763. Partition Labels
Medium

You are given a string `s`. We want to partition the string into as many parts as possible so that each letter appears in at most one part.

Note that the partition is done so that after concatenating all the parts in order, the resultant string should be `s`.

Return *a list of integers representing the size of these parts*.

Example 1:

```
Input: s = "ababcbacadefegdehijhklij"
Output: [9,7,8]
Explanation:
The partition is "ababcbaca", "defegde", "hijhklij".
This is a partition so that each letter appears in at most one part.
A partition like "ababcbacadefegde", "hijhklij" is incorrect, because it
splits s into less parts.
```

Example 2:

```
Input: s = "eccbbbbdec"
Output: [10]
```

Constraints:

- `1 <= s.length <= 500`
- `s` consists of lowercase English letters.
- **Approach 1: Greedy**
- **Intuition**
- Let's try to repeatedly choose the smallest left-justified partition. Consider the first label, say it's `'a'`. The first partition must include it, and also the last occurrence of `'a'`. However, between those two occurrences of `'a'`, there could be other labels that make the minimum size of this partition bigger. For example, in `"abccaddbeffe"`, the minimum first partition is `"abccaddb"`. This gives us the idea for the algorithm: For each letter encountered, process the last occurrence of that letter, extending the current partition `[anchor, j]` appropriately.
- **Algorithm**
- We need an array `last[char] -> index of S where char occurs last`. Then, let `anchor` and `j` be the start and end of the current partition. If we are at a label that occurs last at some index after `j`, we'll extend the partition `j = last[c]`. If we are at the end of

the partition (`i == j`) then we'll append a partition size to our answer, and set the start of our new partition to `i+1`.

```
class Solution {

    public List<Integer> partitionLabels(String S) {

        int[] last = new int[26];

        for (int i = 0; i < S.length(); ++i)

            last[S.charAt(i) - 'a'] = i;



        int j = 0, anchor = 0;

        List<Integer> ans = new ArrayList();

        for (int i = 0; i < S.length(); ++i) {

            j = Math.max(j, last[S.charAt(i) - 'a']);

            if (i == j) {

                ans.add(i - anchor + 1);

                anchor = i + 1;

            }

        }

        return ans;

    }

}
```