Feb 28, Irvine, K-group algorithm study

## Longest Substring with At Most K Distinct Characters

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

**Example 1:**

```
Input: s = "eceba", k = 2
Output: 3
Explanation: T is "ece" which its length is 3.
```

**Example 2:**

```
Input: s = "aa", k = 1
Output: 2
Explanation: T is "aa" which its length is 2.
```

## Longest Substring Without Repeating Characters

Given a string, find the length of the **longest substring** without repeating characters.

**Example 1:**

```
Input: "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

**Example 2:**

```
Input: "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

**Example 3:**

```
Input: "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
             Note that the answer must be a substring, "pwke" is a subsequence and not
a substring.
```

## Minimum Window Substring

Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).

**Example:**

```
Input: S = "ADOBECODEBANC", T = "ABC"
Output: "BANC"
```

- If there is no such window in S that covers all characters in T, return the empty string `""`.
- If there is such window, you are guaranteed that there will always be only one unique minimum window in S.

## Fruit Into Baskets

In a row of trees, the `i`-th tree produces fruit with type `tree[i]`.

You **start at any tree of your choice**, then repeatedly perform the following steps:

1. Add one piece of fruit from this tree to your baskets. If you cannot, stop.
2. Move to the next tree to the right of the current tree. If there is no tree to the right, stop.

Note that you do not have any choice after the initial choice of starting tree: you must perform step 1, then step 2, then back to step 1, then step 2, and so on until you stop.

You have two baskets, and each basket can carry any quantity of fruit, but you want each basket to only carry one type of fruit each.

What is the total amount of fruit you can collect with this procedure?

**Example 1:**

```
Input: [1,2,1]
Output: 3
Explanation: We can collect [1,2,1].
```

**Example 2:**

```
Input: [0,1,2,2]
Output: 3
Explanation: We can collect [1,2,2].
If we started at the first tree, we would only collect [0, 1].
```

**Example 3:**

```
Input: [1,2,3,2,2]
Output: 4
Explanation: We can collect [2,3,2,2].
If we started at the first tree, we would only collect [1, 2].
```

**Example 4:**

```
Input: [3,3,3,1,2,1,1,2,3,3,4]
Output: 5
Explanation: We can collect [1,2,1,1,2].
If we started at the first tree or the eighth tree, we would only collect 4 fruits.
```

1. `1 <= tree.length <= 40000`
2. `0 <= tree[i] < tree.length`

```python
class Solution(object):
    def lengthOfLongestSubstringKDistinct(self, s, k):
        """
        :type s: str
        :type k: int
        :rtype: int
        """
        if s == "" or k == 0:
            return 0

        start = 0
        end = 0
        interval = 0
        counter = 0
        m = dict()
        for end in range(len(s)):
            c = s[end]
            if not m.has_key(c):
                counter += 1
                m[c] = 1
            else:
                m[c] += 1

            while counter > k:
                c2 = s[start]
                m[c2] -= 1
                if m[c2] == 0:
                    del m[c2]
                    counter -= 1
                start += 1
            interval = max(end-start, interval)

        return interval+1
```

```python
class Solution(object):
    def lengthOfLongestSubstring(self, s):
        """
        :type s: str
        :rtype: int
        """
        if s == "":
            return 0

        m = dict()
        start = 0
        end = 0
        interval = 0
        counter = 0

        for end in range(len(s)):
            c = s[end]
            if not m.has_key(c):
                m[c] = 1
            else:
                m[c] += 1
                counter += 1

            while counter > 0:
                c2 = s[start]
                m[c2]-=1
                if m[c2] == 1:
                    counter -= 1
                elif m[c2] == 0:
                    del m[c2]
                start+=1

            interval = max(end-start, interval)

        return interval + 1
```

```python
class Solution(object):
    def minWindow(self, s, t):
        tm = dict()

        for c in t:
            if not tm.has_key(c):
                tm[c] = 0
            tm[c] += 1

        counter = len(t)
        start = 0
        interval = len(s)
        result = ""
        for end in range(len(s)):
            c = s[end]

            if tm.has_key(c):
                if tm[c] > 0:
                    counter -= 1
                tm[c] -= 1

            while counter == 0:
                if end - start < interval:
                    result = s[start:end+1]
                    interval = end - start

                c2 = s[start]
                if tm.has_key(c2):
                    tm[c2]+=1
                    if tm[c2] > 0:
                        counter += 1
                start += 1
        return result
```

```python
class Solution(object):
    def totalFruit(self, tree):
        """
        :type tree: List[int]
        :rtype: int
        """
        start = 0
        end = 0
        counter = 0
        interval = 0
        m = dict()

        for end in range(len(tree)):
            f = tree[end]

            if not m.has_key(f):
                m[f] = 0
                counter+=1
            m[f] += 1

            while counter > 2:
                f2 = tree[start]

                m[f2] -= 1
                if m[f2] == 0:
                    counter -= 1
                    del m[f2]

                start+=1

            interval = max(end-start, interval)

        return interval + 1
```

https://leetcode.com/problems/minimum-window-substring/discuss/26808/Here-is-a-10-line-template-that-can-solve-most-'substring'-problems