Longest Substring with At Least K Repeating Characters

Find the length of the longest substring ***T*** of a given string (consists of lowercase letters only) such that every character in ***T*** appears no less than *k* times.

Example 1:

```
Input:
s = "aaabb", k = 3

Output:
3

The longest substring is "aaa", as 'a' is repeated 3 times.
```

Example 2:

```
Input:
s = "ababbc", k = 2

Output:
5

The longest substring is "ababb", as 'a' is repeated 2 times and 'b' is repeated 3
times.
```

Solution 1

Update:

As pointed out by @hayleyhu, I can just take the first too rare character instead of a rarest. Submitted once, accepted in 48 ms.

```python
def longestSubstring(self, s, k):
    for c in set(s):
        if s.count(c) < k:
            return max(self.longestSubstring(t, k) for t in s.split(c))
    return len(s)
```

Original:

```python
def longestSubstring(self, s, k):
    if len(s) < k:
        return 0
    c = min(set(s), key=s.count)
    if s.count(c) >= k:
        return len(s)
    return max(self.longestSubstring(t, k) for t in s.split(c))
```

If every character appears at least k times, the whole string is ok. Otherwise split by a least frequent character (because it will always be too infrequent and thus can't be part of any ok substring) and make the most out of the splits.

As usual for Python here, the runtime varies a lot, this got accepted in times from 32 ms to 74 ms.

written by StefanPochmann original link here

Solution 2

```java
public int longestSubstring(String s, int k) {
    char[] str = s.toCharArray();
    return helper(str,0,s.length(),k);
}
private int helper(char[] str, int start, int end,  int k){
    if(end<start) return 0;
    if(end-start<k) return 0;//substring length shorter than k.
    int[] count = new int[26];
    for(int i = start;i<end;i++){
        int idx = str[i]-'a';
        count[idx]++;
    }
    for(int i = 0;i<26;i++){
        if(count[i]==0)continue;//i+'a' does not exist in the string, skip it.
        if(count[i]<k){
            for(int j = start;j<end;j++){
                if(str[j]==i+'a'){
                    int left = helper(str,start,j,k);
                    int right = helper(str,j+1,end,k);
                    return Math.max(left,right);
                }
            }
        }
    }
    return end-start;
}
```

written by Nakanu original link here

Solution 3

Sol1: a simple improvement on the naive quaratic solution. The idea is that if a locally longest substr is found, there's no need to check substrs overlapping it.
Sol1 can run O(n) times in some cases, but worst case is O(n2). Anyway the C++ run time is 3ms.

```
int longestSubstring(string s, int k) {
   int max_len = 0;
   for (int first = 0; first+k <= s.size();) {
       int count[26] = {0};
       int mask = 0;
       int max_last = first;
       for (int last = first; last < s.size(); ++last) {
           int i = s[last] - 'a';
           count[i]++;
           if (count[i]<k) mask |= (1 << i);
           else   mask &= (~(1 << i));

           if (mask == 0) {
               max_len = max(max_len, last-first+1);
               max_last = last;
           }
       }
       first = max_last + 1;
   }
   return max_len;
}
```

Sol2: recursive: split the string into substrs by characters of occurrence less than k. Then recursively apply the problem to each substr.
Worst case of Sol2 is O(n), because there are at most 26 levels of recursions. The C++ impl. runs 6ms. I suspect this is because the current test cases does not cover enough cases in favor of this solution in run time.

```
int longestSubstring(string s, int k) {
    return longestSubstring_recur(s, k, 0, s.size());
}

int longestSubstring_recur(const string& s, int k, int first, int last) {
    int count[26] = {0};
    for (int j = first; j < last; ++j) ++count[s[j] - 'a'];

    int max_len = 0;
    for (int j = first; j < last;) {
        while (j < last && count[s[j]-'a']<k) ++j;
        if (j == last) break;
        int l = j;
        while (l < last && count[s[l]-'a']>=k) ++l;
        //all chars appear more than k times
        if (j == first && l == last) return last-first;
        max_len = max(max_len, longestSubstring_recur(s, k, j, l));
        j = l;
    }
    return max_len;
}
```

written by hxtang original link here