Leetcode problem no. 2289

Link- (https://leetcode.com/problems/steps-to-make-array-non-decreasing/description/)

Problem Description:

You are given two strings word1 and word2.

A string x is called **almost equal** to y if you can change **at most** one character in x to make it *identical* to y.

A sequence of indices seq is called **valid** if:

- The indices are sorted in **ascending** order.
- *Concatenating* the characters at these indices in word1 in **the same** order results in a string that is **almost equal** to word2.

Return an array of size word2.length representing the

valid sequence of indices. If no such sequence of indices exists, return an **empty** array.

Note that the answer must represent the *lexicographically smallest array*, **not** the corresponding string formed by those indices.

Example 1:

Input: word1 = "vbcca", word2 = "abc"

Output: [0,1,2]

Explanation:

The lexicographically smallest valid sequence of indices is [0, 1, 2]:

- Change word1[0] to 'a'.
- word1[1] is already 'b'.
- word1[2] is already 'c'.

Intuition:

The intuition is to check if word2 can be formed as a subsequence of word1 with at most one replacement: first, we precompute a suffix array (indices[]) that tells for every position in word1 how many characters of word2 can still be matched from there; then we greedily scan word1 from left to right, directly taking matches where possible, and if a mismatch occurs, we decide whether to use that position as the one allowed replacement (only if the rest of word2 can still be matched according to indices[]); finally, we continue matching the remaining characters—if all of word2 is matched, we return the indices, otherwise return empty.

Code:

```
class Solution {
    public int[] validSequence(String word1, String word2) {
        int len1 = word1.length();
        int len2 = word2.length();
        int[] indices = new int[len1];
        int j = len2-1;
        for(int i=len1-1;i>=0;i--){
```

```
    if(j>=0 && word1.charAt(i)==word2.charAt(j)){
        indices[i] = (i+1>=len1)?1:indices[i+1]+1;
        j--;
    }else{
        indices[i] = (i+1>=len1)?0:indices[i+1];
    }
}
List<Integer> ans = new ArrayList<>();
int fin = -1;
j = 0;
for(int i=0;i<len1;i++){
    if(word1.charAt(i)==word2.charAt(j)){
        ans.add(i);
        j++;
        if(j==len2) break;
    }else{
        if(len2-j-1<=((i+1<len1)?indices[i+1]:0)){
            ans.add(i);
            j++;
            fin = i+1;
            break;
```

```
            }
        }
    }
    if(ans.size()==len2) {
        return convtoarr(ans);
    }
    if(fin==-1){
        return new int[0];
    }
    for(int i=fin;i<len1;i++){
        if(word1.charAt(i)==word2.charAt(j)){
            ans.add(i);
            j++;
        }
        if(j==len2) break;
    }
    return ans.size()==len2 ? convtoarr(ans) : new int[0];
}
private int[] convtoarr(List<Integer> list) {
    return list.stream().mapToInt(Integer::intValue).toArray();
}
```

}

Time Complexity:

O(n)

Space Complexity:

O(n)

Best Time Complexity:

O(n)

Best space complexity:

O(n)