Inc Match Two (Easy)

Problem category:	Strings
Expected difficulty:	1800

Solution

From the problem statement, we know that we can spend multiple operations on the same character to "increase" it to another letter, until it either matches a neighboring character or becomes the letter 'z'.

A key observation to solve this problem is that, at any given time during the game, there can be no adjacent characters that are equal. This implies two important 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 it.
2. When an operation results in a match, characters at opposite sides that are equal also get matched in pairs and are automatically erased.

But there's a catch: if the neighbors on both sides of a character are the same letter, then increasing it will match *three* instead of two, and the length of the string will change parity. Fortunately, if the length is even, there will always be a character that is not surrounded by the same letter and can be increased to match a single neighbor.

It follows that we can always make the string empty if its length is *even*.

The case of odd length is more complicated. To change the parity of the length, we need to check whether it is possible to remove a contiguous sequence of odd length.

The simplest way to do so is to find a *valley* such that we can make its extremities match (before the middle characters can be raised to match them). This can be accomplished with a linear search over the string: as soon as we find a valley of odd length, we check if either extremity can reach the other.

Henceforth, we shall call l and r the indices of the left and right extremities of such a sequence, respectively. Let's denote as *prefix* the elements $a_1, ..., a_{l-1}$ and as *suffix* the elements $a_{r+1}, ..., a_n$.

Let's investigate the prefix first. If its length is even, we can play the game until it is completely erased, in which case we can increase a_l to 'z'. Otherwise, we play the game until a single letter remains. If the latter is less than a_l, we can increase a_l to 'z'; else we can only increase it to 'y' (if it was less than that).

So if we have the choice, we should select a letter that is less than a_l as our remaining letter from the prefix. But when is that possible? As it turns out — although we won't give a proof here —, if there is such a letter in the prefix, then either it can be selected or we would have found a good sequence earlier in our search.

What about the suffix? The reasoning is similar: if there is a letter that is less than a_r in the suffix, then either it is possible to select it or we would find a good sequence later in our search. In both cases, we can use a prefix minimum array to verify its existence efficiently.

Let's call $f(l, r)$ a function that compares the current and maximum values of a_l and a_r and returns true if the sequence $a_l, ..., a_r$ can be erased. Then the algorithm is as follows:

1. Compute prefix and suffix minimum arrays from the input string.
2. For each valley in the string:
   - Extend it as far as possible, to use the maximum letter at both sides.
   - If its length is odd, the answer is either $f(l, r)$ or $f(l + 1, r - 1)$ (if possible).
   - Otherwise, the answer is either $f(l + 1, r)$ or $f(l, r - 1)$ (if possible).

Complexity

Since we make a single pass over the string, the time complexity is $O(n)$.