# Inc Match Two

It is clear from the problem statement that we can spend multiple operations on the same character, and "increase" it to another letter until it either matches a neighboring character (and both get deleted) or becomes the letter '`z`'.

A key observation is that, at any given time during the play, there can be no adjacent characters that are identical. This implies two things:

1. As long as the string length is greater than one, we can choose any character and either increase it to match one of its neighbors or increase a neighbor to match itself.
2. When an operation results in a match, characters at opposite sides that are equal also get matched in pairs and are automatically erased.

However, if the neighbors on both sides of a character are identical and greater than itself, then increasing it will match *three* instead of two, and the length of the string will change parity. We shall denote such sequence as a *valley*.

Fortunately, if the length is even, there will always be at least one character that is not surrounded on both sides by the same letter and so can be increased to match a single neighbor. It follows that we can always empty the string if its length is *even*.

**Odd case**

If the length is odd, we need to change its parity by removing a valley. But checking for the presence of a valley is not sufficient, due to the clamping effect of letter '`z`'.

Let's first suppose the alphabet is infinite. Under this assumption, the only case where the answer is negative is when the input has no valleys. Equivalently, we can say that the largest element in the string is unique and is the only peak or half-peak. Furthermore, its prefix is sorted in ascending order, while the suffix is sorted in descending order.

On the other hand, if the alphabet is finite, there will be an additional case where the answer is negative. It happens when (one of) the greatest element(s) in the string satisfies all of the following conditions:

- it is the last letter of the alphabet
- it is located at an even position
- its prefix is sorted in descending order
- its suffix is sorted in ascending order

These conditions imply that there can be at most three elements with value '`z`', of which the middle one is the culprit. The whole sequence can be thought of as a large valley whose bottom element was reflected on the top. One such example is "`zbazabz`".

The second condition is necessary as it preempts the possibility of forming a removable valley starting or ending at the affected position. (Imagine cutting the string at $s_{i\pm2}$ and replacing this character by '`z`'.)

Note that the longest strings meeting the above criteria would be composed of $25+1+25 = 51$ characters, which explains the problem constraints.

**Solution**

The actual implementation is quite simple: checking whether the conditions for a negative answer can be met. This includes checking the parity of the string length, finding the maximum element, and checking if the prefix and suffix are ordered as described above. The time complexity is $O(n)$.