443. String Compression

Approach #1: Read and Write Heads [Accepted]

Intuition

We will use separate pointers `read` and `write` to mark where we are reading and writing from. Both operations will be done left to right alternately: we will read a contiguous group of characters, then write the compressed version to the array. At the end, the position of the `write` head will be the length of the answer that was written.

Algorithm

Let's maintain `anchor`, the start position of the contiguous group of characters we are currently reading.

Now, let's read from left to right. We know that we must be at the end of the block when we are at the last character, or when the next character is different from the current character.

When we are at the end of a group, we will write the result of that group down using our `write` head. `chars[anchor]` will be the correct character, and the length (if greater than 1) will be `read - anchor + 1`. We will write the digits of that number to the array.

Python

```
class Solution(object):
    def compress(self, chars):
        anchor = write = 0
        for read, c in enumerate(chars):
            if read + 1 == len(chars) or chars[read + 1] != c:
                chars[write] = chars[anchor]
                write += 1
                if read > anchor:
                    for digit in str(read - anchor + 1):
                        chars[write] = digit
                        write += 1
                anchor = read + 1
        return write
```

Java

```
class Solution {
    public int compress(char[] chars) {
        int anchor = 0, write = 0;
        for (int read = 0; read < chars.length; read++) {
            if (read + 1 == chars.length || chars[read + 1] != chars[read]) {
                chars[write++] = chars[anchor];
                if (read > anchor) {
                    for (char c: ("" + (read - anchor + 1)).toCharArray()) {
                        chars[write++] = c;
                    }
                }
                anchor = read + 1;
            }
        }
        return write;
    }
}
```

Complexity Analysis

- Time Complexity: $O(N)$, where N is the length of `chars`.
- Space Complexity: $O(1)$, the space used by `read`, `write`, and `anchor`.

Analysis written by: @awice (https://leetcode.com/awice).

Comments: 7

Sort By

Type comment here... (Markdown is supported)

Preview Post

monil1511 (/monil1511) ★ 7 August 21, 2018 11:09 AM

in the worst case the internal for loop and the string will take O(N) space. This is not constant time solution.

0 Share Reply

trizen (/trizen) ★ 0 July 30, 2018 3:33 AM

Also possible to use global substitution with a regular expression (semi-cheating):

```
def compress(chars)

    a = chars.join.gsub(/(.)\1+/) {|k|
```

Read More

0 Share Reply

spanlabs (/spanlabs) ★ 39 July 11, 2018 10:23 PM

Convert `read - anchor + 1` to string uses extra space O(lgN).

Further more, this problem is somehow worthless, The compressed strings are ambiguous. For example, ["1", "2", "2"]. Algorithms should have practical value. One star for this question.

4 Share Reply

SHOW 2 REPLIES

chaomiaoshuati (/chaomiaoshuati) ★ 3 February 15, 2018 5:13 PM

@awice

The line here is confusing at first.

```
if (read > anchor) {
```

Read More

1 Share Reply

vegito2002@gmail.com (/vegito2002gmailcom) ★ 526 January 8, 2018 9:58 PM

alternative implementation (https://discuss.leetcode.com/topic/108897/simple-easy-to-understand-java-solution/2)

0 Share Reply