

Partition Labels

A string `S` of lowercase letters is given. We want to partition this string into as many parts as possible so that each letter appears in at most one part, and 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.

Note:

1. `S` will have length in range `[1, 500]`.
2. `S` will consist of lowercase letters (`'a'` to `'z'`) only.

Solution 1

1. traverse the string record the last index of each char.
2. using pointer to record end of the current sub string.

```
public List<Integer> partitionLabels(String S) {  
    if(S == null || S.length() == 0){  
        return null;  
    }  
    List<Integer> list = new ArrayList<>();  
    int[] map = new int[26]; // record the last index of the each char  
  
    for(int i = 0; i < S.length(); i++){  
        map[S.charAt(i)-'a'] = i;  
    }  
    // record the end index of the current sub string  
    int last = 0;  
    int start = 0;  
    for(int i = 0; i < S.length(); i++){  
        last = Math.max(last, map[S.charAt(i)-'a']);  
        if(last == i){  
            list.add(last - start + 1);  
            start = last + 1;  
        }  
    }  
    return list;  
}
```

written by [Samuel.Y.T.Ji](#) original link [here](#)

Solution 2

```
def partitionLabels(self, S):
    sizes = []
    while S:
        i = 1
        while set(S[:i]) & set(S[i:]):
            i += 1
        sizes.append(i)
        S = S[i:]
    return sizes
```

written by [StefanPochmann](#) original link [here](#)

Solution 3

The idea is to use sliding window to add all the chars which are not exhausted yet use a hashset to know if current char is new or we have seen it before in the current window

if we find new char, increase counter

if we exhausted all char c (table[c-'a'] == 0) in the current window, then decrease counter

if counter becomes 0, it means that current window is a partition, add it to the result list and reset window

Note that counter is incremented when we see a new char in the current window which has not been seen before and it is decremented when count of current char in table becomes zero,

so at any point, the counter indicates the number of chars in the current window which has not been exhausted.

Hope this helps.

```
class Solution {
    public List<Integer> partitionLabels(String S) {
        List<Integer> res = new ArrayList<>();

        int[] table = new int[26];
        char[] stc = S.toCharArray();
        for(char c:stc)//count the occurrence of each char in string
            table[c-'a']++;

        int i = 0, j = 0, l = S.length(), counter = 0;
        HashSet<Character> hs = new HashSet<>();
        while(j < l){
            char c = stc[j];
            if(!hs.contains(c)){// found new char in current window, so increase counter
                hs.add(c);
                counter++;
            }
            table[c-'a']--;
            j++;
            if(table[c-'a'] == 0){ // decrease the counter as we have exhausted the
                counter--;
                hs.remove(c);
            }
            if(counter == 0){//if counter becomes 0, means current window is a partition
                res.add(j - i);//add length of current window
                i = j;// reset i for next window
            }
        }
        return res;
    }
}
```

written by [ashish53v](#) original link [here](#)

From [Leetcode](#).