Longest Substring with At Most Two Distinct Characters

Given a string, find the length of the longest substring T that contains at most 2 distinct characters.

For example, Given s = `“eceba”`,

T is "ece" which its length is 3.

Solution 1

This question belong to the same category as those such as "longest substring without repeating characters", "minimum window substring", and "substring with concatenation of all words". To solve this kind of question we can use two pointers and a hash table. When the key of the hash table is char, we can simply use an array as the hash table. The most important idea in solving this kind of questions is "how to update the "start" pointer" and the solution to these questions seem usually differ only in this respect.

```
int lengthOfLongestSubstringTwoDistinct(string s) {
    if(s.empty()) return 0;

    int dict[256];
    fill_n(dict,256,0);
    int start = 0, len = 1, count = 0;
    for(int i=0; i<s.length(); i++) {
        dict[s[i]]++;
        if(dict[s[i]] == 1) { // new char
            count++;
            while(count > 2) {
                dict[s[start]]--;
                if(dict[s[start]] == 0) count--;
                start++;
            }
        }
        if(i-start+1 > len) len = i-start+1;
    }
    return len;
}
```

written by morrischen2008 original link here

Solution 2

I submitted this solution two months ago and I can't even remember whether I wrote myself or copied from others. I don't understand it now but find it AC and much shorter than other posts here.

```
int lengthOfLongestSubstringTwoDistinct(string s) {
        int i = 0, j = -1;
        int maxLen = 0;
        for (int k = 1; k < s.size(); k++) {
            if (s[k] == s[k-1]) continue;
            if (j > -1 && s[k] != s[j]) {
                maxLen = max(maxLen, k - i);
                i = j + 1;
            }
            j = k - 1;
        }
        return maxLen > (s.size() - i) ? maxLen : s.size() - i;
    }
```

written by emersonxsu original link here

Solution 3

The main idea is to maintain a sliding window with 2 unique characters. The key is to store the last occurrence of each character as the value in the hashmap. This way, whenever the size of the hashmap exceeds 2, we can traverse through the map to find the character with the left most index, and remove 1 character from our map. Since the range of characters is constrained, we should be able to find the left most index in constant time.

```
public class Solution {
    public int lengthOfLongestSubstringTwoDistinct(String s) {
        if(s.length() < 1) return 0;
        HashMap<Character,Integer> index = new HashMap<Character,Integer>();
        int lo = 0;
        int hi = 0;
        int maxLength = 0;
        while(hi < s.length()) {
            if(index.size() <= 2) {
                char c = s.charAt(hi);
                index.put(c, hi);
                hi++;
            }
            if(index.size() > 2) {
                int leftMost = s.length();
                for(int i : index.values()) {
                    leftMost = Math.min(leftMost,i);
                }
                char c = s.charAt(leftMost);
                index.remove(c);
                lo = leftMost+1;
            }
            maxLength = Math.max(maxLength, hi-lo);
        }
        return maxLength;
    }
}
```

written by kevinhsu original link here