Longest Substring Without Repeating Characters

Given a string, find the length of the longest substring without repeating characters. For example, the longest substring without repeating letters for "abcabcbb" is "abc", which the length is 3. For "bbbbb" the longest substring is "b", with the length of 1.

Solution 1

the basic idea is, keep a hashmap which stores the characters in string as keys and their positions as values, and keep two pointers which define the max substring. move the right pointer to scan through the string , and meanwhile update the hashmap. If the character is already in the hashmap, then move the left pointer to the right of the same character last found. Note that the two pointers can only move forward.

```
    public int lengthOfLongestSubstring(String s) {
         if (s.length()==0) return 0;
         HashMap<Character, Integer> map = new HashMap<Character, Integer>();
         int max=0;
         for (int i=0, j=0; i<s.length(); ++i){
             if (map.containsKey(s.charAt(i))){
                 j = Math.max(j,map.get(s.charAt(i))+1);
             }
             map.put(s.charAt(i),i);
             max = Math.max(max,i-j+1);
         }
         return max;
     }
```

written by cbmbbz original link here

Solution 2

```
/**
 * Solution (DP, O(n)):
 *
 * Assume L[i] = s[m...i], denotes the longest substring without repeating
 * characters that ends up at s[i], and we keep a hashmap for every
 * characters between m ... i, while storing <character, index> in the
 * hashmap.
 * We know that each character will appear only once.
 * Then to find s[i+1]:
 * 1) if s[i+1] does not appear in hashmap
 *    we can just add s[i+1] to hash map. and L[i+1] = s[m...i+1]
 * 2) if s[i+1] exists in hashmap, and the hashmap value (the index) is k
 *    let m = max(m, k), then L[i+1] = s[m...i+1], we also need to update
 *    entry in hashmap to mark the latest occurency of s[i+1].
 *
 * Since we scan the string for only once, and the 'm' will also move from
 * beginning to end for at most once. Overall complexity is O(n).
 *
 * If characters are all in ASCII, we could use array to mimic hashmap.
 */

int lengthOfLongestSubstring(string s) {
    // for ASCII char sequence, use this as a hashmap
    vector<int> charIndex(256, -1);
    int longest = 0, m = 0;

    for (int i = 0; i < s.length(); i++) {
        m = max(charIndex[s[i]] + 1, m);    // automatically takes care of -1 cas
e
        charIndex[s[i]] = i;
        longest = max(longest, i - m + 1);
    }

    return longest;
}
```

Hope you like it :)

written by dragonmigo original link here

Solution 3

```
int lengthOfLongestSubstring(string s) {
        vector<int> dict(256, -1);
        int maxLen = 0, start = -1;
        for (int i = 0; i != s.length(); i++) {
            if (dict[s[i]] > start)
                start = dict[s[i]];
            dict[s[i]] = i;
            maxLen = max(maxLen, i - start);
        }
        return maxLen;
    }
```

written by lightmark original link here