string (which would be invalid).
- To implement this, we count the number of times the current character appears consecutively in both the name and typed strings. If the count in name is greater than in typed, then the typed string cannot be the result of long pressing while typing name, and we

If we complete the traversal without encountering any discrepancies, we then make sure that both pointers have reached the ends of their respective strings. This final check ensures that there aren't any extra characters in the typed string that don't

match with the name. If both pointers are at the end, we return True; otherwise, we return False. The approach makes use of a two-pointer technique to compare the strings efficiently, checking character by character to verify the validity of the long-pressed string.

Solution Approach The provided solution implements a two-pointer technique. Two pointers, i and j, are used to iterate through the name and typed

strings respectively. The approach utilizes the fact that for typed to be a result of long-pressing the keys while typing name, every

character in name must appear in typed in the same order and there must be at least the same number of each character in typed

as there is in name.

characters than required.

Here's the step-by-step breakdown of the algorithm: 1. Initialize two variables, i and j, to 0. These will serve as pointers to iterate through name and typed. 2. Loop through the strings while i < m and j < n, where m is the length of name and n is the length of typed. This will ensure that we are comparing the characters within the bounds of both strings. 3. If at any point, the characters at the current pointers name[i] and typed[j] do not match, we return False as this immediately disqualifies the

typed string from being a valid representation of name caused by long pressing.

- 4. If the characters match, we then count the consecutive appearances of the current character c = name[i] in both strings. This is done by
- looping while the next character is the same as c and incrementing cnt1 or cnt2 for name and typed respectively. 5. After counting the appearances, if cnt1 (the count for name) is greater than cnt2 (the count for typed), we return False because typed has fewer
- 6. Both pointers are then moved to the next character (i + 1 and j + 1), and steps 3-5 are repeated until one of the strings is fully traversed. 7. Finally, we check if both i and j have reached the end of their respective strings (i == m and j == n). If they have, this means typed could indeed be a long-pressed version of name, so we return True. If not, then typed contains additional characters not found in name, and we return
- False. No additional data structures are used in this approach. All that is needed are a few variables to keep track of positions within the
- strings and the counts of the consecutive characters. The solution's correctness relies on the ordered comparison of characters and the counts of consecutive occurrences, which align with the rules of how arrays (strings) are constructed and the problem's constraints regarding long presses.

Let's use a small example to illustrate the solution approach. Assume your friend's name is "sara" and the typed string is "ssaarraa". We'll walk through the algorithm to determine if "ssaarraa" could be a long-pressed version of "sara". 1. Initialize two pointers i and j to 0. 2. As long as i < len(name) and j < len(typed), proceed to compare the characters.

• j = 0, typed[j] = 's'

In name:

In typed:

• i = 0, name[i] = 's'

At the beginning:

Example Walkthrough

3. Both characters match, so we start counting consecutive characters in both strings.

• i = 0 to i = 1, the character changes from 's' to 'a', so cnt1 for 's' in name is 1.

4. Since cnt1 <= cnt2, we proceed.

• j = 2, typed[j] = 'a'

We repeat the counting:

After completing the traversal:

Solution Implementation

Python

Now:

5. Move both pointers to the next set of characters and repeat steps 2-4.

• j = 0 to j = 1, 's' is repeated, and at j = 2, it changes to 'a', so cnt2 for 's' in typed is 2.

• i = 1, name[i] = 'a'

• i moves from 1 to 2, encountering 'r', so cnt1 for 'a' is 1 in name.

• We reach i = 4 (i == len(name)), indicating we've checked all characters in name.

def isLongPressedName(self, name: str, typed: str) -> bool:

Lengths of the input strings

Initialize pointers for name and typed

Loop through both strings simultaneously

name_length = len(name)

typed_length = len(typed)

name_index = typed_index = 0

return False

count_name = count_typed = 0

name_index += 1

count_name += 1

nameIndex++;

C++

public:

class Solution {

typedIndex++;

// If we have reached the end of both strings, the name is correctly typed

for (; nameIndex < nameLength && typedIndex < typedLength; ++nameIndex, ++typedIndex) {</pre>

return nameIndex == nameLength && typedIndex == typedLength;

int nameLength = name.size(), typedLength = typed.size();

bool isLongPressedName(string name, string typed) {

// Iterate through each character of both strings

// If the characters don't match, return false

if (name[nameIndex] != typed[typedIndex]) return false;

int nameIndex = 0, typedIndex = 0;

current_char = name[name_index]

Count consecutive characters in name

Count consecutive characters in typed

• j moves from 2 to 4, with 'a' at indices 2 to 3 before we find 'r', so cnt2 for 'a' is 2 in typed.

Again, cnt1 <= cnt2, so we proceed. Continue this process for each character in name.

have reached the end of their respective strings, the function will return True. "ssaarraa" is a valid long-pressed version of "sara".

Count occurrences of the current character in both strings

• Similarly, we reach j = 8 (j == len(typed)), indicating all characters in typed have been accounted for.

6. Since we have successfully gone through both strings without finding a mismatch or insufficient count of characters in typed, and both pointers

class Solution:

while name_index + 1 < name_length and name[name_index + 1] == current_char:</pre>

while typed_index + 1 < typed_length and typed[typed_index + 1] == current_char:</pre>

while name_index < name_length and typed_index < typed_length:</pre> # If characters at current position do not match, return False if name[name_index] != typed[typed_index]:

```
typed_index += 1
                count_typed += 1
            # If name has more consecutive characters than typed, return False
            if count_name > count_typed:
                return False
            # Move to the next character
            name_index += 1
            typed_index += 1
       # Check if both strings have been fully traversed
        return name_index == name_length and typed_index == typed_length
Java
class Solution {
    public boolean isLongPressedName(String name, String typed) {
        int nameLength = name.length();
        int typedLength = typed.length();
        int nameIndex = 0, typedIndex = 0;
       // Iterate over each character in both strings
       while (nameIndex < nameLength && typedIndex < typedLength) {</pre>
           // If the current characters don't match, return false
            if (name.charAt(nameIndex) != typed.charAt(typedIndex)) {
                return false;
            // Count consecutive characters in the original name
            int nameCharCount = 0;
            char currentChar = name.charAt(nameIndex);
            while (nameIndex + 1 < nameLength && name.charAt(nameIndex + 1) == currentChar) {</pre>
                nameIndex++;
                nameCharCount++;
            // Count consecutive characters in the typed string
            int typedCharCount = 0;
            while (typedIndex + 1 < typedLength && typed.charAt(typedIndex + 1) == currentChar) {
                typedIndex++;
                typedCharCount++;
           // If the original name has more consecutive characters than the typed one, return false
            if (nameCharCount > typedCharCount) {
                return false;
            // Move to the next character
```

```
int nameCharCount = 0, typedCharCount = 0; // Counters for the occurrences of the current character
            char currentChar = name[nameIndex];
            // Count consecutive occurrences in `name`
            while (nameIndex + 1 < nameLength && name[nameIndex + 1] == currentChar) {</pre>
                ++nameIndex;
                ++nameCharCount;
            // Count consecutive occurrences in `typed`
            while (typedIndex + 1 < typedLength && typed[typedIndex + 1] == currentChar) {</pre>
                ++typedIndex;
                ++typedCharCount;
            // If `name` has more consecutive characters than `typed`, the typing is not long-pressed
            if (nameCharCount > typedCharCount) return false;
       // Check that we have iterated through both strings completely
        return nameIndex == nameLength && typedIndex == typedLength;
};
TypeScript
function isLongPressedName(name: string, typed: string): boolean {
    let nameLength = name.length, typedLength = typed.length;
    let nameIndex = 0, typedIndex = 0;
    // Iterate through each character of 'name' and 'typed' simultaneously
    for (; nameIndex < nameLength && typedIndex < typedLength; nameIndex++, typedIndex++) {</pre>
       // If the characters at the current position do not match, it's not long-pressed
        if (name[nameIndex] !== typed[typedIndex]) {
            return false;
        let nameCharCount = 0, typedCharCount = 0; // Counters for occurrences of the current character
```

```
let currentChar = name[nameIndex];
          // Count the consecutive occurrences of the current character in 'name'
          while (nameIndex + 1 < nameLength && name[nameIndex + 1] === currentChar) {</pre>
              nameIndex++;
              nameCharCount++;
          // Count the consecutive occurrences of the current character in 'typed'
          while (typedIndex + 1 < typedLength && typed[typedIndex + 1] === currentChar) {</pre>
              typedIndex++;
              typedCharCount++;
          // If 'name' has more consecutive characters than 'typed', it's not long-pressed
          if (nameCharCount > typedCharCount) {
              return false;
      // Check that both strings have been fully iterated through
      return nameIndex === nameLength && typedIndex === typedLength;
  // You can call isLongPressedName with actual parameters like so:
  // const result = isLongPressedName("alex", "aalleex"); // This should return true
class Solution:
   def isLongPressedName(self, name: str, typed: str) -> bool:
       # Lengths of the input strings
        name_length = len(name)
        typed_length = len(typed)
       # Initialize pointers for name and typed
        name_index = typed_index = 0
        # Loop through both strings simultaneously
        while name_index < name_length and typed_index < typed_length:</pre>
            # If characters at current position do not match, return False
            if name[name_index] != typed[typed_index]:
                return False
            # Count occurrences of the current character in both strings
            count_name = count_typed = 0
            current_char = name[name_index]
            # Count consecutive characters in name
            while name_index + 1 < name_length and name[name_index + 1] == current_char:</pre>
                name_index += 1
                count_name += 1
            # Count consecutive characters in typed
            while typed_index + 1 < typed_length and typed[typed_index + 1] == current_char:</pre>
                typed_index += 1
                count_typed += 1
            # If name has more consecutive characters than typed, return False
            if count_name > count_typed:
                return False
            # Move to the next character
            name_index += 1
            typed_index += 1
```

The time complexity of the given code is O(n), where n is the length of the typed string. This is because the two pointers i and j, which iterate over name and typed strings, can only move forward and each character in both strings will be visited at most once. The space complexity of the code is 0(1) because there are a fixed number of variables used and their space requirement does

not scale with the input size. No additional data structures or dynamic memory allocation is used in this code that would make

the space complexity scale with input size.

Time and Space Complexity

Check if both strings have been fully traversed

return name_index == name_length and typed_index == typed_length