note: the problem actually asks for the number of “matching” pairs of substrings from s and t (as shown in Example 2).
1. enumerate the different character in both s and t, then compute LCP on the left and right in $O(1)$ time. $O(n^2)$.
2. We call $s[i..i+\ell-1]$ and $t[j..j+\ell-1]$ a *maximally matched pair* iff $s[i..i+\ell-1] = t[j..j+\ell-1]$, $s[i-1] \neq t[j-1]$ and $s[i+\ell] \neq s[j+\ell]$. For a pair of substrings in s and t that differs by a single character, the different character must be the left (symmetrically, right) endpoint of a maximally matched pair (i.e. at index $i-1$). The right endpoint of the substring must lie in the maximally matched pair.

Let L be a parameter to be set later. We can find all maximally matched pairs with length $\geq L$ in $O(\frac{n^2}{L})$ time: for each fixed $\delta = i - j$, we repeatedly use LCP in $O(1)$ time to find the next maximally matched pair with length $\geq L$, and this requires $O(\frac{n}{L})$ steps for each δ. Then we can compute the number of solutions adjacent to at least one maximally matched pair with length $\geq L$.

The remaining case is solutions with the different character adjacent to two maximally matched pair with length $< L$, and this implies the total length of the matched substring is $O(L)$. Use hashing to count the number of solutions for each substring with length $< 2L$ (there are $O(nL)$ of them), enumerate the position of the different character in $O(L)$ time for each substring, and we can solve this case in $O(nL^2)$ time. (easy to prevent double counting in the first case.)

Set $L = O(n^{1/3})$, the total running time is $O(n^{5/3})$.